Capítulo 1

Para além dos Perímetros da CiberSegurança com a Infraestrutura como Código (IaC)

Fellipe M. Veiga (PUCPR), Altair O. Santin (PUCPR), Juliano S. Langaro (PUCPR), Juarez de Oliveira (PUCPR), Eduardo K. Viegas (PUCPR)

Abstract

Traditional perimeter-based security approaches have proven insufficient in the face of increasingly sophisticated threats and the growing complexity of modern, dynamic, and automated computing environments. In this context, this short course proposes an integrated approach that combines the principles of Zero Trust Architecture with Infrastructure as Code (IaC) practices, aiming to promote more secure, resilient, and versionable infrastructures from the provisioning stage. The course includes a theoretical foundation covering infrastructure automation with IaC, microsegmentation, identity-based access control, and continuous context analysis—key elements for mitigating contemporary threats. In the practical phase, participants will be guided through the secure provisioning of an infrastructure using tools such as Terraform and Ansible. Critical security aspects will be explored, including attack surface management, configuration drift detection, the use of trusted images, and the implementation of least privilege policies. By the end of the course, participants are expected to understand the challenges of securing code-defined environments and to be able to apply Zero Trust principles in real-world provisioning scenarios, fostering infrastructures that are more reliable and resilient to lateral movements and insider threats.

Resumo

As abordagens tradicionais de segurança baseadas em perímetro têm-se mostrado insuficientes diante da crescente sofisticação das ameaças e da complexidade de ambientes computacionais modernos, dinâmicos e automatizados. Nesse cenário, este minicurso apresenta uma abordagem integrada que combina os princípios da arquitetura Zero Trust com as práticas de Infrastructure as Code (IaC), com o objetivo de promover ambientes mais seguros, resilientes e versionáveis desde o provisionamento. O ambiente de experimentação fornece uma fundamentação sobre automação de infraestrutura com IaC,

microssegmentação, controle de acesso baseado em identidade e análise contínua de contexto — elementos centrais para a mitigação de ameaças contemporâneas. Na etapa prática, os participantes serão orientados a provisionar uma infraestrutura segura utilizando ferramentas como Terraform e Ansible, nas quais serão explorados aspectos críticos de segurança, como o gerenciamento da superfície de ataque, a detecção de desvios de configuração, o uso de imagens confiáveis e a aplicação de políticas de mínimo privilégio. Ao final, espera-se que os participantes compreendam os desafios da segurança em ambientes definidos como código e estejam aptos a aplicar os princípios do modelo Zero Trust em cenários reais de provisionamento, promovendo infraestruturas mais confiáveis e robustas contra movimentações laterais e ataques internos.

1.1. Introdução

A Infraestrutura como Código (*Infrastructure as Code*, IaC) é uma técnica que viabiliza a automação, o provisionamento e a configuração de recursos computacionais de forma eficiente e reprodutível. Essa abordagem organiza o gerenciamento de *data centers* por meio da definição de arquivos legíveis por máquina (*machine-readable files*), eliminando a necessidade de configuração física de hardware ou da utilização de ferramentas interativas [Fowler 2013]. Por exemplo, utilizando Terraform, é possível definir toda a infraestrutura necessária para um cenário de aplicação — servidores, rede, bancos de dados etc. — em arquivos de configuração que podem ser versionados e reaplicados automaticamente, garantindo consistência e agilidade no ambiente de produção [Filho et al. 2025]. De forma complementar, o gerenciamento de contêineres empacotam a aplicação e suas dependências, facilitando o desenvolvimento, o teste e a implantação em ambientes isolados e reprodutíveis, usando Docker [Rodrigues et al. 2025]. No contexto da IaC, a integração entre Terraform e Docker possibilita que o Terraform automatize o provisionamento da infraestrutura subjacente, enquanto o Docker gerencia a implantação e execução das aplicações. Essa combinação promove uma orquestração completa, desde a configuração da infraestrutura até a entrega do ambiente de execução, garantindo controle, rastreabilidade e facilidade de reprodução.

A adoção da IaC otimiza processos, assegura a replicabilidade e promove ambientes computacionais mais padronizados e consistentes [Rahman et al. 2019a]. Muitas organizações que oferecem serviços em nuvem já incorporam tecnologias da IaC em seus fluxos de trabalho. Essa prática evidencia a importância do tema, pois seus benefícios facilitam o provisionamento automatizado e o gerenciamento por meio de *scripts*, reduzindo erros humanos, acelerando a entrega de recursos e garantindo maior controle sobre as configurações ao longo do ciclo de vida da infraestrutura.

Apesar das vantagens, a prática da IaC apresenta desafios significativos relacionados à segurança. A automação, embora eficiente, pode amplificar falhas existentes em configurações manuais, tornando mais evidentes problemas tradicionais, como a exposição acidental de credenciais em arquivos de configuração, permissões excessivas e a falta de controle sobre mudanças. Esses desafios, antes dispersos e difíceis de detectar, tornam-se mais críticos no contexto da IaC, onde erros se propagam rapidamente e em larga escala. Assim, a segurança do código IaC, a proteção da infraestrutura provisionada e o gerenciamento das modificações após o provisionamento são fundamentais para garantir uma implementação segura e resiliente. Os problemas tradicionais de segurança não

desaparecem com a automação, ao contrário, podem ser potencializados, mas a detecção e correção é facilitada. A facilidade proporcionada pela IaC para implementar e configurar recursos com agilidade exige atenção redobrada, mas favorece a segurança por projeto (*security by design*), a manutenção e operação do ambiente devido a programabilidade da IaC.

Por se tratar de uma implementação baseada em código, a identificação e correção de *code smells* – indicadores de problemas de manutenção ou qualidade no código-fonte, tornam-se um recurso para fortalecer a robustez do sistema [Rahman et al. 2019b]. Embora nem todo *code smell* represente uma falha funcional direta, sua presença pode revelar limitações estruturais que, por sua vez, podem gerar *security smells* — padrões de código associados a potenciais vulnerabilidades [Silveira Neto et al. 2024]. Dessa forma, o monitoramento tanto dos *code smells* quanto dos *security smells* facilita a manutenção e a legibilidade do código, além de atuar preventivamente, reduzindo riscos de segurança ao promover um código mais limpo e menos suscetível a falhas exploráveis.

Outro desafio relacionado à segurança em IaC está no tratamento de segredos e credenciais sensíveis nos *scripts* [Rahman et al. 2022]. Elementos como identificadores de usuário, senhas, *tokens* e chaves criptográficas são essenciais para a autenticação entre artefatos. Contudo, a prática comum de armazená-los em texto claro, inclusive em sistemas de controle de versão, expõe a infraestrutura a riscos significativos. Essas falhas de configuração são um fator de exposição recorrente [MITRE ATT&CK 2024], demandando a adoção de técnicas adequadas para proteger segredos dentro em IaC.

Os riscos mais frequentes na gestão de segredos incluem a falha de exposição de credenciais privilegiadas, o uso de segredos embutidos no código, a ausência de rotação periódica e a dependência de processos manuais [Rahman and Anwar 2021]. Embora o uso de cofres de senhas *(vault)* ofereça melhorias, essas soluções ainda apresentam vulnerabilidades, especialmente quando baseadas em arquiteturas com segurança por perímetro. A adoção de uma validação criteriosa e contínua para cada acesso visa reduzir a dependência do *(vault)*, garantindo que, mesmo em caso de exposição de segredos, seu uso seja restrito e monitorado, melhorando a segurança em ambientes da IaC.

O modelo de segurança perimetral tradicional pode apresentar falhas que comprometem a eficácia de mecanismos como *firewalls*, VPN *(Virtual Private Network)*, IDS *(Intrusion Detection System)* e IPS *(Intrusion Prevention System)* [Simioni et al. 2025b, Viegas et al. 2017], tornando-os insuficientes contra ameaças sofisticadas e movimentos laterais na rede. Nesse cenário, surge o modelo *Zero Trust* para, implicitamente, gerenciar por credenciais todos os acessos, seja para usuários, dispositivos ou aplicações. Na prática, esse modelo exige validação rigorosa e contínua de todas as entidades que acessam recursos, aplicando o mínimo privilégio (por padrão), autenticação, autorização e criptografia a cada requisição [Rose et al. 2020]. Essas características são especialmente relevantes em ambientes provisionados como código, onde a agilidade e a escala podem gerar exposição caso a segurança não seja incorporada por projeto *(security by design)*.

Em suma, *Zero Trust* rejeita a confiança automática e adota por padrão o mínimo privilégio, microssegmentação e autenticação multifator (*Multi-factor Authentication*, MFA) — aos fluxos de trabalho que junto com a IaC representam uma estratégia eficaz para mitigar vulnerabilidades em infraestruturas distribuídas e dinâmicas.

Conteúdo. Este minicurso tem como objetivo apresentar uma abordagem teórica e prática para o provisionamento de infraestrutura segura por meio da IaC, com ênfase na aplicação dos princípios do modelo *Zero Trust*. O minicurso abordará os fundamentos da IaC, capacitando os participantes a identificar e mitigar riscos como a exposição de segredos em código e o uso de imagens vulneráveis. Também será realizada uma análise comparativa entre o modelo de segurança perimetral tradicional, suas limitações e o risco de movimentos laterais, justificando a adoção da arquitetura *Zero Trust*. Serão detalhados os princípios desse modelo, como a verificação contínua, o mínimo privilégio e o acesso baseado em contexto. Numaa atividade prática os participantes utilizarão ferramentas como Terraform e Ansible para provisionar e configurar uma infraestrutura segura, sendo incentivados a refletir sobre a arquitetura implementada e as decisões de segurança adotadas.

Estrutura. O documento está organizada da seguinte forma. A seção 1.2 apresenta os fundamentos da IaC e da gestão de segredos. A seção 1.3 destacada as vulnerabilidades, assim como práticas de segurança aplicáveis à IaC, incluindo ferramentas de análise, verificação de imagens (binários) e prevenção de recursos órfãos. A seção 1.4 faz um comparativo entre a segurança tradicional, o modelo *Zero Trust* e os desafios específicos da IaC. A seção 1.5 detalha o ambiente de experimentação, enquanto a seção 1.6 apresenta um estudo de caso guiado por meio de experimentação de laboratório. Por fim, a seção 1.7 sintetiza as vantagens de segurança da abordagem apresentada.

1.2. Fundamentação

A abordagem tradicional de segurança foi estruturada com base em perimetros, geralmente, domínio (rede) local e externo. No domínio local, compartilhado em segmentos de rede interna, se assume que tudo é confiável, enquanto que no domínio externo nada é confiável. Portanto, o simples fato de ter acesso a rede interna já permite alcançar recursos que no cenário atual podem configurar ama superficie de ataque menos protegida. Em ambientes de computação em nuvem e práticas com o uso da IaC, esse modelo tem se mostrado cada vez mais limitado e vulnerável. A automação de infraestrutura em larga escala amplia significativamente a superfície de ataque e, na ausência de controles robustos, pode replicar configurações inseguras de forma massiva. Nesse contexto, o paradigma da *Zero Trust Architecture* (ZTA) ganha relevância ao rejeitar qualquer forma de confiança implícita e exigir validações contínuas de identidade, contexto e postura de segurança, independentemente da origem da solicitação. A integração entre IaC e os princípios do *Zero Trust* configura uma abordagem estratégica para mitigar riscos de exposição de recurso, permitindo que políticas de controle de acesso baseadas em credenciais, segmentação e autenticação sejam definidas e aplicadas diretamente no código da infraestrutura.

A seguir apresentamos a fundamentação em três abordagens principais: a automação por meio da IaC, a superação das limitações do modelo perimetral tradicional e a adoção do paradigma de *Zero Trust*. A partir dessas bases, discutimos os fundamentos técnicos e conceituais necessários para a construção de infraestruturas resilientes e seguras desde as etapas iniciais de seu provisionamento.

1.2.1. Infraestrutura como código (IaC): Conceitos e características

A prática da Infraestrutura como Código (*Infrastructure as Code*, IaC) representa uma mudança de paradigma na forma como as organizações configuram, provisionam e gerenciam seus recursos de tecnologia da informação. Tradicionalmente, esses processos exigiam intervenções manuais demoradas, sujeitas a erros e inconsistências operacionais. Com a adoção da IaC, essas atividades são automatizadas por meio de *scripts* e arquivos de configuração, promovendo ganhos significativos em eficiência, replicabilidade e versionamento, que permite auditabilidade de mudanças.

A IaC proporciona benefícios como o compartilhamento ágil de recursos, escalabilidade eficiente e redução da incidência de erros humanos [Özdoğan et al. 2023]. Essa abordagem utiliza ferramentas que interpretam arquivos de configuração escritos em linguagens declarativas ou imperativas, possibilitando a criação e o gerenciamento de servidores, redes, políticas de segurança, volumes de armazenamento e outros componentes críticos da infraestrutura computacional. A adoção da IaC reduz significativamente o tempo necessário para a implantação de ambientes, além de aumentar a previsibilidade e a gestão operacional [Wettinger et al. 2014]. Tratar a infraestrutura como código-fonte viabiliza práticas como o reuso de componentes, a colaboração entre equipes por meio de *pull requests* e a integração com *pipelines* de entregas contínuas (CI/CD) [Horchulhack et al. 2022a]. Essa integração reforça a padronização das configurações e contribui para a construção de ambientes mais seguros, versionáveis *(versionable)* e consistentes.

As ferramentas que implementam os conceitos da IaC podem ser classificadas de acordo com seus propósitos. Ferramentas de provisionamento como o Terraform[1], são responsáveis pela alocação de recursos em plataformas de computação em nuvem públicos, *on-premise* ou híbridos [Viegas et al. 2020]. Já ferramentas de configuração, como Ansible[2] e Puppet[3], concentram-se na definição de estados desejados e na aplicação de ajustes pós-provisionamento. Essas soluções permitem que a infraestrutura seja descrita por meio de arquivos versionáveis, com suporte a auditoria de mudanças e reutilizáveis, promovendo consistência entre ambientes e integrando-se de forma eficiente a fluxos de *DevOps*, por exemplo. Tipicamente, a IaC pode ser implementada segundo dois paradigmas principais [Chen et al. 2018]:

- **Modelo imperativo.** Nesse modelo, o responsável especifica detalhadamente a sequência de comandos que devem ser executados para criar e configurar os recursos de infraestrutura. A lógica de controle fica sob total responsabilidade do autor do código, que precisa definir a ordem exata das operações. Embora proporcione maior controle e flexibilidade sobre o processo, esse modelo pode tornar os *scripts* mais verbosos, difíceis de manter e propensos a erros em ambientes complexos;
- **Modelo declarativo.** Ao adotar esse modelo, o autor descreve o estado final desejado da infraestrutura, e a ferramenta encarrega-se de interpretar essa descrição e aplicar as ações necessárias para alcançar a configuração pretendida. Isso reduz a complexidade do código e facilita a manutenção, uma vez que o foco está no

[1]https://github.com/hashicorp/terraform
[2]https://github.com/ansible
[3]https://www.puppet.com/community

resultado e não na forma de alcançá-lo. Ferramentas como Terraform utilizam esse modelo, promovendo maior reprodutibilidade, controle de estado e consistência entre ambientes distintos.

