

Plataforma de Serviços de Rede Baseada em containers

Daniel Taveira - a44319

Trabalho realizado sob a orientação de
Prof. Nuno Rodrigues
Prof. Eduardo Costa

Licenciatura em Engenharia Informática
2025-2026

Plataforma de Serviços de Rede Baseada em containers

Relatório da UC de Projeto
Licenciatura em Engenharia Informática
Escola Superior de Tecnologia e Gestão

Daniel Taveira - a44319

2025-2026

A Escola Superior de Tecnologia e de Gestão não se responsabiliza pelas opiniões expressas neste relatório.

Declaro que o trabalho descrito neste relatório é da minha autoria e é
da minha vontade que o mesmo seja submetido a avaliação.

Daniel Taveira - a44319

Dedicatória

(Facultativo) Dedico este trabalho a ...

Agradecimentos

(Facultativo) Agradeço a ...

Resumo

O presente projeto tem como objetivo principal a conceção e implementação de uma plataforma de serviços de rede gerida de forma reprodutível, recorrendo exclusivamente a ferramentas *open source*. A infraestrutura desenvolvida engloba serviços fundamentais para uma rede de pequena a média dimensão, tais como servidor DNS, *Reverse Proxy* HTTPS, serviços *web*, e-mail, servidor de ficheiros e VPN.

Em contraste com abordagens monolíticas tradicionais, todos os serviços são implementados em *containers* (Docker/Podman) através de configuração declarativa, garantindo o versionamento e a reprodutibilidade do ambiente. O trabalho foca-se na aplicação rigorosa de boas práticas de segurança, incluindo o princípio do mínimo privilégio, *hardening* de serviços e encriptação TLS. Adicionalmente, o projeto integra mecanismos avançados de observabilidade (para métricas e *logs* centralizados) e a orquestração de *deployments* através de ferramentas como Ansible e *pipelines* de CI/CD. O resultado final demonstra como um pequeno *data center* pode ser gerido de forma automatizada, ágil e resiliente a falhas.

Palavras-chave: *Containers*, Orquestração, Serviços de Rede, CI/CD, Observabilidade, *Open Source*.

Abstract

The main objective of this project is the design and implementation of a network services platform managed in a reproducible way, using exclusively open-source tools. The developed infrastructure encompasses fundamental services for a small to medium-sized network, such as a DNS server, HTTPS Reverse Proxy, web services, e-mail, file server, and a VPN.

In contrast to traditional monolithic approaches, all services are implemented in containers (Docker/Podman) through declarative configuration, ensuring environment versioning and reproducibility. The work focuses on the strict application of security best practices, including the principle of least privilege, service hardening, and TLS encryption. Additionally, the project integrates advanced observability mechanisms (for metrics and centralized logs) and deployment orchestration using tools like Ansible and CI/CD pipelines. The final result demonstrates how a small data center can be managed in an automated, agile, and fault-resilient manner.

Keywords: Containers, Orchestration, Network Services, CI/CD, Observability, Open Source.

Índice

1	Introdução	1
1.1	Contextualização e Motivação	1
1.2	Objetivos	1
1.3	Estrutura do Documento	2
2	Estado da Arte	3
2.1	Containerização e Orquestração Leve	3
2.2	Observabilidade Centralizada	4
2.3	Segurança e CI/CD em Ambientes Locais	4
3	Levantamento das Necessidades Funcionais	5
3.1	Serviços de Rede a Disponibilizar	5
3.2	Requisitos Técnicos e de Arquitetura	6
3.3	Ambiente Base: Virtualização e Sistema Operativo	7
4	Arquitetura e Implementação	9
4.1	Configuração do Ambiente Base	9
4.2	Automação da Infraestrutura com Ansible	10
4.3	Implementação dos Serviços Base	12
5	Testes/Avaliação/Discussão	15
6	Conclusões	17

