



UNIVERSIDADE D
COIMBRA

Nuno Pedro Rodrigues Simões

APLICACÕES DE TECNOLOGIAS
INFRASTRUCTURE AS CODE (IAC) EM
SEGURANÇA

Dissertação no âmbito do Mestrado em Segurança Informática, orientada pelos
Professores Tiago Cruz e Paulo Simões e apresentada ao Departamento de
Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de
Coimbra.

Julho 2025



DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA
FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE
COIMBRA

Nuno Pedro Rodrigues Simões

APLICAÇÕES DE TECNOLOGIAS
Infrastructure as Code (IaC) EM SEGURANÇA

Dissertação no âmbito do Mestrado em Segurança Informática, orientada
pelos Professores Tiago Cruz e Paulo Simões e apresentada ao Departamento
de Engenharia Informática da Faculdade de Ciências e Tecnologia da
Universidade de Coimbra.

Julho 2025



DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA
**FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA**

Nuno Pedro Rodrigues Simões

SAFETY IN INFRASTRUCTURE AS CODE (IAC)

**Dissertation in the context of the Master in CyberSecurity, advised by
Professor Tiago Cruz and Professor Paulo Simões and presented to the
Department of Informatics Engineering of the Faculty of Sciences and
Technology of the University of Coimbra.**

July 2025

Agradecimentos

Gostaria de expressar a minha profunda gratidão aos meus orientadores, o Professor Doutor Tiago Cruz e o Professor Doutor Paulo Simões, pelo seu acompanhamento, orientação e constante disponibilidade ao longo deste trabalho. A sua experiência e os seus conselhos foram fundamentais para a concretização desta dissertação.

Agradeço ainda aos colegas que partilharam comigo esta caminhada académica e com quem tive a oportunidade de colaborar ao durante deste percurso.

À minha família, em especial à minha esposa e às minhas duas filhas, deixo o meu profundo agradecimento pelo apoio incondicional, incentivo e encorajamento constantes. A vossa compreensão foi essencial em todas as fases deste percurso.

Não posso deixar de agradecer aos meus pais, pelos valores que sempre me transmitiram e pelo exemplo de perseverança e dedicação que sempre me guiaram.

A todos, o meu sincero agradecimento.

Resumo

A crescente complexidade das infraestruturas computacionais tem evidenciado as limitações dos métodos tradicionais de gestão, sublinhando a necessidade de automação e controlo centralizado. Esta dissertação explora a aplicação de tecnologias de Infrastructure as Code (IaC), com foco na melhoria da eficiência, escalabilidade e segurança na gestão de infraestruturas em ambientes baseados na *cloud*, nomeadamente na Amazon Web Services (AWS). A metodologia integra o *Terraform* para o aprovisionamento dos recursos na AWS, o *Ansible* para a gestão contínua da infraestrutura e o *GitHub* para a gestão centralizada do código. Adicionalmente, o *SonarQube* e o *Checkov* são utilizados para analisar e reforçar a conformidade de segurança, assegurando a fiabilidade das alterações implementadas. O cenário proposto visa reduzir erros humanos, melhorar a eficiência das implementações e reforçar a resiliência da infraestrutura na AWS. Com base na implementação prática e respetiva validação, este estudo mostra, que IaC podem otimizar as operações e fortalecer a segurança em ambientes computacionais modernos. Além disso, a adoção de metodologias Development, Security and Operations (DevSecOps) promove a integração contínua de segurança no ciclo de vida da infraestrutura, facilitando a deteção precoce de vulnerabilidades e a aplicação de políticas de controlo rigorosas em ambientes AWS. Os resultados obtidos indicam que a combinação destas tecnologias possibilita ganhos operacionais significativos e uma postura pró-ativa face às ameaças emergentes em *cloud*. Espera-se que este trabalho contribua para incentivar uma adoção mais ampla de IaC, especialmente em plataformas AWS, impulsionando a inovação e segurança nos sistemas futuros.

Palavras-Chave

Infrastructure as Code (IaC), Terraform, Ansible, GitHub, SonarQube, Ckeckov, Segurança, Automação, Computação na nuvem.

Abstract

The increasing complexity of computing infrastructures has highlighted the limitations of traditional management methods, underlining the need for automation and centralized control. This dissertation explores the application of IaC technologies, focusing on improving efficiency, scalability, and security in the management of infrastructures in cloud-based environments, namely on AWS. The methodology integrates *Terraform* for provisioning resources on AWS, *Ansible* for continuous infrastructure management, and *GitHub* for centralized code management. Additionally, *SonarQube* and *Checkov* are used to analyze and strengthen security compliance, ensuring the reliability of implemented changes. The proposed scenario aims to reduce human error, improve deployment efficiency, and enhance infrastructure resilience on AWS. Based on practical implementation and corresponding validation, this study demonstrates that IaC tools can optimize operations and strengthen security in modern computational environments. Moreover, the adoption of DevSecOps methodologies promotes the continuous integration of security throughout the infrastructure lifecycle, facilitating early vulnerability detection and the enforcement of strict control policies in AWS environments. The results indicate that combining these technologies enables significant operational gains and fosters a proactive stance towards emerging threats in the cloud. It is expected that this work will encourage a broader adoption of IaC, especially on AWS platforms, driving innovation and security in future systems.

Keywords

Infrastructure as Code (IaC), Terraform, Ansible, GitHub, SonarQube, Checkov, Security, Automation, Cloud Computing.

Conteúdo

1	Introdução	1
1.1	Motivação	1
1.2	Contexto	2
1.3	Objectivos	2
1.4	Estrutura do documento	3
2	Abordagem adotada e planeamento	5
2.1	Abordagem adotada	5
2.2	Planeamento do trabalho	5
2.3	Sumário	8
3	Contexto tecnológico	9
3.1	Gestão de identidades	9
3.1.1	Autenticação e controlo de acesso	10
3.2	<i>Development and operations</i>	12
3.3	Análise do código	12
3.3.1	<i>Sast</i>	12
3.3.2	<i>Code smells</i>	13
3.3.3	<i>Test smells</i>	15
3.3.4	<i>Security smells</i>	16
3.3.5	Comparativo entre os vários tipos de <i>smells</i>	18
3.4	<i>Pipeline CI/CD</i>	18
3.4.1	Integração contínua - CI	19
3.4.2	Entrega contínua - CD	19
3.5	<i>DevSecOps</i>	19
3.6	Ferramentas e tecnologias para <i>IaC</i>	21
3.6.1	<i>Terraform</i>	21
3.6.2	<i>Ansible</i>	24
3.6.3	<i>SonarQube</i>	27
3.6.4	<i>Checkov</i>	30
3.7	Outras ferramentas e iniciativas relevantes	30
3.7.1	<i>Log4j</i>	30
3.7.2	<i>CVE-2021-44228 (Log4Shell)</i>	30
3.7.3	<i>Web application firewall</i>	31
3.8	Sumário	31
4	Análise do estado da arte	33
4.1	Metodologia adotada	33
4.1.1	Metodologia para a revisão sistemática	33

4.1.2	Identificação de trabalhos relevantes	35
4.1.3	Estratégia para a seleção de trabalhos relevantes	36
4.2	Análise individualizada dos artigos	37
4.3	Discussão	50
4.3.1	Automação e segurança	50
4.3.2	Deteção e mitigação de vulnerabilidades	50
4.3.3	Gestão de segredos	51
4.3.4	<i>Smells</i>	52
4.3.5	Formação e consciencialização	53
4.4	Conclusão	53
5	Desenho e implementação do cenário de avaliação	55
5.1	Descrição do cenário	55
5.1.1	Arquitetura geral	55
5.1.2	Mecanismos de segurança	57
5.1.3	Configuração do <i>identity and access management</i> na AWS . . .	58
5.1.4	<i>GitHub Actions</i>	59
5.1.5	<i>Terraform</i>	62
5.1.6	<i>Ansible</i>	64
5.2	<i>SonarQube</i>	66
5.3	<i>Checkov</i>	67
5.3.1	Objetivo	67
5.3.2	Configuração	67
5.4	Cenário final	68
5.5	Sumário	70
6	Resultados observados	71
6.1	Possíveis problemas a observar — SOA	71
6.2	Resultados CVK	73
6.2.1	Identificação CVK	73
6.2.2	Regras CVK identificadas	73
6.2.3	Limitações	76
6.3	Análise dos resultados	76
6.3.1	Classificação por gravidade	78
6.4	Sumário	80
7	Conclusão	81
	Referências	83

Acrónimos

ACM AWS Certificate Manager.

ALB Application Load Balancer.

ASG Auto Scaling Group.

AWS Amazon Web Services.

CD Continuous Delivery.

CGI Common Gateway Interface.

CI Continuous Integration.

CIA Confidentiality, Integrity, and Availability.

CNA Cloud-Native Architecture.

CPU Central Processing Unit.

CVE Common Vulnerabilities and Exposures.

CVSS Common Vulnerability Scoring System.

CWE Common Weakness Enumeration.

DAC Discretionary Access Control.

DAST Dynamic Application Security Testing.

DDE Deficient Data Encapsulation.

DDS Dense Declarative Structure.

DevOps Development and Operations.

DevSecOps Development, Security and Operations.

DMC DOML Model Checker.

DNS Domain Name System.

DOML DevOps Modeling Language.

DoS Denial of Service.

DSL Domain-Specific Language.

EBS Elastic Block Store.

EC2 Elastic Compute Cloud.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IaC Infrastructure as Code.

IAM Identity and Access Management.

IAST Interactive Application Security Testing.

IoT Internet of Things.

IP Internet Protocol.

KMS Key Management Service.

LDA Latent Dirichlet Allocation.

MAC Mandatory Access Control.

MD5 Message Digest 5.

MEC Multi-access Edge Computing.

MFA Multi-Factor Authentication.

MTTR Mean Time to Recovery.

OIDC OpenID Connect.

OPA Open Policy Agent.

RAM Random Access Memory.

RBAC Role-Based Access Control.

SaaS Software as a Service.

SAST Static Application Security Testing.

SDLC Software Development Life Cycle.

SHA-1 Secure Hash Algorithm 1.

SLIC Security Linter for Infrastructure as Code.

SSH Secure Socket Shell.

SSL Secure Sockets Layer.

SSO Single Sign-On.

TAMA Test Pattern Miner for Ansible.

TI Tecnologia da Informação.

TIC Tecnologias da Informação e Comunicação.

TLS Transport Layer Security.

UFW Uncomplicated Firewall.

VCS Version Control System.

VPC Virtual Private Cloud.

WAF Web Application Firewall.

WSGI Web Server Gateway Interface.

YAML YAML Ain't Markup Language.

Lista de Figuras

2.1	Diagrama de <i>Gantt</i> com o planeamento da dissertação	6
3.1	Arquitectura Identity and Access Management (IAM) [12]	10
3.2	Development and Operations (DevOps) Continuous Integration / Continuous Delivery [15]	12
3.3	<i>Pipeline</i> Continuous Integration (CI)/Continuous Delivery (CD) [20]	19
3.4	<i>DevSecOps - Architecture Design</i> [23]	21
3.5	<i>Terraform - WorkFlow</i> [24]	23
3.6	<i>Terraform - Workspaces</i> [29]	24
3.7	Estrutura do <i>Ansible</i> [31]	25
3.8	<i>WorkFlow</i> do <i>Ansible</i> [32]	26
3.9	Funcionamento do <i>SonarQube</i> [35]	29
4.1	Diagrama de <i>PRISMA</i> do processo de revisão de literatura [49] . .	35
4.2	Artigos seleccionados por ano de publicação	36
5.1	Estrutura geral de ficheiros	56
5.2	<i>IAM - Identify Provider</i>	58
5.3	<i>IAM - Role</i>	59
5.4	Estrutura geral do Github Actions	60
5.5	Configuração de <i>secrets</i> no <i>GitHub Actions</i>	61
5.6	Arquitetura da infraestrutura provisionada com <i>Terraform</i>	63
5.7	<i>Workflow</i> de configuração automatizada com <i>Ansible</i>	65
5.8	Identificação de erros pelo <i>SonarQube</i>	66
5.9	Comando de execução da ferramenta <i>Checkov</i>	67
5.10	Identificação de verificações pelo <i>Checkov</i>	68
5.11	Arquitetura final da infraestrutura provisionada com <i>Terraform</i> . . .	70

Lista de Tabelas

3.1	Comparativo entre os vários tipos de <i>smells</i>	18
4.1	Palavras-chave usadas no processo de pesquisa	34
4.2	Quantidade de artigos por plataforma e por palavra-chave	34
4.3	Ferramentas analisadas e suas valências	51
5.1	Estrutura geral de ficheiros do projeto	57
6.1	Problemas e más práticas em Infrastructure as Code (IaC)	72
6.2	Relatório de problemas detetados pelo <i>Checkov</i>	75
6.3	Associação entre <i>findings</i> do <i>Checkov</i> e CVE detetadas	76

Capítulo 1

Introdução

1.1 Motivação

As práticas tradicionais revelam-se inadequadas para assegurar uma gestão integrada de todos os recursos, especialmente em infraestruturas de grande escala, com elevado grau de interdependência e complexidade crescente. Muitas vezes, nem todos os equipamentos são devidamente geridos ou atualizados e, em muitos casos, dispositivos são desativados ou abandonados sem comunicação formal, o que compromete a visibilidade, segurança e a continuidade das operações [1].

Nos últimos anos, as infraestruturas informáticas têm registado uma evolução significativa, tornando-se cada vez mais extensas, interdependentes e complexas. Esta evolução coloca em causa as capacidades dos métodos tradicionais de gestão, impondo novos desafios às equipas de operações. Os tempos de resposta tornam-se mais lentos, o aprovisionamento de recursos torna-se ineficiente e a configuração manual de cada componente permanece um processo trabalhoso, demorado e suscetível a erros [2]. Estas limitações revelam-se particularmente críticas face às exigências atuais de maior flexibilidade, escalabilidade e fiabilidade na gestão de infraestruturas.

Para ultrapassar estas limitações, torna-se essencial adotar soluções automatizadas que assegurem eficiência, rapidez e precisão, minimizando erros humanos e permitindo que as infraestruturas acompanhem o ritmo das exigências tecnológicas e do mercado. A integração de modelos como o Development, Security and Operations (DevSecOps), aliada ao uso de ferramentas de análise de conformidade de políticas [3], à deteção de anomalias [4] e soluções de automação baseadas em IaC, contribui decisivamente para uma gestão mais segura e resiliente das infraestruturas [5].

Deste modo, a presente dissertação procura analisar como o recurso a práticas de automação, em conjunto com ferramentas de IaC, pode mitigar as fragilidades e promover uma abordagem segura e eficiente na gestão de infraestruturas modernas.

1.2 Contexto

A evolução e a complexidade das infraestruturas e dos sistemas computacionais requerem novas soluções para responder às crescentes necessidades de flexibilidade, rapidez e segurança nos processos de gestão e manutenção. Para satisfazer essas necessidades, surgiram tecnologias e metodologias que visam automatizar e centralizar a gestão de infraestruturas, com destaque para o conceito das ferramentas de IaC, cujo objetivo é melhorar a eficiência operacional, reduzir erros manuais e reforçar a segurança dos ambientes de Tecnologia da Informação (TI) [2] [6].

Tradicionalmente, estas operações eram realizadas manualmente, gerando um cenário em que as equipas trabalhavam de forma isolada, com pouca comunicação e sem um ponto centralizado para documentar alterações. A adoção de ferramentas como *Terraform* e *Ansible* transformou este paradigma, promovendo a automatização de processos e a melhoria da visibilidade sobre os recursos geridos [7] [8].

Com estas ferramentas, as práticas de gestão evoluíram, permitindo às equipas integrar processos, colaborar com maior transparência e documentar as mudanças de forma controlada. Esta abordagem facilita a coordenação entre desenvolvimento, operações e segurança, contribuindo para a redução de conflitos e para a melhoria da resiliência operacional da infraestrutura [9] [10].

A utilização de ferramentas de IaC simplifica a gestão e assegura a integração de práticas de segurança desde o início do ciclo de vida da infraestrutura. Isto garante que os sistemas sejam implementados com maior conformidade e robustez face a falhas. A automação centralizada promovida pela IaC não só aumenta a produtividade das equipas, como também introduz uma camada essencial de segurança, reduzindo o impacto de alterações não autorizadas e melhorando a capacidade de rastreabilidade das operações [5] [11].

1.3 Objectivos

Esta dissertação tem como propósito analisar de que forma a utilização de práticas de IaC, com recurso a ferramentas como *Terraform* e *Ansible*, pode contribuir para a melhoria da segurança em ambientes computacionais, nomeadamente através da identificação e mitigação de más práticas e vulnerabilidades.

Para tal, foi desenvolvido um cenário prático de avaliação, com integração de *pipelines* CI/CD e ferramentas de análise estática, com o objetivo de validar o impacto destas práticas na segurança da infraestrutura.

Através deste trabalho, pretende-se responder às seguintes questões orientadoras:

- Como contribuem ferramentas de IaC, como o *Terraform* e o *Ansible*, para a segurança na gestão de infraestruturas em *cloud*?

- Qual é a eficácia da análise estática na detecção de vulnerabilidades?

Neste contexto, foram desenvolvidas as seguintes iniciativas:

- Construção de um cenário com *pipelines* CI/CD, baseado em Amazon Web Services (AWS), *Terraform*, *Ansible* e *GitHub Actions*, com o objetivo de avaliar a aplicação prática de ferramentas de IaC num ambiente automatizado de provisionamento e configuração de infraestrutura;
- Integração de ferramentas de análise estática no ciclo de desenvolvimento, com o propósito de avaliar a sua eficácia na detecção de más práticas, configurações inseguras e potenciais vulnerabilidades;
- Com base na análise do estado da arte, realizou-se a correlação das falhas identificadas com boas práticas de segurança e com vulnerabilidades conhecidas.

1.4 Estrutura do documento

O resto desta dissertação está organizado da seguinte forma:

- Capítulo 2 – discute a abordagem adotada para este trabalho, assim como o planeamento das diferentes fases do trabalho.
- Capítulo 3 – introduz os conceitos base abordados neste trabalho. Inicialmente, é abordada a gestão de identidades, com destaque para o papel do IAM na definição de acessos seguros em ambientes distribuídos. É apresentado o conceito de DevOps, seguido da análise de código estático, que conduz a uma abordagem de DevSecOps. Além disso, são introduzidas ferramentas de IaC, como o *Terraform* e o *Ansible*, utilizadas para o provisionamento e a gestão da infraestrutura, respetivamente. Apresenta-se o *SonarQube* como ferramenta de análise de código estático, utilizada na identificação de vulnerabilidades e na verificação da conformidade com padrões de segurança. É ainda apresentada a ferramenta *Checkov*, útil na verificação da conformidade de código IaC com boas práticas de segurança. Por fim, é discutida a vulnerabilidade *Log4j* (CVE-2021-44228) e as estratégias de mitigação como o uso de Web Application Firewall (WAF) na proteção de aplicações *web*.
- Capítulo 4 – apresenta um *survey* dos trabalhos de investigação mais relevantes na temática desta dissertação.
- Capítulo 5 – apresenta o desenho e a implementação do cenário de avaliação desenvolvido no âmbito deste trabalho. É descrita a arquitetura da infraestrutura, os mecanismos de segurança implementados e a configuração do sistema de gestão de identidades e acessos IAM na AWS. São ainda detalhados os processos de automação através do *GitHub Actions*, bem como o uso

das ferramentas *Terraform* e *Ansible* para provisionamento e configuração da infraestrutura. O capítulo inclui também a integração das ferramentas *SonarQube* e *Checkov*, com foco na análise de segurança e na verificação da conformidade. Por fim, é apresentado o cenário final, construído com base nos resultados obtidos através da análise Static Application Security Testing (SAST) realizada com o *Checkov*.

- Capítulo 6 – apresenta os resultados observados durante a execução do cenário de avaliação. São analisados os problemas identificados com base no modelo SOA, bem como os resultados obtidos através da ferramenta *Checkov*, incluindo a identificação de regras. Por fim, procede-se a uma análise detalhada dos resultados.
- Capítulo 7 – conclui o documento e discute o trabalho a realizar no futuro.

Capítulo 2

Abordagem adotada e planeamento

2.1 Abordagem adotada

A abordagem adotada para esta dissertação baseia-se na execução de tarefas sequenciais, conforme ilustrado na Figura 2.1. O trabalho encontra-se estruturado em tarefas interdependentes, organizadas de forma a que cada etapa contribua gradualmente e de forma fundamentada para o desenvolvimento e consolidação do cenário final.

2.2 Planeamento do trabalho

Com o objetivo de estruturar e orientar o desenvolvimento do trabalho, foi elaborado um plano detalhado de atividades, representado na Figura 2.1. Este plano apresenta a distribuição temporal das tarefas realizadas ao longo da dissertação.

As tarefas realizadas permitiram estabelecer uma base sólida de conhecimento sobre as ferramentas de Infrastructure as Code (IaC), preparando as bases para o seu aperfeiçoamento e implementação prática. A consolidação dos conhecimentos adquiridos possibilitou a organização estruturada das fases seguintes da dissertação, nas quais se aprofundaram as competências relativas às ferramentas previamente exploradas, nomeadamente *Terraform*, *Ansible*, *GitHub*, *SonarQube* e *Checkov*, com o objetivo de aperfeiçoar a sua utilização e implementar o cenário proposto.

A Figura 2.1 descreve o planeamento das tarefas desenvolvidas durante o processo de aprofundamento do conhecimento teórico e do início da exploração de algumas ferramentas, com vista à elaboração do cenário de avaliação.

Tarefas	Setembro	Outubro	Novembro	Dezembro	Janeiro	Fevereiro	Março	Abril	Maior	Junho
Âmbito da Tese										
Análise dos conceitos ferramentas de Infrastructure as Code (IaC)										
Trabalho exploratório em Terraform										
Trabalho exploratório em Ansible										
Trabalho exploratório em Github										
Elaboração do relatório intermédio										
Elaboração detalhada do cenário										
Trabalho exploratório em SonarQube										
Aperfeiçoamento do conhecimento da ferramenta Terraform										
Aperfeiçoamento do conhecimento da ferramenta Ansible										
Aperfeiçoamento do conhecimento da ferramenta Github										
Aperfeiçoamento do conhecimento da ferramenta SonarQube										
Implementação final do cenário										
Inclusão da ferramenta Checkov										
Validação / Avaliação do cenário										
Elaboração do relatório final										

Figura 2.1: Diagrama de *Gantt* com o planeamento da dissertação

As tarefas realizadas permitiram estabelecer uma base sólida de conhecimento sobre as ferramentas de IaC, preparando as bases para o seu aperfeiçoamento e implementação prática. A consolidação dos conhecimentos adquiridos possibilitou a organização estruturada das fases seguintes da dissertação, nas quais se aprofundaram as competências relativas às ferramentas previamente exploradas, nomeadamente *Terraform*, *Ansible*, *GitHub*, *SonarQube* e *Checkov*, com o objetivo de aperfeiçoar a sua utilização e implementar o cenário proposto.

Adicionalmente, procedeu-se à elaboração detalhada do referido cenário, finalizando com a redação e apresentação do relatório final.

As tarefas foram organizadas da seguinte forma:

- Âmbito da dissertação:
 - Definição do tema e delimitação dos objetivos da dissertação;
 - Aprofundamento teórico sobre os fundamentos da investigação.
- Análise dos conceitos das ferramentas de IaC:
 - Revisão bibliográfica sobre práticas e ferramentas de IaC;
 - Identificação e seleção de ferramentas relevantes para o cenário proposto.
- Trabalho exploratório em *Terraform*:
 - Investigação inicial e testes na criação de recursos de infraestrutura;
 - Desenvolvimento de *scripts* básicos de aprovisionamento.
- Trabalho exploratório em *Ansible*:
 - Exploração das funcionalidades da gestão da configuração com *Ansible*;
 - Criação de *playbooks* simples para configuração automatizada.

- Trabalho exploratório em *GitHub*:
 - Configuração de repositórios para a gestão do código-fonte;
 - Integração preliminar de fluxos de trabalho com *GitHub Actions*.
- Elaboração do relatório intermédio:
 - Compilação dos resultados obtidos nas fases exploratórias;
 - Planeamento das próximas etapas do trabalho.
- Elaboração detalhada do cenário:
 - Definição da arquitetura detalhada do cenário;
 - Especificação dos requisitos técnicos e de segurança.
- Trabalho exploratório em *SonarQube*:
 - Configuração inicial do *SonarQube* para a análise do código;
 - Avaliação da sua integração com *pipelines* de Continuous Integration (CI)/Continuous Delivery (CD).
- Aperfeiçoamento do conhecimento da ferramenta *Terraform*:
 - Desenvolvimento avançado de *scripts* para aprovisionamento automatizado;
 - Integração com ambientes de *cloud* definidos.
- Aperfeiçoamento do conhecimento da ferramenta *Ansible*:
 - Criação de *playbooks* mais robustos e dinâmicos;
 - Automação de processos de configuração pós-provisionamento.
- Aperfeiçoamento do conhecimento da ferramenta *GitHub*:
 - Implementação de *pipelines* de CI/CD otimizadas com *GitHub Actions*;
 - Gestão colaborativa de código e integração de revisões.
- Aperfeiçoamento do conhecimento da ferramenta *SonarQube*:
 - Integração contínua da análise da qualidade do código;
 - Avaliação da cobertura e análise de vulnerabilidades.
- Implementação final do cenário:
 - Criação da infraestrutura completa de acordo com o design planeado;
 - Integração dos serviços e ferramentas de análise.
- Inclusão da ferramenta *Checkov*:
 - Implementação do *Checkov* para a análise da segurança estática de IaC;
 - Identificação e correção de más práticas e vulnerabilidades.

- Validação e avaliação do cenário:
 - Realização de testes finais e avaliação dos resultados;
 - Documentação dos problemas detetados e soluções aplicadas.
- Elaboração do relatório final:
 - Consolidação dos resultados obtidos;
 - Preparação e redação do relatório final da dissertação.

2.3 Sumário

A abordagem e o planeamento apresentados no capítulo mostram uma estrutura baseada em tarefas sequenciais, organizadas em fases interdependentes. Primeiramente, foram realizadas as tarefas iniciais, focadas na análise teórica e na exploração preliminar das ferramentas. De seguida, aprofundou-se o conhecimento nas ferramentas já analisadas, com o objetivo de implementar o cenário proposto.

Importa ainda referir que, ao longo do desenvolvimento, foram realizados ajustes no planeamento inicial, garantindo uma evolução contínua e alinhada com os objetivos estabelecidos, o que resultou na redação e na apresentação do relatório final.

Capítulo 3

Contexto tecnológico

Este capítulo apresenta uma visão geral dos conceitos de IaC com o objetivo de contextualizar os tópicos abordados ao longo desta dissertação. É apresentada uma introdução sobre a gestão de identidade através do Identity and Access Management (IAM), seguida da definição do conceito de Development and Operations (DevOps), da análise de código estático e da apresentação dos diferentes tipos de *smells* que podem ser identificados. De seguida, discute-se a integração de CI/CD nos processos de desenvolvimento, nos quais as análises de código são incorporadas, dando origem ao conceito de Development, Security and Operations (DevSecOps). Por fim, apresentam-se as ferramentas de IaC mais utilizadas: *Terraform* para aprovisionamento, *Ansible* para gestão da infraestrutura, e *GitHub Actions* como elemento do processo de CI/CD em articulação com ferramentas de análise de código, sendo que, no contexto deste trabalho, serão ainda integradas as ferramentas *SonarQube* e *Checkov*, com o objetivo de realizar a análise estática do código e avaliar a conformidade e segurança das configurações implementadas.

3.1 Gestão de identidades

O acesso a recursos, como aplicações, ficheiros e dados, deve ser concedido aos utilizadores que, cada vez mais, estão fora das instalações da organização e que já não se encontram protegidos apenas por *firewalls* locais, como sucedido no passado. Hoje em dia, a forma de trabalhar é bastante diversificada e muitos utilizadores já não estão fisicamente nas instalações da empresa. Por essa razão, necessário definir estratégias de controlar, de forma segura e eficiente, o que os utilizadores têm ou não autorização para aceder. A evolução para ambientes em *cloud* e a crescente mobilidade dos utilizadores requerem arquiteturas robustas de gestão de identidades e acessos, que transcendam aos modelos de segurança baseados unicamente no perímetro físico [12].

