**Documentação Técnica  
Infraestrutura AWS com Terraform**

Henrique Rocha Bomfim  
TecHacker – 7º Semestre de Engenharia da Computação

Nome: Henrique Rocha Bomfim TecHacker 7º Sem Eng Comp

Documentação Completa do Projeto: Infraestrutura AWS com Terraform

# 1. Introdução e Objetivo do Projeto

Este documento detalha a conceção, implementação e configuração de uma infraestrutura robusta e segura na Amazon Web Services (AWS) utilizando Terraform. O objetivo principal é hospedar uma aplicação web moderna, composta por um frontend (Next.js) e um backend (API FastAPI com PostgreSQL), em múltiplos ambientes (ex: Desenvolvimento, Produção). A infraestrutura foi projetada com foco em segurança, alta disponibilidade, monitoramento e automação, incorporando ferramentas como Wazuh para deteção de intrusão (IDS/HIDS), um Application Load Balancer (ALB) para distribuição de tráfego e acesso via DNS, e um pipeline de CI/CD utilizando GitHub Actions para automação de deploy. Esta documentação serve como um registo completo do desenvolvimento e da arquitetura final implantada.

# 2. Configuração Inicial do Ambiente AWS e Desenvolvimento

## 2.1. Criação de Utilizador IAM Admin e Configuração de Credenciais

Conforme as melhores práticas da AWS, um utilizador IAM com permissões administrativas foi criado para gerir os recursos, evitando o uso da conta root. As credenciais de acesso programático (Access Key ID e Secret Access Key) deste utilizador foram configuradas localmente utilizando o AWS Command Line Interface (AWS CLI) com aws configure, estabelecendo o perfil default que o Terraform utiliza para autenticação.

2.2. Instalação do Terraform

O Terraform foi instalado no ambiente de desenvolvimento local para permitir a definição e o provisionamento da infraestrutura como código. O PATH do sistema foi configurado para incluir o executável do Terraform.

# 3. Estrutura do Projeto Terraform (INFRA-PROJETO-AWS)

O projeto Terraform foi organizado de forma modular para promover a reutilização e a clareza, e preparado para gerir múltiplos ambientes.

## 3.1. Visão Geral da Estrutura de Ficheiros

INFRA-PROJETO-AWS/

├── modules/

│ ├── ec2/ # Módulo para instâncias EC2

│ │ ├── main.tf

│ │ ├── outputs.tf

│ │ └── variables.tf

│ ├── security/ # Módulo para Security Groups

│ │ ├── main.tf

│ │ ├── outputs.tf

│ │ └── variables.tf

│ └── vpc/ # Módulo para VPC e componentes de rede

│ ├── main.tf

│ ├── outputs.tf

│ └── variables.tf

├── main.tf # Configuração principal do módulo raiz (orquestração)

├── variables.tf # Declaração de variáveis de entrada globais do projeto

├── terraform.tfvars # Valores padrão ou para o workspace 'default'

├── dev.tfvars # Valores específicos para o ambiente de Desenvolvimento

├── prod.tfvars # Valores específicos para o ambiente de Produção (exemplo)

├── outputs.tf # Saídas globais do projeto (ex: DNS do ALB, IPs)

├── provider.tf # Configuração do provedor AWS

├── database\_app.tf # Configuração do RDS para a aplicação e Secrets Manager

├── load\_balancer.tf # Configuração do Application Load Balancer

├── user\_data\_frontend.sh # Script de inicialização para instâncias frontend

├── user\_data\_backend.sh # Script de inicialização para instâncias backend

└── user\_data\_wazuh\_server.sh # Script de inicialização para o servidor Wazuh

## 3.2. Módulo Raiz

Localizado na pasta INFRA-PROJETO-AWS/, o módulo raiz contém:

main.tf: Define os recursos globais (como a configuração IAM para EC2, o perfil da instância, os data sources para os scripts de user data, e os recursos para o servidor Wazuh) e chama os módulos locais (VPC, Security, EC2).

variables.tf: Declara todas as variáveis de entrada do projeto (ex: região, CIDRs, URLs de repositórios, configurações de banco de dados, tipos de instância).

terraform.tfvars e ficheiros específicos por ambiente (ex: dev.tfvars, prod.tfvars): Fornecem os valores para as variáveis de entrada, permitindo customização por ambiente.

outputs.tf: Define quais informações serão exibidas após a aplicação do Terraform (ex: IP público do frontend, endpoint do RDS, DNS do ALB).

provider.tf: Configura o provedor AWS, especificando a região e o perfil de credenciais.

database\_app.tf: Define os recursos para o banco de dados RDS PostgreSQL da aplicação e o armazenamento seguro das suas credenciais no AWS Secrets Manager.

load\_balancer.tf: Define os recursos para o Application Load Balancer do frontend.