A Proposta Original do Projeto **A1**

B Outro(s) Apêndice(s) **B1**

Lista de Tabelas

Lista de Figuras

4.1	Verificação do endereço IP e acesso SSH ao Ubuntu Server.	10
4.2	Estrutura do repositório e ficheiro de inventário do Ansible.	11
4.3	Execução bem-sucedida do <i>Playbook</i> de instalação do Docker.	12
4.4	Interface de administração do servidor DNS (Pi-hole) a correr em <i>container</i> . .	13

Capítulo 1

Introdução

A administração de sistemas e redes tem sofrido uma evolução drástica nos últimos anos. A necessidade de implementar e gerir serviços de forma rápida, segura e sem interrupções impulsionou a adoção de tecnologias de *containerização* e práticas de automação de infraestruturas.

1.1 Contextualização e Motivação

Numa infraestrutura tecnológica de pequena ou média dimensão, a gestão manual de múltiplos serviços (como DNS, *web*, *proxies* e VPNs) torna-se insustentável a longo prazo, sendo propensa a erros humanos e dificultando a recuperação em caso de falhas. A adoção do paradigma de "Infraestrutura como Código" (IaC) e de arquiteturas baseadas em *containers* permite mitigar estes problemas, oferecendo ambientes isolados, padronizados e facilmente replicáveis.

1.2 Objetivos

O objetivo central deste projeto é construir uma plataforma de serviços de rede baseada em *containers* e orquestração leve, focada em boas práticas de segurança, CI/CD (*Continuous Integration / Continuous Deployment*) e observabilidade.

De forma mais específica, pretende-se:

- Implementar serviços típicos de rede (DNS, *Reverse Proxy* HTTPS, Web, E-mail, Servidor de Ficheiros e VPN) em *containers* (Docker/Podman).
- Demonstrar a gestão de um pequeno *data center* de forma totalmente reproduzível utilizando apenas ferramentas *open source*.
- Integrar mecanismos de observabilidade para a recolha de métricas e *logs* centralizados.
- Automatizar a configuração e o *deploy* com recurso a Ansible e a uma *pipeline* simples de CI/CD.

1.3 Estrutura do Documento

O presente relatório encontra-se organizado da seguinte forma: o Capítulo 2 apresenta o Estado da Arte das tecnologias envolvidas; o Capítulo 3 detalha o levantamento das necessidades funcionais e a escolha do ambiente base; o Capítulo 4 descreve a arquitetura e implementação da plataforma; o Capítulo 5 aborda os testes e resultados obtidos; por fim, o Capítulo 6 sintetiza as conclusões do trabalho.

Capítulo 2

Estado da Arte

A transição de infraestruturas baseadas em máquinas virtuais monolíticas para arquiteturas orientadas a *containers* revolucionou a administração de sistemas. Este capítulo explora as tecnologias e os paradigmas atuais que suportam a criação de plataformas de serviços de rede ágeis, seguras e observáveis.

2.1 Containerização e Orquestração Leve

Tradicionalmente, a segregação de serviços exigia a alocação de máquinas virtuais independentes, o que resultava num elevado *overhead* de recursos e numa gestão complexa. Atualmente, motores de *containerização* como o Docker e o Podman tornaram-se o padrão da indústria. Estas ferramentas permitem encapsular uma aplicação e todas as suas dependências num ambiente isolado, que partilha o *kernel* do sistema operativo *host*.

O Podman, em particular, destaca-se pela sua arquitetura *daemonless* e pela capacidade de executar *containers* sem privilégios de *root*, indo ao encontro das melhores práticas de segurança (o princípio do mínimo privilégio). Para a orquestração de infraestruturas de pequena a média dimensão, ferramentas declarativas como o Docker Compose oferecem a "orquestração leve"ideal. Permitem definir infraestruturas complexas como código (IaC - *Infrastructure as Code*) de forma totalmente reprodutível e facilmente versionável.