3.1.1 Autenticação e controlo de acesso

A autenticação e controlo de acesso, também conhecida como IAM, é uma forma de assegurar que apenas indivíduos autorizados tenham acesso a recursos específicos dentro de uma organização. O IAM possibilita que as organizações definam e administrem as identidades dos utilizadores e as respetivas políticas de acesso. Além disso, o IAM deve gerir as sessões dos utilizadores enquanto estes acedem aos recursos, o que pode incluir a definição de limites para a duração das sessões e a monitorização da atividade do utilizador durante as mesmas.

A Figura 3.1 mostra uma arquitetura de referência para IAM, ilustrando a dinâmica dos processos de provisionamento, autenticação, autorização e gestão de políticas.

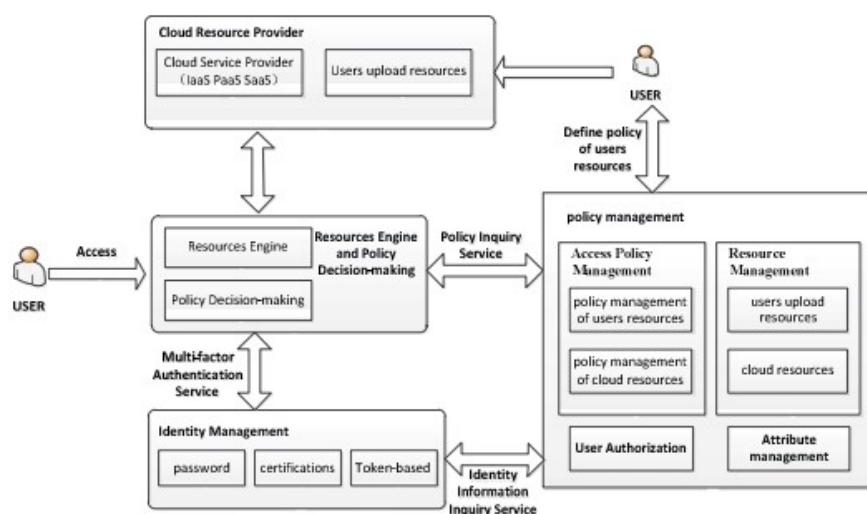


Figura 3.1: Arquitectura IAM [12]

As principais funções são as seguintes:

- **Provisionamento de identidades:** O provisionamento de identidades define a criação e revogação de contas de utilizador.
- **Autenticação:** A autenticação assegura que o utilizador é quem afirma ser. A identidade deve ser verificada utilizando pelo menos dois mecanismos com palavra-passe combinada com Multi-Factor Authentication (MFA) e/ou certificado digital, entre outros.
- **Autorização:** Disponibiliza um interface para a aplicação das regras de autorização quando os utilizadores tentam realizar operações no sistema. Estas regras aplicam-se tanto ao acesso a dados no sistema como às operações que podem ser realizadas sobre esses dados.
- **Gestão de políticas:** O módulo das gestão de políticas aplica as políticas adequadas a um utilizador e determina os recursos para os quais esse utilizador está autorizado [12].

Gestão de identidades

A gestão de identidades é essencial para garantir a segurança no acesso a recursos em ambientes de nuvem [13].

De seguida, detalham-se as principais funcionalidades e os mecanismos que suportam a gestão de identidades:

- Repositório centralizado: Armazena e gere informações da identidade num único local confiável.
- Gestão de atributos: Os atributos dos utilizadores são geridos de forma centralizada para melhorar o desempenho e a experiência do utilizador.
- Provisão e desprovisionamento: Automatiza processos para reduzir custos e tempo.
- Auditorias e relatórios: Permite criar relatórios sobre identidades e redefinir políticas organizacionais.
- Gestão de senhas: Serviços de autoatendimento para recuperação e redefinição, reduzindo envolvimento de suporte.
- Gestão de contas de serviço: Trata contas administrativas e de *login* de servidores e aplicações.
- Single Sign-On (SSO) e federação: Facilita acesso único a múltiplos serviços sem múltiplos inícios de sessão.

Gestão de acessos

A gestão de acessos assegura que apenas os utilizadores autorizados podem aceder aos recursos, conforme políticas predefinidas. Os acessos são concedidos ou revogados de forma ágil e sem necessidade de intervenção manual, aumentando a eficiência em função dos papéis desempenhados e garantindo um acesso apropriado [13].

Destacam-se diferentes tipos de controlo de acesso dentro da gestão de acessos, os quais contribuem para uma implementação eficaz da segurança e da gestão da informação:

- Controlo de acesso obrigatório - Mandatory Access Control (MAC).
- Controlo de acesso discricionário - Discretionary Access Control (DAC).
- Controlo de acesso baseado em funções - Role-Based Access Control (RBAC).
- Controlo de acesso baseado em políticas.
- Controlo de acesso baseado em risco.

3.2 Development and operations

O conceito de DevOps resulta da junção de "desenvolvimento" e "operações" e tem como finalidade promover ciclos de desenvolvimento contínuos, permitindo disponibilizar *software* com maior rapidez, de modo fiável e eficiente. Baseia-se nas práticas de integração contínua (CI), entrega contínua (CD) e implementação contínua. Na integração contínua, todas as alterações no código são frequentemente integradas no processo de automação da construção de *builds*. Já a entrega contínua automatiza o lançamento de *software*, possibilitando implementações rápidas e frequentes, assegurando a qualidade através de testes automatizados. A implementação do conceito de DevOps trouxe benefícios significativos para organizações em todo o mundo. Empresas líderes, como a *Amazon*, *Netflix* e *Google*, implementaram as práticas de DevOps, tendo essa abordagem produzido substanciais melhorias na produtividade, estabilidade e aceleração da inovação. No entanto, a adoção do DevOps ainda enfrenta desafios, como a falta de conhecimento técnico específico, a complexidade da modernização de sistemas *legacy*, lacunas na cultura organizacional e dificuldades na comunicação das equipas [14].

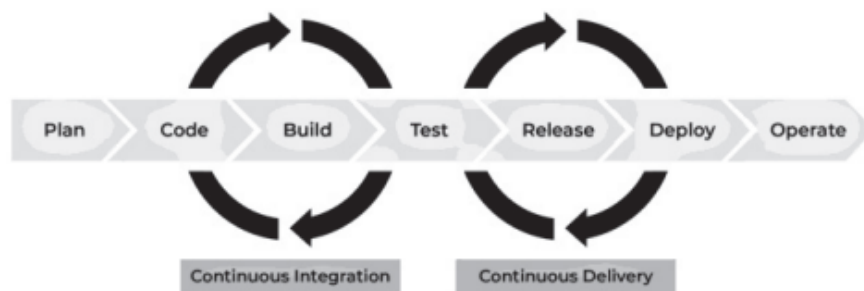


Figura 3.2: DevOps Continuous Integration / Continuous Delivery [15]

A Figura 3.2 descreve as várias fases do ciclo de desenvolvimento e *deployment* de *software* nas práticas de *Continuous Integration* / *Continuous Delivery*.

3.3 Análise do código

A análise do código permite identificar possíveis vulnerabilidades de segurança, problemas no desempenho e outras anomalias no código que podem resultar em falhas ou violações de segurança. Este processo desempenha um papel essencial no ciclo de vida do desenvolvimento do *software*, pois contribui para melhorar a qualidade do código, reduzir custos e verificar se as aplicações cumprem os padrões relevantes de segurança e conformidade [16].

3.3.1 Sast

Static Application Security Testing (SAST) é uma técnica de análise de código-fonte ou de binários que não requer a execução do *software*. Esta técnica tem com

objetivo identificar possíveis erros, vulnerabilidades e falhas no código, devendo ser aplicada nas fases iniciais do desenvolvimento de *software*, permitindo reduzir custos no processo de correção e mitigar riscos associados à exploração futura de vulnerabilidades [17].

Gestão dos resultados dos testes

É necessário ter atenção na gestão e na manutenção dos resultados do SAST:

- Perda de dados: Informações críticas sobre vulnerabilidades podem ser perdidas devido a falhas no sistema ou a eliminação acidental.
- Falha no suporte: Resultados desatualizados após alterações no código podem levar à não detecção de vulnerabilidades, impactando negativamente a segurança do *software*.
- Perda de contexto: A ausência de um histórico organizado dificulta a rastreabilidade das vulnerabilidades corrigidas e o entendimento das mudanças realizadas no código.
- Volumes de dados extensos: Projetos de grande dimensão podem gerar uma enorme quantidade de dados, exigindo ferramentas com maior capacidade e eficácia na análise e no armazenamento dos resultados.
- Segurança dos dados: Os resultados do SAST podem conter informações sensíveis. É essencial implementar mecanismos para prevenir acessos não autorizados e perdas de dados.
- Confiabilidade reduzida: A ausência de uma gestão adequada pode levar os programadores a ignorar os resultados, diminuindo a eficiência do processo.

3.3.2 Code smells

Code smells, no âmbito de IaC, correspondem a padrões, práticas ou outras características do código (presentes nos ficheiros de configuração usados para configurar, gerar e implementar infraestruturas como código) que podem indiciar inconsistências, más escolhas de *design* ou violações de padrões, as quais podem levar a problemas de manutenção ou defeitos. Contudo, os *code smells* não representam necessariamente *bugs* ou erros técnicos, pois o código continua funcional. No entanto, são sinais de alerta que requerem maior atenção e uma análise mais cuidada [18].

Os *code smells* podem ocorrer devido a:

- Complexidade crescente da infraestrutura: À medida que os sistemas evoluem em quantidade e complexidade, a gestão da infraestrutura pode ficar mais difícil.

- Violações de boas práticas: Uso de padrões inadequados para modularidade, estrutura ou encapsulamento, bem como escolhas inadequadas de encriptação.
- Dependências obsoletas: Uso de comandos ou configurações desatualizadas.

Os *code smells* podem ser detetados de diversas formas, incluindo:

- Ferramentas automatizadas: Ferramentas que auxiliam na identificação de padrões problemáticos de *design* e implementação.
- Revisões manuais: Programadores experientes podem identificar práticas inadequadas durante o processo de revisão de código.
- Testes e validações: Identificação de *commits* associados a defeitos, que podem auxiliar na deteção de *code smells*.

De acordo com [18], os *code smells* podem ser classificados em dois níveis:

- *Code smells* relativos à implementação:
 - Expressões complexas (ICE): Código difícil de compreender.
 - Tarefas incompletas (IIT): Comentários como "*to-do*" ou "*fix-me*".
 - Uso de declarações obsoletas (IDS): Comandos desatualizados.
 - Condicionais incompletas (IIC): Condicionais sem cláusula "*else*".
- *Code smells* relativos ao *design*:
 - Abstrações multifacetadas (DMF): Violações do princípio da responsabilidade única.
 - Modularização insuficiente (DIM): Ficheiros excessivamente grandes.
 - Modularidade enfraquecida (DWM): Elevada interdependência e reduzida coesão.
 - Estruturas densas (DDS): Dependências excessivas.
 - Encapsulamento deficiente (DDE): Uso inadequado de variáveis globais.

Os programadores têm à sua disposição diversas práticas para mitigar a criação de *code smells*, das quais se destacam as seguintes:

- Modularização: Criação de módulos pequenos e coesos, de modo a reduzir a complexidade e as dependências excessivas.
- Validação contínua: Utilização de ferramentas para detetar problemas de *design* ou implementação.

- Manutenção contínua: Melhoria da estrutura e organização dos ficheiros, de forma a torná-los mais legíveis e alinhados com as boas práticas.
- Automação: Implementação de práticas de CI/CD para detetar e corrigir problemas.
- Documentação: Definições escritas e mantidas atualizadas das escolhas de *design* e modularidade existentes.
- Formação das equipas: Formação dos programadores para reconhecer e evitar a criação de *code smells*.

3.3.3 Test smells

Test smells, no âmbito da IaC, são padrões ou características potencialmente inadequadas presentes em *scripts* ou casos de teste que, embora não causem falhas imediatas, podem comprometer a eficácia, a manutenção e a fiabilidade dos testes. Estes sinais indicam potenciais problemas na conceção ou na implementação desses testes, tornando-os mais sensíveis a erros, dificultando a compreensão da sua abrangência e complicando a deteção de problemas aquando de mudanças frequentes no código ou na infraestrutura [9].

Os *test smells* podem surgir devido a padrões ou práticas inadequadas na criação de testes para código ou infraestrutura, tais como:

- Falta de experiência dos programadores: Práticas inadequadas na escrita de testes devido a conhecimento limitado de boas práticas.
- Utilização deficiente de padrões: Uso de estruturas de teste mal configuradas ou existência de dependências complexas.
- Negligência no design dos testes: Não são aplicadas as melhores práticas para a modularidade, clareza e eficiência.
- Automação: Falta de implementação adequada de práticas de CI/CD para deteção precoce e correção dos problemas.
- Condições específicas da infraestrutura: Testes que dependem de ambientes locais ou recursos externos suscetíveis a variações.

Os *test smells* podem ser identificados das seguintes formas:

- Análise manual: Verificação manual dos *scripts* de teste por programadores experientes com o intuito de identificar problemas estruturais ou padrões inadequados.
- Ferramentas de análise estática: Utilização de ferramentas especializadas para a verificação de código.

Os *test smells* podem enquadrar-se em três tipos distintos:

- *Assertion roulette*: Utilização de múltiplas *assertion* sem informações claras sobre o que falhou, dificultando a identificação de problemas.
- *Local only testing*: Testes que dependem exclusivamente do ambiente local.
- *Remote mystery guest*: Dependência de recursos externos (como *APIs* ou servidores) sem clareza sobre as suas configurações ou estado.

Entre as melhores práticas recomendadas para evitar *test smells* incluem-se as seguintes estratégias:

- Modularizar os testes:
 - Dividir os testes grandes em testes mais pequenos e específicos.
 - Evitar dependências complexas entre testes.
- Documentar os recursos externos:
 - Garantir que as dependências externas estejam bem descritas e definidas.
 - Reduzir o uso de recursos externos sempre que possível.
- Automatizar validações:
 - Utilizar ferramentas para detetar *test smells* automaticamente, como *linters* e analisadores.
- Simular ambientes de produção:
 - Configurar testes em ambientes que reflitam as condições reais de produção, aumentando a fiabilidade.
- Priorizar clareza e manutenção:
 - Utilizar nomes e mensagens descritivas em *assertion*.
 - Garantir que os testes sejam fáceis de compreender e manter.

3.3.4 *Security smells*

Security smells são padrões recorrentes em *scripts* de código que indicam potenciais problemas de segurança. Estes padrões não correspondem necessariamente a vulnerabilidades, mas devem ser encarados como sinais de alerta que necessitam de verificação e, se necessário, posterior mitigação [11].

As principais causas do aparecimento de *security smells* incluem:

- Falta de sensibilização sobre segurança: Os programadores podem não estar conscientes das consequências de práticas inseguras, como o uso de senhas *hard-coded*.
- Limitações das ferramentas: Ferramentas de gestão de credenciais e configurações seguras são subutilizadas ou pouco conhecidas.
- Pressão para entrega rápida: A urgência em entregar rapidamente o produto final pode provocar problemas, como a omissão de verificações de integridade ou a configuração incorreta de endereços Internet Protocol (IP).
- Complexidade do ambiente: Ambientes com múltiplos serviços e grandes dependências podem resultar em erros de configuração.

Os tipos mais comuns de *security smells* são os seguintes:

- *Hard-coded secret*: Informações sensíveis, como senhas ou chaves criptográficas, inseridas diretamente no código-fonte.
- *Empty password*: Utilização de senhas vazias, facilitando ataques de força bruta.
- *Unrestricted IP address binding*: Configuração de serviços para aceitar conexões de qualquer endereço IP, mais concretamente 0.0.0.0, expondo o sistema a um leque demasiado amplo de ameaças.
- *No integrity check*: Omissão da verificação da integridade da origem do código-fonte utilizado em pacotes ou repositórios.
- *Using Hypertext Transfer Protocol (HTTP) without Transport Layer Security (TLS)*: Comunicação sem criptografia, tornando-a vulnerável a ataques do tipo *man-in-the-middle*.
- *Using weak cryptography algorithms*: Utilização de algoritmos criptográficos obsoletos, como Message Digest 5 (MD5) ou Secure Hash Algorithm 1 (SHA-1).
- *Suspicious comment*: Comentários deixados no código-fonte que indicam fragilidades ou funcionalidades incompletas, como *“to-do”* ou *“fix-me”*.
- *Administrator by default*: Manutenção da configuração padrão com privilégios administrativos desnecessários.

Para identificar *security smells*, usam-se essencialmente três abordagens distintas [19]:

- Ferramentas de análise estática: Como o Security Linter for Infrastructure as Code (SLIC), que identifica automaticamente práticas inseguras.
- Revisões manuais: Programadores ou especialistas em segurança analisam os *scripts*.
- Auditorias de conformidade: Avaliações baseadas em normas de segurança reconhecidas, como a Common Weakness Enumeration (CWE).

3.3.5 Comparativo entre os vários tipos de *smells*

A Tabela 3.1 apresenta uma comparação resumida entre os três tipos de *smells* anteriormente analisados.

Aspeto	Code Smells	Test Smells	Security Smells
Foco	Design e qualidade do código.	Qualidade e eficácia dos testes.	Segurança do sistema.
Impacto Principal	Facilidade de manutenção e legibilidade.	Confiabilidade e cobertura dos testes.	Proteção contra ataques e vulnerabilidades.
Exemplo de smells	Código duplicado, métodos longos.	Dados <i>hard-coded</i> , dependências frágeis.	Configurações públicas, senhas no código.
Detectado por	Revisões de código, <i>linters</i> .	Revisões de testes, ferramentas específicas.	Ferramentas de análise estática e dinâmica.
Soluções	Refatorização, modularização.	Reestruturação de testes, isolamento.	Criptografia, validação de entradas.

Tabela 3.1: Comparativo entre os vários tipos de *smells*

3.4 Pipeline CI/CD

O termo *pipeline* CI/CD designa um conjunto de práticas e ferramentas utilizadas para melhorar o processo de desenvolvimento do *software*, automatizando as *builds*, os testes e o *deployment*. Esta abordagem permite alterações no código de forma rápida, fiável e rastreável. Na prática, corresponde ao encadeamento de duas fases (Figura 3.3):

- Integração contínua (CI): Criação, teste e integração automática de alterações do código num repositório.
- Entrega contínua (CD): Envio automático das alterações do código para ambientes preparados para produção, ficando pendente de aprovação.

A Figura 3.3 descreve as várias fases do pipeline do ciclo de desenvolvimento e *deployment* de *software* no *Continuous Integration* e *Continuous Delivery*.

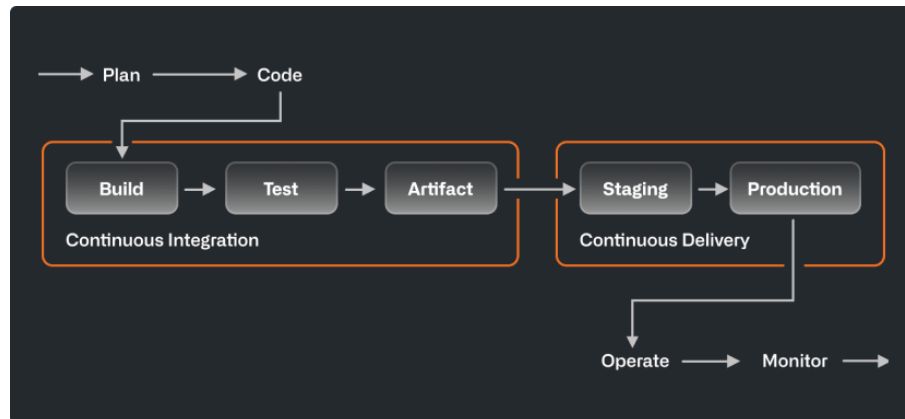


Figura 3.3: Pipeline CI/CD [20]

3.4.1 Integração contínua - CI

A Integração contínua (CI) é um processo de desenvolvimento de *software* onde os vários programadores que estão a construir o produto integram, regularmente, as suas alterações no código, num repositório partilhado. Cada integração desencadeia um processo automatizado de compilação e teste, garantindo que o novo código se integra corretamente com a base do código existente. São utilizados testes automatizados para validar as alterações e identificar possíveis problemas.

3.4.2 Entrega contínua - CD

A Entrega contínua (CD) é uma prática que estende a Integração contínua (CI), focando-se na automatização do processo de lançamento do *software*. O objetivo da (CD) é garantir que o código esteja sempre disponível para ser implementado em produção. Todo o processo, desde a integração, teste e construção, até à preparação para implantação é automatizado e inclui uma ampla gama de testes automatizados (funcionais, de integração, de aceitação, etc.) assegurando a qualidade. Desta forma, promove-se a comunicação entre as equipas de desenvolvimento e operações, mantendo o código continuamente validado e reduzindo a probabilidade de problemas ou vulnerabilidades atingirem o ambiente de produção [21].

3.5 DevSecOps

O paradigma de DevSecOps tem como objetivo integrar a segurança em cada etapa do processo de desenvolvimento de *software*, garantindo um produto final mais robusto e seguro. A Figura 3.4 descreve um processo típico de DevSecOps, que inclui dez fases distintas:

- **Planeamento:** Fase inicial que antecede o desenvolvimento do *software*, em que são identificadas as necessidades e definidos os objetivos da entrega. A segurança deve ser considerada desde o início, com identificação e mitigação de riscos reduzindo a superfície de ataque. A inclusão de medidas de segurança nesta fase é crucial para a proteção global do projeto.
- **Codificação:** Os programadores implementam os requisitos utilizando uma linguagem de programação, submetendo frequentemente o seu código a um repositório dedicado. Nesta fase, verifica-se a qualidade do código e a conformidade com práticas de segurança ao longo do desenvolvimento.
- **Teste:** Um programador ou uma equipa é responsável por testar e validar o código, identificando e reportando problemas gerais e de segurança. A arquitetura DevSecOps combina métodos de teste e ferramentas automatizadas para garantir segurança contínua.
- **Inspeção do Ambiente:** Antes do lançamento do *software* é realizada uma análise de segurança do ambiente de produção fornecido pela empresa. Um ambiente inseguro compromete a segurança do sistema. Esta fase visa identificar e resolver potenciais riscos de segurança no ambiente de produção, garantindo a segurança global do ambiente e dos equipamentos e reforçando a fiabilidade do *software* a ser entregue.
- **Lançamento:** Pode ser revisto manualmente pelo proprietário ou realizado automaticamente através do *pipeline* DevOps. Os pacotes de *software* podem ser adulterados ou sofrer engenharia reversa após o lançamento. As organizações devem adotar medidas preventivas para garantir que os utilizadores acedem a versões fidedignas antes do lançamento oficial.
- **Monitorização:** Estabelece uma monitorização de segurança eficaz em todas as fases do serviço em produção. Permite *feedback* e resposta a problemas em tempo real. A equipa de operações toma medidas adequadas com base na gravidade e origem do problema, formando um processo de reparação em ciclo fechado. A arquitetura DevSecOps reforça a segurança operacional, incorporando práticas automatizadas (eventualmente baseadas em tecnologias como *Policy as Code* [22]) e estabelecendo mecanismos de *feedback* que aceleram a deteção, localização e correção de problemas. Assim sendo, a monitorização pode alargar-se ao domínio da auditoria de conformidade da infraestrutura, dado o seu papel vital para o acompanhamento, aplicação e manutenção de quaisquer políticas de segurança, seja no domínio *cloud-native* ou mesmo ao nível do dispositivo. Refira-se ainda que, quando apropriadamente implementados, estes mecanismos são ainda determinantes para suporte de atividades forenses (como a análise *post-mortem* de incidentes).
- **Deteção de Intrusão:** Deteta atividades suspeitas no tráfego de rede, emitindo alertas e analisando o sistema para identificar violações. Sistemas comuns incluem deteção profunda e horizontal, com análise do tráfego de todos os dispositivos na rede.

- **Resposta de Defesa Ativa:** Após identificar comportamentos intrusivos, o sistema bloqueia as intrusões e alerta as equipas de operações sobre as medidas necessárias. Um sistema com defesa ativa e eficaz protege contra ameaças internas e externas.
- **Previsão e Resposta:** Analisa comportamentos anteriores para identificar padrões de intrusão, recolhendo e analisando informações sobre vulnerabilidades. Após a implementação do *software*, muitas vulnerabilidades de segurança são descobertas em produção. As equipas devem acompanhar fontes atualizadas da divulgação de vulnerabilidades, realizar autoinspeções e implementar correções antes que essas falhas sejam exploradas.
- **Ajustamento:** Trata os problemas identificados na fase de deteção de intrusão, ajustando os métodos de deteção e resposta. Inclui melhorias contínuas nas soluções e ferramentas de segurança, bem como ajustes na lógica de defesa profunda e horizontal.

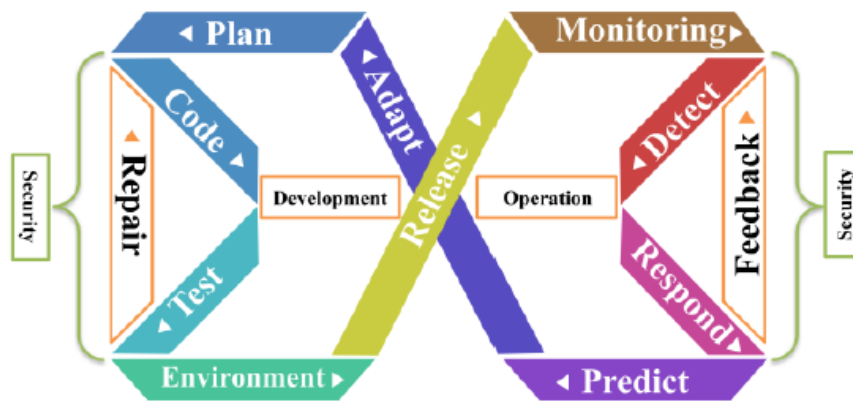


Figura 3.4: *DevSecOps - Architecture Design* [23]

Estas dez fases permitem que as equipas de desenvolvimento e operações compreendam o estado operacional do produto, corrijam os problemas existentes e gerem novos requisitos para a fase de planeamento, iniciando um novo ciclo de iteração [23].

3.6 Ferramentas e tecnologias para IaC

Nesta secção abordaremos as principais ferramentas e tecnologias de IaC, nomeadamente *Terraform*, *Ansible*, *SonarQube* e *Checkov*.

3.6.1 *Terraform*

O *Terraform* é uma ferramenta da empresa *HashiCorp* para a implementação de infraestrutura como código, utilizando uma linguagem declarativa que permite criar, alterar e manter recursos locais e na nuvem de forma segura e eficiente.

Permite definir recursos através de ficheiros de configuração, sendo capaz de gerir componentes de baixo nível (como computação, armazenamento e rede), bem como componentes de alto nível, como entradas de Domain Name System (DNS) e recursos de Software as a Service (SaaS) [24].

Estrutura de configuração do *Terraform*

Os ficheiros de configuração do *Terraform* devem ter a seguinte extensão (*.tf). Os nomes dos ficheiros não seguem uma nomenclatura obrigatória, embora exista uma convenção não vinculativa de nomes, descritos abaixo [25]:

- *main.tf*: Ficheiro principal, contém os blocos de recursos que definem a infraestrutura a ser criada.
- *variables.tf*: Define as variáveis de entrada, especificando os respetivos tipos, descrições e/ou valores padrão.
- *provider.tf*: Declara os provedores necessários para configurar e interagir com sistemas externos ou plataformas de nuvem.
- *output.tf* : Lista as saídas que devem ser geradas após a conclusão bem-sucedida da operação “*apply*”.
- **.tfvars*: Especifica os valores concretos atribuídos às variáveis de ambiente.

