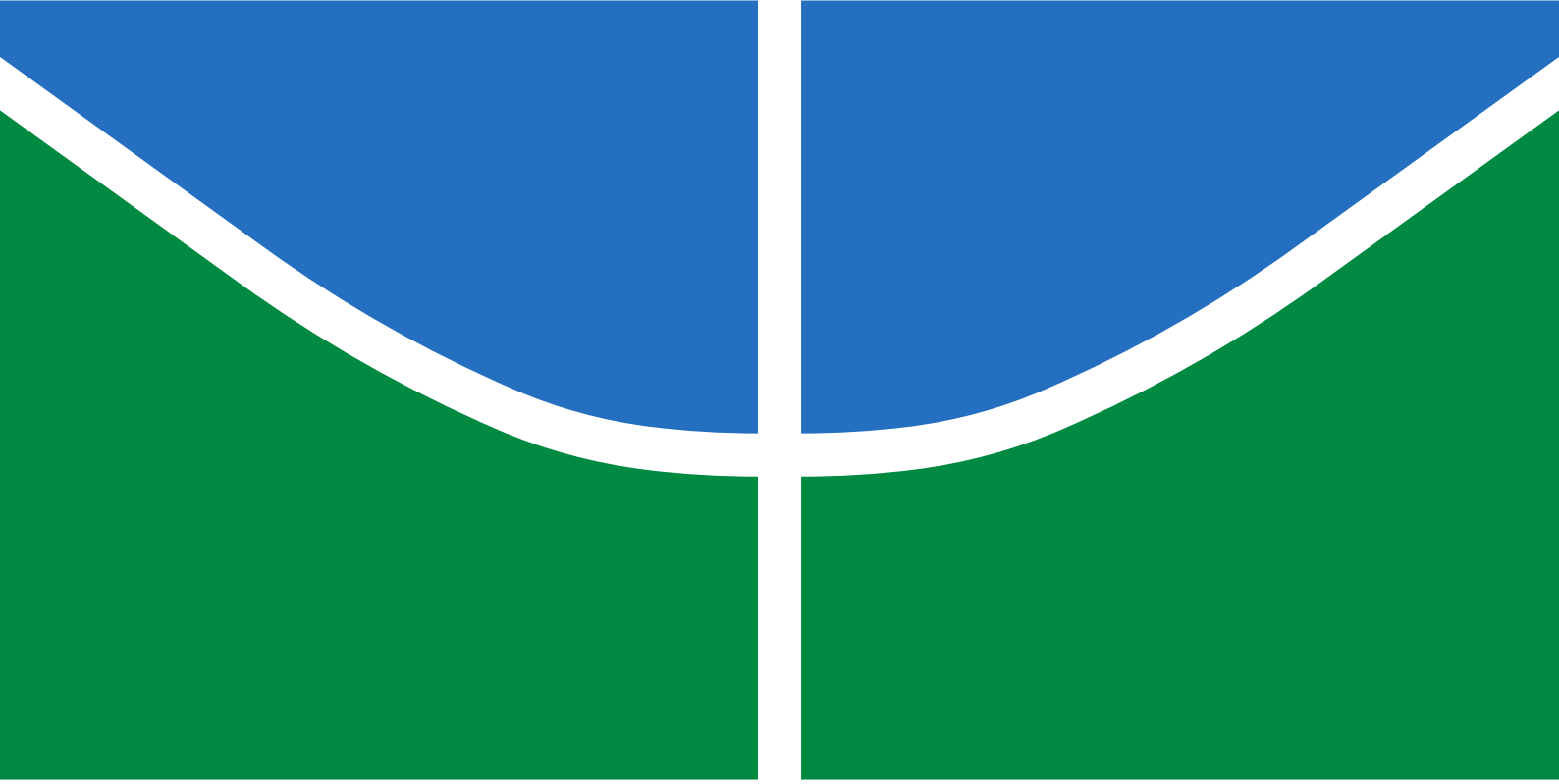


Universidade de Brasília - UnB

Faculdade de tecnologia - FT

Engenharia de Redes de Comunicação

DevSecOps e criação de pipeline para automação de descoberta de vulnerabilidades

Autores: André Cavalcanti Ribeiro; Victor Lyra Seidl

Orientador: Prof. Dr. Flávio Elias Gomes de Deus

Coorientador: Prof. Dr. Robson de Oliveira Albuquerque

Brasília, DF

2021

André Cavalcanti Ribeiro; Victor Lyra Seidl

DevSecOps e criação de pipeline para automação de descoberta de vulnerabilidades

Monografia submetida ao curso de graduação em Engenharia de Redes de Comunicaçãoda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Redes de Comunicação.

Universidade de Brasília - UnB

Faculdade de tecnologia - FT

Orientador: Prof. Dr. Flávio Elias Gomes de Deus

Coorientador: Prof. Dr. Robson de Oliveira Albuquerque

Brasília, DF

2021

André Cavalcanti Ribeiro; Victor Lyra Seidl

DevSecOps e criação de pipeline para automação de descoberta de vulnerabilidades/ André Cavalcanti Ribeiro; Victor Lyra Seidl. – Brasília, DF, 2021-

125 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Flávio Elias Gomes de Deus

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade de tecnologia - FT , 2021.

1. engenharia de redes de comunicação. 2. devsecops). 3. segurança. 4. integração contínua. I. Prof. Dr. Flávio Elias Gomes de Deus . II. Universidade de Brasília. III. Faculdade de Tecnologia - FT. IV. DevSecOps e criação de pipeline para automação de descoberta de vulnerabilidades

CDU 02:141:005.6

André Cavalcanti Ribeiro; Victor Lyra Seidl

DevSecOps e criação de pipeline para automação de descoberta de vulnerabilidades

Monografia submetida ao curso de graduação em Engenharia de Redes de Comunicaçãoda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Redes de Comunicação.

Brasília, DF, 19 de Maio de 2021:

Prof. Dr. Flávio Elias Gomes de Deus
Orientador

Flávio Henrique Rocha e Silva
Examinador Interno

Aurélio Ribeiro Costa
Examinador Externo

Brasília, DF
2021

Agradecimentos

Agradeço à minha família pelo incondicional apoio ao longo de toda minha vida, aos meus amigos por todos os momentos amistosos, a minha namorada por toda o companheirismo, à minha cachorra pelos passeios diários, ao meu amigo e dupla de PFG, Victor Lyra, por todo o trabalho duro e dedicação em conjunto neste trabalho, ao Flávio Henrique pelas experiências trocadas e apoio ao longo de todo o projeto e, por fim, ao professor Flávio Elias por todas as oportunidades e confiança ao longo dos meus últimos três anos de graduação.

André Cavalcanti Ribeiro

Agradeço aos meus pais pelo suporte ao longo de todo o curso e de toda a vida e por proporcionar as melhores condições para que eu chegasse até aqui, aos irmãos e amigos por todo o carinho e momentos incríveis juntos. Agradeço também ao Flávio Henrique e ao Igor por todo o conhecimento transmitido e pelo companheirismo durante todo o projeto, ao André, pelas incontáveis horas que passamos juntos na realização deste trabalho e durante o curso. Por fim, agradeço aos nossos orientadores por todos os conselhos e pela confiança no trabalho.

Victor Lyra Seidl

Agradecemos às instituições envolvidas no suporte dessa pesquisa através do projeto 01/2019 do Ministério da Cidadania por meio da Secretaria Nacional de Assistência Social - SNAS/MC e do projeto com a Polícia Federal, por intermédio da Diretoria Técnico-Científica - DITEC/PF, processo: 08059.000888/2019-35.

Agradecemos também ao Laboratório de Tecnologias da Tomada de Decisão (LATITUDE) - UnB por todo o suporte ao projeto e pela disponibilização da infraestrutura necessária para sua realização.

Resumo

Em decorrência do rápido e frequente avanço da tecnologia na área de hardware computacionais, tornou-se possível a aquisição de equipamentos capazes de rodar - em notebooks, tablets ou em simples aparelhos celulares - aplicações de grandes complexidades. Paralelamente, o estudo de linguagens de programação e a exploração de *framework* vem se tornando cada vez mais acessível. Tais fatos fragilizam o processo de criação e desenvolvimento de várias aplicações, uma vez que puderam passar a ser feitas por qualquer pessoa estudiosa e/ou curiosa, dispensando a necessidade de um amplo estudo prévio por parte de profissionais devidamente qualificados para tal.

Por um lado, embora as facilidades geradas pelo cenário acima possam de fato parecer promissoras, elas também trazem uma maior preocupação no que tange aos quesitos de segurança e confiabilidade. Desenvolvedores inexperientes podem cometer alguns deslizes em seus códigos, deixando-os vulneráveis a ataques conhecidos ou, ainda, na intenção de alcançar maior praticidade, acabarem utilizando pacotes em suas aplicações que poderão vir a comprometer toda a segurança do código.

Para amenizar tais problemas, a implementação da metodologia DevSecOps nas empresas vem adquirindo cada vez mais espaço e relevância, uma vez que auxilia a equipe de desenvolvedores a criar novas ferramentas e aplicações pensando prévia e prioritariamente nos quesitos de segurança. Desde o começo do desenvolvimento, o código já é testado e inspecionado, o que diminui as chances de uma possível refatoração do código e também garante que os passos futuros terão um respaldo maior em segurança.

Este trabalho aplica os principais pilares da metodologia DevSecOps na criação de uma aplicação desenvolvida utilizando os *frameworks* React e NodeJS, ambas utilizando a linguagem *JavaScript*. Para isso, utilizou-se o gerenciador de códigos *Gitlab* e seu ambientes de *Continuous Integration/Continuous Delivery* - CI/CD, responsável pela execução dos testes de segurança definidos na *pipeline* e também pela disponibilização da aplicação para utilização dos alunos da Universidade de Brasília - (UnB).

Palavras-chaves: DevSecOps, Segurança, Integração contínua, Entrega contínua, Análise de vulnerabilidades

Abstract

Due to the rapid and frequent advancement of technology in the area of computational hardware, it became possible to acquire equipment capable of running - in notebooks, tablets or in simple cellular devices - highly complex applications. At the same time, the study of programming languages and the exploitation of framework has become increasingly accessible. Such facts weaken the process of creation and development of various applications, since they could be made by any person who is studious and / or curious, dispensing with the need for a broad prior study by professionals duly qualified to do so.

On the one hand, although the facilities generated by the scenario above may indeed look promising, they also bring greater concern with regard to safety and reliability. Inexperienced developers may make some mistakes in their code, leaving them vulnerable to known attacks or, in the intention of achieving greater practicality, end up using packages in their applications that may compromise the security of the code.

To alleviate such problems, the implementation of the DevSecOps methodology in companies has been acquiring more and more space and relevance, since it helps the team of developers to create new tools and applications with prior and priority thinking about security issues. Since the beginning of the development, the code has been tested and inspected, which reduces the chances of a possible refactoring of the code and also ensures that future steps will have a greater support in security.

This work applies the main pillars of the DevSecOps methodology in the creation of an application developed using the frameworks React and NodeJS, both using the JavaScript language. For this, we used the code manager Gitlab and its Continuous Integration / Continuous Delivery - CI/CD environments, responsible for executing the security tests defined in the pipeline and also for the availability of the application for use by students at the University of Brasília - (UnB).

Keywords: DevSecOps, Security, Continuous Integration, Continuous Delivery, Vulnerability analysis

Sumário

Lista de ilustrações

Lista de tabelas

Lista de códigos

Lista de abreviaturas e siglas

API	- *Application Programming Interface*: Interface de programação de aplicação
APK	- *Android Package*: Pacote Android
CD	- *Continuous Deployment*: Entrega Contínua
CI	- *Continuous Integration*: Integração Contínua
CSS	- *Cascading Style Sheets*: Folhas de Estilo em Cascata
CVE	- *Common Vulnerabilities and Exposures*: Vulnerabilidades e Exposições Comuns
CWE	- *Common Weakness Enumeration*: Enumeração de vulnerabilidades comuns
DAST	- *Dynamic application security testing*: Teste dinâmico de segurança da aplicação
DOM	- *Document Object Model*: Modelo de Objeto de Documento
DoS	- *Denial of Service*: Negação de serviço
HSTS	- *HTTP Strict Transport Security*: Segurança Estrita de Transporte HTTP
I/O	- *Input/Output*: Entrada/Saída
JSON	- *JavaScript Object Notation*: Notação de Objetos JavaScript
JWT	- *JSON Web Token*: Token Web JSON
NVD	- *National Vulnerability Database*: Banco de dados Nacional de Vulnerabilidades
ORM	- *Object-Relational Mapping*: Mapeamento Objeto-Relacional
RCE	- *Remote Code Execution*: Execução Remota de Código
REST	- *Representational State Transfer*: Transferência Representacional de Estado

SAST - *Static application security testing*: Teste estático de segurança da aplicação

SCM - *Source Code Management*: Gerenciador de código fonte

SDLC - *Software Development Life Cycle*: Ciclo de vida de desenvolvimento do software

SGBD - *Sistema de Gerenciamento de Bancos de Dados*

TLS - *Transport Layer Security*: Segurança da Camada de Transporte

XML - *Extensible Markup Language*: Linguagem de Marcação Extensível

YAML - *YAML Ain't Markup Language*: YAML não é Linguagem de Marcação

1 Introdução

A partir do ano de 2010, as empresas passaram a implementar amplamente a cultura de DevOps, o que se mostrou benéfico para o desenvolvimento e implementação de novos recursos aos projetos, uma vez que proporcionava uma maior aproximação entre os times de desenvolvimento e operações, permitindo, dessa forma, que todo o time envolvido no projeto tivesse a oportunidade de trabalhar simultaneamente e realizassem suas entregas de forma mais integrada (KIM et al., 2016). Essa nova abordagem no desenvolvimento de aplicações se deu como uma alternativa relativa às metodologias ágeis - difundidas em meados dos anos 2000 (KIM et al., 2016), que focavam a publicação dos novos recursos o mais rápido possível sem, entretanto, realizar um trabalho que garantisse maior proximidade com a equipe responsável pela operacionalização de todo projeto. Em decorrência disso, inúmeros retrabalhos se faziam necessários, a fim de que o projeto chegasse a uma correta execução (HEMON et al., 2020).

A implantação da cultura de DevOps, entretanto, não absolve as empresas dos cuidados constantes e necessários relativos à segurança das aplicações lançadas. Recentes estudos mostram que a taxa de crimes cibernéticos tem aumentado substancialmente ao longo dos anos. Na primeira metade de 2018, por exemplo, vazaram 4.5 bilhões de dados armazenados, número esse que corresponde a um aumento na ordem de 133% em relação ao registrado em 2017 (TOMAS; LI; HUANG, 2019). Considerando os dados até aqui apresentados fica evidente que a ênfase a ser dada na segurança das informações implementadas no projeto devem ser repensadas e avaliadas com a máxima cautela, na tentativa de evitar o vazamento de dados sigilosos dos usuários da aplicação.

A cultura do DevSecOps foi pensada e desenvolvida essencialmente para minimizar os riscos de segurança enfrentados pelas empresas. Com o intuito de integrar as questões relativas à segurança ao longo de todo o desenvolvimento da aplicação, elas não mais se encontram à direita do ciclo de vida do desenvolvimento do projeto e sim à esquerda (HSU, 2018). Ou seja, tais atividades estão presentes do início da concepção da aplicação até a sua conclusão, deixando de ser realizadas tão somente após o projeto ter sido concluído ou até mesmo implementado.

A Figura 1.1 apresenta uma representação dos estágios de trabalho tomados ao se utilizar da cultura DevSecOps de uma maneira geral.

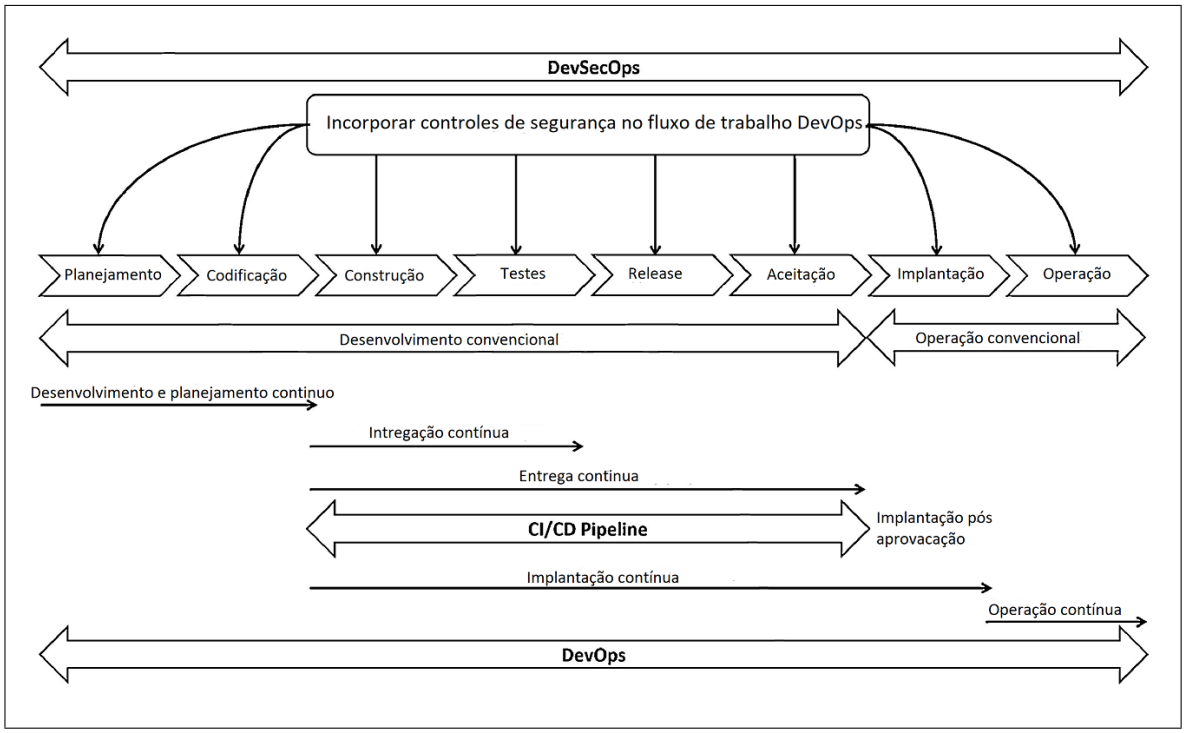


Figura 1.1 – Fluxo de etapas DevSecOps. Modificada. Fonte: (KUMAR; GOYAL, 2020).

1.1 Definição do problema

Muitas vezes, ao se desenvolver uma nova aplicação, os quesitos relacionados à segurança podem acabar sendo negligenciados, sobretudo por requererem maior tempo e esforço para produção de um código seguro o que, consequentemente, atrasaria sua disponibilização imediata. Com isso, o produto final pode ter a utilização de pacotes e bibliotecas de terceiros acessados na internet, sem verificação de sua credibilidade ou, até mesmo, permitir injeção de SQL no banco de dados. As falhas previamente listadas são identificadas e ranqueadas como falhas gravíssimas, de acordo com a *Open Web Application Security Project* - OWASP (OWASP, 2017l).

Entretanto, a despeito de que se tomem todos os cuidados básicos relacionados às falhas gravíssimas, em diversas aplicações, o projeto desenvolvido ainda assim poderá estar vulnerável a falhas menos conhecidas e mais específicas da, ou até mesmo das linguagens utilizadas. Podemos atribuir a falta de verificação como sendo um dos fatores que corroboram para que ocorra falha de segurança, podendo ocasionar desde um ataque simples - na qual a aplicação ficará fora do ar - até uma possível quebra de sigilo dos dados dos usuários que utilizam da aplicação, como ocorrido no caso de ataque a base de dados do eBay, em 2014 (SIDHU; SAKHUJA; ZHOU, 2016).

1.2 Objetivos

Este projeto busca viabilizar e implementar os conceitos da prática de DevSecOps às fases de desenvolvimento de uma aplicação base, utilizando ferramentas *open source*, projetando e fornecendo assim não somente um ambiente capaz de realizar a integração e entrega contínua do *software* mas, sobretudo, que elas ocorram em ambiente capaz de analisar e verificar se o código final atende aos requisitos estabelecidos.

Objetivos específicos

Com a implementação de tais conceitos ao desenvolvimento do código, a aplicação resultante certamente será mais segura contra ataques conhecidos. Ademais, o ambiente responsável pela execução dos testes previamente definidos terá a capacidade de inibir que uma nova versão da aplicação fique disponível, caso os requisitos de segurança não tenham sido alcançados. Alertando, dessa forma, os desenvolvedores que, por sua vez, terão o dever de ajustar o produto.

Portanto, visando o alcance dos resultados previamente definidos, os seguintes passos devem ser seguidos:

- o desenvolvimento do código seguindo as diretrizes de segurança definidas ao longo de todo o ciclo de vida do projeto;
- testagem estática de segurança do código;
- testagem dinâmica de segurança do código;
- disponibilização da aplicação segura para acessos.

1.3 Organização do trabalho

Os capítulos a seguir são organizados da seguinte forma:

- capítulo 2 apresenta os conceitos base que foram utilizados para o desenvolvimento deste trabalho;
- capítulo 3 discorre sobre a implementação e ferramentas utilizadas no desenvolvimento do projeto;
- capítulo 4 apresenta os resultados e discussões relacionadas;
- por fim, o capítulo 5 faz uma discussão final da solução e propõe pontos a serem explorados em trabalhos futuros sobre o tema.

2 Fundamentação teórica

Para a compreensão do problema abordado e dos conceitos utilizados na proposta de solução, é necessário citar alguns pontos e ferramentas que foram utilizados para atingir o objetivo proposto. Esse capítulo traz uma breve explicação desses conceitos fundamentais.

2.1 Segurança de aplicações Web

Segundo Michael E. Whitman, segurança pode ser definida como "proteçao contra adversários – aqueles que fariam mal, intencionalmente ou não" (WHITMAN; MATTORD, 2012). O CNSS (Comittee on National Security Systems) define segurança da informação como sendo a proteção da informação e seus elementos críticos, incluindo sistemas e hardware que utilizam, armazenam e transmitem informação (CNSS, 2016).

O CNSS desenvolveu um conceito que vem sendo adotado como padrão dentro da indústria de segurança da informação desde o desenvolvimento do *mainframe*, denominado *CIA triad*. Esse modelo baseia-se em três conceitos fundamentais, nos quais se baseia a segurança cibernética das organizações. São eles:

- **Confidencialidade**: caracteriza-se pela ocultação ou não divulgação de informações ou recursos. Uma informação é confidencial quando está protegida de divulgação ou exposição a indivíduos ou sistemas não autorizados. Quando sistemas ou indivíduos não autorizados visualizam determinada informação, sua confidencialidade é comprometida (WHITMAN; MATTORD, 2012);

- **Integridade**: caracteriza-se por dois conceitos interligados: integridade de dados e integridade de sistemas. O primeiro garante que a informação - tanto armazenada quanto em transmissão – bem como programas, sejam modificados apenas de uma maneira específica e desde que previamente autorizados. Já a integridade de sistemas, garante o funcionamento do sistema da forma como foi designado, livre de qualquer manipulação não autorizada do sistema. Portanto, a perda de integridade de dados ou sistemas, é a modificação ou manipulação não autorizada de informação (STALLINGS, 2017);

- **Disponibilidade**: a disponibilidade de dados e sistemas garante que a informação está acessível a todos os usuários que necessitam utilizá-la. A perda de disponibilidade é caracterizada por qualquer perturbação que houver no acesso e uso da

informação provida pelo sistema. Quanto mais crítico um serviço ou componente, maior será o nível de disponibilidade por ele requerida (STALLINGS, 2017).

Whitman define ainda 4 outras características críticas quando se trata de transmissão de informação e do uso dela. São elas (WHITMAN; MATTORD, 2012):

- **Acurácia**: uma informação é acurada quando está livre de erros e quando possui o valor que o usuário final espera. Se houver qualquer modificação, intencional ou não, da informação, ela não é mais acurada;
- **Autenticidade**: refere-se ao fato de a informação ser genuína e original e não uma reprodução. Uma informação é autêntica quando está no mesmo estado na qual foi criada, armazenada ou transferida. Este conceito é frequentemente utilizado para referir-se a certeza com relação à origem do objeto, dados ou informação em questão;
- **Utilidade**: uma informação é útil, ou seja, tem algum valor, quando pode servir determinado propósito. Um mesmo dado ou informação pode ter diferentes utilidades quando interpretado por diferentes públicos;
- **Posse**: A posse de uma informação refere-se ao conceito de propriedade e controle do conjunto de dados e informações.

A segurança de computadores e sistemas baseia-se nos conceitos de Confidencialidade, Disponibilidade e Integridade para definir políticas de segurança, estabelecer diferentes graus de possíveis ameaças e estabelecer meios utilizados para delas se proteger e garantir que a informação em questão não viole esses três conceitos básicos. Os conceitos definidos por Whitman como tríade CIA estendida trata de alguns outros meios e conceitos utilizados para definir características de uma determinada informação e, baseado nelas, atribuir valor aos dados em questão. Ao ter qualquer uma dessas características violadas ou destruídas, o valor atribuído àquela informação é alterado.

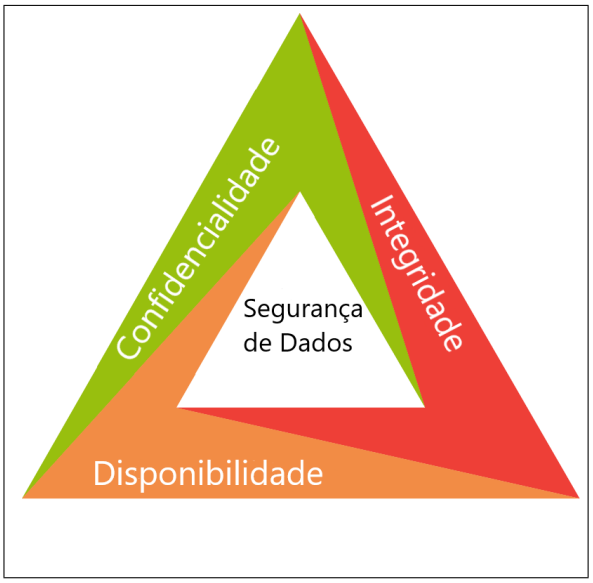


Figura 2.1 – Modelo adotado como padrão no ramo de segurança da informação, denominado CIA *triad* - Confidencialidade, Integridade, Disponibilidade. Modificada. Fonte: (DEVOPEDIA, 2016)

2.1.1 Vulnerabilidades de aplicações

A popularização da internet e da *World Wide Web* nos anos 2000, mudou radicalmente a forma como enxergamos a internet. Ela deixou de ser meramente um meio de transmissão de documentos, passando a ser um meio de transmissão de aplicações. Como resultado, surgiram novos tipos de *exploits*, ou seja, novos tipos de exploração de vulnerabilidades, que tomam vantagem do usuário e não da rede ou do servidor. Essa é uma mudança fundamental que ainda é verdadeira nos dias atuais, uma vez que a maioria dos *hackers* atuais atacam aplicações *web* executadas via *browser*, em detrimento de sistemas operacionais e outros softwares (HOFFMAN, 2020).

Objetivando o enfrentamento ao crescente número de ataques às aplicações *web*, diversas organizações surgiram com o intuito de pesquisar, classificar e disseminar informações relacionadas a essa área de interesses, dentre elas, a *OWASP - Open Web Application Security Project* - OWASP é uma organização sem fins lucrativos, dedicada a aprimorar a segurança de software, possibilitando que as organizações concebam, desenvolvam, adquiram, operem e mantenham aplicações confiáveis. Todos os projetos, ferramentas, documentos, fóruns e capítulos são gratuitos e disponíveis a quem se interessar em aperfeiçoar a segurança de software (OWASP, 2021a).

A OWASP descreve os 10 principais riscos de segurança para aplicações *web* no documento *OWASP Top Ten*. Trata-se de um documento de conscientização - voltado para desenvolvedores e profissionais de segurança de aplicações web - e goza de amplo consenso entre os profissionais, no que tange aos riscos mais críticos inerentes à segurança.

De acordo com a associação, empresas devem adotar esse documento para iniciar o processo e possam garantir que suas aplicações minimizem esses riscos (OWASP, 2017l).

Os 10 maiores riscos descritos são:

2.1.1.1 Injeção de código

Injeção de código é um método no qual um atacante insere código malicioso em um processo em execução e transfere a execução para o código malicioso. Dessa forma, o atacante pode obter controle do processo em execução, fazendo com que outros processos sejam executados, arquivos do sistema sejam modificados, entre outros. Se o programa é executado em um nível de privilégio superior do que o do atacante, é possível haver um escalonamento de privilégios (RILEY; JIANG; XU, 2010).

Alguns dos tipos mais comuns de injeção de código são:

- **SQL**: caracteriza-se pela injeção de código realizada em bancos de dados relacionais que utilizam SQL *(Standard Query Language)*. Ocorre quando a entrada de dados realizada pelo usuário não é filtrada, ou seja, não estabelece um caracter de escape, e essa entrada é passada ao servidor por meio de uma *query* SQL. (THIYAB et al., 2017) Ou seja, o atacante tem a capacidade de manipular consultas ao banco de dados, o que o torna capaz de ter acesso a informações privilegiadas;

- **NoSQL**: caracteriza-se pela injeção de código em bancos de dados não relacionais. Ocorre de forma similar à injeção de código em bancos de dados relacionais, exceto pelo fato de utilizar uma linguagem procedural - como por exemplo, JavaScript ou alguma linguagem disponibilizada pelo próprio SGBD - para realizar a busca no banco de dados, em vez de uma linguagem declarativa, como SQL (HOU et al., 2016);

- **Injeção de comandos do sistema operacional**: injeções de comando podem ocorrer em aplicações que utilizam uma entrada do usuário como parâmetro para executar comandos do sistema operacional. Comparado a outros tipos de injeção de código, como injeção SQL, injeções de comando não são muito predominantes. No entanto, pode trazer consequências significantes e custosas. O impacto desse tipo de ataque pode ir desde perda de confidencialidade de dados e integridade até acesso remoto não autorizado ao sistema que hospeda a aplicação vulnerável (STASINOPOULOS; NTANTOGIAN; XENAKIS, 2019).

Portanto, a injeção de código pode resultar em danos severos, como perda ou corrupção de dados. A OWASP define uma aplicação vulnerável a ataques de injeção de código quando satisfaz alguns desses pontos (OWASP, 2017b):

- dados fornecidos pelo usuário não é validado, filtrado e sanitizado pela aplicação;
- consultas dinâmicas ou chamadas não parametrizadas utilizadas diretamente no interpretador;
- dados hostis são utilizados como parâmetros para buscas em ORM(Object Relational Mapping) para extrair registros sensíveis;
- dados hostis são concatenados ou utilizados diretamente, de forma que o comando contenha a estrutura e os dados hostis em suas consultas dinâmicas ou comandos.

Para detectar esse tipo de falha, é possível utilizar ferramentas de revisão de código, utilizar testes automatizados dos parâmetros de entrada do usuário. Além disso, também é possível detectar tais vulnerabilidades utilizando ferramentas SAST e DAST na *pipeline* de CI/CD (OWASP, 2017b).

Para prevenção desse tipo de falha, é necessário manter os dados separados dos comandos e consultas realizados pelos usuário.

A melhor opção para prevenção desse tipo de ataque é a utilização de uma API segura, que evita o uso do interpretador completamente e fornece uma interface parametrizada, ou utiliza ferramentas ORM. Também é possível utilizar uma lista branca, contendo usuários ou servidores considerados confiáveis, validando a entrada do lado do servidor (OWASP, 2017b).

2.1.1.2 Falha de autenticação

Em uma aplicação web com gerenciamento de sessão, uma sessão é criada quando o usuário envia suas credenciais para o servidor e essas credenciais são verificadas no banco de dados, autenticando o usuário e fornecendo as devidas permissões à aplicação. Falha de autenticação é um tipo de vulnerabilidade de aplicações web que ocorre devido a configuração indevida do gerenciamento de sessão da aplicação. Depois que a autenticação é realizada, uma sessão será criada para ativar a comunicação de dados entre o usuário especificado e o servidor. Se um atacante conseguir acesso a essa sessão de algum usuário, contornando o processo de autenticação realizado pela aplicação, esse cenário é tratado como uma falha de autenticação (HASSAN et al., 2018).

De acordo com o OWASP, a aplicação tem falhas na autenticação se (OWASP, 2017c):

- permite ataques automatizados como *credential stuffing*, em que o atacante consegue utilizar uma lista de usuários e senhas para autenticar na aplicação;
- permite a utilização de senhas padrão e/ou fracas;

- utiliza senhas em texto claro, criptografadas ou utiliza algoritmos de *hash* fracos;
- expõe IDs de sessão na URL;
- não invalida a sessão após um certo período de tempo.