Além da automação e da padronização, a IaC possibilita a criação de ambientes efêmeros, a realização de testes automatizados de infraestrutura e a reversão controlada de mudanças (*rollback*). Esses recursos ampliam significativamente a segurança operacional, a rastreabilidade das alterações e a velocidade de entrega de ambientes, ao permitir ciclos de desenvolvimento mais ágeis e confiáveis. Ambientes efêmeros, por exemplo, podem ser criados sob demanda para testes específicos e destruídos em seguida, reduzindo custos e diminuindo a superfície de ataque. Testes automatizados garantem que configurações estejam em conformidade antes da aplicação em produção, enquanto mecanismos de *rollback* permitem reverter mudanças de forma segura em caso de falhas.

No entanto, os benefícios trazidos pela IaC também introduzem novos desafios. A gestão de segredos sensíveis, como credenciais e chaves criptográficas, torna-se mais complexa quando embutida em código ou disseminada entre arquivos e repositórios. A detecção de derivações de estado (*drifts*) — situações em que a infraestrutura real diverge da definida no código — exige monitoramento constante e ferramentas que verifiquem a consistência. Dessa forma, a adoção segura da IaC requer a implementação de controles adicionais, revisão contínua e validações automatizadas ao longo de todo o ciclo de vida da infraestrutura.

1.2.1.1. Orquestração com Docker Compose

Além das ferramentas específicas da IaC para provisionamento e configuração, é comum a utilização de *containers* para estruturar ambientes modulares, isolados e portáveis. O Docker Compose[4] é uma ferramenta que permite definir e orquestrar múltiplos *containers* por meio de um único arquivo de configuração, o que facilita o controle de dependências, a padronização e a reprodutibilidade do ambiente, especialmente em fluxos de desenvolvimento e testes contínuos.

O código 1.1 apresenta um exemplo de arquivo `docker-compose.yml` que configura dois *containers*: um servidor de banco de dados MariaDB[5] e uma instância da aplicação GLPI[6], um sistema de gerenciamento de serviços de TI amplamente utilizado.

Esta configuração YAML define um ambiente Docker Compose com dois serviços: `mariadb` e `glpi`. O serviço `mariadb` utiliza a imagem oficial do MariaDB 10.5, configura variáveis de ambiente para a senha do usuário `root` e o nome do banco de dados, além de montar um volume para garantir persistência dos dados. O serviço `glpi` baseia-se em uma imagem personalizada, expõe a porta 8080 no host, ajusta o fuso horário e depende do serviço `mariadb`, assegurando que o banco de dados esteja ativo antes de sua inicialização. Ambos os serviços são conectados por meio de uma rede dedicada, e volumes para persistência são configurados para o armazenamento dos dados do banco,

[4]https://docs.docker.com/compose/

[5]https://mariadb.org/

[6]https://glpi-project.org/

Código Fonte 1.1. YAML exemplo no Docker Compose para execução do MariaDB e GLPI.

```yaml
version: "3.8" # Versão do Docker-compose
services:
  mariadb: # Serviço MariaDB
    image: mariadb:10.5 # Imagem
    container_name: mariadb_serv
    environment: # Variáveis de ambiente
      MYSQL_ROOT_PASSWORD: rootpass
      MYSQL_DATABASE: glpidb
    volumes:  # Volumes
      - db_data:/var/lib/mysql
    networks: # Redes
      - glpi_net

  glpi: # Serviço GLPI
    image: diouxx/glpi # Imagem
    container_name: glpi_serv
    ports: # Portas mapeadas
      - "8080:80"
    environment: # Variáveis de ambiente
      TIMEZONE: America/Sao_Paulo
    depends_on: # Serviço de dependência
      - mariadb
    networks: # Redes
      - glpi_net

volumes:
  db_data:

networks:
  glpi_net:
```

garantir integridade e isolamento. Na prática, este exemplo define os *containers* de forma declarativa, garantindo que o ambiente de aplicação esteja pronto para uso com um único comando: `docker-compose up -d`.

1.2.1.2. Descrevendo recursos com Terraform

O Terraform é uma ferramenta da IaC desenvolvida pela HashiCorp que permite o provisionamento, gerenciamento e atualização de recursos de forma automatizada e declarativa. Utilizando arquivos de configuração escritos em *HashiCorp Configuration Language* (HCL), o Terraform possibilita a descrição do estado desejado de uma infraestrutura, incluindo servidores, redes, balanceadores de carga, bancos de dados e outros serviços em provedores como AWS, Azure, Google Cloud, entre outros. Ao interpretar os arquivos, o Terraform realiza um plano de execução (`terraform plan`) que compara o estado atual com o estado desejado, calculando as ações necessárias para convergência. Esse mecanismo facilita a padronização de ambientes, o controle de mudanças e a rastreabilidade das operações, sendo especialmente útil em equipes que adotam práticas de DevOps e CI/CD. Além disso, o gerenciamento de estado permite detectar derivações e aplicar correções com segurança, promovendo consistência e versionamento na gestão da infraestrutura.

Código Fonte 1.2. Exemplo de configuração Terraform.

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "~> 2.4"    # Versão requerida
    }
  }
}

# Provedor local (não exige configuração adicional)
provider "local" {}

# Recurso: arquivo de texto gerado pelo Terraform
resource "local_file" "exemplo" {
  filename = "exemplo.txt"
  content  = "Este é um exemplo prático com Terraform."
}
```

Para ilustrar, este exemplo prático apresenta a definição de um recurso básico de infraestrutura utilizando a ferramenta Terraform. O Código 1.2 especifica a criação de uma instância de Máquina Virtual (VM) em uma plataforma de computação em nuvem compatível com o provedor local.

Este *script* do Terraform configura o provedor local, que permite ao Terraform gerenciar os recursos do sistema local. Ele começa especificando o provedor necessário (*hashicorp/local*) com uma restrição de versão (> 2.4), garantindo a compatibilidade com uma versão estável. O bloco provider é declarado, mas não requer nenhuma configuração específica. Em seguida, um recurso do tipo *local_file* é definido, denominado exemplo. Este recurso instrui o Terraform a criar um arquivo chamado "*exemplo.txt*"contendo o texto: "*Este é um exemplo prático com Terraform*". O *script* é simples, mas demonstra como o Terraform pode ser usado para gerar arquivos locais como parte da automação de infraestrutura.

A Figura 1.1 apresenta uma organização típica de diretórios contendo arquivos do Terraform. Após a declaração dos recursos, a execução do comando `terraform plan` gera um plano de execução, conforme ilustrado na Figura 1.2. Esse plano permite ao usuário revisar as alterações antes de aplicá-las, analisando o código de configuração e comparando-o com o estado atual da infraestrutura.

O resultado é um plano detalhado que indica precisamente quais recursos serão criados (marcados com +), modificados (~) ou destruídos (−). Cada recurso é descrito com seus atributos, incluindo valores já definidos e aqueles que só serão conhecidos após a aplicação (*known after apply*). Ao final, um resumo como `Plan: 1 to add, 0 to change, 0 to destroy` fornece uma visão das mudanças previstas, garantindo controle e previsibilidade sobre as modificações no ambiente.

Esse modelo de execução contribui significativamente para a segurança e a previsibilidade da infraestrutura, ao permitir a revisão prévia das alterações, a verificação da conformidade com as políticas organizacionais e a prevenção de efeitos inesperados.

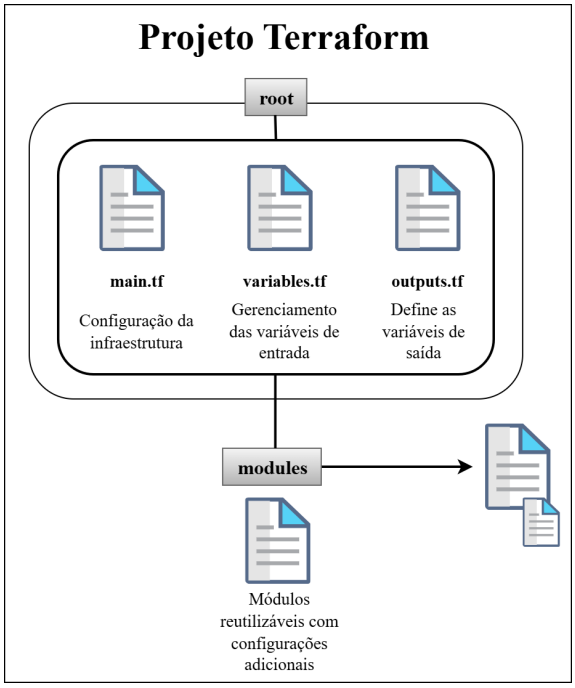

Figura 1.1. Exemplo de estrutura de diretórios de um projeto Terraform.

terraform plan

```
Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_s3_bucket.example will be created
  + resource "aws_s3_bucket" "example" {
      + acl    = (known after apply)
      + arn    = (known after apply)
      + bucket = "meu-bucket-de-exemplo-12345"
      + id     = (known after apply)
      + tags   = {
          + Environment = "Dev"
          + ManagedBy   = "Terraform"
        }

      # (outros atributos omitidos para brevidade)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

Figura 1.2. Saída do comando `terraform plan` para o exemplo apresentado no código 1.2.

Além disso, as mudanças no código torna-se auditável, reutilizável e facilmente integrável a *pipelines* automatizados, favorecendo práticas de DevOps e promovendo maior controle ao longo do ciclo de vida da infraestrutura.

Código Fonte 1.3. Playbook Ansible para instalar e iniciar o Apache.

```
- name: Instala e inicia o servidor web Apache
  hosts: servidores_web
  become: yes

  tasks:
    - name: Garante que o pacote do Apache2 está instalado
      ansible.builtin.apt:
        name: apache2
        state: present
        update_cache: yes

    - name: Garante que o serviço do Apache está iniciado
      ansible.builtin.service:
        name: apache2
        state: started
        enabled: yes
```

1.2.1.3. Configuração com Ansible

Além do provisionamento de recursos, a automação das configurações é uma etapa fundamental para assegurar consistência e segurança nos ambientes. O Ansible é uma ferramenta amplamente adotada para esse propósito, permitindo a definição do estado desejado dos sistemas por meio de arquivos YAML, que são simples, legíveis e facilmente versionáveis. Sua abordagem é baseada em um modelo declarativo, no qual são especificadas as tarefas que devem ser realizadas para alcançar uma determinada configuração, como a instalação de pacotes, a edição de arquivos, a criação de usuários ou a ativação de serviços. O Ansible opera de forma *agentless*, ou seja, não requer a instalação de agentes nos nós gerenciados. Em vez disso, utiliza conexões SSH para executar comandos remotamente, o que reduz a complexidade de manutenção e aumenta a portabilidade. Os *playbooks* — arquivos que descrevem as tarefas — são organizados em blocos reutilizáveis, o que favorece a modularização e a padronização de configurações em diferentes ambientes. Além disso, o Ansible integra-se facilmente a *pipelines* de CI/CD, permitindo aplicar configurações automaticamente após o provisionamento de infraestrutura, o que o torna uma peça chave em fluxos de trabalho DevOps baseados em IaC.

O código 1.3, apresenta um exemplo básico de *playbook* do Ansible que instala o servidor *web* Apache[7] em uma máquina Linux.

Este *playbook* do Ansible automatiza a instalação e a inicialização do servidor web Apache em um grupo de hosts denominado `servidores_web`. A execução é realizada com privilégios elevados (`become: yes`), garantindo permissões suficientes para modificar configurações e instalar pacotes em nível de sistema. O *playbook* é composto por duas tarefas principais:

- **Garantia que o Apache esteja instalado**: para tanto, utiliza o módulo nomeado `ansible.builtin.apt` para instalar o pacote `apache2`. A opção de atualiza-

[7] https://www.apache.org/

ção do cache de pacotes é ativada, assegurando que a versão mais recente disponível no repositório seja utilizada;

- **Garanta que o serviço Apache esteja ativo e habilitado**: emprega o módulo `ansible.builtin.service` para iniciar o serviço `apache2` e configurá-lo para iniciar automaticamente junto com o sistema operacional;

Com essas duas tarefas, o *playbook* garante que o servidor Apache esteja corretamente instalado, ativo e configurado para iniciar automaticamente em todas as máquinas de destino definidas no inventário. Essa abordagem simplifica o gerenciamento de servidores web e assegura consistência na configuração dos ambientes.

Docker, Terraform e Ansible são ferramentas complementares amplamente utilizadas em conjunto para provisionar, configurar e operar ambientes de infraestrutura programável, especialmente em contextos baseados em DevOps e automação. Em um fluxo típico, o Terraform é utilizado inicialmente para provisionar os recursos de infraestrutura, como máquinas virtuais, redes, volumes de armazenamento e balanceadores de carga, seja em nuvem pública ou *on-premise* (ambientes locais). Após o provisionamento, o Ansible pode ser utilizado para configurar esses recursos, instalando pacotes, ajustando serviços e aplicando políticas de segurança de forma declarativa. O Docker, por sua vez, pode ser utilizado para encapsular aplicações e suas dependências em *containers* portáveis e reprodutíveis, garantindo isolamento entre os serviços. Um cenário prático que ilustra essa integração seria o provisionamento de um cluster de servidores em uma nuvem privada (*on-premise*) com Terraform, seguido pela configuração automática de cada nó com Ansible para instalação do Docker e de serviços de orquestração (como Docker Compose ou Kubernetes). Em seguida, as aplicações são implantadas como *containers*, garantindo agilidade, consistência e escalabilidade no ambiente.