Workflow *Terraform*

Os cinco comandos principais no fluxo de trabalho do *Terraform* são:

- *Terraform init* - Prepara a pasta de trabalho, realizando o *download* e a instalação dos *providers* especificados.
- *Terraform validate* - Verifica se a configuração descrita nos ficheiros é válida.
- *Terraform plan* - Descreve as alterações que serão realizadas para alinhar a infraestrutura com a configuração definida.
- *Terraform apply* - Aplica, criando ou atualizando a infraestrutura, conforme descrito no *Terraform plan*.
- *Terraform destroy* - Elimina a infraestrutura criada anteriormente.

A Figura 3.5 ilustra o fluxo de trabalho do *Terraform* com base em cada uma destas fases.

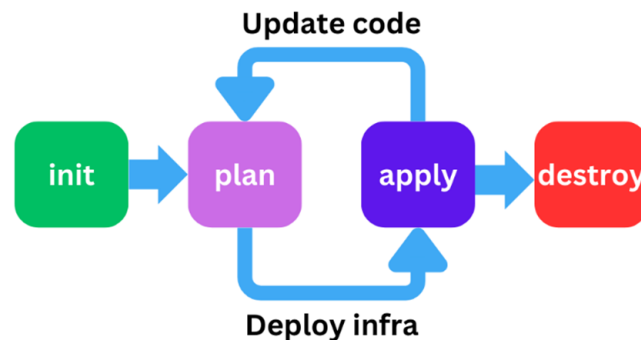


Figura 3.5: *Terraform - WorkFlow* [24]

Terraform state

Como método de gestão, o *Terraform* armazena o estado da infraestrutura e das configurações num ficheiro denominado *terraform.tfstate*. Este ficheiro permite ao *Terraform* determinar quais as alterações a efetuar. Antes de qualquer operação, o *Terraform* compara o novo ficheiro de configuração com o estado atual, definido no ficheiro *.tfstate*, antes de atualizar a infraestrutura real [26].

Os ficheiros de estado não devem ter versões, nem devem ser armazenados localmente, uma vez que tal pode expô-los a alterações indevidas e comprometer a segurança e a integridade da infraestrutura. Uma forma segura de os gerir é o uso de um *backend* remoto.

Desta forma, os ficheiros de estado não ficam armazenados localmente, o que aumenta:

- *Segurança* - Os ficheiros de estado podem armazenar detalhes confidenciais, como chaves, *tokens* e senhas.
- *Colaboração* - Em ambientes colaborativos, a gestão centralizada do estado facilita o trabalho em equipa.

O ficheiro *terraform.tfstate* é criado automaticamente pelo *Terraform* aquando da primeira execução do comando *terraform apply*. Este ficheiro não deve ser editado manualmente, pois pode comprometer o funcionamento do sistema.

O ficheiro *terraform.tfstate.backup* é gerado no mesmo diretório do ficheiro e é atualizado sempre que o estado é alterado. Este ficheiro pode ser utilizado para restaurar uma versão anterior do estado, em caso de falha.

Recomendações para gerir os *backups* de estado do *Terraform*:

- Armazenar o ficheiro *terraform.tfstate.backup* num local diferente do ficheiro *terraform.tfstate*.

- Implementar uma metodologia para realizar cópias de segurança regulares do ficheiro *terraform.tfstate.backup*, garantindo sempre uma versão recente disponível.
- Testar periodicamente os *backups* criados, assegurando que os mesmos podem ser restaurados com sucesso e de forma funcional [27].

Workspaces Terraform

O *Terraform Workspaces* permite criar e gerir múltiplos estados dentro de uma única configuração. Isto é particularmente útil e eficaz quando se trabalha com vários ambientes, tais como desenvolvimento, preparação ou testes e produção.

Para a criação de três ambientes – *prod*, *qa* e *dev* – devem ser criados três ficheiros *.tfvars* com nomes claros [28]:

- *variáveis-dev.tfvars*
- *variáveis-qa.tfvars*
- *variáveis-prod.tfvars*

A gestão de múltiplos ambientes é geralmente realizada utilizando *workspaces* no *Terraform*. Dependendo do *workspace* em uso, o ficheiro *.tfvars* apropriado será aplicado. A Figura 3.6 descreve os vários *Terraform Workspaces* possíveis.



Figura 3.6: Terraform - Workspaces [29]

3.6.2 Ansible

O *Ansible* é uma ferramenta baseada em *Python* que funciona através da linha de comandos e foi desenvolvida em *open-source*, sendo atualmente gerida pela *Red Hat*. Automatiza o aprovisionamento, a gestão de configurações, a instalação de software e a orquestração.

Com o *Ansible* podemos configurar e atualizar sistemas, instalar *software*, automatizar tarefas diárias e aprovisionar infraestrutura [30].

Estrutura do Ansible

No ficheiro *Playbook*, são descritas as operações a executar. O *Inventory* é o ficheiro onde estão definidos os *hosts* nos quais serão realizadas as tarefas enumeradas no *Playbook*. Os *Modules* são responsáveis pela execução das ações descritas no *Playbook* (por exemplo, instalar pacotes). Os *Plugins* acrescentam funcionalidades adicionais, como relatórios ou *logs*. A comunicação com os *hosts* definidos no *Inventory* é realizada através de uma ligação segura (via Secure Socket Shell (SSH)), onde são aplicadas as operações especificadas no *Playbook*.

A Figura 3.7 mostra a estrutura de funcionamento do *Ansible*, indicando os principais componentes e a comunicação entre eles.

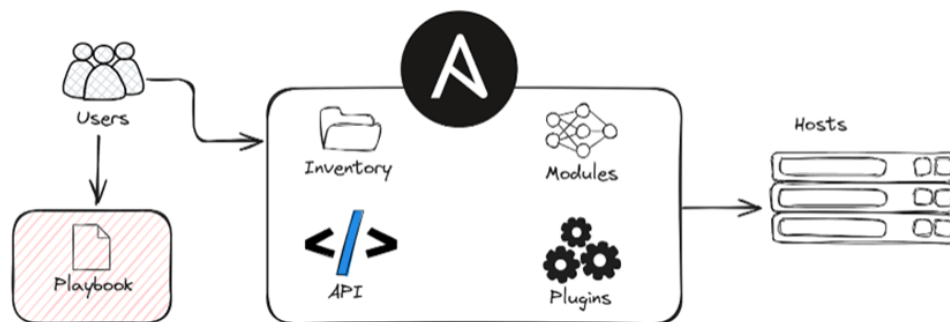


Figura 3.7: Estrutura do *Ansible* [31]

O *Playbook* é um ficheiro no formato YAML Ain't Markup Language (YAML) (um formato de ficheiro de texto simples e intuitivo para humanos), no qual se define um conjunto de tarefas a executar nos servidores (*hosts*). Descreve-se o que deve ser realizado, como instalar pacotes, configurar serviços ou gerir ficheiros.

O *Inventory* contém uma lista de servidores (*hosts*) que o *Ansible* gere, estando normalmente definido num ficheiro ou gerado de forma dinâmica (por exemplo, a partir de *APIs*).

O *Ansible* pode ser controlado através da *API*, permitindo a integração com outras ferramentas e plataformas.

Componentes do Ansible

Estes componentes desempenham uma função no processo de configuração, gestão e automação de infraestrutura:

- *Modules* - São *scripts* que executam tarefas específicas nos *hosts*, como copiar ficheiros, reiniciar serviços ou configurar *firewalls*. O *Ansible* possui muitos módulos integrados e, também, é possível criar módulos personalizados.
- *Plugins* - Adicionam funcionalidades extras ao *Ansible*, como manipulação de conexões, *callbacks* personalizados e *logs*.

- *Hosts* - Servidores ou dispositivos de rede que o *Ansible* gere. As tarefas definidas nos *playbooks* são executadas nesses *hosts*.

Descrição do funcionamento

O método de funcionamento do *Ansible* baseia-se na gestão de infraestruturas, permitindo a automatização e a simplificação de tarefas de administração, o que garante eficiência e consistência nos ambientes geridos.

A Figura 3.8 ilustra o funcionamento do *Ansible* e os seus principais componentes.

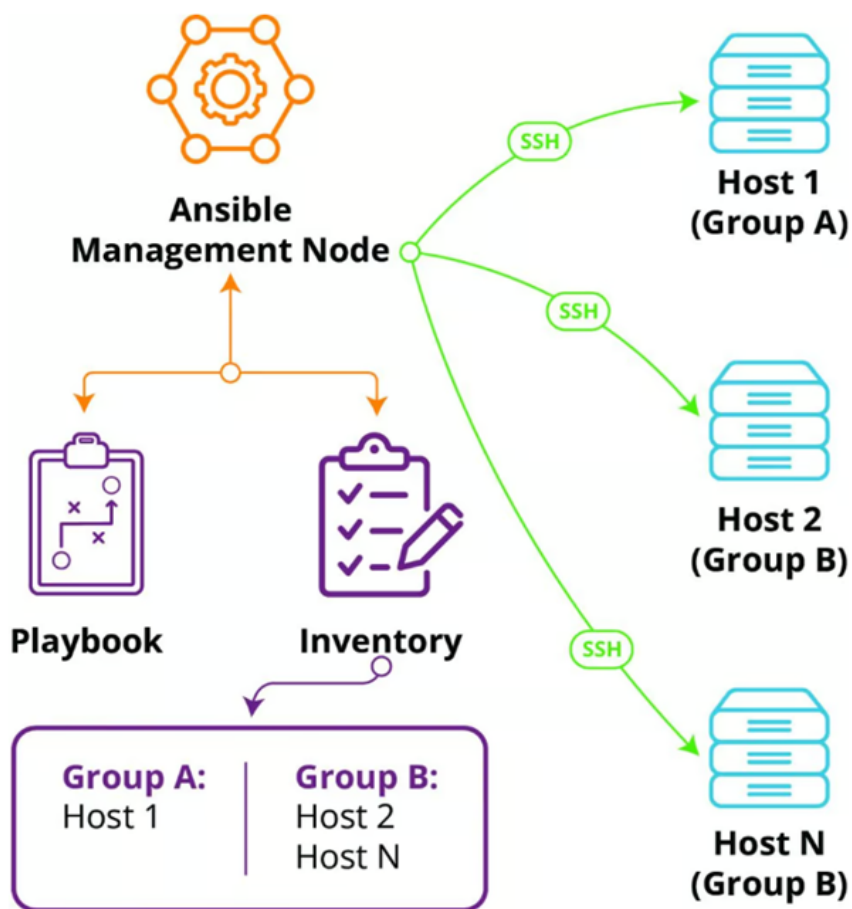


Figura 3.8: *WorkFlow* do *Ansible* [32]

Os principais elementos do *Ansible* desempenham papéis essenciais na automatização e gestão da infraestrutura. Abaixo descrevem-se os componentes que integram o seu funcionamento:

- **Nó de gestão - (*Management Node*)**
 - Ponto central de controlo do *Ansible*. Executa os *playbooks*, gere a execução das tarefas e estabelece ligação com os *hosts* do destino.
 - Não é necessária a instalação de agentes ou de software adicional nos *hosts* geridos, o que simplifica a sua configuração.

- *Playbooks*
 - Ficheiros de configuração e automação escritos no formato YAML.
 - Contêm as instruções a executar nos *hosts*, como a instalação de pacotes ou a configuração de serviços.
- Inventário
 - Ficheiro que enumera os *hosts* geridos. Estes podem ser agrupados em categorias para organizar as tarefas a executar.
- Conexão SSH
 - O *Ansible* utiliza o SSH como método de ligação aos *hosts* para executar os módulos remotamente, eliminando a necessidade de *software* adicional nos mesmos.
- Execução de tarefas
 - O *Ansible* executa as tarefas descritas no *playbook* nos *hosts* definidos no ficheiro do inventário. Após a execução, os módulos são automaticamente removidos.

3.6.3 SonarQube

O *SonarQube* é uma ferramenta de análise de código estática que tem como objetivo avaliar a conformidade do código-fonte, verificando se este está de acordo com um conjunto de regras prédefinidas. A ferramenta identifica problemas no código, como *bugs*, problemas de manutenção (*code smells*) e vulnerabilidades de segurança, atribuindo métricas como a complexidade do código e o número de linhas de código. Caso o código viole uma regra, o *SonarQube* regista uma *issue*, para que seja analisada e posteriormente resolvida [33].

É utilizado por mais de 85.000 organizações e tem suporte para análise de cerca de 25 linguagens de programação. Está preparado para classificar as violações em três tipos principais: *Bugs*, Vulnerabilidades e *Code Smells*. A ferramenta possui um sistema para atribuição de níveis de severidade [34]:

Classificações de severidade atribuídas pelo *SonarQube*:

- *Blocker* - Problemas críticos que devem ser corrigidos imediatamente.
- *Critical* - Problemas de alta prioridade que precisam de atenção urgente.
- *Major* - Problemas relevantes, mas que podem ser resolvidos de forma menos urgente.
- *Minor* - Problemas de baixa gravidade, que podem ser corrigidos mais tarde.
- *Info* - Questões informativas ou sugestões de melhorias que não afetam diretamente o funcionamento do código.

Problemas classificados como *Blocker* e *Minor* tendem a ser corrigidos mais rapidamente do que os classificados como *Critical* ou *Major*, o que é contraditório, tendo em conta o impacto potencial dos problemas mais graves [34].

Utilização do *SonarQube* pelos programadores

Análise de práticas e comportamentos relacionados com o uso do *SonarQube* pelos programadores [34]:

- Mais de 80% dos programadores consideram que as mensagens de alerta reportadas pelo *SonarQube* são relevantes para melhorar o *design* e a implementação de *software*.
- Os programadores geralmente corrigem os problemas reportados durante a implementação de novas funcionalidades ou correção de *bugs*, em vez de criar tarefas ou *branches* específicas para o efeito.
- Apenas 22% dos programadores rejeitam *pull requests* com base nas violações reportadas.

Constrangimentos na correção de problemas reportados pelo *SonarQube*:

- Apenas 8,76% das violações identificadas são corrigidas.
- O tempo mediano para corrigir um problema é de 18,99 dias, embora 30% das correções demorem mais de um ano.
- Fatores incluem a priorização de outras atividades, configurações inadequadas das regras e a falta de sensibilização quanto à importância de algumas violações.

Funcionamento do *SonarQube*

Nesta secção, apresenta-se o funcionamento do sistema de análise estática do código, com a integração entre o *SonarQube* e o *GitHub*, fundamentais para garantir a qualidade e a segurança no desenvolvimento de *software*.

A Figura 3.9 apresenta o sistema de automatização do processo de análise estática de código durante o ciclo de desenvolvimento, com a integração do *SonarQube* e o *GitHub* assegurando a consistência do sistema.

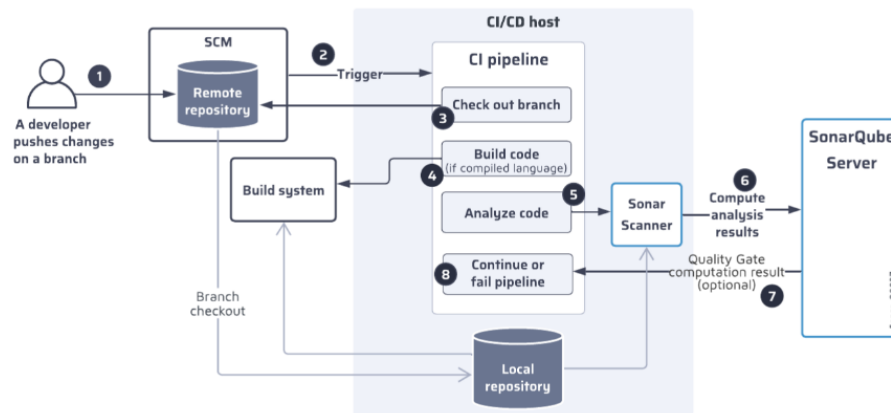


Figura 3.9: Funcionamento do *SonarQube* [35]

Como ilustrado na Figura 3.9, este processo inclui vários componentes:

- *Developer* (Programador)
 - O processo inicia-se com o programador a desenvolver o código e a enviá-lo para o repositório remoto no *GitHub*.
- *Repository*
 - As alterações feitas no código são armazenadas no repositório do *GitHub*.
 - O repositório aciona um gatilho (*trigger*), que pode ser um *webhook* ou uma integração configurada para executar determinadas ações automaticamente.
- *GitHub*
 - O *GitHub* atua como um ponto central. Assim que recebe o gatilho do repositório, aciona o *Sonar Scanner*.
 - A ativação ocorre através de uma integração previamente configurada entre o *GitHub* e o ambiente de análise do código.
- *Sonar Scanner*
 - O *Sonar Scanner* executa a análise estática do código, verificando a existência de possíveis problemas no código:
 - * Erros de sintaxe.
 - * Violações de boas práticas.
 - * Vulnerabilidades de segurança.
 - * Código duplicado e outros problemas de qualidade.
- *SonarQube*

- Após a análise concluída pelo *Sonar Scanner*, os resultados são enviados para o servidor *SonarQube*.
- O *SonarQube* armazena e apresenta os resultados da análise numa interface gráfica, onde podem ser analisados pelos programadores.
- Estes relatórios ajudam a identificar áreas de melhoria no código antes da sua implementação em produção.

3.6.4 Checkov

O *Checkov* é uma ferramenta de análise estática de segurança para infraestrutura como código IaC focada em *Terraform*, mantida pela *BridgeCrew* [36]. Foi concebida para identificar más configurações e vulnerabilidades em *scripts Terraform*. A principal vantagem na utilização do *Checkov* é o seu abrangente repositório de políticas de segurança integradas, que ultrapassa as mil regras, permitindo assim uma verificação alargada de boas práticas e potenciais vulnerabilidades. Destaca-se ainda pelas elevadas taxas de precisão e pela baixa incidência de falsos positivos [37].

3.7 Outras ferramentas e iniciativas relevantes

3.7.1 Log4j

O *Log4j* [38] é uma biblioteca de registos de *logs* baseada em *Java*, desenvolvida pela *Apache Software Foundation* [39]. É utilizada por aplicações *Java* para registo de mensagens de erro, monitorização do funcionamento do sistema, assim como *debugging* e análise de eventos.[40]

3.7.2 CVE-2021-44228 (Log4Shell)

O CVE-2021-44228 [41], é uma vulnerabilidade crítica de execução remota de código, descoberta em dezembro de 2021, na biblioteca *Log4j* até à versão 2.14.1. A falha permite que um atacante injete uma *string* maliciosa nos campos de *log* que, ao serem processados pela aplicação, resultam na execução de código arbitrário controlado pelo atacante [40].

Como funciona o ataque

1. O atacante injeta uma **string maliciosa** num campo que será registado por *Log4j* (por exemplo, no cabeçalho *User-Agent*, mensagens de erro ou entradas do utilizador).
2. O *Log4j* interpreta essa *string* e realiza um *lookup remoto via JNDI*.

3. O código existente no servidor do atacante é descarregado e executado no sistema da vítima.

Impacto

- Severidade máxima: Common Vulnerability Scoring System (CVSS) 10.0 [42]
- Afeta milhões de dispositivos e aplicações baseadas em *Java* (incluindo *Minicraft*, *iCloud*, Amazon Web Services (AWS), *Apache Struts*, entre outros).
- Facilita a execução de malware e o acesso completo a servidores vulneráveis.

Medidas de mitigação

- Atualizar para versões seguras do *Log4j* ($\geq 2.16.0$).
- Desativar os JNDI lookups sempre que possível.
- Implementar validações robustas de entrada e saída de dados.
- Utilizar uma arquitetura de rede com controlo de acesso e segmentação.
- Monitorizar e aplicar estratégias de defesa em profundidade [40].

3.7.3 Web application firewall

Uma Web Application Firewall (WAF) é uma ferramenta de segurança concebida para monitorizar, filtrar e bloquear o tráfego de dados entre utilizadores e aplicações *web*. Atua como uma barreira que inspeciona as comunicações na camada de aplicação, analisando os pedidos com base em regras predefinidas para detetar e bloquear tráfego malicioso.

Atua como intermediário entre os utilizadores e a aplicação, examinando cada pedido para identificar tentativas de exploração de vulnerabilidades, como injeções de código, ataques de Denial of Service (DoS) ou tentativas de acesso não autorizado, protegendo assim as aplicações e serviços *web*.

Ao proteger os sistemas contra ameaças direcionadas às aplicações, a WAF contribui para a preservação da integridade, confidencialidade e disponibilidade dos dados e serviços disponibilizados online [43].

3.8 Sumário

Este capítulo descreve os fundamentos de IaC, destacando a importância da gestão de identidades (IAM) para garantir acessos seguros em ambientes distribuí-

dos, especialmente na nuvem.

As práticas de DevOps promovem a integração entre as áreas de desenvolvimento e operações, enquanto os mecanismos de SAST e a identificação de diferentes tipos de *smells* — *code smells*, *test smells* e *security smells* — indicam padrões que podem comprometer a qualidade e a segurança do software.

Descrevem-se ainda as práticas de integração e entrega contínuas (CI/CD), fundamentais para a automação dos processos de construção, teste e *deployment*.

O modelo DevSecOps é introduzido como uma evolução do DevOps, integrando a segurança em todas as fases do ciclo de vida de desenvolvimento.

São também apresentadas ferramentas essenciais para o provisionamento, gestão e automação de infraestrutura, nomeadamente o *Terraform*, o *Ansible*, o *SonarQube* e o *Checkov*.

Finalmente, abordam-se temas, como a biblioteca *Log4j*, a vulnerabilidade CVE-2021-44228 e a utilização de WAF como mecanismo de proteção de aplicações *web*.

Capítulo 4

Análise do estado da arte

Neste capítulo, são apresentados os resultados de uma análise do estado da arte baseada em trabalhos de investigação centrados no tema de IaC.

Na Secção 4.1 é apresentada a metodologia seguida para a preparação deste estado da arte. A síntese individual de cada um dos artigos científicos analisados é apresentada na Secção 4.2. A Secção 4.3 apresenta uma discussão sobre as principais conclusões extraídas da análise, considerando cinco aspetos distintos: automação e segurança, deteção e mitigação de vulnerabilidades, gestão de segredos, formação e conscientização. Finalmente, a Secção 4.4 conclui este capítulo.

4.1 Metodologia adotada

Nesta secção, é descrita a metodologia de pesquisa utilizada, os motores de pesquisa usados e o processo de seleção dos artigos que compõem a análise do estado da arte, até à identificação das 31 publicações científicas que foram analisadas em maior detalhe.

4.1.1 Metodologia para a revisão sistemática

Numa primeira fase, foi utilizado o motor de pesquisa *Google Scholar* [44], com as palavras-chave relacionadas com IaC, como pesquisa inicial para obter uma visão geral sobre este tema. O resultado inicial foi de 50 artigos, que foram lidos mesmo que o seu conteúdo não estivesse totalmente alinhado com o tema em análise. Este constituiu o primeiro contacto com o tema.

De seguida, foram adotados passos mais sistematizados. A aplicação *Rayyan* [45] foi utilizada para recolha e classificação. As plataformas *IEEE Explore* [46], *ScienceDirect* [47] e *Wiley* [48] foram utilizadas para realizar pesquisas mais detalhadas e aprofundadas sobre IaC.

As pesquisas na *IEEE Explore* e na *ScienceDirect* foram realizadas em novembro de 2024 e as da *Wiley* em dezembro de 2024.

A Tabela 4.1 apresenta o conjunto de palavras-chave utilizadas na realização do processo de pesquisa.

Tabela 4.1: Palavras-chave usadas no processo de pesquisa

	Palavras-chave
1	(Infrastructure as code OR IAC)
2	(Infrastructure as code OR IAC) AND Security
3	(Infrastructure as code OR IAC) AND (Ansible OR Terraform OR Chef)
4	(Infrastructure as code OR IAC) AND (Security Code test)
5	(Infrastructure as code OR IAC) AND Vulnerability
6	(Infrastructure as code OR IAC) AND (Policy as code)

A Tabela 4.2 indica a quantidade total de artigos encontrados em cada plataforma, com base nas palavras-chave anteriormente referidas.

Tabela 4.2: Quantidade de artigos por plataforma e por palavra-chave

Palavras-Chave	IEEE Explore	ScienceDirect	Wiley
1	656	1000	7
2	108	126	5
3	32	32	0
4	13	84	4
5	24	70	1
6	15	77	3
	848	1389	20

Os critérios de inclusão dos artigos acima quantificados foram os seguintes:

- Por data:
 - artigos publicados entre 2019 e 2024.
- Na plataforma de pesquisa científica *ScienceDirect*
 - Escolha dentro de *Subject areas*, por *Computer Science*
- Obras escritas em inglês
- Artigos com ficheiro *pdf* disponível
- Artigos que analisam o tema

Os critérios de exclusão aplicados foram os seguintes:

- Artigos sem ficheiro *pdf* disponível
- Artigos que não analisam o tema principal

4.1.2 Identificação de trabalhos relevantes

A Figura 4.1 apresenta o diagrama de *PRISMA* utilizado no processo de revisão de literatura. Durante a identificação, foram encontrados 2.257 artigos nas três plataformas. Destes, foram removidos 1.298 artigos duplicados. Na primeira fase de triagem, foram analisados 959 artigos com base em critérios preliminares de inclusão, como título e resumo. Destes, 14 foram excluídos por não estarem disponíveis. Na segunda etapa, os 945 artigos restantes foram avaliados, considerando critérios detalhados de inclusão e exclusão. Posteriormente, 914 artigos foram eliminados por não serem considerados relevantes para o estudo. Por fim, na etapa de inclusão, foram selecionados 31 artigos, que passaram por uma leitura integral e foram considerados relevantes para a revisão sistemática.

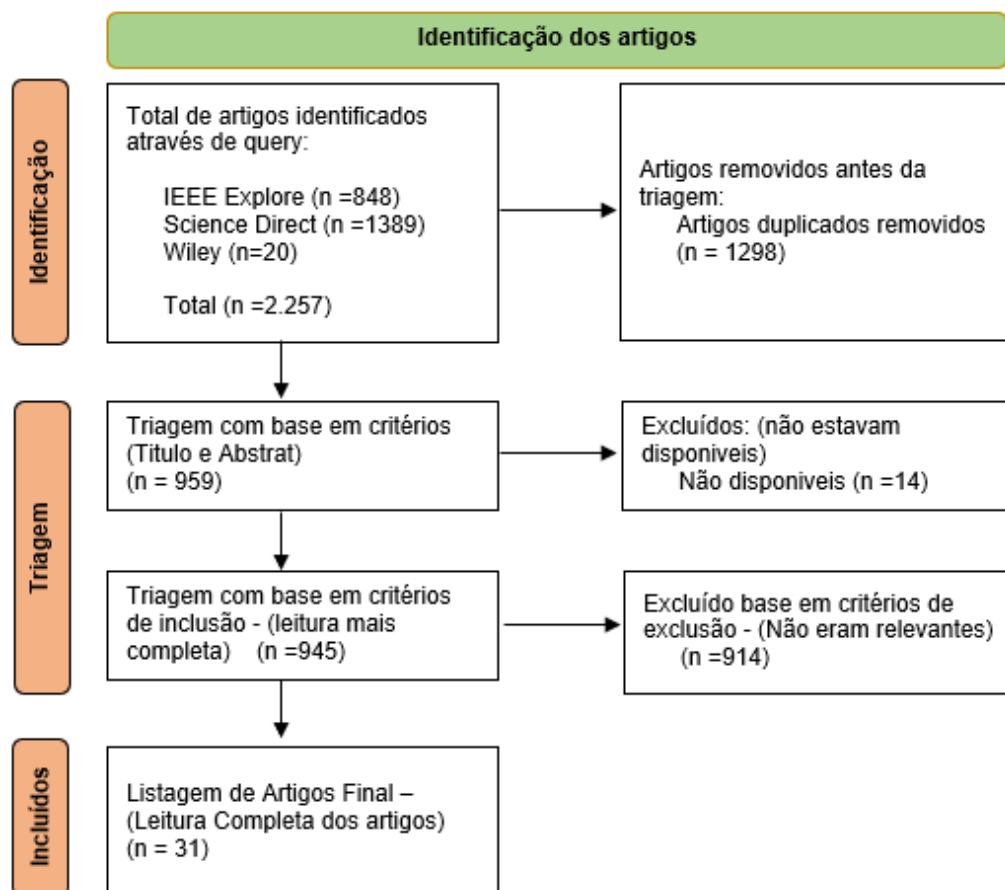


Figura 4.1: Diagrama de *PRISMA* do processo de revisão de literatura [49]

4.1.3 Estratégia para a seleção de trabalhos relevantes

Para a seleção dos artigos, foram considerados três critérios principais:

- Título.
- Leitura do resumo (*abstract*).
- Leitura completa do artigo.