Para prevenção desse tipo de ataque, é possível (OWASP, 2017c):

- não realizar o *deploy* de aplicações com credenciais padrão, especialmente para usuários administradores;
- implementar verificação de força de senha;
- limitar ou aumentar o atraso de tentativas de login mal sucedidas. Registrar todas as falhas e alertar os administradores assim que ataques de força bruta forem detectados;
- utilizar um gerenciador de sessão do lado do servidor seguro, que gera identificadores de sessão aleatoriamente assim que uma nova sessão é registrada.

O impacto desse tipo de ataque varia de acordo com o privilégio que foi obtido ao autenticar-se. Caso o atacante tenha obtido privilégio de administrador da aplicação ou usuário com altas permissões, é possível causar danos sérios à vítima, desde danos à reputação até financeiros.

2.1.1.3 Exposição de dados sensíveis

Exposição de dados sensíveis ocorre quando a aplicação ou os serviços associados a ela (banco de dados, por exemplo), expõe dados pessoais de forma inadequada, permitindo que esses dados sejam de conhecimento de outros.

Esse tipo de falha ocorre por fatores como: armazenamento de informação sensível em texto claro, transmissão de informação sensível em texto claro, criptografia fraca ou inexistente, exposição de informação sensível por meio de consultas de dados, entre outros (CWE, 2017). Para detecção de uma aplicação vulnerável a esse tipo de falha, é necessário determinar a proteção dos dados, tanto em trânsito quanto armazenados.

A prevenção desse tipo de falha passa pelos seguintes pontos (OWASP, 2017d):

- classificação dos dados processados, armazenados e transmitidos pela aplicação e identificação dos dados sensíveis;
- criptografar todos os dados que não estão em trânsito;
- não armazenar dados sensíveis desnecessariamente;

- criptografar todos os dados em trânsito utilizando protocolos seguros como TLS e diretivas como HSTS.

2.1.1.4 XML external entities (XXE)

XML external entities permitem a inclusão de dados dinamicamente de um determinado recurso no momento de análise do arquivo. Esse recurso pode ser explorado por atacantes para incluir dados maliciosos de URIs externas ou dados confidenciais hospedados no sistema local. Se os softwares que analisam XML não estiverem configurados corretamente para lidar com entidades externas, são forçados a acessarem a URI especificada (JAN; NGUYEN; BRIAND, 2015).

A aplicação pode ser vulnerável a esse tipo de ataque se: utiliza arquivos XML diretamente ou aceita *uploads* de arquivos XML, especialmente de fontes não confiáveis, ou insere dados não confiáveis em documentos XML, que posteriormente é analisada por um software que processará esse arquivo.

Para prevenir esse tipo de ataque, é recomendável utilizar formatos de dados menos complexos, como JSON, por exemplo, ou apenas desabilitar o uso de entidade externa no analisador XML. Além disso, ferramentas de análise estática de código podem ajudar a detectar possíveis falhas de XXE no código fonte da aplicação (OWASP, 2017e).

2.1.1.5 Falha de controle de acesso

O controle de acesso de uma aplicação cria uma política de controle de privilégios de usuários, garantindo que usuários tenham acesso somente àquilo que foi pretendido. Uma falha no controle de acesso de uma aplicação web pode ter impacto significativo ao fazer com que usuários comuns se passem por administradores, por exemplo, tendo o poder de criar, acessar, remover e atualizar dados.

Uma aplicação pode estar vulnerável a falhas de controle de acesso se (OWASP, 2017f):

- é possível ignorar checagens de controle de acesso modificando a URL, estado interno da aplicação, página HTML ou utilizando uma ferramenta de ataque à API;
- é possível escalar privilégios, ou seja, agir como usuário sem estar autenticado ou agir como administrador autenticando-se como usuário comum;
- manipulação de metadados, como adulteração de *tokens JWT*;
- forçar a navegação à páginas que necessitam de autenticação como um usuário não autorizado ou a páginas de administrador como usuários comuns;

- acessar a API sem o controle de acesso para os métodos HTTP POST, PUT e DELETE.

Para detectar esse tipo de falha, é possível utilizar ferramentas de SAST e DAST para detectar a ausência de controle de acesso. No entanto, é necessário a testagem manual para garantir os privilégios corretos para cada tipo de usuário.

É possível utilizar algumas técnicas de prevenção a esse tipo de falha, como: registrar falhas de controle de acesso e emitir alertas, invalidar *tokens* de autenticação assim que o usuário realizar o *logout* da aplicação. Além disso, é importante que as funcionalidades de controle de acesso sejam testadas nas etapas de testes funcionais e unitários da aplicação (OWASP, 2017f).

2.1.1.6 Configurações incorretas de segurança

Configurações incorretas de segurança, que englobam páginas e contas padrão, páginas não utilizadas, arquivos e diretórios não protegidos, sistemas não atualizados, podem fazer com que atacantes adquiram conhecimento indevido do sistema, da aplicação, ou até mesmo acesso não autorizado (OWASP, 2017g).

Esse tipo de falha pode acontecer em qualquer nível da aplicação, desde os serviços de rede e servidores que servem como base para o funcionamento da aplicação até bancos de dados, *frameworks*, máquinas virtuais e contêineres. Para detectar esse tipo de falha, é importante que sejam executadas varreduras automatizadas para alerta e possível solução do problema (OWASP, 2017g).

2.1.1.7 *Cross-site scripting* (XSS)

Cross site scripting é uma ameaça que permite que o atacante execute código no *browser* da vítima, com a finalidade de ganhar acesso a informações como *cookies*, senhas, número de cartões de crédito, entre outros (GUPTA; GUPTA, 2017). Para explorar as vulnerabilidades XSS, o atacante cria e injeta um código malicioso na aplicação, de modo que, pareça ser um componente não malicioso. Esse código é executado no domínio da própria aplicação. Ataques XSS podem ter grande impacto para o usuário, resultando em roubo de contas, senhas, download de softwares maliciosos, entre outros.

Existem três tipos principais de ataques XSS, atacando o browser do cliente (OWASP, 2017h), são elas:

- *Reflected XSS*: a aplicação ou a API contém entrada do usuário não validada como parte da saída do HTML. Um atacante poderia explorar essa falha executando arbitrariamente códigos HTML e JavaScript no *browser* da vítima;

- *Stored XSS*: a aplicação ou a API armazena conteúdo de entrada do usuário não sanitizado e é visto posteriormente por outro usuário ou administrador;
- *DOM XSS*: *frameworks*, páginas e APIs que incluem dinamicamente dados controlados por um possível atacante são vulneráveis a *DOM XSS*.

Para prevenção desse tipo de ataque, é possível: evitar utilizar dados de requisições HTTP não confiáveis na saída do HTML, em elementos como *body*, atributos, JavaScript, CSS ou URL, aplicar codificação sensível ao contexto ao modificar o documento no *browser* no lado do cliente. Além disso, ferramentas automatizadas de verificação de código podem auxiliar na descoberta precoce desse tipo de vulnerabilidade (OWASP, 2017h).

2.1.1.8 Desserialização insegura

Serialização é o processo de transformar um objeto em um formato de dados que pode ser restaurado depois, normalmente utilizado para salvar dados ou enviar dados. Já o processo de desserialização é o inverso, transformar dados estruturados em um certo formato e construí-lo em um objeto (OWASP, 2017k).

Algumas linguagens oferecem uma capacidade nativa de desserialização de objetos. No entanto, essas capacidades podem ser adaptadas pelo atacante para operar diferentemente quando tratando dados maliciosos. Esse tipo de ataque pode ser utilizado em ataques DoS, controle de acesso e RCE.

Uma aplicação é considerada vulnerável a esse tipo de ataque se desserializa dados fornecidos por um atacante, podendo resultar em: ataques relacionados a estruturas de dados e objetos, em que o atacante modifica a lógica da aplicação ou consegue executar códigos remotamente se existirem classes que modificam seu comportamento com a desserialização (OWASP, 2017i).

Para prevenir-se desse tipo de falha, a alternativa mais segura é não aceitar objetos serializados de fontes não confiáveis.

2.1.1.9 Utilização de componentes com vulnerabilidades conhecidas

Componentes utilizados por aplicações web como bibliotecas, *frameworks* e outros módulos de software são executados com o mesmo nível de privilégio da aplicação. Se um componente vulnerável é explorado, um ataque pode resultar em perda de dados ou perda de controle do servidor. Aplicações e APIs com vulnerabilidades conhecidas podem enfraquecer as defesas da aplicação e possibilitar diversos tipos de ataques (OWASP, 2017l).

A aplicação é considerada vulnerável se (OWASP, 2017j):

- não se sabe as versões dos componentes utilizados, tanto do lado do cliente quanto do lado do servidor;
- não são realizadas atualizações constantes de plataforma, frameworks e dependências;
- não são realizadas varreduras por vulnerabilidades de dependências regularmente.

Para prevenir-se, é possível (OWASP, 2017j):

- remover dependências não utilizadas e recursos desnecessários;
- monitorar continuamente CVEs e NVDs dos componentes utilizados;
- obter componentes apenas de fontes oficiais, utilizando *links* seguros.

2.1.1.10 Registros e monitoração insuficiente

Quando *logs* e monitoração da aplicação são insuficientes, é mais difícil detectar anormalidades e comportamentos suspeitos nos sistemas, que podem ter sido causados por um ataque. A monitoração da aplicação é insuficiente se (OWASP, 2017a):

- eventos que podem ser auditados, com *logins* e falhas de *logins* não são registrados;
- avisos ou erros não geram mensagens ou geram mensagens inadequadas;
- *logs* de aplicações e APIs não são monitorados quanto à atividade suspeita;
- ferramentas de varredura ativa de segurança não geram alertas.

Para se prevenir desse tipo de falha, é possível: garantir que os logs possam ser consumidos por um gerenciador centralizado, garantir que transações e requisições de alto valor tenham uma auditoria e controle de integridade para garantir alteração ou remoção, estabelecer monitoração eficiente e alertas de forma que atividades suspeitas possam ser detectadas e a resposta possa ser rápida (OWASP, 2017a).

2.2 Desenvolvimento integrado na nuvem

Esta seção busca esclarecer as metodologias DevOps e DevSecOps e seus conceitos relacionados, que foram utilizados na implementação da solução.

2.2.1 DevOps

Os primeiros registros que se tem de discussão da metodologia DevOps foi em 2009, em que John Allspaw e Paul Hammond fizeram uma apresentação intitulada "*10+ deploys a day: Dev and Ops Cooperation at Flickr*". Patrick Debois - interessado no assunto e frustrado com a grande separação que havia entre a equipe de operações e a de desenvolvimento – criou, em conjunto com John e Paul, a primeira conferência para tratar do assunto DevOps, denominada "DevOps days", ocorrida em outubro de 2009. A partir de então, a metodologia DevOps foi adquirindo cada vez mais popularidade até que, em 2011, Cameron Haight, da empresa Gartner, apresentou quais seriam suas expectativas para DevOps nos anos vindouros, o que deu maior destaque ainda ao movimento que estava se iniciando. Não demorou muito até que diversas empresas adotassem as novas práticas propostas (WATTS, 2019a).

Historicamente, equipes de desenvolvimento e de operações têm problemas de comunicação e colaboração. Em parte, isso ocorre pois as equipes têm objetivos, lideranças e responsabilidades distintos. Enquanto a equipe de desenvolvimento cuida do desenvolvimento de novos recursos e funcionalidades para o software, a equipe de operações trabalha para dar suporte ao software, com atividades como provisionamento de infraestrutura, por exemplo (WATTS, 2019b).

A metodologia DevOps integra os mundos de desenvolvimento e operações, utilizando desenvolvimento automatizado e monitoração de infraestrutura. DevOps é uma mudança organizacional, em que, em vez de grupos separados em diferentes silos, executando suas operações separadamente, equipes multifuncionais trabalham em entregar valor ao produto continuamente. Essa abordagem facilita a entrega de valor ao produto mais rapidamente, reduzindo problemas devido à falta de comunicação entre as equipes e aumentando a capacidade de resolução de problemas. Portanto, essa metodologia tem como um de seus objetivos operacionalizar a Integração Contínua e a Entrega Contínua. Com isso, processos das equipes de desenvolvimento e operação se tornam mais rápidos e eficientes, e empresas podem gastar menos e produzir mais (EBERT et al., 2016).

(BASS; WEBER; ZHU, 2015) define DevOps como um conjunto de práticas que têm como objetivo reduzir o tempo entre o envio de código para o sistema e essa alteração ser efetivada em produção, ao mesmo tempo, garantindo alta qualidade.

Nesse sentido, define-se algumas etapas para alcançar uma integração efetiva entre as equipes de desenvolvimento e operações (WATTS, 2019c):

- fase de planejamento - identificação e rastreio: nesta etapa são definidas todas as ideias e fluxos de trabalho que serão utilizados durante o desenvolvimento garantindo, assim, uma visão geral do produto para os gerentes do projeto;

- fase de desenvolvimento: com o auxílio de ferramentas de versionamento de código, desenvolve continuamente o produto, de modo que cada novo recurso desenvolvido possibilite que o desenvolvedor integre essas mudanças ao código existente. É importante, nesta fase, realizar alterações em pequenos lotes, de modo que possíveis problemas possam ser identificados rapidamente, mantendo a aplicação sempre pronta para produção (KIM et al., 2016);

- fase de testes: utilizando práticas de integração contínua, realiza a construção da aplicação e os testes o mais cedo possível dentro do ciclo de vida de desenvolvimento do software;

- fase de implantação: etapa que acontece após resolução de possíveis *bugs*, o software é implantado no ambiente ao qual foi designado;

- fase de gerenciamento: nesta fase, a aplicação já foi implantada e é onde ocorre a monitoração e interpretação dos dados dos usuários, garantindo segurança e alta disponibilidade.

2.2.1.1 Integração Contínua

Integração Contínua é uma prática adotada no desenvolvimento de software em que membros da equipe integram seu trabalho frequentemente. Usualmente essa integração ocorre ao menos uma vez ao dia, levando a múltiplas integrações diárias. Cada integração é verificada por uma construção automática, incluindo testes automáticos para detectar erros o mais rapidamente possível (FOWLER, 2006). Tal ação leva as equipes a detectarem prematuramente possíveis erros no processo de desenvolvimento, proporcionando entrega de valor ao software mais rápida e consistentemente.

Algumas práticas essenciais para estabelecer uma integração contínua eficiente e que podem ser adotadas para estabelecer uma metodologia DevOps de desenvolvimento de software são (FOWLER, 2006):

Manter um único sistema de gerenciamento de código fonte que recepcione todos os arquivos necessários para a construção do software, bem como arquivos de teste e instalação. Assim sendo, quando se quiser executar a aplicação em uma nova máquina, será necessário apenas fazer o download dos arquivos armazenados no SCM. Isso é importante para evitar conflitos entre trabalhos exercidos por diferentes desenvolvedores e a possibilidade de criação de diferentes *branches* com base na linha principal, para desenvolvimento de novos recursos para a aplicação.

O processo para transformar o código em executável, assim como os testes que garantem seu correto funcionamento, deve acontecer de forma automatizada, assim que o desenvolvedor integrar seu código ao SCM. Com isso, o processo de construção da

aplicação e execução dos testes não depende da intervenção humana, além de garantir que todas as dependências e requisitos para execução do software estejam bem definidas.

Define-se as três capacidades que a criação de uma prática de integração contínua deve respeitar (KIM et al., 2016):

- um conjunto de testes automatizados que validem que a aplicação se encontra apta para implantação;
- uma cultura que interrompa toda a linha de produção assim que algum dos testes de validação falhar;
- desenvolvedores trabalhando em pequenos lotes em vez de grandes alterações.

Os testes são uma parte essencial do processo de integração contínua. Quando não há um conjunto de testes automatizados, ao gerar a *build* do código, ou seja, transformar o código fonte em um artefato executável, isso significa apenas que a aplicação pode ser compilada e montada. Portanto, é essencial a execução de testes automatizados, que possam proporcionar a confiança de que a aplicação esteja funcionando (HUMBLE; FARLEY, 2010).

Testes unitários têm como objetivo testar o funcionamento de pequenas partes do código isoladamente. Em geral não necessitam da execução total da aplicação e podem ser executados separadamente. Esse tipo de teste não se conecta com banco de dados e serviços auxiliares (HUMBLE; FARLEY, 2010).

Testes de componente testam o comportamento de diversos componentes da aplicação, que podem ser entendidos como partes ou porções do software em questão. Esse tipo de teste, em geral, também não necessita do funcionamento completo da aplicação. No entanto, costumam utilizar banco de dados e serviços auxiliares (HUMBLE; FARLEY, 2010).

Testes de aceitação são utilizados para garantir que a aplicação está dentro dos critérios de aceitação definidos pelas regras de negócio, incluindo características como capacidade, disponibilidade, segurança, entre outros. Esse tipo de teste têm um melhor desempenho se forem executados em um ambiente similar ao ambiente de produção (HUMBLE; FARLEY, 2010).

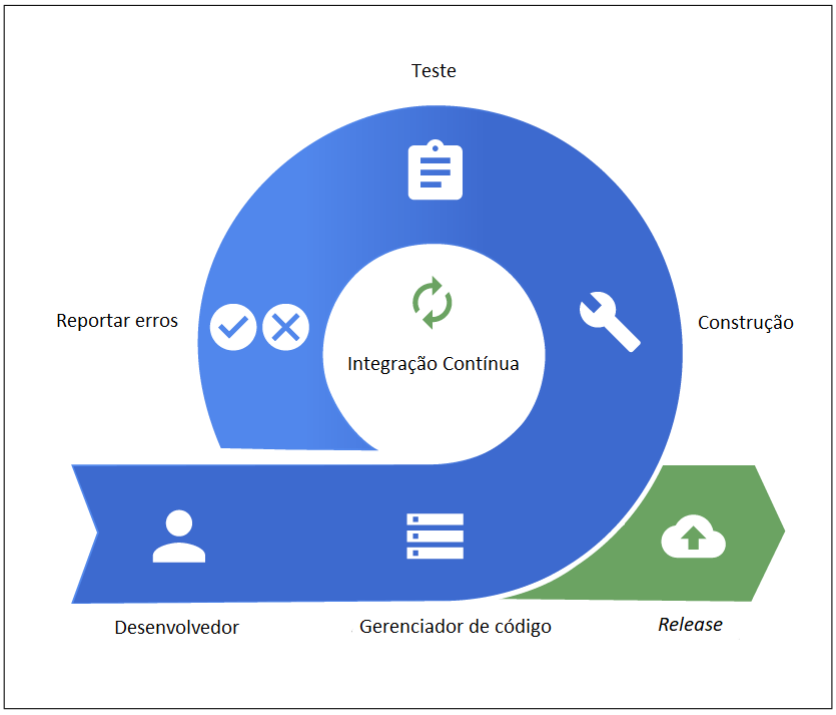


Figura 2.2 – Exemplo de fluxo de *pipeline* de integração contínua. Modificada. Fonte: (PAGERDUTY, 2021)

2.2.1.2 Entrega Contínua

Durante o processo de desenvolvimento de um software, todos aqueles que fazem parte do processo, têm um objetivo em comum: fazer com que o processo de *release*, ou seja, o processo de distribuição do software desenvolvido, seja uma atividade de baixo risco. A melhor maneira de se alcançar tal objetivo é realizar o lançamento da aplicação com a máxima frequência possível (HUMBLE; FARLEY, 2010). Dessa maneira, será possível realizar pequenas mudanças em cada lançamento, diminuindo o risco de causar problemas durante a atividade. Além disso, também torna possível a obtenção de *feedback* quase instantâneos sobre as alterações feitas.

A entrega contínua é uma extensão da integração contínua, em que códigos recém criados são incorporados ao SDLC da aplicação. As construções do software com mudanças que foram automaticamente integradas utilizando a metodologia de integração contínua geram uma nova *release*, que está pronta para ser implantada em qualquer ambiente após testes iniciais, tais como testes unitários automatizados e de integração (WATTS, 2019d). Portanto, para obter uma entrega contínua eficiente, se faz necessário um processo de integração contínua bem estruturado, a fim de que a construção e testes da aplicação sejam processos automáticos e contínuos. O processo de entrega contínua exige também que exista um mecanismo facilmente repetível, replicável e automatizado de geração de artefatos prontos para entrar em produção.

Quando este artefato é lançado automaticamente em produção, é chamado de

implantação contínua. Entretanto, quando a intervenção humana é necessária para lançar artefato em produção, caracteriza-se uma entrega contínua.

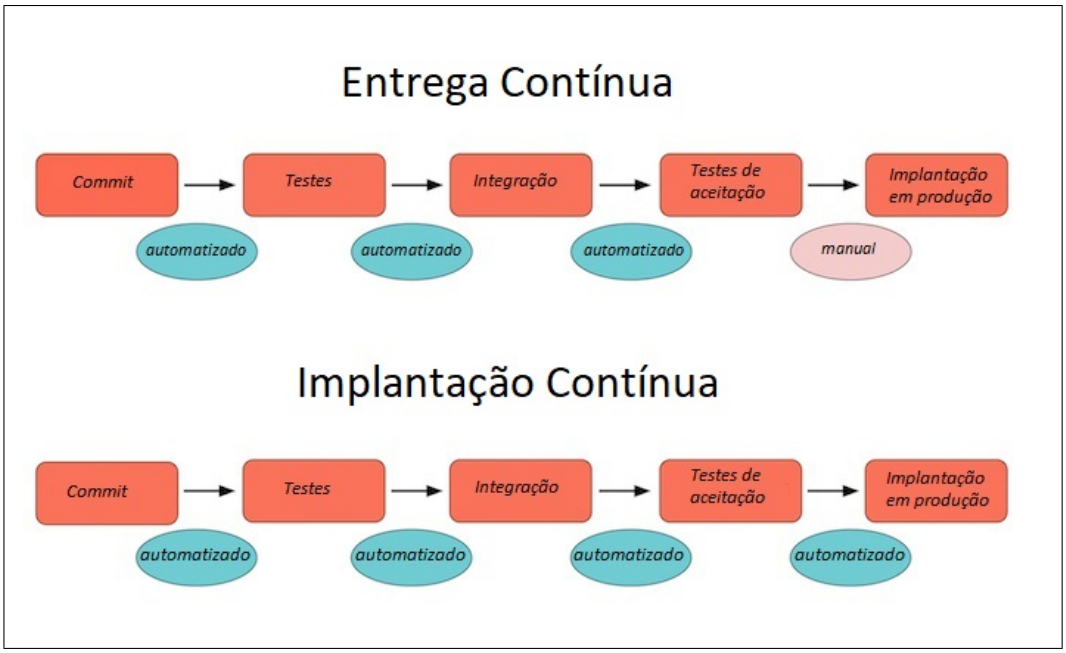


Figura 2.3 – Ilustração do modelo de entrega contínua e implantação contínua. Modificada. Fonte: (PAUL, 2018)

2.2.2 Práticas DevOps

Ao utilizar a metodologia DevOps, existem algumas práticas que são comumente adotadas nos processos de integração e entrega contínua, assim como práticas culturais e de gerenciamento de infraestrutura. Algumas delas são infraestrutura como código, testes contínuos e monitoração contínua.

O design de infraestrutura consiste em definir e configurar a infraestrutura que o software necessita para ser executado, tipicamente envolvendo *scripts* para: instanciar e conectar máquinas, instalar e configurar o software necessário para execução da aplicação, instanciar e executar serviços auxiliares para operação da aplicação (ARTAC et al., 2017). A infraestrutura como código (IaC) é uma prática emergente utilizada para especificar a estrutura subjacente de aplicações e softwares e suas respectivas configurações (SCHWARZ; STEFFENS; LICHTER, 2018).

A utilização da prática de infraestrutura como código, procura-se alcançar os seguintes objetivos (MORRIS, 2016):

- infraestrutura suporta e possibilita mudanças;
- mudanças constantes na infraestrutura;

- é possível definir, provisionar e gerenciar os recursos necessários por meio de código;
- soluções para problemas de infraestrutura podem ser resolvidos por meio de implementação e testagem de código;
- equipes podem recuperar-se fácil e rapidamente de falhas.

A prática de testes contínuos tem como objetivo testar a aplicação durante todo o ciclo de vida, testando continuamente do início ao fim, esses processos são comumente denominados *shift left* e *shift right* (ZIMMERER, 2018).

A realização de testes precoces ao longo do ciclo de vida do software permite reduzir custos com possíveis problemas futuros do software, já que o *feedback* do funcionamento da aplicação é contínuo. Ao integrar testes em *pipelines* de integração contínua, é possível definir explicitamente controles de qualidade de código (*quality gates*), garantindo a qualidade do código e obtendo um *feedback* mais rápido do estado da aplicação (ZIMMERER, 2018).

Já a monitoração contínua tem como princípio monitorar o software em produção, coletando dados das aplicações e notificando a equipe de desenvolvimento sempre que existir algum problema. Essa etapa é essencial para o feedback da performance da aplicação em produção, com métricas como tempo de atividade, respostas do sistema, volumes das transações e estabilidade geral do sistema (BOSE, 2020).

A monitoração contínua de aplicações facilita a resposta a possíveis incidentes, diminui o tempo de inatividade do sistema e traz informações sobre como as alterações realizadas no software afetaram o usuário final.

2.2.2.1 Contêineres

Um contêiner é um nível unitário de software que empacota e abrange código e todas as suas dependências, de forma que a aplicação seja executada de maneira rápida e confiável, ainda que haja uma transição do ambiente de execução, uma vez que uma característica dos contêineres é justamente o isolamento de toda a infraestrutura e ambiente no qual está inserido. Uma imagem de um contêiner é um pacote de software executável, leve e independente que inclui todos os componentes necessários para executar uma aplicação: códigos, ferramentas de sistema, bibliotecas do sistema e configurações. Denomina-se contêiner uma imagem que está sendo executada (DOCKER, 2021b).

Em se tratando de contêineres, é frequente o questionamento sobre o porquê de não se utilizar apenas máquinas virtuais como ambientes de execução de software, quando se pretende separar ambientes distintos. Os contêineres são uma abstração da camada de aplicação, permitindo agrupar conjuntos de código e dependências simultaneamente. Dessa maneira, múltiplos contêineres podem ser executados em uma mesma máquina,

compartilhando o núcleo do sistema operacional (*kernel*) com o próprio sistema hospedeiro e outros contêineres.

Já as máquinas virtuais, por sua vez, abstraem a camada de hardware de uma máquina, ou seja, um mesmo hardware pode hospedar diferentes sistemas operacionais. No entanto, máquinas virtuais comportam sistemas operacionais completos dentro delas, que são administrados por um *hypervisor* ou tecnologia semelhante.

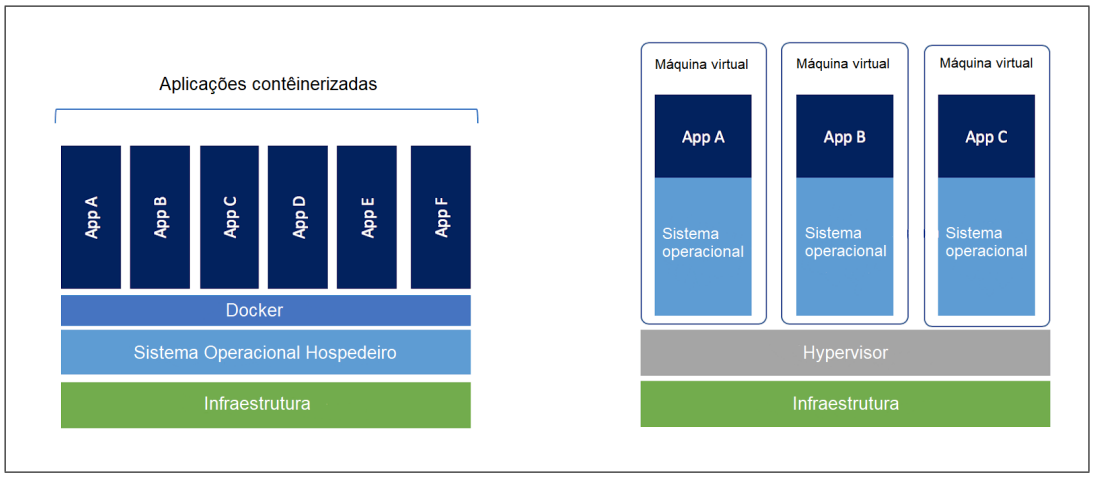


Figura 2.4 – Comparação entre máquinas virtuais e contêineres. Modificada. Fonte: (DOCKER, 2021b)

Contêineres utilizam tecnologias a nível de *kernel* para isolá-los da máquina hospedeira, como *namespaces* e *cgroups (control groups)*. O *user namespace* realiza a separação entre usuários do contêiner e da máquina hospedeira, garantindo que um usuário *root* no contêiner não terá os mesmos privilégios na máquina hospedeira. O *process namespace* é responsável por gerenciar apenas os processos que rodam no contêiner, separando-os dos demais processos que são executados na máquina hospedeira. Já os *control groups* permitem que seja implementado limite e gerenciamento de recursos que os contêineres consomem, como memória, disco e I/O (MERKEL, 2014).

A utilização de contêineres enquanto virtualização permite que a aplicação faça uso apenas de pequenas porções protegidas do sistema operacional, proporcionando uma utilização mais eficiente de recursos. Além disso, criar e destruir contêineres é mais rápido e simples, uma vez que dispensa a necessidade de iniciar ou desligar todo um sistema operacional (MERKEL, 2014).

2.2.3 DevSecOps

O conceito de integrar a área de segurança à metologia DevOps teve seu primeiro registro em 2012, no *blog* intitulado *"DevOps Needs to Become DevOpsSec"*, escrito por Neil MacDonald, analista da Gartner (KUMAR; GOYAL, 2020). Desde então, o tema mais popularmente conhecido como DevSecOps ganha cada vez mais relevância, pois se

tornou claro a necessidade de uma maior preocupação com os quesitos de segurança das aplicações (MYRBAKKEN; COLOMO-PALACIOS, 2017). Francois Raynaud, fundador da conferência DevSecCon, define DevSecOps como sendo a utilização da metodologia DevOps para segurança, difundindo as informações necessárias para toda a equipe envolvida no projeto para que, assim, a segurança seja implementada no tempo correto e da forma correta (CARTER, 2017). Definição semelhante à encontrada em (KUMAR; GOYAL, 2020), onde se define DevSecOps como sendo a utilização do DevOps associado a controles de segurança contínuos com o intuito de garantir, durante todo o ciclo de vida da aplicação, que os quesitos de segurança foram pensados e implementados.

Já o estudo feito sobre o tema de pesquisa DevSecOps realizado por (MYRBAKKEN; COLOMO-PALACIOS, 2017) chegou a conclusão de que as literaturas que estudam e pesquisam sobre este novo modelo de implementação definem DevSecOps como sendo uma evolução necessária do DevOps, tendo o propósito de integrar controles e processos de segurança durante processo de desenvolvimento do *software* ao promover uma maior aproximação e colaboração entre os times de desenvolvimento, operação e segurança.

2.2.3.1 Princípios

Define-se os princípios do DevSecOps, baseados no *CAMS - Culture, Automation, Measurement, Sharing*, com o acréscimo de implementar a segurança no começo do desenvolvimento, ou seja, o *Shift Security Left* (MYRBAKKEN; COLOMO-PALACIOS, 2017).

- **Cultura**: mantendo a mesma ideia da cultura DevOps, onde os times de desenvolvimento de operacionalidade trabalham em conjunto, a cultura DevSecOps agrega a esse conjunto o time de segurança, promovendo assim que os times realizem suas funções com as demandas de segurança definidas. Essa aproximação dos times proporciona, logo no início do projeto, denominado como estágio de planejamento, que a equipe como um todo entenda e aceite que a segurança é uma responsabilidade coletiva. Práticas como definir métricas específicas de segurança, onde todos estejam de acordo, alinha a equipe e proporciona um direcionamento para atingir a segurança almejada;
- **Automatização**: além de automatizar as tarefas de *build*, *deploy* e testes, o DevSecOps implementa também a automatização das tarefas de segurança, permitindo assim que os processos ganhem velocidade operacional e não necessitem de intervenção manual para serem executados. O intuito é automatizar o máximo possível as tarefas de segurança para que essas não se tornem um gargalo na *pipeline* e causem uma queda no rendimento da equipe;

- **Métricas**: monitoração de métricas definidas para analisar como a performance e estabilidade da aplicação se encontram são atividades definidas na prática DevOps. Já para o DevSecOps, adicionam-se também métricas de segurança, que serão utilizadas para indicar como a aplicação responde aos possíveis ataques que podem aparecer ao estar em ambiente de produção;

- **Compartilhamento**: promover a troca de informações entre conhecimentos, ferramentas utilizadas, técnicas e dificuldade entre os times do projeto, para que assim toda a equipe esteja ciente das dificuldades enfrentadas. Com isso, os processos de segurança pensados e desenvolvidos podem ser aprimorados, tornando-os mais direcionados e específicos para as necessidades da equipe;

- **Shift Security Left**: rompendo com a ideia de analisar a segurança da aplicação somente no final dos estágios de desenvolvimento, o *Shift Security Left* integra o time de segurança em todos os estágios de desenvolvimento, desde as etapas de planejamento até a parte de operacionalidade, como podemos ver na Figura 1.1. Isso permite que ao longo de todo o ciclo de vida do projeto os requisitos de seguranças sejam pensados e implementados.