1.2.1.4. Validação contínua com Checkov

Um dos principais desafios na adoção da IaC é garantir que o código da infraestrutura esteja em conformidade com boas práticas de segurança antes de ser aplicado em ambientes de produção. A natureza automatizada e reprodutível da IaC, embora traga ganhos em eficiência, também amplifica o impacto de erros de configuração ou permissões indevidas, por exemplo. Nesse contexto, a análise estática de código se torna uma etapa importante do processo, permitindo inspecionar os arquivos de configuração sem executá-los, com o objetivo de detectar vulnerabilidades, omissões e desvios de compatibilidade de forma antecipada. Ferramentas como o Checkov[8], CloudFormation Linter[9] e kube-linter[10] aplicam regras pré-definidas para examinar arquivos Terraform, CloudFormation ou Kubernetes, sinalizando práticas inseguras como a ausência de criptografia, exposição de serviços à internet pública, uso de senhas em texto claro ou a falta de controle de identidade e acesso. A integração dessas ferramentas a fluxos de CI/CD permite que a validação de segurança

[8]https://www.checkov.io/

[9]https://github.com/aws-cloudformation/cfn-lint

[10]https://kube-linter.io/

checkov saída

```
checkov -d ./infraestrutura/terraform

By bridgecrew.io | version: 2.3.187

terraform scan results:

Passed checks: 4, Failed checks: 3, Skipped checks: 0

Check: CKV_AWS_19: "Ensure all data stored in the S3 bucket is securely encrypted at
rest"
  FAILED for resource: aws_s3_bucket.data_bucket
  File: /main.tf:10-21
  Guide: https://docs.bridgecrew.io/docs/s3_16-enable-bucket-encryption-in-rest

  Code lines: 10-21
  10 | resource "aws_s3_bucket" "data_bucket" {
  11 |   bucket = "meu-bucket-de-dados-importantes"
  ...
  18 |   # FALHA: Bloco de criptografia do servidor ausente
  20 | }

Check: CKV_AWS_21: "Ensure all data stored in the S3 bucket has versioning enabled"
  FAILED for resource: aws_s3_bucket.data_bucket
  File: /main.tf:10-21
  Guide: https://docs.bridgecrew.io/docs/s3_15-enable-versioning

Check: CKV_AWS_54: "Ensure S3 bucket has block public policy enabled"
  FAILED for resource: aws_s3_bucket.data_bucket
  File: /main.tf:10-21
  Guide: https://docs.bridgecrew.io/docs/s3_10-enable-block-public-policy
```

Figura 1.3. Resultado da validação com Checkov, identificando falhas em um recurso Terraform.

ocorra de forma automatizada e contínua, reduzindo significativamente os riscos antes do provisionamento efetivo da infraestrutura.

No exemplo a seguir, utiliza-se o comando *checkov -d ./infraestrutura/terraform* para analisar um projeto Terraform que cria um bucket S3 na AWS[11]. A Figura 1.3 apresenta a execução do Checkov nesse diretório, evidenciando as verificações realizadas e os possíveis alertas gerados. Essa ferramenta pode ser facilmente utilizada pelo usuário para identificar vulnerabilidades e garantir a conformidade das configurações antes da aplicação das mudanças na infraestrutura.

A verificação de segurança efetuada pelo Checkov realizou um total de sete checagens, das quais quatro foram aprovadas e três reprovadas. Todas as falhas estão relacionadas ao recurso `aws_s3_bucket.data_bucket`, definido entre as linhas 10 e 21 do arquivo `main.tf`. Especificamente, o bucket não possui criptografia do lado do

[11] https://aws.amazon.com

servidor para dados em repouso, o versionamento está desabilitado e falta a configuração para bloqueio de políticas públicas, o que representa riscos significativos à segurança, sendo três práticas inseguras segundo referências de conformidade como CIS Benchmarks [Center for Internet Security (CIS)] e NIST [Mell and Grance 2011]. Para cada falha são fornecidas referências a documentações detalhadas que orientam na correção dessas vulnerabilidades e na melhoria da segurança do bucket.

Outras ferramentas complementares, como tfsec[12], TFLint[13] e Terrascan[14], também podem ser empregadas em conjunto para fortalecer a qualidade e a segurança do código IaC, oferecendo diferentes níveis de análise estática, detecção de vulnerabilidades e boas práticas específicas para ambientes Terraform.

1.2.2. Ferramentas e ecossistemas da IaC

A crescente adoção da IaC estimulou o desenvolvimento de um ecossistema diversificado de ferramentas especializadas, que atuam em diferentes etapas do ciclo de vida da infraestrutura, considerando desde o provisionamento de recursos até a configuração, monitoramento, segurança, versionamento e integração com *pipelines* de entrega contínua.

Ferramentas como Terraform e OpenTofu[15] são amplamente utilizadas para o provisionamento declarativo de recursos em provedores de nuvem como AWS, Azure[16], GCP[17], além de ambientes *on-premise*. Por meio de arquivos de configuração, os administradores descrevem o estado desejado da infraestrutura, enquanto os motores dessas ferramentas aplicam as mudanças necessárias com base em planos de execução versionáveis. A *HashiCorp Configuration Language* (HCL), utilizada pelo Terraform, consolidou-se como referência nesse domínio, por seu equilíbrio entre simplicidade e expressividade [Wang 2022].

No contexto da configuração de sistemas e aplicações, ferramentas como Ansible, SaltStack[18] e Puppet possibilitam o gerenciamento automatizado de pacotes, serviços, arquivos e políticas. Essas soluções atuam de forma complementar ao provisionamento, assegurando que os recursos recém-criados sejam inicializados e configurados corretamente conforme os padrões definidos pela equipe técnica. A combinação de ferramentas de provisionamento e configuração reforça a consistência entre ambientes — desenvolvimento, testes e produção — além de aprimorar a rastreabilidade das alterações realizadas [Rahman et al. 2019a].

Além das ferramentas principais, há um conjunto crescente de soluções voltadas à verificação, segurança e conformidade da infraestrutura codificada. Ferramentas como Trivy[19], Checkov, Tfsec e Terrascan realizam análises estáticas nos arquivos de configuração, identificando problemas como recursos públicos expostos inadvertidamente, ausência de criptografia, falhas no controle de acesso e presença de credenciais em texto claro

[12]https://aquasecurity.github.io/tfsec/latest/
[13]https://github.com/terraform-linters/tflint
[14]https://runterrascan.io/
[15]https://opentofu.org
[16]https://azure.microsoft.com
[17]https://cloud.google.com
[18]https://saltproject.io/
[19]https://trivy.dev

[Reddy Konala et al. 2023]. Essas boas práticas garantem proteção contra configurações inseguras ou fralhas de configuração não intencionais.

O ecossistema da IaC inclui serviços de armazenamento seguro de segredos, como o HashiCorp Vault[20], que se integram aos *scripts* de provisionamento e configuração para fornecer chaves, *tokens* e senhas de forma segura e auditável. Combinados com políticas de controle de acesso e autenticação por provedor de identidada (IdP), esses serviços previnem a exposição de informações sensíveis e permitem a aplicação de princípios fundamentais de segurança, como mínimo privilégio e tempo de vida limitado das credenciais [Oliveira et al. 2024].

Além disso, ferramentas de integração contínua, como GitLab CI[21], GitHub Actions[22] e Jenkins[23], atuam como orquestradores no *pipeline* automatizado de infraestrutura. Elas possibilitam a execução programada ou condicional de planos de provisionamento, validações de segurança, testes e implantações, promovendo práticas de auditoria contínua, versionamento, *rollback* e rastreabilidade — aspectos essenciais para ambientes dinâmicos e baseados em microserviços.

1.2.3. Gestão de segredos e práticas de proteção

Em ambientes automatizados com IaC, a gestão de segredos constitui um aspecto crítico para assegurar a proteção de credenciais, chaves de API, certificados e *tokens* de acesso utilizados durante o provisionamento e a configuração de recursos. Como os *scripts* da IaC circulam por repositórios versionados, *pipelines* de CI/CD e múltiplos ambientes de execução, qualquer vazamento acidental de segredos representa um sério risco à integridade e à segurança da infraestrutura [Rahman et al. 2021a, Rahman et al. 2021b].

Embora ferramentas como Ansible, Terraform e Kubernetes ofereçam recursos robustos para automação, o manuseio seguro de informações sensíveis depende da adoção de boas práticas. O uso de arquivos criptografados (como o Vault do Ansible) ou integrações com sistemas externos de gerenciamento de segredos — como HashiCorp Vault, AWS Secrets Manager e Azure Key Vault — é amplamente recomendado para reduzir a superfície de exposição [Yuliarman et al. 2023, Oliveira et al. 2024, Vehent, Julien 2018].

A adoção de soluções especializadas para gerenciamento de segredos (*vaults*) viabiliza não apenas o armazenamento seguro, mas também funcionalidades como rotação automática de credenciais, controle de acesso baseado em políticas e registro detalhado de mudanças [Abreu et al. 2020]. A simples detecção de segredos em código-fonte não é suficiente sem uma estratégia efetiva de mitigação e prevenção [Rahman et al. 2022]. Nesse contexto, a aplicação de práticas como o uso de *placeholders*, a integração com ferramentas automatizadas de escaneamento de segredos em *pipelines* (como Gitleaks e TruffleHog) e a implementação do princípio do minimo privilégio tornam-se componentes essenciais de uma postura segura em ambientes com IaC.

Além disso, observa-se um movimento crescente em direção à incorporação de

[20]https://developer.hashicorp.com/vault
[21]https://docs.gitlab.com/ci
[22]https://github.com/features/actions
[23]https://www.jenkins.io

autenticação não interativa e baseada em múltiplos fatores, como *One-Time Password* (OTP) e certificados digitais, com o objetivo de automatizar o acesso a sistemas sem comprometer a segurança. Dessa forma, a gestão de segredos em ambientes com IaC requer uma abordagem DevSecOps (desenvolvimento, segurança e operação) mais desenvolvida em que segurança, automação, operação e gestão atuem de forma integrada. Essa abordagem garante que segredos sejam tratados como ativos sensíveis, com ciclos de vida bem definidos, rastreabilidade completa e mecanismos de proteção desde a definição no código até a execução em produção.

A gestão segura de credenciais é também um elemento central para a aplicação prática do modelo *Zero Trust* em ambientes automatizados com IaC. Em vez de depender de autenticações interativas (com intervenção de um ator humano) — suscetíveis a falhas operacionais e vazamentos, devido a exposição de credenciais quando o humano não pode estar presente — esse modelo privilegia mecanismos automatizados, granulares e versionáveis, que reforçam o princípio de não confiar implicitamente em nenhum ator, mesmo nos limites do perímetro da rede interna.

Uma abordagem recomendada para a gestão segura de credenciais em ambientes com IaC é a utilização de cofres de senhas (*vault*), como o HashiCorp Vault, que permitem armazenar e distribuir segredos de forma segura e programática. Contas de serviço utilizadas em *pipelines* de CI/CD ou em *scripts* de automação podem acessar os recursos por meio de *tokens* gerados sob demanda, com tempo de vida limitado e escopo específico. Essa estratégia elimina a necessidade de armazenar senhas ou chaves de acesso diretamente no código-fonte ou em repositórios versionados, reduzindo significativamente o risco de exposição e comprometimento.

As permissões associadas a essas contas de serviço seguem o princípio do mínimo privilégio (*least privilege*) e são definidas por meio de políticas de acesso declarativas, codificadas nos próprios arquivos de infraestrutura. Isso garante que os acessos sejam configurados de forma rastreável, por versionamento, e coerente com as diretrizes de segurança da organização, promovendo maior controle sobre o ciclo de vida de permissões de acesso.

A autenticação não interativa também possibilita a execução de tarefas recorrentes, atualizações automatizadas e verificações de integridade sem intervenção humana, aumentando a escalabilidade e a resiliência dos sistemas. Dessa forma, o processo de provisionamento e manutenção de infraestrutura torna-se não apenas mais eficiente, mas também mais seguro, uma vez que os mecanismos de autenticação e autorização são continuamente validados e monitorados por sistemas autoamtizados.

1.3. Segurança e riscos em ambientes automatizados com IaC

A prática da IaC transformou a forma como ambientes computacionais são provisionados e gerenciados, promovendo agilidade, padronização e escalabilidade. Contudo, essa automação intensiva também introduz novos vetores de risco à cibersegurança, exigindo atenção quanto à exposição, inclusive de informações sensíveis, à validação das configurações e à manutenção do código da IaC.

Ao tratar a infraestrutura como código-fonte, os riscos tradicionalmente associados

ao desenvolvimento de *software* passam a afetar diretamente essa camada, tornando o código IaC um ativo estratégico. Qualquer falha nesse código — como concessão de permissões excessivas, exposição indevida de portas de rede ou inclusão de segredos embutidos (*tokens*, senhas, chaves de API) — pode comprometer múltiplos ambientes simultaneamente, especialmente em cenários com replicação automatizada [Rahman et al. 2019b].

Um dos desafios recorrentes nesse contexto são os chamados *security smells*, padrões de codificação que indicam potenciais vulnerabilidades. Ferramentas amplamente adotadas, como Ansible, Terraform e Chef, podem potencializar tais falhas sistêmicas, incluindo o uso de senhas fixadas diretamente no código (*hardcoded*)[NIST 2025], a ausência de validação de entradas e o emprego de parâmetros inseguros por padrão[Basak et al. 2022, Rahman et al. 2021a]. Essas práticas tornam-se especialmente perigosas quando o código IaC é versionado em repositórios públicos ou mantido sem controles adequados, ampliando significativamente a superfície de ataque.

Outro aspecto relevante é o fenômeno conhecido como *configuration drift*, que ocorre quando o estado real da infraestrutura diverge do estado declarado no código. Essa discrepância pode ser causada por alterações manuais no código, falhas na aplicação dos *scripts* ou inconsistências na sincronização entre ambientes. O resultado é uma infraestrutura instável e propensa a erros, com impactos diretos na confiabilidade, na rastreabilidade e na segurança dos sistemas [Rahman et al. 2019a].

A segurança de ambientes automatizados por meio da IaC está intrinsecamente ligada à confiabilidade dos artefatos utilizados durante o provisionamento. Entre esses, destacam-se as imagens (binários) do sistema operacional e de *containers*, que servem como base para instanciar servidores, serviços e aplicações [Horchulhack et al. 2022b]. Quando mal configuradas, desatualizadas ou não validadas, essas imagens podem conter fragilidades conhecidas, *malwares* ou configurações inseguras, comprometendo a integridade, a estabilidade e a segurança de toda a infraestrutura.

Imagens comprometidas ampliam a superfície de ataque e facilitam a exploração por agentes maliciosos, sobretudo em ambientes de larga escala, nos quais a replicação automatizada a partir de um mesmo códgo pode propagar falhas para dezenas ou centenas de instâncias desse *template* – arquivos reutilizáveis que definem a configuração desejada da infraestrutura [Queen 2024]. Nesse contexto, a verificação contínua e automatizada de imagens, utilizando ferramentas específicas de análise de vulnerabilidades, torna-se uma prática importante para garantir arquiteturas seguras e resilientes baseadas em IaC.