Foram utilizadas as funcionalidades da plataforma *Rayyan* para auxiliar na categorização dos artigos, através dos filtros disponíveis:

- *All articles*
- *Undecided*
- *Excluded*
- *Maybe*
- *Included*

Esta plataforma também possui uma funcionalidade de detecção de duplicados, que facilitou a eliminação dos 1.298 artigos duplicados. A Figura 4.2 apresenta a distribuição dos artigos selecionados por ano de publicação, totalizando 31 artigos incluídos na análise final.

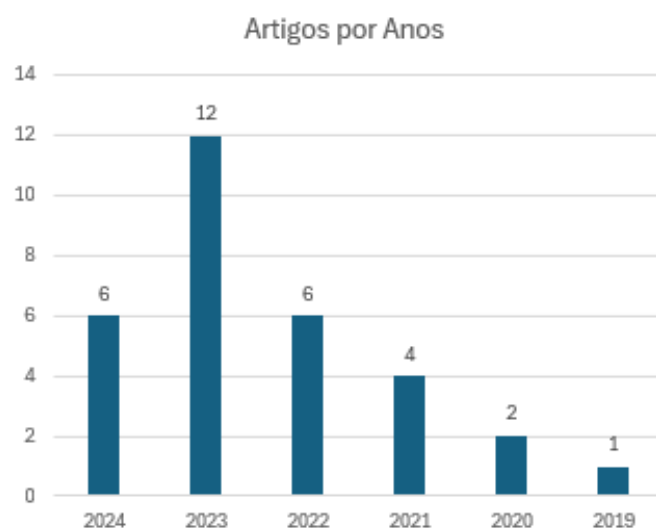


Figura 4.2: Artigos selecionados por ano de publicação

4.2 Análise individualizada dos artigos

As publicações selecionadas por meio do processo anteriormente descrito foram depois analisadas individualmente e em detalhe, apresentando-se de seguida uma síntese de cada uma dessas publicações.

Chiari et al. [2] apresenta a DevOps Modeling Language (DOML), uma nova forma de estrutura da infraestrutura como código IaC, com o objetivo de facilitar a configuração e a operacionalização das infraestruturas de Tecnologia da Informação (TI). A sua estrutura é composta por três camadas: camada de aplicação, camada de envelope de infraestrutura e camada de implantação da infraestrutura. O DOML Model Checker (DMC) corrige discrepâncias ou erros antes da fase de implantação, resultando num aumento de qualidade e segurança dos projetos de infraestrutura. O ICG permite transformar modelos em código IaC executável para *Terraform* ou *Ansible*, reduzindo assim o esforço manual. A avaliação empírica comparou a DOML com outras linguagens de IaC existentes, que incluem *Terraform* e *Cloudify*, utilizando instâncias de implementação prática no caso do sistema *WordPress*. O DOML pode transformar-se numa solução para abordar a complexidade em torno da implementação de práticas de IaC em ambientes DevOps.

Alghawli et al. [50] descreve a necessidade do mercado dispor de soluções que implementem e garantam a fiabilidade na infraestrutura, a segurança das aplicações *web* e a existência de um mecanismo que detete vulnerabilidades em todas as fases do desenvolvimento do produto. É destacada a implementação de uma cultura de DevOps, pois essa cultura reduz o tempo de entrega, teste e implementação de aplicações *web*. O processo DevSecOps é utilizado para garantir a segurança, tendo a capacidade de detetar vulnerabilidades automaticamente e cumprir as regras de segurança em todo o processo. Foi elaborado um algoritmo para efetuar uma avaliação quantitativa de risco, automatizada e baseada em dados estatísticos e na metodologia FAIR, com o objetivo de calcular um valor quantitativo de risco, permitindo assim uma previsão mais eficiente dos custos associados à segurança. Numa avaliação de riscos de segurança, os dados Confidentiality, Integrity, and Availability (CIA) - confidencialidade, integridade e disponibilidade - são os principais parâmetros a ter em consideração. Este valor do risco é calculado com base na probabilidade dos eventos que podem ameaçar os três pilares do CIA. O *Terraform* é usado para implementar a automatização da infraestrutura, enquanto o *Jenkins* automatiza o processo de CI/CD integrado com o *SonarQube*, que efetua a deteção de vulnerabilidades no código.

Ghosh et al. [7] analisa como as organizações estão a usar, cada vez mais, uma estratégia *multi-cloud*, destacando os seus benefícios, mas também os seus desafios, como a complexidade acrescida e o aumento de custos devido à gestão e à transferência de dados. Como forma de mitigar estes desafios, o artigo propõe um "*wrapper*" personalizado que opera sobre o *Terraform*, tendo como objetivo simplificar a orquestração numa infraestrutura *multi-cloud*, através de um único ficheiro de configuração. Foi descrito um cenário das tecnologias envolvidas, com orquestração de *cloud*, princípios de IaC e o funcionamento do *Terraform*. Os resultados revelam diferenças de desempenho entre os fornecedores de *cloud*. A *Azure*

apresentou os maiores tempos de provisionamento e desaprovisionamento, embora o consumo de Central Processing Unit (CPU) e Random Access Memory (RAM) tenha sido semelhante ao da plataforma AWS. O amplo consenso no uso de *Terraform*, tanto em termos de suporte da comunidade como pela sua ampla utilização, reforça a sua aplicabilidade em implementações reais, assegurando de forma consistente a sincronização entre a infraestrutura real e a configuração planejada.

Bessghaier et al. [18] investiga os *code smells* nos ficheiros de IaC na ferramenta *Puppet*. Foram analisados 82 projetos *open-source* e a investigação centrou-se em 12 tipos de *smells* IaC, tanto a nível de implementação como de *design*. Foram identificados padrões que aparecem com frequência em conjunto com diferentes *code smells*, presentes simultaneamente nos mesmos ficheiros ou módulos de código de IaC. Devido a essa coocorrência, alguns *smells* estão frequentemente associados, como no caso de Deficient Data Encapsulation (DDE) e Dense Declarative Structure (DDS). Isto sugere que a presença de um *smell* pode ser indicativa da existência de outros. A implementação de medidas e mecanismos para a deteção, prevenção e correção de "*smells*" é essencial para o desenvolvimento de IaC sustentável e de qualidade. Deve ser desenvolvida a criação de soluções automatizadas para correção de *smells* e, posteriormente, estabelecer uma relação entre cada tipo de *smell* e as mudanças no código.

Abbas et al. [6] descreve que a gestão de infraestruturas é realizada através de processos manuais, sem qualquer tipo de automação ou sistematização, o que desperdiça muito tempo e por ser realizado por humanos, existe uma grande probabilidade de criação de erros e ineficiências. Foi realizado um estudo de caso para comprovar os benefícios da utilização de ferramentas de IaC. Os valores recolhidos mostram que a redução do tempo de implementação passou de dias para minutos e que houve um aumento da reutilização de recursos. O Mean Time to Recovery (MTTR) passou de 6 horas para 30 minutos, com os custos a sofrerem, também, uma redução de \$50.000 para \$35.000 por mês. Com o uso de IaC, aumentaram a conformidade e a segurança, assim como diminuíram os problemas de configuração e de vulnerabilidades. O documento termina sugerindo que o futuro da IaC passa pela integração com as tecnologias emergentes, incluindo *edge computing*, Internet of Things (IoT), inteligência artificial e *machine learning*, destacando o potencial da infraestrutura imutável e ambientes *multi-cloud*.

Paul et al. [51] aborda o processo de automação da conformidade em ambientes de computação na nuvem, utilizando o Open Policy Agent (OPA) na plataforma AWS. É realizada uma análise aos ficheiros *CloudFormation Templates*, utilizando uma abordagem DevSecOps, que incorpora considerações de segurança e conformidade no *pipeline* de desenvolvimento CI/CD. Caso sejam identificadas vulnerabilidades, estas são corrigidas. Esta solução, baseada no OPA, utiliza um mecanismo de políticas de código aberto que permite desenvolver e validar políticas, recorrendo a uma linguagem declarativa de alto nível chamada *Rego*. Desta forma, é possível avaliar políticas de forma dinâmica, com base no processamento de pedidos e dados em tempo real. Além disso, as políticas são tratadas como objetos de código, juntamente com as configurações de infraestrutura,

sendo adequadas para orquestração e sistemas contentorizados, em *Kubernetes*. A automatização da conformidade contribui para melhorar a colaboração, reduzir custos e aumentar a agilidade.

Begoug et al. [8] descreve como as ferramentas de IaC, como *Ansible*, *Terraform* e *AWS CloudFormation* se tornaram as plataformas que os programadores mais utilizam na gestão dos recursos de infraestrutura de forma automatizada. Foram identificadas vantagens, nomeadamente na forma como são implementados o desenvolvimento do *software*, o controlo de versões e os testes automatizados. Como desvantagens, foram apontadas a complexidade do código e a necessidade de formação contínua dos programadores. Foi realizado um estudo sobre o *Stack Overflow*, abrangendo mais de 100 mil *posts* entre 2011 e 2022. Este foi dividido em sete tópicos principais, utilizando a técnica de Latent Dirichlet Allocation (LDA). Foram analisadas, como pontos positivos, as configurações de servidores e a gestão de ficheiros. Em contraponto, áreas como *pipelines* de desenvolvimento de *software* foram identificados como problemáticas. O *Terraform* e *Ansible* registaram um crescimento significativo no período entre o ano de 2018 e 2019. As ferramentas de IaC devem ser continuamente melhoradas e deve ser disponibilizada documentação mais clara e de melhor qualidade. É necessária a criação de *frameworks* específicos para *debugging* e validação de *scripts*, bem como uma estrutura de formação especializada. Desta forma, o IaC pode transformar-se numa das ferramentas mais importantes para o desenvolvimento de *software* moderno.

Zhang et al. [9] descreve que a utilização de ferramentas de IaC mudou a forma como as infraestruturas computacionais são mantidas, recorrendo à automação e escalabilidade através de ferramentas como o *Ansible*. Contudo, é necessário mitigar os “*test smell*” gerados, que podem causar consequências significativas em larga escala. Foi conduzido um inquérito com 30 profissionais, sobre a qualidade de programação em ficheiros de *Ansible*, cujos resultados revelaram que, entre os três tipos de “*test smells*”, o *Local only testing* é o mais reconhecido como sendo problemático (87%), seguido pelo *Remote mystery guest* (53%) e por fim, o *Assertion roulette*, que teve menor aceitação (21%). É importante encontrar formas de detetar os *test smells* para IaC, considerando as especificidades das linguagens de domínio específico Domain-Specific Language (DSL) usadas em ferramentas como *Ansible*. Deve-se alertar os programadores para os aspetos negativos de *test smells* e desenvolver ferramentas automatizadas para identificar os seus padrões. A qualidade de *scripts IaC* é fundamental para a cibersegurança, pois configurações incorretas podem introduzir vulnerabilidades. Assim, a deteção de *test smells* e a realização de testes não são apenas uma questão de eficiência, mas também uma medida essencial para proteger contra ameaças em infraestruturas cada vez mais complexas e interligadas.

Bahaweres et al. [52] descreve como o uso de IaC, em conjunto com o *Terraform* e *Kubernetes*, constitui uma solução para a implementação de processos de recuperação de desastres em infraestruturas baseadas na nuvem. Após desastres naturais ou falhas de *hardware/software* é necessário garantir a continuidade dos serviços e das infraestruturas de TI, mas os métodos convencionais de recu-

peração de desastres envolvem processos lentos, manuais e suscetíveis a erros humanos. O uso do IaC é indicado como sendo o melhor método para a criação e gestão da infraestrutura de uma forma automatizada, reduzindo o tempo e a complexidade. Como ferramenta para orquestração de contentores é destacado o *Kubernetes*, devido à sua maior eficiência em cenários de implementação e recuperação de *software*. O objetivo é avaliar a eficácia da utilização do *Terraform* e do *Kubernetes*, tendo os seus resultados indicado que o *Terraform* reduz o tempo necessário para a criação de infraestruturas em até 235%, enquanto o *Kubernetes* acelera os tempos de implementação de *software* em 530%. Além disso, a recuperação utilizando *Terraform* e *Kubernetes* foi 190% mais rápida do que o uso de métodos manuais. A interligação das duas tecnologias demonstra ser uma solução eficaz para recuperação de desastres, eliminando a necessidade de infraestrutura redundante, o que reduz custos e aumenta a agilidade do sistema.

Prakash et al. [53] descreve a integração de DevOps e computação em nuvem como uma evolução significativa no desenvolvimento de *software*. O DevOps é a ligação entre o desenvolvimento (*Dev*) e as operações (*Ops*). Esta ligação cria um ambiente de colaboração entre as equipas, automação e melhoria contínua ao longo do ciclo de vida do Software Development Life Cycle (SDLC). A computação em nuvem fornece recursos escaláveis sob demanda. A utilização de IaC, testes automatizados e a automação de processos como CI/CD, proporciona um desenvolvimento de *software* mais adaptado às necessidades de cada projeto. A integração de diferentes tecnologias e ferramentas no fluxo de trabalho pode ser complexa e demorada. Existem desafios relacionados com a segurança de dados e proteção contra ciberataques que exigem métodos de verificação em ambientes de nuvem. Como solução, propõem a utilização de tecnologias como o *Docker* e *Kubernetes* para simplificar a integração e implementação de *software*, além da aplicação de medidas de segurança automatizadas, como monitorização contínua, encriptação de dados, controlo de acesso e ferramentas de gestão de nuvem para monitorizar recursos e definir estratégias de escalonamento automático. Empresas como *Netflix* e *Etsy* demonstram o impacto positivo da integração, utilizando ferramentas como *Terraform* e serviços AWS para gestão de infraestrutura. A utilização de DevOps e computação em nuvem continuará a liderar a transformação digital.

Supit et al. [54] descreve que o uso de Cloud-Native Architecture (CNA) e DevOps acelerou o processo de desenvolvimento de *software* e o provisionamento de infraestrutura numa organização de veículos elétricos. Através de entrevistas e observações, foram identificados quatro problemas no processo de desenvolvimento SDLC: processos não padronizados, tecnologia inadequada, lacunas nas competências dos programadores e falta de documentação adequada. A adoção de CNA e DevOps com o uso de ferramentas como IaC, é usada para automatizar a configuração de infraestrutura e minimizar erros humanos. Recomendam o uso de Version Control System (VCS) para monitorizar alterações e reduzir erros no código, e, por último, a Integração Contínua/Entrega Contínua CI/CD para automatizar o processo das atualizações. Através desta solução, verificou-se uma redução de 67% no tempo de provisionamento de infraestrutura e implantação de aplicações.

Zhang et al. [55] descreve como a utilização de IaC permite uma gestão automatizada de infraestruturas computacionais em grande escala. As orquestrações da infraestrutura utilizam ferramentas conhecidas, como o *Ansible*. Apesar de todos os benefícios conhecidos, como a automação e a redução de erro humano, as práticas de IaC são suscetíveis à ocorrência de problemas na qualidade do código, que podem levar a consequências graves em sistemas críticos. Foi realizado um estudo que envolveu 22.445 *commits* do repositório de código aberto do *Ansible* que categorizou e analisou defeitos em componentes do programador, utilizando técnicas qualitativas e quantitativas de análise de dados. Entre 12 componentes analisados no *Ansible*, o componente *Module* destacou-se como o mais propenso a defeitos, representando 69,23% dos ficheiros *Python* afetados. Outros componentes relevantes incluem o *Executor* (6,23%) e *Utils* (15%), que são cruciais para o funcionamento de *playbooks* e operações utilitárias. A criação de taxonomias abrangentes pode facilitar a identificação e categorização de defeitos exclusivos do IaC, contribuindo para a melhoria da ciência da qualidade em DevOps. Além disso, a priorização de componentes com maior prevalência de defeitos pode otimizar os esforços de manutenção e reparação automática de código.

Numa lógica similar, orientada para a automatização e otimização de processos, temos o artigo de pesquisa de Santos et al. [56]. Este descreve como o conceito de Multi-access Edge Computing (MEC) tem como objetivo garantir que os serviços de computação e armazenamento estejam próximos dos utilizadores finais, oferecendo tempos de resposta mais rápidos e latência mais baixa para as aplicações, assim como para o IoT. Esta descentralização conduz a novos desafios relacionados com a segurança e a gestão eficiente de recursos distribuídos. A computação de *edge* aproxima os dispositivos dos recursos de processamento, permitindo uma redução da latência e melhoria no desempenho geral da rede. A segurança representa uma preocupação, com desafios relacionados com a autenticação, gestão de chaves e proteção de dados em sistemas distribuídos. O uso de ferramentas de IaC mostrou-se ser importante na automação da implantação de aplicações em ambientes baseados em MEC. Considerando as ferramentas de IaC que foram examinadas - *Ansible*, *Terraform* e *TOSCA* - cada uma apresenta diferentes abordagens de como implementar e automatizar infraestruturas e serviços. O uso de contentores e arquiteturas baseadas em microserviços tem-se mostrado promissor para a sua implantação. No que diz respeito à escalabilidade, as técnicas de escalonamento horizontal e vertical são cruciais para assegurar os serviços MEC. Relativamente ao isolamento e à segurança, métodos como o *slicing* são amplamente utilizados para garantir que múltiplos serviços que partilham recursos MEC não interfiram negativamente entre si. Por fim, a automação e a interoperabilidade são viabilizadas através de ferramentas IaC, aliadas a padrões como o *TOSCA*, que são essenciais para assegurar a interoperabilidade entre sistemas heterogêneos.

Petrovic et al. [57] apresenta uma abordagem em que utiliza o potencial do *ChatGPT* para análise estática de código no *design* de *scripts* de IaC, utilizando as ferramentas *Terraform* e *Ansible*, para DevSecOps. Estas tecnologias desempenham um papel fundamental na definição de infraestruturas de *data centers* e

na gestão de configuração de *software*, respetivamente. A proposta baseia-se na capacidade do *ChatGPT* identificar vulnerabilidades nos *scripts* submetidos. Ferramentas como o *IaC Scan Runner* são utilizadas para a análise de código estático. Este software verifica sintaxe, semântica, dependências vulneráveis e exposição de credenciais. Foi utilizado via *API Python* para analisar *scripts IaC*, oferecendo recomendações detalhadas e identificando falhas de segurança. A análise comparativa com ferramentas tradicionais mostrou que o *ChatGPT* é capaz de identificar falhas não detetadas pelo *IaC Scan Runner*, com a vantagem adicional de propor soluções. Existem limitações e desafios associados à abordagem baseada em *ChatGPT*. A sua utilização implica custos significativos, dependendo do volume de chamadas à *API*. Além disso, algumas respostas do *ChatGPT* não são confiáveis, diminuindo a confiança dos programadores. Desta forma, a sua aplicação como ferramenta autónoma em *pipelines* de produção não é recomendada.

Zeini et al. [5] propõe uma *framework* de gestão de riscos que integra práticas de DevSecOps no sentido de assegurar a segurança desde as fases iniciais do ciclo de desenvolvimento, associadas à utilização de IaC. O uso de IaC traz benefícios significativos, como a automação e consistência na configuração de infraestruturas, mas sem práticas de segurança adequadas pode provocar a criação de vulnerabilidades. O *framework* está dividido em três áreas: formação e gestão de conhecimento, para aumentar a consciência sobre segurança nas equipas; a avaliação de riscos, classificando vulnerabilidades e desenvolvendo estratégias para mitigar potenciais ameaças; e, por último, a monitorização contínua, para garantir que alterações na infraestrutura sejam monitorizadas e controladas em tempo real. Existem ameaças específicas associadas ao IaC, como o aumento dos serviços expostos, nomeadamente o SSH e RDP, devido à configuração incorreta e à falta de compreensão do modelo de responsabilidade partilhada em ambientes de computação na nuvem. Outras ameaças são o crescimento da *Shadow IT*, de servidores "zombie", dos efeitos de *deploy blue/green* e os riscos em ambientes *edge* e *fog*. Embora o *framework* seja um passo significativo para melhorar a resiliência do IaC, são reconhecidas limitações, incluindo a necessidade de estudos adicionais para expandir o *framework* e a criação de ferramentas específicas para testes e modelação de ameaças.

Sokolowski et al. [58] aborda os desafios no uso da fiabilidade que os programas de IaC proporcionam, fundamentalmente para o desenvolvimento da automação de implementações de aplicações em nuvem. Os programas de IaC evoluíram de simples *scripts* de configuração para estruturas mais semelhantes a *software* tradicional, com complexidade crescente das aplicações atuais. Falhas na programação em IaC podem impedir a implementação correta de aplicações, introduzindo vulnerabilidades críticas de segurança. Técnicas implementadas na engenharia de *software*, como testes e verificações, são pouco utilizadas em IaC. Os testes de integração são lentos, dispendiosos e limitados no que diz respeito à abrangência de configurações. Para enfrentar estes desafios, é proposta uma solução chamada *ProTI*, uma ferramenta que automatiza os testes em programas de IaC, com técnicas modernas como *fuzzing* (testes aleatórios) e testes baseados em propriedades. O *ProTI* foi concebido para reduzir o esforço manual envolvido nos testes e, ao mesmo tempo, aumentar a confiança nos resultados. A ferramenta permite testar múltiplas configurações de forma rápida, abrangendo uma maior variedade de

cenários e aumentando a robustez do programa. Entre as principais funcionalidades do *ProTI* destacam-se a automação da criação de *mocks*, simplificação de testes unitários, uso de geradores de valores aleatórios para testes de configurações variadas, suporte a testes baseados em propriedades para validar comportamentos específicos e a introdução de verificações automáticas de propriedades essenciais em IaC.

Konala et al. [59] explora o estado atual das técnicas e ferramentas de análise estática de configuração em *scripts* de IaC. O uso crescente das ferramentas de IaC tem gerado benefícios na gestão de infraestruturas, mas também introduziu riscos significativos, como *code smells*, potenciais problemas de *design* e *security smells* e potenciais vulnerabilidades de segurança. A pesquisa atual foca-se principalmente no provisionamento de infraestruturas e menos na gestão de configuração e construção de imagens. A maioria das ferramentas concentra-se na detecção de *security smells* usando expressões regulares, mas essa técnica tem limitações, nomeadamente não ser totalmente automatizada, e a sua eficácia ser afetada por mudanças no *software*. O estudo revela que ferramentas como *Ansible*, *Chef* e *Puppet* são as mais usadas e que a detecção de "*security smells*" é prioritária devido ao risco na criação de vulnerabilidades. Existem lacunas na disponibilidade de *datasets* abrangentes, fundamentais para desenvolver técnicas avançadas, através de *machine learning* e análise de padrões. Ferramentas como *Checkov*, *KICS* e *Semgrep* têm mostrado que são mais-valias importantes, mas ainda estão em estágios iniciais de maturidade. O artigo conclui que a análise de configuração estática é crucial para garantir a qualidade e segurança de *scripts* IaC.

Alonso et al. [16] analisa os desafios e benefícios da integração da filosofia DevSecOps, recorrendo ao uso de IaC, através de um *framework* para o desenvolvimento e gestão de infraestruturas que sejam confiáveis e seguras. Através do IaC, a automação de tarefas relacionadas com o aprovisionamento, configuração e gestão de infraestruturas é realizada de forma eficiente e consistente em ambientes de computação na nuvem. São destacados desafios como fragmentação de ferramentas, necessidade de conhecimentos especializados, confiabilidade do código e desvios de configuração. O processo de DevSecOps é dividido em quatro fases principais: planeamento, criação e aprovisionamento do IaC; verificação da confiabilidade do IaC; lançamento e configuração e a monitorização. O *framework* proposto inclui ferramentas como o *DOML Editor*, para análise de segurança estática SAST e ambientes *sandbox* para testes dinâmicos e o uso de metodologias de análise dinâmica Dynamic Application Security Testing (DAST) para monitorizar o desempenho em tempo real. A utilização de *machine learning* é aplicada na detecção de anomalias. A utilização de uma *framework* com qualidade em DevSecOps, pode minimizar os problemas de interoperabilidade, reduzir a curva de aprendizagem e automatizar a criação de código de infraestruturas confiáveis. A integração de mecanismos de monitorização e segurança aumentou a resiliência das infraestruturas geridas por IaC, alinhando-se com os princípios da triade CIA (Confidencialidade, Integridade e Disponibilidade).

Falazi et al. [60] apresenta uma revisão sistemática da literatura sobre os desafios e soluções na adoção da filosofia DevSecOps. O estudo analisa 54 artigos, identificando 21 desafios e 31 soluções. Os resultados foram organizados em quatro

temas principais: Pessoas, Práticas, Ferramentas e Infraestrutura. No tema Pessoas, os desafios incluem a falta de comunicação entre equipas de desenvolvimento, operações e segurança e a insuficiência de conhecimento especializado. Como solução, destaca-se a adoção de estratégias de comunicação colaborativa com responsabilidades claramente definidas e a implementação de programas de formação para programadores. Nas Práticas, os desafios são a baixa automatização de processos de segurança e a existência de poucos métodos padronizados na avaliação contínua de segurança nos *pipelines*. Como solução, o estudo propõe a implementação do conceito de *Shifting Left*, com a identificação de problemas de segurança nas fases iniciais do ciclo, a automação de testes de conformidade e a integração de práticas contínuas de segurança. Em relação às Ferramentas, os desafios incluem a complexidade das ferramentas DevSecOps, a falta de documentação adequada e a possível criação de vulnerabilidades em *containers* e nos *pipelines* CI/CD, aumentando assim a superfície de ataque. Entre as soluções, destacam-se a padronização de ferramentas, a centralização de documentação, o uso de ferramentas combinadas como Interactive Application Security Testing (IAST), SAST e DAST, plataformas de orquestração para isolamento de *containers* e a análise estática de *scripts* IaC para identificar *security smells*. Por fim, no tema Infraestrutura, os desafios incluem a dificuldade de aplicar DevSecOps em ambientes complexos e em ambientes regulamentados, como infraestruturas de saúde e ambientes *air-gapped*. As soluções propostas incluem o uso de IaC para automatizar configurações e reduzir erros.

Acheampong et al. [61] destaca a crescente importância da cibersegurança face ao aumento exponencial do cibercrime e da sofisticação dos ataques. Com esse intuito é proposta uma metodologia de *design*, para implementação e automação de um *Cyber Range* criado em ambientes virtuais e na *cloud*, utilizando a ferramenta *Ansible*. A criação de cenários de segurança é fundamental para a formação de profissionais de cibersegurança, com simulações de ataques e defesas em infraestruturas automatizadas, integrando conceitos de DevSecOps e IaC. O *Cyber Range* é escalável, reutilizável e retrata ataques de redes e de aplicações *web*. Para uma implementação mais acessível, utiliza ferramentas *open-source*. Entre as principais ferramentas implementadas estão *Docker*, *VMware vSphere*, *GNS3* para emulação de redes e aplicações como *OWASP ZAP*, *DVWA*, *Infection Monkey* e *Nessus*, integradas através de *scripts* *Ansible* para automatizar a configuração e orquestração da infraestrutura. O estudo apresenta seis cenários de segurança distintos, explorando tópicos como a verificação de vulnerabilidades em aplicações *web* com *OWASP ZAP*, a monitorização de ataques e segurança de código com *Contrast Security*, a análise de servidores *web* com *Nikto*, a simulação de ataques automatizados com *Infection Monkey*, o reconhecimento e a exploração de topologias de rede utilizando *Nmap* e a avaliação detalhada da rede com *Nessus*. As vulnerabilidades encontradas são mapeadas para a base de conhecimento *MITRE ATT&CK*, ajudando as equipas a planearem defesas mais eficazes. Os resultados indicam que o *Cyber Range* proposto é eficiente na preparação de profissionais de cibersegurança, oferecendo um ambiente prático e ágil para treino. Para trabalhos futuros, é sugerida a exploração de plataformas de realidade virtual (VR) para expandir os conceitos da formação em cibersegurança.