2.2.3.2 Práticas

Algumas das práticas associadas ao DevSecOps são (MYRBAKKEN; COLOMO-PALACIOS, 2017):

- **Modelagem de ameaças e avaliação de risco**: a integração do time de segurança logo no começo do ciclo de vida proporciona que a equipe do projeto tenha detalhado quais os riscos que serão avaliados e traçar metas sobre como poderão ser minimizados. Para isto, o método de modelagem de ameaças consiste em prever com antecedência como os ataques à aplicação poderão ocorrer, servindo assim como um guia para tomada de decisões;

- **Testes Contínuos**: a automatização e implantação das tarefas de segurança ao longo do ciclo de vida do projeto, tornam possível realizar frequentes testes na aplicação. Dessa forma, é possível identificar quando o código foi alterado, propiciando a verificação de quaisquer possíveis fragilidades encontradas. Caso ocorram, impedirá que o código passe para produção ou, ainda, forçará que o código atual volte para uma versão estável, processo esse conhecido como *rollback*;

- **Monitoramento e histórico**: com a definição e implantação das métricas - aliada à automatização dos testes - é possível gerar relatórios contendo evidências de que as medidas de segurança estão sendo o suficiente ou então identificar, através dos

históricos de ocorrências, conhecidos como *logs*, onde e o que está vulnerável à ataques;

- **Segurança como código**: definição de políticas de segurança, tais como os testes integrados, configuração da rede e os seus acessos. Tais políticas possibilitam que definições padrão possam rodar automaticamente em paralelo a outras atividades, garantindo que as configurações iniciais serão respeitadas e implementadas;
- ***Red-Team* e exercícios de segurança**: é atribuição do *Red-Team* identificar os pontos de fragilidade na segurança. Esse time tem como principal objetivo testar, na prática, a aplicação ou software, identificando se foi implementado o conceito da criação de uma equipe responsável por tentar atacar e achar pontos de vulnerabilidades no produto.

2.2.3.3 Testes de segurança

Os testes de segurança são divididos em duas categorias principais. Testes estáticos e testes dinâmicos.

Testes estáticos, também denominados testes de caixa branca são capazes de detectar vulnerabilidades de software mais cedo no ciclo de vida do desenvolvimento da aplicação por meio de análises estática do código, fornecendo ao desenvolvedor alertas no código que potencialmente são vulnerabilidades (ALORAINI et al., 2019). A análise estática requer apenas o código fonte da aplicação, não sendo necessária a sua execução, por esse motivo, não são capazes de detectar falhas de execução da aplicação. No entanto, por detectar as falhas mais cedo, é mais simples e menos custoso corrigir vulnerabilidades detectadas nesta etapa.

Testes dinâmicos, também denominados teste de caixa preta, procuram por vulnerabilidades de segurança por meio da simulação de ataques externos na aplicação, enquanto ela está em execução (PETERSON, 2020). Ao contrário dos testes estáticos, as vulnerabilidades detectadas nesta etapa são mais custosas de serem corrigidas, pois existe a necessidade de uma versão funcional da aplicação.

2.2.4 Comparativo entre DevOps e DevSecOps

Com os conceitos de DevOps e DevSecOps definidos, é possível notar que ambas metodologias possuem uma base similar, é inevitável que ocorra uma comparação entre elas, buscando levantar os pontos positivos e negativos de cada uma. De acordo com um estudo realizado por J. Bjornholm, O DevSecOps ainda não foi amplamente difundido pelo fato dos desenvolvedores considerarem que aumentaria consideravelmente o tempo de execução da *pipeline*, o que consequentemente acarretaria em um aumento no tempo necessário para a aplicação ser disponibilizada (BJÖRNHOLM, 2020).

O estudo então, utilizando de uma aplicação *web* para servir como aplicação base para o teste, descreveu duas *pipelines* distintas: uma sem etapas de verificação de segurança, assemelhando-se ao DevOps, e uma com etapas de verificação de segurança, assemelhando-se ao DevSecOps. O resultado final obtido foi que o tempo da pipeline aumentou em média 33% para sua finalização (BJÖRNHOLM, 2020).

Tal resultado comprova que há sim um aumento no tempo de execução da *pipeline*, o que pode interferir nos princípios das metodologias ágeis e reduzir assim o tempo até a disponibilização da aplicação. Porém, o aumento no tempo de execução da *pipeline* faz valer a ausência de verificações básicas de segurança? Tais discussões serão melhor abordadas no capítulo 5, onde serão utilizados os resultados obtidos neste projeto para embasar as conclusões finais dos alunos-autores acerca do tema desta subseção.

2.3 Melhores práticas de desenvolvimento - Twelve Factor App

Com o crescimento das metodologias ágeis e das práticas DevOps, o desenvolvimento de software sofreu alterações visando atender a disponibilização contínua da aplicação. Com isso, os projetos são comumente pensados e implementados como *SaaS - Software as a Service*, também conhecida como aplicação *web*, o que altera como se deve construir a aplicação para adequá-lá e tornar possível sua futura utilização. Visando os novos desafios a serem enfrentados, surgiu a metodologia *Twelve-Factor App* documentando as boas práticas para construir com eficiência um *SaaS* (WURSTER et al., 2017).

Entendendo e visualizando a necessidade da criação de uma metodologia modelo para outros desenvolvedores terem como base, colaboradores diretamente envolvidos na criação da plataforma em nuvem *Heroku*, utilizaram sua experiência como base para modelar os princípios básicos de desenvolvimento, operação e automação para que a aplicação *web* criada consiga ser executada corretamente (WIGGINS, 2017).

Os princípios dessa metodologia são (WURSTER et al., 2017):

1. Base de código: utilização de uma base de código e vários *deploys*, ou seja, diversos ambientes de execução da aplicação. Cada base de código tem uma correlação de um-para-um com a aplicação;

2. Dependências: explicitamente declarar e isolar dependências. Além disso, utilizar uma ferramenta de isolamento de dependências no ambiente em que a aplicação é executada. Dessa forma, apenas as dependências declaradas na aplicação são utilizadas. Isso é importante para garantir que a aplicação por si só possa ser executada em qualquer ambiente;

3. Configuração: armazenar a configuração no ambiente: separação rígida entre configuração e código, utilização de variáveis de ambiente para armazenamento de configuração. A utilização de variáveis de ambiente permite que configurações sejam facilmente alteradas ao mudar de ambiente, separadamente do código. É importante também que variáveis de ambiente não sejam armazenadas nas bases de código, sendo parâmetros específicos de cada ambiente e nem estejam *hardcoded* no código, pois violaria a premissa de separação de código e configuração;

4. Serviços de apoio: tratar serviços de apoio como recursos anexados. Serviços como banco de dados, cache, entre outros, que conectam-se à aplicação por meio da rede, devem poder ser trocados entre serviços locais (ambiente de desenvolvimento) e serviços externos (ambiente de produção) sem nenhuma mudança no código, apenas na configuração;

5. Construir, lançar e executar: separação rígida entre os diferentes estágios da aplicação. No estágio de construção, ocorre a conversão do código em um executável. No estágio de lançamento, ou *release*, onde o executável resultante do processo de construção junta-se com as configurações de cada ambiente e a aplicação está em um estado pronto para ser executada em determinado ambiente. Já no estágio de execução, a aplicação é executada em um ambiente, utilizando o artefato gerado pelo processo de *release*;

6. Processos: executar a aplicação em um ou mais processos, que não armazenam estado. Ao surgir a necessidade de persistir dados, é necessário um serviço de apoio, como um banco de dados;

7. Vínculo de portas: exportar serviços via vínculo de portas, escutando requisições que chegam até ela. Dessa forma, aplicações e serviços de apoio são acessíveis por meio de uma URL. Esse tópico está relacionado com o tópico 4. Utilizando-se dessa prática, uma aplicação pode facilmente tornar-se serviço de apoio de outro serviço;

8. Concorrência: arquitetar a aplicação para lidar com diversas cargas de trabalho, atribuindo a cada tipo de trabalho um tipo de processo;

9. Descartabilidade: maximizar a robustez com inicialização e desligamento rápido. Os processos da aplicação são descartáveis, podendo ser iniciados ou parados a qualquer momento, facilitando o escalonamento, rápido *deploy* e mudanças de configuração;

10. Paridade entre desenvolvimento e produção: manter os ambientes de desenvolvimento, homologação e produção o mais similares possível. Ou seja, utilizar as mesmas técnicas de *deploy*, serviços auxiliares, versões de dependências e componentes secundários;

11. Logs: tratar *logs* como fluxos de eventos. *Logs* podem prover informações sobre o comportamento do app em execução. Em produção esses *logs* devem ser coletados e encaminhados para visualização ou armazenamento;

12. Processos administrativos: executar tarefas de administração/gestão, como migrações de banco de dados, por exemplo, em processos pontuais e em ambientes idênticos.

3 Arquitetura proposta e Implementação

Neste capítulo, será evidenciado o problema abordado e uma descrição da arquitetura para a aplicação utilizada como estudo de caso e para a construção da *pipeline* de integração e entrega contínua, seguido da implementação de cada passo realizado. Ao final do capítulo, será possível compreender as ferramentas utilizadas no projeto, como elas se relacionam entre si, como elas foram utilizadas para resolver o problema abordado e a implementação de cada etapa da *pipeline* e o papel de cada uma na implementação de uma solução DevSecOps.

Os resultados referentes a cada etapa estão descritos no capítulo 4.

3.1 Descrição do problema e proposta de solução

Nesta seção, será feita uma descrição do problema referente à segurança de aplicações e uma proposta para solucioná-lo, definindo as 10 etapas que foram utilizadas como base para implementação da *pipeline*.

3.1.1 Descrição do problema

Aspectos relacionados à segurança de aplicações eram comumente deixados para o final do ciclo de vida do desenvolvimento de software. No entanto, a testagem de segurança em estágios iniciais do ciclo de vida é de extrema importância para economizar custos ao final do ciclo de vida da aplicação e garantir que as testagens de segurança foram cobertas antes de realizar o *deploy* da aplicação.

Nesse trabalho serão implementados conceitos de DevSecOps, por meio do estudo de caso de uma aplicação que terá seus processos de construção, montagem e execução automatizados, por meio de uma *pipeline* de integração contínua e entrega contínua, abrangendo testes de segurança no início do ciclo de vida do desenvolvimento do software.

3.1.2 Arquitetura proposta

A *pipeline* foi definida em 10 etapas, como evidenciado na figura 3.1.

As etapas de construção, controle de qualidade e testes são partes essenciais para a implementação de uma integração contínua, como mencionado na subseção 2.2.1.1. O funcionamento correto da aplicação, a integração com os serviços auxiliares e a adequação do código-fonte a padrões pré-definidos são garantidos nestas etapas.

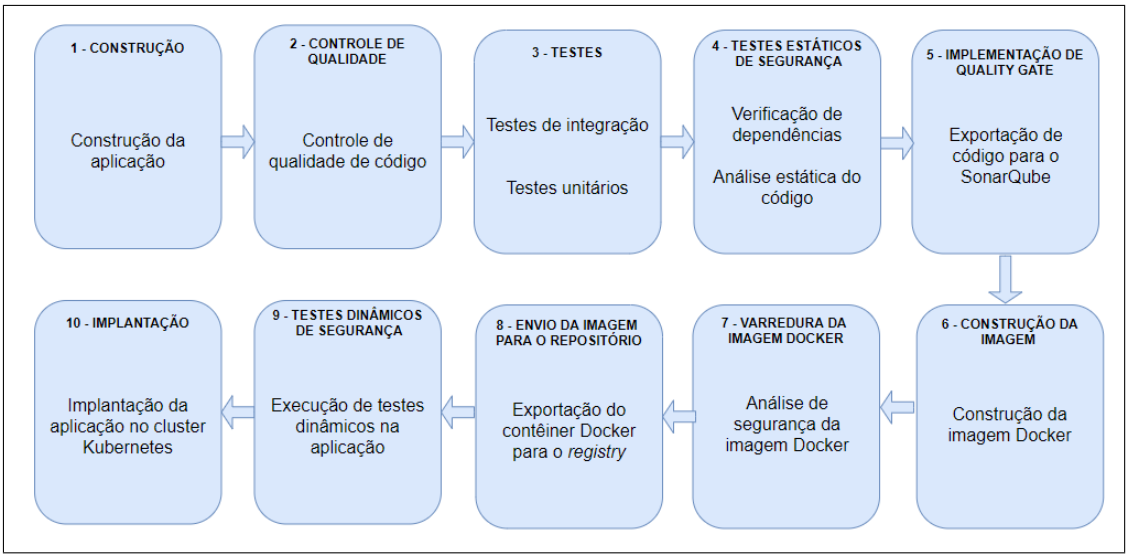


Figura 3.1 – Proposta de estágios para desenvolvimento de pipeline DevSecOps.

A etapa de verificação estática de segurança é dividida em duas etapas: verificação de dependências e análise estática do código.

A etapa de verificação de dependências realiza uma varredura em todas as dependências da aplicação, buscando por vulnerabilidades conhecidas e reportadas contidas nessas bibliotecas. Esse estágio tem como finalidade prevenir-se do componente número 9 do OWASP Top 10, utilização de componentes com vulnerabilidades conhecidas, descrito na subseção 2.1.1.9.

Já a etapa de verificação estática de código realiza uma busca por padrões de código considerados inseguros. Essa etapa previne diversas vulnerabilidades elencadas no OWASP Top 10, como injeção de código (2.1.1.1), XSS (2.1.1.7), configurações incorretas de segurança (2.1.1.6) e exposição de dados sensíveis (2.1.1.3).

Na etapa seguinte, é utilizada uma ferramenta de gerenciamento de código para que atue como centralizador dos códigos desenvolvidos. Nesta etapa são definidos *quality gates*, que podem ser entendidos como políticas de qualidade da aplicação, e podem ser definidos pela equipe de desenvolvimento. Os *quality gates* definem padrões de qualidade ao código, relacionados a cobertura de código, *code smells* e falhas de segurança.

Ao passar pelos testes estáticos e controles de qualidade, a imagem Docker está pronta para ser gerada. Na etapa de construção de imagem ela é criada.

Na etapa de verificação de vulnerabilidades em imagens Docker é realizada uma varredura na imagem utilizada como base para construir a aplicação. Essa etapa tem como finalidade encontrar vulnerabilidades ou até *malwares* contidos na imagem.

Mesmo com o isolamento entre contêiner e máquina hospedeira, a utilização de contêineres não garante a segurança da aplicação. A utilização de imagens com paco-

tes vulneráveis e imagens maliciosas podem comprometer a segurança da aplicação. Um exemplo de falha de segurança relacionada a contêineres foi identificada no CVE-2019-5736, em que a utilização de um contêiner malicioso poderia romper o isolamento entre máquina hospedeira e contêiner, dando ao atacante privilégios de usuário *root* a nível de execução de código na máquina hospedeira (NVD, 2021).

Além disso, contêineres utilizam o mesmo núcleo (*kernel*) da máquina hospedeira, o que pode fazer com que imagens que contenham códigos maliciosos possam explorar vulnerabilidades de *kernel*, que são constantemente descobertas e exploradas. Apenas em 2019, 170 vulnerabilidades de *kernel* Linux foram descobertas (DETAILS, 2019).

Após essa etapa, garantindo a ausência de vulnerabilidades na imagem Docker, a imagem é enviada para o repositório de imagens Docker (*registry*).

No passo seguinte, são executados os testes dinâmicos de segurança na aplicação, como definido em 2.2.3.3.

Com isso, a imagem já pode ser implantada no cluster Kubernetes, passo executado na etapa de Implantação. Este passo executa o que foi definido em 2.2.1.2 como Entrega Contínua.

As etapas aqui citadas têm como finalidade implementar o *workflow* DevSecOps, conforme definido na introdução (1), em que testes de segurança são realizados em todas as etapas do processo. Desde o planejamento de uma aplicação segura, verificação de qualidade de código, testes estáticos antes da construção da aplicação, testes dinâmicos após a construção da aplicação e, por fim, a implantação.

Vale ressaltar que erros ou vulnerabilidades encontrados em quaisquer dos passos citados farão com que o processo seja interrompido, até que o desenvolvedor corrija tais pendências.

3.2 Construção do ambiente

Nesta seção serão descritos aspectos referentes ao desenvolvimento da aplicação modelo para a implementação da solução, sua arquitetura e ferramentas utilizadas para o desenvolvimento.

3.2.1 Desenvolvimento da aplicação modelo

A aplicação desenvolvida no projeto conta com três frentes, sendo um *back-end* e dois *front-end*. O *front-end* é dividido em *web* e *mobile* e caracteriza-se por tudo aquilo que o usuário consegue interagir em uma aplicação, ou seja, o código do lado do cliente. Já o *back-end* é o código do lado do servidor, que está por trás da interface do usuário, contendo também o acesso a serviços auxiliares, como bancos de dados.

Esta aplicação fornece um ambiente completo aos alunos do Departamento de Engenharia Elétrica (ENE), proporcionando a marcação de horários com professores registrados. Utilizando de uma aplicação móvel (*mobile*), os alunos visualizam os professores e seus respectivos horários disponíveis, podendo marcar um desses horários. Já os professores cadastrados no ambiente possuem uma *dashboard* web para controlar quais dos seus horários estão preenchidos, informando também quem é o aluno a ser atendido.

3.2.2 Divisão da estrutura - Repositórios e *pipelines*

Para cada frente do projeto - *back-end*, *web* e *mobile* - foi criado um repositório no Gitlab do LATITUDE-UnB cada um com seu respectivo arquivo de definição de *pipeline*.

As *pipelines* descritas e implementadas para *web* e *back-end* são muito semelhantes, seguindo os passos descritos nas próximas sessões. Já a *pipeline* implementada para o *mobile*, por ser diferente das demais e seguir outro fluxo, será explicada na seção 3.4.11, exclusiva para tal.

3.2.2.1 Linguagem base

A linguagem *JavaScript*, anunciada em 1995 pela companhia Netscape, vem sendo amplamente utilizada por desenvolvedores web desde então. Fornecendo uma abordagem orientada a objetos, o *JavaScript* passou a ser implementada na criação de várias aplicações, não sendo mais atrelada somente à web (JENSEN; MØLLER; THIEMANN, 2009). Essa linguagem é a mais utilizada no *Github*, sendo a base para vários projetos hospedados na plataforma (ZAPPONI, 2021).

A aplicação utilizada no projeto foi desenvolvida utilizando uma *stack* JavaScript, ou seja, implementação de *software* e bibliotecas que rodem utilizando a mesma linguagem base, composta por: NodeJS, React e React Native.

3.2.2.2 Back-end desenvolvido utilizando NodeJS

O NodeJS é um software de código aberto que utiliza do interpretador de JavaScript, desenvolvido pela Google, conhecido como V8, para execução *server-side*. Também caracterizado como um ambiente de execução assíncrono orientado a eventos, o NodeJS é capaz de realizar a tratativa de requisições simultâneas sem que elas causem um bloqueio no funcionamento do servidor (FOUNDATION, 2021).

3.2.2.3 Bancos de dados

Para complementar a utilização do NodeJS, responsável por atender as chamadas HTTP realizadas ao servidor (GET, POST e PUT), foi implementado três tipos de bancos da dados para o armazenamento das informações necessários da aplicação. São eles:

- PostgresSQL: Banco relacional, ou seja, com estruturas e esquemas bem definidos, guardando assim todos os dados em tabelas previamente estabelecidas;
- MongoDB: Banco não relacional, ou seja, não há uma estrutura e esquemas bem definidos, o que proporciona assim uma maior performance. O banco MongoDB utiliza de armazenamento do tipo documento, assemelhando-se a um formato encontrado em arquivos JSON;
- Redis: Banco chave-valor, um tipo de banco de dados não relacional que armazena dados como um conjunto de pares, onde a chave se torna um identificador único.

3.2.2.4 Front-end web desenvolvido utilizando React

O React é uma biblioteca *open source* utilizada para criação de interface de usuário, desenvolvida pelo Facebook. É utilizada como base para criação de *Single Page Application - SPA*, ou seja, aplicações que contém apenas uma página, com o objetivo de melhorar a experiência do usuário, proporcionando assim a manipulação da *Document Object Model - DOM* do navegador e, consequentemente, a possibilidade de escrever códigos para a web que antes necessitariam de HTML e CSS, utilizando somente do JavaScript (FACEBOOK, 2021b).

3.2.2.5 Front-end *mobile* desenvolvido utilizando React Native

Extensão do React desenvolvida para proporcionar a utilização da biblioteca no ambiente mobile. Utilizando de uma linguagem não nativa aos celulares, o React Native possibilita aos desenvolvedores a criação de aplicações nativas multiplataforma, ou seja, utilizando apenas o JavaScript é possível gerar a versão iOS e Android da aplicação como se a mesma tivesse sido desenvolvida nas linguagens base dos sistemas em questão (FACEBOOK, 2021c).

3.2.2.6 Gerenciador de pacotes

A comunidade de programadores que utilizam do JavaScript em seus programas contam com gerenciadores de pacotes contendo pedaços de códigos prontos para serem reutilizados, proporcionando assim que a comunidade consiga integrar pedaços de códigos de fontes terceiras e utilizar no código. Tais gerenciadores contam com mais de 300 mil pacotes disponíveis para utilização em suas bibliotecas, como é o caso do *Node Package Manager - NPM* (FACEBOOK, 2021d).

No projeto foi utilizado do gerenciador de pacotes Yarn, versão 1.22.4, desenvolvido em colaboração do Facebook, Google, Exponent e Tilde, com sua primeira versão disponibilizada em 2016. O Yarn foi escolhido por ser um gerenciador mais rápido, confiável e seguro em comparação ao NPM (FACEBOOK, 2021d), contendo menos pacotes

disponíveis mas proporcionando uma melhor verificação sobre sua biblioteca disponível, se tornando assim uma opção mais atrativa para um projeto que preza pela segurança e confiabilidade das dependências utilizadas.

3.2.2.7 Arquivo de configurações

Nas três frentes do projeto encontra-se o arquivo *package.json*. Este arquivo é responsável por armazenar informações relevantes, como por exemplo qual é a versão do NodeJS que foi utilizado no projeto e a descrição dos *scripts* desenvolvidos, como o *script* que realiza os testes unitários ou de integração, explicado em 3.4.3. É neste arquivo que também se encontra a lista de dependências do projeto, ou seja, quais bibliotecas e suas respectivas versões foram utilizadas. Ao se utilizar do Yarn para adicionar novas dependências, tais informações também ficarão neste arquivo.

Com a listagem de quais dependências são necessárias no projeto, ao utilizar do comando "yarn", explicado em 3.4, é criado um novo diretório chamado *node_modules* na raiz do projeto. Dentro desse diretório criam-se outros diretórios baseados nas dependências listadas no arquivo *package.json*, e dentro dessas pastas estão os códigos base para o correto funcionamento das dependências.

3.2.2.8 Nginx

Para a disponibilização do acesso à aplicação web desenvolvida, utilizou-se o Nginx *Ingress controller* para fazer o balanceamento de carga HTTP dos recursos do Kubernetes, que será explicado a seguir, disponibilizando a chamada às APIs e também o acesso à aplicação via *browser* (NGINX, 2021).

3.2.2.9 Kubernetes

Por fim, para o controle e gerenciamento dos contêineres utilizados no projeto, foi utilizado o Kubernetes. Tal software atua como um orquestrador de contêineres e provê gerenciamento de recursos e serviços (KUBERNETES, 2021a). A utilização e implementação do Kubernetes no projeto será abordado com maior profundidade na subseção 3.4.10.

3.2.2.10 Arquitetura da solução proposta

A estrutura final da arquitetura implementada para atender ao ambiente descrito até então, juntando todos os software e bibliotecas, segue como a exemplificada na Figura 3.2.

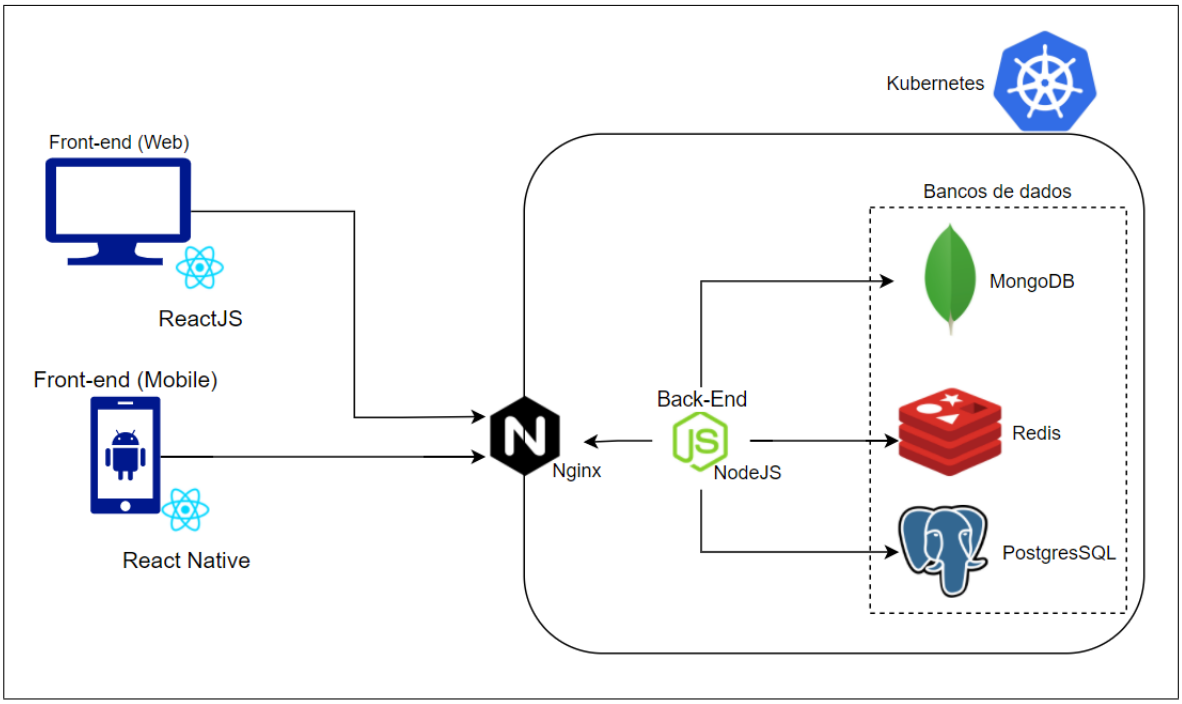


Figura 3.2 – Arquitetura da aplicação proposta.

3.3 Ferramenta para Integração e Entrega contínua

Existem diversas ferramentas que possibilitam a implementação de integração e entrega contínua. Algumas delas são (KATALON, 2020):

- Gitlab;
- Github;
- CircleCI;
- TeamCity;
- Bamboo.

Para a finalidade desse trabalho, foi escolhida a plataforma GitLab, disponibilizada pelo laboratório Latitude. O GitLab é uma plataforma que permite gerenciar o código e criar *pipelines* de CI/CD utilizando apenas uma ferramenta, dispensando a necessidade de uso de gerenciadores externos como GitHub ou BitBucket.

A configuração dos mecanismos de integração e entrega contínua do GitLab se dá por meio de um arquivo denominado *.gitlab-ci.yml*, localizado na raiz do diretório. As instruções definidas nesse arquivo YAML são executadas a cada alteração realizada no repositório.

Uma *pipeline* consiste em um ou mais estágios. Cada estágio contém um ou mais *scripts*, denominados *jobs*, que podem ser executados em sequência ou em paralelo, por meio de um agente denominado *GitLab Runner*.

GitLab runner é a aplicação que executa junto com o GitLab para executar *jobs* definidos na pipeline. Um *runner* está associado a um *executor*, que refere-se ao ambiente no qual o *job* é executado. Neste projeto foi utilizado o GitLab *runner* versão 13.10.0.

Existem diferentes *runners* disponibilizados pelo GitLab (GITLAB, 2021a). Entre eles:

- SSH;
- Shell;
- Parallels;
- VirtualBox;
- Docker;
- Docker Machine;
- Kubernetes.

Para execução do projeto, escolheu-se utilizar o Docker *executor*. Com ele, é possível criar um serviço com ambientes dependentes para executar determinado *job*, o que é útil neste projeto para executar testes que dependem de banco de dados e realizar a construção da imagem Docker, por exemplo. O Docker executor traz também a facilidade de simplesmente definir as imagens que serão utilizadas para rodar os *jobs* e elas são automaticamente baixadas de um determinado repositório externo.

Para a *pipeline* executada neste projeto, foi utilizada a imagem Docker node:12.18.0 como base para todos os estágios da pipeline, mesma imagem utilizada como base para construção das aplicações *back-end* e *web* ou seja, todos os scripts definidos nos estágios da *pipeline* são executados nessa imagem, exceto aqueles explicitamente definidos para utilizar outra.

Na definição dos *jobs* do GitLab, foi utilizado constantemente *cache* e *artifacts*, definidos nos códigos 1 e 2, respectivamente.

```
1       cache:
2         paths:
3           - node_modules/
```

Código 1: Cache de dependências

```
1       artifacts:
2         paths:
3           - [nome_do_arquivo]
```

Código 2: Geração de artefato

A palavra chave **cache**, no GitLab, tem como objetivo prover um mecanismo de cache, que pode ser utilizado para diminuir o tempo de execução dos **jobs**. Isso acontece por meio da reutilização do conteúdo gerado por um *job* previamente executado (GITLAB, 2020a). A utilização do cache no projeto é útil para carregar as dependências da aplicação através dos diferentes estágios da *pipeline*.

Já a palavra chave **artifacts** tem como objetivo armazenar o resultado dos *jobs* executados (GITLAB, 2020b). A utilização dos *artifacts* no projeto tem como finalidade principal armazenar os relatórios dos testes realizados na aplicação. Esses relatórios ficam disponíveis para visualização e *download* na interface do GitLab, permitindo ao desenvolvedor *feedback* instantâneo dos testes.

3.4 Implementação da solução

Nesta seção serão descritas com detalhes as etapas implementadas na *pipeline*, definidas na figura 3.1.

3.4.1 Etapa 1 - Construção

Esta etapa tem como objetivo realizar o *download* de todas as dependências necessárias para executar a aplicação. Para isso, define-se o *script* definido no código 3. O comando *yarn*, na linha 4, lê o arquivo *package.json* definido na raiz do projeto e realiza o download de todas as dependências contidas neste arquivo. O *download* é feito apenas uma vez na *pipeline* e o diretório *node_modules*, que contém os códigos das dependências, é passado para os outros *jobs*.

```
1     run_dependencies:
2       stage: build
3       script:
4         - "yarn"
5       cache:
6         paths:
7           - node_modules/
8         policy: push
```

Código 3: Execução da etapa de construção

3.4.2 Etapa 2 - Controle de qualidade

Esta etapa realiza a análise de qualidade do código fonte recém adicionado ao repositório. Essa verificação, comumente chamada de *lint*, garante que o código que foi integrado à aplicação está de acordo com normas de estilo da linguagem, reduzindo erros e melhorando a qualidade geral do código. A utilização de ferramentas de *lint* pode acelerar o desenvolvimento e reduzir custos, especialmente em linguagens interpretadas, que não passam por uma etapa de compilação, como JavaScript e Python, ao descobrir erros mais cedo no ciclo de desenvolvimento (BELLAIRS, 2019). Por esse motivo a etapa de *lint* ocorre logo após a construção da aplicação.

Foi utilizado o ESLint, versão 6.8.0, que é uma ferramenta de análise estática de código para descobrir rapidamente problemas no código, podendo ser executado em uma *pipeline* de integração contínua. Além disso, também é customizável, podendo utilizar analisadores customizáveis e criar as próprias regras de análise (ESLINT, 2019).

O *job* foi definido no GitLab como ilustrado no código 4.

```
1    lint_test:
2      stage: quality_control
3      script:
4        - "yarn lint"
5      cache:
6        paths:
7          - node_modules/
8        policy: pull
9      artifacts:
10       paths:
11         - report.html
```

Código 4: Execução da etapa de controle de qualidade

A linha 3 define o script a ser executado dentro do contêiner. O comando **yarn lint** executa o seguinte comando, definido no arquivo **package.json**: "eslint -f html -o report.html .", que executa uma verificação do ESLint, exportando um relatório em HTML da análise, para fornecer um *feedback* imediato para o desenvolvedor.