2.2 Observabilidade Centralizada

Manter um conjunto de serviços distribuídos exige mais do que uma simples monitorização; exige observabilidade. O estado da arte baseia-se em *stacks open source* complementares:

- **Métricas:** O Prometheus atua como o motor principal de recolha de métricas em formato *time-series*, trabalhando em conjunto com o Grafana para a visualização avançada de dados através de *dashboards* dinâmicos.
- **Logs:** Soluções como a *stack* ELK (Elasticsearch, Logstash, Kibana) ou o Loki (otimizado pela Grafana Labs para ambientes de *containers*) permitem agregar *logs* de múltiplos serviços num único ponto central. Isto facilita substancialmente o processo de *troubleshooting*.

2.3 Segurança e CI/CD em Ambientes Locais

A segurança *by design* é um requisito fundamental na gestão de redes modernas. Isto inclui o uso sistemático de *Reverse Proxies* (como o Nginx, Traefik ou Caddy), que gerem a terminação TLS/SSL de forma automática, e o *hardening* de imagens base.

A adoção de *pipelines* de CI/CD (Integração e Entrega Contínuas) deixou de ser um processo exclusivo do desenvolvimento de *software* e passou a integrar a gestão de infraestruturas (GitOps). A utilização de repositórios Git acoplados a *runners* locais permite que qualquer alteração na infraestrutura (como uma atualização de versão do servidor DNS ou a adição de um novo serviço Web) seja testada antes do *deploy* para o ambiente de produção. Esta prática reduz o erro humano e garante um *rollback* fiável em caso de falha dos serviços.

Capítulo 3

Levantamento das Necessidades Funcionais

Com base no guião do projeto proposto, o objetivo central consiste na criação de um "data center" de serviços de rede que possa ser gerido de forma reproduzível, recorrendo exclusivamente a ferramentas *open source*. Para atingir este objetivo, as necessidades do projeto dividem-se em duas categorias principais: os serviços a disponibilizar (requisitos funcionais) e os requisitos técnicos e de arquitetura (requisitos não-funcionais).

3.1 Serviços de Rede a Disponibilizar

A infraestrutura deve contemplar um conjunto de serviços típicos de uma rede de pequena/média dimensão. Os serviços a implementar são os seguintes:

- **Servidor DNS:** Implementação de um servidor com capacidade recursiva e autoritativa para a resolução de nomes na rede interna.
- **Reverse Proxy HTTPS:** Criação de um ponto de entrada único e seguro para o encaminhamento de tráfego para os diversos serviços internos.
- **Serviços Web e de E-mail:** Alojamento de páginas *web* e disponibilização de um sistema de gestão básica de correio eletrónico.

- **Servidor de Ficheiros:** Sistema centralizado para o armazenamento e a partilha de ficheiros na rede.
- **VPN (*Virtual Private Network*):** Implementação de um serviço que permita o acesso remoto e seguro à infraestrutura interna de *containers*.

3.2 Requisitos Técnicos e de Arquitetura

Para além do correto funcionamento dos serviços, a plataforma tem de respeitar rigorosos critérios de implementação e gestão:

- **Containerização e Configuração Declarativa:** Todos os serviços têm de ser executados em *containers* (Docker ou Podman). A configuração deve ser feita de forma declarativa (utilizando ficheiros como o `docker-compose.yml`), o que garante a sua reproduzibilidade e permite o versionamento.
- **Observabilidade e Monitorização:** É obrigatória a integração de uma *stack open source* (por exemplo, Prometheus, Grafana, Loki ou ELK/EFK) para a recolha contínua de métricas, gestão de *logs* centralizados e configuração de alertas básicos (como latência, erros 5xx ou saturação de CPU/RAM).
- **Segurança (*Hardening*):** O projeto exige a aplicação do princípio do mínimo privilégio nos *containers*, gestão segura e isolada de segredos, encriptação obrigatória de tráfego (TLS) e uma política de atualização automatizada das imagens dos *containers*.
- **Pipeline de CI/CD:** Deve ser implementada uma *pipeline* simples (utilizando, por exemplo, o Git em conjunto com um *runner* local) para testar automaticamente as alterações de configuração e assegurar um *deploy* controlado para o ambiente de produção do laboratório.