Para mitigar esses riscos, é fundamental integrar ferramentas de análise estática e verificadores de configuração ao ciclo de desenvolvimento da IaC. Soluções como Trivy, Checkov, tfsec, Terrascan e Docker Content Trust[24] realizam inspeções automatizadas nos *scripts* e nos binários das imagens utilizadas, identificando vulnerabilidades conhecidas, violações de políticas organizacionais e desvios em relação às boas práticas de segurança [Reddy Konala et al. 2023, Chiari et al. 2022]. Essas ferramentas funcionam como recurso de análise preventiva no ciclo DevSecOps, promovendo boas práticas de cibersegurança desde as fases iniciais do processo de automação (programabilidade) da infraestrutura.

[24]https://docs.docker.com/engine/security/trust/

Neste caso, a segurança das imagens em IaC é garantida pelo uso de repositórios confiáveis, com autenticidade e integridade dos artefatos por meio de assinaturas digitais. Soluções como o Sigstore[25], especialmente com sua ferramenta Cosign[26], permitem assinar imagens no formato OCI e registrar essas assinaturas em *logs* que permitem rastrear as mudanças, assegurando que apenas imagens verificáveis e não adulteradas sejam implantadas [Lorenc 2021].

A adoção de assinatura digital necessita que fatores como usabilidade, integração com ambientes automatizados e suporte à *pipelines* de CI/CD estejam disponíveis a fim de garantir a eficácia e a adoção em larga escala dessas tecnologias [Kalu et al. 2025]. Esses elementos tornam-se indispensáveis para compor uma cadeia de fornecimento segura de software dentro de infraestruturas baseadas em código. Obviamente, não é necessariamente indispensável o uso de DevSecOps com IaC, mas como essa abordagem tem a automatização como objetivo e mais comumente mencionada lá.

Cosign é uma ferramenta do ecossistema Sigstore projetada para assinar, verificar e armazenar assinaturas de imagens de *containers* de forma segura, automatizada e compatível com fluxos de DevSecOps. Seu principal objetivo é garantir que apenas imagens autenticadas e provenientes de fontes confiáveis sejam utilizadas em ambientes de produção, reduzindo o risco de execução de artefatos adulterados ou maliciosos.

O fluxo de assinatura com Cosign/Sigstore caracteriza-se por três etapas principais que reforçam a integridade e a autenticidade de imagens de *containers*:

1. **Assinatura da imagem.** O desenvolvedor ou sistema automatizado utiliza o Cosign para gerar ou obter uma chave criptográfica (ou certificado, muitas vezes emitido de forma efêmera pela infraestrutura do Sigstore) e assinar a imagem de *container* com base em seu *digest* criptográfico (normalmente um hash SHA256) [Boulden 2023]. Isso garante que a assinatura esteja vinculada de forma única ao conteúdo da imagem;

2. **Armazenamento e registro.** A assinatura, juntamente com o certificado utilizado, é armazenada no próprio registro de imagens (como Docker Hub, GHCR ou Harbor), em um local separado mas associado ao *digest* da imagem. Paralelamente, os metadados dessa assinatura são registrados em um *log* de transparência, como o Rekor, o que permite auditoria pública e impede alterações silenciosas, promovendo a imutabilidade e a confiança no histórico da imagem.

3. **Validação no deployment.** Durante o processo de implantação, ferramentas de política como Kyverno[27], OPA/Gatekeeper[28] ou validações automatizadas no *pipeline* verificam a presença e a validade da assinatura digital. Imagens não assinadas ou cuja assinatura não corresponda ao certificado confiável são automaticamente rejeitadas, evitando implantações de artefatos comprometidos.

Esse fluxo garante que apenas imagens autenticadas e verificáveis sejam executadas

[25]https://www.sigstore.dev/
[26]https://github.com/sigstore/cosign
[27]https://kyverno.io/
[28]https://github.com/open-policy-agent/gatekeeper

em ambientes sensíveis, contribuindo para a construção de cadeias de suprimento de software seguras e versionáveis.

Ao incorporar esse processo no *pipeline* da IaC, obtém-se uma camada adicional de confiança automática e verificável, uma vez que somente imagens assinadas e auditadas são provisionadas, complementando a verificação estática e a validação de recursos existentes. Essas assinaturas funcionam como certificados de origem, sendo fundamentais para suportar ambientes seguros e alinhados ao modelo *Zero Trust*. Adicionalmente, o uso de repositórios oficiais ou internos validados contribui para o controle do ciclo de vida das imagens, evitando atualizações não autorizadas ou modificações maliciosas. Esse controle, aliado à inspeção contínua, fortalece a gestão dos recursos utilizados na infraestrutura automatizada.

Além disso, a incorporação de práticas de segurança em todas as etapas do *pipeline* de entrega contínua é um princípio fundamental do paradigma DevSecOps. Isso envolve a execução de testes automatizados nos *scripts* da IaC, a aplicação consistente do princípio do menor privilégio, a gerstão de mudanças e a segmentação dos ambientes de acordo com seus respectivos níveis de risco. Incorporar a segurança como parte integrante do ciclo de vida da infraestrutura, e não como uma fase posterior, é essencial para a construção de ambientes resilientes e versionáveis [Alonso et al. 2023].

Nesse cenário, é igualmente importante evitar inconsistências que levam a criação de recursos órfãos, ou *orphaned resources* — componentes que permanecem ativos mesmo após terem sido removidos do código de infraestrutura. Esses recursos não referenciados consomem recursos computacionais, violam os processos regulares de verificação, auditoria e aplicação de políticas, além de representarem potenciais vetores de ataque e gerarem custos operacionais desnecessários em grande escala [CloudOptimo 2025].

A mitigação desses riscos requer a adoção de mecanismos de verificação automatizada, como os fornecidos por serviços como AWS Config e AWS Trusted Advisor, bem como a implementação de políticas de *decommissioning* que detectem e eliminem recursos não declarados explicitamente no código da IaC. Esse tipo de controle de manutenção é importante para garantir ambientes mais seguros, eficientes e gerenciais [Mangera 2025].

Dessa forma, a segurança em ambientes automatizados com IaC demanda uma abordagem multidimensional, que abrange desde a adoção de boas práticas de codificação e controle de versões até o uso de ferramentas automatizadas para verificação, gestão segura de segredos e aplicação contínua de políticas de manutenibilidade. O foco não se limita à proteção do código em si, mas se estende à preservação da integridade, confidencialidade e disponibilidade de toda a infraestrutura como código, de maneira proativa, escalável e versionável.

1.3.1. *Code smells* e *Security smells* em IaC

No contexto da engenharia de *software*, o termo *code smell* refere-se a indícios de problemas estruturais no código que, embora não causem falhas funcionais imediatas, podem comprometer sua manutenibilidade, legibilidade e robustez. Em ambientes baseados em IaC, essas fragilidades assumem um papel ainda mais crítico, pois afetam diretamente a segurança e a confiabilidade dos ambientes computacionais provisionados. Os *code smells*

presentes em *scripts* IaC frequentemente antecedem ou facilitam o surgimento de vulnerabilidades conhecidas como *security smells* [Rahman et al. 2021a]. Estes representam padrões recorrentes de práticas inseguras ou mal estruturadas, como o uso de senhas e *tokens* em texto claro, a concessão excessivas de privilégios de acesso, a exposição de interfaces administrativas ou a ausência de controles explícitos de autenticação. Embora essas práticas possam inicialmente parecer inofensivas, elas frequentemente se tornam vetores de exploração para agentes maliciosos, comprometendo a integridade do ambiente automatizado.

A presença desses *security smells* está frequentemente ligada à pressa na entrega, à ausência de revisão por pares e testes, e à falta de políticas formais de desenvolvimento seguro. Ferramentas de análise estática específicas para IaC têm sido utilizadas para identificar essas falhas, possibilitando que as equipes corrijam os problemas antes da execução dos *scripts*. Entre os principais tipos de *security smells* identificados na literatura, destacam-se:

- **Uso de segredos embutidos**: refere-se à prática de inserir credenciais, como senhas, chaves de API ou tokens de acesso diretamente nos arquivos de configuração ou *scripts* IaC. Essa abordagem facilita a exposição acidental desses dados sensíveis em repositórios públicos ou internos, aumentando o risco de comprometimento da infraestrutura. Além disso, dificulta a rotação e o controle centralizado dos segredos, negligêncaindo boas práticas de segurança.

- **Configurações permissivas**: envolve a ausência ou inadequação de restrições de acesso a recursos, incluindo a abertura desnecessária de portas TCP (*Transmission Control Protocol*), a exposição pública de interfaces administrativas ou a definição de permissões excessivas para usuários e serviços. Essas configurações ampliam a superfície de ataque, facilitando a exploração por agentes maliciosos e aumentando a probabilidade de incidentes de segurança.

- **Ausência de validação**: refere-se à falta de mecanismos para verificar a integridade, autenticidade ou conformidade das configurações aplicadas pela IaC. Isso pode incluir a ausência de validações de entrada, testes automatizados ou checagens de políticas de segurança que garantam que o estado provisionado corresponde às diretrizes definidas. A consequência é a possível implantação de configurações incorretas, inseguras ou incompatíveis, comprometendo boas práticas de segurança dos ambientes.

- **Dependências não verificadas**: diz respeito ao uso de imagens de sistema operacional, contêineres, módulos ou outros recursos cuja origem, integridade e atualizações de segurança não são verificadas adequadamente. Essa prática pode introduzir vulnerabilidades conhecidas, *backdoors* ou componentes desatualizados na infraestrutura, ampliando riscos e dificultando a manutenção com políticas de segurança e a rastreabilidade com o versionamento.

A mitigação desses *security smells* requer a adoção de ferramentas automatizadas de análise estática e auditoria de mudanças, além da implementação de padrões de

Tabela 1.1. Principais vulnerabilidades em IaC, descrições e ferramentas de mitigação

Vulnerabilidade	**Descrição**	**Mitigação (Ferramenta)**
Uso de segredos embutidos	Credenciais, tokens ou senhas codificados diretamente no código-fonte.	Gestão de segredos: Gitleaks, HashiCorp Vault, AWS Secrets Manager
Configurações permissivas	Falta de restrições adequadas ou permissões excessivas.	Análise estática: Checkov, tfsec, OPA/Gatekeeper, Kyverno
Ausência de validação	Falta de avaliação sistêmica da integridade e conformidade das configurações.	Validação e testes: Terraform Validate, Terrascan
Dependências sem auditabilidade de mudanças	Uso de imagens ou módulos sem verificação de origem e atualizações.	Scanner de vulnerabilidades: Trivy, Docker Content Trust, Cosign/Sigstore
Versões desatualizadas de ferramentas	Ferramentas e módulos IaC desatualizados podem conter falhas.	Gerenciamento de dependências: Renovate, Dependabot
Configuração incorreta/mal configurada	Parâmetros incorretos ou inadequados que podem comprometer a segurança.	Análise estática e validação: Checkov, tfsec, terrascan; testes em ambiente isolado
Gerenciamento inadequado de segredos	Falta de controle de acesso ou armazenamento seguro dos segredos utilizados.	Cofres de segredos: HashiCorp Vault, AWS Secrets Manager
Falta de validação e testes automatizados	Ausência de processos que verifiquem a qualidade e segurança dos *scripts*.	Integração contínua: GitLab CI, GitHub Actions, Jenkins; ferramentas de teste IaC
Permissões excessivas em recursos	Concessão de privilégios sem adoção do principio do mínimo privilégio.	Políticas de acesso: OPA/Gatekeeper, Kyverno; revisão de permissões IaC
Uso de módulos ou binários de imagens não confiáveis	Inclusão de componentes de fontes desconhecidas ou sem auditoria prévia.	Assinatura e verificação de imagens: Cosign/Sigstore, Docker Content Trust
Ausência de monitoramento e *logging* adequado	Falta de registros de versionamento que permitam auditoria de mudanças e detecção de atividades suspeitas em recursos provisionados.	Ferramentas de monitoramento: AWS CloudTrail, Azure Monitor; alertas configurados
Provisionamento de recursos não seguros	Criação de recursos com configurações inseguras ou sem proteção adequada.	Análise estática, testes e políticas: tfsec, Checkov, Kyverno; revisão manual
Falta de controle de versão e mudanças	Ausência de versionamento, auditoria e *rollback*.	Sistemas de versionamento: Git e ferramentas de revisão

desenvolvimento seguro[29]. Integração de revisões contínuas e verificações de segurança nos *pipelines* de entrega permite que, em tempo, se identifique falhas e a correção seja realizada antes da aplicação das configurações. Essas práticas não apenas fortalecem a segurança da infraestrutura, mas também promovem a maturidade operacional das equipes responsáveis pela gestão de ambientes definidos como código.

A Tabela 1.1 apresenta as principais vulnerabilidades encontradas em ambientes da IaC, acompanhadas de descrições e das ferramentas recomendadas para sua mitigação. A tabela destaca problemas comuns como o uso de segredos embutidos no código, configurações permissivas e mal configuradas, além da ausência de validação e testes automatizados,

[29]https://www.microsoft.com/en-us/securityengineering/sdl/practices

que podem comprometer a segurança e o funcionamento adequado da infraestrutura. Também são evidenciados riscos relacionados ao uso de versões desatualizadas, dependências decorrentes de mudanças não auditadas e imagens não confiáveis, reforçando a importância da gestão adequada de segredos, controle de privilégios e monitoramento contínuo. As ferramentas citadas, como Checkov, tfsec, Trivy, Cosign e HashiCorp Vault oferecem soluções automatizadas para detecção, validação e gestão, integrando-se aos pipelines de desenvolvimento para garantir práticas seguras e auditoria de mudanças ao longo do ciclo de vida da infraestrutura.

Na próxima seção, serão apresentadas as práticas recomendadas para a gestão segura de segredos e credenciais em ambientes IaC, com foco no uso de cofres de senhas (*vault*), políticas de rotação e controle de acesso.

1.4. Da segurança perimetral à arquitetura *Zero Trust*

Durante décadas, o modelo tradicional de cibersegurança das organizações usa o modelo baseado em proteção de perimetro. Essa abordagem assume que, uma vez autenticados, os usuários tem acesso a uma rede delimitada por um perimetro construído por meio de *firewalls*, VPNs ou *proxies* etc. Dentro desse perimetro todos (usuários e sistemas) são confiáveis e podem acessar os recursos compartilhados. No cenário atual, caracterizado pelo trabalho remoto, mobilidade corporativa, computação em nuvem, e da crescente complexidade dos ambientes distribuídos, esse paradigma tornou-se limitado porque se perde a noção de perímetro, tornando muito díficil garantir uma proteção eficaz diante dos vetores de ataque atuais [Simioni et al. 2025a, Rose et al. 2020].