3.4.3 Etapa 3 - Testes unitários e de integração

Nesta etapa se pretende realizar os testes unitários e de integração na aplicação. Os testes unitários, aplicados na frente *web* buscam testar os componentes desenvolvidos. Por exemplo, testar se um botão criado pode ser clicado e se ao ser ativado o mesmo é capaz de realizar alguma ação definida, que causaria uma reação na página. Já os testes do *back-end* realizam os testes de integração, ou seja, testam a integração e a resposta dos *endpoints* da aplicação com os serviços auxiliares.

A ferramenta utilizada no projeto para realizar tais testes no projeto foi o Jest versão 26.1.0. Atualizada constantemente pelo Facebook, o Jest possibilita realizar ambos os testes mencionados em aplicações que utilizem da linguagem JavaScript (FACEBOOK, 2021a).

Para os testes *web*, foi criado o *script* definido no código 5.

```
1     tests:
2         stage: tests
3         script:
4             - "yarn test:coverage"
5         artifacts:
6             when: always
7         paths:
8             - coverage/coverage-final.json
9             - coverage/lcov.info
10            - reports/test-reporter.xml
```

Código 5: Execução da etapa de testes unitários para *web*

A linha 3 define o script a ser executado no *job*. Na linha 4 executa-se o comando **yarn test:coverage**, definido no arquivo **package.json**. Este comando será responsável por utilizar do Jest implementado no código para realizar os testes do projeto, fazendo uso dos arquivos de testes escritos pelos desenvolvedores.

A execução dos testes gera relatórios contendo quanto e quais partes do código foram testados.

O script utilizado para realizar os testes do *back-end* está definido no código 6.

Para execução dos testes de integração, utiliza-se dos três serviços auxiliares utilizados pela aplicação: Redis, Mongo e Postgres, definidos nas linhas 4 a 6. Feito isso, executa-se o comando **yarn test**, que executa o seguinte comando do Jest: jest –coverage –forceExit, que executa os testes exportando um arquivo de cobertura de código.

Detalhes da definição dos testes encontram-se no anexo C.

```
tests:
stage: test
services:
    - postgres
    - mongo
    - redis:alpine
script:
    - "yarn test"
cache:
    paths:
        - node_modules/
    policy: pull
artifacts:
    when: always
    paths:
        - tests/coverage/test-reporter.xml
        - tests/coverage/lcov.info
```

Código 6: Execução da etapa de testes de integração para o *back-end*

3.4.4 Etapa 4 - Testes estáticos

Nesta etapa, pretende-se realizar testes estáticos de segurança automatizados, como definido na subseção 2.2.3.3, permitindo que a pipeline seja interrompida se houver alguma falha detectada.

Para alcançar esse objetivo, foram executados dois tipos de verificação: verificação de vulnerabilidade de dependências e verificação estática do código fonte. Estas etapas ocorrem em paralelo na *pipeline*, proporcionando uma execução mais rápida.

3.4.4.1 Verificação de dependências

Para executar a verificação das dependências, foi utilizada a ferramenta OWASP dependency-check, versão 6.0.5, que é uma ferramenta SCA (Software Composition Analysis) que tenta detectar vulnerabilidades divulgadas publicamente nas dependências de um projeto. Isso é feito determinando se existe um identificador CPE (Common Platform Enumeration) para a dependência. Se existir, um relatório é gerado, associando CVEs (Common Vulnerability Exposure) às dependências vulneráveis (OWASP, 2019).

As bibliotecas instaladas na aplicação são executadas com privilégio total na aplicação, permitindo acessar dados, escrever em arquivos e acessar a internet. Ou seja, uma vulnerabilidade em uma dessas bibliotecas poderia enfraquecer a segurança de toda a aplicação (SECURITY, 2014). Um exemplo de vulnerabilidade de dependência descoberta em 2021, que poderia representar grandes riscos para quem a utilizasse, é a vulnerabilidade do Netmask, em que uma interpretação imprópria de um endereço IP poderia levar a inclusão de arquivos remotos por um possível atacante (HASHIM, 2021) (CVE, 2021).

A verificação do Dependency Check utiliza o banco de dados fornecido pelo NVD (DATABASE, 2021). Ou seja, sempre que for executado na *pipeline*, o *download* do banco de dados é executado, permitindo a verificação sempre com os CVEs mais recentes.

Para execução desta etapa, com o código do OWASP Dependency-Check no repositório, executou-se os passos definidos no código 7.

```
1     dependency_scanning:
2         image: openjdk:8-jre-slim
3         stage: scan_dependencies
4         dependencies:
5             - tests
6         script:
7             - dependency-check/bin/dependency-check.sh -f JSON -f HTML \
8               --project "AgendaENE" -s . -o tests/coverage  \
9               --data dependency-check/data
10        cache:
11            paths:
12                - node_modules/
13        artifacts:
14            when: always
15            paths:
16                - tests/coverage/dependency-check-report.html
17                - tests/coverage/dependency-check-report.json
```

Código 7: Execução da verificação de dependências

A linha 2 define a imagem utilizada neste *job*, openjdk:8, imagem Java utilizada para executar o *dependency-check*. As linhas 7 a 9 executam a verificação, utilizando os formatos JSON e HTML como saída para o relatório, nomeando o projeto como AgendaENE e exportando os dados para o diretório tests/coverage, que veio do *job* anterior, por meio das dependências definidas na linha 4. Dessa forma, o relatório dos testes da aplicação e dos testes de segurança encontram-se no mesmo diretório.

A linha 14 garante que os artefatos sejam exportados mesmo que o *job* não tenha sucesso.

3.4.4.2 Verificação estática de segurança do código

A verificação estática de segurança do código tem como objetivo descobrir vulnerabilidades analisando somente o código fonte da aplicação, como definido em 2.2.3.3. Para execução desta análise, foi utilizada a ferramenta NodeJS Scan (ABRAHAM, 2021b), versão 4.4, que utiliza as bibliotecas libsast (ABRAHAM, 2021a) e semgrep (CORP, 2021) para procurar por padrões de código inseguros, utilizando como base as vulnerabilidades descritas no OWASP Top 10. Para implementação desta etapa, foi utilizado o código 8.

```
1     sast:
2         image: python
3         stage: test
4         before_script:
5             - pip3 install --upgrade njsscan
6         script:
7             - njsscan ./src --html -o sast_result.html
8         artifacts:
9             when: always
10            paths:
11                - sast_result.html
```

Código 8: Execução de verificação estática de segurança

Como o NodeJSScan foi escrito em Python, utilizou-se uma imagem Python para executar esse *job*. Realiza-se a instalação do pacote **njsscan** na linha 5, seguido da execução da verificação na linha 7, executando a verificação estática na raiz do projeto e gerando um relatório em HTML para análise do desenvolvedor.

3.4.5 Etapa 5 - Implementação de *quality gate*

Nesta etapa, pretende-se realizar a análise estática do projeto, também permitindo que a *pipeline* seja interrompida caso o código não atenda aos requisitos mínimos de qualidade do código, definido como *quality gate*.

3.4.5.1 Ferramenta de análise estática de código e implementação de *quality gate*

A utilização de ferramentas de análise estática automática, também conhecidas como *Automatic Static Analysis Tools - ASATs*, promove uma melhora do produto desenvolvido, identificando e alertando falhas no projeto sem ter que rodar a aplicação. Ao se realizar uma varredura no código base desenvolvido, essas ferramentas são capazes de identificar falhas de segurança e más práticas de programação (MARCILIO et al., 2019).

A ASAT implementada no projeto foi o SonarQube, versão 8.6.1, uma ferramenta *open-source* desenvolvida pela SonarSource em 2008. Sendo capaz de se integrar com o Gitlab, foi utilizada a versão gratuita do software, hospedado em uma máquina virtual fornecida pelo Latitude-UnB, para assegurar que o projeto corresponda ao parâmetros aceitáveis definidos pelo Sonar (SONARSOURCE, 2021a).

Tais parâmetros de aceite são chamados de políticas de qualidade, também conhecidos como *quality gate* (SONARSOURCE, 2021b). Para o projeto foram implementados as políticas de não possuir linhas duplicadas, considerado uma má prática de programação, pelo menos 75% de cobertura de código, não possuir mais de 10 possíveis falhas no código, conhecido também como *code smell* e não possuir nenhuma dependência com

vulnerabilidade de risco médio, alto ou crítico sendo utilizada no código. A análise de dependências é feita por meio da análise do relatório de dependências obtido na etapa anterior. Com isso, ao escanear o código, o SonarQube irá informar se tais parâmetros foram atendidos para só então dar prosseguimento a *pipeline*. Caso ocorra uma falha no *quality gate* a ferramenta travaria a *pipeline* e exibiria em quais arquivos estão sendo encontradas falhas, facilitando para o desenvolvedor entender onde está o erro e corrigi-lo.

Para execução desta etapa, foi necessário a utilização de duas dependências instaladas nos códigos do *web* e do *back-end*, sendo elas o *sonarqube-scanner* (BELLINGARD, 2021) e *jest-sonar-reporter* (WLATSCHIHA, 2021), juntamente com a implementação do *plugin dependency check* para o SonarQube (CHECK, 2021). O *jest-sonar-reporter* proporciona que o resultado obtido ao executar os testes utilizando do Jest, explicado em 3.4.3, seja convertido para um formato no qual o SonarQube é capaz de interpretar. Já o *sonarqube-scanner* possibilita uma análise facilitada em projetos que utilizem da linguagem JavaScript, não sendo necessária a instalação ou utilização de outras ferramentas externas específicas. Por fim, integrou-se o *plugin dependency check* no SonarQube, proporcionando assim que a ferramenta consiga interpretar o resultado gerado na etapa explicada na subseção 3.4.4.1, enriquecendo assim as informações disponíveis em sua *dashboard*.

Para execução desta etapa executou-se os passos definidos no código 9.

A linha 4 define a imagem utilizada neste *job*. Uma imagem fornecida pela própria empresa criadora do SonarQube, a SonarSource, responsável por executar as operações necessárias para os *scripts* descritos a seguir e por fim conectar os resultados com a instância do SonarQube rodando na infraestrutura do Latitude-UnB. A linha 7 permite que o *job* tenha acesso aos arquivos gerados na etapa anterior, a etapa de testes. A linha 9 informa a localização que a tarefa da análise fica salva em cache. A linha 10 define o endereço IP e a porta em que a ferramenta está rodando no Latitude-UnB.

Já as linhas 16 a 29 são os parâmetros necessários para utilização da ferramenta no código, seguindo a documentação definida em (SONARSOURCE, 2021c). A linha 16 apresenta o parâmetro de login, onde é passado o *token* de autenticação do usuário do SonarQube com permissão de execução de análise do projeto. A linha 17 é definido o nome do projeto que será exibido na *dashboard* da ferramenta. Já a linha 18 define como verdadeiro o parâmetro que irá forçar o SonarQube esperar a verificação e medição do *quality gate*, interrompendo a *pipeline* caso as políticas não sejam atendidas. A linha 19 apresenta os parâmetros *sources* e *tests* que definem, respectivamente, a pasta do projeto que estão os arquivos principais da aplicação e a pasta do projeto onde estão os arquivos escritos para realização dos testes.

As linhas 20 e 21 definem quais diretórios e arquivos do projeto serão levados em conta para análise. Da linha 23 à 26, utilizando das dependências e *plugin* explicado

```
sonarqube-check:
  stage: sonar
  image:
    name: sonarsource/sonar-scanner-cli:latest
    entrypoint: [""]
  dependencies:
    - tests
  variables:
    SONAR_USER_HOME: "${CI_PROJECT_DIR}/.sonar"
    SONAR_HOST_URL: "http://$SONAR_IP:9000"
  cache:
    key: "${CI_JOB_NAME}"
    paths:
        - .sonar/cache
  script:
    - sonar-scanner -Dsonar.login=$SONAR_TOKEN
    -Dsonar.projectKey=AgendaENE-FrontEnd
    -Dsonar.projectName=AgendaENE-FrontEnd
    -Dsonar.qualitygate.wait=true
    -Dsonar.sources=src -Dsonar.tests=src
    -Dsonar.inclusions=**
    -Dsonar.test.inclusions='src/**/**/*.spec.js,src/**/**/*.spec.jsx,
     src/**/**/*.test.js,src/**/**/*.test.jsx,src/**/**/*.spec.tsx'
    -Dsonar.javascript.lcov.reportPaths=coverage/lcov.info
    -Dsonar.testExecutionReportPaths=reports/test-reporter.xml
    -Dsonar.dependencyCheck.htmlReportPath=coverage/
        dependency-check-report.html
    -Dsonar.dependencyCheck.jsonReportPath=coverage/
        dependency-check-report.json
```

Código 9: Implementação de *quality gate*

previamente, é integrado os arquivos e *reports* gerados na etapa de testes da *pipeline*, passando os resultados obtidos para o SonarQube interpretá-los e inferir, por exemplo, a porcentagem do código que foi testado para checar se o valor estipulado no *quality gate* foi atendido e um *report* de quais depedências do projetos apresentam vulnerabilidades, explicada na subseção 3.4.4.1.

3.4.6 Etapa 6 - Construção da imagem

Nesta seção será explicada brevemente a ferramenta Docker, utilizada como ferramenta para contêinerização da aplicação, assim como os benefícios de sua utilização. Além disso, é detalhado também como foram construídas as imagens utilizadas na aplicação e a implementação da construção automatizada da imagem na *pipeline*.

3.4.6.1 Ferramenta de contêinerização da aplicação

Para este projeto, optou-se por contêinerizar todos os elementos da aplicação. Existem diversos benefícios ao aplicar essa prática em desenvolvimento de software que utilizam da prática DevOps. Alguns benefícios dessa prática são (KANG; LE; TAO, 2016):

- contêineres são mais leves: por compartilhar do sistema operacional do sistema hospedeiro, contêineres não necessitam de um sistema operacional por aplicação, possibilitando uma maior eficiência na gestão de recursos;
- portabilidade de código: ao utilizar contêineres, é possível empacotar o software e todas suas dependências dentro de uma única instância denominada imagem. Isso possibilita isolar a aplicação do ambiente em que ela será executada, possibilitando que o *deploy* seja realizado em qualquer ambiente;
- simplicidade de configuração: a abordagem de execução de imagens utilizada pelos contêineres possibilita que o software seja executado necessitando de menos pontos de configuração, tornando o processo de instalação, desinstalação e atualização mais simples;
- gerenciamento do ciclo de vida: a utilização de contêineres versionados permite o rastreio de diferentes *releases* em diferentes ambientes, permitindo um simples *rollback* de versão se houver algum problema;
- utilização de recursos: contêineres são mais eficientes do que máquinas virtuais ao acessar a memória e no I/O de dispositivos, e ao mesmo tempo fornece um bom isolamento entre aplicações e entre diferentes contêineres.

Para construção e execução de contêineres, foi utilizada a ferramenta Docker, versão 20.10.0. Docker utiliza uma arquitetura cliente-servidor, no qual o cliente conversa com o Docker *daemon*, que executa tarefas como execução e construção dos contêineres. O cliente e o daemon podem ser executados na mesma máquina ou não. A comunicação entre os dois acontece por meio de uma API REST.

As imagens utilizadas pelo Docker são armazenadas em repositórios denominados *registry*. A Figura 3.3 ilustra o funcionamento dos três componentes da arquitetura.

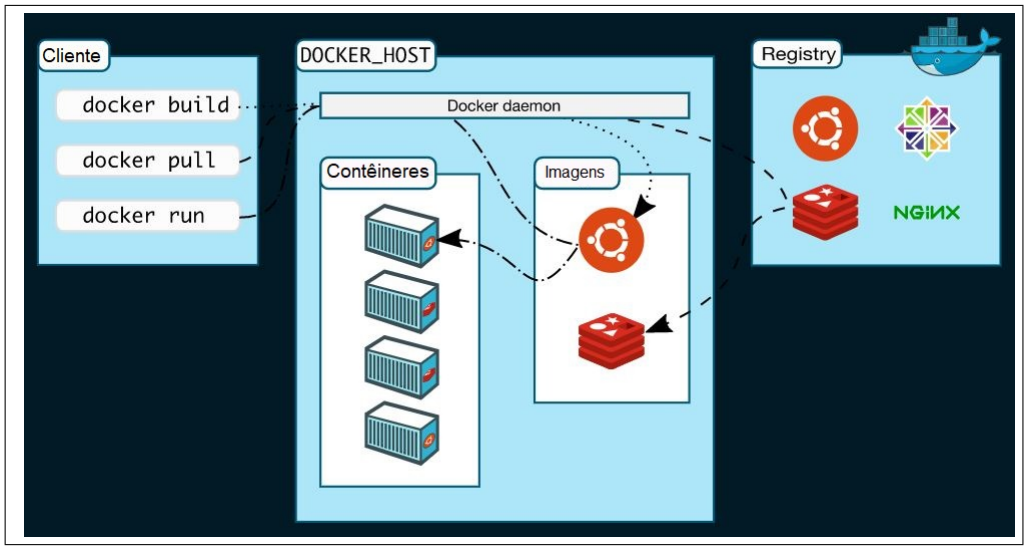


Figura 3.3 – Arquitetura dos componentes Docker. Modificada. Fonte: (DOCKER, 2021a)

3.4.6.2 Construção automatizada da imagem

O *script* utilizado para realizar o *build* da imagem está definido no código 10.

```
1  docker_build:
2      image: docker:latest
3      stage: build_image
4      services:
5          - docker:dind
6      before_script:
7          - docker login -u "$DOCKER_HUB" -p "$DOCKER_HUB_PASS"
8      script:
9          - docker build -t $IMAGE_TAG .
10         - docker save $IMAGE_TAG -o devsecops_image.tar
11     cache:
12       key: "$CI_COMMIT_REF_NAME"
13       paths:
14         - devsecops_image.tar
15       policy: push
```

Código 10: Execução da construção automatizada da imagem

Como descrito na linha 2, neste passo foi utilizada a imagem **docker:latest** para executar os *scripts*. Além disso, o *job* necessita do serviço **docker:dind**, definido nas linhas 4 e 5. A imagem docker:latest contém tudo que é necessário para conectar-se ao Docker Daemon, mas não inicia o Docker Daemon por padrão. Já a imagem **docker:dind** tem como *entrypoint* o comando *["dockerd-entrypoint.sh"]*, que inicia o Daemon. Além disso, ela expõe as portas 2375 e 2376, necessárias para que o cliente conecte-se a ela.

Portanto, os scripts, definidos nas linhas 7, 9 e 10, são executados na imagem **docker:latest** (cliente) e quem executa os processos é a imagem adicionada como serviço

docker:dind.

O comando definido na linha 7 realiza o login no *registry*. Caso não seja especificado, é realizado o login no Docker Hub, que foi utilizado neste projeto. As variáveis **DOCKER_HUB** e **DOCKER_PASS** foram definidas na interface do GitLab e contêm as credenciais necessárias para autenticação no Docker Hub.

Na linha 9, é executado o *build* da imagem. O parâmetro **-t** realiza o *tag* da imagem, que garante informações sobre o nome e a versão da imagem que está sendo construída nesta etapa. Para diferenciar e ter um registro de todas as imagens utilizadas utilizou-se a variável **IMAGE_TAG**, definida como *latdevsecops/frontend:$CI_COMMIT_SHORT_SHA* para a *web* e *latdevsecops/backend:$CI_COMMIT_SHORT_SHA* para o *back-end*.

A palavra *latdevsecops* refere-se ao usuário cadastrado no Docker Hub e *frontend* e *back-end* referem-se aos seus respectivos repositórios no *registry*. Já a variável *$CI_COMMIT_SHORT_SHA* é uma variável predefinida do GitLab, atribuída a uma versão reduzida do *hash* do *commit* registrado no repositório. Dessa forma, a versão da imagem exportada para o Docker Hub está atrelada ao *commit* realizado no GitLab, facilitando o *job* de implantação da próxima etapa, e permitindo o armazenamento de um histórico de imagens e suas respectivas versões.

O *build* da imagem utiliza o Dockerfile definido na raiz do projeto para construir cada camada das suas respectivas imagens. Após realizar o *build* da imagem, salva-se a imagem em um arquivo .tar, para que seja utilizado em *jobs* futuros.

O Dockerfile, utilizado no comando docker build, é um documento em texto que contém comandos que um usuário poderia utilizar na linha de comando para montar uma imagem (DOCKER, 2019a). O Dockerfile da *web* e do *back-end* da aplicação são diferentes. O do *back-end* foi definido como descrito no código 11.

```
1 FROM node:12.18.0
2 WORKDIR /app
3 COPY . .
4 EXPOSE 3333
5 RUN yarn && yarn build
6 ENTRYPOINT yarn sequelize db:migrate && yarn start
```

Código 11: Dockerfile *back-end*

O comando **FROM** define a imagem base na qual será construída a imagem final. Neste caso, foi utilizada a imagem **node:12.18.0**, que contêm a versão do NodeJS e do Yarn que foram utilizadas no projeto.

Na linha 2, executa-se o comando WORKDIR /app, definindo o diretório de trabalho, seguido do comando COPY . ., que copia todos os arquivos que estão na raiz do projeto para o diretório /app dentro do contêiner.

Na linha 4, é exposta a porta 3333, utilizada como canal de comunicação pela aplicação.

Na linha 5, é executado o comando **RUN**, executando o comando yarn, para realizar o *download* das dependências e o comando **yarn build**, que prepara a aplicação para produção, compilando o código e criando um diretório para os arquivos recém criados. O comando **RUN** é executado como etapa na construção da imagem, já o comando **ENTRYPOINT**, na linha 6, é o comando executado quando o contêiner é executado. Neste caso, o comando **yarn sequelize db:migrate** realiza a criação das tabelas necessárias no banco de dados e **yarn start** inicia a aplicação.

Já o Dockerfile utilizado para a *web* está descrito no código 12.

```
1  FROM node:12.18.0 as build_stage
2  WORKDIR /app
3  COPY . .
4  RUN yarn && yarn build
5
6  FROM nginx:stable-alpine
7  COPY --from=build_stage /app/build /usr/share/nginx/html
8  COPY --from=build_stage /app/nginx/nginx.conf /etc/nginx/
9      conf.d/default.conf
10 EXPOSE 80
11 CMD ["nginx", "-g", "daemon off;"]
```

Código 12: Dockerfile web

Neste caso, foi utilizado uma ferramenta disponibilizada pelo Docker, denominada *multi-stage builds*, que possibilita a utilização de múltiplas imagens base, cada uma delas iniciando um novo estágio da *build*. É possível copiar artefatos de um estágio para o outro (DOCKER, 2019b). Com isso, é possível utilizar uma imagem **Node** apenas para instalar o que for necessário, deixando a aplicação pronta para produção, e, feito isso, o Nginx utiliza estes arquivos para expôr a aplicação.

As linhas 1 a 4 são similares ao executado na imagem do *back-end*, copiando o código para dentro do contêiner, realizando o *download* das dependências e o *build* da aplicação. Já as linhas de 6 a 11, utilizam o Nginx para iniciar a aplicação. Para isso, utiliza-se a imagem nginx:stable-alpine como base e nas linhas 7 e 8 é realizado a cópia dos artefatos gerados pelo estágio anterior, copiando o conteúdo gerado pela *build* para o diretório */usr/share/nginx/html*, que o Nginx utiliza como padrão para guardar o site estático, e em seguida copiando a configuração do Nginx para dentro do contêiner.

Por fim, nas duas últimas linhas, a porta 80 do NGINX é exposta e, ao executar o contêiner, o comando *nginx -g daemon off;* é executado, iniciando a instância NGINX. O arquivo nginx.conf, utilizado nesta etapa, encontra-se no anexo D.

3.4.7 Etapa 7 - Varredura da imagem Docker

Nesta etapa foi utilizada a ferramenta **Trivy** (AQUASECURITY, 2021), versão 0.18.1, que proporciona uma varredura de vulnerabilidades encontradas nos pacotes e dependências do sistema base utilizado pela imagem. Além disso, essa ferramenta proporciona uma solução pensada para utilização em *pipelines* CI/CD.

O Trivy realiza o download do banco de dados de CVEs externo e analisa os pacotes instalados na imagem Docker. Caso encontre algum pacote que esteja presente na lista de CVEs, um alerta é gerado.

O script utilizado para realizar a verificação está definido no código 13.

```
1   image_scan:
2       stage: image_scan
3       image:
4           name: aquasec/trivy:latest
5           entrypoint: [""]
6       script:
7           - trivy --exit-code 0 --no-progress --severity LOW
8               --output scanning-report.txt -i devsecops_image.tar
9           - trivy --exit-code 1 --no-progress --severity CRITICAL
10              --output scanning-report.txt -i devsecops_image.tar
11          - trivy --exit-code 1 --no-progress --severity HIGH
12              --output scanning-report.txt -i devsecops_image.tar
13          - trivy --exit-code 1 --no-progress --severity MEDIUM
14              --output scanning-report.txt -i devsecops_image.tar
15
16      cache:
17        key: "$CI_COMMIT_REF_NAME"
18        paths:
19          - devsecops_image.tar
20        policy: pull
21      artifacts:
22          when: always
23          paths:
24              - scanning-report.txt
```

Código 13: Execução da varredura da imagem Docker

Para execução desta etapa, foi utilizado o contêiner disponibilizado pelo próprio Trivy, **aquasec/trivy:latest**, contendo o código a ser executado. Por padrão, o contêiner roda o Trivy assim que é executado. Para mudar esse padrão, na linha 5, sobrescreveu-se o *entrypoint* padrão do contêiner, antes definido como ["trivy"] e agora como vazio.

O Trivy possui uma ferramenta muito útil para criação de uma pipeline DevSecOps, que é a definição do código de saída do programa baseado na severidade do CVE encontrado. Ou seja, cabe a quem estiver implementando a pipeline definir o padrão de

qualidade esperado no projeto. Neste projeto, foi definido como padrão interromper a *pipeline* se a vulnerabilidade for de severidade crítica, alta ou média e apenas reportar se houver alguma vulnerabilidade de baixa severidade. Todas essas vulnerabilidades são reportadas para um relatório armazenado em um arquivo texto para que possa ser analisado pelo desenvolvedor.

3.4.8 Etapa 8 - Envio da imagem para o repositório

Com a finalização dos testes na imagem e garantindo a ausência de vulnerabilidades, a imagem está pronta para ser enviada para o *registry*. Para isso, criou-se *script* definido no código 14.

```
1     docker_push:
2       image: docker:latest
3       stage: push
4       services:
5           - docker:dind
6       before_script:
7           - docker load -i devsecops_image.tar
8           - docker login -u "$DOCKER_HUB" -p "$DOCKER_HUB_PASS"
9       script:
10          - docker push $IMAGE_TAG
11      cache:
12        key: "$CI_COMMIT_REF_NAME"
13        paths:
14          - devsecops_image.tar
15        policy: pull
```

Código 14: Envio da imagem para o repositório

Utilizou-se novamente a mesma ideia apresentada em 3.4.6.2, utilizando uma imagem Docker como base e um serviço auxiliar **docker:dind**. Feito isso, carregou-se a imagem criada na etapa de construção da imagem na linha 7 e realizou-se o login no *registry*, utilizando as credenciais de login e senha, que foram atribuídas às variáveis **DOCKER_HUB** e **DOCKER_PASS**, que estão definidas no projeto do GitLab CI. Feito isso, o *push* para o *registry* é realizado, na linha 10.

3.4.9 Etapa 9 - Testes dinâmicos de segurança

Nesta etapa, foi utilizada a ferramenta OWASP Zed Attack Proxy (ZAP), versão 2.10.0, para realizar os testes dinâmicos de segurança. O ZAP é uma ferramenta de varredura de aplicações *open source* disponibilizada pela OWASP (OWASP, 2021c). Essa ferramenta disponibiliza uma solução pensada para ser utilizada em soluções de CI/CD,

em que é possível realizar varreduras por meio de linhas de comando, utilizando o contêiner **owasp/zap2docker**.

As varreduras disponibilizadas pelo contêiner do OWASP dividem-se em varredura passiva e ativa. A varredura passiva analisa as requisições e respostas HTTP enviadas para a aplicação que está sendo testada, sem modificar o conteúdo dos pacotes. Já a verificação ativa consiste em realizar ataques comuns à aplicação nos alvos selecionados (OWASP, 2021b) (OWASP, 2021d).

As varreduras disponibilizadas por essa ferramenta são (OWASP, 2021f):

- *Baseline scan*: executa uma varredura passiva da aplicação por um minuto, seguido por uma varredura *spider*, que é uma ferramenta utilizada para descobrir recursos da aplicação, como URLs, por exemplo. O *spider* visita as URLs e realiza um *scan* passivo dos recursos (OWASP, 2021e);
- *API Scan*: utiliza de uma definição de API, gerada pelo OpenAPI ou GraphQL, para realizar uma varredura ativa.

Neste projeto, foram realizados o *API scan* no *back-end* e o *baseline scan* na *web*. Para o *back-end*, utilizou-se a definição da API gerada pelo padrão OpenAPI, que define um padrão independente de linguagem, para definição de APIs REST (OPENAPI, 2021). A definição encontra-se no anexo E.

O *script* utilizado para executar os testes dinâmicos foi criado como ilustrado no código 15.

Esse passo utiliza a imagem do OWASP para realizar os testes dinâmicos na aplicação. Para isso, define os serviços auxiliares e a própria aplicação como serviço, nas linhas 4 a 10, e as variáveis necessárias para iniciá-los, nas linhas 11 a 13. Nas linhas 15 e 16, prepara-se e executa-se o programa **get_token.py**. Esse *script* (Anexo F) tem como finalidade realizar requisições para a API, criando usuários e sessões na aplicação, para que a varredura utilize o *token* para fazer as análises de segurança nas URLs que necessitam de autenticação.

As linhas 18 e 19 definem a execução da varredura. O parâmetro -t aponta para o arquivo JSON que contém a definição da API, o parâmetro -z aponta para o arquivo de configuração **options.prop**. Esse arquivo define redefine valores no cabeçalho dos pacotes HTTP que são enviados para aplicação. O *script* get_token.py popula este arquivo com o valor do *token* de sessão que acabou de ser criado. Com isso, ao realizar requisições para URLs que necessitem de autenticação, o campo "Authorization"será populado com o *token* de sessão. O arquivo *options.prop* foi definido como ilustrado no código 16.

```yaml
owasp_test:
  stage: dynamic_tests
  image: owasp/zap2docker-stable
  services:
    - mongo
    - postgres
    - name: redis:alpine
      alias: redis
    - name: latdevsecops/backend:$IMAGE_TAG
      alias: app
  variables:
    POSTGRES_DB: $POSTGRES_DB
    POSTGRES_PASSWORD: $POSTGRES_PASSWORD
  before_script:
    - pip install faker
    - python get_token.py
  script:
    - zap-api-scan.py -t AgendaENE.json -f openapi -z
        "-configfile options.prop" -r report.html
  artifacts:
    when: always
    paths:
        - report.html
```

Código 15: Execução dos testes dinâmicos

```
    'replacer.full_list(0).description=auth1',
    'replacer.full_list(0).enabled=true',
    'replacer.full_list(0).matchtype=REQ_HEADER',
    'replacer.full_list(0).matchstr=Authorization',
    'replacer.full_list(0).regex=false',
    'replacer.full_list(0).replacement=TOKEN'
```

Código 16: Arquivo options.prop

A varredura para a *web* foi realizada de forma semelhante, adicionando o serviço do *front-end* ao campo *services*, e realizando o *baseline scan*.

3.4.10 Etapa 10 - Implantação

Nesta seção pretende-se detalhar as ferramentas Kubernetes, utilizada como orquestrador de contêineres, Rancher, gerenciador do cluster Kubernetes e Helm, gerenciador de pacotes Kubernetes. Além disso, detalha-se também informações sobre a infraestrutura provisionada para execução da aplicação e como foi implementada a entrega contínua.

3.4.10.1 Orquestração de contêineres

Ferramentas de orquestração de contêineres podem ser definidas como plataformas para integração e gerenciamento de contêineres em escala. Essas plataformas simplificam o gerenciamento de contêineres fornecendo maneiras de gerenciar múltiplos contêineres, para propósito de disponibilidade, escalabilidade e rede (KHAN, 2017).

A estrutura de ferramentas de orquestração de contêineres consiste de três elementos: gerenciamento de recursos, agendamento e gerenciamento de serviços. A camada de gerenciamento de recursos tem como objetivo maximizar a utilização e minimizar a interferência entre contêineres que competem por recursos como CPU, memória e espaço em disco. A camada de agendamento tem como finalidade a utilização eficiente de recursos, com capacidades como: replicação e escalonamento, reagendamento para reiniciar contêineres automaticamente e *rolling deployment* para realizar *upgrades* e *downgrades* de versão automaticamente. Por último, a camada de gerenciamento de serviço fornece capacidades adicionais para realizar implantações de aplicações complexas, como grupos e *namespaces* para criar domínios diferentes de contêineres, dependências para definir vínculos entre diferentes contêineres, balancemento de carga, entre outros (JAWARNEH et al., 2019).