3.3 Ambiente Base: Virtualização e Sistema Operativo

Para suportar a plataforma e garantir o isolamento do *host* físico, a infraestrutura base será implementada em ambiente virtualizado.

- **Virtualização (VMware):** A plataforma escolhida para o *hypervisor* local é o VMware (Workstation/Player). Esta escolha justifica-se pela sua estabilidade, excelente gestão de redes virtuais (Bridged/NAT) necessárias para simular o tráfego externo/interno e eficiência na alocação de recursos.
- **Sistema Operativo (*Guest OS*):** Para o sistema operativo que irá alojar os *containers*, optou-se por uma distribuição Linux orientada a servidores, nomeadamente o **Ubuntu Server 24.04 LTS** (ou alternativamente o **Debian 12**). Estas distribuições foram selecionadas por possuírem vasto suporte da comunidade, excelente compatibilidade nativa com o Docker/Podman e com o Ansible, e por não incluírem interfaces gráficas (*headless*), o que minimiza a superfície de ataque e o consumo de recursos.

Capítulo 4

Arquitetura e Implementação

Este capítulo detalha o processo prático de implementação da plataforma de serviços de rede. A abordagem adotada seguiu as melhores práticas de "Infraestrutura como Código" (IaC), garantindo que todo o ambiente pudesse ser reconstruído de forma automatizada e previsível.

4.1 Configuração do Ambiente Base

O primeiro passo consistiu na preparação do *host* físico que iria albergar os *containers*. Para simular um ambiente de *data center* isolado, mas acessível para testes, recorreu-se ao *hypervisor* VMware para criar uma máquina virtual (VM) a correr o Ubuntu Server 24.04 LTS.

A interface de rede da VM foi configurada em modo *Bridged*, permitindo-lhe obter um endereço IP dedicado na rede local. A Figura 4.1 demonstra o acesso inicial ao servidor por SSH e a verificação da sua configuração de rede.

O acesso remoto por SSH é crucial, uma vez que permite operar o servidor de forma *headless* (sem interface gráfica), otimizando o consumo de recursos e preparando o terreno para a automação remota.

```

root@ubuntu-server:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:b4:12:59 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.1.252/24 metric 100 brd 192.168.1.255 scope global dynamic ens33
        valid_lft 3537sec preferred_lft 3537sec
    inet6 2001:8a0:fc04:9200:20c:29ff:feb4:1259/64 scope global dynamic mngtmpaddr noprefixroute
        valid_lft 89938sec preferred_lft 89938sec
    inet6 fe80::20c:29ff:feb4:1259/64 scope link
        valid_lft forever preferred_lft forever
root@ubuntu-server:~#

```

Figura 4.1: Verificação do endereço IP e acesso SSH ao Ubuntu Server.

4.2 Automação da Infraestrutura com Ansible

Para cumprir o requisito de reprodutibilidade do projeto, optou-se por não realizar instalações manuais no servidor. Em vez disso, introduziu-se o **Ansible** como ferramenta de gestão de configuração. O Ansible liga-se ao servidor por SSH e executa instruções definidas em ficheiros YAML (*Playbooks*), garantindo que o estado final do servidor é sempre o esperado.

O ponto de partida da automação é o ficheiro de inventário (`inventory.ini`), onde se define o endereço IP e o utilizador de acesso ao servidor de destino, conforme ilustrado na Figura 4.2.

Nota sobre o Ambiente de Execução (Control Node)

É importante salientar uma particularidade da arquitetura de *deploy*. Uma vez que a máquina física do administrador opera com o sistema operativo Windows, que não suporta nativamente a execução do Ansible, adotou-se o **WSL (Windows Subsystem for Linux)** como *Control Node*.