O modelo de arquitetura de *Zero Trust* (*Zero Trust Architecture*, ZTA) surge como uma resposta a essas limitações, reformulando a noção de confiança na infraestrutura de TI. O modelo *Zero Trust* adota o princípio "*nunca confie, sempre verifique*" [Rose et al. 2020]. Isso isignifica que nenhum dispositivo, usuário ou aplicação deve ser considerado confiavel de forma implícita, mesmo que esteja dentro dentro de um perimetro (no domínio interno de uma organização). A validação da confiança passa a ser contínua e contextual, considerando múltiplas camadas de proteção como identidade, postura do dispositivo, localização, contexto da solicitação e nível de privilégio concedido.

Nos fundamentos técnicos da ZTA destacam-se a microsegmentação da rede, a autenticação contínua e multifatores, controle de acesso granular baseado em políticas (como ABAC ou RBAC [Abreu et al. 2017]) e a visibilidade contextual sobre os fluxos de tráfego e operações. Esses princípios são implementados com o suporte de ferramentas como *gateways* de acesso definido por software (*Software Defined Perimeter*, SDP), sistemas de identidade baseado em provedor (*Identity Provider*, IdP), cofres de segredos e serviços de verificação contínua da postura de segurança [Syed et al. 2022, Marquis 2024, Matthew Kosinski 2024].

A integração entre o modelo *Zero Trust* e a abordagem da IaC é particularmente estratégica em ambientes nativos de nuvem computacional. Ao tratar a infraestrutura como código, torna-se possível aplicar controles de autenticação, autorização e segmentação de forma padronizada e reproduzível, desde as fases iniciais do ciclo de provisionamento. Por exemplo, ao utilizar Terraform para orquestrar recursos na nuvem computacional, é viável incorporar regras que restringem o tráfego por região

geográfica, exigência de autenticação baseada em chaves rotativas armazenadas em cofres seguros e configuração de segmentações lógicas por serviço ou domínio de aplicação [Rahman et al. 2022, Oliveira et al. 2024].

Ao contrário da segurança perimetral, que frequentemente se apoia em regras estáticas e segmentações baseadas em topologia física, a integração entre IaC e o modelo *Zero Trust* viabiliza uma infraestrutura adaptável, capaz de responder dinamicamente a mudanças no contexto operacional. Essa flexibilidade é fundamental para mitigar ameaças atuais como movimentações laterais em redes internas, uso de credenciais comprometidas, ataques originados de dentro da organização e falhas de configuração — vetores amplamente documentados em repositórios de inteligência [MITRE ATT&CK 2024, de Oliveira et al. 2023].

Para superar as limitações do modelo tradicional baseado em perímetro, é preciso adotar mecanismos mais refinados de controle, como a microssegmentação. Essa técnica isola aplicações, serviços e cargas de trabalho em domínios lógicos distintos, permitindo a aplicação de políticas específicas para cada fluxo de comunicação. O controle do tráfego interno, também chamado de tráfego "*Leste-Oeste*", torna-se assim mais preciso e contextual, reduzindo drasticamente a superfície de ataque e limitando o impacto de comprometimento a abrangencia local. A microssegmentação permite:

- **Isolamento efetivo:** Segmenta aplicações, cargas de trabalho e ambientes em nível granular: por serviço, função, equipe ou sensibilidade dos dados. Isso viabiliza a aplicação de políticas de segurança específicas e contextuais. Dessa forma, mesmo que uma parte da infraestrutura seja comprometida, o impacto permanece restrito aquele segmento específico;

- **Redução da superfície de ataque:** Ao limitar a comunicação entre componentes apenas ao que é estritamente necessário, a microssegmentação restringe significativamente os caminhos disponíveis para atacantes realizarem movimentações laterais. Isso dificulta a propagação de ameaças internas e externas, reduzindo a probabilidade de exploração de vulnerabilidades em cadeia. Recursos expostos ao público podem ser rigidamente isolados de sistemas sensíveis, como bancos de dados ou backends;

- **Proteção robusta:** A separação lógica e dinâmica de ambientes permite reforçar a segurança de contêineres, máquinas virtuais, dispositivos e microsserviços. Isso impede que falhas ou acessos indevidos em um componente comprometam o restante da infraestrutura. Essa abordagem também facilita a implementação de controles de acesso baseados em identidade e contexto, além de permitir respostas mais ágeis e precisas a incidentes.

Portanto, a transição da segurança perimetral para modelos baseados em *Zero Trust* exige não apenas uma reformulação da arquitetura de TI, mas também uma mudança cultural e técnica mais profundas. Nesse cenário, a automação da infraestrutura por meio da IaC torna-se um habilitador estratégico, ao permitir que controles de segurança, políticas de acesso e práticas de conformidade sejam aplicados de forma programática, repetível e versionável. Essa abordagem facilita a gestão, aumenta a resiliência operacional

e proporciona agilidade para responder a ameaças em tempo real, revando a resposta a incidentes de segurança à velocidade compatível com à complexidade dos ambientes modernos.

1.4.1. ZTA aplicada à segurança com IaC

O modelo de ZTA surgiu como resposta às limitações dos paradigmas tradicionais de segurança baseados em perímetro, nos quais se assume que todos os elementos dentro da rede corporativa são confiáveis por padrão. Essa suposição tem se tornado cada vez mais inadequada diante da crescente sofisticação dos vetores de ataque, da adoção massiva de serviços em nuvem e da mobilidade organizacional [Viegas et al. 2020]. Conforme estabelecido pelo NIST na publicação SP 800-207[30], a filosofia *Zero Trust* sustenta que nenhuma entidade, seja interna ou externa, deve ser automaticamente confiável [Rose et al. 2020].

A implementação do modelo ZTA requer políticas de acesso restritas, fundamentadas em identidade, contexto e avaliações dinâmicas de risco, com validações contínuas para cada solicitação de acesso. Entre os elementos centrais estão a autenticação multifator, a microsegmentação da rede, o monitoramento constante de tráfego e o uso de fontes confiáveis para verificação de identidade e autorização, promovendo uma postura de segurança adaptativa e centrada na minimização da confiança implícita.

No contexto da IaC, a adoção do modelo *Zero Trust* é estratégica ao possibilitar que princípios como mínimo privilégio, verificação contínua e controle automatizado sejam incorporados desde a comcepção da infraestrutura. Utilizando arquivos versionáveis, é possível declarar e aplicar políticas de segurança durante o provisionamento, assegurando que os recursos estejam adequadamente protegidos antes mesmo da infraestrutura entrarem em operação [Syed et al. 2022]. A convergência entre ZTA e IaC facilita a adoção de práticas como microsegmentação e isolamento granular de componentes, reduzindo significativamente o risco de movimentações laterais em cenários de comprometimento. Ferramentas como o *Open Policy Agent* (OPA) podem ser integradas aos *pipelines* de CI/CD para validar automaticamente as políticas declarativas, impedindo a promoção de configurações inseguras aos ambientes produtivos.

Além disso, soluções como o Twingate[31] e o Zscaler[32], baseadas no modelo de *Zero Trust Network Access* (ZTNA), reforçam essa abordagem ao viabilizar conexões seguras com base na identidade do usuário, no contexto da solicitação e no estado do dispositivo. A integração dessas ferramentas com práticas da IaC permite a construção de arquiteturas dinâmicas, versionáveis (com suporte a auditoria de mudanças) e alinhadas aos princípios de segurança *by design*. A IaC exerce papel central na concretização dessa estratégia ao descrever, de forma declarativa, o estado desejado dos ambientes de TI — seja em nuvens computacionais públicas, privadas (*on-premise*), híbridas ou em infraestrutura local. Essa representação programável permite a aplicação sistemática de controles de segurança, viabilizando práticas como o versionamento de políticas de acesso, controle automatizado de mudanças e rastreabilidade completa das modificações realizadas.

Com a sobreposição de camadas de segurança, como autenticação multifator (MFA),

[30] https://csrc.nist.gov/pubs/sp/800/207/final
[31] https://www.twingate.com/docs/
[32] https://www.zscaler.com/br

Tabela 1.2. Comparação entre Segurança Perimetral e *Zero Trust Architecture*

Característica	Segurança Perimetral	Zero Trust Architecture
Modelo de Confiança	Confiança baseada na localização (dentro da rede igual a confiável)	Nunca confie, sempre verifique; nenhuma localização é automaticamente confiável
Autenticação e Autorização	Autenticação no ponto de entrada; acesso geralmente irrestrito após entrada	Autenticação contínua e contextual, com autorização mínima (mínimo privilégio)
Proteção de Recursos	Protege a rede como um todo, com foco no perímetro	Protege cada recurso individualmente, baseado em identidade, contexto e políticas
Comportamento em Nuvem computacional e Mobilidade	Ineficaz para nuvem, dispositivos móveis e trabalho remoto	Projetado para ambientes distribuídos, nuvem, dispositivos móveis e acesso remoto
Visibilidade e Monitoramento	Monitoramento focado no perímetro, com pouca visibilidade interna	Monitoramento contínuo e granular de usuários, dispositivos e comportamento
Controle de Privilégios	Acesso amplo após autenticação inicial	Mínimo privilégio e segmentação de acesso

controle de sessão, gestão de identidade e análise comportamental, a arquitetura passa a operar de forma sinérgica e com controle contínuo. Essa abordagem fortalece mecanismos de autenticação e autorização, ao mesmo tempo em que promove conformidade regulatória e mitiga riscos com maior eficácia. Nesse contexto, a automação não elimina a necessidade de verificação, ao contrário, amplia a segurança ao tornar políticas de mudanças auditáveis, reproduzíveis e integradas ao ciclo de vida da infraestrutura de TI.

Para facilitar a compreensão das diferenças entre os modelos de segurança discutidos, segurança perimetral tradicional e ZTA, a Tabela 1.2 apresenta uma síntese comparativa dos principais conceitos, objetivos, benefícios, riscos e boas práticas. Essa visualização evidencia como o modelo *Zero Trust*, quando aliado às práticas da IaC, representa uma evolução significativa frente às limitações do paradigma tradicional baseado em perímetro, promovendo uma estratégia de defesa mais robusta, dinâmica e alinhada aos desafios de lidar com a complexidade da cibersegurança em ambientes programáveis.

1.4.2. Considerações finais da parte teórica

A revisão teórica apresentada nesta seção estabeleceu os fundamentos para compreender a abordagem da IaC e sua importância no cenário atual da cibersegurança. Inicialmente, foi analisado como a IaC promove uma mudança significativa nas práticas de provisionamento e gerenciamento de infraestrutura, ao substituir processos manuais por mecanismos automatizados e versionáveis, em conformidade com os princípios do DevSecOps.

Na sequência, a análise contrapôs essa abordagem aos modelos tradicionais de segurança baseados em perímetro. Embora esses modelos tenham sido eficazes em contextos centralizados, mostram-se insuficientes diante da crescente complexidade, descentralização e dinamicidade dos ambientes de computação em nuvem e de infraestruturas distribuídas. As arquiteturas *Zero Trust* destacam-se como uma abordagem robusta, pautada no

princípio de "confiança zero". Ou seja, em vez de presumir confiança, o foco recai na microsegmentação, na validação contínua de identidade e na aplicação de políticas de segurança orientadas pelo contexto.

As seções também ressaltaram a importância de boas práticas na gestão de segredos, na automação segura e na validação contínua da configuração, reconhecendo esses componentes como necessários para assegurar a integridade e a confiabilidade da infraestrutura definida como código. Monstrou-se que a integração entre IaC e *Zero Trust* favorecem as necessidades atuais de segurança viabilizando a construção de ambientes computacionais mais resilientes, versionáveis e alinhados aos requisitos de *Security By Design*.

Essa base conceitual sustenta a transição para a próxima seção, que apresentará a parte prática por meio do ambiente de experimentação desenvolvido para esse minicurso, onde serão detalhados os elementos estruturantes da arquitetura proposta, bem como os mecanismos e ferramentas selecionados para sua implementação em um cenário voltado à segurança e à gestão da infraestrutura.

1.5. Ambiente de experimentação

Esta seção apresenta a experimentação com dois objetivos complementares: (1) traduzir em código os princípios teóricos da IaC e *Zero Trust* discutidos nas seções anteriores e (2) fornecer aos participantes um roteiro reproduzível que possa ser adaptado a cenários reais. Ao concluir a experimentação o participante deverá ser capaz de provisionar, segmentar e monitorar uma infraestrutura mínima, entendendo a diferença entre o modelo perimetral e ZTA no contexto da IaC.

1.5.1. Arquitetura de referência

Para cada fase da experimentação, será apresentada uma arquitetura de referência, que incluirá alguns serviços de rede típicos, selecionados por sua relevância e ampla adoção no mercado de TI.

Essa arquitetura base será mantida para as demais etapas para facilitar o entendimento e proporcionar ao participante a continuidade e aprofundamento nas práticas propostas. A figura 1.4 apresenta uma visão geral da arquitetura base.

Para tanto, os componentes escolhidos para as etapas de experimentação são:

- **GLPI (porta 8080)**: sistema de gestão de chamados e ativos.
- **MariaDB**: *backend* relacional para persistência do GLPI.
- **Servidor de Backup**: ambiente leve (utilizando uma distribuição Linux Alpine) para simulações de rotina de salvamento de dados via rede.
- **Servidor de Automação**: execução de rotinas com Ansible e Checkov para aplicação de configurações seguras.
- **Twingate Connector**: agente de acesso remoto seguro que evita exposição de portas e restringe acessos com base em identidade e postura de dispositivo.