A Figura 3.4 mostra como as três camadas estão dispostas considerando a infraestrutura base.

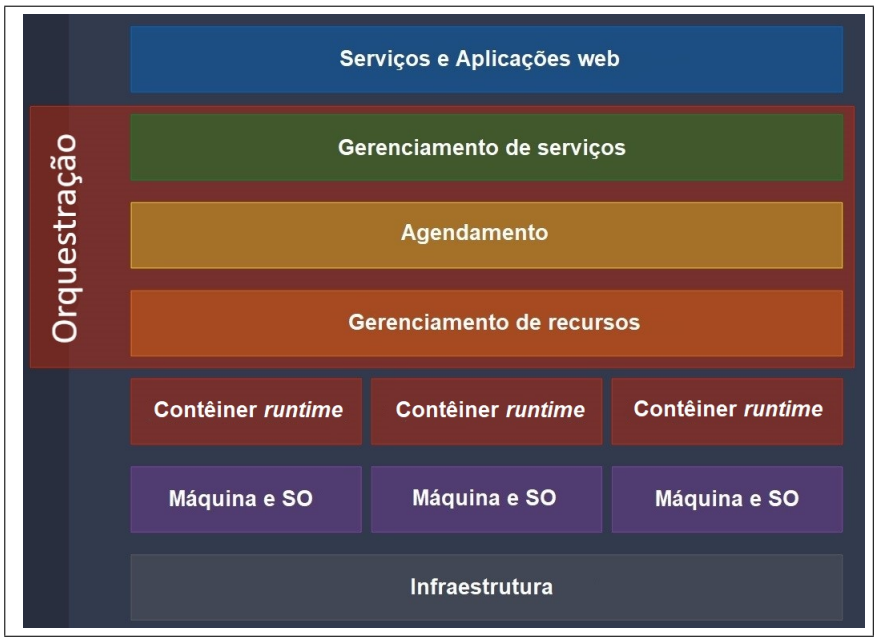


Figura 3.4 – Orquestração de contêineres. Modificada. Fonte: (DEVOPEDIA, 2021)

Neste trabalho foi utilizada a plataforma de código aberto Kubernetes para este fim.

Um *cluster* Kubernetes é constituído de nós, que executam as aplicações contêine-

rizadas, que são divididos em *worker* e *control plane*. O *worker* hospeda os contêineres que compõem a carga de trabalho da aplicação, no Kubernetes, denominados *pods*, enquanto o *control plane* gerencia os nós *worker* e *pods* do *cluster* (KUBERNETES, 2021b). Neste trabalho, foi utilizado um *cluster* simples, contendo um *control plane* e um *worker*.

O recurso pelo qual a aplicação que está sendo executada no *cluster* Kubernetes é exposta é denominado *Service*, que define um conjunto lógico de *pods* e políticas para acessá-los (KUBERNETES, 2021d).

Já a carga de trabalho da aplicação é definida por meio de declarações denominadas *deployments*. Neles são definidos o estado desejado da aplicação e o Kubernetes mudará o estado atual para o estado desejado em uma taxa controlada.

Os serviços e *deployments* são definidos por meio de arquivos YAML, e foram definidos neste projeto utilizando o Helm.

3.4.10.2 Gerenciamento do Cluster Kubernetes

O gerenciamento de cluster Kubernetes complexos pode se tornar um grande desafio, já que diversos arquivos de configuração serão criados. A ferramenta Rancher é uma plataforma de código aberto que atua no gerenciamento de clusters Kubernetes. O Rancher é um software desenvolvido para equipes que adotam contêineres na sua infraestrutura que aborda desafios operacionais e de segurança no gerenciamento de *clusters* Kubernetes em suas máquinas (RANCHER, 2021b).

Dentre as capacidades do Rancher que auxiliam o gerenciamento do *cluster* estão (RANCHER, 2021a):

- *deploy* e monitoração de *clusters* em qualquer infraestrutura: o Rancher realiza o *deploy* do Kubernetes e configura todos os seus componentes automaticamente. Além disso, monitora a saúde do cluster nativamente;
- interface gráfica para gerenciamento de carga de trabalho: a ferramenta provê uma forma simples de realizar o *deploy* de serviços utilizando o Kubernetes e ter visibilidade completa do cluster. É possível definir regras, segredos, *ingress controllers*, e outras configurações por meio da interface gráfica;
- catálogos privados e globais: por meio do Helm, o Rancher disponibiliza catálogos de diferentes serviços que podem ser executados rapidamente.

Neste projeto, foi utilizada uma distribuição do Kubernetes disponibilizada pelo Rancher, denominada RKE (Rancher Kubernetes Engine), que cria um *cluster* Kubernetes por meio de contêineres Docker.

3.4.10.3 Gerenciamento de pacotes Kubernetes

Com integração nativa com o Rancher, o Helm é uma ferramenta de gerenciamento de pacotes para o Kubernetes. Com ele é possível descrever a estrutura da aplicação utilizando catálogos Helm, denominadas *helm charts*, que contém todas as definições necessárias para rodar uma aplicação em um *cluster* Kubernetes, e gerenciá-las utilizando comandos simples.

A utilização do Helm traz algumas vantagens como (HELM, 2021):

- gerenciamento de complexidade: é possível descrever qualquer aplicação utilizando catálogos, o que torna o processo de instalação simples e repetível;
- simples de atualizar e de realizar *rollbacks*: o Helm torna o versionamento da aplicação mais simples, tornando o processo de atualização e *rollback* simples, com a execução de um comando;
- simples de compartilhar: é possível compartilhar seus próprios catálogos e utilizar catálogos pré existentes.

O Helm cria a seguinte estrutura de diretórios:

```
chart/
    Chart.yaml
    values.yaml
    charts/
    templates/
```

O diretório *templates* contém os templates dos serviços, *deployments*, *ingress* e outros componentes do cluster Kubernetes.

O arquivo values.yaml contém os valores padrão para o catálogo, que podem ser sobrescritos utilizando o comando **helm upgrade**.

O arquivo Chart.yaml contém uma descrição da *chart*, e o subdiretório charts pode conter *subcharts*.

No diretório templates, foram definidas as *charts* necessárias para realizar o *deploy* da aplicação, contendo os *deployments* e serviços Kubernetes. Nestes catálogos, estão referenciadas as variáveis definidas no arquivo values.yaml.

Os catálogos utilizados para implantação da aplicação estão descritos no anexo G.

3.4.10.4 Infraestrutura de provisionamento

A infraestrutura utilizada para execução da aplicação foi providenciada por meio da alocação de máquinas virtuais no provedor de nuvem Google Cloud Platform (GCP) (GOOGLE, 2021). As máquinas virtuais foram alocadas com os seguintes recursos:

- 2 vCPUs, 4GB de memória - utilizada para execução do Rancher Server - versão 2.5.6;
- 4 vCPUs, 16GB de memória - utilizada para execução do cluster Kubernetes - versão 1.20.5.

Os passos utilizados para instalação do Rancher Server e do *cluster* Kubernetes encontram-se nos anexos H e I, respectivamente.

Para tornar os serviços acessíveis, utilizou-se o serviço de DNS da Google, criando um registro do tipo A para acessar a interface do Rancher server, apontando para o IP público do servidor (VM) e um outro registro *wildmask*, feito para que toda requisição com um subdomínio seja gerenciada pelo NGINX *ingress controller*, que faz o redirecionamento dos registros que contém um cabeçalho HTTP válido para os respectivos serviços, que irão redirecionar as requisições aos *pods* da aplicação, conforme exemplificado pela Figura 3.5.

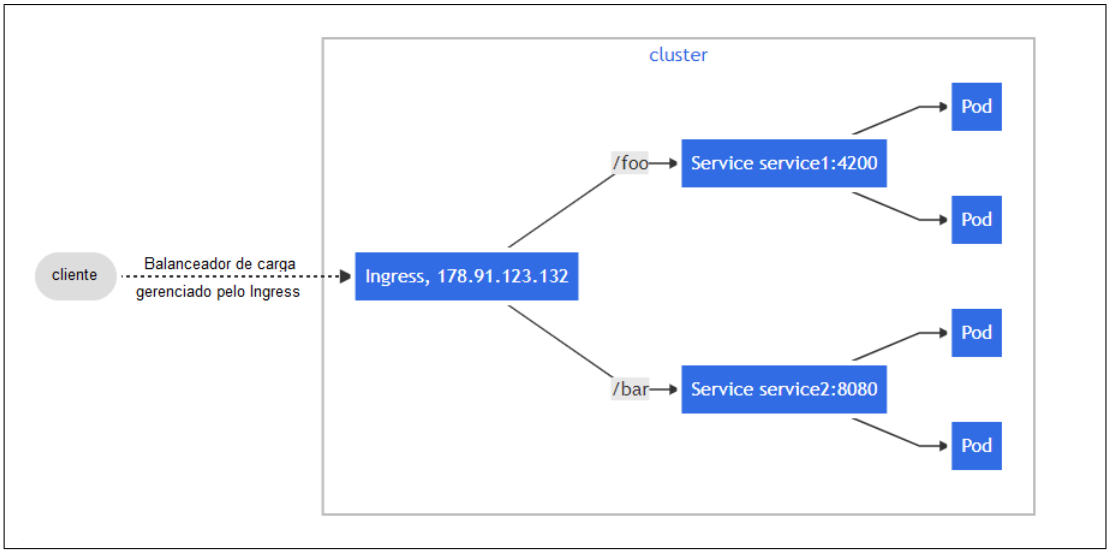


Figura 3.5 – Esquemático de funcionamento do *ingress controller*. Modificada. Fonte: (KUBERNETES, 2021c)

Para navegação segura utilizando o protocolo HTTPS, foi utilizado um certificado TLS gerado pela Let's Encrypt (ENCRYPT, 2021b). A Let's encrypt é uma Autoridade Certificadora gratuita, automatizada e *open source*. O objetivo da Let's Encrypt é tornar possível a configuração de um servidor HTTPS e fazê-lo obter automaticamente um certificado confiável sem intervenção humana. Isso é realizado através do uso do agente de

gerenciamento de certificado no servidor web. proporcionando a utilização do protocolo HTTPS (ENCRYPT, 2021a). Para geração desses certificados, foi utilizado o CertBot, que é um software *open source* para gerar certificados TLS utilizando a Let's Encrypt como Autoridade Certificadora. O CertBot renova seus certificados a cada 60 dias (CERTBOT, 2021).

3.4.10.5 Automação da entrega contínua

Para automatizar o processo de implantação no cluster Kubernetes, utilizou-se o script definido no código 17.

```
1     deploy:
2         stage: deploy
3         image: dtzar/helm-kubectl
4         script:
5             - echo $KUBECONFIG | base64 -d > ./KUBECONFIG
6             - helm --kubeconfig=/builds/devsecops/agendaene/KUBECONFIG
7                 upgrade -n agendaene --install agendaene .
8                 --set=agendaene.tag=$CI_COMMIT_SHORT_SHA
```

Código 17: Automação da entrega contínua

Como descrito na linha 3, foi utilizada uma imagem que contém o Helm instalado para realizar a implantação, descrita nas linhas 5 a 8. A linha 5 decodifica o arquivo em base 64 denominado kubeconfig.yaml, que está definido como variável no GitLab. Este arquivo contém todas as informações necessárias para que um serviço se comunique com a API do *cluster* Kubernetes. O comando **helm upgrade** realiza a atualização da nova *release* para uma nova versão do catálogo. O parâmetro –kubeconfig utiliza o recém decodificado arquivo kubeconfig para realizar a comunicação com o *cluster*. Para garantir que a aplicação seja instalada caso não tenha sido instalada ainda, utiliza-se o parâmetro –install. Como é possível executar diferentes *namespaces* em um mesmo *cluster*, o parâmetro -n define o *namespace* em que a aplicação será executada. Já o parâmetro –set substitui a *tag* da aplicação, definida no arquivo values.yaml pela *tag* que corresponde ao *hash* do *commit*, atualizando o cluster com a imagem que foi construída na etapa 6.

3.4.11 *Pipeline* de geração do executável móvel

A frente mobile utiliza de tecnologias próprias para esta arquitetura, que não compõem o escopo do presente estudo. Dessa forma, foi desenvolvida uma *pipeline* distinta das demais frentes, focando somente na construção do arquivo executável para instalação em dispositivos que utilizem o sistema operacional *Android*.

Para execução desta etapa executou-se passos definidos no código 18.

A linha 1 define a imagem utilizada neste *job*, contendo todo o ambiente necessário para execução dos comandos, com destaque na implantação de todas as dependências em Java necessárias para execução dos comandos. Nas linhas 6, 7 e 8 estão descritas as variáveis globais de sistema a serem utilizadas nos comandos, sendo definições de versionamento de ferramentas necessárias para criação do arquivo *Android Package - APK*. Tais variáveis são interpretadas e atribuídas devido a imagem já estar preparada para interpretá-las. As linhas 10 e 11 definem as duas etapas descritas no *job*.

Da linha 12 a 16 define-se o estágio do download das dependências do projeto, como descrito em 3.4. As linhas 20 e 21 executam uma verificação e instalação dos pacotes próprios da imagem utilizada. Da linha 22 a 23 são executados comandos responsáveis por instalar e atualizar a versão do NodeJS na imagem, definido para atualizar e rodar na versão 12.16.2, e, por fim, utilizando do comando descrito na linha 25 para verificar se a versão instalada é a esperada.

A linha 26 realiza o *download* de pacotes a serem utilizados na máquina, como o *wget* para realizar requisições HTTP, possibilitando assim que sejam feitos *download* a arquivos na internet e o *unzip*, responsável por extrair os dados de arquivos compactados. Agora tendo a disponibilidade de tais pacotes, nas linhas 27, 28 e 29 é realizado o *download* do arquivo *Software Development Kit - SDK* do *android* na versão definida na linha 8, seguido pelo comando da linha 30, responsável por criar o diretório "android-sdk-linux"e extrair os dados do arquivo baixado para tal diretório. Este arquivo SDK é responsável por fazer a tradução do JavaScript implementado no projeto para a linguagem Java, nativa dos sistemas *android*.

Já nas linhas 31 e 32 são definidos as variáveis globais necessárias para utilização de comandos próprios do *android*. Como exposto, foi configurado as variáveis "ANDROID_HOME"e "PATH"para serem interpretadas seguindo o diretório "android-sdk-linux", criado no passo anterior, contendo os arquivos necessários para a correta execução das variáveis pelo sistema.

Com a imagem agora preparada para executar comandos nativos do *android*, o comando da linha 36 altera o diretório atual para a pasta *android* do projeto, pasta essa gerada pela biblioteca React Native. Já dentro do diretório é executado o comando "chmod +x ./gradlew", como mostrado na linha 37, que garante que o comando "gradlew", responsável pela construção do arquivo APK - linhas 38 e 39 - seja executável no sistema. Por fim, as linhas 40, 41 e 42 apontam para o diretório onde o arquivo foi gerado, tornando-o disponível para *download* ao final da execução da *pipeline*.

```
image: openjdk:8-jdk
cache:
paths:
    - node_modules/
variables:
ANDROID_COMPILE_SDK: "28"
ANDROID_BUILD_TOOLS: "28.0.2"
ANDROID_SDK_TOOLS: "4333796"
stages:
    - install_dependencies
    - build
run_dependencies:
     image: node
     stage: install_dependencies
     script:
        -  yarn
assembleDebug:
    stage: build
    script:
      -  apt-get --quiet update --yes
      -  apt-get --quiet install --yes npm
      -  npm cache clean -f
      -  npm install -g n
      -  n 12.16.2
      -  node -v
      -  apt-get --quiet install --yes wget tar unzip lib32stdc++6 lib32z1
      -  wget --quiet --output-document=android-sdk.zip
         https://dl.google.com/android/repository
         /sdk-tools-linux-${ANDROID_SDK_TOOLS}.zip
      -  unzip -d android-sdk-linux android-sdk.zip
      -  export ANDROID_HOME=$PWD/android-sdk-linux
      -  export PATH=$PATH:$PWD/android-sdk-linux/platform-tools/
      -  set +o pipefail
      -  yes | android-sdk-linux/tools/bin/sdkmanager --licenses
      -  set -o pipefail
      -  cd android
      -  chmod +x ./gradlew
      -  ./gradlew clean
      -  ./gradlew assembleRelease
artifacts:
    paths:
        - ./android/app/build/outputs/apk/release
```

Código 18: Geração do APK *mobile*

4 Resultados e Discussões

Neste capítulo, serão evidenciados os resultados obtidos em cada passo da *pipeline*, seguido por uma breve análise dos relatórios e registros de cada estágio no GitLab. Ao final do capítulo, será possível entender um pouco mais sobre as verificações realizadas e as vulnerabilidades descobertas.

4.1 Etapa 1 - Construção

A figura 4.1 evidencia o sucesso desta etapa, realizando o download das dependências da aplicação e fazendo o upload desses arquivos para o próximo estágio utilizá-lo.

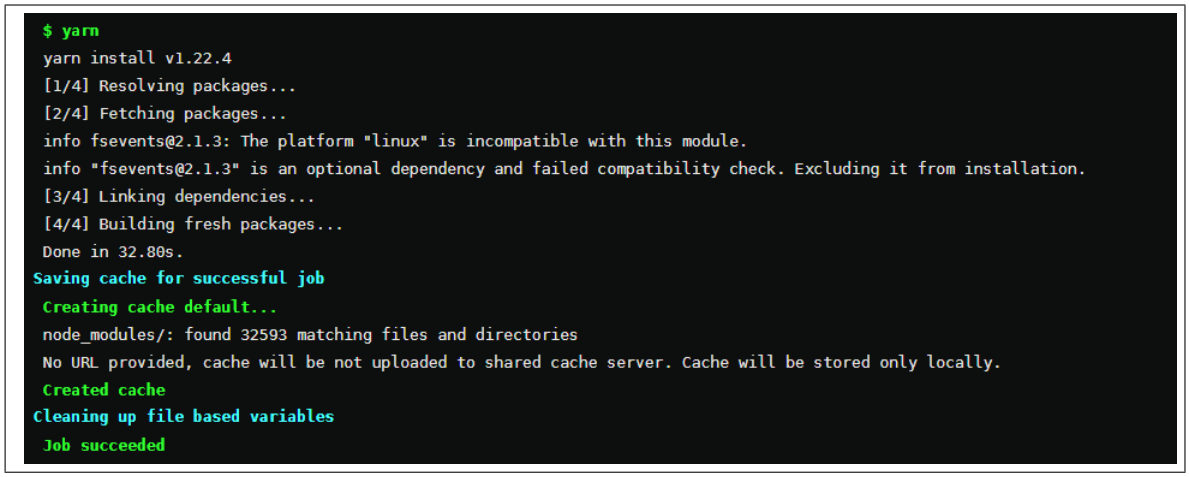


Figura 4.1 – Logs de execução da etapa de construção

4.2 Etapa 2 - Controle de qualidade

A figura 4.2 evidencia a execução desta etapa, realizando o download das dependências da etapa anterior, realizando o teste de *lint* e realizando o upload do relatório da verificação.

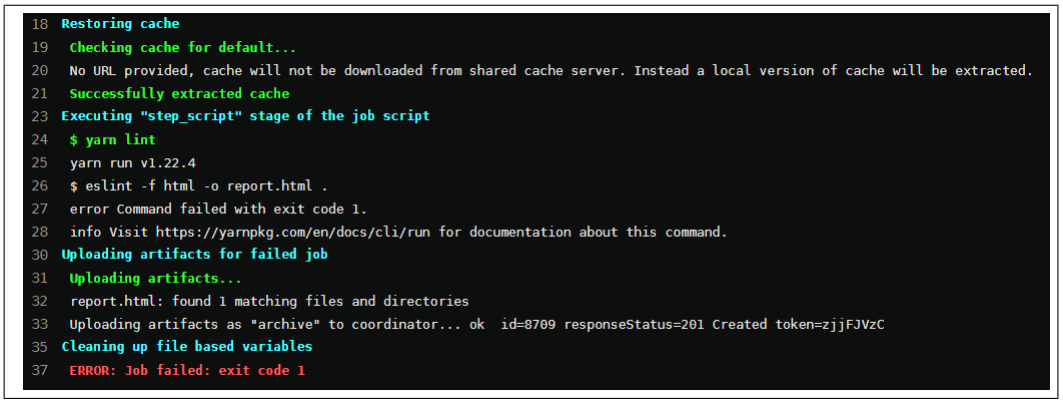


Figura 4.2 – Logs de execução da etapa controle de qualidade

A falha desta etapa ocorre conforme esperado, já que foram detectadas erros nesta etapa, conforme figura 4.3

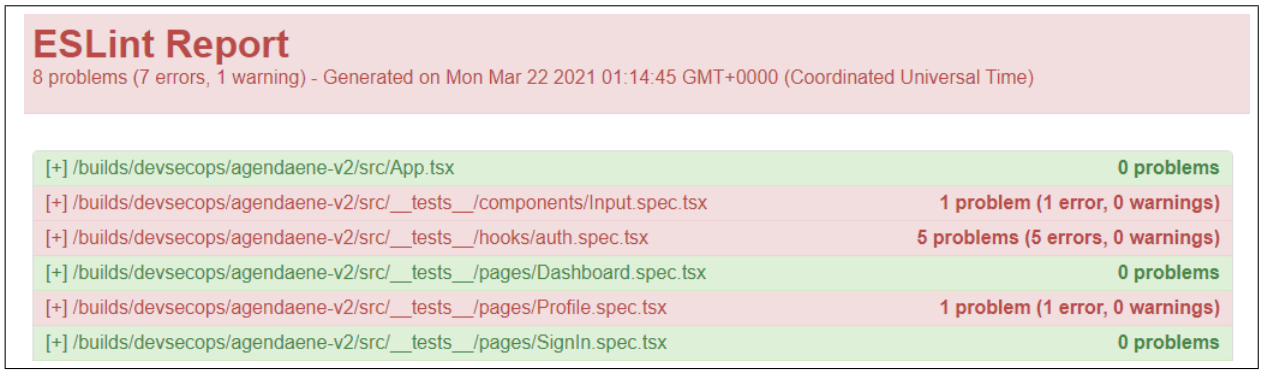
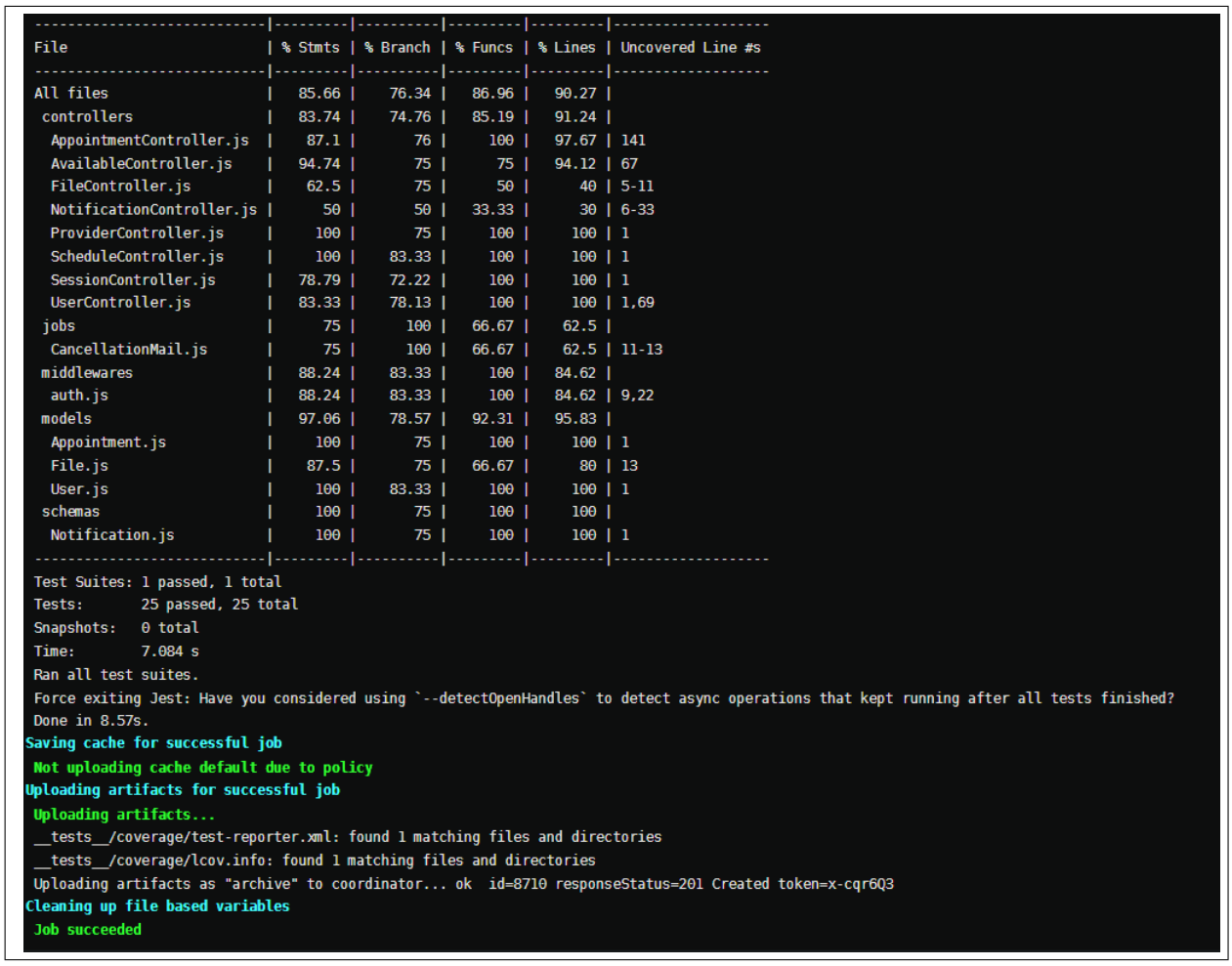


Figura 4.3 – Relatório de erros da etapa de lint

4.3 Etapa 3 - Testes unitários e de integração

A figura 4.4 ilustra a o sucesso desta etapa, realizando os testes de integração da aplicação e o *upload* dos artefatos.



```
----------------------------|---------|----------|---------|---------|-------------------
File                        | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------------------------|---------|----------|---------|---------|-------------------
All files                   |   85.66 |    76.34 |   86.96 |   90.27 |
 controllers                |   83.74 |    74.76 |   85.19 |   91.24 |
  AppointmentController.js  |    87.1 |       76 |     100 |   97.67 | 141
  AvailableController.js    |   94.74 |       75 |      75 |   94.12 | 67
  FileController.js         |    62.5 |       75 |      50 |      40 | 5-11
  NotificationController.js |      50 |       50 |   33.33 |      30 | 6-33
  ProviderController.js     |     100 |       75 |     100 |     100 | 1
  ScheduleController.js     |     100 |    83.33 |     100 |     100 | 1
  SessionController.js      |   78.79 |    72.22 |     100 |     100 | 1
  UserController.js         |   83.33 |    78.13 |     100 |     100 | 1,69
 jobs                       |      75 |      100 |   66.67 |    62.5 |
  CancellationMail.js       |      75 |      100 |   66.67 |    62.5 | 11-13
 middlewares                |   88.24 |    83.33 |     100 |   84.62 |
  auth.js                   |   88.24 |    83.33 |     100 |   84.62 | 9,22
 models                     |   97.06 |    78.57 |   92.31 |   95.83 |
  Appointment.js            |     100 |       75 |     100 |     100 | 1
  File.js                   |    87.5 |       75 |   66.67 |      80 | 13
  User.js                   |     100 |    83.33 |     100 |     100 | 1
 schemas                    |     100 |       75 |     100 |     100 |
  Notification.js           |     100 |       75 |     100 |     100 | 1
----------------------------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       25 passed, 25 total
Snapshots:   0 total
Time:        7.084 s
Ran all test suites.
Force exiting Jest: Have you considered using `--detectOpenHandles` to detect async operations that kept running after all tests finished?
Done in 8.57s.
Saving cache for successful job
 Not uploading cache default due to policy
Uploading artifacts for successful job
 Uploading artifacts...
 __tests__/coverage/test-reporter.xml: found 1 matching files and directories
 __tests__/coverage/lcov.info: found 1 matching files and directories
 Uploading artifacts as "archive" to coordinator... ok  id=8710 responseStatus=201 Created token=x-cqr6Q3
Cleaning up file based variables
 Job succeeded
```

Figura 4.4 – Logs de execução da etapa de testes de integração para o *back-end*

Já a figura 4.5 ilustra a etapa de testes unitários e o *upload* dos artefatos.

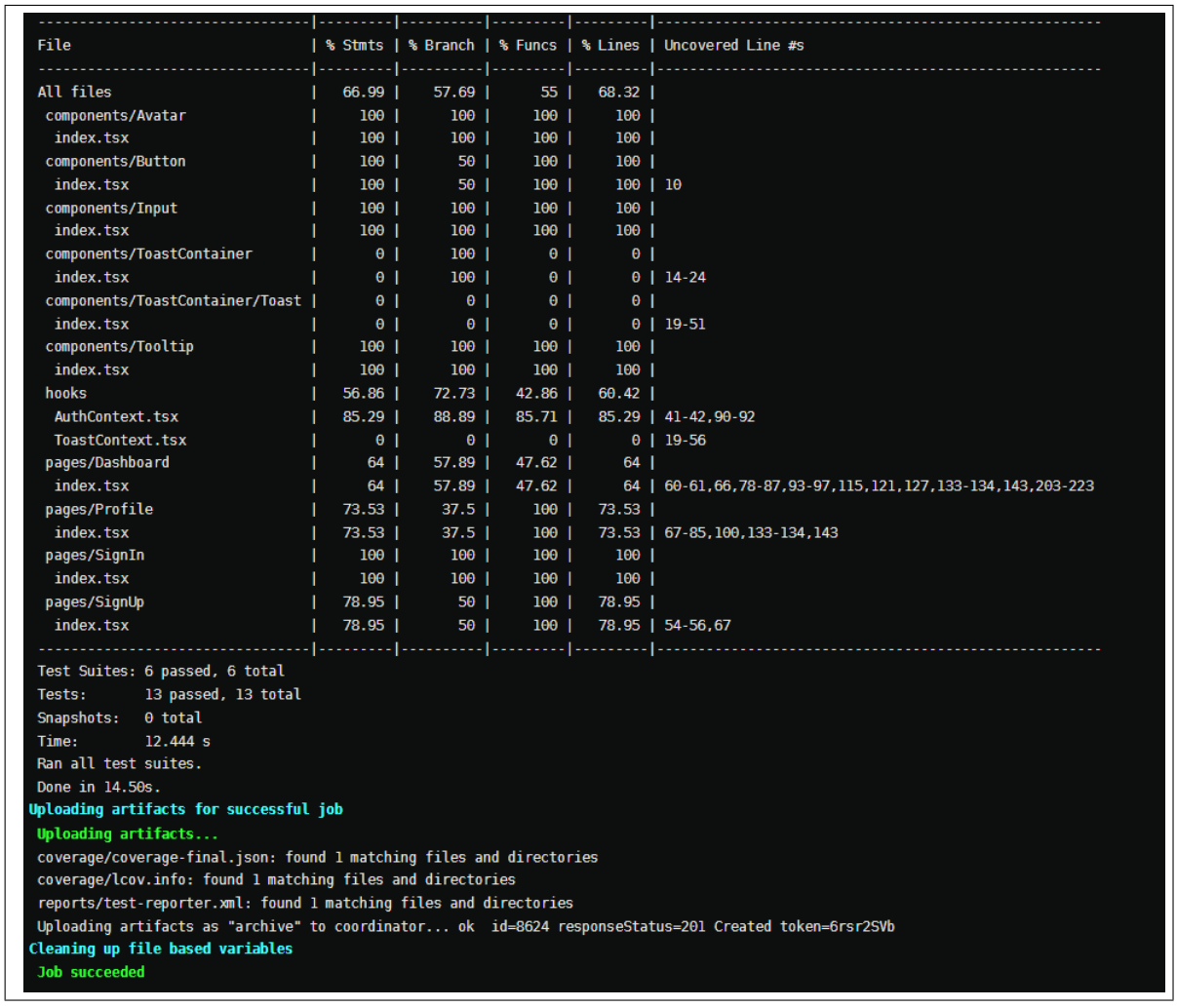


Figura 4.5 – Logs de execução da etapa de testes unitários para a *web*

4.4 Etapa 4 - Testes estáticos

Nesta etapa pretende-se analisar os resultados obtidos na etapa de testes estáticos, dividido em duas frentes: verificação de dependências e verificação estática de segurança.