O WSL fornece um ambiente Linux completo e integrado, permitindo a instalação do Ansible e a execução transparente dos *Playbooks*. Através desta abordagem, o terminal WSL comunica de forma segura via protocolo SSH com a máquina virtual VMware (*Managed Node*), aplicando as configurações sem necessitar de ferramentas adicionais ou de dual-boot no sistema hospedeiro.

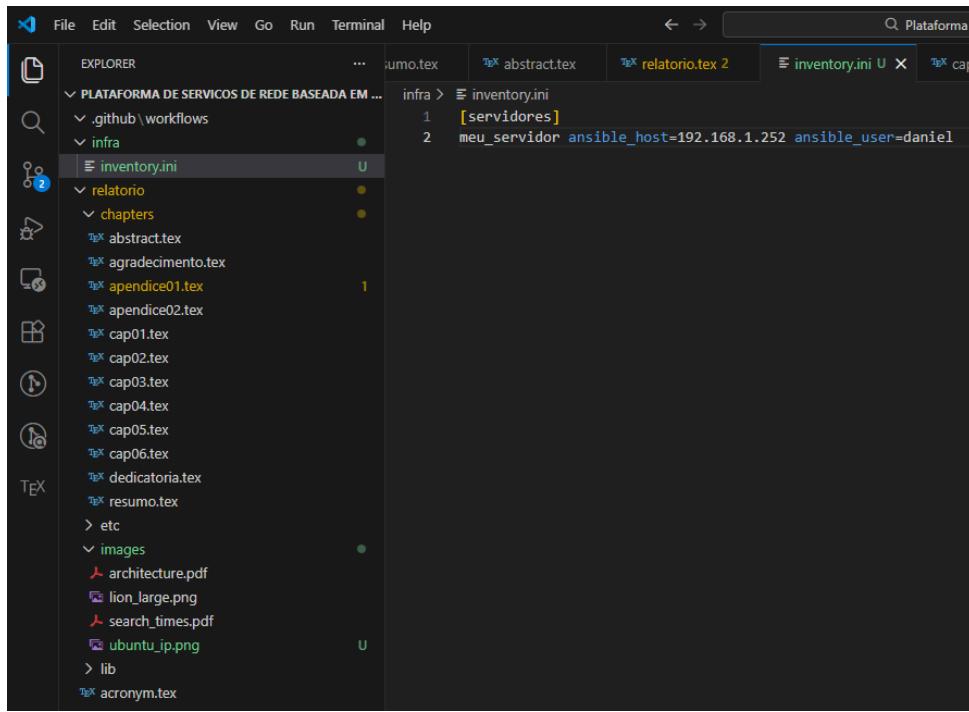


Figura 4.2: Estrutura do repositório e ficheiro de inventário do Ansible.

Com o inventário configurado, o passo seguinte consistiu na criação do *Playbook* responsável por instalar as dependências base do sistema, nomeadamente o motor de *containers* (Docker), que servirá de alicerce para todos os serviços subsequentes.

Para materializar a instalação do motor de *containers*, desenvolveu-se o *Playbook* `setup_docker.yml`. Este *script* declarativo é responsável por executar as seguintes tarefas no servidor de destino:

1. Atualização da *cache* dos repositórios APT.
2. Instalação de pacotes utilitários e de certificados de segurança.
3. Instalação do motor principal (`docker.io`) e do *plugin docker-compose-v2*, fundamentais para a orquestração dos serviços da plataforma.
4. Ativação do serviço *daemon* do Docker no `systemd`, garantindo a sua inicialização automática em caso de reinício do servidor.

- Adição do utilizador remoto ao grupo `docker`, implementando o princípio de menor privilégio ao evitar o uso constante de credenciais `root` para a gestão de *containers*.

