



# Instituto Superior de Engenharia

Politécnico de Coimbra

DEPARTAMENTO DE INFORMÁTICA E SISTEMAS

## Relatório de estágio

Relatório de Estágio para a obtenção do grau de licenciado em  
Informática

Autor

**Ricardo Oliveira Ribeiro**

Orientador

**Francisco José Simões Duarte**

Supervisores na empresa OneSource

**Uriel Martins**

**Ricardo Martins**

Coimbra janeiro, 2024



INSTITUTO POLITÉCNICO  
DE COIMBRA

INSTITUTO SUPERIOR  
DE ENGENHARIA  
DE COIMBRA

## **RESUMO**

Cada vez mais a relevância dos ambientes de orquestração de containers e das aplicações virtualizadas é inquestionável. Isso deve-se a uma variedade de fatores no contexto tecnológico atual. A agilidade na implementação de serviços é primordial diante ao aumento exponencial da demanda dos clientes e da necessidade incessantes de introduzir novas funcionalidades para melhorar a experiência do utilizador e manter a sua fidelidade. Além disso, a portabilidade dos serviços é facilitada pela virtualização de recursos, permitindo que as aplicações operem de forma consistente em diferentes ambientes, seja localmente ou na nuvem. A tolerância a falhas também desempenha um papel crucial, especialmente considerando que cada vez mais serviços críticos são hospedados online, exigindo uma infraestrutura robusta e resiliente para garantir a disponibilidade contínua. Esses elementos convergem para impulsionar a adoção e a evolução dos containers como uma tecnologia fundamental para a modernização e aprimoramento das infraestruturas de IT em todo o mundo.

## **ABSTRACT**

The increasing relevance of container orchestration environments and virtualized applications is undeniable. This is due to a variety of factors in the current technological landscape. Agility in service implementation is paramount given the exponential increase in customer demand and the constant need to introduce new features to enhance user experience and maintain their loyalty. Additionally, service portability is facilitated by resource virtualization, enabling applications to operate consistently across different environments, whether locally or in the cloud. Fault tolerance also plays a crucial role, especially considering that an increasing number of critical services are hosted online, demanding a robust and resilient infrastructure to ensure continuous availability. These elements converge to drive the adoption and evolution of containers as a fundamental technology for the modernization and enhancement of IT infrastructures worldwide.

## **EPÍGRAFE**

Fear is the enemy of progress.

## **AGRADECIMENTOS**

Acima de todos dedico este trabalho aos meus pais, cuja compreensão e apoio tanto emocional como financeiro foram a razão de cada conquista alcançada por mim não só nesta etapa mas como durante toda a minha vida. À minha Maria, por me suportar nos momentos mais desafiantes deste projeto onde o meu humor não era a minha melhor faceta. Ao ISEC por me proporcionar um desenvolvimento cognitivo superior e oferecer um ambiente de crescimento social de forma inconsciente. E à OneSource pela oportunidade oferecida de provar o meu valor.

## ÍNDICE

Resumo . . . . .	i
Abstract . . . . .	ii
Epígrafe . . . . .	iii
Dedicatória . . . . .	iv
Agradecimentos . . . . .	iv
Índice . . . . .	1
Índice de tabelas . . . . .	5
Índice de figuras . . . . .	6
Índice de código . . . . .	7
Lista de abreviaturas . . . . .	7
Lista de siglas e acrónimos . . . . .	8
Lista de símbolos . . . . .	9
1 Introdução . . . . .	9
1.1 Enquadramento . . . . .	9
1.2 Entidade de acolhimento . . . . .	9
1.3 Objetivos . . . . .	9
1.4 Plano e metodologia de trabalho . . . . .	10
2 Virtualização . . . . .	11
2.1 Tipos de virtualização . . . . .	11
2.2 Containers e a sua Orquestração . . . . .	12
2.2.1 Contêiners VS Virtualização . . . . .	12
2.3 Docker e Kubernetes . . . . .	13
2.4 Vantagens da orquestração de containers . . . . .	14

2.5	Ferramentas de virtualização . . . . .	14
3	Serviços de rede . . . . .	16
3.1	DHCP . . . . .	16
3.1.1	Introdução . . . . .	16
3.1.2	Componentes . . . . .	16
3.1.3	Processo . . . . .	16
3.1.4	Tempo de vida . . . . .	16
3.1.5	Segurança . . . . .	17
3.1.6	DHCP no meu projeto . . . . .	17
3.2	DNS . . . . .	17
3.2.1	Introdução . . . . .	17
3.2.2	Hierarquia . . . . .	17
3.2.3	Processo . . . . .	18
3.2.4	Registos . . . . .	18
3.2.5	Segurança . . . . .	19
3.2.6	DNSSEC . . . . .	19
3.2.7	DNS no meu projeto . . . . .	19
3.3	VPN . . . . .	19
3.3.1	Introdução . . . . .	19
3.3.2	Funcionamento . . . . .	20
3.3.3	Tipos . . . . .	20
3.3.4	Benefícios . . . . .	21
3.3.5	VPN no meu projeto . . . . .	21
3.4	Serviços de diretorias . . . . .	21
3.4.1	Introdução . . . . .	21
3.4.2	Conceitos . . . . .	21
3.4.3	Funcionalidades . . . . .	21
3.4.4	Tipos . . . . .	22
3.5	SSL . . . . .	22
3.5.1	Introdução . . . . .	22
3.5.2	Funcionamento . . . . .	22
3.5.3	Aplicações . . . . .	22
3.5.4	SSL no meu projeto . . . . .	23
4	Container runtime . . . . .	24
4.1	Configuration File . . . . .	24
4.2	Imagens . . . . .	24
4.3	Componentes . . . . .	25
4.3.1	Deployment . . . . .	25
4.3.2	Namespace . . . . .	26