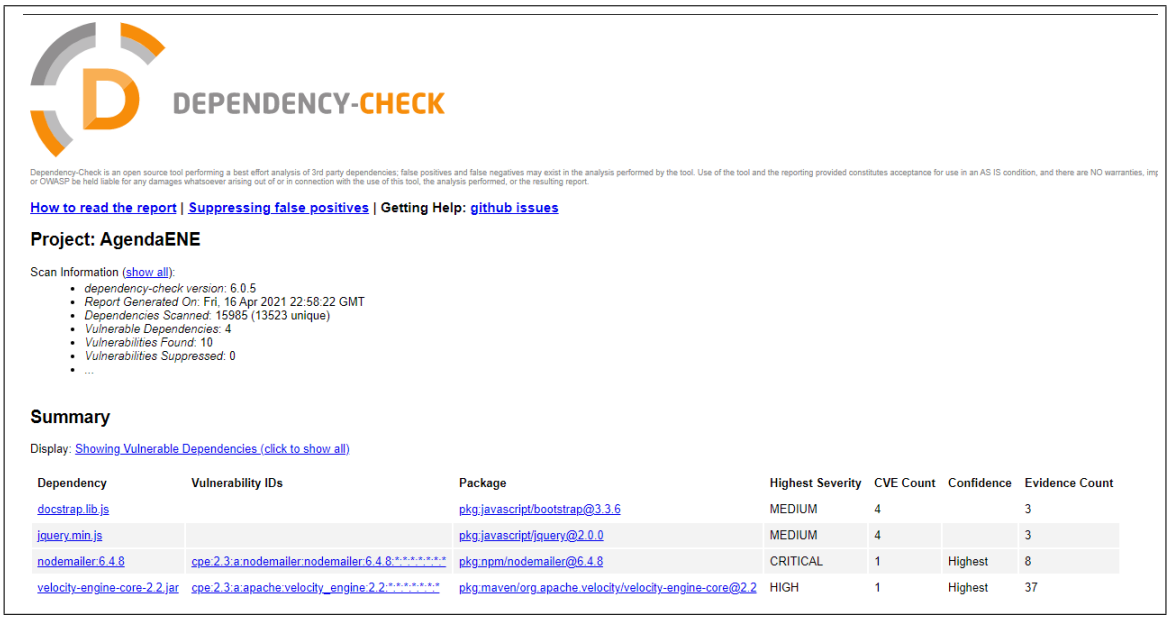
4.4.1 Verificação de dependências

A figura 4.6 ilustra a finalização desta etapa, finalizando os testes de vulnerabilidade de dependências e fazendo o *upload* dos relatórios.

A figura 4.7 ilustra as vulnerabilidades encontradas nas dependências utilizadas no projeto.

```
[INFO] Writing report to: /builds/devsecops/agendaene-backend-tests/__tests__/coverage/dependency-check-report.json
[INFO] Writing report to: /builds/devsecops/agendaene-backend-tests/__tests__/coverage/dependency-check-report.html
Saving cache for successful job
Creating cache default...
node_modules/: found 32593 matching files and directories
No URL provided, cache will be not uploaded to shared cache server. Cache will be stored only locally.
Created cache
Uploading artifacts for successful job
Uploading artifacts...
__tests__/coverage/dependency-check-report.html: found 1 matching files and directories
__tests__/coverage/dependency-check-report.json: found 1 matching files and directories
Uploading artifacts as "archive" to coordinator... ok  id=8712 responseStatus=201 Created token=kHHo_BSB
Cleaning up file based variables
Job succeeded
```

Figura 4.6 – Logs de execução da etapa de verificação de dependências



DEPENDENCY-CHECK

How to read the report | Suppressing false positives | Getting Help: github issues

Project: AgendaENE

Scan Information (show all):

- *dependency-check version*: 6.0.5
- *Report Generated On*: Fri, 16 Apr 2021 22:58:22 GMT
- *Dependencies Scanned*: 15985 (13523 unique)
- *Vulnerable Dependencies*: 4
- *Vulnerabilities Found*: 10
- *Vulnerabilities Suppressed*: 0
- ...

Summary

Display: Showing Vulnerable Dependencies (click to show all)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
docstrap.lib.js		pkg:javascript/bootstrap@3.3.6	MEDIUM	4		3
jquery.min.js		pkg:javascript/jquery@2.0.0	MEDIUM	4		3
nodemailer:6.4.8	cpe:2.3:a:nodemailer:nodemailer:6.4.8:*:*:*:*:*:*:*	pkg:npm/nodemailer@6.4.8	CRITICAL	1	Highest	8
velocity-engine-core-2.2.jar	cpe:2.3:a:apache:velocity_engine:2.2:*:*:*:*:*:*:*	pkg:maven/org.apache.velocity/velocity-engine-core@2.2	HIGH	1	Highest	37

Figura 4.7 – Relatório resultante da etapa de verificação de dependências

Ao primeiro pacote reportado, **pkg:javascript/bootstrap@3.3.6**, foram associados os CVEs: CVE-2018-14040, CVE-2018-14041, CVE-2018-14042, CVE-2019-8331, todos eles associados ao CWE-79 - neutralização imprópria de entrada ao gerar a página web, o que pode gerar um ataque XSS (ENUMERATION, 2021c). Ao segundo pacote **pkg:javascript/jquery@2.0.0**, foram associados os CVEs: CVE-2015-9251, CVE-2019-11358, CVE-2020-11022, CVE-2020-11023, também referenciando o CWE-79. Ao pacote **pkg:npm/nodemailer@6.4.8**, associa-se o CVE-2020-7769 e o CWE-74 - neutralização imprópria de elementos especiais na saída utilizada em componente relacionados (ENUMERATION, 2021a) - podendo levar a injeção de código. Por último, **pkg:maven/ org.apache.velocity/velocity-engine-core@2.2**, relacionado ao CVE-2020-13936, também possui vulnerabilidades de injeção de código.

4.4.2 Verificação estática de segurança

A figura 4.8 mostra os *logs* obtidos no GitLab ao executar esta etapa. A falha se deve à vulnerabilidade encontrada e ilustrada na figura 4.9

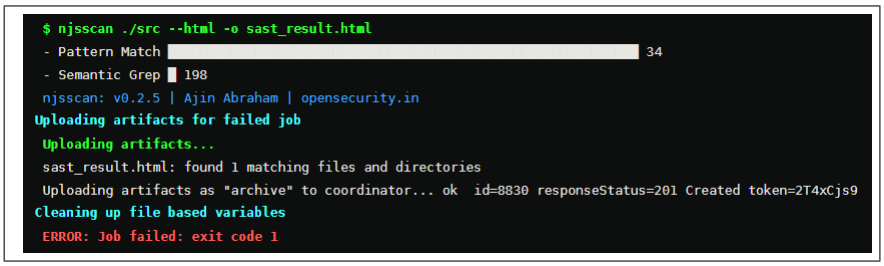


Figura 4.8 – Relatório resultante da verificação estática de segurança do código

A figura 4.9 ilustra o relatório gerado pela verificação estática de segurança.

njsscan: v0.2.6 | Ajin Abraham | opensecurity.in

RULE ID node_nosqli_injection
OWASP A1: Injection
CWE CWE-943: Improper Neutralization of Special Elements in Data Query Logic
DESCRIPTION Untrusted user input in findOne() function can result in NoSQL Injection.
SEVERITY ERROR

FILES

File src/app/controllers/AppointmentController.js
Match Position 9 - 11
Line Number(s) 64: 81
Match String

```
const hourStart = startOfHour(parseISO(date));

// check for past dates
if (isBefore(hourStart, new Date())) {
    return res
        .status(400)
        .json({ error: 'Past dates are not permitted' });
}

// check date availability

const checkAvailability = await Appointment.findOne({
    where: {
        provider_id,
        canceled_at: null,
        date: hourStart,
    },
});
```

File src/app/controllers/NotificationController.js
Match Position 39 - 11
Line Number(s) 6: 8
Match String

```
const checkIsProvider = await User.findOne({
    where: { id: req.userId, provider: true },
});
```

File src/app/controllers/ScheduleController.js
Match Position 41 - 11
Line Number(s) 8: 10
Match String

```
const checkUserProvider = await User.findOne({
    where: { id: req.userId, provider: true },
});
```

File src/app/controllers/UserController.js
Match Position 34 - 11
Line Number(s) 21: 23
Match String

```
const userExists = await User.findOne({
    where: { email: req.body.email },
});
```

Figura 4.9 – Relatório resultante da verificação estática de segurança do código

O relatório acusa uma vulnerabilidade de Injeção de código, associada ao CWE-943. Essa parte do código utiliza a função findOne() (MONGOOSE, 2021) para buscar um

documento no banco de dados utilizando dados provenientes da requisição realizada pelo usuário. Ou seja, o usuário poderia manipular esta requisição para buscar determinados dados no banco de dados.

4.5 Etapa 5 - Implementação de quality gates

A figura 4.10 ilustra a finalização desta etapa, conectando-se com o SonarQube e exportando o código. Observa-se também, que nos *logs* é reportado, na linha 6737, que o código passou pelo *quality gate* estabelecido.

```
6727  20:04:57.330 DEBUG: Upload report
6728  20:04:57.676 DEBUG: POST 200 http://172.17.0.3:9000/api/ce/submit?projectKey=AgendaENE&projectName=AgendaENE | time=344ms
6729  20:04:57.678 INFO: Analysis report uploaded in 348ms
6730  20:04:57.688 DEBUG: Report metadata written to /builds/devsecops/agendaene-backend-tests/.scannerwork/report-task.txt
6731  20:04:57.688 INFO: ------------ Check Quality Gate status
6732  20:04:57.689 INFO: Waiting for the analysis report to be processed (max 300s)
6733  20:04:57.757 DEBUG: GET 200 http://172.17.0.3:9000/api/ce/task?id=AXi9YzB3tFyMWjrG_mQt | time=68ms
6734  20:05:02.841 DEBUG: GET 200 http://172.17.0.3:9000/api/ce/task?id=AXi9YzB3tFyMWjrG_mQt | time=80ms
6735  20:05:07.891 DEBUG: GET 200 http://172.17.0.3:9000/api/ce/task?id=AXi9YzB3tFyMWjrG_mQt | time=48ms
6736  20:05:07.926 DEBUG: GET 200 http://172.17.0.3:9000/api/qualitygates/project_status?analysisId=AXi9Yzf63H4ez17G5nn9 | time=26ms
6737  20:05:07.948 INFO: QUALITY GATE STATUS: PASSED - View details on http://172.17.0.3:9000/dashboard?id=AgendaENE
6738  20:05:07.958 DEBUG: Post-jobs :
6739  20:05:07.974 DEBUG: eslint-bridge server will shutdown
6740  20:05:13.428 INFO: Analysis total time: 40.704 s
6741  20:05:13.432 INFO: ------------------------------------------------------------------------
6742  20:05:13.432 INFO: EXECUTION SUCCESS
6743  20:05:13.432 INFO: ------------------------------------------------------------------------
6744  20:05:13.432 INFO: Total time: 42.803s
6745  20:05:13.531 INFO: Final Memory: 13M/44M
6746  20:05:13.531 INFO: ------------------------------------------------------------------------
6747 Saving cache for successful job
6748  Creating cache sonarqube-check...
6749  .sonar/cache: found 66 matching files and directories
6750  Archive is up to date!
6751  Created cache
6753 Cleaning up file based variables
6755  Job succeeded
```

Figura 4.10 – Logs de execução da etapa de implementação de *quality gates*

Na página principal do SonarQube os projetos são exibidos, como ilustra a figura 4.11, assim como as análises de vulnerabilidades, *bugs* e *code smells* realizadas. As vulnerabilidades de segurança encontradas referem-se àquelas descobertas pela verificação de dependências e que foram exportadas para o SonarQube por meio do *plugin* do *dependency-check*. Dessa forma, o SonarQube atua como centralizador de verificação de cobertura de código, qualidade de código e vulnerabilidades.

Observa-se que ambos projetos estão marcados como *failed*. Isso acontece pois nos dois casos não foi cumprido o *quality gate* estabelecido. No primeiro caso, foram encontradas pelo menos uma vulnerabilidade de dependência de severidade média, além da cobertura de código abaixo do esperado. No segundo, a cobertura de código também ficou abaixo do valor esperado de 75%.

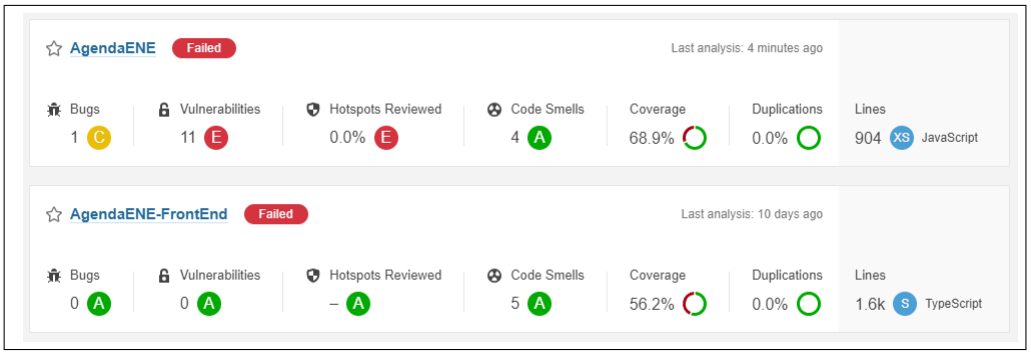


Figura 4.11 – Página principal do SonarQube, com as análises dos repositórios

4.6 Etapa 6 - Construção da imagem

A figura 4.12 evidencia o sucesso da construção da imagem Docker, em que o código definido no Dockerfile é executado, a imagem é gerada utilizando a *hash* do *commit* e salva como arquivo de extensão **tar** para ser utilizada nas próximas etapas.

```
138  src/queue.js -> dist/queue.js
139  src/routes.js -> dist/routes.js
140  src/server.js -> dist/server.js
141  Done in 0.32s.
142  Removing intermediate container f7079d668c3d
143   ---> 26dd204c12ba
144  Step 6/6 : ENTRYPOINT yarn sequelize db:migrate && yarn start
145   ---> Running in d6e67e7e9031
146  Removing intermediate container d6e67e7e9031
147   ---> 893e14390e7f
148  Successfully built 893e14390e7f
149  Successfully tagged latdevsecops/backend:3d84ba0f
150  $ docker save $IMAGE_TAG -o devsecops_image.tar
152 Saving cache for successful job
153  Creating cache master...
154  devsecops_image.tar: found 1 matching files and directories
155  No URL provided, cache will be not uploaded to shared cache server. Cache will be stored only locally.
156  Created cache
158 Cleaning up file based variables
160  Job succeeded
```

Figura 4.12 – Logs de execução da etapa de construção da imagem Docker

4.7 Etapa 7 - Varredura da imagem Docker

A figura 4.13 evidencia os *logs* do GitLab para a verificação de segurança da imagem Docker. Observa-se aqui a quantidade grande de vulnerabilidades de severidade crítica (206), enquanto 2598 vulnerabilidades de severidade baixa foram detectadas. Isso evidencia a importância da utilização de imagens que contenham apenas o essencial para a aplicação ser executada, sem pacotes desnecessários. Como a imagem utilizada foi a **node:12.18.0**, que realiza a instalação de diversos pacotes (NODE, 2021), além de conter alguns pacotes antigos, a quantidade de vulnerabilidades tende a ser mais alta.

A figura 4.14 evidencia parte dos resultados obtidos com a varredura da imagem.

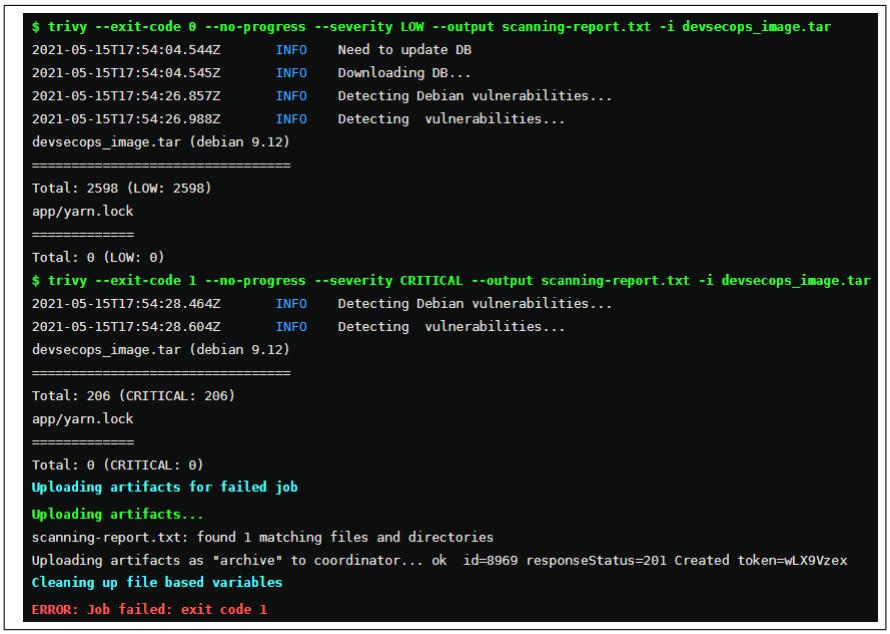


Figura 4.13 – Logs de execução da etapa de varredura da imagem Docker

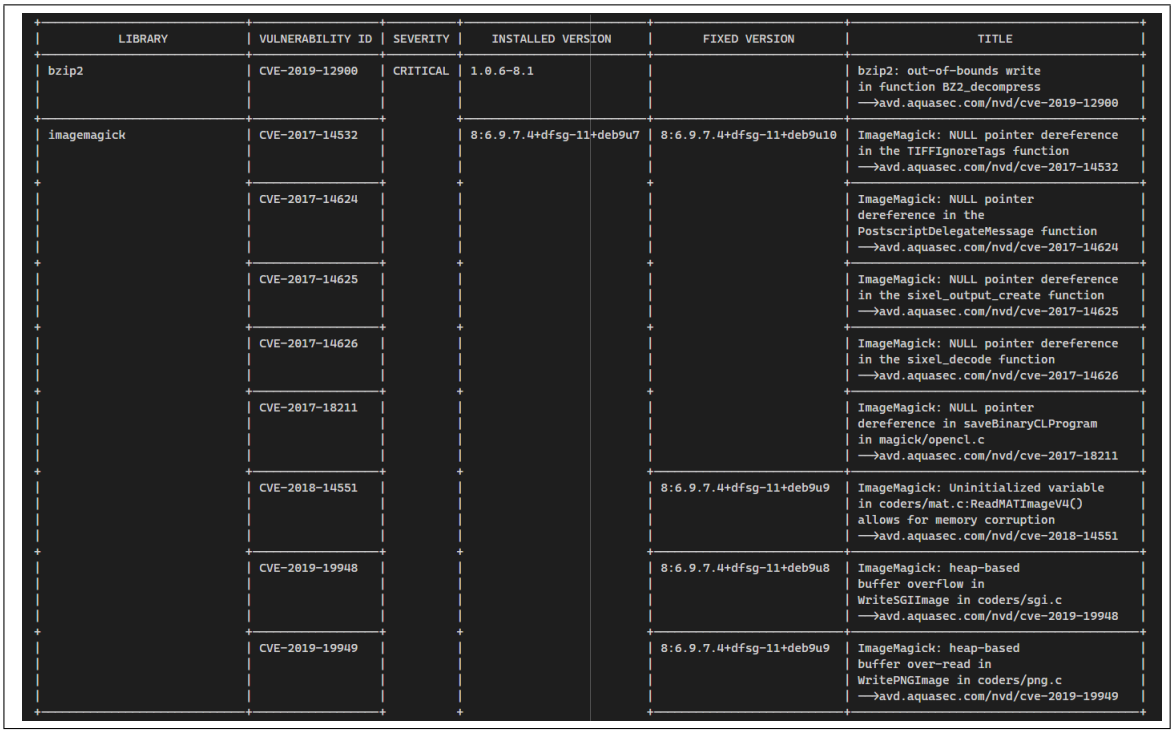


Figura 4.14 – Parte das vulnerabilidades encontradas com o Trivy

Entre os pacotes vulneráveis identificados, encontra-se o bzip2, associada à vulnerabilidade *out-of-bounds write*, descrita no CWE-787. Alguns dos prejuízos que podem ocorrer ao explorar essa vulnerabilidade são no escopo da integridade e disponibilidade do software: modificação de memória, DoS, execução não autorizada de códigos e comandos (ENUMERATION, 2021b). O pacote imagemagick também foi identificado como vulnerabilidade, todas elas associada à vulnerabilidade *NULL pointer deference*, descrita

no CWE-476. A exploração desta vulnerabilidade pode acarretar danos à integridade, disponibilidade e confidencialidade do software, causando leitura e escrita de memória, execução não autorizada de códigos e comandos e DoS.

4.8 Etapa 8 - Envio da imagem para o repositório

Nesta etapa, é realizado o *push* da imagem, após realizada a varredura de vulnerabilidades. Os *logs* de execução do GitLab na figura 4.15 evidenciam o sucesso desta etapa:

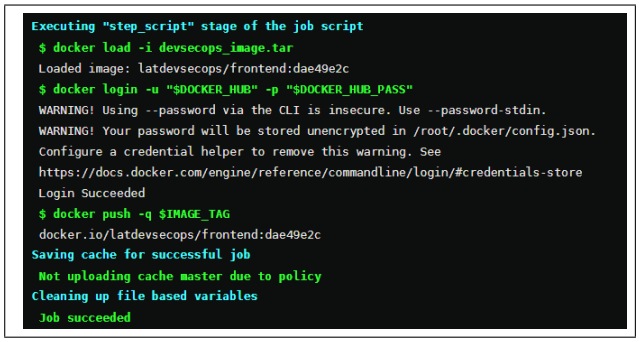


Figura 4.15 – Logs de execução do comando docker push

4.9 Etapa 9 - Testes Dinâmicos

As figuras 4.16 e 4.17 ilustram a execução desta etapa e a exportação dos artefatos. Observa-se que a falha desse estágio é esperada, já que vulnerabilidades foram encontradas.

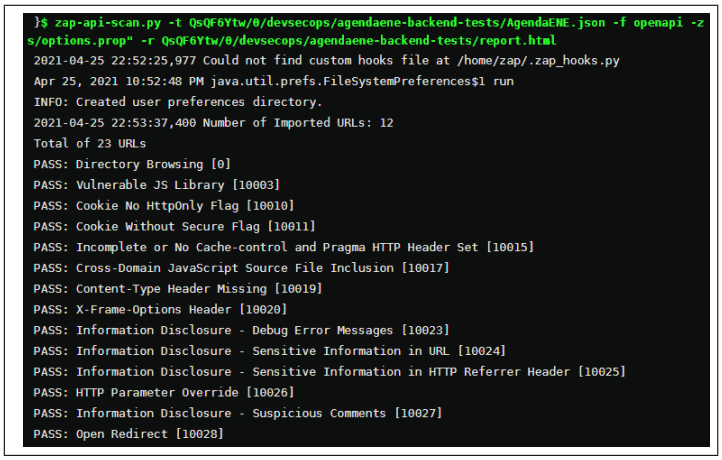


Figura 4.16 – Logs de execução dos testes dinâmicos do OWASP ZAP

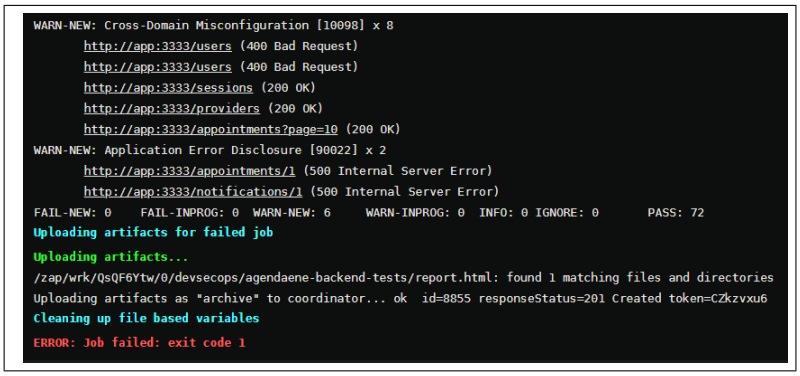
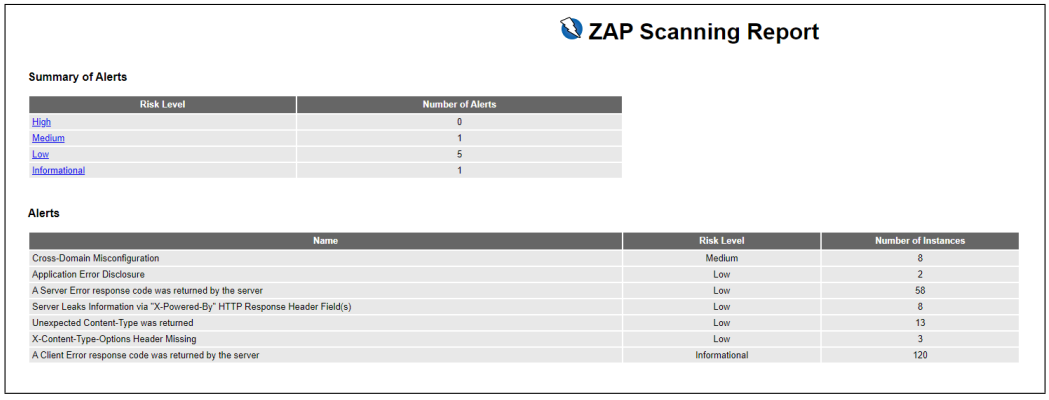


Figura 4.17 – Logs de execução dos testes dinâmicos do OWASP ZAP

O relatório de vulnerabilidades do *back-end*, exportado como resultado da verificação, está ilustrado na figura 4.18.



ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	1
Low	5
Informational	1

Alerts

Name	Risk Level	Number of Instances
Cross-Domain Misconfiguration	Medium	8
Application Error Disclosure	Low	2
A Server Error response code was returned by the server	Low	58
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	8
Unexpected Content-Type was returned	Low	13
X-Content-Type-Options Header Missing	Low	3
A Client Error response code was returned by the server	Informational	120

Figura 4.18 – Relatório exportado como resultado da execução dos testes dinâmicos

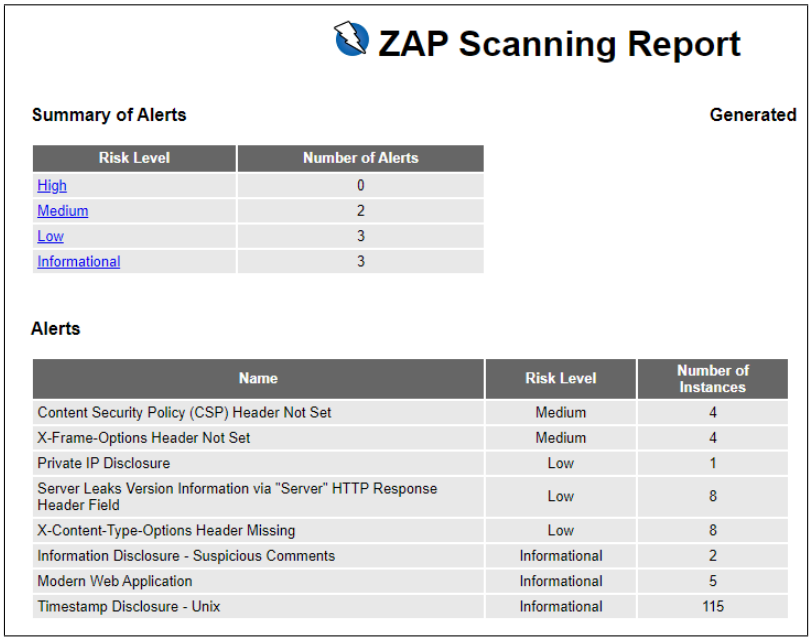
Como observa-se na figura, foram encontradas uma vulnerabilidade de risco médio, cinco de risco baixo e uma de nível informacional. São elas:

- Falha na configuração de *Cross-Domain* (CWE-264): Esta vulnerabilidade foi identificada por meio da análise do campo de resposta **Access-Control-Allow-Origin: ***, permitindo que qualquer origem acesse os recursos da aplicação, ou seja, não há restrição de domínios que têm acesso ao servidor, permitindo requisições de terceiros. Isso poderia ser utilizado por atacantes para acessar dados que não necessitam de autenticação. Essa vulnerabilidade está associada à categoria CWE-264 - Permissões, privilégios e controles de acesso;

- Divulgação de erro de aplicação (CWE-200): a varredura identificou informação sensível nas requisições em que foram retornados erro 500 pelo servidor, associando ao CWE-200 - exposição de informação sensível para usuário não autorizado. No entanto, ao executar a requisição manualmente, não observa-se nenhuma informação além do próprio erro, o que leva a crer que o alerta é um falso positivo;

- Vazamento de informação do servidor via cabeçalho de resposta HTTP "X-Powered-By"(CWE-200): a identificação desta vulnerabilidade ocorreu por meio da análise do cabeçalho HTTP **X-Powered-By**, que é retornado em todas as requisições do servidor, fornecendo informações para um possível atacante sobre quais *frameworks* e componentes foram utilizados para construir a aplicação, o que pode ser utilizado para explorar outras vulnerabilidades e ataques;
- Retorno inesperado de Content-Type: a resposta de algumas URLs retornou um arquivo com o cabeçalho Content-Type: text/html de forma indevida, considerando que todas as URLs deveriam retornar Content-Type:application/json. Não foi identificado nenhum CWE relacionado a esta falha;
- Cabeçalho X-Content-Type-Options não encontrado (CWE-16): a varredura não encontrou o cabeçalho X-Content-Type-Options, que quando configurada para *nosniff*, impede a realização *MIME-sniffing*, técnica utilizada por alguns navegadores mais antigos para identificar o tipo de conteúdo de um arquivo, podendo levar a vulnerabilidades XSS. Esse alerta foi associado ao CWE-16 - Falhas associadas à configuração de software.

Para a aplicação *web*, os alertas gerados estão ilustrados na figura 4.19.



ZAP Scanning Report

Summary of Alerts **Generated**

Risk Level	Number of Alerts
High	0
Medium	2
Low	3
Informational	3

Alerts

Name	Risk Level	Number of Instances
Content Security Policy (CSP) Header Not Set	Medium	4
X-Frame-Options Header Not Set	Medium	4
Private IP Disclosure	Low	1
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	8
X-Content-Type-Options Header Missing	Low	8
Information Disclosure - Suspicious Comments	Informational	2
Modern Web Application	Informational	5
Timestamp Disclosure - Unix	Informational	115

Figura 4.19 – Relatório exportado como resultado da execução dos testes dinâmicos para a *web*

Como ilustrado na figura 4.19, foram gerados dois alertas de risco médio, 3 de risco baixo e 3 informacionais. São eles:

- Cabeçalho Content-Security-Policy (CSP) não configurado (CWE-16): o cabeçalho CSP permite aos administradores do site ter controle sobre os recursos que o agente de usuário é permitido carregar para um certa página (MOZILLA, 2021), o que permite detectar e mitigar ataques XSS, por exemplo;

- Cabeçalho X-Frame-Options não configurado (CWE-16): o cabeçalho X-Frame-Options é utilizado para indicar se o navegador deve ou não renderizar uma página, o que pode ser útil para proteção contra ataques de *click jacking*, em que o atacante cria botões ou links maliciosos na página, que podem realizar ataques como roubo de credenciais e instalação de *malware*;

- Divulgação de IP privado (CWE-200): foi encontrada na resposta da aplicação, o endereço IP no qual responde a API. A informação de um IP interno pode ser útil para que o atacante realize outros tipos de ataques;

- Vazamento de informação do servidor por meio do cabeçalho "Server"(CWE-200): o pacote HTTP de resposta contém informação sobre o servidor web no qual está hospedado (nginx/1.18.0), informação que pode ser utilizada por atacantes para identificar vulnerabilidades no servidor;

- Cabeçalho X-Content-Options não encontrado (CWE-16): alerta similar ao gerado na análise do *back-end*.

4.10 Etapa 10 - Implantação

Nesta etapa, a imagem que está sendo executada no cluster Kubernetes é atualizada, utilizando a imagem recém criada. Os *logs* desta etapa encontram-se na figura 4.20.