Hassan et al. [62] realiza um estudo empírico sobre a qualidade de *scripts* de

teste *Ansible* em projetos de código aberto, destacando a importância da detecção e mitigação de *bugs* em *scripts* de IaC. A pesquisa baseia-se na análise de 4.831 *scripts* de teste extraídos de 104 repositórios de software de código aberto (OSS) no *GitHub*, identificando padrões recorrentes associados a *bugs* e propondo recomendações práticas para melhorar a qualidade e confiabilidade dos testes. Entre as descobertas, 1,8% dos *scripts* analisados contêm *bugs*, sendo que 45,2% dos repositórios apresentam pelo menos um *script* de teste afetado. Os *bugs* foram classificados em sete categorias principais: configuração, dependência, idempotência, *logging*, desempenho, segurança e estilo. O estudo identifica também três padrões de teste que frequentemente se correlacionam com a presença de *bugs*: "*assertion roulette*", "*local only testing*" e "*remote mystery guest*". Esses padrões dificultam o rastreamento de falhas, comprometem a reprodutibilidade dos testes e podem afetar negativamente a infraestrutura provisionada. O uso da ferramenta Test Pattern Miner for Ansible (TAMA) é proposto para identificar padrões problemáticos. Esta ferramenta, baseada em análise automática de *scripts*, facilita a priorização de inspeções e a correção de falhas, sendo particularmente útil em ambientes DevOps onde a automação é essencial.

Dalvi [63] propõe um sistema de infraestrutura em *cloud* baseado em DevOps com um mecanismo de *self-service*, simplificando e automatizando a gestão de recursos em ambientes *multi-cloud*. Centros de excelência em *Cloud* (CCoE) foram criados para desenvolver procedimentos padrão operacionais e modelos de IaC para as equipes de desenvolvimento tornando-os mais seguros e fáceis de trabalhar. Estes *templates* permitem que os programadores usem recursos sem intervenção direta das equipes de infraestrutura, reduzindo, desta forma, atrasos operacionais e minimizando erros de configuração. A metodologia apresentada realça a utilização de ferramentas, como o *Terraform* para a criação destes *templates*, armazenados num repositório centralizado, que incorporam padrões de configuração definidos pelas equipes de infraestrutura, segurança, rede e operações. Foi criada uma interface gráfica para permitir que os utilizadores personalizem variáveis dentro de limites predefinidos, garantindo, assim, que os recursos provisionados estejam alinhados com as melhores práticas e requisitos organizacionais. O artigo conclui que a adoção de um sistema automatizado baseado em IaC para entrega de infraestrutura *cloud* melhora a produtividade das equipes, aumenta a agilidade dos programadores e assegura uma gestão mais eficiente e segura em ambientes *multi-cloud*.

Sorour et al. [64] retrata a implementação de uma solução baseada em DevOps e IaC para automatizar o mecanismo de *self-service* em laboratórios de *software*, adaptando-se às necessidades impostas pela transição para o ensino remoto durante a pandemia de COVID-19. A proposta utiliza uma arquitetura escalável e altamente disponível na *cloud*, aproveitando tecnologias modernas como a virtualização e orquestração, o uso de *containers*, para compensar a inexistente estrutura de *hardware*, facilitar a gestão de escalabilidade e a mitigação de fraudes ou plágio. Para a criação de imagens de servidor é utilizado o *Packer*; para gestão de configuração, o *Ansible*; e para o provisionamento de infraestrutura, o *Terraform*. A orquestração é feita com recurso a *Kubernetes*, aproveitando a sua capacidade de escalabilidade automática e recuperação de falhas. O *backend* foi desenvolvido em *NodeJS* com *ExpressJS*, a interface de utilizador foi construída com *React*

e *NextJS*, enquanto os dados são armazenados em *MongoDB*. O sistema garante que instâncias predefinidas com o conteúdo necessário para os exames sejam configuradas e disponíveis para utilização em menos de três minutos, melhorando a eficiência operacional e reduzindo o tempo de configuração. Os testes mostraram que a arquitetura lida com cargas súbitas graças à escalabilidade e auto-recuperação proporcionadas pelo *Kubernetes*. O uso de *templates* predefinidos assegura a conformidade e a consistência da infraestrutura, minimizando erros humanos. O artigo conclui que esta abordagem representa uma solução prática e eficaz para instituições educacionais que necessitam de realizar exames práticos *online*, mantendo padrões elevados de confiabilidade e desempenho, e sugere que futuras investigações explorem integrações adicionais para reforçar a segurança e a gestão de acessos.

Reis et al. [65] analisa os conceitos e os desafios enfrentados pelos programadores na criação e manutenção de especificações em *Docker* e *Docker-Compose*. A pesquisa foi realizada com recurso a um inquérito com 120 participantes, incluindo estudantes e profissionais, que consideraram o desenvolvimento de *Dockerfiles* e ficheiros *docker-compose.yml* como um processo complexo, especialmente para programadores com menos experiência. Destacam-se a identificação de dependências do sistema, a validação do funcionamento correto dos *containers* e a depuração de problemas em *containers* mal configurados. Profissionais mais experientes relatam maior eficiência em algumas etapas, mas consideram que tarefas como depuração de *containers* e configuração de dependências são exigentes. Já programadores menos experientes recorrem a abordagens baseadas em tentativa e erro, com forte dependência de recursos externos, como documentação e *fóruns online*. A análise revela que 78% dos participantes não recorrem a *plugins* ou ferramentas adicionais para o desenvolvimento de *Dockerfiles* e 83% para *docker-compose.yml*, no entanto, mencionam a utilização de *linters*, realce de sintaxe *highlighters* e gestores locais de imagens e *containers*. O artigo conclui que existe uma necessidade significativa de melhorias nas ferramentas e metodologias de suporte ao desenvolvimento de especificações *Docker* e *Docker-Compose*.

Kumara et al. [66] apresenta uma análise sistemática da literatura sobre IaC, explorando boas e más práticas na utilização de linguagens, como *Ansible*, *Puppet* e *Chef*. A análise foi realizada com recurso a 67 fontes de literatura cinzenta, aplicando técnicas de análise qualitativa para identificar conceitos recomendados e anti-padrões no desenvolvimento de código. A revisão propõe uma definição integrada de IaC, dividida em três áreas: tipos de operações de gestão, métodos de implementação e propriedades desejadas nos ambientes geridos. Estas incluem virtualização, consistência, imutabilidade, auditabilidade e reprodutibilidade, todas estas fundamentais para garantir a eficiência e a segurança na gestão de recursos computacionais, como infraestrutura, *software*/plataformas e aplicações. Foram identificadas dez categorias, tais como a escrita de código para pessoas (priorizando legibilidade e modularidade), o uso correto de ferramentas IaC para garantir idempotência e reprodutibilidade, a separação de dados de configuração do código e a proteção de dados com encriptação. Foram abordados conceitos específicos para cada linguagem, como a criação de *templates* reutilizáveis no *Ansible*, o uso de perfis e funções no *Puppet* e a gestão de *cookbooks* no *Chef*, assim como a importância de ferramentas para monitorização e controlo de ver-

sões para manter a rastreabilidade e a consistência nas alterações à infraestrutura. Quatro categorias principais de más práticas foram identificadas, tais como violações aos princípios de idempotência e modularidade, dependência excessiva de configurações manuais e uso inadequado de módulos. A falta de padronização e a compreensão limitada das melhores práticas por parte dos programadores leva à introdução de vulnerabilidades, bem como dificuldade na manutenção e evolução de sistemas baseados em IaC. O estudo conclui que a adoção de conceitos bem estabelecidos, bem como a mitigação de antipadrões, são cruciais para melhorar a qualidade e a segurança dos projetos de IaC.

Muthoni et al. [67] explora os desafios e oportunidades na adoção de computação em nuvem para a continuidade de negócios em Instituições de Ensino Superior (IES), utilizando uma *framework* baseada em IaC. A utilização de ferramentas como *Terraform* e *Ansible* na migração de infraestruturas locais para ambientes na nuvem demonstrou que a automação melhora a eficiência, reduz os custos e aumenta a escalabilidade dos sistemas de recuperação de desastres e continuidade de negócio. A pesquisa foi realizada em três universidades, recorrendo a questionários aplicados a 92 profissionais de Tecnologias da Informação e Comunicação (TIC), incluindo administradores de sistemas, gestores de TI e técnicos. Como resultado, destaca-se a falta de especialização, suporte inadequado, altos custos de conectividade à *Internet*, ausência de regulamentações específicas e preocupações com a segurança de dados. O estudo identificou que a existência de processos manuais é um fator crítico, que aumenta o risco de erro humano e os custos operacionais. A abordagem proposta envolve uma *framework* estruturada em cinco etapas: planeamento, integração, automação, otimização e implantação. Estas etapas incluem a formação de equipas, alinhamento de processos de negócios, implementação de ferramentas IaC para automatizar a configuração de infraestrutura e aplicações e a aplicação de políticas de segurança. O *Terraform* foi utilizado para a criação da infraestrutura base na nuvem, enquanto o *Ansible* geriu a configuração e instalação do *software*. A automação proporcionada por IaC reduziu drasticamente o tempo necessário para configurar e recuperar sistemas, de horas para poucos minutos, garantindo uma escalabilidade flexível e uma recuperação mais fiável dos sistemas.

Rahman et al. [68] explora os desafios e as melhores práticas na gestão de segredos em *scripts* de IaC, como *Ansible*, *Puppet* e *Terraform*. O objetivo é fornecer orientações para profissionais melhorarem a segurança das suas infraestruturas e *scripts* IaC. O artigo realizou uma revisão de literatura "cinzenta" com base em 38 artefactos da *Internet*, incluindo *blogs* e apresentações, identificando 12 práticas essenciais para a gestão de segredos. Entre elas, destacam-se a organização de dados, com diretórios bem estruturados e hierarquias claras; a gestão de palavras-passe para *Ansible Vault*, que inclui evitar o armazenamento de ficheiros com credenciais; e o controlo de acesso, que implementa políticas para restringir a exposição de segredos. A priorização da encriptação para variáveis críticas, a separação de ambientes e artefactos para facilitar a gestão, a monitorização e o registo de operações para garantir a rastreabilidade e a rotação periódica de segredos reduzem o risco de exposição prolongada. Também são abordados tópicos como a segurança da camada de transporte, para proteger a transmissão de dados com TLS e a limitação da autenticação, restringindo tentativas de lo-

gin e duração de sessões. Os resultados do estudo mostram que uma aplicação inadequada destas práticas pode levar a consequências graves, como acesso não autorizado à infraestrutura provisionada. Por outro lado, a adoção das práticas identificadas pode facilitar o desenvolvimento seguro de *scripts IaC*, promovendo maior conformidade com padrões de segurança.

Rahman et al. [11] aborda a identificação e a análise de *security smells* existentes em *scripts* de IaC, destacando padrões de código que podem indicar potenciais problemas de segurança. Os autores analisaram 3.339 *scripts* de ferramentas como *Ansible*, *Chef* e *Puppet*, identificando nove categorias principais de *security smells*. Estas incluem segredos codificados (*hard-coded secrets*), ausência de verificações de integridade, utilização de senhas vazias e algoritmos de criptografia fracos. Estes padrões não são necessariamente vulnerabilidades, mas representam práticas com o potencial de comprometer a segurança, caso não sejam corrigidas. De acordo com as descobertas, os segredos codificados foram os que mais se destacaram, com milhares de ocorrências em todos os conjuntos de dados analisados. Este padrão inclui senhas, chaves privadas e nomes de utilizador integrados diretamente nos *scripts*. Outros problemas frequentes incluem ligações a endereços IP não restritos, falta de proteção em conexões HTTP e comentários no código que expõem informações sobre falhas ou funcionalidades incompletas. A ferramenta SLIC realiza análises estáticas em *scripts IaC*, identificando padrões problemáticos com base em regras pré-definidas. Os resultados indicam que, em média, cerca de 25% dos *scripts* analisados contêm pelo menos um *security smell*, com uma densidade de até 25 ocorrências por mil linhas de código. Para uma adoção de boas práticas, destaca-se o uso de ferramentas especializadas para gestão de segredos (por exemplo, *HashiCorp Vault*), a validação de integridade de pacotes e a utilização de algoritmos de criptografia robustos. Os programadores de IaC devem frequentar formação específica para adquirir consciência e aplicar conceitos em termos de segurança.

Palma et al. [69] apresenta um catálogo de 46 métricas para proceder à avaliação da qualidade do código de IaC no *Ansible*, uma das ferramentas mais utilizadas na automação de infraestrutura. As métricas foram desenvolvidas tendo como base características estruturais do código. Entre as vantagens da sua utilização está a possibilidade de identificar problemas de forma antecipada, evitando abordagens baseadas em tentativa e erro, que são mais propensas a falhas. O catálogo está organizado em três categorias: métricas gerais, para todas as linguagens de IaC; métricas específicas para *playbooks*, que são ficheiros principais no *Ansible*; e métricas relacionadas às listas de tarefas (*tasks*). Como exemplos, destacam-se o número de linhas de código, *loops*, condições, decisões e comandos externos, bem como a complexidade do texto (medida por entropia). Adicionalmente, métricas específicas para o *Ansible*, como o número de módulos depreciados, blocos de tratamento de erros e variáveis únicas, ajudam a diagnosticar a complexidade e a propensão a erros do código. A avaliação da qualidade do código IaC é um desafio em sistemas complexos que utilizam microserviços e componentes *serverless*. As métricas fornecem indicadores de qualidade, facilitando a identificação de defeitos e permitindo a deteção de *code smells* durante a manutenção e o crescimento dos sistemas. Estas métricas estão definidas para o *Ansible*, mas está prevista a sua expansão para suportar outras ferramentas de IaC, como *Chef* e *Puppet*.

Rahman et al. [70] apresenta uma taxonomia de defeitos para *scripts* de IaC, desenvolvida através da análise qualitativa de 1.448 *commits* relacionados com defeitos em repositórios de código aberto. O objetivo é ajudar os profissionais a compreenderem a natureza dos defeitos em *scripts* IaC e a melhorar a sua qualidade. A taxonomia identifica oito categorias de defeitos: dados de configuração, condicionais, dependências, documentação, idempotência, segurança, serviços e sintaxe. Entre os principais resultados, destacam-se os defeitos relacionados com dados de configuração, como configurações erradas com variáveis, que são os mais frequentes. A categoria de idempotência é específica de IaC e refere-se a defeitos que comprometem a consistência do ambiente quando o mesmo *script* é executado repetidamente. Outros defeitos incluem erros em dependências mal especificadas, documentação imprecisa e problemas de segurança, como a exposição de segredos em *logs*. Foi realizado também um inquérito a 66 profissionais, sendo a categoria de idempotência que recebeu a maior aceitação (79%). Estes destacaram a relevância de ferramentas que ajudam a identificar e a mitigar defeitos, como análise estática de código e testes automatizados. A taxonomia e a frequência dos defeitos, medida em mais de 80 mil *commits*, mostram que os dados de configuração são a categoria dominante em todos os repositórios analisados. Como exemplo, foi descrito que um defeito num *script* IaC criou uma interrupção que resultou em perdas comerciais de 150 milhões de dólares para a AWS. O artigo conclui que compreensão dos defeitos e adoção de ferramentas específicas para a sua mitigação podem reduzir os riscos associados ao uso de IaC, melhorando a qualidade e a confiabilidade dos *scripts*. Além disso, a investigação sugere que ferramentas automatizadas, como o ACID, podem apoiar a identificação de categorias de defeitos em larga escala e orientar estratégias de correção e prevenção.

Rahman et al. [19] investiga padrões de segurança, conhecidos como *security smells*, em *scripts* de IaC, que podem indicar vulnerabilidades ou práticas de programação inseguras. A análise empírica foi realizada sobre 15.232 *scripts* recolhidos de repositórios *open-source*, resultando na identificação de sete tipos principais de *security smells*: *admin* por defeito, *password* vazia, segredos codificados, vínculos de IP inválidos, comentários suspeitos, uso de HTTP sem TLS e uso de algoritmos de criptografia fracos. Estes padrões não representam propriamente vulnerabilidades imediatas, mas aumentam significativamente os riscos de falhas de segurança. A ferramenta SLIC foi desenvolvida para realizar análises estáticas automáticas e identificar essas ocorrências em *scripts*. Durante a pesquisa, foram encontrados mais de 21 mil casos de *security smells*, incluindo 1.326 senhas codificadas diretamente nos *scripts*. A análise revelou que alguns padrões, como os segredos codificados, podem persistir nos *scripts* por até 98 meses, destacando a falta de monitorização e correção proativa. A taxa de ocorrência foi particularmente alta para segredos codificados, enquanto outros *smells*, como "*admin* por defeito", foram menos frequentes. Para investigar a perceção de profissionais, foram enviados relatórios de *bugs* relacionados a 1.000 ocorrências aleatórias. Mais de 69% das ocorrências foram reconhecidas pelos programadores como relevantes e a necessitar de correção. São recomendadas estratégias para mitigar esses problemas, incluindo a utilização de ferramentas como Vault para gestão de segredos, aplicações de criptografia adequada e a implementação de *guidelines* cla-

ras para o desenvolvimento seguro de IaC.

4.3 Discussão

Nesta secção, apresenta-se uma discussão dos trabalhos analisados, organizada em torno de cinco temas: automação e segurança; deteção e mitigação de vulnerabilidades; gestão de segredos; *smells*; e formação e consciencialização.

4.3.1 Automação e segurança

A automação proporcionada pelas ferramentas de IaC, tornou-se num pilar fundamental para a implementação de segurança de infraestruturas computacionais. As ferramentas de IaC reduziram as intervenções manuais, diminuindo assim a probabilidade de erros humanos, aumentando consequentemente a consistência das configurações, o que contribui para minimizar a criação de vulnerabilidades. A utilização de ferramentas como o *Terraform* e *Ansible* permitiu a criação e gestão de infraestruturas com um maior grau de eficiência e segurança, simplificando a conformidade através de boas práticas [2][50] [18][6][8]. Ferramentas como o *GitHub Actions* facilitam a integração CI/CD, automatizando *pipelines* de desenvolvimento com validações e testes em cada etapa do processo, o que permite a deteção precoce de falhas [20][21]. Por sua vez, o *SonarQube* contribui significativamente para a segurança ao efetuar análises estáticas do código, identificando vulnerabilidades e *code smells* antes da implementação efetiva das alterações [35][33][34].

Enumeração das ferramentas

A Tabela 4.3 apresenta as principais ferramentas, descrevendo as suas valências e os benefícios que proporcionam à automação, segurança e qualidade da infraestrutura e do código.

4.3.2 Deteção e mitigação de vulnerabilidades

A integração de ferramentas de IaC em *pipelines* de CI/CD, em conjunto com ferramentas de análise estática [70] [60], possibilita a identificação e correção automatizadas de vulnerabilidades. Estudos realizados destacam que o uso de ferramentas automatizadas, em conjunto com o uso de práticas padronizadas e *frameworks* inovadores, têm apresentado resultados positivos na mitigação de riscos associados a configurações incorretas e a padrões inseguros. [8] [5] [50].

A integração de práticas de DevSecOps, como apresentado [5], é essencial para assegurar que a segurança esteja presente desde as fases iniciais do ciclo de desenvolvimento. *Frameworks* baseados nessa filosofia incluem a avaliação contínua de

Tabela 4.3: Ferramentas analisadas e suas valências

Ferramenta	Valências e benefícios
<i>Terraform</i>	Automação e provisionamento de infraestrutura declarativa; suporte <i>multi-cloud</i> ; facilita conformidade e escalabilidade.
<i>Ansible</i>	Gestão de configuração; automação de tarefas repetitivas.
<i>GitHub Actions</i>	Automação de <i>pipelines</i> de CI/CD; integração contínua com controle de versões; permite testes e validações automatizadas.
<i>SonarQube e Checkov</i>	Análise estática de código; detecção de vulnerabilidades e <i>code smells</i> ; reforça segurança e qualidade do código.

riscos, a classificação de vulnerabilidades e a monitorização em tempo real, permitindo a identificação e mitigação de ameaças antes que estas comprometam a infraestrutura [62].

Outro aspeto problemático é o impacto que as possíveis vulnerabilidades podem ter nos ambientes de produção [51]. As falhas em relação à idempotência ou dependências mal configuradas podem levar a interrupções de serviço, com custos significativos, como indicado por [70]. A introdução de ferramentas automatizadas na fase dos testes e validação[18], reduz significativamente esses riscos, oferecendo meios eficientes para validar configurações e manter a infraestrutura estável.

4.3.3 Gestão de segredos

A gestão de segredos em IaC deve ser realizada de uma forma segura para garantir a segurança de infraestruturas computacionais, especialmente em ambientes altamente automatizados. Segredos, como chaves de acesso, senhas e *tokens*, são componentes críticos que, se mal geridos, podem comprometer toda a infraestrutura. Estudos como o de [68][11] destacam que práticas inadequadas, como segredos codificados diretamente nos *scripts*, representam uma das vulnerabilidades mais comuns. A persistência desses segredos durante longos períodos, em alguns casos por até 98 meses, agrava ainda mais os riscos [19].

Entre as melhores práticas para gestão de segredos, os estudos sugerem:

- Armazenamento seguro: Ferramentas como *HashiCorp Vault* são fortemente recomendadas para o armazenamento e gestão de segredos de forma centralizada. Estas permitem a encriptação de alto nível e controlo de acesso [68] [19].
- Separação de ambientes: Manter segredos por ambiente (desenvolvimento, teste e produção) é essencial para evitar que acessos não autorizados num

ambiente comprometam os outros [68].

- Mudança de segredos: A implementação de sistemas de rotação de segredos reduz os riscos de segredos comprometidos. Ferramentas automatizadas podem ajudar a implementar essa prática de maneira consistente [68].
- Auditoria e rastreamento: Monitorar o uso de segredos e registrar operações relacionadas a eles são medidas importantes para identificar atividades suspeitas e responder rapidamente a incidentes [11].

4.3.4 *Smells*

Code smells

Os *code smells* em IaC são indicadores de má qualidade no código, como encapsulamento deficiente ou estruturas densas. Estes problemas podem dificultar a manutenção e aumentar a probabilidade de erros. Segundo [18], a coexistência de *smells* como "*Deficient encapsulation*" e "*Dense structure*" sugere que a presença de um pode indicar a existência de outros. Identificar e corrigir esses problemas é essencial para manter a sustentabilidade e a qualidade dos *scripts IaC*.

Test smells

Os *test smells* são problemas associados a *scripts* de teste, como dependências não documentadas ou testes limitados a ambientes locais. Foram identificados três principais *test smells* em *scripts* de Ansible[9]:

- *Local only testing* (87%): Testes restritos a ambientes locais.
- *Remote mystery guest* (53%): Falta de documentação de dependências remotas.
- *Assertion roulette* (21%): Falta de clareza nos erros reportados.

Security smells

Security smells indicam padrões inseguros que podem levar a vulnerabilidades críticas, como segredos codificados, uso de HTTP sem TLS [19] ou algoritmos de criptografia fracos. Estudos [68] destacaram que cerca de 25% dos *scripts* de IaC analisados continham pelo menos um *security smell*, sendo os "*hard-coded secrets*" os mais prevalentes.

Ferramentas e práticas para mitigação de *smells*

Para mitigar esses *smells*, ferramentas automatizadas[11] são cruciais. Estas permitem a detecção precoce de padrões inseguros e ajudam na aplicação de boas

práticas, como:

- Gestão segura de segredos: Utilização de ferramentas como *HashiCorp Vault*
- Automação de testes: Uso de *frameworks* para validação de configurações.
- Análise contínua: Integração de práticas de DevSecOps para avaliação e mitigação de riscos ao longo do ciclo de vida da infraestrutura.

4.3.5 Formação e consciencialização

A formação e a consciencialização desempenham um papel importante no sucesso da adoção eficaz das ferramentas de IaC e na mitigação de riscos associados à sua utilização e à adoção de práticas adequadas. A falta de conhecimentos técnicos e de consciencialização sobre segurança é um dos principais desafios para o uso eficaz de ferramentas de IaC. Estudos como o de [68] evidenciam que práticas inadequadas, como o armazenamento de segredos em *scripts*, resultam muitas vezes de uma compreensão limitada dos riscos associados. Da mesma forma, [9] demonstra que a falta de preparação dos programadores contribui para a presença de *test smells* que dificultam a confiabilidade e a manutenção do código [9]. Com formação especializada e consciencialização, as organizações podem reduzir significativamente os erros humanos, melhorar a qualidade dos *scripts IaC* e aumentar a resiliência das infraestruturas. Além disso, as equipas bem preparadas têm maior capacidade de integrar ferramentas avançadas e resolver problemas de maneira proativa, minimizando interrupções e riscos de segurança.

4.4 Conclusão

O capítulo sobre o estado da arte começa com a descrição da metodologia utilizada na revisão sistemática. A ferramenta *Rayyan* foi utilizada para a categorização dos artigos e o diagrama *PRISMA* foi aplicado no processo de identificação dos artigos mais relevantes, resultando em 31 artigos selecionados. A análise dos artigos abrange cinco áreas, a automação e segurança, a deteção e mitigação de vulnerabilidades, a gestão de segredos, a identificação de *smells* e, por último, a importância da formação e consciencialização das equipas para reduzir erros humanos e aumentar a qualidade e resiliência das infraestruturas.

O conceito de utilização de ferramentas de IaC representa uma mudança nos procedimentos e no modo de trabalhar, constituindo uma nova abordagem para a modernização de infraestruturas computacionais. Esta transformação é alcançada através da automação, escalabilidade e reforço das políticas de segurança em ambientes cada vez mais complexos. A implementação de ferramentas IaC exige uma abordagem bem planeada e estruturada, de modo a maximizar os benefícios proporcionados pela automação, ao mesmo tempo que se mitigam riscos associados, como práticas inadequadas, configurações inseguras e falta de conhecimento técnico.

A utilização de ferramentas como o *Terraform* e o *Ansible*, aliadas a *frameworks* e ao conceito de DevSecOps, constitui uma solução eficaz para o provisionamento e a gestão eficazes da infraestrutura e da segurança.

Programas de formação e sensibilização das equipas são indispensáveis para reduzir erros humanos na codificação e para promover a adoção de padrões de segurança. Apesar de desafios como a fragmentação de ferramentas, a complexidade dos ambientes *multi-cloud* e a presença de *security smells*, as soluções propostas — como a automação de testes, a integração de CI/CD e *frameworks* unificados — revelam-se úteis para transformar a utilização de ferramentas de IaC numa base sólida para a gestão de infraestruturas modernas.

Conclui-se que o sucesso das ferramentas de IaC depende de uma combinação de tecnologia, processos e pessoas. A implementação estratégica destas práticas, ferramentas e *frameworks* garante não apenas a eficiência operacional, mas também a resiliência e a segurança das infraestruturas, permitindo que as organizações retirem pleno partido dos benefícios de ambientes automatizados e mais seguros.

Capítulo 5

Desenho e implementação do cenário de avaliação

5.1 Descrição do cenário

Como referido na Secção 1.2, a adoção de práticas baseadas em IaC, utilizando ferramentas como o *Terraform* e o *Ansible*, representa uma transição significativa face aos métodos tradicionais de gestão de infraestrutura [2][50]. Esta abordagem permite intervenções automatizadas, consistentes e escaláveis, fomentando a colaboração eficiente entre as equipas de desenvolvimento, operações e segurança [16] [68].

No âmbito desta dissertação, foi desenvolvido um cenário que simula um ambiente realista de infraestrutura em AWS, onde o ciclo de vida da infraestrutura é automatizado — desde o provisionamento, passando pela configuração e validação, até à eventual destruição do que foi provisionado — recorrendo às ferramentas *Terraform* e *Ansible*, orquestradas por *GitHub Actions*.