## Relatório de Estágio

4.3.3	Pod . . . . .	26
4.3.4	Secret . . . . .	27
4.3.5	Role-Based Access Control . . . . .	28
4.3.6	ServiceAccount . . . . .	29
4.3.7	ReplicaSet . . . . .	29
4.3.8	Service . . . . .	30
4.3.9	Ingress . . . . .	31
4.3.10	Volumes . . . . .	32
4.3.11	Persistent Volume Claim . . . . .	32
4.3.12	Storage Class . . . . .	33
4.4	StatefulSet . . . . .	34
5	Sistema de Orquestração de Containers . . . . .	36
5.1	Sistema Distribuído (Cluster) . . . . .	36
5.2	Kubernetes . . . . .	36
5.2.1	Arquitetura do Kubernetes . . . . .	36
5.2.2	Kubernetes Control Plane . . . . .	37
5.3	Gestão de Container . . . . .	38
5.3.1	Fases dos Pods e Suas Transições . . . . .	38
5.4	Kubernetes e o Container Runtime . . . . .	38
6	Projeto . . . . .	39
6.1	Descrição . . . . .	39
6.2	Pré-requisitos . . . . .	39
6.3	Prvisionamento . . . . .	39
6.4	Instalação das ferramentas cliente . . . . .	40
6.5	Comunicação no cluster com TLS . . . . .	40
6.6	Ficheiros de configuração Kube . . . . .	41
6.7	Encriptação de Dados . . . . .	41
6.8	Os binaries mais recentes do kubernetes . . . . .	41
6.9	Componentes do Master . . . . .	42
6.10	Controll plane . . . . .	42
6.11	Load-Balancer . . . . .	42
6.12	Worker nodes . . . . .	43
6.12.1	Worker 1: . . . . .	43
6.12.2	Worker 2: . . . . .	44
6.13	Kubectl . . . . .	44
6.14	Pod Networking . . . . .	44
6.15	RBAC permissions . . . . .	44
6.16	DNS . . . . .	45
6.17	Smoke Test . . . . .	45



7	Testes do projeto . . . . .	46
7.1	MariaDB . . . . .	46
7.1.1	O que é? . . . . .	46
7.1.2	Como o implementei no meu cluster? . . . . .	46
7.1.3	Dificuldades . . . . .	47
7.2	REDIS . . . . .	48
7.2.1	O que é? . . . . .	48
7.2.2	Como o implementei no meu cluster? . . . . .	48
7.3	MinIO . . . . .	48
7.3.1	O que é? . . . . .	48
7.3.2	Como o implementei no meu cluster? . . . . .	49
7.4	MariaDB VS MinIO VS REDIS . . . . .	49
7.4.1	Comparação Geral . . . . .	50
7.5	Wordpress . . . . .	50
7.5.1	O que é? . . . . .	50
7.5.2	Como o implementei no meu cluster? . . . . .	50
7.5.3	Teste . . . . .	51
7.6	ModSecurity . . . . .	52
7.6.1	O que é? . . . . .	52
7.6.2	Instalação Direta no Nó . . . . .	52
7.7	Gitlab . . . . .	53
7.7.1	O que é? . . . . .	53
7.7.2	Implementação no Cluster Kubernetes . . . . .	54
7.8	Nextcloud . . . . .	55
7.8.1	O que é? . . . . .	55
7.8.2	Como o implementei no meu cluster? . . . . .	55
7.9	Prometheus & Grafana . . . . .	56
7.9.1	O que são? . . . . .	56
7.9.2	Como implementei no meu cluster? . . . . .	56
8	Conclusão . . . . .	58
	Referências bibliográficas . . . . .	59
	Anexos . . . . .	60
	Anexo A - mariaDB yaml files . . . . .	61
	Anexo B - REDIS yaml files . . . . .	62
	Anexo C - minIO yaml files . . . . .	63
	Anexo D - gitlab yaml files . . . . .	64
	Anexo E - nextcloud yaml files . . . . .	65
	Anexo F - Prometheus + Grafana yaml files . . . . .	66

## ÍNDICE DE TABELAS

2.1	Tabela de comparação de ferramentas de virtualização. . . . .	15
2.2	Tabela de comparação de ferramentas de containers . . . . .	15
7.1	Leaderboard de Avaliação Geral . . . . .	50