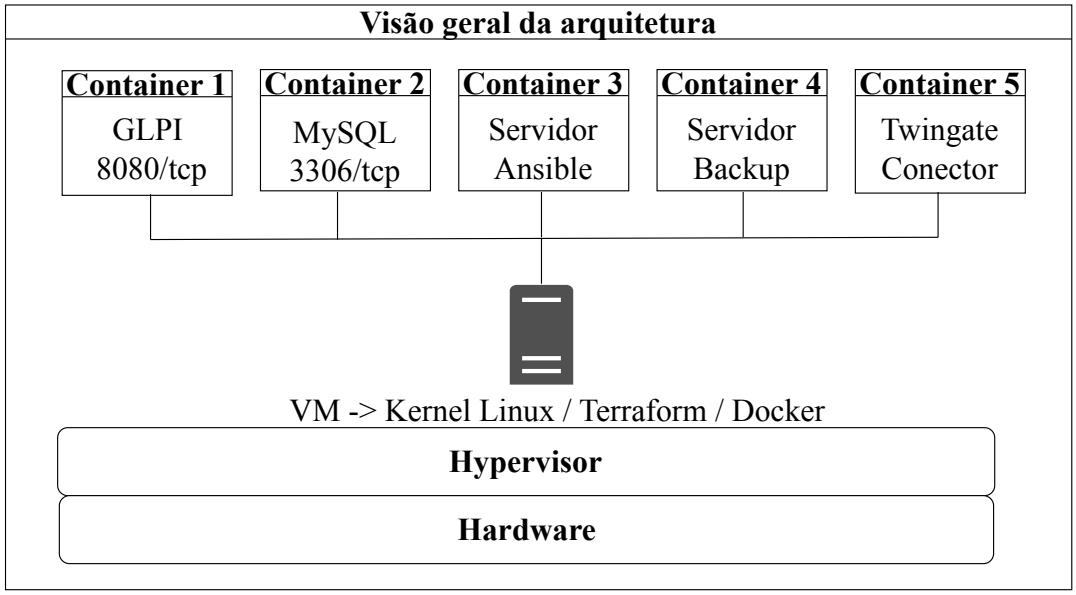

Figura 1.4. Visão geral da arquitetura do ambiente de experimentação

Para apoiar essa arquitetura, será utilizada a tecnologia de *containers*, permitindo isolamento e portabilidade entre as etapas do laboratório. Nesse contexto, destacam-se duas ferramentas fundamentais:

- Dockerfile: é um arquivo declarativo que descreve, passo a passo, como construir uma imagem Docker customizada. Nele são definidas as dependências, o ambiente de execução e os *scripts* necessários para inicializar os serviços.
- Docker Compose: é uma ferramenta que permite a orquestração de múltiplos *containers*. Utilizando um arquivo YAML, define como os serviços interagem, suas redes, volumes e demais parâmetros operacionais, simplificando a execução e manutenção de aplicações distribuídas.

Essas ferramentas serão exploradas na prática a partir da Etapa 1 da experimentação. A figura 1.5 ilustra a diferença conceitual entre Dockerfile e Docker Compose.

1.5.2. Descrição de componentes da arquitetura

- **Perímetro Virtual**: Refere-se a um agrupamento lógico de ativos críticos, como servidores de aplicação, bancos de dados, sistemas de automação (e.g., Ansible) e servidores de backup. Diferente do modelo tradicional baseado em perímetro físico, essa abordagem adota um perímetro definido por *software*, protegido por políticas de acesso baseadas em identidade, contexto e critérios explícitos de autorização.
- **Terraform/Ansible (IaC)**: O Terraform é utilizado para provisionar a infraestrutura de forma declarativa e automatizada, incluindo a criação dos *Connectors* do Twingate, assegurando rastreabilidade e padronização. De forma complementar, o Ansible atua na configuração pós-provisionamento, executando rotinas de automação

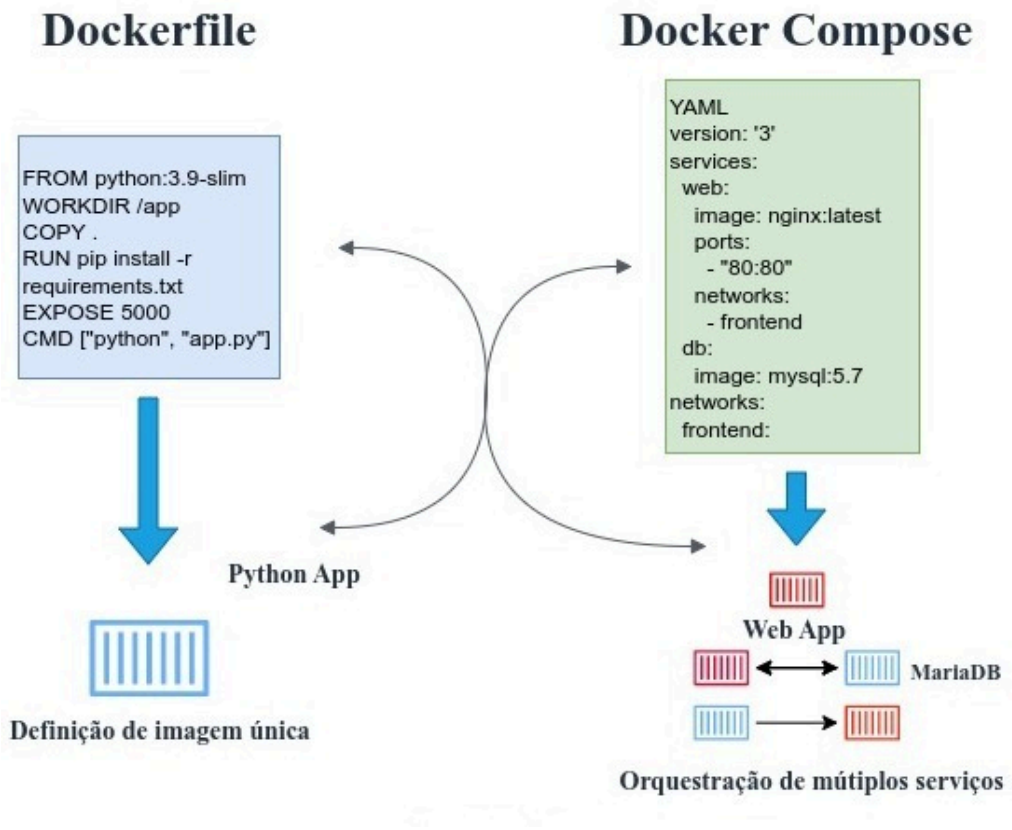

Figura 1.5. Diferença entre Dockerfile e Docker-compose

voltadas à aplicação de políticas de segurança, instalação de serviços e ajuste de parâmetros do sistema. Essa integração entre as ferramentas assegura a consistência, a conformidade e a eficiência na entrega dos ativos de infraestrutura.

- **Twingate**: Plataforma que substitui VPNs tradicionais ao oferecer acesso seguro, invisível e segmentado. Seus principais componentes são:
  - *Twingate Client*: Instalado no dispositivo do usuário, estabelece túneis criptografados para os recursos autorizados.
  - *Twingate Connector*: Componente leve implantado no Perímetro Virtual. Realiza conexões de saída para o plano de controle, sem exigir portas abertas no firewall.
  - *Twingate Control Plane*: Núcleo orquestrador disponível em nuvem, responsável por autenticação, verificação da postura dos dispositivos e aplicação de políticas, sem interferir no tráfego direto.

1.5.3. Fluxo Operacional da Arquitetura

A seguir é apresentada a sequencia de passos (*walkthrough*) para desenvolvimento da infraestrutura como código da experimentação.

1. **Definição da Infraestrutura como Código**: O administrador descreve a infraestrutura e as políticas de acesso via arquivos Terraform, especificando recursos e

conexões seguras.

2. **Provisionamento Automatizado**: Através do Terraform, os recursos são provisionados em nuvem ou *on-premises*. Pelo Ansible os *Connectors* do Twingate são implantados e configurados automaticamente.

3. **Solicitação de Acesso**: O usuário utiliza o Twingate Client para iniciar o acesso a um recurso específico.

4. **Autenticação e Autorização**: O Client comunica-se com o Control Plane, que realiza autenticação com provedores de identidade, como o Google Workspace ou Azure AD, e verifica a conformidade do dispositivo.

5. **Estabelecimento do Túnel Seguro**: Após autorização, um túnel *peer-to-peer* criptografado é estabelecido entre o *Client* e o *Connector*, garantindo privacidade e baixa latência.

6. **Acesso Específico ao Recurso**: O acesso é concedido apenas ao recurso autorizado, impedindo a visualização ou o tráfego para outros componentes da rede.

A arquitetura implementada no ambiente de experimentação combina os princípios do *Zero Trust Network Access* (ZTNA) e da *Infrastructure as Code* (IaC) com o objetivo de estabelecer um ambiente seguro, resiliente e automatizado, assegurando a proteção granular de recursos, credenciais e fluxos operacionais. Essa arquitetura é composta por dois componentes principais, que atuam de forma integrada para mitigar riscos e reduzir a superfície de exposição.

A Figura 1.6 ilustra uma arquitetura de segurança baseada no modelo de *Zero Trust* (Confiança Zero), cujo objetivo é proteger o acesso a recursos computacionais sensíveis por meio de segmentação e verificação contínua de identidade. A abordagem apresentada combina práticas de *Infrastructure as Code*, utilizando o Terraform para o provisionamento automatizado da infraestrutura, com a solução de acesso remoto seguro do Twingate, uma plataforma de *Zero Trust Network Access* (ZTNA).

O princípio central do modelo *Zero Trust* é sintetizado no princípio *"nunca confie, sempre verifique"*. Nenhum usuário ou dispositivo é confiável por padrão, independentemente de sua localização, sendo cada solicitação de acesso avaliada com base em identidade, contexto e postura de segurança.

1.5.4. Benefícios da Arquitetura

A seguir são relacionados os principais benefícios do cenário de experimentação.

- **Segurança granular**: Elimina-se a confiança implícita, com acesso fundamentado em identidade, contexto e segurança do dispositivo.

- **Superfície de ataque reduzida**: O uso exclusivo de conexões de saída e ausência de portas abertas limita significativamente as possibilidades de exploração externa.

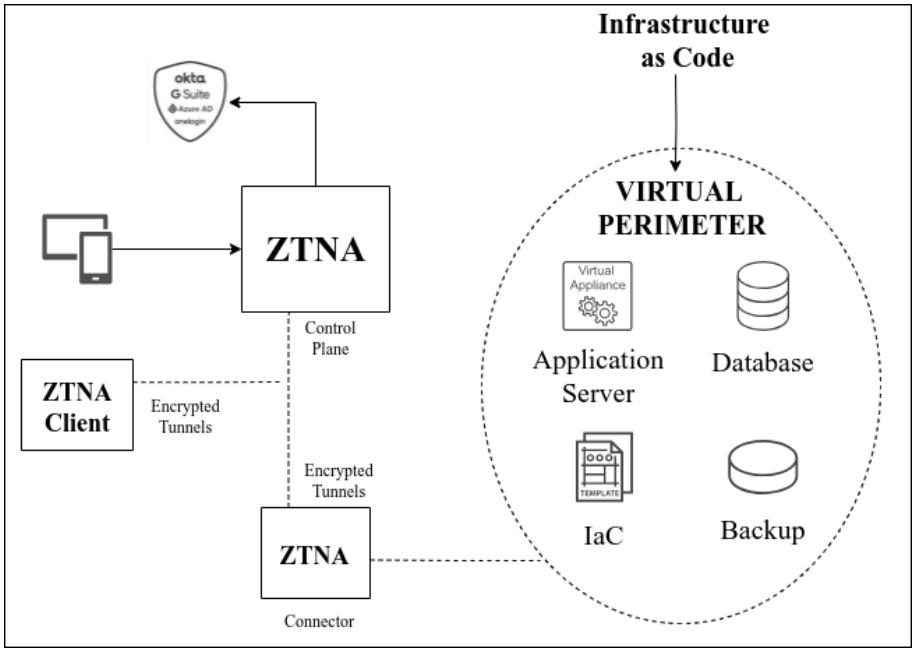

Figura 1.6. Visão geral da arquitetura do ambiente de experimentação.

- **Automação e gestão**: A configuração como código permite versionamento, auditoria de mudanças e replicação de ambientes com consistência e segurança.
- **Experiência transparente ao usuário**: O acesso é contínuo e automatizado, dispensando o uso de VPNs manuais.

1.5.5. Etapas da experimentação

Esse cenário laboratorial propõe a construção de um ambiente seguro e controlado por meio de quatro etapas evolutivas: provisionamento manual com Docker Compose, automação com Terraform, aplicação de políticas de segurança utilizando Ansible, Checkov e Twingate, e por fim, a fase de testes e validação da infraestrutura.

O objetivo é demonstrar como o modelo da IaC, aliado ao paradigma de *Zero Trust*, pode elevar substancialmente o nível de segurança e gestão em ambientes computacionais modernos.

Ao longo das atividades práticas, os participantes percorrem uma trajetória que vai desde o provisionamento tradicional com Docker Compose até a implementação de uma arquitetura automatizada e segura, baseada em segmentação lógica, validação contínua e controle de acesso rigoroso, incorporando os princípios da IaC de forma aplicada.

Etapa 1 - Provisionamento com Docker Compose

Nesta etapa introdutória, o ambiente da aplicação é configurado manualmente utilizando o Docker Compose. O objetivo é familiarizar os participantes com os componentes que serão posteriormente automatizados, fornecendo uma base prática para a compreensão da infraestrutura.

As instruções completas para a realização dessa etapa estão disponíveis no re-

positório do minicurso no GitLab[33], onde são apresentados, de forma estruturada, os procedimentos necessários para a inicialização dos *containers* e preparação do ambiente.

Durante essa fase, os participantes são orientados a analisar o conteúdo do arquivo Dockerfile, responsável pela definição e construção das imagens utilizadas no projeto, bem como a estrutura do arquivo `docker-compose.yml`, que organiza os serviços, redes e volumes empregados na composição do ambiente.

Além disso, são realizados procedimentos de inspeção e análise do ambiente por meio de comandos Docker, permitindo observar *logs*, configurações e o comportamento dos *containers* em execução.

Esta etapa tem como finalidade apresentar os fundamentos práticos da infraestrutura da aplicação e estabelecer uma base comparativa para as atividades de automação que serão desenvolvidas nas etapas seguintes.

Etapa 2 - Provisionamento automatizado com Terraform (IaC)

Nesta fase do minicurso, busca-se aplicar os princípios da *Infrastructure as Code* (IaC) por meio da ferramenta Terraform, com o objetivo de automatizar o provisionamento dos mesmos recursos anteriormente configurados manualmente.

Essa comparação permite evidenciar os benefícios operacionais em termos de rastreabilidade, padronização e reprodutibilidade.

Na sequência, introduzem-se os fundamentos da infraestrutura como código e prepara o ambiente para configurações adicionais nas etapas seguintes do minicurso.

As instruções detalhadas para a execução estão disponíveis no repositório do minicurso no GitLab[33], abrangendo desde a estruturação dos arquivos até a execução do provisionamento automatizado.

Etapa 3 - Aplicação de segurança com Ansible (IaC) e Checkov

Na terceira etapa do minicurso, são aplicadas práticas de automação e validação da infraestrutura provisionada por meio das ferramentas Ansible e Checkov, com foco em reforço da segurança e conformidade. Esta fase complementa o provisionamento realizado anteriormente, agregando camadas de configuração, inspeção e boas práticas de segurança.

Entre os tópicos abordados, destacam-se a análise dos arquivos de configuração (.tf), a configuração do `SSH` e chave pública nos *containers* e a comparação entre as abordagens manual (Docker Compose) e automatizada (Terraform).