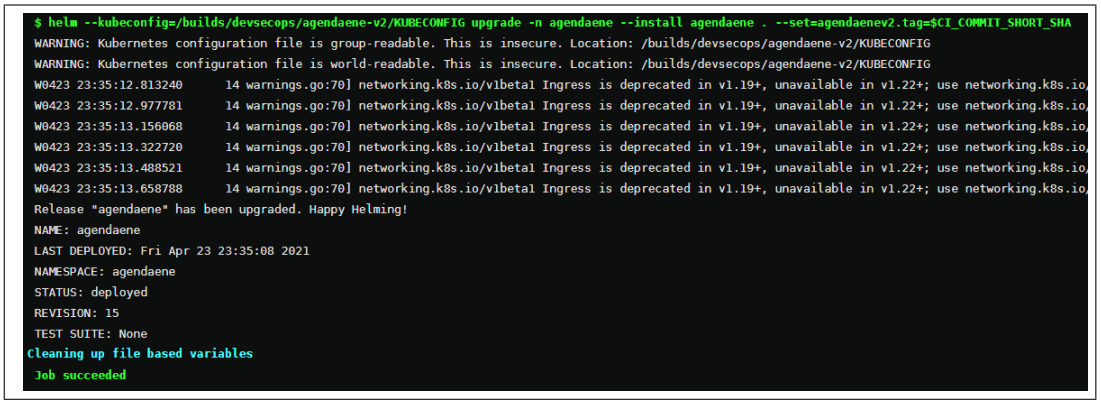


Figura 4.20 – Logs de execução da etapa de implantação no cluster Kubernetes

O Rancher e as aplicações sendo executadas estão ilustradas nas imagens 4.21 e 4.22

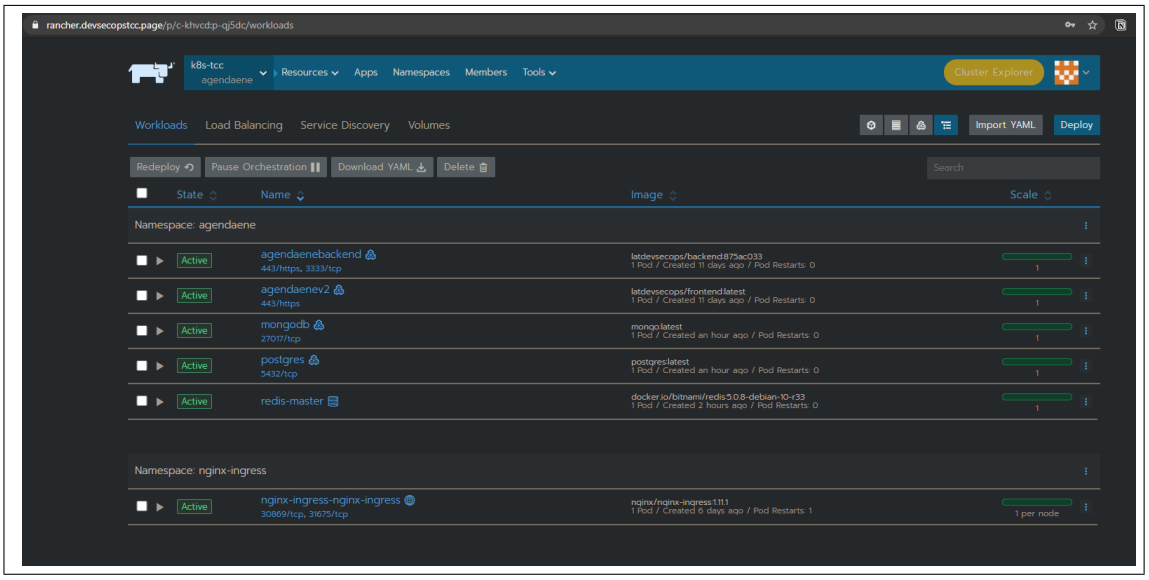


Figura 4.21 – *Workloads* ativos no Rancher

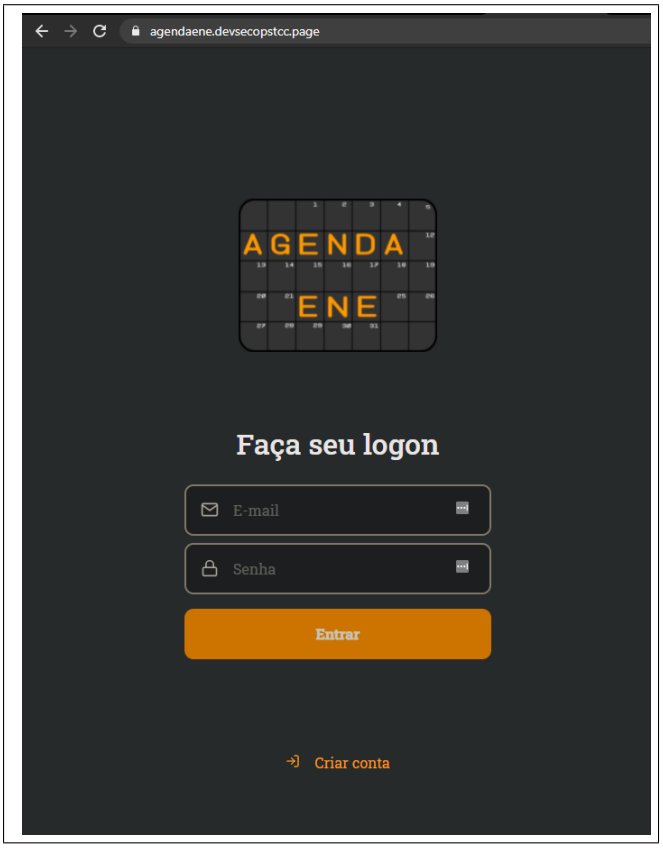


Figura 4.22 – Aplicação em execução na url agendaene.devsecopstcc.page

4.11 Tempo da *pipeline*

Com a utilização de uma metodologia como a DevOps em um projeto, há um impacto direto na forma com que as equipes interagem entre si e, consequentemente, como o

projeto é concluído. No casos do desenvolvimento de software, tal metodologia proporciona um ganho significativo no tempo necessário para realizar o *deploy* da aplicação, como se pôde ver no estudo de caso (CALLANAN; SPILLANE, 2016).

Tal estudo levou em consideração o caso de uma empresa de comércio eletrônico na Austrália que enfrentava dificuldades pelo seu alto tempo de espera para um *deploy* ser realizado, variando tal tempo de 2 dias até semanas. Para resolver este cenário, a empresa implementou a metodologia DevOps, proporcionando assim uma aproximação entre os times e automatização de várias etapas no processo de *deploy*, o que resultou na possibilidade da realização de entregas em questão de horas.

Visando este impacto no tempo de execução, foi realizado um estudo acerca do tempo que cada etapa da *pipeline* desenvolvida leva para ser concluída. Observa-se que os tempos das etapas obtido ao final do trabalho corroboram com o que foi observado em 2.2.4, como ilustra a Tabela 1. Para as etapas com grande variação de tempo, foi calculada uma média das últimas 10 execuções.

Tabela 1 – Tempo de execução de cada etapa da *pipeline*

Metodologia		Etapas	Tempo	Tempo total	
DevSecOps	DevOps	**Construção**	53 segundos	10min58s	2h23min
		Controle de qualidade	20 segundos		
		Testes	30 segundos		
		Implementação de *quality gate*	1min27s		
		Construção da imagem	3min55s		
		Envio da imagem para o repositório	3min28s		
		Implantação	25 segundos		
		Testes estáticos	2min3s (testes estáticos) + 7min50s (verificação de dependências)		
		Varredura da imagem Docker	1min9s		
		Testes dinâmicos	2h01 min		

Como ilustrado na tabela 1, os testes de segurança, especialmente os testes dinâmicos, que ocupam 2h01 minutos da *pipeline*, tomam grande parte do tempo da execução do processo completo. Isso ocorre por conta da varredura ativa, que realiza ataques à aplicação e espera sua resposta. O tempo obtido nesta etapa pode variar de acordo com a

complexidade do software em análise, tempo de resposta e quantidade de *endpoints* testados, bem como os recursos computacionais utilizados para realizar a varredura e execução da aplicação.

4.12 *Pipeline* de geração do executável móvel

A Figura 4.23 evidencia os logs do Gitlab no final da execução da pipeline descrita em 3.4.11. Nota-se que houve o *upload* de um artefato localizado em "*./android/app/build/outputs/apk/release*", sendo este diretório o local onde a APK gerada utilizando dos scripts descritos na pipeline se encontra.

```
4383  Uploading artifacts for successful job
4384   Uploading artifacts...
4385   ./android/app/build/outputs/apk/release: found 3 matching files and directories
4386   Uploading artifacts as "archive" to coordinator... ok  id=8585 responseStatus=201 Created token=Pr3vqHjD
4388  Cleaning up file based variables
4390   Job succeeded
```

Figura 4.23 – Logs de execução da pipeline desenvolvida para a frente mobile

5 Conclusões e trabalhos futuros

Nesse projeto, foi realizado um estudo relacionado a segurança de aplicações web, desenvolvendo uma pipeline DevSecOps utilizando uma aplicação em JavaScript como base, automatizando processos desde a construção da aplicação até a sua implantação, passando por diversos testes de funcionalidade e de segurança. Com isso foram encontradas diversas vulnerabilidades no software, como dependências com falhas de segurança, padrões inseguros utilizados durante o desenvolvimento, pacotes vulneráveis utilizados na imagem Docker e informações importantes do servidor sendo vazadas nos pacotes de resposta. A resolução desses problemas é essencial para o lançamento da aplicação em produção.

No entanto, os testes realizados no projeto não substituem ou eliminam a necessidade da verificação humana. Novas vulnerabilidades e possibilidades de ataque surgem a todo momento, tornando essencial a verificação humana da segurança da aplicação. As ferramentas automatizadas são uma forma de eliminar parte do esforço exercido ao realizar verificações de segurança a cada versão de lançamento do software.

Vale ressaltar também que a implementação eficiente de DevSecOps exige a melhoria contínua do processo. A área de segurança de software está em constante evolução, assim como ferramentas de análise de vulnerabilidades, integração contínua e entrega contínua. Deve haver um *feedback* contínuo dos usuários finais da aplicação para que, a cada lançamento de versão, aspectos relacionados à *pipeline* DevSecOps, funcionalidades e experiência do usuário sejam melhoradas.

Como ilustrado em 4.11, os testes de segurança ocupam grande parte do tempo de execução de uma pipeline DevSecOps. Ao adotar essa metodologia, deve-se ter em mente que a velocidade de entrega poderá ser afetada. No entanto, as vulnerabilidades descobertas mais rapidamente agregam valor ao processo e na qualidade do produto final. Portanto, deve haver uma análise que considere os ganhos obtidos com as verificações de segurança e mitigação dos riscos envolvidos com a diminuição de tempo obtido na entrega quando se prescinde dessas verificações.

5.1 Trabalhos Futuros

Para a verificação completa da aplicação no que tange os aspectos de segurança, é necessário ir além do escopo de aplicações *web* que foi desenvolvido neste trabalho, desenvolvendo testes de aplicação e de segurança para a arquitetura *mobile*.

Além disso, para a segurança completa do processo, é necessário também a utili-

zação de padrões de segurança e descoberta de vulnerabilidades do *cluster* Kubernetes e da infraestrutura subjacente. Para tanto, deve-se realizar análises de segurança utilizando ferramentas adequadas para cada camada da infraestrutura.

Referências

ABRAHAM, A. *Libsast*. 2021. <https://github.com/ajinabraham/libsast>. [Acessado online no dia 19 de abril de 2021]. Citado na página 39.

ABRAHAM, A. *Node JS Scan*. 2021. <https://github.com/ajinabraham/nodejsscan>. [Acessado online no dia 19 de abril de 2021]. Citado na página 39.

ALORAINI, B. et al. An empirical study of security warnings from static application security testing tools. *Journal of Systems and Software*, Elsevier, v. 158, p. 110427, 2019. Citado na página 23.

AQUASECURITY. *Trivy*. 2021. <https://github.com/aquasecurity/trivy>. [Acessado online no dia 14 de abril de 2021]. Citado na página 47.

ARTAC, M. et al. Devops: introducing infrastructure-as-code. In: IEEE. *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. [S.l.], 2017. p. 497–498. Citado na página 18.

BASS, L.; WEBER, I.; ZHU, L. *DevOps: A Software Architect's Perspective*. 1. ed. [S.l.]: Addison-Wesley Professional, 2015. Citado na página 14.

BELLAIRS, R. *What is lint code? And why is linting it important?* 2019. <https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33>. [Acessado online no dia 13 de abril de 2021]. Citado na página 36.

BELLINGARD, F. *Sonar Scanner Package*. 2021. <https://github.com/bellingard/sonar-scanner-npm>. [Acessado online no dia 20 de abril de 2021]. Citado na página 41.

BJÖRNHOLM, J. *Performance of DevOps compared to DevSecOps: DevSecOps pipelines benchmarked!* 2020. Citado 2 vezes nas páginas 23 e 24.

BOSE, S. *Continuous Monitoring in DevOps*. 2020. <https://www.browserstack.com/guide/continuous-monitoring-in-devops>. [Acessado online no dia 02 de abril de 2021]. Citado na página 19.

CALLANAN, M.; SPILLANE, A. Devops: Making it easy to do the right thing. *IEEE Software*, v. 33, n. 3, p. 53–59, 2016. Citado na página 72.

CARTER, K. Francois raynaud on devsecops. *IEEE Software*, IEEE, v. 34, n. 5, p. 93–96, 2017. Citado na página 21.

CERTBOT. *About CertBot*. 2021. <https://certbot.eff.org/about/>. [Acessado online no dia 11 de maio de 2021]. Citado na página 55.

CHECK, D. *Dependecy Check Plugin*. 2021. <https://github.com/dependency-check/dependency-check-sonar-plugin>. [Acessado online no dia 20 de abril de 2021]. Citado na página 41.

CNSS. *CNSS Responsibilities*. 2016. <http://www.cnss.gov/CNSS/about/about.cfm>. [Acessado online no dia 20 de março de 2021]. Citado na página 4.

CORP, R. to. *Semgrep*. 2021. <https://github.com/returntocorp/semgrep>. [Acessado online no dia 19 de abril de 2021]. Citado na página 39.

CVE. *CVE-2021-29418*. 2021. <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-29418>. [Acessado online no dia 14 de abril de 2021]. Citado na página 38.

CWE. *CWE CATEGORY: OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure*. 2017. <https://cwe.mitre.org/data/definitions/1029.html>. [Acessado online no dia 25 de março de 2021]. Citado na página 9.

DATABASE, N. V. *National Vulnerability Database*. 2021. <https://nvd.nist.gov/vuln/search>. [Acessado online no dia 14 de abril de 2021]. Citado na página 39.

DETAILS, C. *Linux Kernel vulnerabilities statistics*. 2019. <https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33>. [Acessado online no dia 10 de abril de 2021]. Citado na página 29.

DEVOPEDIA. *Information Security Principles*. 2016. <https://devopedia.org/information-security-principles>. [Acessado online no dia 20 de março de 2021]. Citado 2 vezes nas páginas x e 6.

DEVOPEDIA. *Container Orchestration*. 2021. <https://devopedia.org/container-orchestration>. [Acessado online no dia 05 de abril de 2021]. Citado 2 vezes nas páginas x e 51.

DOCKER. *Dockerfile Reference*. 2019. <https://docs.docker.com/engine/reference/builder/>. [Acessado online no dia 13 de abril de 2021]. Citado na página 45.

DOCKER. *Use multi-stage builds*. 2019. <https://docs.docker.com/develop/develop-images/multistage-build/>. [Acessado online no dia 14 de abril de 2021]. Citado na página 46.

DOCKER. *Docker Overview*. 2021. <https://docs.docker.com/get-started/overview/>. [Acessado online no dia 03 de abril de 2021]. Citado 2 vezes nas páginas x e 44.

DOCKER. *What is a Container?* 2021. <https://www.docker.com/resources/what-container>. [Acessado online no dia 20 de março de 2021]. Citado 3 vezes nas páginas x, 19 e 20.

EBERT, C. et al. Devops. *IEEE Software*, v. 33, n. 3, p. 94–100, 2016. Citado na página 14.

ENCRYPT, L. *How it works*. 2021. <https://letsencrypt.org/how-it-works/>. [Acessado online no dia 11 de maio de 2021]. Citado na página 55.

ENCRYPT, L. *Let's Encrypt*. 2021. <https://letsencrypt.org/pt-br/>. [Acessado online no dia 11 de maio de 2021]. Citado na página 54.

ENUMERATION, C. W. *CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')*. 2021. <https://cwe.mitre.org/data/definitions/74.html>. [Acessado online no dia 22 de abril de 2021]. Citado na página 62.

ENUMERATION, C. W. *CWE-787: Out-of-bounds Write*. 2021. <https://cwe.mitre.org/data/definitions/787.html>. [Acessado online no dia 15 de maio de 2021]. Citado na página 66.

ENUMERATION, C. W. *CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')*. 2021. <https://cwe.mitre.org/data/definitions/79.html>. [Acessado online no dia 22 de abril de 2021]. Citado na página 62.

ESLINT. *ESLint*. 2019. <https://eslint.org/>. [Acessado online no dia 13 de abril de 2021]. Citado na página 36.

FACEBOOK. *Jest*. 2021. <https://jestjs.io/pt-BR/docs/getting-started>. [Acessado online no dia 16 de abril de 2021]. Citado na página 37.

FACEBOOK. *React*. 2021. <https://pt-br.reactjs.org/>. [Acessado online no dia 07 de abril de 2021]. Citado na página 31.

FACEBOOK. *React native*. 2021. <https://reactnative.dev/>. [Acessado online no dia 07 de abril de 2021]. Citado na página 31.

FACEBOOK. *Yarn*. 2021. <https://engineering.fb.com/2016/10/11/web/yarn-a-new-package-manager-for-javascript/>. [Acessado online no dia 23 de abril de 2021]. Citado na página 31.

FOUNDATION, O. *NodeJS*. 2021. <https://nodejs.org/en/>. [Acessado online no dia 07 de abril de 2021]. Citado na página 30.

FOWLER, M. Continuous integration. 2006. Citado na página 15.

GITLAB. *Cache dependencies in GitLab CI/CD*. 2020. <https://docs.gitlab.com/ee/ci/caching/>. [Acessado online no dia 19 de abril de 2021]. Citado na página 35.

GITLAB. *Job artifacts*. 2020. <https://docs.gitlab.com/ee/ci/pipelines/job_artifacts.html>. [Acessado online no dia 19 de abril de 2021]. Citado na página 35.

GITLAB. *GitLab Executor*. 2021. <https://docs.gitlab.com/runner/executors/README.html>. [Acessado online no dia 03 de abril de 2021]. Citado na página 34.

GITLAB. *Install GitLab Runner manually on GNU/Linux*. 2021. <https://docs.gitlab.com/runner/install/linux-manually.html>. [Acessado online no dia 21 de abril de 2021]. Citado na página 86.

GOOGLE. *Google Cloud Platform*. 2021. <https://cloud.google.com/>. [Acessado online no dia 19 de abril de 2021]. Citado na página 54.

GUPTA, S.; GUPTA, B. B. Cross-site scripting (xss) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management*, Springer, v. 8, n. 1, p. 512–530, 2017. Citado na página 11.

HASHIM, A. *Serious Vulnerability In Netmask npm Package Risked 270K+ Projects*. 2021. <https://latesthackingnews.com/2021/03/31/serious-vulnerability-in-netmask-npm-package-risked-270k-projects>. [Acessado online no dia 14 de abril de 2021]. Citado na página 38.

HASSAN, M. M. et al. Broken authentication and session management vulnerability: a case study of web application. *International Journal of Simulation Systems, Science & Technology*, v. 19, n. 2, p. 6–1, 2018. Citado na página 8.

HELM. *What is Helm?* 2021. <https://helm.sh/>. [Acessado online no dia 07 de abril de 2021]. Citado na página 53.

HEMON, A. et al. From agile to devops: Smart skills and collaborations. *Information Systems Frontiers*, Springer, v. 22, n. 4, p. 927–945, 2020. Citado na página 1.

HOFFMAN, A. *Web Application Security*. 1. ed. [S.l.]: O'Reilly, 2020. Citado na página 6.

HOU, B. et al. Mongodb nosql injection analysis and detection. In: *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*. [S.l.: s.n.], 2016. p. 75–78. Citado na página 7.

HSU, T. H.-C. *Hands-On Security in DevOps: Ensure continuous security, deployment, and delivery with DevSecOps*. [S.l.]: Packt Publishing Ltd, 2018. Citado na página 1.

HUMBLE, J.; FARLEY, D. *Continuous Delivery: Reliable software releases through build, test, and deployment automation*. 1. ed. [S.l.]: Addison-Wesley Signature series, 2010. Citado 2 vezes nas páginas 16 e 17.

JAN, S.; NGUYEN, C. D.; BRIAND, L. Known xml vulnerabilities are still a threat to popular parsers and open source systems. In: *2015 IEEE International Conference on Software Quality, Reliability and Security*. [S.l.: s.n.], 2015. p. 233–241. Citado na página 10.

JAWARNEH, I. M. A. et al. Container orchestration engines: A thorough functional and performance comparison. In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. [S.l.: s.n.], 2019. p. 1–6. Citado na página 51.

JENSEN, S. H.; MØLLER, A.; THIEMANN, P. Type analysis for javascript. In: SPRINGER. *International Static Analysis Symposium*. [S.l.], 2009. p. 238–255. Citado na página 30.

KANG, H.; LE, M.; TAO, S. Container and microservice driven design for cloud infrastructure devops. In: *2016 IEEE International Conference on Cloud Engineering (IC2E)*. [S.l.: s.n.], 2016. p. 202–211. Citado na página 43.

KATALON. *Best 14 CI/CD tools*. 2020. <https://www.katalon.com/resources-center/blog/ci-cd-tools/>. [Acessado online no dia 03 de abril de 2021]. Citado na página 33.

KHAN, A. Key characteristics of a container orchestration platform to enable a modern application. *IEEE Cloud Computing*, v. 4, n. 5, p. 42–48, 2017. Citado na página 51.

KIM, G. et al. *The DevOps Handbook:: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. [S.l.]: IT Revolution, 2016. Citado 3 vezes nas páginas 1, 15 e 16.

KUBERNETES. *Kubernetes*. 2021. <https://kubernetes.io/pt-br/>. [Acessado online no dia 07 de maio de 2021]. Citado na página 32.

KUBERNETES. *Kubernetes Components*. 2021. <https://kubernetes.io/docs/concepts/overview/components/>. [Acessado online no dia 9 de maio de 2021]. Citado na página 52.

KUBERNETES. *Kubernetes Ingress*. 2021. <https://kubernetes.io/docs/concepts/services-networking/ingress/>. [Acessado online no dia 11 de maio de 2021]. Citado 2 vezes nas páginas x e 54.

KUBERNETES. *Service Resources*. 2021. <https://kubernetes.io/docs/concepts/services-networking/service/#service-resource>. [Acessado online no dia 17 de março de 2021]. Citado na página 52.

KUMAR, R.; GOYAL, R. Modeling continuous security: A conceptual model for automated devsecops using open-source software over cloud (adoc). *Computers & Security*, Elsevier, v. 97, p. 101967, 2020. Citado 4 vezes nas páginas x, 2, 20 e 21.

MARCILIO, D. et al. Are static analysis violations really fixed? a closer look at realistic usage of sonarqube. In: IEEE. *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. [S.l.], 2019. p. 209–219. Citado na página 40.

MERKEL, D. Docker: Lightweight linux containers for consistent development and deployment. 2014. Citado na página 20.

MONGOOSE. *Model.findOne()*. 2021. <https://mongoosejs.com/docs/api.html#model_Model.findOne>. [Acessado online no dia 25 de abril de 2021]. Citado na página 63.

MORRIS, K. *Infrastructure as Code: Managing servers in the cloud*. 1. ed. [S.l.]: O'Reilly, 2016. Citado na página 18.

MOZILLA. *Content-Security-Policy*. 2021. <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers/Content-Security-Policy>. [Acessado online no dia 5 de maio de 2021]. Citado na página 70.

MYRBAKKEN, H.; COLOMO-PALACIOS, R. Devsecops: a multivocal literature review. In: SPRINGER. *International Conference on Software Process Improvement and Capability Determination*. [S.l.], 2017. p. 17–29. Citado 2 vezes nas páginas 21 e 22.

NGINX. *Nginx*. 2021. <https://nginx.org/en/>. [Acessado online no dia 07 de maio de 2021]. Citado na página 32.

NODE. *Image layer details*. 2021. <https://hub.docker.com/layers/node/library/node/12.18.0/images/sha256-7ad08ad4aef88011145a250e60e798822fdfdd786903dccf4521c077ed2dd5c0?context=explore>. [Acessado online no dia 22 de abril de 2021]. Citado na página 65.

NVD. *CVE-2019-5736*. 2021. <https://nvd.nist.gov/vuln/detail/CVE-2019-5736>. [Acessado online no dia 10 de abril de 2021]. Citado na página 29.

OPENAPI. *OpenAPI Especification*. 2021. <https://swagger.io/specification/>. [Acessado online no dia 21 de abril de 2021]. Citado na página 49.

OWASP. *A10:2017-Insufficient Logging Monitoring*. 2017. <https://owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2526Monitoring>. [Acessado online no dia 29 de março de 2021]. Citado na página 13.

OWASP. *A1:2017-Injection*. 2017. Citado 2 vezes nas páginas 7 e 8.

OWASP. *A2:2017-Broken Authentication*. 2017. <https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication>. [Acessado online no dia 25 de março de 2021]. Citado 2 vezes nas páginas 8 e 9.

OWASP. *A3:2017 - Sensitive Data Exposure*. 2017. <https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure>. [Acessado online no dia 27 de março de 2021]. Citado na página 9.

OWASP. *A4:2017-XML External Entities (XXE)*. 2017. <https://owasp.org/www-project-top-ten/2017/A4_2017-XML_External_Entities_(XXE)>. [Acessado online no dia 28 de março de 2021]. Citado na página 10.

OWASP. *A5:2017 - Broken Access Control*. 2017. <https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control>. [Acessado online no dia 27 de março de 2021]. Citado 2 vezes nas páginas 10 e 11.

OWASP. *A6:2017 - Security Misconfiguration*. 2017. <https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration>. [Acessado online no dia 27 de março de 2021]. Citado na página 11.

OWASP. *A7:2017-Cross-Site Scripting (XSS)*. 2017. <https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS)>. [Acessado online no dia 28 de março de 2021]. Citado 2 vezes nas páginas 11 e 12.

OWASP. *A8:2017-Insecure Deserialization*. 2017. <https://owasp.org/www-project-top-ten/2017/A8_2017-Insecure_Deserialization>. [Acessado online no dia 28 de março de 2021]. Citado na página 12.

OWASP. *A9:2017-Using Components with Known Vulnerabilities*. 2017. <https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities>. [Acessado online no dia 29 de março de 2021]. Citado 2 vezes nas páginas 12 e 13.

OWASP. *Deserialization Cheat Sheet¶*. 2017. <https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html#:~:text=Deserialization%20is%20the%20reverse%20of,native%20capability%20for%20serializing%20objects.> [Acessado online no dia 28 de março de 2021]. Citado na página 12.

OWASP. *OWASP Top Ten*. 2017. <https://owasp.org/www-project-top-ten/>. [Acessado online no dia 23 de março de 2021]. Citado 3 vezes nas páginas 2, 7 e 12.

OWASP. *OWASP Dependency-Check*. 2019. <https://owasp.org/www-project-dependency-check/>. [Acessado online no dia 14 de abril de 2021]. Citado na página 38.

OWASP. *About the OWASP foundation*. 2021. <https://owasp.org/about/>. [Acessado online no dia 23 de março de 2021]. Citado na página 6.

OWASP. *Active Scan*. 2021. <https://www.zaproxy.org/docs/desktop/start/features/ascan/>. [Acessado online no dia 21 de abril de 2021]. Citado na página 49.

OWASP. *OWASP Zed Attack Proxy (ZAP)*. 2021. <https://www.zaproxy.org/>. [Acessado online no dia 21 de abril de 2021]. Citado na página 48.

OWASP. *Passive Scan*. 2021. <https://www.zaproxy.org/docs/desktop/start/features/pscan/>. [Acessado online no dia 21 de abril de 2021]. Citado na página 49.

OWASP. *Spider*. 2021. <https://www.zaproxy.org/docs/desktop/start/features/spider/>. [Acessado online no dia 21 de abril de 2021]. Citado na página 49.

OWASP. *ZAP Docker user guide*. 2021. <https://www.zaproxy.org/docs/docker/about/>. [Acessado online no dia 21 de abril de 2021]. Citado na página 49.

PAGERDUTY. *What is Continuous Integration*. 2021. <https://www.pagerduty.com/resources/learn/what-is-continuous-integration/>. [Acessado online no dia 19 de março de 2021]. Citado 2 vezes nas páginas x e 17.

PAUL, F. *Continuous Delivery, Continuous Deployment, and Continuous Integration: What's the Difference?* 2018. <https://blog.newrelic.com/engineering/continuous-delivery-continuous-deployment-continuous-integration/>. [Acessado online no dia 19 de março de 2021]. Citado 2 vezes nas páginas x e 18.

PETERSON, J. *Dynamic Application Security Testing: DAST Basics*. 2020. <https://resources.whitesourcesoftware.com/blog-whitesource/dast-dynamic-application-security-testing>. [Acessado online no dia 19 de abril de 2021]. Citado na página 23.

RANCHER. *How Rancher strenghtens Kubernetes*. 2021. <https://rancher.com/why-rancher/rancher-strengthens-kubernetes/>. [Acessado online no dia 05 de abril de 2021]. Citado na página 52.

RANCHER. *Why Rancher?* 2021. <https://rancher.com/why-rancher/>. [Acessado online no dia 05 de abril de 2021]. Citado na página 52.

RILEY, R.; JIANG, X.; XU, D. An architectural approach to preventing code injection attacks. *IEEE Transactions on Dependable and Secure Computing*, v. 7, n. 4, p. 351–365, 2010. Citado na página 7.

SCHWARZ, J.; STEFFENS, A.; LICHTER, H. Code smells in infrastructure as code. In: *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*. [S.l.: s.n.], 2018. p. 220–228. Citado na página 18.

SECURITY, C. The unfortunate reality of insecure libraries. *Constrast Security*, 2014. Citado na página 38.

SIDHU, J.; SAKHUJA, R.; ZHOU, D. *Attacks on eBay*. 2016. [Acessado online no dia 24 de março de 2021]. Citado na página 2.

SONARQUBE. *SonarQube*. 2021. <https://hub.docker.com/_/sonarqube>. [Acessado online no dia 21 de abril de 2021]. Citado na página 88.

SONARSOURCE. *SonarQube*. 2021. <https://docs.sonarqube.org/latest/>. [Acessado online no dia 09 de abril de 2021]. Citado na página 40.

SONARSOURCE. *SonarQube*. 2021. <https://docs.sonarqube.org/latest/user-guide/quality-gates/>. [Acessado online no dia 16 de abril de 2021]. Citado na página 40.

SONARSOURCE. *SonarQube*. 2021. <https://docs.sonarqube.org/latest/analysis/analysis-parameters/>. [Acessado online no dia 19 de abril de 2021]. Citado na página 41.

STALLINGS, W. *Cryptography and Network Security*. 1. ed. [S.l.]: Pearson, 2017. Citado 2 vezes nas páginas 4 e 5.

STASINOPOULOS, A.; NTANTOGIAN, C.; XENAKIS, C. Commix: Automating evaluation and exploitation of command injection vulnerabilities in web applications. *International Journal of Information Security*, Springer, v. 18, n. 1, p. 49–72, 2019. Citado na página 7.

THIYAB, R. M. et al. The impact of sql injection attacks on the security of databases. In: *Proceedings of the 6th International Conference of Computing & Informatics*. [S.l.: s.n.], 2017. p. 323–331. Citado na página 7.

TOMAS, N.; LI, J.; HUANG, H. An empirical study on culture, automation, measurement, and sharing of devsecops. In: *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. [S.l.: s.n.], 2019. p. 1–8. Citado na página 1.

WATTS, S. *A Brief History of DevOps*. 2019. <https://www.bmc.com/blogs/devops-history/>. [Acessado online no dia 09 de março de 2021]. Citado na página 14.

WATTS, S. *DevOps - Basic introduction*. 2019. <https://www.bmc.com/blogs/devops-basics-introduction/>. [Acessado online no dia 09 de março de 2021]. Citado na página 14.

WATTS, S. *DevOps Architecture*. 2019. <https://www.bmc.com/blogs/devops-architecture/>. [Acessado online no dia 17 de março de 2021]. Citado na página 14.

WATTS, S. What is ci/cd. 2019. Citado na página 17.

WHITMAN, M. E.; MATTORD, H. J. *Principles of Information Security*. 4. ed. [S.l.]: Course Technology, 2012. ISBN 1-111-13821-4. Citado 2 vezes nas páginas 4 e 5.

WIGGINS, A. *The Twelve-Factor App*. 2017. <https://12factor.net/pt_br/>. [Acessado online no dia 02 de abril de 2021]. Citado na página 24.

WLATSCHIHA, C. *Sonar Jest Package*. 2021. <https://github.com/3dmind/jest-sonar-reporter#readme>. [Acessado online no dia 20 de abril de 2021]. Citado na página 41.