A análise realizada permite avaliar o impacto da utilização destas ferramentas na segurança da infraestrutura, integrando-as em *pipelines* de integração e entrega contínuas (CI/CD) e aplicando ferramentas de análise estática de segurança (SAST), como o *SonarQube* e o *Checkov*, com o objetivo de identificar más configurações, vulnerabilidades e não conformidades face às boas práticas [16][60].

Esta integração reforça a postura de segurança e assegura uma gestão segura, confiável e contínua de ambientes em nuvem, fortalecendo a resiliência da infraestrutura [52] [53].

5.1.1 Arquitetura geral

A infraestrutura implementada é composta por diversos recursos, conforme ilustrado no diagrama da Figura 5.1. De seguida, enumeram-se os principais elementos:

A Figura 5.1 apresenta a estrutura hierárquica dos ficheiros do cenário, evidenciando a organização entre o aprovisionamento da infraestrutura realizado com o *Terraform*, a respetiva configuração efetuada com o *Ansible* e os mecanismos de integração contínua implementados com o *GitHub Actions*.

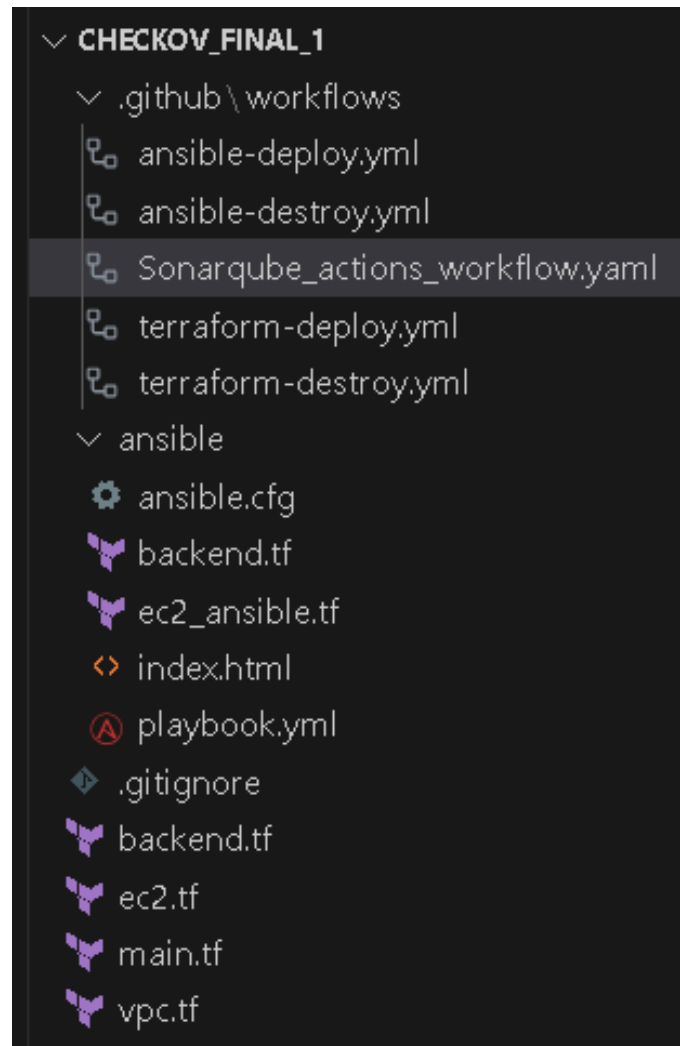


Figura 5.1: Estrutura geral de ficheiros

Com o objetivo de organizar a implementação da infraestrutura e assegurar a integração contínua com práticas de segurança, foi definida uma estrutura de ficheiros modular. Esta organização proporciona uma separação clara entre o aprovisionamento da infraestrutura, a sua configuração e a análise de segurança, facilitando a gestão e a escalabilidade do projeto.

A Tabela 5.1 apresenta a estrutura de ficheiros implementada, incluindo uma breve descrição das respetivas funcionalidades:

Ficheiro	Descrição
.github/workflows/ansible-deploy.yml	Workflow do <i>GitHub Actions</i> para realizar o <i>deploy</i> da configuração via <i>Ansible</i> .

Ficheiro / Diretoria	Descrição
.github/workflows/ansible-destroy.yml	<i>Workflow do GitHub Actions para destruir/reverter a configuração aplicada via Ansible.</i>
.github/workflows/Sonarqube_actions_workflow.yml	<i>Workflow do GitHub Actions para análise de qualidade e segurança de código utilizando SonarQube.</i>
.github/workflows/terraform-deploy.yml	<i>Workflow do GitHub Actions para realizar o deploy da infraestrutura com Terraform.</i>
.github/workflows/terraform-destroy.yml	<i>Workflow do GitHub Actions para destruir a infraestrutura criada via Terraform.</i>
ansible/ansible.cfg	Ficheiro de configuração do <i>Ansible</i> , definindo parâmetros de execução, como inventário e opções de ligação.
ansible/backend.tf	Definição do <i>backend</i> para armazenamento remoto do estado do <i>Terraform</i> relacionado à configuração <i>Ansible</i> .
ansible/ec2_aws.tf	Definições em Terraform para a criação de instâncias Elastic Compute Cloud (EC2) configuradas com <i>Ansible</i> .
ansible/index.html	Página <i>web</i> HyperText Markup Language (HTML) a ser disponibilizada nas instâncias configuradas.
ansible/playbook.yml	<i>Playbook</i> principal do <i>Ansible</i> que define as tarefas a serem executadas nas instâncias.
.gitignore	Ficheiro que especifica padrões de ficheiros/pastas a serem ignorados pelo <i>Github</i> .
backend.tf	Ficheiro principal de <i>backend</i> do <i>Terraform</i> para gestão do estado da infraestrutura.
ec2.tf	Definições de instâncias EC2 na AWS via <i>Terraform</i> .
main.tf	Ficheiro principal do <i>Terraform</i> que organiza os recursos e módulos da infraestrutura.
vpc.tf	Definições da Virtual Private Cloud (VPC) e sub-redes, configuradas através do <i>Terraform</i> .

Tabela 5.1: Estrutura geral de ficheiros do projeto

5.1.2 Mecanismos de segurança

Foram aplicadas diversas práticas de segurança no projeto:

- Definição de *Security Groups* com regras mínimas necessárias;

- Utilização do Key Management Service (KMS) para encriptação de logs e volumes Elastic Block Store (EBS);
- Políticas IAM baseadas no princípio de menor privilégio;
- Armazenamento seguro de variáveis sensíveis em sistemas como *AWS Secrets Manager* ou *GitHub Secrets*.

Para complementar a segurança da infraestrutura, foi realizada uma análise estática de código (SAST) utilizando o *SonarQube* e, posteriormente, o *Checkov*, com o objetivo de identificar vulnerabilidades e assegurar a conformidade com boas práticas de desenvolvimento seguro.

5.1.3 Configuração do *identity and access management* na AWS

A configuração do IAM na AWS foi realizada com base no princípio do menor privilégio, garantindo que cada utilizador, serviço ou aplicação possui apenas as permissões estritamente necessárias para desempenhar as suas funções específicas. Este princípio diminui significativamente o risco de acessos indevidos e fortalece a segurança da infraestrutura [5] [16].

Gestão de identidades

Este serviço possibilita integrar a gestão de identidade e autenticação de utilizadores com sistemas externos compatíveis com protocolos como o OpenID Connect (OIDC), incluindo o *GitHub*.

A Figura 5.2 ilustra a configuração do IAM na AWS, refletindo a aplicação do princípio do menor privilégio.

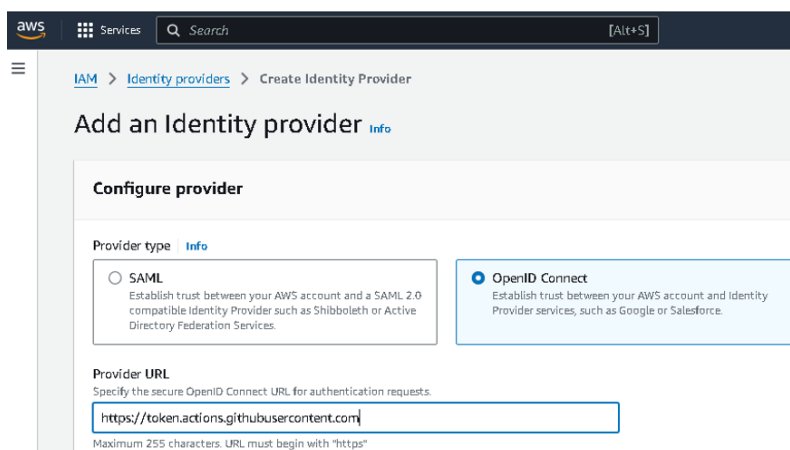


Figura 5.2: IAM - Identify Provider

- OIDC: Permite à AWS estabelecer um canal de confiança com o *GitHub* possibilitando a autenticação de utilizadores ou aplicações que interagem com recursos da AWS.

- Integração com *GitHub Actions*: Configuração que permite aos fluxos de trabalho do *GitHub Actions* (fluxos de trabalho de CI/CD aceder, de forma segura, a recursos na AWS).

Criação de *role*

O *Role* foi configurado para aplicar políticas baseadas no princípio do menor privilégio, permitindo apenas o acesso aos recursos estritamente necessários para a execução das tarefas no *GitHub Actions*. Desta forma, o utilizador definido fica autorizado a executar apenas as ações previamente autorizadas.

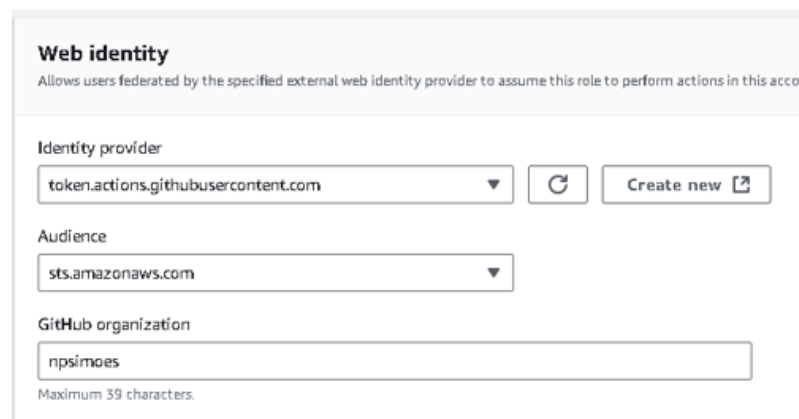


Figura 5.3: *IAM - Role*

A Figura 5.3 apresenta a criação de um *Role* IAM com políticas restritivas.

O acesso foi restringido à organização *GitHub* e ao utilizador *npsimoes*. Esta configuração assegura que apenas fluxos de trabalho de repositórios pertencentes a este utilizador possuem acesso, evitando acessos não autorizados de outros repositórios *GitHub*.

5.1.4 *GitHub Actions*

Foi criado um repositório no *GitHub* como plataforma de controlo de versões e gestão de código-fonte, o qual permite armazenar os ficheiros e diretórios necessários para a configuração e gestão de infraestrutura automatizada através de IaC com o uso de *Terraform*.

No *GitHub Actions* foi configurado para executar os processos de *terraform plan*, *terraform apply*, a execução de *ansible-playbook*, bem como as fases de validação com o *SonarQube*, ativados mediante a ativação manual por parte do utilizador, através dos botões disponibilizados na Figura 5.4.

Desta forma, é possível implementar um *pipeline* de CI/CD completo e seguro, garantindo a automatização em cada etapa do processo, com base numa interação inicial controlada.

A Figura 5.4 ilustra a configuração dos *workflows* no *GitHub Actions*, organizados para automatizar o provisionamento, a configuração da infraestrutura e as fases de implementação de um pipeline CI/CD.

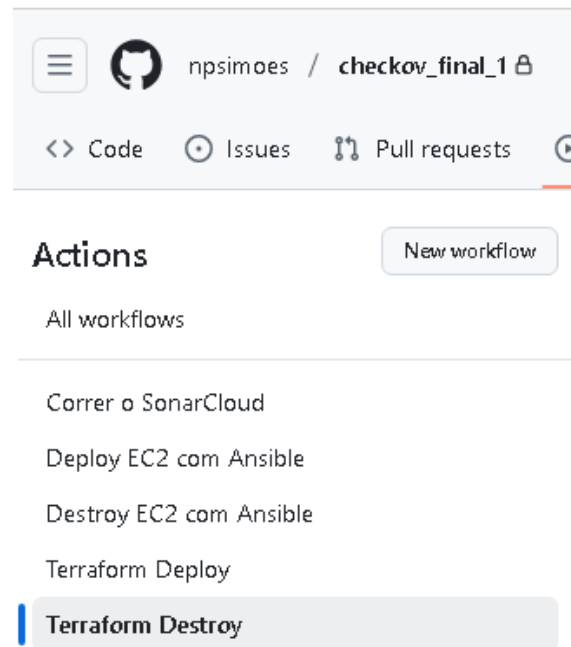


Figura 5.4: Estrutura geral do Github Actions

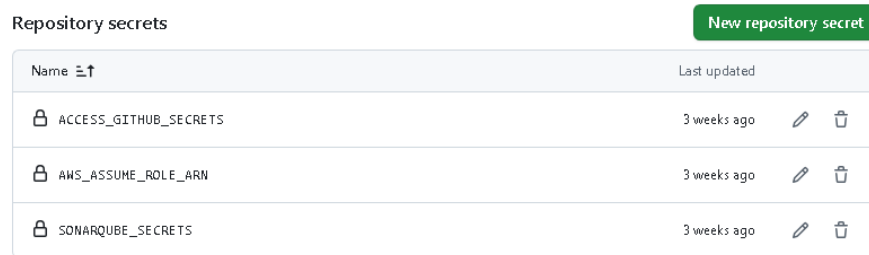
O ciclo de funcionamento decorre da seguinte forma:

1. *Terraform* efetua o *deploy* da infraestrutura;
2. *Ansible* configura e instala serviços nas instâncias EC2, previamente criados;
3. Posteriormente, todo o ambiente é passível de ser destruído de forma controlada.

Esta possibilidade de eliminar a infraestrutura é relevante no contexto académico deste cenário, uma vez que contribui para a redução de custos associados à utilização de recursos na AWS, permitindo um uso mais eficiente e responsável dos serviços em nuvem.

Com o objetivo de reforçar a segurança do *pipeline* de integração contínua, foram configurados *Repository secrets* no *GitHub Actions*, conforme ilustrado na Figura 5.5. Estes *secrets*, armazenam informações sensíveis de forma segura, reduzindo o risco de exposição accidental e de acessos indevidos durante a execução dos *workflows*.

A Figura 5.5 ilustra a configuração dos *Repository secrets* no *GitHub Actions*, utilizados para armazenar credenciais e variáveis sensíveis de forma segura, assegurando a proteção dos *pipelines* de integração contínua.



Repository secrets		New repository secret
Name ↕↑	Last updated	
ACCESS_GITHUB_SECRETS	3 weeks ago	
AWS_ASSUME_ROLE_ARN	3 weeks ago	
SONARQUBE_SECRETS	3 weeks ago	

Figura 5.5: Configuração de *secrets* no *GitHub Actions*

Na configuração do repositório, foram definidos *secrets* como:

- ACCESS_GITHUB_SECRETS
- AWS_ASSUME_ROLE_ARN
- SONARQUBE_SECRETS

Estes segredos incluem credenciais de acesso à AWS, tokens de autenticação para o *SonarQube* e outras variáveis sensíveis necessárias para o funcionamento seguro dos *workflows*. A utilização de *secrets* previne a exposição de credenciais no código-fonte e nos registos de execução, mitigando riscos como o *secret leakage* [71] e o *secret sprawl* [72], situações que exploram a divulgação não intencional de credenciais.

Adicionalmente, foi implementada uma política de rotação periódica de chaves, pode ocorrer a cada 30, 60 ou, no máximo, 90 dias, reduzindo assim a janela de exposição em caso de comprometimento.

Terraform

O *workflow* do *GitHub Actions* foi desenvolvido para automatizar a implementação de infraestrutura na AWS utilizando o *Terraform*. O processo está dividido em duas fases principais: *setup* e *apply*.

Na fase de *setup*, o *workflow* realiza as seguintes operações:

- Assume uma *role* da AWS com permissões necessárias, utilizando OIDC.
- Cria um *S3 Bucket*, destinado ao armazenamento do *Terraform state*, que mantém o registo do estado atual da infraestrutura criada ou mantida, possibilitando a persistência e a partilha segura das informações de estado entre diferentes execuções.

- Cria uma tabela *DynamoDB*, para garantir o bloqueio (*lock*) do estado durante a execução do *Terraform*, prevenindo conflitos e corrupções que possam ocorrer em caso de execuções concorrentes.

Na fase de *apply*, o *workflow* executa:

- Instalação da versão especificada do *Terraform*.
- Inicialização do ambiente *Terraform* (`terraform init`), utilizando o *backend* *S3* configurado para gestão do estado e a tabela *DynamoDB* para controlo de *lock*.
- Geração de um plano de execução (`terraform plan`) e aplicação do mesmo (`terraform apply`).
- Criação de um ficheiro `id_rsa` [73] localmente, contendo a chave privada decodificada e configurada com permissões de segurança restritivas.

SonarQube

O *workflow* do *SonarQube* no *GitHub Actions* foi desenvolvido para executar uma análise de qualidade e segurança de código utilizando o *SonarCloud*.

Este processo possibilita avaliar a qualidade do código, identificando vulnerabilidades e *code smells*, promovendo a adoção de práticas de desenvolvimento seguro e a melhoria contínua da qualidade do *software*.

5.1.5 *Terraform*

Esta secção apresenta o diagrama utilizado no cenário da infraestrutura desenvolvida, destacando a sua estrutura modular. Para orquestrar este processo, a infraestrutura foi desenvolvida com recurso à ferramenta *Terraform*, devido à sua capacidade de modularização e da facilidade de integração com os serviços da AWS.

A infraestrutura foi desenvolvida na *cloud* da AWS, na região `us-east-1`, onde foi criada uma VPC configurada com alta disponibilidade e segmentação de rede. Para assegurar a resiliência e a continuidade do serviço, a VPC foi distribuída por duas *Availability Zones* distintas (`us-east-1a` e `us-east-1b`). Esta distribuição geográfica permite mitigar o impacto de falhas em datacenters individuais, promovendo uma arquitetura tolerante a falhas e alinhada com as boas práticas de alta disponibilidade.

Foram definidos os seguintes componentes:

- Rede VPC:
 - Duas sub-redes públicas, associadas a um *Internet Gateway*, permitindo o acesso direto à *Internet*.

- Duas sub-redes privadas, com acesso controlado à *Internet* através de uma *NAT Gateway* e respetiva *Route Table*.
- Tabelas de *routing* para gestão do tráfego de rede.
- Balanceamento de carga (*Load Balancer* e *Target Groups*).
- Instâncias e escalabilidade *Auto Scaling Group* (ASG) e *Launch Templates*).
- Segurança (*Security Groups* e *roles IAM*).
- *Backend Terraform*:
 - *Backend S3*
 - *DynamoDB*

O processo de provisionamento da infraestrutura segue o fluxo ilustrado na Figura 5.6:

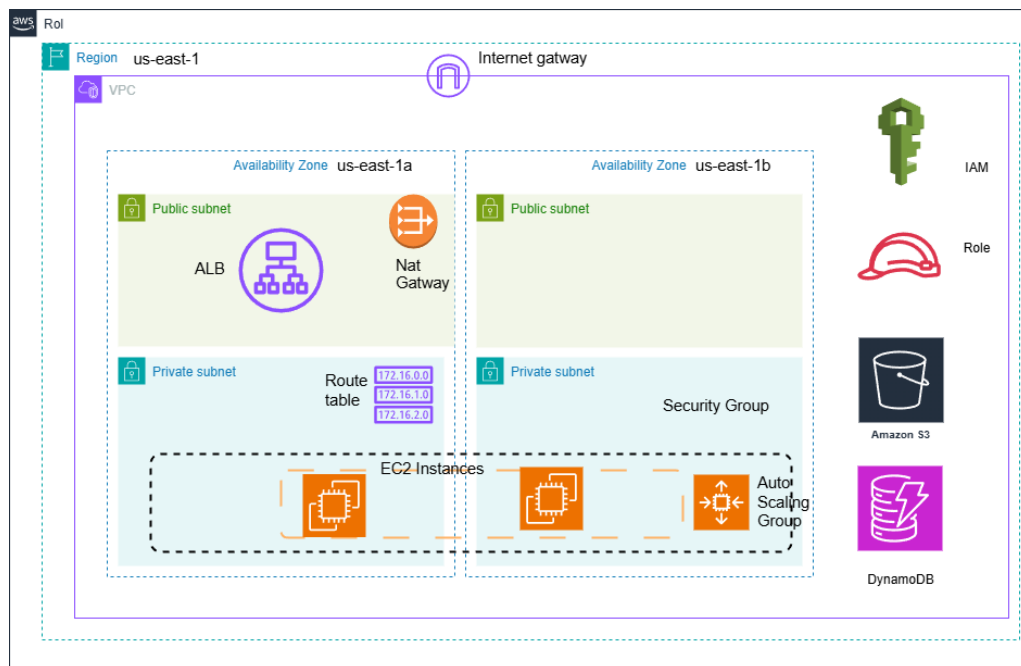


Figura 5.6: Arquitetura da infraestrutura provisionada com *Terraform*

A Figura 5.6 apresenta os componentes da infraestrutura provisionada com *Terraform* na AWS.

O tráfego de entrada é encaminhado através de um *Load Balancer* público do tipo *Application Load Balancer*, configurado para operar exclusivamente sobre o protocolo Hypertext Transfer Protocol Secure (HTTPS). O acesso é realizado por intermédio de um endereço DNS público temporário, atribuído dinamicamente pela AWS (por exemplo, `https://app-1b-632836935.us-east-1.elb.amazonaws.com`) e protegido por um certificado digital *self-signed* gerado através do serviço AWS Certificate Manager (ACM) da AWS.

O *Load Balancer* distribui os pedidos recebidos para um ASG, o qual mantém entre as duas instâncias EC2. Estas instâncias estão distribuídas por duas sub-redes privadas distintas — uma em cada *Availability Zone* (*us-east-1a* e *us-east-1b*) — garantindo, assim, uma elevada disponibilidade e tolerância a falhas. A arquitetura multi-zona assegura a continuidade do serviço mesmo em caso de interrupção numa das zonas de disponibilidade, reforçando a resiliência da infraestrutura.

Estas instâncias encontram-se associadas a *Security Groups* com políticas restritivas: apenas permitem acesso à porta 443 (HTTPS), proveniente do grupo de segurança do *Load Balancer*, assim como uma porta 22 (SSH) para gestão remota, posteriormente utilizada pelo *Ansible*. O tráfego de saída é autorizado na totalidade. A ligação à *Internet* a partir das sub-redes privadas é estabelecida através de um *NAT Gateway*, que utiliza um endereço IP elástico e está associado à *Subnet* Pública.

Por fim, a infraestrutura é complementada com a geração automática de inventário para ser usado pelo *Ansible*, o qual recolhe dinamicamente os endereços IP privados das instâncias lançadas, permitindo a posterior configuração automatizada. Esse mesmo inventário é armazenado e transferido para um *Bucket S3*, onde ficam igualmente alojadas as chaves privadas necessárias para a autenticação.

5.1.6 *Ansible*

Esta secção apresenta o diagrama utilizado no cenário descrito na Secção 5.1.5, no qual a infraestrutura foi criada através do *Terraform*, incluindo a provisão de uma instância EC2. Nesta instância são aplicadas as configurações definidas no *Playbook* do *Ansible*, que permite orquestrar tarefas de instalação, configuração e *hardening* de servidores, assegurando reprodutibilidade e eliminando erros manuais [16].

A execução deste passo é despoletada através da ação "Deploy EC2 com Ansible", descrita na Figura 5.5.

Fluxo de execução do *Playbook*

O *Playbook* implementa o seguinte fluxo de operações:

- Atualizar pacotes: Atualização dos repositórios e pacotes de sistema.
- Instalação de serviços: Instalação do *Apache* [74], *OpenSSL* [75], *Python 3* [76], *Certbot* [77] e módulos Web Server Gateway Interface (WSGI) [78].
- Ativação de módulos *Apache*: Ativação dos módulos *Secure Sockets Layer* (SSL) e *Common Gateway Interface* (CGI).
- Criação de certificado SSL: Geração de um certificado autoassinado para encriptação HTTPS.

- Criação de diretórios e páginas:
 - Diretório `/var/www/html/cgi-bin` para scripts CGI.
 - Página principal `index.html`.
- Desenvolvimento de *script CGI*: Script Python que apresenta o `hostname` e o `uptime` do servidor.
- Configuração do *VirtualHost HTTPS*: Configuração segura para escutar na porta 443 e servir o conteúdo protegido por SSL.
- Ativação e validação da configuração: Ativação do site SSL e verificação da integridade da configuração *Apache*.
- *Firewall Uncomplicated Firewall (UFW)*: Abertura da porta 443 (HTTPS) e bloqueio da porta 80 (HTTP).
- Serviço *Apache*: Ativação do serviço *Apache* para garantir a sua execução em reinícios do sistema.

A Figura 5.7 apresenta os componentes da infraestrutura provisionada com *Terraform* na AWS incluindo a infraestrutura provisionada para o *Ansible*

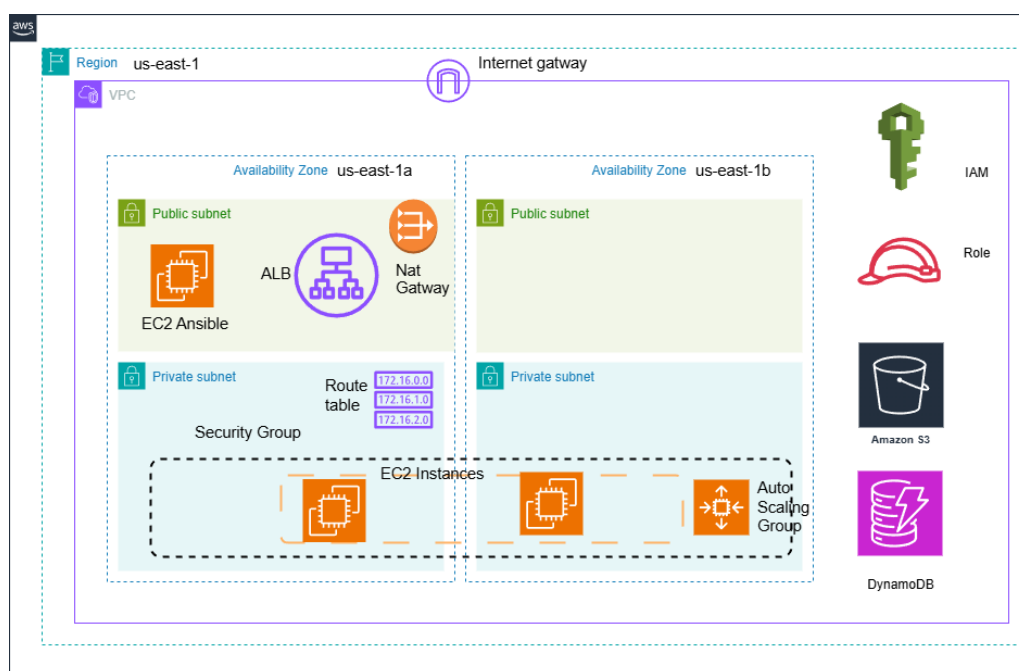


Figura 5.7: *Workflow* de configuração automatizada com *Ansible*

A infraestrutura automatizada recorre à integração entre *Terraform* e *Ansible* para aprovisionar os recursos na AWS, e para configurar, de forma orquestrada, as instâncias EC2 lançadas. Após a criação das instâncias por meio de um *Auto Scaling Group*, o *Terraform* identifica os respectivos endereços IP privados e gera dinamicamente um ficheiro de *inventory* no formato interpretado pelo *Ansible*.