## 3.3. Módulos Locais

Módulo VPC (modules/vpc/): Responsável pela criação da Virtual Private Cloud (VPC), subnets públicas e privadas em múltiplas Zonas de Disponibilidade, Internet Gateway (IGW), NAT Gateway (com Elastic IP) e as tabelas de rotas associadas.

Módulo Security (modules/security/): Gere os Security Groups. Cria grupos para o ALB, frontend, backend, banco de dados RDS e para o servidor Wazuh, com regras específicas para cada um.

Módulo EC2 (modules/ec2/): Define um módulo genérico para criar instâncias EC2, chamado múltiplas vezes para as instâncias do frontend, backend e servidor Wazuh.

# 4. Componentes da Infraestrutura e Decisões de Design

## 4.1. Rede (VPC - Virtual Private Cloud)

Definição: Uma VPC customizada (ex: 10.0.0.0/16) é criada para cada ambiente (dev, prod) para isolar os recursos.

Subnets:

Públicas: Múltiplas subnets públicas distribuídas em diferentes AZs para alta disponibilidade, hospedando o ALB e o servidor Wazuh (com acesso restrito).

Privadas: Múltiplas subnets privadas em AZs distintas para as instâncias EC2 do backend e o banco de dados RDS.

Conectividade: IGW para subnets públicas e NAT Gateway para acesso de saída das subnets privadas. Tabelas de rotas específicas.

## 4.2. Segurança

Security Groups (SGs):

ALB SG: Permite tráfego HTTP/S da internet.

Frontend SG: Permite tráfego na porta da aplicação (ex: 80) somente do ALB SG.

Backend SG: Permite tráfego na porta da API (ex: 8000) somente do Frontend SG.

RDS App SG: Permite tráfego na porta do PostgreSQL (5432) somente do Backend SG.

Wazuh Server SG: Permite tráfego HTTPS (443) para o dashboard de IPs específicos (my\_home\_ip\_cidr) e tráfego dos agentes (1514/1515) dos SGs do frontend e backend.

Acesso SSH: O acesso via porta 22 é desabilitado em favor do AWS Systems Manager Session Manager.

IAM (Identity and Access Management):

Uma IAM Role (ec2\_role) é criada para as instâncias EC2 com políticas para Session Manager, leitura de segredos do Secrets Manager (credenciais do BD, chave SSH do GitHub). Um IAM Instance Profile (ec2\_profile) associa a role às instâncias.

AWS Secrets Manager:

Credenciais do banco de dados RDS da aplicação e a chave SSH privada para deploy do backend são armazenadas de forma segura.

## 4.3. Cômputo (EC2 - Elastic Compute Cloud)

Instância Frontend: Em subnet pública, associada ao frontend\_sg, executa Next.js via user\_data\_frontend.sh. Acessada via ALB.

Instância Backend: Em subnet privada, associada ao backend\_sg, executa API FastAPI via user\_data\_backend.sh com acesso seguro ao GitHub.

Instância Wazuh Server: Em subnet pública (dashboard com SG restrito), executa Wazuh "all-in-one" via user\_data\_wazuh\_server.sh.

## 4.4. Banco de Dados (RDS PostgreSQL)

Uma instância RDS PostgreSQL (ex: db.t3.micro) para a aplicação.

Localizada em subnets privadas, protegida por SG, não publicamente acessível, armazenamento criptografado.

## 4.5. Balanceamento de Carga (Application Load Balancer - ALB)

Um ALB público para o frontend, com listeners HTTP (e opcionalmente HTTPS).

Target Group apontando para as instâncias do frontend, com health checks.

Acesso via nome DNS estável do ALB.

## 4.6. Monitoramento de Segurança e Deteção de Intrusão (Wazuh - IDS/HIDS)

Wazuh Server: Instância EC2 dedicada para a instalação "all-in-one".

Agentes Wazuh: Instalados nas instâncias frontend e backend, reportando ao servidor Wazuh.

Funcionalidade: Coleta de logs, deteção de anomalias, verificação de integridade, etc.

# 5. Scripts de User Data

Scripts de inicialização (user\_data) automatizam a configuração das instâncias EC2:

user\_data\_frontend.sh: Instala Node.js, Git, Nginx. Clona o repositório frontend, instala dependências, builda o Next.js, configura Nginx como proxy reverso, inicia a aplicação com pm2, e instala/configura o agente Wazuh.

user\_data\_backend.sh: Instala Python, Git, etc. Configura chave SSH para clonar o repositório backend privado (branch aws-deploy). Cria ambiente virtual, instala dependências. Busca credenciais do BD do Secrets Manager. Executa migrações Alembic. Inicia a API com systemd e Gunicorn. Instala e configura o agente Wazuh.

user\_data\_wazuh\_server.sh: Baixa e executa o script de instalação "all-in-one" do Wazuh.

# 6. Versionamento e Modularização

Modularização: Código Terraform dividido em módulos locais (VPC, Security, EC2) para organização e reutilização.

Versionamento: Projeto gerido usando Git. Branch aws-deploy na API para configurações AWS. Terraform especifica versões de provedores.