Dentre as ações previstas nesta etapa, destacam-se:

- Aplicação de rotinas de configuração automatizada em serviços como o GLPI, utilizando *playbooks* do Ansible;
- Implementação de práticas de segurança, como a habilitação de HTTPS, a configuração de certificados digitais e ajustes nos serviços para reduzir vulnerabilidades;
- Utilização da ferramenta Checkov para validar os arquivos de infraestrutura declarativa, identificando potenciais falhas de conformidade antes da aplicação em

[33] https://projects.ppgia.pucpr.br/secplab/sbseg25-IaC-ZT

ambientes produtivos.

A integração entre Ansible e Checkov exemplifica uma abordagem DevSecOps no contexto da *Infrastructure as Code*, permitindo que políticas de segurança e requisitos operacionais sejam aplicados de forma automatizada e auditável. Esta etapa fortalece a confiabilidade da infraestrutura e prepara o ambiente para integração com mecanismos de controle de acesso e microssegmentação.

As instruções detalhadas para execução estão disponíveis no repositório do minicurso no GitLab[33], com roteiros organizados para facilitar a aplicação das tarefas automatizadas.

Etapa 4 - Diagnóstico e validação do ambiente provisionado

Na etapa final do minicurso, são realizados testes de conectividade, diagnóstico de rede e análise de exposição de serviços, para validar os mecanismos de segmentação e controle de acesso implementados ao longo do ambiente provisionado.

As instruções detalhadas para a execução dessa etapa encontram-se disponíveis no repositório do minicurso no GitLab[33], incluindo os comandos e procedimentos recomendados para simulação de acesso e varredura.

Durante esta fase, os participantes devem realizar operações básicas de diagnóstico, utilizando comandos como `ping, curl, traceroute, dig`, entre outros, com o propósito de verificar a conectividade entre os serviços e identificar possíveis falhas de comunicação.

Além disso, são conduzidos testes de acesso seguro, simulando tentativas de conexão a serviços protegidos pelo mecanismo de *Zero Trust Network Access* (ZTNA), implementado com a solução Twingate. O objetivo é observar as restrições de visibilidade impostas, validando se a política de microssegmentação está sendo aplicada de forma adequada.

Como parte da simulação de ameaças, é utilizada a ferramenta `nmap` para realizar varreduras de portas e serviços, alinhando a atividade à tática TA0007 – Discovery do *framework* MITRE ATT&CK. Essa prática permite avaliar a eficácia das medidas de proteção frente a tentativas de mapeamento da infraestrutura por agentes não autorizados.

Por fim, com base nas respostas obtidas e na visibilidade dos serviços, os participantes devem refletir sobre o grau de exposição da infraestrutura e a efetividade das medidas de segurança aplicadas ao longo do ciclo de provisionamento.

Essas etapas estão resumidas na Tabela 1.3 com as ferramentas utilizadas, atividades e abordagem de segurança aplicada.

1.5.6. Considerações Finais

A integração entre o modelo *Zero Trust* e práticas da IaC representa uma evolução na proteção de ambientes computacionais modernos. A combinação do Terraform e Ansible com o Twingate viabiliza a implementação prática de políticas de acesso segmentado e com alterações auditável, promovendo escalabilidade, segurança e conformidade regulatória. Trata-se de uma abordagem estratégica para organizações que buscam reduzir riscos,

Tabela 1.3. Resumo das Etapas do Laboratório Guiado com IaC e *Zero Trust*

Etapa	Ferramentas	Atividades Principais	Abordagem de Segurança
1	Docker Compose	Provisionamento manual dos *containers* GLPI, MariaDB, Backup e Ansible.	Sem controle de acesso ou segmentação; ambiente exposto.
2	Terraform (IaC)	Reprovisionamento automatizado dos recursos da Etapa 1.	Provisionamento automatizado com rastreabilidade e início da proteção declarada.
3	Checkov + Terraform + Ansible	Análise estática com Checkov, Reprovisionamento com Terraform, Aplicação de configuração HTTPS (GLPI), e inicialização do serviço Twingate.	Validação pré-provisionamento e aplicação de políticas de segurança automatizadas.
4	Ansible + Twingate	Testes de conectividade segura via Twingate. Validação de segmentação, visibilidade mínima e política de acesso.	Acesso remoto seguro com base em identidade e postura do dispositivo (modelo *Zero Trust*).

controlar acessos com precisão e manter a resiliência operacional em ambientes híbridos ou distribuídos.

1.6. Laboratório guiado: Provisionamento com Docker Compose, Terraform, Ansible e *Zero Trust*

Esta seção apresenta um cenário de laboratório dividido em quatro etapas evolutivas, que podem ser realizadas de forma autônoma pelos participantes.

O laboratório tem como objetivo mostrar, na prática, a transição de um ambiente manual para uma infraestrutura automatizada, segura e baseada no modelo *Zero Trust*.

Para a execução das atividades, todos os arquivos, *scripts* e instruções detalhadas estão disponíveis no repositório GitLab oficial do minicurso. Cada etapa está alinhada aos fundamentos discutidos na seção 1.5.5, promovendo o reforço conceitual por meio da prática aplicada.

Etapa 1 – Provisionamento Manual com Docker Compose

Nesta etapa inicial, os serviços GLPI, MariaDB, servidor de *backup* e servidor de automação são configurados de forma manual, utilizando arquivos declarativos do Docker Compose. A atividade permite compreender a estrutura básica do ambiente, suas dependências e a organização da rede interna entre os *containers*. Essa abordagem serve como base para comparações posteriores com processos automatizados.

Etapa 2 – Provisionamento Automatizado com Terraform (IaC)

Com o uso do Terraform, os mesmos recursos da etapa anterior são recriados mas de forma automatizada, adotando o modelo declarativo da IaC. A automação introduz benefícios como rastreabilidade, versionamento e controle de estado.

Etapa 3 – Configuração com Ansible e Validação com Checkov

Após o provisionamento, são aplicadas rotinas de configuração com Ansible, que incluem práticas de melhoria de segurança, ajustes de HTTPS, instalação e ativação do

Twingate. Também é utilizada a ferramenta Checkov para validação da infraestrutura declarada, com foco na conformidade e prevenção de falhas antes da aplicação final. Essa etapa consolida a segurança do ambiente antes de seu uso.

Etapa 4 – Integração *Zero Trust* e Diagnóstico de Rede

Na etapa final, são realizados testes de conectividade e diagnóstico para validar o isolamento e as restrições de acesso aplicadas com o Twingate. Os participantes simulam cenários de descoberta de rede e varredura de serviços, observando a aplicação das políticas de segmentação e visibilidade mínima. A atividade reforça o entendimento sobre os efeitos práticos da arquitetura *Zero Trust* no controle de acessos e na redução da superfície de ataque.

Este ensaio de laboratório propõe a construção de um ambiente seguro e controlado por meio de quatro etapas evolutivas: provisionamento manual com Docker, automação com Terraform e aplicação de políticas de segurança com Ansible, Checkov e Twingate. O objetivo é demonstrar como o modelo de *Infrastructure as Code* (IaC), quando aliado ao paradigma de Confiança Zero (*Zero Trust*), pode elevar significativamente o nível de segurança e governança em ambientes computacionais modernos.

Durante a experimentação os participantes transitam de um provisionamento tradicional (com Docker Compose) até uma arquitetura segura e automatizada, baseada em segmentação, validação contínua e controle restrito de acesso, utilizando IaC.

Ao final da experimentação, espera-se que os participantes:

- Compreendam o ciclo completo de provisionamento e proteção de infraestrutura, desde a criação de ambientes até a aplicação de políticas de segurança automatizadas;
- Reflitam sobre os benefícios da IaC na redução de falhas humanas, na padronização de ambientes e na capacidade de rastrear mudanças;
- Sejam capazes de integrar práticas modernas de segurança, análise estática de infraestrutura, e acesso remoto seguro via Twingate;
- Visualizem, por meio da aplicação prática, o uso de *Zero Trust* em ambientes controlados e reprodutíveis, mitigando riscos e fortalecendo a postura de segurança;
- Pèrcebam na experimentação práticas reais de proteção, como a execução de varreduras de rede com ferramentas como o `nmap`, relacionadas à tática *Discovery* (TA0007) do MITRE ATT&CK, ampliando a compreensão sobre possíveis vetores de ataque e medidas preventivas.
- Dessa forma, o minicurso proporciona não apenas uma abordagem conceitual, mas também uma vivência prática sobre a construição de ambientes mais resilientes, versionáveis e alinhados com os desafios atuais da cibersegurança.

As quatro etapas descritas nesta seção estruturam o percurso prático de transição entre um ambiente tradicional e uma arquitetura segura com aplicação dos princípios de

Zero Trust. A descrição fornecida tem caráter conceitual, destacando os papéis e objetivos de cada ferramenta.

Todos os recursos necessários, incluindo *scripts*, arquivos de configuração e orientações técnicas, estão organizados no repositório oficial do minicurso, cuja descrição detalhada pode ser consultada na [subsec:repositorio]Subseção 1.6.4 (Estrutura de Arquivos e Repositório GitLab).

1.6.1. Objetivos de aprendizagem

a) Visualizar a diferença entre provisionamento manual e declarativo;

b) Avaliar ferramentas distintas para construir imagens de *containers* e provisionamento de infraestrutura como Dockerfile, Docker compose e Terraform;

c) Compreender noções de funcionamento e aplicação das ferramentas da IaC (Terraform e Ansible);

d) Compreender como práticas de segurança podem ser automatizadas com IaC;

e) Aplicar *Zero Trust* para restringir acessos, reduzir exposição e proteger os ativos digitais;

f) Avaliar os benefícios de políticas de segmentação e validação de dispositivos em ambientes reais.

1.6.2. Cenário de Laboratório

A execução do cenário de laboratório pode ser realizada de forma autônoma, mesmo na ausência de tutor. Cada uma das quatro etapas descritas na seção anterior está detalhadamente documentada no repositório GitLab do minicurso, incluindo os arquivos de configuração, *scripts* e orientações de execução.

Para cada etapa, recomenda-se que o participante siga a sequência de instruções disponível nos respectivos arquivos README.md, presentes em cada diretório do repositório. Esses arquivos contêm explicações sobre o objetivo da etapa, dependências envolvidas e passos operacionais necessários. A estrutura modular adotada permite que o cenário seja reproduzido com flexibilidade em diferentes ambientes, promovendo autonomia no aprendizado.

1.6.3. Requisitos

Antes de iniciar o laboratório, é essencial que o ambiente do participante esteja preparado com as ferramentas necessárias para a execução das etapas propostas. São requeridos: Docker (versão 20.10 ou superior), Docker Compose, Terraform (versão 1.0 ou superior), Ansible, Checkov e Git (utilizado para clonar o repositório do projeto). Esses componentes são fundamentais para viabilizar a construção progressiva da infraestrutura, desde o provisionamento inicial até a aplicação automatizada de políticas de segurança.

O sistema operacional recomendado é Linux, preferencialmente com suporte a *containers* (como Ubuntu ou Debian). Caso o participante não disponha de um ambiente

Tabela 1.4. Estrutura de diretórios do repositório GitLab

Diretório	Conteúdo Principal
`/etapa1/`	Arquivos do Docker Compose e `Dockerfile` para provisionamento manual
`/etapa2/`	Arquivos `.tf` para provisionamento automatizado com Terraform
`/etapa3/`	*Playbooks* do Ansible, arquivos de configuração HTTPS e validação com Checkov
`/etapa4/`	*Scripts* de testes de conectividade e varredura, voltados à análise com ZTNA
`/docs/`	Documentação complementar com requisitos, instruções gerais e guias de instalação
`README.md`	Arquivo principal com introdução ao laboratório, instruções de uso e licenciamento

compatível, é possível utilizar uma máquina virtual com distribuição Linux ou *containers* pré-configurados.

Os *links* para instalação, versões recomendadas e instruções de configuração estão disponíveis no diretório docs/ambiente do repositório GitLab do minicurso.

1.6.4. Estrutura de Arquivos e Repositório GitLab

O repositório do minicurso no GitLab[34] foi organizado de forma modular, com diretórios separados para cada etapa do ensaio. Isso permite que os participantes identifiquem facilmente os arquivos e *scripts* relacionados a cada fase da experimentação.

A tabela 1.4 apresenta um resumo da estrutura principal do repositório. Essa estrutura foi projetada para facilitar a navegação, a execução progressiva das etapas e o reuso dos componentes em outros projetos de infraestrutura.

1.6.5. Segurança com *Zero Trust* no ambiente

A integração com o modelo *Zero Trust* não é apenas complementar, mas central para o ensaio de laboratório. Neste contexto, ele é implementado por meio do Twingate, que incorpora os seguintes princípios de segurança:

- **Microssegmentação de serviços**: o acesso é concedido por recurso, e não à rede inteira. Por exemplo, um usuário pode acessar apenas o GLPI, sem ter visibilidade dos servidores de backup ou banco de dados.
- **Eliminação da confiança implícita**: a localização na rede (interna ou externa) não é critério de confiança. Todo acesso é autenticado, autorizado e registrado.
- **Acesso baseado em identidade e contexto**: o Twingate Control Plane realiza autenticação integrada a diretórios (AD, Google Workspace etc.) e verifica postura de segurança dos dispositivos.
- **Redução da superfície de ataque**: os serviços não precisam expor portas TCP públicas nem estar acessíveis na internet. O conector Twingate inicia a conexão de forma reversa (saída), mantendo os ativos invisíveis para escaneamentos.

Essa abordagem reforça a confidencialidade, integridade e disponibilidade dos recursos, mesmo em ambientes com múltiplos usuários ou acessos remotos.

[34]Disponível em: https://projects.ppgia.pucpr.br/secplab/sbseg25-IaC-ZT

1.6.6. Integração Progressiva do *Zero Trust* no ensaio de laboratório

Ao longo das quatro etapas, os participantes acompanham a transição de um ambiente tradicional e com exposição a ataques para uma infraestrutura segura, automatizada e aderente a *Zero Trust*. O Twingate é gradualmente integrado como ferramenta de microssegmentação e controle de acesso, conforme descrito a seguir:

- **Etapa 1 (Docker Compose)**: os serviços GLPI, MariaDB, backup e Ansible são expostos diretamente à rede interna, sem qualquer controle de identidade ou segmentação. Trata-se de um ambiente vulnerável por padrão, configurado manualmente via `docker-compose.yml`.
- **Etapa 2 (Terraform)**: os mesmos serviços são provisionados de forma automatizada por meio do Terraform. Embora o ambiente ainda não esteja conectado ao Twingate, foram executados comandos que preparam a infraestrutura para a etapa seguinte.
- **Etapa 3 (Ansible + Checkov)**: nesta fase, são aplicadas configurações de melhoria de segurança no GLPI, ativação do protocolo HTTPS e validações automatizadas com o Checkov. Adicionalmente, o serviço do Twingate é instalado e iniciado, viabilizando a aplicação de políticas de acesso e garantindo que os ativos permaneçam invisíveis fora do escopo autorizado.
- **Etapa 4 (Ansible + Twingate)**: os testes finais são acessos aos serviços via Twingate, validando o funcionamento da segmentação e a aplicação das políticas configuradas. O acesso é autenticado, segmentado e limitado com base no princípio da confiança mínima.