Este ficheiro de inventário é produzido localmente durante a execução do *GitHub Actions* e inclui os parâmetros de conexão SSH necessários para permitir a automatização da configuração remota, como o utilizador (*ubuntu*) e a chave privada gerada. O ficheiro é posteriormente transferido para o *Bucket S3*, onde fica também armazenada a chave privada gerada automaticamente, associada ao par de chaves *RSA (4096 bits)*.

Com o inventário disponível, o *playbook* do *Ansible* pode ser executado para aplicar as tarefas de configuração nas instâncias EC2. Este *playbook* define a instalação de um servidor *nginx*, bem como configuração uma página *web* personalizada, a qual exhibe dinamicamente o IP público, hostname, data/hora e tempo de atividade da máquina EC2 em que se encontra.

Deste modo, a articulação entre *Terraform* e *Ansible* possibilita uma entrega contínua e totalmente automatizada de infraestrutura e serviços, reduzindo a necessidade de intervenção manual e garantindo a consistência das configurações aplicadas em todos os componentes da infraestrutura.

5.2 SonarQube

Para complementar a segurança do ciclo de desenvolvimento e garantir a adoção de boas práticas de codificação segura, foi integrada no processo de CI/CD uma etapa de análise estática de código fonte (SAST), utilizando a plataforma *SonarQube*. A implementação desta integração está detalhada na Secção 5.1.4, onde se descreve a configuração do *pipeline* e a forma como a análise de qualidade de código foi automatizada.

A Figura 5.8 apresenta a configuração do *Security Group* criada com *Terraform*, destacando as regras de acesso definidas e os alertas de segurança identificados pela ferramenta *SonarQube*.

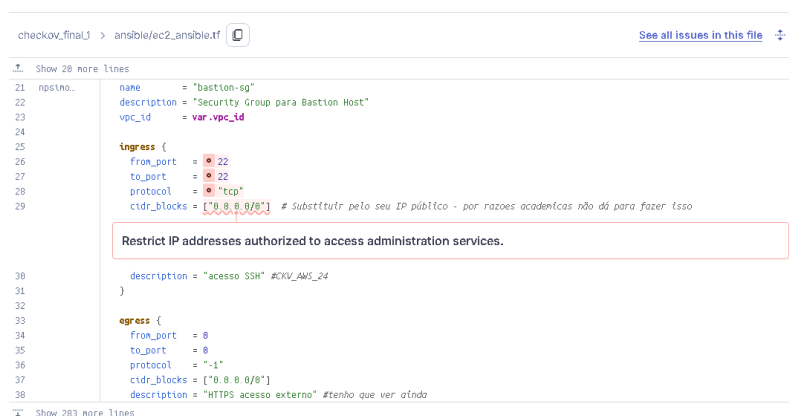


Figura 5.8: Identificação de erros pelo *SonarQube*

A análise SAST foi automatizada através de um *workflow* dedicado no *GitHub Actions*, sendo desencadeado manualmente via evento *workflow_dispatch* e descrito na Figura 5.5

Em seguida, é utilizado o recurso `sonarqube-scan-action` disponibilizado pela plataforma, para enviar o código para análise no *SonarCloud*, um serviço baseado na plataforma *SonarQube*.

No nosso cenário, apenas estas entradas foram assinaladas pelo *SonarQube* como relevantes, o que acabou por se revelar bastante dececionante, considerando as expectativas elevadas relativamente à capacidade de deteção desta ferramenta.

5.3 Checkov

A avaliação contínua de segurança em configurações de infraestrutura é fundamental para prevenir vulnerabilidades antes da implantação em ambientes de produção. Ferramentas especializadas para análise de IaC tornam-se essenciais para a deteção de más práticas e garantir conformidade com padrões de segurança reconhecidos. Neste contexto, foi integrada a ferramenta *Checkov* [16].

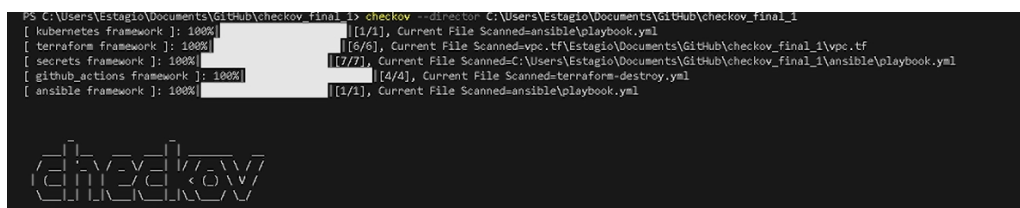
5.3.1 Objetivo

O *Checkov* foi integrado no processo com o objetivo de:

- Detetar más configurações e vulnerabilidades em ficheiros *Terraform*.
- Destacar a necessidade de reforçar a análise de segurança da infraestrutura como código, dado que o *SonarQube* apresentou resultados insatisfatórios na deteção de vulnerabilidades..

5.3.2 Configuração

A execução do *Checkov* foi realizada através do seguinte comando:



```
PS C:\Users\Estagio\Documents\GitHub\checkov_final_1> checkov --directory C:\Users\Estagio\Documents\GitHub\checkov_final_1
[ kubernetes framework ]: 100% [1/1], Current File Scanned=ansible\playbook.yml
[ terraform framework ]: 100% [6/6], Current File Scanned=vpc.tf\Estagio\Documents\GitHub\checkov_final_1\vpc.tf
[ secrets framework ]: 100% [7/7], Current File Scanned=C:\Users\Estagio\Documents\GitHub\checkov_final_1\ansible\playbook.yml
[ github_actions framework ]: 100% [4/4], Current File Scanned=terraform-destroy.yml
[ ansible framework ]: 100% [1/1], Current File Scanned=ansible\playbook.yml

checkov
```

Figura 5.9: Comando de execução da ferramenta *Checkov*

A Figura 5.9 ilustra a execução do *Checkov* para análise de diferentes ficheiros de configuração de infraestrutura.

Em contraste com o *SonarQube*, o *Checkov* revelou-se uma mais-valia substancial na análise de infraestrutura como código. A ferramenta detetou diversas más práticas não identificadas, incluindo permissões excessivas, ausência de encriptação e falta de registos de atividades (*logging*) em componentes críticos. A sua

utilização permitiu definir um conjunto de boas práticas de segurança, posteriormente integrado no cenário.

A Figura 5.10 apresenta o resultado da análise de segurança realizada pelo *Checkov*, destacando a conformidade dos recursos configurados em *Terraform* com boas práticas de segurança.

```
Check: CKV_AWS_328: "Ensure that ALB is configured with defensive or strictest desync mitigation mode"
PASSED for resource: aws_lb.app_lb
File: \ec2.tf:145-159
Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-networking-policies/bc-aws-328
Check: CKV_AWS_131: "Ensure that ALB drops HTTP headers"
PASSED for resource: aws_lb.app_lb
File: \ec2.tf:145-159
Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-networking-policies/ensure-that-alb
Check: CKV_AWS_91: "Ensure the ELBv2 (Application/Network) has access logging enabled"
PASSED for resource: aws_lb.app_lb
File: \ec2.tf:145-159
Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-logging-policies/bc-aws-logging-22
Check: CKV_AWS_261: "Ensure HTTP HTTPS Target group defines Healthcheck"
PASSED for resource: aws_lb_target_group.alb_ec2_tg
File: \ec2.tf:169-187
Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-general-policies/ensure-aws-kendra-ses-customer-managed-keys-cmk
```

Figura 5.10: Identificação de verificações pelo *Checkov*

A integração do *Checkov* foi determinante para aumentar o nível de segurança e a maturidade da infraestrutura como código. Esta experiência reforça a importância de uma escolha criteriosa das ferramentas de análise, ajustada às necessidades específicas de cada contexto.

5.4 Cenário final

No seguimento do que foi descrito na Secção 5.1.5, este capítulo descreve a infraestrutura final desenvolvida com recurso a tecnologias IaC, detalhando a sua composição, os mecanismos de segurança integrados e as boas práticas aplicadas durante o processo de aprovisionamento.

Ao longo deste capítulo, são apresentados os principais recursos criados e descritos os componentes finais da infraestrutura provisionada com o *Terraform*, com especial destaque para os aspetos relacionados com a segurança e a resiliência operacional, analisados com o *Checkov*.

O processo final do aprovisionamento da infraestrutura segue o fluxo ilustrado na Figura 5.11.

Estrutura final do cenário do *Terraform*

- VPC
 - VPC Flow Logs ativado
 - * Destino: *CloudWatch Logs*
 - * Protegido com KMS com rotação de chaves e política explícita (CKV2_AWS_64)
 - * Retenção dos logs por 365 dias (CKV_AWS_338)

- * *IAM Role* com política restrita à escrita no grupo de *logs* (CKV_AWS_355)
- Grupos de Segurança
 - *Load Balancer* (alb-sg)
 - * *Ingress*: porta 443 (HTTPS) de qualquer origem (CKV_AWS_23)
 - * *Egress*: todo o tráfego permitido com justificação (CKV_AWS_23 + CKV_AWS_382)
 - *EC2 Web Server* (ec2-sg)
 - * *Ingress*:
 - Porta 22 (SSH) restrita ao *CIDR* da VPC 10.0.0.0/16 (CKV_AWS_24)
 - Porta 443 (HTTPS) apenas do *alb-sg* (CKV_AWS_23)
 - * *Egress*: total (CKV_AWS_23 + CKV_AWS_382)
- Load Balancer (HTTPS)
 - *Application Load Balancer*
 - HTTPS (porta 443)
 - Certificado *self-signed* com validade de 1 ano
 - Política de segurança TLS forçada: *ELBSecurityPolicy-TLS-1-2-2017-01* (CKV_AWS_103)
 - Registo de *logs* de acesso ativado (CKV_AWS_91)
 - * *Bucket S3* com política de escrita para o serviço *ELB*
 - *drop_invalid_header_fields* = *true* para proteção extra (CKV_AWS_131)
- WAF
 - *Web ACL* associado ao *Application Load Balancer* (ALB) (CKV2_AWS_28)
 - Regra 1: *AWSManagedRulesCommonRuleSet*
 - * Proteção contra *SQLi*, *XSS*
 - Regra 2: *AWSManagedRulesKnownBadInputsRuleSet*
 - * Proteção contra *exploits* conhecidos, como *Log4Shell*
 - Métricas ativadas com envio para *CloudWatch*
- EC2 Launch Template
 - Metadados com *http_tokens* = "required" (CKV_AWS_79)
 - Protege contra ataques de *SSRF* e reforça segurança da instância
- Inventário e Chave SSH
 - Inventário dinâmico gerado e encriptado
 - Ambos armazenados no *Buket S3* com encriptação *AES256*

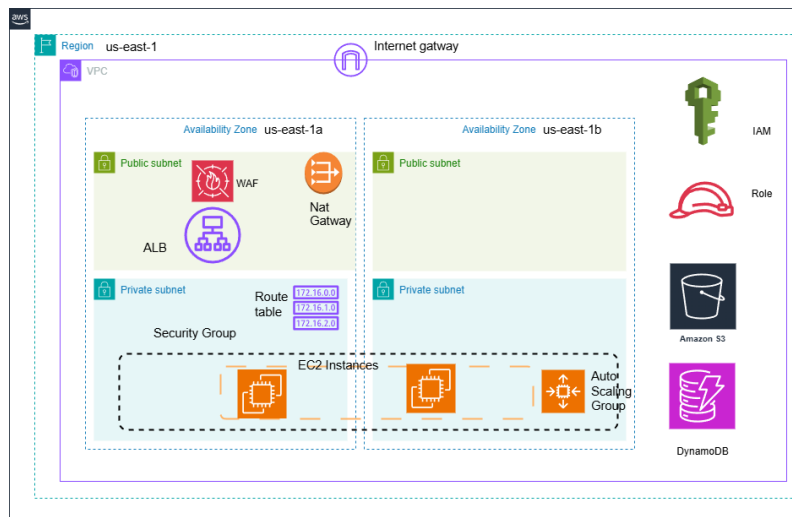


Figura 5.11: Arquitetura final da infraestrutura provisionada com *Terraform*

A Figura 5.11 apresenta os componentes da infraestrutura final da provisionada com *Terraform* na AWS após análise do *Checkov*.

Com o objetivo de reforçar a segurança da infraestrutura, foram integrados mecanismos de monitorização, controlo de acesso e proteção contra ameaças. Em conjunto, estas medidas asseguram uma infraestrutura segura, resiliente e alinhada com os princípios fundamentais de segurança.

No que respeita à camada de configuração automatizada pelo *Ansible*, não foram identificadas recomendações ou alertas.

5.5 Sumário

O capítulo descreve o cenário desenvolvido e as atividades realizadas no contexto da dissertação. O cenário utiliza o *Terraform* para o aprovisionamento, o *Ansible* para a gestão da infraestrutura e o *GitHub Actions* como ferramenta de automação e gestão centralizada. Este *workflow* é complementado pelo *SonarQube* e *Checkov*, que realizam análise de segurança do código.

Foram integradas ferramentas de análise estática de código, como o *SonarQube* e o *Checkov*, sendo que este último demonstrou maior eficácia na deteção de más práticas de configuração e possibilitou a aplicação de melhorias significativas à segurança da infraestrutura.

Na configuração do IAM na AWS, foi implementado o princípio do menor privilégio, garantindo que apenas o utilizador necessário possui as permissões adequadas. Por fim, foi utilizada a plataforma *GitHub* como ambiente de gestão de código-fonte.

Capítulo 6

Resultados observados

Neste capítulo, apresenta-se a análise dos resultados obtidos no cenário desenvolvido, com destaque para a identificação de problemas e más práticas associadas à utilização de IaC. Inicialmente, apresenta-se uma lista de situações potencialmente identificáveis, com base na análise do estado da arte. De seguida, identificam-se os problemas através da avaliação prática realizada com uma ferramenta de análise de código estática SAST, nomeadamente o *Checkov*. Os resultados obtidos com a ferramenta *SonarQube* não são apresentados, uma vez que não foram considerados relevantes.

6.1 Possíveis problemas a observar — SOA

A análise do estado da arte permitiu sistematizar diversos problemas recorrentes na utilização de IaC, os quais serviram de referência para a avaliação crítica do cenário desenvolvido.

A Tabela 6.1 apresenta uma compilação dos principais *smells* e más práticas observados.

Problema / Smell	Descrição resumida
Segredos codificados	Inclusão de senhas, <i>tokens</i> ou chaves diretamente nos <i>scripts</i> IaC. [11, 19]
Utilização de HTTP sem TLS	<i>Endpoints</i> configurados sem encriptação segura (sem HTTPS). [19]
Falta de idempotência	Execuções sucessivas não produzem o mesmo estado. [70]
Recursos com permissões excessivas	Definições como “*” em regras de acesso (ex.: AWS e IAM). [51]
Módulos não reutilizáveis	Falta de parametrização ou <i>hardcoding</i> . [66]
Infraestrutura órfã / <i>zombies</i>	Recursos não utilizados e esquecidos na <i>cloud</i> . [5]

Problema / Smell	Descrição resumida
Comentários suspeitos	Comentários que indicam vulnerabilidades ou falhas por corrigir. [19]
<i>Scripts</i> sem validação / testes	Ausência de qualquer tipo de teste automatizado ou validação. [58]
Abuso de variáveis globais	Uso indiscriminado de variáveis partilhadas sem encapsulamento. [69]
<i>Smells</i> de encapsulamento deficiente	Múltiplos recursos definidos num único bloco sem modularidade. [18]
Dependências não documentadas	<i>Scripts</i> assumem dependências externas sem referência. [9]
Código denso e não legível	Falta de formatação, comentários ou organização. [66]
<i>Shadow IT / Deploys</i> manuais	<i>Scripts</i> paralelos ou alterações manuais fora do ciclo CI/CD. [5]
Dados de configuração duplicados	Repetição de valores em múltiplos ficheiros ou blocos. [70]
Erros de sintaxe silenciosos	Erros que passam despercebidos até tempo de execução. [57]
Ausência de controlo de versões	<i>Scripts</i> sem versões ou modificados fora de VCS. [54]
Falta de validação de integridade	Falta de ‘ <i>hashes</i> ’ para <i>downloads</i> de binários. [11]
Ambientes de execução indefinidos	Não especificar versões de ferramentas, <i>providers</i> , etc. [2]
Dependência de IPs fixos	Utilização de IPs <i>hardcoded</i> em regras de <i>firewall</i> . [19]
Inexistência de <i>rollback</i> automático	Falta de planeamento em caso de falha de ‘ <i>apply</i> ’. [58]

Tabela 6.1: Problemas e más práticas em IaC

A Tabela 6.1 serve de base para a comparação com os problemas identificados durante a verificação do cenário, realizada com recurso a ferramentas automáticas como o *Checkov*. Esta abordagem permitiu mapear falhas práticas em relação às más práticas teoricamente conhecidas, revelando que várias destas se encontravam presentes no código original antes das intervenções corretivas.

6.2 Resultados CVK

6.2.1 Identificação CVK

Os identificadores *CKV_AWS_XXX* são códigos atribuídos a verificações de segurança específicas aplicadas pela ferramenta *Checkov* em configurações de infraestrutura como código IaC destinadas à AWS. A ferramenta *Checkov*, realiza análises estáticas sobre ficheiros de configuração como *Terraform*, com o objetivo de identificar configurações inseguras ou não conformes com as boas práticas de segurança na *cloud*.

Cada código *CKV_AWS_XXX* representa uma regra individual que define uma determinada verificação. Por exemplo, o *CKV_AWS_20* corresponde a uma verificação que assegura que os *buckets S3* têm encriptação do lado do servidor ativada, enquanto o *CKV_AWS_57* identifica permissões excessivas atribuídas em políticas IAM que permitem ações sobre todos os recursos (""). A utilização destes identificadores facilita o rastreio e a documentação dos problemas identificados.

Estas verificações são utilizadas em *pipelines* CI/CD, verificações de segurança e revisões de código, para garantir que a infraestrutura declarada respeita princípios como o princípio do menor privilégio, o controlo de acesso restrito e a proteção de dados sensíveis [79]. Além disso, os identificadores permitem ignorar verificações específicas de forma seletiva, quando devidamente justificado, utilizando diretivas *inline* (ex.: *#checkov:skip=CKV_AWS_20*) [80].

Ao fornecer mensagens de erro associadas a estes identificadores, o *Checkov* ajuda programadores e equipas de segurança a localizar rapidamente vulnerabilidades na configuração e a aplicar as correções apropriadas. A maioria das regras é acompanhada por recomendações de mitigação, exemplos de código vulnerável e respetivas sugestões de correção, como se ilustra na figura 5.10.

6.2.2 Regras CVK identificadas

As regras listadas foram aplicadas sobre os ficheiros *Terraform* e revelaram várias não conformidades com os princípios de segurança em *cloud*. Esta análise deu origem a intervenções concretas e fundamentadas.

A Tabela 6.2 apresenta o relatório completo dos problemas identificados pela ferramenta *Checkov* na infraestrutura analisada.

Check ID	Recurso	Descrição da Regra	Problema Detetado
CKV-AWS-8	aws-instance.server-ansible	Volume EBS não encriptado	Adicionar <i>encrypted = true</i> no <i>root-block-device</i>
CKV_AWS_11	aws_vpc.vpc	Ativar VPC flow logs	Recurso <i>aws_flow_log</i> ausente

Check ID	Recurso	Descrição da Regra	Problema Detetado
CKV_AWS_12	aws_vpc.vpc	<i>Security group default</i> não restrito	Criar <i>aws_default_security_group</i> com regras revogadas
CKV_AWS_23	aws_security-group.nginx	<i>Security group</i> deve ter descrição	Ficheiro main.tf
CKV_AWS_24	aws_security_group.nginx	<i>No ingress from 0.0.0.0/0 to port 22</i>	Ficheiro main.tf
CKV_AWS_24	aws_security_group.nginx	<i>No ingress from 0.0.0.0/0 to port 22</i>	Ficheiro main.tf
CKV_AWS_28	aws_lb.app_lb	ALB público deve estar protegido com WAF	Falta associação com <i>WebACL</i>
CKV_AWS_31	my-acl	WAF sem <i>logging</i>	ec2.tf:374-404
CKV_AWS_41	aws_instance.web1	IAM role está ligada à instância EC2	Ficheiro main.tf
CKV_AWS_64	Corrigido	Chave KMS sem política definida	Adicionada <i>policy</i> no <i>aws-kms-key.cloudwatch_logs_key</i>
CKV_AWS_79	aws_instance.server_ansible	<i>Metadata Service V1</i> não ativado	ec2_ansible.tf
CKV_AWS_88	aws-instance.server_ansible	EC2 com IP público	ec2_ansible.tf
CKV_AWS_91	aws_lb.app_lb	ALB sem <i>access logging</i> ativado	Falta configuração de <i>access_logs</i>
CKV_AWS_103	aws_lb_listener.alb_listener	Usar TLS >= 1.2	Política obsoleta: <i>ELBSecurityPolicy-2016-08</i>
CKV_AWS_126	aws_instance.web1	<i>Monitoring</i> detalhado desativado	Ficheiro main.tf
CKV_AWS_130	aws_subnet.public_subnet_2	<i>Subnets</i> não devem ter IP público por omissão	Ficheiro vpc.tf
CKV_AWS_131	aws_lb.app_lb	O ALB deve remover headers HTTP inválidos	Falta <i>'drop_invalid_header_fields'</i>
CKV_AWS_135	aws_instance.server_ansible	EC2 não otimizado para EBS	ec2_ansible.tf
CKV_AWS_150	app_lb	<i>Load Balancer</i> sem proteção de apagamento	ec2.tf:145-159
CKV_AWS_158	vpc_flow_logs	<i>Log Group</i> sem encriptação KMS	vpc.tf:30-38

<i>Check ID</i>	Recurso	Descrição da Regra	Problema Detetado
CKV_AWS_192	aws_wafv2 _web_acl.my_acl	WAF não previne <i>Log4j2</i>	ec2.tf
CKV_AWS_260	aws_security _group.nginx	<i>Ingress</i> aberto a 0.0.0.0/0 na porta 80	Ficheiro main.tf
CKV_AWS_261	aws_lb_target _group.alb _ec2_tg	<i>Target group</i> sem <i>health-check</i> definido	Falta bloco <i>health-check</i>
CKV_AWS_288	aws_iam_role _policy.ansible _policy	IAM com permissões amplas	ec2_ansible.tf
CKV_AWS_290	aws_iam_role _policy.flow_logs _policy	IAM sem restrição de escrita	vpc.tf
CKV_AWS_338	aws_cloudwatch _log_group.vpc _flow_logs	<i>Log retention</i> inferior a 1 ano	vpc.tf
CKV_AWS_355	aws_iam_role _policy.flow_logs _policy	Política IAM permite '*' como recurso	vpc.tf
CKV_AWS_382	aws_security _group.bastion _sg	<i>Egress</i> aberto a 0.0.0.0/0 (todos os protocolos)	ec2_ansible.tf

Tabela 6.2: Relatório de problemas detetados pelo *Checkov*

Entre os problemas detetados e listados na Tabela 6.2, destacam-se algumas falhas críticas. A ausência de um WAF configurado (CKV_AWS_28) e mesmo depois da sua implementação, a incapacidade do WAF em mitigar a vulnerabilidade *Log4Shell* (CKV_AWS_192) as quais representavam riscos significativos e expunham a infraestrutura a ataques conhecidos, mas que foram posteriormente resolvidos, garantindo a conformidade e aumentando a segurança do ambiente.

Foram ainda identificados volumes de armazenamento e *logs* sem encriptação (CKV_AWS_8, CKV_AWS_158), bem como instâncias com IP público (CKV_AWS_88), aumentando a superfície de ataque, sendo que esta exposição se justifica pela utilização temporária do *Ansible* apenas para configurar o restante cenário. A ausência de registos de auditoria (CKV_AWS_91) e políticas IAM com permissões excessivas (CKV_AWS_288, CKV_AWS_355) foram igualmente apontadas como deficiências relevantes a corrigir. Estes resultados reforçam a importância da aplicação de boas práticas de segurança desde a fase de definição da infraestrutura.

6.2.3 Limitações

A análise da infraestrutura desenvolvida foi suportada por ferramentas especializadas que possibilitaram a identificação de vulnerabilidades e más práticas em configurações de IaC.

Nesta secção, são discutidas as principais condicionantes observadas durante o processo de avaliação, bem como as implicações destas restrições na eficácia da detecção de problemas de segurança.

- Necessidade de ajuste e reprogramação manual dos resultados;
- Cobertura limitada à análise estática.

Apesar do esforço na integração de ferramentas de análise estática de segurança, verificou-se que os resultados variam dependendo da ferramenta utilizada. Esta constatação reforça a importância de uma seleção cuidadosa das ferramentas de análise.

A Tabela 6.3 apresenta a associação entre os *findings* detetados pela ferramenta *Checkov* e as respetivas vulnerabilidades Common Vulnerabilities and Exposures (CVE) identificadas, com uma descrição sucinta de cada correspondência.

Checkov ID	CVE Associada	Descrição
CKV-AWS-23	CVE-2021-44228	A vulnerabilidade <i>Log4Shell</i> pode ser explorada em configurações de grupos de segurança mal configurados, permitindo a execução remota de código.

Tabela 6.3: Associação entre *findings* do *Checkov* e CVE detetadas

O CVE-2021-44228 foi a única vulnerabilidade de relevância identificada no cenário analisado pela ferramenta *Checkov*.

6.3 Análise dos resultados

Apesar de a infraestrutura ter sido desenvolvida com atenção às melhores práticas recomendadas para a utilização de IaC, a análise detalhada realizada com a ferramenta *Checkov* revelou um conjunto significativo de falhas de configuração que poderiam comprometer a segurança do cenário implementado. Esta constatação realça a complexidade inerente à gestão de segurança em ambientes automatizados e destaca a importância da utilização de ferramentas específicas de análise de código para validação contínua.

A ferramenta *Checkov* revelou-se eficiente, identificando 27 verificações de segurança distintas (*checks*) e propondo medidas concretas para aumentar a segurança

do cenário desenvolvido. Todos os problemas identificados foram corrigidos de forma iterativa, à medida que surgiam, seguindo uma abordagem de melhoria contínua. A análise permitiu validar a eficácia das práticas inicialmente implementadas, ao mesmo tempo que expôs vulnerabilidades críticas que, de outra forma, poderiam ter passado despercebidas.

Por outro lado, a utilização da ferramenta *SonarQube* revelou-se insatisfatória no contexto deste trabalho. Embora corretamente integrada no pipeline CI/CD, a ferramenta apresentou limitações na análise de ficheiros *Terraform* e *Ansible*, oferecendo um *feedback* pouco relevante para a correção dos problemas identificados.

Segue-se a síntese dos problemas encontrados, destacando não só a sua natureza e impacto, mas também as medidas corretivas implementadas para reforçar a segurança da infraestrutura:

- Armazenamento não encriptado (CKV-AWS-8, CKV_AWS_158): Foram detetados volumes EBS e grupos de *logs CloudWatch* sem encriptação através de KMS. Esta ausência de cifragem expõe os dados em repouso, violando boas práticas de proteção da informação sensível.
- Ausência de mecanismos de monitorização e *logging* (CKV_AWS_11, CKV_AWS_31, CKV_AWS_91): Recursos críticos como VPC e ALB não tinham de registos de auditoria, impedindo a rastreabilidade e dificultando a deteção de acessos ou comportamentos anómalos.
- Regras de segurança permissivas em grupos (CKV_AWS_24, CKV_AWS_260, CKV_AWS_382): Foram encontradas configurações que permitem tráfego de entrada sem restrição (por exemplo, de 0.0.0.0/0 nas portas 22 e 80) e regras de saída abertas a todos os destinos e protocolos. Estas configurações representam riscos sérios de exposição e ataque externo.
- Falta de restrições em políticas IAM (CKV_AWS_288, CKV_AWS_290, CKV_AWS_355): Diversas políticas atribuídas a funções IAM concediam permissões excessivamente abrangentes, incluindo o uso de * como recurso, violando o princípio do menor privilégio.
- Deficiências em proteção de aplicações web (CKV_AWS_28, CKV_AWS_192): O ALB encontrava-se exposto publicamente sem a proteção de um WAF e foi também detetado que ele não implementava mitigação adequada à vulnerabilidade *Log4Shell* (CVE-2021-44228), representando uma ameaça crítica com possibilidade de execução remota de código.
- Configurações fracas de segurança em EC2 (CKV_AWS_79, CKV_AWS_88, CKV_AWS_135): As instâncias EC2 analisadas revelaram várias debilidades como ausência do *Instance Metadata Service v2*, presença de IP público e não estarem otimizadas para EBS, o que reduz a resiliência e aumenta a superfície de ataque.
- Omissão de controlos de integridade e desempenho (CKV_AWS_126, CKV_AWS_261, CKV_AWS_338): Incluem-se falhas como ausência de *health checks* em *target groups*, retenção de *logs* inferior a 1 ano e desativação de monitorização detalhada em instâncias EC2.