WURSTER, M. et al. Developing, deploying, and operating twelve-factor applications with tosca. In: *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services*. [S.l.: s.n.], 2017. p. 519–525. Citado na página 24.

ZAPPONI, C. *A SMALL PLACE TO DISCOVER LANGUAGES IN GITHUB*. 2021. <https://githut.info/>. [Acessado online no dia 06 de abril de 2021]. Citado na página 30.

ZIMMERER, P. Strategy for continuous testing in idevops. In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. [S.l.: s.n.], 2018. p. 532–533. Citado na página 19.

Anexos

ANEXO A – Instalação do GitLab Runner

A instalação do GitLab runner foi realizada na máquina disponibilizada pelo Latitude, para isso, foi utilizado o tutorial disponibilizado pela documentação do GitLab GitLab (2021b), utilizando o seguinte comando para realizar o *download* do arquivo binário:

```
sudo curl -L --output /usr/local/bin/gitlab-runner \
"https://gitlab-runner-downloads.s3.amazonaws.com/  \
latest/binaries/gitlab-runner-linux-amd64"
```

Em seguida, executou-se um comando de alteração de permissão para dar permissão de execução ao arquivo:

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

Por fim, os seguintes comandos foram executados, para instalar e iniciar o *runner*, respectivamente.

```
sudo gitlab-runner install
sudo gitlab-runner start
```

Feito isso, o usuário é guiado por um processo de instalação. Ao terminar o processo, é necessário registrar o runner no GitLab, utilizando do *token* especificado na interface do GitLab, com o seguinte comando:

```
gitlab-runner register
```

Terminando o processo, o *runner* estará disponível para utilização.

Figura A.1 – *Runner* disponível na interface do GitLab

ANEXO B – Instalação SonarQube

Para instalação do SonarQube na máquina disponibilizada pelo Latitude, optou-se por executar o servidor contêinerizado. Utilizou-se o o contêiner disponibilizado pelo SonarQube SonarQube (2021), mapeando as portas utilizadas pela aplicação às da máquina hospedeira. Para isso, utilizou-se o seguinte comando:

```
docker run -d --name sonarqube -p 9000:9000 -p 9092:9092 sonarqube
```

ANEXO C – Testes unitários e de integração

Arquivo auxliar para os testes de integração:

```
1  import request from 'supertest';
2  import factory from './factories';
3  import app from '../src/app';
4
5  const createUser = async () => {
6      const user = await factory.attrs('User');
7
8      const userResponse = await request(app).post('/users').send(user);
9
10     const userId = userResponse.body.id;
11
12     return userId;
13 };
14
15 const createProvider = async () => {
16     const provider = await factory.attrs('Provider');
17
18     const responseProvider = await request(app).post('/users').send(provider);
19
20     const providerId = responseProvider.body.id;
21
22     return providerId;
23 };
24
25 const createUserSession = async () => {
26     const response = await request(app).post('/sessions').send({
27         email: 'test@test.com',
28         password: '123456',
29     });
30
31     const { token } = response.body;
```

```
32
33      return token;
34  };
35
36  export { createUser, createProvider, createUserSession };
```

Testes de integração:

```
1   import request from 'supertest';
2   import bcrypt from 'bcryptjs';
3
4   import app from '../../src/app';
5
6   import factory from '../factories';
7   import truncate from '../util/truncate';
8
9   import {
10      createUser,
11      createProvider,
12      createUserSession,
13  } from '../TestDependencies';
14
15
16  describe('User creation', () => {
17
18      beforeEach(async () => {
19          await truncate();
20      });
21
22
23      it('Should be able to register a user', async () => {
24          const user = await factory.attrs('User');
25
26          const response = await request(app).post('/users').send(user);
27
28          expect(response.body).toHaveProperty('id');
29      });
30
31      it('Should not be able to register with duplicated email', async () => {
32          const user = await factory.attrs('User');
```

```
33
34          await request(app).post('/users').send(user);
35
36          const response = await request(app).post('/users').send(user);
37
38          expect(response.status).toBe(400);
39      });
40      it('should encrypt user password when a new user is created', async () => {
41          const user = await factory.create('User', {
42              password: '123456',
43          });
44
45          const compareHash = await bcrypt.compare('123456', user.password_hash);
46
47          expect(compareHash).toBe(true);
48      });
49
50      it('Should not be able to create user with password
51         less than 6 characters', async () => {
52          const user = await factory.attrs('User', {
53              password: '12345',
54          });
55
56          const response = await request(app).post('/users').send(user);
57
58          expect(response.status).toBe(400);
59      });
60  });
61
62  describe('Session creation', () => {
63
64      it('Should be able to create a session', async () => {
65          const user = await factory.attrs('User');
66
67          await request(app).post('/users').send(user);
68
69          const response = await request(app).post('/sessions').send({
70              email: 'test@test.com',
71              password: '123456',
```

```
72          });
73          expect(response.status).toBe(200);
74      });
75
76      it('Should not be able to create a session
77          when validation fails', async () => {
78          await createUser();
79
80          const response = await request(app).post('/sessions').send({
81              email: 'notanemail',
82              password: '123456',
83          });
84          expect(response.status).toBe(400);
85      });
86
87      it('Should not be able to create a session when
88          user is not found', async () => {
89          await createUser();
90
91          const response = await request(app).post('/sessions').send({
92              email: 'email_does_not_exist@test.com',
93              password: '123456',
94          });
95          expect(response.status).toBe(401);
96      });
97
98      it('Should not be able to create a session when password
99          does not match', async () => {
100         await createUser();
101
102         const response = await request(app).post('/sessions').send({
103             email: 'test@test.com',
104             password: 'passworddoesnotmatch',
105         });
106         expect(response.status).toBe(401);
107     });
108 });
109
110 describe('User update', () => {
```

```
111     beforeEach(async () => {
112         await truncate();
113     });
114
115     it('Should be able to update the user', async () => {
116         const user = await factory.attrs('User');
117
118         await request(app).post('/users').send(user);
119
120         const response = await request(app).post('/sessions').send({
121             email: 'test@test.com',
122             password: '123456',
123         });
124
125         const { token } = response.body;
126
127         const userUpdated = await request(app)
128             .put('/users')
129             .set('Authorization', `Bearer ${token}`)
130             .send({
131                 name: 'Updated user',
132             });
133         expect(userUpdated.status).toBe(200);
134     });
135
136     it('Should not be able to update the user
137        when validation fails', async () => {
138         await createUser();
139
140         const token = await createUserSession();
141
142         const userUpdated = await request(app)
143             .put('/users')
144             .set('Authorization', `Bearer ${token}`)
145             .send({
146                 name: 'Updated user',
147                 email: 'notanemail',
148             });
149         expect(userUpdated.status).toBe(400);
```

```
150     });
151
152     it('Should not be able to update the email
153        if it already exists', async () => {
154         await createUser();
155
156         const token = await createUserSession();
157
158         const user2 = {
159             name: 'Test user',
160             email: 'test2@test.com',
161             password: '123456',
162             provider: false,
163         };
164
165         await request(app).post('/users').send(user2);
166
167         const userUpdated = await request(app)
168             .put('/users')
169             .set('Authorization', `Bearer ${token}`)
170             .send({
171                 name: 'Updated user',
172                 email: 'test2@test.com',
173             });
174         expect(userUpdated.status).toBe(400);
175     });
176
177     it('Should not be able to update the password if the
178        old password does not match', async () => {
179         await createUser();
180
181         const token = await createUserSession();
182
183         const userUpdated = await request(app)
184             .put('/users')
185             .set('Authorization', `Bearer ${token}`)
186             .send({
187                 name: 'Updated user',
188                 email: 'test@test.com',
```

```
189                 oldPassword: '123456412',
190                 password: '1234567',
191                 confirmPassword: '1234567',
192             });
193         expect(userUpdated.status).toBe(401);
194     });
195 });
196
197 describe('Appointments', () => {
198     beforeEach(async () => {
199         await truncate();
200     });
201
202     it('Should be able to set an appointment', async () => {
203         const providerId = await createProvider();
204
205         await createUser();
206
207         const token = await createUserSession();
208
209         const appointment = await request(app)
210             .post('/appointments')
211             .set('Authorization', `Bearer ${token}`)
212             .send({
213                 provider_id: providerId,
214                 date: '2021-07-03T18:00:00-03:00',
215             });
216
217         expect(appointment.status).toBe(200);
218     });
219
220     it('Should not be able to create an appointment with a user that
221        is not a provider', async () => {
222         const user = await factory.attrs('User', {
223             email: 'user@test.com',
224             password: '123456',
225         });
226
227         const userResponse = await request(app).post('/users').send(user);
```

```
228
229         const userId = userResponse.body.id;
230
231         await createUser();
232
233         const token = await createUserSession();
234
235         const appointment = await request(app)
236             .post('/appointments')
237             .set('Authorization', `Bearer ${token}`)
238             .send({
239                 provider_id: userId,
240                 date: '2021-07-03T18:00:00-03:00',
241             });
242
243         expect(appointment.status).toBe(401);
244     });
245
246     it('Should not be able to create an appointment without
247        specifying provider_id or date', async () => {
248         await createUser();
249
250         const token = await createUserSession();
251
252         const appointment = await request(app)
253             .post('/appointments')
254             .set('Authorization', `Bearer ${token}`)
255             .send({
256                 date: '2021-07-03T18:00:00-03:00',
257             });
258
259         expect(appointment.status).toBe(400);
260     });
261
262     it('Should not be able to create an appointment
263        in a past date', async () => {
264         const providerId = await createProvider();
265
266         await createUser();
```

```
267
268         const token = await createUserSession();
269
270         const appointment = await request(app)
271             .post('/appointments')
272             .set('Authorization', `Bearer ${token}`)
273             .send({
274                 provider_id: providerId,
275                 date: '2000-07-02T18:00:00-03:00',
276             });
277
278         expect(appointment.status).toBe(400);
279     });
280
281     it('Should not be able to create two appointments
282        at the same time', async () => {
283         await createUser();
284
285         const token = await createUserSession();
286
287         const providerId = await createProvider();
288
289         await request(app)
290             .post('/appointments')
291             .set('Authorization', `Bearer ${token}`)
292             .send({
293                 provider_id: providerId,
294                 date: '2021-07-02T18:00:00-03:00',
295             });
296
297         const appointment = await request(app)
298             .post('/appointments')
299             .set('Authorization', `Bearer ${token}`)
300             .send({
301                 provider_id: providerId,
302                 date: '2021-07-02T18:00:00-03:00',
303             });
304         expect(appointment.status).toBe(400);
305     });
```

```
306
307     it('Should be able to list all appointments', async () => {
308
309         await createUser();
310
311         const token = await createUserSession();
312
313         const response = await request(app)
314             .get('/appointments')
315             .set('Authorization', `Bearer ${token}`);
316
317         expect(response.status).toBe(200);
318     });
319
320     it('Should be able to delete an appointment', async () => {
321         await createUser();
322
323         const token = await createUserSession();
324
325         const providerId = await createProvider();
326
327         const appointment = await request(app)
328             .post('/appointments')
329             .set('Authorization', `Bearer ${token}`)
330             .send({
331                 provider_id: providerId,
332                 date: '2022-07-02T18:00:00-03:00',
333             });
334
335         const appointmentId = appointment.body.id;
336
337         const deletedAppointment = await request(app)
338             .delete(`/appointments/${appointmentId}`)
339             .set('Authorization', `Bearer ${token}`)
340             .send({
341                 provider_id: providerId,
342                 date: '2021-07-02T18:00:00-03:00',
343             });
344
```

```
345         expect(deletedAppointment.status).toBe(200);
346     });
347
348     it('Should not be able to delete an appointment without being
349        the user who created it', async () => {
350         await createUser();
351
352         const token = await createUserSession();
353
354         const providerId = await createProvider();
355
356         const appointment = await request(app)
357             .post('/appointments')
358             .set('Authorization', `Bearer ${token}`)
359             .send({
360                 provider_id: providerId,
361                 date: '2022-07-02T18:00:00-03:00',
362             });
363
364         const appointmentId = appointment.body.id;
365
366         const user = await factory.attrs('User', {
367             email: 'user@test.com',
368             password: '123456',
369         });
370
371         await request(app).post('/users').send(user);
372
373         const session = await request(app).post('/sessions').send({
374             email: 'user@test.com',
375             password: '123456',
376         });
377
378         const tokenUser2 = session.body.token;
379
380         const deletedAppointment = await request(app)
381             .delete(`/appointments/${appointmentId}`)
382             .set('Authorization', `Bearer ${tokenUser2}`)
383             .send({
```

```
384                 provider_id: providerId,
385                 date: '2021-07-02T18:00:00-03:00',
386             });
387
388         expect(deletedAppointment.status).toBe(401);
389     });
390 });
391
392 describe('Availability', () => {
393     beforeEach(async () => {
394         await truncate();
395     });
396
397     it('Should be able to list appointments available', async () => {
398         const date = new Date();
399         const timestamp = date.getTime();
400
401         await createUser();
402
403         const token = await createUserSession();
404
405         const providerId = await createProvider();
406
407         const response = await request(app)
408             .get(`/providers/${providerId}/available?date=${timestamp}`)
409             .set('Authorization', `Bearer ${token}`);
410         expect(response.status).toBe(200);
411     });
412
413     it('Should not be able to list appointments available when
414        date is not provided', async () => {
415         await createUser();
416
417         const token = await createUserSession();
418
419         const providerId = await createProvider();
420
421         const response = await request(app)
422             .get(`/providers/${providerId}/available?date}`)
```

```
423             .set('Authorization', `Bearer ${token}`);
424         expect(response.status).toBe(400);
425     });
426 });
427
428 describe('Providers', () => {
429     beforeEach(async () => {
430         await truncate();
431     });
432
433     it('Should be able to list all providers', async () => {
434         const provider = await createProvider();
435         console.log(provider);
436
437         const user = await createUser();
438         console.log(user);
439
440         const token = await createUserSession();
441         console.log(token);
442
443         const response = await request(app)
444             .get('/providers')
445             .set('Authorization', `Bearer ${token}`);
446         expect(response.status).toBe(200);
447     });
448 });
449
450 describe('Schedule', () => {
451     beforeEach(async () => {
452         await truncate();
453     });
454
455     it('Should be able to list all schedules', async () => {
456         await createProvider();
457
458         const response = await request(app).post('/sessions').send({
459             email: 'testprovider@test.com',
460             password: '123456',
461         });
```

```
462
463         const { token } = response.body;
464
465         const date = new Date();
466
467         const scheduleResponse = await request(app)
468             .get(`/schedule?date=${date}`)
469             .set('Authorization', `Bearer ${token}`);
470
471         expect(scheduleResponse.status).toBe(200);
472     });
473
474     it('Should not be able to show schedules if the user
475        is not a provider', async () => {
476         await createUser();
477
478         const userToken = await createUserSession();
479
480         const date = new Date();
481
482         const scheduleResponse = await request(app)
483             .get(`/schedule?date=${date}`)
484             .set('Authorization', `Bearer ${userToken}`);
485
486         expect(scheduleResponse.status).toBe(401);
487     });
488 });
```

ANEXO D – Arquivo nginx.conf

O arquivo nginx.conf, utilizado na construção da imagem *web*, foi definido como:

```
server {
listen 80;
location / {
root /usr/share/nginx/html;
index index.html index.htm;
try_files $uri $uri/ /index.html =404;
}
}
```

ANEXO E – Definição da API

```
{
"swagger" : "2.0",
"info" : {
  "description" : "This is a simple API",
  "version" : "1.0.0",
  "title" : "Simple Inventory API",
  "contact" : {
    "email" : "you@your-company.com"
  },
  "license" : {
    "name" : "Apache 2.0",
    "url" : "http://www.apache.org/licenses/LICENSE-2.0.html"
  }
},
"host" : "app:3333",
"schemes" : [ "http" ],
"paths" : {
  "/users" : {
    "post" : {
      "summary" : "adds an user",
      "operationId" : "addUser",
      "description" : "Adds an user to the system",
      "consumes" : [ "application/json" ],
      "produces" : [ "application/json" ],
      "parameters" : [ {
        "in" : "body",
        "name" : "UserItem",
        "description" : "User item to add",
        "schema" : {
          "$ref" : "#/definitions/UserItem"
        }
      } ],
      "responses" : {
        "200" : {
          "description" : "user created"
```

```
36            },
37            "400" : {
38              "description" : "validation fails"
39            }
40          }
41        },
42        "put" : {
43          "summary" : "updates an user",
44          "operationId" : "updateUser",
45          "description" : "Updates an user to the system",
46          "consumes" : [ "application/json" ],
47          "produces" : [ "application/json" ],
48          "parameters" : [ {
49            "in" : "body",
50            "name" : "UserItem",
51            "description" : "User item to update",
52            "schema" : {
53              "$ref" : "#/definitions/UserUpdateItem"
54            }
55          }, {
56            "in" : "header",
57            "name" : "Authorization",
58            "type" : "string",
59            "required" : true
60          } ],
61          "responses" : {
62            "200" : {
63              "description" : "user updated"
64            },
65            "400" : {
66              "description" : "validation fails"
67            },
68            "401" : {
69              "description" : "password does not match"
70            }
71          }
72        }
73      },
74      "/sessions" : {
```

```
75        "post" : {
76          "summary" : "create a session",
77          "operationId" : "createSession",
78          "description" : "Create a session",
79          "consumes" : [ "application/json" ],
80          "produces" : [ "application/json" ],
81          "parameters" : [ {
82            "in" : "body",
83            "name" : "SessionItem",
84            "description" : "Session item to add",
85            "schema" : {
86              "$ref" : "#/definitions/SessionItem"
87            }
88          } ],
89          "responses" : {
90            "200" : {
91              "description" : "session created"
92            },
93            "400" : {
94              "description" : "validation fails"
95            },
96            "401" : {
97              "description" : "Password does not match"
98            }
99          }
100       }
101     },
102     "/providers" : {
103       "get" : {
104         "summary" : "Get list of teachers",
105         "operationId" : "teachersList",
106         "description" : "Returns a list of teachers",
107         "parameters" : [ {
108           "in" : "header",
109           "name" : "Authorization",
110           "type" : "string",
111           "required" : true
112         } ],
113         "responses" : {
```

```
114           "200" : {
115             "description" : "Get list of teachers"
116           }
117         }
118       }
119     },
120     "/appointments" : {
121       "post" : {
122         "summary" : "create an appointment",
123         "operationId" : "createAppointment",
124         "description" : "Create an appointment",
125         "consumes" : [ "application/json" ],
126         "produces" : [ "application/json" ],
127         "parameters" : [ {
128           "in" : "body",
129           "name" : "AppointmentItem",
130           "description" : "Appointment to add",
131           "schema" : {
132             "$ref" : "#/definitions/AppointmentItem"
133           }
134         }, {
135           "in" : "header",
136           "name" : "Authorization",
137           "type" : "string",
138           "required" : true
139         } ],
140         "responses" : {
141           "401" : {
142             "description" : "Can only create appointments with teachers"
143           },
144           "400" : {
145             "description" : "Past dates/Appoitment date not available"
146           },
147           "200" : {
148             "description" : "Appointment created"
149           }
150         }
151       },
152       "get" : {
```

```
153         "summary" : "Get list of appointments",
154         "operationId" : "appointmentList",
155         "description" : "Returns a list of appointments",
156         "parameters" : [ {
157           "in" : "query",
158           "name" : "page",
159           "type" : "integer",
160           "description" : "Number of appointment page"
161         }, {
162           "in" : "header",
163           "name" : "Authorization",
164           "type" : "string",
165           "required" : true
166         } ],
167         "responses" : {
168           "200" : {
169             "description" : "Get list of appointments"
170           }
171         }
172       }
173     },
174     "/appointments/{appointmentId}" : {
175       "delete" : {
176         "summary" : "Delete an appointment",
177         "operationId" : "deleteAppointment",
178         "description" : "Delete an appointment",
179         "consumes" : [ "application/json" ],
180         "produces" : [ "application/json" ],
181         "parameters" : [ {
182           "in" : "path",
183           "name" : "appointmentId",
184           "type" : "integer",
185           "required" : true,
186           "description" : "numericID of appointment"
187         }, {
188           "in" : "header",
189           "name" : "Authorization",
190           "type" : "string",
191           "required" : true
```

```
192         } ],
193         "responses" : {
194           "200" : {
195             "description" : "Appointment deleted"
196           }
197         }
198       }
199     },
200     "/schedule" : {
201       "get" : {
202         "summary" : "Show schedule for a day",
203         "operationId" : "showSchedule",
204         "description" : "Show schedule for a day",
205         "parameters" : [ {
206           "in" : "query",
207           "name" : "date",
208           "type" : "string",
209           "format" : "date-time",
210           "required" : true,
211           "description" : "numericID of appointment"
212         }, {
213           "in" : "header",
214           "name" : "Authorization",
215           "type" : "string",
216           "required" : true
217         } ],
218         "responses" : {
219           "200" : {
220             "description" : "Show schedule"
221           }
222         }
223       }
224     },
225     "/notifications" : {
226       "get" : {
227         "summary" : "Show notifications",
228         "operationId" : "showNotifications",
229         "description" : "Show notifications",
230         "parameters" : [ {
```

```
231           "in" : "header",
232           "name" : "Authorization",
233           "type" : "string",
234           "required" : true
235         } ],
236         "responses" : {
237           "200" : {
238             "description" : "Show notifications"
239           }
240         }
241       }
242     },
243     "/notifications/{notificationId}" : {
244       "put" : {
245         "summary" : "updates a notification",
246         "operationId" : "updateNotification",
247         "description" : "Updates a notification",
248         "parameters" : [ {
249           "in" : "body",
250           "name" : "UserItem",
251           "description" : "User item to update",
252           "schema" : {
253             "$ref" : "#/definitions/UserUpdateItem"
254           }
255         }, {
256           "in" : "path",
257           "name" : "notificationId",
258           "type" : "string",
259           "required" : true
260         }, {
261           "in" : "header",
262           "name" : "Authorization",
263           "type" : "string",
264           "required" : true
265         } ],
266         "responses" : {
267           "401" : {
268             "description" : "Only teachers"
269           },
```

```
270           "200" : {
271             "description" : "OK"
272           }
273         }
274       }
275     },
276     "/providers/{providerId}/available" : {
277       "get" : {
278         "summary" : "list of available appointments",
279         "operationId" : "availableAppointments",
280         "description" : "Shows a list of available appointments",
281         "parameters" : [ {
282           "in" : "path",
283           "name" : "providerId",
284           "type" : "string",
285           "required" : true
286         }, {
287           "in" : "query",
288           "name" : "timestamp",
289           "type" : "string",
290           "required" : true
291         }, {
292           "in" : "header",
293           "name" : "Authorization",
294           "type" : "string",
295           "required" : true
296         } ],
297         "responses" : {
298           "400" : {
299             "description" : "Invalid date"
300           },
301           "200" : {
302             "description" : "OK"
303           }
304         }
305       }
306     }
307   },
308   "definitions" : {
```

```
309     "UserItem" : {
310       "type" : "object",
311       "required" : [ "name", "email", "password", "provider" ],
312       "properties" : {
313         "name" : {
314           "type" : "string",
315           "example" : "teste2"
316         },
317         "email" : {
318           "type" : "string",
319           "example" : "teste@teste.com"
320         },
321         "password" : {
322           "type" : "string",
323           "example" : 123456
324         },
325         "provider" : {
326           "type" : "boolean",
327           "example" : false
328         }
329       }
330     },
331     "UserUpdateItem" : {
332       "type" : "object",
333       "required" : [ "name", "email", "oldPassword", "password",
334         "confirmPassword" ],
335       "properties" : {
336         "name" : {
337           "type" : "string",
338           "example" : "teste2"
339         },
340         "email" : {
341           "type" : "string",
342           "example" : "teste@teste.com"
343         },
344         "oldPassword" : {
345           "type" : "string",
346           "example" : 123456
347         },
```

```
348         "password" : {
349           "type" : "string",
350           "example" : 123456
351         },
352         "confirmPassword" : {
353           "type" : "string",
354           "example" : 123456
355         }
356       }
357     },
358     "SessionItem" : {
359       "type" : "object",
360       "required" : [ "email", "password" ],
361       "properties" : {
362         "email" : {
363           "type" : "string",
364           "example" : "teste@teste.com"
365         },
366         "password" : {
367           "type" : "string",
368           "example" : 123456
369         }
370       }
371     },
372     "AppointmentItem" : {
373       "type" : "object",
374       "required" : [ "provider_id", "date" ],
375       "properties" : {
376         "provider_id" : {
377           "type" : "integer",
378           "example" : 1
379         },
380         "date" : {
381           "type" : "string",
382           "format" : "date-time",
383           "example" : "2021-07-06T15:00:00-03:00",
384           "default" : "2021-07-06T15:00:00-03:00"
385         }
386       }
```

```
387         }
388     }
389 }
```

ANEXO F – Script Python para criar sessões

```python
import requests
import json
import logging
from faker import Faker

fake = Faker()
APP_URL = 'http://app:3333'

def user_request():
    data = {
    'name': fake.name(),
    'email': fake.email(),
    'password': '123456',
    'provider': 'false'}

    user_request = requests.post(url=APP_URL + '/users',
                                 data=json.dumps(data),
                                 allow_redirects=True,
                                 verify=False,
                                 headers= {'Content-Type':
                                           'application/json'})

    email = data['email']
    password = data['password']

    return email, password

def session_request(email, password):
    session_data = {
        'email': email,
        'password': password
    }

```

```
34          session_request = requests.post(url=APP_URL + '/sessions',
35                                          data=json.dumps(session_data),
36                                          allow_redirects=True,
37                                          verify=False,
38                                          headers= {'Content-Type':
39                                                    'application/json'})
40
41          token = 'Bearer ' + session_request.json()['token']
42
43          return token
44
45      def create_replacer(token):
46          replacer = ['replacer.full_list(0).description=auth1',
47                      'replacer.full_list(0).enabled=true',
48                      'replacer.full_list(0).matchtype=REQ_HEADER',
49                      'replacer.full_list(0).matchstr=Authorization',
50                      'replacer.full_list(0).regex=false',
51                      'replacer.full_list(0).replacement=' + token]
52
53          return replacer
54
55      def write_to_file(replacer):
56          with open('options.prop', 'a') as fle:
57              for line in replacer:
58                  fle.write(line + '\n')
59
60      email, password = user_request()
61      logging.warning('Created user')
62      token = session_request(email, password)
63      logging.warning('Session created')
64      replacer = create_replacer(token)
65      write_to_file(replacer)
66      logging.warning('Token replaced')
```

ANEXO G – Catálogos Helm

Catálogo do *backend*:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.agendaenebackend.name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: 1
  selector:
    matchLabels:
      app: {{  .Values.agendaenebackend.name }}
  template:
    metadata:
      labels:
        app: {{  .Values.agendaenebackend.name }}
    spec:
      imagePullSecrets:
        - name: {{  .Values.global.registrySecrets }}
      containers:
        - name: {{ .Values.agendaenebackend.name  }}
          securityContext:
            privileged: true
          image: {{  .Values.agendaenebackend.image }}
          ports:
            - name: http
              containerPort: 3333
              protocol: TCP
          resources:
            requests:
              memory: {{ .Values.agendaenebackend.resources.
                requests.memory }}
              cpu: {{ .Values.agendaenebackend.resources.requests.cpu }}
          ports:
          - containerPort: 3333
          imagePullPolicy: Always
```

Catálogo do *frontend*:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.agendaenev2.name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: 1
  selector:
    matchLabels:
      app: {{  .Values.agendaenev2.name }}
  template:
    metadata:
      labels:
        app: {{  .Values.agendaenev2.name }}
    spec:
      imagePullSecrets:
        - name: {{  .Values.global.registrySecrets }}
      containers:
        - name: {{ .Values.agendaenev2.name  }}
          securityContext:
            privileged: true
          image: {{  .Values.agendaenev2.image }}
          ports:
            - name: front-port
              containerPort: 80
              protocol: TCP
          resources:
            requests:
              memory: {{ .Values.agendaenev2.resources.requests.memory }}
              cpu: {{ .Values.agendaenev2.resources.requests.cpu }}
          ports:
          - containerPort: 80
          imagePullPolicy: Always
```

Catálogo do *ingress* para o backend:

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
```

```
metadata:
  name: {{ .Values.agendaenebackend.name }}
  namespace: {{ .Values.namespace }}
  annotations:
    {{- toYaml .Values.podAnnotations | nindent 12 }}
spec:
  rules:
    - host: {{ .Values.agendaenebackend.hosts}}
      http:
        paths:
          - backend:
              serviceName: {{ .Values.agendaenebackend.name }}
              servicePort: 80
```

Catálogo do *ingress* para o *frontend*:

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: {{ .Values.agendaenev2.name }}
  namespace: {{ .Values.namespace }}
  annotations:
    {{- toYaml .Values.podAnnotations | nindent 12 }}
spec:
  rules:
    - host: {{ .Values.agendaenev2.hosts}}
      http:
        paths:
          - backend:
              serviceName: {{ .Values.agendaenev2.name }}
              servicePort: 80
```

Catálogo do serviço:

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Values.agendaenebackend.name }}
  namespace: {{ .Values.namespace }}
spec:
```

```
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: {{ .Values.agendaenebackend.targetPort}}
      protocol: TCP
      name: backend-port
  selector:
      application:  {{ .Values.agendaenebackend.name }}

---
apiVersion: v1
kind: Service
metadata:
  name: {{ .Values.agendaenev2.name }}
  namespace: {{ .Values.namespace }}
spec:
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: {{ .Values.agendaenev2.targetPort}}
      protocol: TCP
      name: front-port
  selector:
      application:  {{ .Values.agendaenev2.name }}
```

Catálogo de valores padrão (values.yaml):

```
# Default values for deployment-chart.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

replicaCount: 1
namespace: agendaene

global:
  registrySecrets:  dockerhub-latdevsecops

agendaenev2:
  enable: true
  name: agendaenev2
```

```
14    image: latdevsecops/frontend:latest
15    targetPort: 80
16    resources:
17      requests:
18        cpu: "500m"
19        memory: "512Mi"
20
21    ingress:
22      enabled: true
23      annotations:
24        kubernetes.io/ingress.class: traefik
25        traefik.ingress.kubernetes.io/router.tls: "true"
26        raefik.ingress.kubernetes.io/router.tls.certresolver: default
27      hosts:
28        - agenda.devsecopstcc.page
29
30  agendaenebackend:
31    enable: true
32    name: agendaenebackend
33    image: latdevsecops/backend:latest
34    resources:
35      requests:
36        cpu: "500m"
37        memory: "512Mi"
38    ingress:
39      enabled: true
40      annotations:
41        kubernetes.io/ingress.class: traefik
42        traefik.ingress.kubernetes.io/router.tls: "true"
43        raefik.ingress.kubernetes.io/router.tls.certresolver: default
44      hosts:
45        - agendaene-api.devsecopstcc.page/
46      targetPort: 3333
47
48  image:
49    pullPolicy: IfNotPresent
50
51  imagePullSecrets: []
52  nameOverride: ""
```

```yaml
fullnameOverride: ""

serviceAccount:
  create: true
  annotations: {}
  name:

podAnnotations:
  kubernetes.io/ingress.class: traefik
  traefik.ingress.kubernetes.io/router.tls: "true"
  traefik.ingress.kubernetes.io/router.tls.certresolver: default

podSecurityContext: {}
  # fsGroup: 2000

securityContext: {}

service:
  type: ClusterIP
  port: 80

ingress:
  enabled: true
  annotations: {}
    # kubernetes.io/ingress.class: nginx
    # kubernetes.io/tls-acme: "true"
  hosts:
    - host: agendaene.devsecopstcc.page
      paths: /
    - host: agenda.devsecopstcc.page/api/?(.*)
      paths: /
  tls: []

resources: {}


nodeSelector: {}

tolerations: []
```

```
92
93 affinity: {}
```

ANEXO H – Instalação do Rancher Server

O Rancher server foi instalado por meio da execução do contêiner Docker, utilizando um certificado gerado pela Let's Encrypt, com o seguinte comando:

```
docker run -d --restart=unless-stopped   -p 80:80 -p 443:443
-v <path_fullchain.pem>:/etc/rancher/ssl/cert.pem
-v <path_privkey.pem>:/etc/rancher/ssl/key.pem
-v <path_cert.pem>:/etc/rancher/ssl/cacerts.pem
--privileged rancher/rancher:latest
```

ANEXO I – Instalação do cluster Kubernetes

Para instalação do cluster Kubernetes, utilizou-se o seguinte comando Docker:

```
docker run -d --privileged --restart=unless-stopped --net=host
-v /etc/kubernetes:/etc/kubernetes
-v /var/run:/var/run  rancher/rancher-agent:v2.5.6
--server https://rancher.devsecopstcc.page/ --token \$TOKEN
--ca-checksum \$CA_CHECKSUM --etcd --controlplane --worker
```