A execução deste *Playbook* a partir da máquina local do administrador garante que o servidor *host* fica imediatamente pronto a receber os serviços da plataforma. A Figura 4.3 ilustra o resultado (em jargão técnico, o *Play Recap*) de uma execução bem-sucedida do Ansible contra a máquina virtual, demonstrando a idempotência da ferramenta (aplicação das configurações sem causar erros em execuções repetidas).

```
daniel@Dani-Fixo:/mnt/c/Users/Dani/Desktop/Plataforma de servicos de rede baseada em containers/infra$ ansible-playbook -i inventory.ini setup_docker.yml -K
BECOME password:

PLAY [Configuração inicial do servidor e instalação do Docker] *****

TASK [Gathering Facts] *****
ok: [meu_servidor]

TASK [Atualizar a cache do apt (repositórios)] *****
changed: [meu_servidor]

TASK [Instalar pacotes pré-requisitos do sistema] *****
changed: [meu_servidor]

TASK [Instalar o Docker e o Docker Compose] *****
changed: [meu_servidor]

TASK [Garantir que o serviço do Docker arranca automaticamente com o sistema] *****
ok: [meu_servidor]

TASK [Adicionar o utilizador atual ao grupo docker (para não exigir sudo)] *****
changed: [meu_servidor]

PLAY RECAP *****
meu_servidor      : ok=6   changed=4   unreachable=0   failed=0   skipped=0   rescued=0   ignored=0
```

Figura 4.3: Execução bem-sucedida do *Playbook* de instalação do Docker.

4.3 Implementação dos Serviços Base

Garantida a configuração do *host*, avançou-se para o *deploy* dos primeiros serviços da plataforma. O guião estipula o uso de definição declarativa, pelo que se adotou o Docker Compose para orquestrar os *containers*.

O primeiro bloco de infraestrutura foca-se em dois serviços essenciais:

- Reverse Proxy (Nginx):** Atua como o ponto de entrada único da rede, expondo as portas 80 e 443. Futuramente, será responsável por encaminhar os pedidos para os serviços corretos e gerir os certificados TLS.

- **Servidor DNS (Pi-hole):** Implementado para atuar como servidor autoritativo e recursivo da rede interna, permitindo resolução de nomes personalizada e bloqueio de domínios indesejados a nível de rede.

Para manter a filosofia de Infraestrutura como Código, os ficheiros `docker-compose.yml` não são criados manualmente no servidor. Em vez disso, foi desenvolvido um novo *Playbook* de Ansible (`deploy_servicos.yml`) responsável por transferir a pasta de serviços do *Control Node* para o servidor e executar o comando de inicialização (`docker compose up -d`). A Figura 4.4 comprova a execução bem-sucedida do processo, demonstrando o acesso ao painel de administração do servidor DNS acabado de instanciar.