Algumas destas configurações não foram completamente resolvidas e continuam a ser assinaladas, uma vez que se trata de um cenário académico, sem IP público fixo, DNS públicos ou outras configurações de registo de *logs*, com o objetivo de evitar custos elevados na AWS. O cenário foi concebido com a possibilidade de ser criado e posteriormente destruído na totalidade, situação que não ocorreria num ambiente de produção real e permanente.

6.3.1 Classificação por gravidade

Os problemas detetados foram classificados de acordo com a sua gravidade, com base no impacto potencial na segurança:

- Gravidade alta: 5 problemas
- Gravidade média: 15 problemas
- Gravidade baixa: 7 problemas

A classificação detalhada encontra-se listada a seguir:

Gravidade alta (5 problemas)

- CKV_AWS_28: ALB exposto sem proteção por WAF.
- CKV_AWS_192: WAF configurado sem mitigação para *Log4Shell* (CVE-2021-44228).
- CKV_AWS_88: Instâncias EC2 com IP- público.
- CKV_AWS_355: Política IAM permite * como recurso (permissões globais).
- CKV_AWS_8 / CKV_AWS_158: Volumes EBS e grupos de *logs CloudWatch* sem encriptação (dados em repouso não protegidos).

Gravidade média (15 problemas)

- CKV_AWS_11: VPC sem *flow logs* (ausência de registos de auditoria de rede).
- CKV_AWS_31: WAF sem *logging* ativado.
- CKV_AWS_91: *Load Balancer* sem *logging* de acesso ativado.
- CKV_AWS_24: *Security group* com acesso SSH (porta 22) aberto para 0.0.0.0/0.
- CKV_AWS_260: *Security group* com acesso HTTP (porta 80) aberto para 0.0.0.0/0.

- CKV_AWS_382: *Security group* com *egress* sem restrição 0.0.0.0/0, todos os protocolos).
- CKV_AWS_288: Política IAM com permissões excessivas.
- CKV_AWS_290: Política IAM sem restrição de escrita.
- CKV_AWS_79: Instância EC2 sem *Instance Metadata Service v2*.
- CKV_AWS_135: Instância EC2 não otimizada para EBS.
- CKV_AWS_150: *Load Balancer* sem proteção contra apagamento.
- CKV_AWS_126: Instância EC2 sem monitorização detalhada.
- CKV_AWS_261: *Target group* sem *health check* definido.
- CKV_AWS_338: *Log group* com retenção inferior a 1 ano.
- CKV_AWS_131: *Load Balancer* não configura remoção de *headers* HTTP inválidos.

Gravidade baixa (7 problemas)

- CKV_AWS_23: *Security group* sem descrição.
- CKV_AWS_12: *Security group default* não restrito (regras padrão permissivas).
- CKV_AWS_64: Chave KMS sem política definida (corrigido).
- CKV_AWS_41: IAM role associada a instância EC2 diretamente (prática desaconselhada, mas não crítica).
- CKV_AWS_103: *Load Balancer* a usar política TLS obsoleta (precisa de atualização para $TLS \geq 1.2$).
- CKV_AWS_130: *Subnet* pública com IP público por omissão.
- CKV_AWS_261: *Target group* sem *health check* definido (impacta alta disponibilidade, mas não segurança imediata).

A ferramenta *Checkov* mostrou eficácia na deteção de vulnerabilidades conhecidas e problemas de configuração em IaC, proporcionando uma análise rigorosa e alinhada com as boas práticas de segurança. Por outro lado, o *SonarQube*, apesar da correta integração no pipeline CI/CD, revelou-se menos adequado ao contexto específico deste trabalho.

6.4 Sumário

Este capítulo apresentou uma análise dos resultados obtidos no cenário desenvolvido, com foco na identificação de vulnerabilidades e más práticas associadas à utilização de ferramentas de IaC.

A análise permitiu identificar 27 falhas de configuração, classificadas por gravidade e cujas correções foram implementadas de forma iterativa. A comparação entre as ferramentas revelou que o *Checkov* demonstrou maior eficácia do que o *SonarQube* no contexto da análise realizada.

Os resultados obtidos evidenciam a importância da utilização de ferramentas específicas de análise de segurança no desenvolvimento de infraestrutura como código, bem como da adoção de práticas de DevSecOps para garantir a robustez e conformidade da infraestrutura.

Capítulo 7

Conclusão

Os resultados obtidos ao longo desta dissertação reforçam a relevância de integrar práticas de DevSecOps no desenvolvimento e gestão de infraestrutura como código IaC. Um dos principais objetivos consistiu em evidenciar que a automação, suportada por ferramentas como o *Terraform* e o *Ansible*, aliada a *pipelines* de integração e entrega contínuas (CI/CD), pode melhorar significativamente a eficiência operacional e a segurança de ambientes em nuvem [2] [50] [16].

A análise realizada com a ferramenta *Checkov* permitiu identificar 27 falhas de configuração, as quais foram posteriormente classificadas por gravidade e que potencialmente comprometiam a segurança e a conformidade do ambiente. Esta abordagem não só validou os objetivos iniciais da dissertação, como também destacou a complexidade e os riscos inerentes ao uso de IaC em contextos reais, mesmo quando se seguem boas práticas recomendadas [11] [60] [51].

O processo iterativo de correção das vulnerabilidades detetadas reforçou a importância de uma validação contínua, confirmando que ferramentas específicas de análise de código estático, como o *Checkov*, assumem um papel importante na garantia da segurança e da conformidade da infraestrutura [36] [79].

Os resultados obtidos evidenciam a necessidade de adoção de práticas rigorosas de segurança desde as fases iniciais do ciclo de vida da infraestrutura, promovendo uma postura proativa face a ameaças emergentes e fortalecendo a resiliência global do sistema [52] [53].

Deste modo, esta dissertação evidencia a importância da integração de ferramentas de análise de segurança no *pipeline* de desenvolvimento e da adoção de metodologias DevSecOps, fundamentais para garantir a robustez, a conformidade e a segurança contínua em ambientes baseados em *cloud*, contribuindo para operações mais seguras, eficientes e confiáveis.

Trabalho futuro

No futuro, poderá ser realizada uma avaliação mais extensiva do cenário proposto, incluindo a sua aplicação em contextos empresariais reais, de modo a medir o impacto na redução de vulnerabilidades, na melhoria da eficiência operaci-

onal e no aumento da conformidade com normas de segurança. A integração de práticas complementares, como o uso de DAST e a expansão da cobertura com outras ferramentas de análise, constituirá uma evolução natural deste trabalho.

Pretende-se que esta investigação contribua não apenas para validar os benefícios técnicos das tecnologias abordadas, mas também para fornecer orientações práticas e fomentar a adoção de melhores práticas de segurança e gestão em ambientes de IaC.

Referências

- [1] D. Oppenheimer, A. Ganapathi, and D. Patterson, “Why do Internet services fail, and what can be done about it?” in *4th USENIX Symp. on Internet Technologies and Systems (USITS’03)*, Seattle WA, USA, 2003. [Online]. Available: <http://roc.cs.berkeley.edu/papers/usits03.pdf>
- [2] M. Chiari, B. Xiang, S. Canzoneri, G. N. Nedeltcheva, E. D. Nitto, L. Blasi, D. Benedetto, L. Niculut, and I. Škof, “Doml: A new modeling approach to infrastructure-as-code,” *Information Systems*, vol. 125, p. 102422, 11 2024. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0306437924000802>
- [3] J. Henriques, F. Caldeira, T. Cruz, and P. Simoes, “A forensics and compliance auditing framework for critical infrastructure protection,” *International Journal of Critical Infrastructure Protection*, vol. 42, p. 100613, 2023, doi: 10.1016/j.ijcip.2023.100613.
- [4] J. Henriques, F. Caldeira, T. Cruz, and P. Simões, “Combining k-means and xgboost models for anomaly detection using log datasets,” *Electronics*, vol. 9, no. 7, 2020, doi: 10.3390/electronics9071164.
- [5] A. Zeini, R. G. Lennon, and P. Lennon, “Securing infrastructure as code (iac) through devsecops:a comprehensive risk management framework,” *2023 Cyber Research Conference - Ireland, Cyber-RCI 2023*, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10671452>
- [6] S. I. Abbas and A. Garg, “Integrating emerging technologies with infrastructure as code in distributed environments,” *Proceedings of the 3rd International Conference on Applied Artificial Intelligence and Computing, ICAAIC 2024*, pp. 1138–1144, 2024. [Online]. Available: <https://ieeexplore.ieee.org/document/10575600>
- [7] A. Ghosh, S. Srivastava, and P. Supraja, “Streamlining multi-cloud infrastructure orchestration: Leveraging terraform as a battle-tested solution,” *2024 International Conference on Cognitive Robotics and Intelligent Systems, ICC - ROBINS 2024*, pp. 911–915, 2024. [Online]. Available: <https://ieeexplore.ieee.org/document/10533995>
- [8] M. Begoug, N. Bessghaier, A. Ouni, E. A. Alomar, and M. W. Mkaouer, “What do infrastructure-as-code practitioners discuss: An empirical study on stack overflow,” *International Symposium on Empirical*

- Software Engineering and Measurement*, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10304847>
- [9] Y. Zhang, F. Wu, and A. Rahman, "Practitioner perceptions of ansible test smells," *Proceedings - IEEE 20th International Conference on Software Architecture Companion, ICSA-C 2023*, pp. 325–327, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10092644>
- [10] R. Opdebeeck, A. Zerouali, and C. D. Roover, "Control and data flow in security smell detection for infrastructure as code: Is it worth the effort?" *Proceedings - 2023 IEEE/ACM 20th International Conference on Mining Software Repositories, MSR 2023*, pp. 534–545, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10174011>
- [11] A. Rahman and L. Williams, "Different kind of smells: Security smells in infrastructure as code scripts," *IEEE Security and Privacy*, vol. 19, pp. 33–41, 5 2021. [Online]. Available: <https://ieeexplore.ieee.org/document/9388795>
- [12] Y. Yang, X. Chen, G. Wang, and L. Cao, "An identity and access management architecture in cloud," *Proceedings - 2014 7th International Symposium on Computational Intelligence and Design, ISCID 2014*, vol. 2, pp. 200–203, 4 2015.
- [13] I. Indu and P. M. Anand, "Identity and access management for cloud web services," *2015 IEEE Recent Advances in Intelligent Computational Systems, RAICS 2015*, pp. 406–410, 6 2016.
- [14] A. Nayanajith and R. Wickramarachchi, "Challenges affecting the successful adoption of devops practices: A systematic literature review," *ICARC 2024 - 4th International Conference on Advanced Research in Computing: Smart and Innovative Trends in Next Generation Computing Technologies*, pp. 311–315, 2024.
- [15] M. S. Ali and D. Puri, "Optimizing devops methodologies with the integration of artificial intelligence," *2024 3rd International Conference for Innovation in Technology, INOCON 2024*, 2024.
- [16] J. Alonso, R. Piliszek, and M. Cankar, "Embracing iac through the devsecops philosophy: Concepts, challenges, and a reference framework," *IEEE Software*, vol. 40, pp. 56–62, 1 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/9915031>
- [17] Y. Leokhin, T. Fatkhulin, and M. Kozhanov, "Research of static application security testing technique problems and methods for solving them," *2024 Systems of Signals Generating and Processing in the Field of on Board Communications, SOSG 2024 - Conference Proceedings*, 2024.
- [18] N. Bessghaier, M. Begoug, C. Mebarki, A. Ouni, M. Sayagh, and M. W. Mkaouer, "On the prevalence, co-occurrence, and impact of infrastructure-as-code smells," *Proceedings - 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2024*, pp. 23–34, 2024. [Online]. Available: <https://ieeexplore.ieee.org/document/10589858>

-
- [19] A. Rahman, C. Parnin, and L. Williams, "The seven sins: Security smells in infrastructure as code scripts," *Proceedings - International Conference on Software Engineering*, vol. 2019-May, pp. 164–175, 5 2019. [Online]. Available: <https://ieeexplore.ieee.org/document/8812041>
- [20] GitHub, "What is ci/cd?" <https://github.com/resources/articles/devops/ci-cd>. [Online]. Available: <https://github.com/resources/articles/devops/ci-cd>
- [21] A. M. Mowad, H. Fawareh, and M. A. Hassan, "Effect of using continuous integration (ci) and continuous delivery (cd) deployment in devops to reduce the gap between developer and operation," *Proceedings - 2022 23rd International Arab Conference on Information Technology, ACIT 2022*, 2022.
- [22] J. Henriques, F. Caldeira, T. Cruz, and P. Simões, "An automated closed-loop framework to enforce security policies from anomaly detection," *Computers & Security*, vol. 123, p. 102949, 2022, doi: 10.1016/j.cose.2022.102949.
- [23] T. Chen and H. Suo, "Design and practice of security architecture via devsecops technology," *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, vol. 2022-October, pp. 310–313, 2022.
- [24] env0, "Terraform tutorial: Getting started with terraform," <https://www.env0.com/blog/terraform-tutorial>. [Online]. Available: <https://www.env0.com/blog/terraform-tutorial>
- [25] Y. Bhalla, V. Hemamalini, and S. Mishra, "Automating hadoop cluster on aws cloud using terraform," *Proceedings of the 1st IEEE International Conference on Networking and Communications 2023, ICNWC 2023*, 2023.
- [26] HashiCorp, "State | terraform | hashicorp developer," <https://developer.hashicorp.com/terraform/language/state>, accessed: Nov. 19, 2024. [Online]. Available: <https://developer.hashicorp.com/terraform/language/state>
- [27] DevOpsSchool.com, "What is terraform.tfstate and terraform.tfstate.backup file in terraform?" <https://www.devopsschool.com/blog/what-is-terraform-tfstate-backup-file-in-terraform/>, accessed: Nov. 19, 2024. [Online]. Available: <https://www.devopsschool.com/blog/what-is-terraform-tfstate-backup-file-in-terraform/>
- [28] S3CloudHub, "Enhancing infrastructure as code with terraform workspaces," <https://medium.com/@S3CloudHub/enhancing-infrastructure-as-code-with-terraform-workspaces-4ea9cb26b3ed>, accessed: Nov. 19, 2024. [Online]. Available: <https://medium.com/@S3CloudHub/enhancing-infrastructure-as-code-with-terraform-workspaces-4ea9cb26b3ed>
- [29] env0, "Terraform tutorial: Getting started with terraform," <https://www.env0.com/blog/terraform-tutorial>. [Online]. Available: <https://www.env0.com/blog/terraform-tutorial>

- [30] Liquid Web, “What is ansible and what does it do?” <https://www.liquidweb.com/blog/what-is-ansible/>, accessed: Nov. 20, 2024. [Online]. Available: <https://www.liquidweb.com/blog/what-is-ansible/>
- [31] env0, “Mastering ansible playbooks: Examples and step by step guide,” <https://www.env0.com/blog/mastering-ansible-playbooks-step-by-step-guide>. [Online]. Available: <https://www.env0.com/blog/mastering-ansible-playbooks-step-by-step-guide>
- [32] Liquid Web, “What is ansible and what does it do?” <https://www.liquidweb.com/blog/what-is-ansible/>. [Online]. Available: <https://www.liquidweb.com/blog/what-is-ansible/>
- [33] V. Lenarduzzi, F. Lomio, H. Huttunen, and D. Taibi, “Are sonarqube rules inducing bugs?” *SANER 2020 - Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution, and Reengineering*, pp. 501–511, 2020.
- [34] D. Marcilio, R. Bonifacio, E. Monteiro, E. Canedo, W. Luz, and G. Pinto, “Are static analysis violations really fixed? a closer look at realistic usage of sonarqube,” *IEEE International Conference on Program Comprehension*, vol. 2019-May, pp. 209–219, 5 2019.
- [35] SonarSource, “Sonarqube analysis overview,” <https://docs.sonarsource.com/sonarqube-server/latest/analyzing-source-code/analysis-overview/>. [Online]. Available: <https://docs.sonarsource.com/sonarqube-server/latest/analyzing-source-code/analysis-overview/>
- [36] Bridgecrew, “Checkov - infrastructure as code static analysis tool,” <https://www.checkov.io/>. [Online]. Available: <https://www.checkov.io/>
- [37] R. Singh, A. Yeboah-Ofori, S. Kumar, and A. Ganiyu, “Fortifying cloud devsecops security using terraform infrastructure as code analysis tools,” *International Conference on Electrical and Computer Engineering Researches, ICECER 2024*, 2024.
- [38] Cybersecurity and Infrastructure Security Agency (CISA), “Apache log4j vulnerability guidance,” <https://www.cisa.gov/news-events/news/apache-log4j-vulnerability-guidance>. [Online]. Available: <https://www.cisa.gov/news-events/news/apache-log4j-vulnerability-guidance>
- [39] The Apache Software Foundation, “Welcome to the apache software foundation,” <https://www.apache.org/>. [Online]. Available: <https://www.apache.org/>
- [40] S. Feng and M. Lubis, “Defense-in-depth security strategy in log4j vulnerability analysis,” *Proceedings - International Conference Advancement in Data Science, E-Learning and Information Systems, ICADEIS 2022*, 2022.
- [41] National Institute of Standards and Technology, “Nvd - cve-2021-44228,” <https://nvd.nist.gov/vuln/detail/CVE-2021-44228>. [Online]. Available: <https://nvd.nist.gov/vuln/detail/CVE-2021-44228>

-
- [42] “What is cvss - common vulnerability scoring system,” <https://www.sans.org/blog/what-is-cvss/>. [Online]. Available: <https://www.sans.org/blog/what-is-cvss/>
- [43] Imperva, “What is WAF | types, security & features explained,” <https://www.imperva.com/learn/application-security/what-is-web-application-firewall-waf/>. [Online]. Available: <https://www.imperva.com/learn/application-security/what-is-web-application-firewall-waf/>
- [44] Google, “Google scholar,” <https://scholar.google.com/>. [Online]. Available: <https://scholar.google.com/>
- [45] Rayyan Systems, “Rayyan: Ai-powered systematic review management platform,” <https://www.rayyan.ai/>. [Online]. Available: <https://www.rayyan.ai/>
- [46] IEEE, “IEEE Xplore Digital Library,” <https://ieeexplore.ieee.org/Xplore/home.jsp>. [Online]. Available: <https://ieeexplore.ieee.org/Xplore/home.jsp>
- [47] Elsevier, “Sciencedirect - peer-reviewed journals,” <https://www.sciencedirect.com/search>. [Online]. Available: <https://www.sciencedirect.com/search>
- [48] John Wiley & Sons, “Wiley online library,” <https://onlinelibrary.wiley.com/>. [Online]. Available: <https://onlinelibrary.wiley.com/>
- [49] PRISMA Statement, “Prisma 2020 flow diagram,” <https://www.prisma-statement.org/prisma-2020-flow-diagram>. [Online]. Available: <https://www.prisma-statement.org/prisma-2020-flow-diagram>
- [50] A. S. A. Alghawli and T. Radivilova, “Resilient cloud cluster with devsecops security model, automates a data analysis, vulnerability search and risk calculation,” *Alexandria Engineering Journal*, vol. 107, pp. 136–149, 11 2024. [Online]. Available: <https://www.sciencedirect.com/science/inproceedings/pii/S1110016824007567>
- [51] A. Paul, R. Manoj, and S. Udhayakumar, “Amazon web services cloud compliance automation with open policy agent,” *Proceedings - 2024 International Conference on Expert Clouds and Applications, ICOECA 2024*, pp. 313–317, 2024. [Online]. Available: <https://ieeexplore.ieee.org/document/10612535>
- [52] R. B. Bahaweres and F. M. Najib, “Provisioning of disaster recovery with terraform and kubernetes: A case study on software defect prediction,” *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, pp. 183–189, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10295601>
- [53] M. P. D. Prakash and N. Sharma, “The convergence of devops and cloud computing: A redefining software development,” *Proceedings of the IEEE International Conference Image Information Processing*, pp. 800–805, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10537710>

- [54] C. A. Supit, A. A. Pangeran, K. O. C. Laban, I. I. Gutandjala, and A. Ramadhan, "Incorporating cloud native architecture and devops culture to improve company agility," *2023 3rd International Conference on Intelligent Cybernetics Technology and Applications, ICICyTA 2023*, pp. 290–294, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10428951>
- [55] Y. Zhang, M. Rahman, F. Wu, and A. Rahman, "Quality assurance for infrastructure orchestrators: Emerging results from ansible," *Proceedings - IEEE 20th International Conference on Software Architecture Companion, ICSA-C 2023*, pp. 322–324, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10092684>
- [56] A. Santos, J. Bernardino, and N. Correia, "Automated application deployment on multi-access edge computing: A survey," *IEEE Access*, vol. 11, pp. 89 393–89 408, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10225499>
- [57] N. Petrovic, "Chat gpt-based design-time devsecops," *2023 58th International Scientific Conference on Information, Communication and Energy Systems and Technologies, Icest 2023 - Proceedings*, pp. 143–146, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10187247>
- [58] D. Sokolowski and G. Salvaneschi, "Towards reliable infrastructure as code," *Proceedings - IEEE 20th International Conference on Software Architecture Companion, ICSA-C 2023*, pp. 318–321, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10092598>
- [59] P. R. R. Konala, V. Kumar, and D. Bainbridge, "Sok: Static configuration analysis in infrastructure as code scripts," *Proceedings of the 2023 IEEE International Conference on Cyber Security and Resilience, CSR 2023*, pp. 281–288, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10224925>
- [60] G. Falazi, U. Breitenbucher, F. Leymann, M. Stotzner, E. Ntontos, U. Zdun, M. Becker, and E. Heldwein, "On unifying the compliance management of applications based on iac automation," *2022 IEEE 19th International Conference on Software Architecture Companion, ICSA-C 2022*, pp. 226–229, 2022. [Online]. Available: <https://www.sciencedirect.com/science/inproceedings/pii/S0950584921001543>
- [61] R. Acheampong, T. C. Balan, D. M. Popovici, and A. Rekeraho, "Security scenarios automation and deployment in virtual environment using ansible," *14th International Conference on Communications, COMM 2022 - Proceedings*, 2022. [Online]. Available: <https://ieeexplore.ieee.org/document/9817150>
- [62] M. M. Hassan and A. Rahman, "As code testing: Characterizing test quality in open source ansible development," *Proceedings - 2022 IEEE 15th International Conference on Software Testing, Verification and Validation, ICST 2022*, pp. 208–219, 2022. [Online]. Available: <https://ieeexplore.ieee.org/document/9787876>

- [63] A. Dalvi, "Cloud infrastructure self service delivery system using infrastructure as code," *3rd IEEE 2022 International Conference on Computing, Communication, and Intelligent Systems, ICCIS 2022*, pp. 455–460, 2022. [Online]. Available: <https://ieeexplore.ieee.org/document/10037603>
- [64] A. Sorour and A. Hamdy, "Devops and iac to automate the delivery of hands-on software lab exams," *Proceedings - 2022 6th International Conference on Computer, Software and Modeling, ICCSM 2022*, pp. 28–35, 2022. [Online]. Available: <https://ieeexplore.ieee.org/document/9982716>
- [65] D. Reis, B. Piedade, F. F. Correia, J. P. Dias, and A. Aguiar, "Developing docker and docker-compose specifications: A developers' survey," *IEEE Access*, vol. 10, 2022. [Online]. Available: <https://ieeexplore.ieee.org/document/9658534>
- [66] I. Kumara, M. Garriga, A. U. Romeu, D. D. Nucci, F. Palomba, D. A. Tamburri, and W. J. van den Heuvel, "The do's and don'ts of infrastructure code: A systematic gray literature review," *Information and Software Technology*, vol. 137, p. 106593, 9 2021. [Online]. Available: <https://www.sciencedirect.com/science/inproceedings/pii/S0950584921000720>
- [67] S. Muthoni, G. Okeyo, and G. Chemwa, "Infrastructure as code for business continuity in institutions of higher learning," *International Conference on Electrical, Computer, and Energy Technologies, ICECET 2021*, 2021. [Online]. Available: <https://ieeexplore.ieee.org/document/9698544>
- [68] A. Rahman, F. L. Barsha, and P. Morrison, "Shhh: 12 practices for secret management in infrastructure as code," *Proceedings - 2021 IEEE Secure Development Conference, SecDev 2021*, pp. 56–62, 2021. [Online]. Available: <https://ieeexplore.ieee.org/document/9652648>
- [69] S. D. Palma, D. D. Nucci, F. Palomba, and D. A. Tamburri, "Toward a catalog of software quality metrics for infrastructure code," *Journal of Systems and Software*, vol. 170, p. 110726, 12 2020. [Online]. Available: <https://www.sciencedirect.com/science/inproceedings/pii/S0164121220301618>
- [70] A. Rahman, E. Farhana, C. Parnin, and L. Williams, "Gang of eight: A defect taxonomy for infrastructure as code scripts," *Proceedings - International Conference on Software Engineering*, pp. 752–764, 6 2020. [Online]. Available: <https://ieeexplore.ieee.org/document/9284113>
- [71] Xygeni, "Secret leakage - all you need to know | xygeni," <https://xygeni.io/blog/all-you-need-to-know-about-secret-leakage/>. [Online]. Available: <https://xygeni.io/blog/all-you-need-to-know-about-secret-leakage/>
- [72] HashiCorp, "How do you centrally manage and secure secret credentials, keys, and passwords?" <https://www.hashicorp.com/pt/resources/what-is-secret-sprawl-why-is-it-harmful>. [Online]. Available: <https://www.hashicorp.com/pt/resources/what-is-secret-sprawl-why-is-it-harmful>

- [73] Ask Ubuntu, “openssh - does ssh key need to be named id_rsa? - ask ubuntu,” <https://askubuntu.com/questions/30788/does-ssh-key-need-to-be-named-id-rsa>. [Online]. Available: <https://askubuntu.com/questions/30788/does-ssh-key-need-to-be-named-id-rsa>
- [74] Apache Software Foundation, “Welcome! - the apache http server project,” <https://httpd.apache.org/>. [Online]. Available: <https://httpd.apache.org/>
- [75] OpenSSL Software Foundation, “Openssl,” <https://www.openssl.org/>. [Online]. Available: <https://www.openssl.org/>
- [76] Python Software Foundation, “Download python | python.org,” <https://www.python.org/downloads/>. [Online]. Available: <https://www.python.org/downloads/>
- [77] Electronic Frontier Foundation, “Certbot,” <https://certbot.eff.org/>. [Online]. Available: <https://certbot.eff.org/>
- [78] WSGI Project, “Wsgi — wsgi.org,” <https://wsgi.readthedocs.io/en/latest/>. [Online]. Available: <https://wsgi.readthedocs.io/en/latest/>
- [79] Bridgecrew, “Github - bridgecrewio/checkov: Prevent cloud misconfigurations and find vulnerabilities during build-time in infrastructure as code, container images and open source packages with checkov by bridgecrew,” <https://github.com/bridgecrewio/checkov>. [Online]. Available: <https://github.com/bridgecrewio/checkov>
- [80] Prisma Cloud, “Terraform resource scans - checkov,” <https://www.checkov.io/5.Policy%20Index/terraform.html>. [Online]. Available: <https://www.checkov.io/5.Policy%20Index/terraform.html>