O *Zero Trust* no ambiente de experimentação, atua como o eixo transversal de cibersegurança, oferecendo autenticação baseada em identidade, controle de postura de dispositivo, política de acesso mínimo e eliminação da confiança implícita.

1.6.7. Política de Acesso (Exemplo *Zero Trust*)

A política demonstrada no Código 1.4 exemplifica como uma abordagem *Zero Trust* define explicitamente os recursos, os perfis permitidos, a postura esperada dos dispositivos e os requisitos de autenticação adicional (como MFA e geolocalização).

1.6.8. Síntese: Benefícios da Integração IaC + *Zero Trust*

A abordagem integrada neste laboratório promove:

- **Gestão de configuração**: com Terraform e Ansible, toda a infraestrutura e suas regras são codificadas e versionáveis.
- **Segurança na origem (Shift-Left)**: com Checkov, falhas conhecidas são detectadas antes da execução da infraestrutura.
- **Resiliência e visibilidade reduzida**: com Twingate, o ambiente se torna invisível a agentes externos e acessível apenas mediante autenticação.

Código Fonte 1.4. Política Zero Trust simulada para acesso ao banco de dados.

```
policy_name: "db-prod-access"
description: "Controla o acesso ao banco de dados principal."

resources:
  - "database.prod.internal"

principals:
  - group: "database_admins"
  - user: "service_account_backup@example.com"

conditions:
  - device_posture: "compliant"
  - mfa: "required"
  - location: "BR"

action: "allow"
```

- **Eficiência operacional**: a reimplantação de ambientes seguros pode ser feita em minutos com um único comando.

Essa integração reflete as tendências mais atuais em cibersegurança, possibilitando que mesmo equipes pequenas apliquem práticas de elevado nível na proteção de seus ambientes computacionais.

1.7. Considerações sobre a experimentação

Esta seção final retoma os principais conceitos abordados ao longo do minicurso, com ênfase na integração entre *Infrastructure as Code* (IaC) e o modelo de segurança *Zero Trust*. A proposta foi conduzir os participantes por uma trilha estruturada, da fundamentação teórica à aplicação prática, promovendo uma visão crítica e aplicada aos desafios atuais da segurança em ambientes automatizados.

1.7.1. Integração entre IaC e *Zero Trust*: Conceitos Fundamentais

A combinação entre IaC e *Zero Trust* representa uma evolução em relação aos modelos tradicionais de segurança baseados em perímetro. Essa integração oferece maior resiliência operacional, adaptabilidade e proteção dinâmica, com base em três princípios:

- **Confiança mínima**: nenhum recurso ou identidade deve ser implicitamente confiável;
- **Validação contínua**: autenticações e autorizações devem ocorrer a cada nova interação;
- **Visibilidade segmentada**: controle granular sobre acessos, horários e escopos de interação.

Esses princípios foram operacionalizados na arquitetura experimental proposta ao longo das atividades práticas.

1.7.2. Automação, Versionamento e Conformidade Adaptativa

A utilização de arquivos declarativos via IaC viabiliza o provisionamento automatizado, auditável e consistente da infraestrutura. Essa abordagem reduz a incidência de erros manuais e facilita o versionamento, contribuindo para uma recuperação rápida em cenários de falha [Wang 2022].

Quando combinada ao modelo *Zero Trust Network Access* (ZTNA), essa estratégia potencializa:

- A conformidade dinâmica com políticas de segurança em constante evolução;
- A resposta proativa a ameaças emergentes;
- A reprodutibilidade de ambientes com segurança incorporada desde a concepção.

1.7.3. Microssegmentação e Redução da Superfície de Ataque

A microssegmentação, pilar estruturante do *Zero Trust*, foi aplicada para isolar recursos críticos como o GLPI e o banco de dados MariaDB, limitando os fluxos de rede a conexões estritamente autorizadas. Essa estratégia inibe movimentações laterais em caso de comprometimento inicial.

A autenticação contextual, baseada na análise contínua de identidade, localização e permissões, contribui para reduzir riscos de reutilização de credenciais e escalonamento indevido de privilégios.

1.7.4. Competências Desenvolvidas e Aplicação Prática

Ao longo do minicurso, os participantes foram desafiados a aplicar os conceitos em um ambiente realista, utilizando ferramentas modernas e práticas seguras de provisionamento. Espera-se que, ao final, tenham desenvolvido as seguintes competências:

- Provisionar ambientes seguros com ferramentas da IaC;
- Aplicar os princípios fundamentais de *Zero Trust*;
- Avaliar criticamente estratégias de segurança adotadas em infraestrutura;
- Identificar vulnerabilidades e propor melhorias com base na experimentação.

1.7.5. Conclusão

A experimentação conduzida reforça o papel da IaC e do modelo *Zero Trust* como fundamentos para a construção de ambientes seguros, versionáveis e alinhados às práticas contemporâneas de DevSecOps e gestão de cibersegurança.

A arquitetura utilizada mostrou, de forma prática, que é possível incorporar controles como verificação contínua, isolamento de recursos e autenticação contextual em

fluxos automatizados de infraestrutura. Com isso, foi evidenciada a viabilidade técnica e o potencial formativo dessa abordagem, promovendo sua aplicabilidade em contextos reais de operação e segurança organizacional.

Referências

[Abreu et al. 2020] Abreu, V., Santin, A. O., Viegas, E. K., and Cogo, V. V. (2020). *Identity and Access Management for IoT in Smart Grid*, page 1215–1226. Springer International Publishing.

[Abreu et al. 2017] Abreu, V., Santin, A. O., Viegas, E. K., and Stihler, M. (2017). A multi-domain role activation model. In *2017 IEEE International Conference on Communications (ICC)*, page 1–6. IEEE.

[Alonso et al. 2023] Alonso, J., Piliszek, R., and Cankar, M. (2023). Embracing iac through the devsecops philosophy: Concepts, challenges, and a reference framework. *IEEE Software*, 40(1):56–62.

[Basak et al. 2022] Basak, S. K., Neil, L., Reaves, B., and Williams, L. (2022). What are the practices for secret management in software artifacts? In *2022 IEEE Secure Development Conference (SecDev)*, page 69–76. IEEE.

[Boulden 2023] Boulden, S. (2023). Sigstore - signature sorcery. RXRW Blog. Accessed 2025-06-14.

[Center for Internet Security (CIS)] Center for Internet Security (CIS). Foundational cloud security with cis benchmarks. Blog Post.

[Chen et al. 2018] Chen, W., Wu, G., and Wei, J. (2018). An approach to identifying error patterns for infrastructure as code. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 124–129.

[Chiari et al. 2022] Chiari, M., Pascalis, M. D., and Pradella, M. (2022). Static analysis of infrastructure as code: a survey. *arXiv preprint arXiv:2206.10344*.

[CloudOptimo 2025] CloudOptimo (2025). Detecting orphaned resources using aws config rules. Acesso em: 14 jun. 2025.

[de Oliveira et al. 2023] de Oliveira, P. R., Santin, A. O., Horchulhack, P., Viegas, E. K., and de Matos, E. (2023). A dynamic network-based intrusion detection model for industrial control systems. In *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, page 1496–1501. IEEE.

[Filho et al. 2025] Filho, A. G., Viegas, E. K., Santin, A. O., and Geremias, J. (2025). A dynamic network intrusion detection model for infrastructure as code deployed environments. *Journal of Network and Systems Management*, 33(4).

[Fowler 2013] Fowler, M. (2013). Infrastructure as code. Publicado originalmente no blog de Martin Fowler com contribuições de Kief Morris.

[Horchulhack et al. 2022a] Horchulhack, P., Viegas, E. K., and Santin, A. O. (2022a). Detection of service provider hardware over-commitment in container orchestration environments. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, page 6354–6359. IEEE.

[Horchulhack et al. 2022b] Horchulhack, P., Viegas, E. K., and Santin, A. O. (2022b). Detection of service provider hardware over-commitment in container orchestration environments. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, page 6354–6359. IEEE.

[Kalu et al. 2025] Kalu, K. G., Okorafor, S., Singla, T., Torres-Arias, S., and Davis, J. C. (2025). Why johnny signs with sigstore: Examining tooling as a factor in software signing adoption in the sigstore ecosystem. In *arXiv preprint arXiv:2503.00271*.

[Lorenc 2021] Lorenc, D. (2021). Cosign image signatures. *Sigstore Blog*. Accessed 2025-06-14.

[Mangera 2025] Mangera, V. (2025). Shedding light on orphaned cloud resources – ghosts haunting. LinkedIn. Acesso em: 14 jun. 2025.

[Marquis 2024] Marquis, Y. A. (2024). From theory to practice: Implementing effective role-based access control strategies to mitigate insider risks in diverse organizational contexts. *Journal of Engineering Research and Reports*, 26(5):138–154.

[Matthew Kosinski 2024] Matthew Kosinski, A. F. (2024). Identity and access management?

[Mell and Grance 2011] Mell, P. and Grance, T. (2011). The nist definition of cloud computing. Special Publication 800-145, National Institute of Standards and Technology (NIST).

[MITRE ATT&CK 2024] MITRE ATT&CK (2024). MITRE ATT&CK Framework. https://attack.mitre.org/. Acesso em: 12 set. 2024.

[NIST 2025] NIST (2025). How do i create a good password? Technical report, National Institute of Standards and Technology. Acesso em: 08 de maio de 2025.

[Oliveira et al. 2024] Oliveira, J., Santin, A., Viegas, E., and Horchulhack, P. (2024). A non-interactive one-time password-based method to enhance the vault security. In Barolli, L., editor, *Advanced Information Networking and Applications*, pages 201–213, Cham. Springer Nature Switzerland.

[Queen 2024] Queen, C. (2024). What is infrastructure as code security? Accessed: 2024-09-14.

[Rahman et al. 2019a] Rahman, A., Mahdavi-Hezaveh, R., and Williams, L. (2019a). A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108:65–77.

[Rahman et al. 2019b] Rahman, A., Parnin, C., and Williams, L. (2019b). The seven sins: Security smells in infrastructure as code scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 164–175.

[Rahman et al. 2021a] Rahman, A., Rahman, M. R., Parnin, C., and Williams, L. (2021a). Security smells in ansible and chef scripts: A replication study. *ACM Transactions on Software Engineering and Methodology*, 30(1):1–31.

[Rahman et al. 2021b] Rahman, A., Rahman, M. R., Parnin, C., and Williams, L. (2021b). Security smells in ansible and chef scripts: A replication study. *ACM Trans. Softw. Eng. Methodol.*, 30(1).

[Rahman and Anwar 2021] Rahman, M. A. and Anwar, Z. (2021). Secret management in cloud native environments. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, pages 54–62.

[Rahman et al. 2022] Rahman, M. R., Imtiaz, N., Storey, M.-A., and Williams, L. (2022). Why secret detection tools are not enough: It's not just about false positives - an industrial case study. *Empirical Software Engineering*, 27(3).

[Reddy Konala et al. 2023] Reddy Konala, P. R., Kumar, V., and Bainbridge, D. (2023). Sok: Static configuration analysis in infrastructure as code scripts. In *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 281–288.

[Rodrigues et al. 2025] Rodrigues, M. G., Viegas, E. K., Santin, A. O., and Enembreck, F. (2025). A mlops architecture for near real-time distributed stream learning operation deployment. *Journal of Network and Computer Applications*, 238:104169.

[Rose et al. 2020] Rose, S., Borchert, O., Mitchell, S., and Connelly, S. (2020). Zero trust architecture. NIST Special Publication 800-207, National Institute of Standards and Technology, Gaithersburg, MD. Acesso em: 08 de maio de 2025.

[Silveira Neto et al. 2024] Silveira Neto, M., Malucelli, A., and Reinehr, S. (2024). Can personality types be blamed for code smells? pages 196–205.

[Simioni et al. 2025a] Simioni, J., Viegas, E. K., Santin, A., and Horchulhack, P. (2025a). An early exit deep neural network for fast inference intrusion detection. In *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*, SAC '25, page 730–737. ACM.

[Simioni et al. 2025b] Simioni, J. A., Viegas, E. K., Santin, A. O., and de Matos, E. (2025b). An energy-efficient intrusion detection offloading based on dnn for edge computing. *IEEE Internet of Things Journal*, 12(12):20326–20342.

[Syed et al. 2022] Syed, N. F., Shah, S. W., Shaghaghi, A., Anwar, A., Baig, Z., and Doss, R. (2022). Zero trust architecture (zta): A comprehensive survey. *IEEE Access*, 10:57143–57179.

[Vehent, Julien 2018] Vehent, Julien (2018). *Securing DevOps: Security in the Cloud*, page 384. Simon and Schuster.

[Viegas et al. 2020] Viegas, E., Santin, A., Bachtold, J., Segalin, D., Stihler, M., Marcon, A., and Maziero, C. (2020). Enhancing service maintainability by monitoring and auditing sla in cloud computing. *Cluster Computing*, 24(3):1659–1674.

[Viegas et al. 2017] Viegas, E., Santin, A., Neves, N., Bessani, A., and Abreu, V. (2017). A resilient stream learning intrusion detection mechanism for real-time analysis of network traffic. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, page 1–6. IEEE.

[Wang 2022] Wang, R. (2022). *Infrastructure as Code Patterns and Practices: With Examples in Python and Terraform*. Book Collection ITPro. Manning, Shelter Island, NY, 1st edition. Acesso em: 07 maio de 2025.

[Wettinger et al. 2014] Wettinger, J., Breitenbücher, U., and Leymann, F. (2014). Comparing and combining deployment automation approaches. In *Proceedings of the 2014 International Conference on Web Services (ICWS)*, pages 305–312. IEEE.

[Yuliarman et al. 2023] Yuliarman, S., Ridwan, S. H., and Resi, S. B. (2023). *Implementation of Hashicorp Vault as Multi-Factor Data Communication Security in Integration with Modern Infrastructure*. SNTE.

[Özdoğan et al. 2023] Özdoğan, E., Ceran, O., and ÜSTÜNDAĞ, M. (2023). Systematic analysis of infrastructure as code technologies. *Gazi University Journal of Science Part A: Engineering and Innovation*, 10.