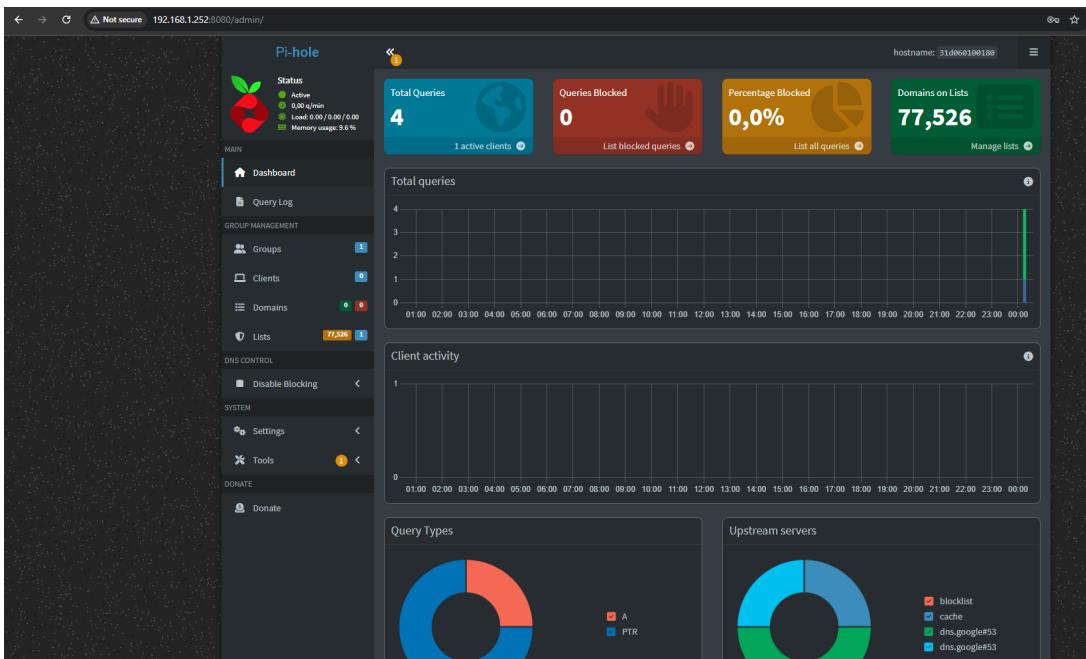


Figura 4.4: Interface de administração do servidor DNS (Pi-hole) a correr em *container*.

Procedimentos de *Troubleshooting* Ativos

Durante a fase de implementação e orquestração do serviço DNS (Pi-hole) via Docker Compose, identificou-se uma limitação comportamental no *container*: a variável de ambiente `WEBPASSWORD` definida declarativamente pode não ser aplicada de imediato se o volume do *container* já se encontrar persistido com configurações vazias ou padrão.

Para ultrapassar esta limitação sem comprometer o estado do *host* físico, recorreu-se a técnicas de *troubleshooting* avançado. Utilizando o comando **docker exec**, foi possível interagir diretamente com a *shell* interna do *container* em tempo de execução e aplicar a correção (neste caso, a definição da palavra-passe de acesso) sem necessidade de reconfigurar todo o serviço:

```
# Comando executado via SSH no servidor host para injecao no
# container
sudo docker exec dns_pihole pihole setpassword 'nova_password'
```

Este tipo de intervenção pontual demonstra a flexibilidade das arquiteturas baseadas em *containers*: os problemas podem ser isolados e resolvidos diretamente dentro do ecossistema do serviço afetado, mantendo a integridade do servidor *host* e dos restantes serviços da plataforma.

Capítulo 5

Testes/Avaliação/Discussão

Este capítulo apresenta os testes realizados para verificar que o projeto desenvolvido cumpre os objetivos assumidos e resolve, de facto, o problema descrito na Análise/Modelação.

Para uma melhor compreensão, os resultados de cada teste devem ser precedidos de uma descrição, mesmo que resumida, do teste realizado e dos resultados esperados.

Os resultados do trabalho são comentados, acrescentando-lhe valor:

- O que é que se pode inferir ou conjecturar dos resultados obtidos?
- O que poderia/deveria ter sido feito de forma diferente?
- Onde se foi além dos objetivos iniciais?
- Quais os objetivos que ficaram por cumprir, e porquê ?

Capítulo 6

Conclusões

As conclusões devem sintetizar e proporcionar uma perspetiva unificadora ao trabalho efetuado. Poderá ser feita uma breve referência a trabalhos de outros com semelhanças ao efetuado e ao conhecimento que resultou do trabalho efetuado, bem como sugestões de trabalho futuro. A coerência do documento implica que as conclusões devem ser coerentes com as ideias expostas na introdução.

Apêndice A

Proposta Original do Projeto



Curso de Licenciatura em Engenharia Informática
Projeto 3º Ano - Ano letivo de 2016/2017

<Título do projeto>

Orientador: <Nome do orientador>

Coorientador: <Nome do coorientador>

1 Objetivo

<Objetivo do projeto>

2 Detalhes

<Detalhes que julguem ser necessários>

3 Metodologia de trabalho

<Eventual metodologia de trabalho>

Dimensão da equipa:

Recursos necessários:

Apêndice B

Outro(s) Apêndice(s)

Listagens de código fonte, texto/imagens produzidos por testes complementares, etc.