# 7. CI/CD (Integração e Entrega/Deploy Contínuos) com GitHub Actions

Para automatizar o ciclo de vida da infraestrutura, um pipeline de CI/CD é configurado utilizando GitHub Actions.

## 7.1. Workflow

Trigger: Em push para branches principais (ex: main para dev, production para prod) ou em Pull Requests.

Jobs:

Validate & Plan (CI):

terraform init

terraform fmt -check

terraform validate

tflint (linting adicional)

tfsec / checkov (análise de segurança)

terraform plan -var-file="<ambiente>.tfvars" -out=tfplan (plano específico do ambiente)

O plano é armazenado como artefacto e/ou postado em PRs.

Apply (CD):

Disparado manualmente (para prod) ou automaticamente (para dev) após CI bem-sucedido.

terraform apply tfplan

## 7.2. Configuração

Ficheiro YAML em .github/workflows/.

Uso de GitHub Secrets para credenciais AWS.

# 8. Gerenciamento de Múltiplos Ambientes (dev, prod)

Para gerir ambientes distintos como desenvolvimento (dev) e produção (prod) com o mesmo código base Terraform, a estratégia adotada é o uso de Terraform Workspaces em conjunto com ficheiros de variáveis específicos por ambiente.

## 8.1. Terraform Workspaces

Criação:

terraform workspace new dev

terraform workspace new prod

# ... (para outros ambientes como staging)

Seleção:

terraform workspace select dev

Isolamento de Estado: Cada workspace mantém um ficheiro de estado (terraform.tfstate) separado, geralmente num backend remoto como S3, garantindo que as operações num ambiente não afetem os outros.

## 8.2. Ficheiros de Variáveis por Ambiente

Ficheiros como dev.tfvars e prod.tfvars são criados na raiz do projeto.

Estes ficheiros contêm os valores das variáveis que diferem entre os ambientes (ex: tipos de instância, contagem de instâncias, nomes de recursos prefixados pela variável environment, configurações de banco de dados, etc.).

Exemplo (prod.tfvars):

environment = "prod"

instance\_type\_frontend = "t3.small" # Tipos de instância mais robustos para produção

instance\_type\_backend = "t3.small"

wazuh\_server\_instance\_type = "t3.large" # Servidor Wazuh pode precisar de mais recursos em prod

db\_instance\_class\_postgres = "db.t3.small" # Banco de dados mais robusto para produção

# Para produção, considere desabilitar skip\_final\_snapshot e configurar um período de retenção de backup para o RDS

# rds\_skip\_final\_snapshot = false

# rds\_backup\_retention\_period = 7

enable\_deletion\_protection\_alb = true # Habilitar proteção contra exclusão para o ALB em produção

# Outras configurações de produção (ex: mais réplicas para EC2 via Auto Scaling Groups,

# configurações de logging mais detalhadas, políticas de backup mais rigorosas, etc.)

Execução:

terraform workspace select prod

terraform plan -var-file="prod.tfvars"

# Após revisão cuidadosa do plano:

terraform apply -var-file="prod.tfvars"

A variável environment (definida em cada <ambiente>.tfvars) é usada para nomear e marcar recursos, permitindo fácil identificação e prevenindo conflitos de nome entre os diferentes ambientes.

Esta abordagem permite reutilizar a base de código modularizada para provisionar e gerir múltiplos ambientes de forma consistente e isolada, ajustando apenas os parâmetros necessários para cada estágio do ciclo de vida da aplicação.

# 9. Conclusão e Próximos Passos Pós-Deploy

Após a execução bem-sucedida do terraform apply para um ambiente específico (como dev ou prod):

Verificar a funcionalidade completa do frontend (acessado via DNS do ALB) e do backend.

Confirmar conectividade com o banco de dados RDS e a correta execução das migrações da aplicação.

Acessar o dashboard do Wazuh, verificar se o servidor está operacional e se todos os agentes (frontend, backend, e no próprio servidor Wazuh) estão reportando corretamente.

Validar o pipeline de CI/CD, garantindo que as automações de plan e apply funcionam como esperado para os diferentes ambientes.

Realizar testes de segurança (incluindo a verificação dos alertas e configurações do Wazuh) e monitoramento de performance para validar a eficácia das ferramentas implementadas.

Continuar a otimização dos custos, revisando os tipos de instância e o uso de recursos.

Para o ambiente de produção, planear e implementar estratégias de backup mais robustas para os bancos de dados e volumes EBS, bem como planos de recuperação de desastres.

Considerar a implementação de monitoramento mais avançado com CloudWatch Alarms e dashboards customizados.

Esta infraestrutura estabelece uma base sólida, segura, monitorada e automatizada para a aplicação web, utilizando práticas modernas de Infraestrutura como Código (IaC) e DevOps. Está preparada para evoluir com as necessidades do projeto, suportando múltiplos ambientes e facilitando a entrega contínua de valor.