## ÍNDICE DE FIGURAS

1.1	Plano de trabalho diagrama de gantt . . . . .	10
7.1	MariaDB Login . . . . .	47
7.2	MariaDB Password . . . . .	47
7.3	MySQL interface . . . . .	47
7.4	Redis PING . . . . .	48
7.5	MinIO Credentials . . . . .	49
7.6	MinIO Test . . . . .	49
7.7	Wordpress Initial config . . . . .	51
7.8	Wordpress Site config . . . . .	52
7.9	ModSecurity Test . . . . .	53
7.10	Gitlab Login . . . . .	55
7.11	Nextcloud Test . . . . .	56
7.12	Grafana dashboard . . . . .	57

## LISTINGS

4.1	Exemplo de um ficheiro de configuração para utilizar uma imagem . . . . .	25
4.2	Utilização de uma imagem diretamente no cluster . . . . .	25
4.3	Exemplo de um ficheiro de configuração de um deployment . . . . .	25
4.4	Exemplo da criação de um namespace diretamente . . . . .	26
4.5	Exemplo de um ficheiro de configuração de um namespace . . . . .	26
4.6	Exemplo de um ficheiro de configuração de um pod . . . . .	26
4.7	Criação de um ficheiro secret . . . . .	27
4.8	Aplicação de um ficheiro secret a um pod . . . . .	27
4.9	Criação de um RBAC role . . . . .	28
4.10	Criação de um RoleBinding . . . . .	28
4.11	Criação de um ServiceAccount que age como conta pelos pods . . . . .	29
4.12	Criação de um replicaset que garante sempre 3 replicas ativas de um POD com imagem nginx . . . . .	29
4.13	Exemplo de ClusterIP . . . . .	30
4.14	Exemplo de NodePort . . . . .	30
4.15	Exemplo de LoadBalancer . . . . .	30
4.16	Exemplo de ExternalName . . . . .	31
4.17	Exemplo de um ficheiro de configuração para um ingress . . . . .	31
4.18	Exemplo de um ficheiro de configuração para um volume . . . . .	32
4.19	Exemplo de um ficheiro de configuração para um PVC . . . . .	33
4.20	Exemplo de um ficheiro de configuração para uma storage Class . . . . .	33
4.21	Utilização de um storage Class em um PVC . . . . .	33
4.22	Utilização de um storage Class em um PVC . . . . .	34

## **LISTA DE SIGLAS E ACRÓNIMOS**

VM	Virtual Machine
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
ISEC	Instituto Superior de Engenharia de Coimbra
ISP	Internet Service Provider
TLD	Top Level Domain
DDOS	Distributed Denial of Service
TLS	Transport Layer Security
SSL	Secure Socket Layer
AD	Active Directory
LDAP	Lightweight Directory Access Protocol
CA	Certificate Authority
CNI	Container Network Interface
RBAC	Role Base Access Control
SGBDR	Sistema de Gestão de Base de Dados Relacional
CI	Continuous Integration
CD	Continuous Delivery

# 1 INTRODUÇÃO

Este relatório tem o propósito de detalhar as tarefas executadas durante o decorrer do primeiro semestre do ano letivo de 2023/2024. Estas atividades foram desenvolvidas durante a unidade curricular Projeto ou Estágio, parte integrante da licenciatura em Engenharia Informática, com uma especialização voltada para o ramo de Redes e Administração de Sistemas. Este curso é oferecido pelo ISEC. O estágio mencionado no presente documento resultou de uma parceria com a organização OneSource.

## 1.1 Enquadramento

## 1.2 Entidade de acolhimento

*OneSource é uma empresa de TI especializada nas áreas de comunicações de dados, segurança, redes e gestão de sistemas, incluindo consultoria, auditoria, design, desenvolvimento e administração de longo prazo de soluções de TI especializadas para redes corporativas, instituições do setor público, serviços públicos e operadoras de telecomunicações.*<sup>1</sup>

## 1.3 Objetivos

*"Art.º 9º Os objectivos específicos dependem do projeto/estágio a desenvolver. Em termos gerais, pretende-se que os alunos realizem um trabalho que lhes possibilite a integração/aplicação das competências previamente adquiridas nas várias unidades curriculares da LEI, particularmente nas do ramo que frequentaram."*<sup>2</sup> Neste estágio foi-me proposto desenvolver competências face a manipulação micro-serviços/aplicações implementadas em cluster kubernetes. Compreender o funcionamento dos mesmos e como disponibiliza-los em ambiente de produção.

---

<sup>1</sup>OneSource Website

<sup>2</sup>Regulamento UCPE

## 1.4 Plano e metodologia de trabalho

O estágio teve início no dia 02 de Outubro de 2023, tendo terminado no dia 12 de Fevereiro de 2024. Foi feito o seguinte plano de trabalhos, de acordo com a proposta de estágio, e também de acordo com os requisitos definidos pela entidade de acolhimento:

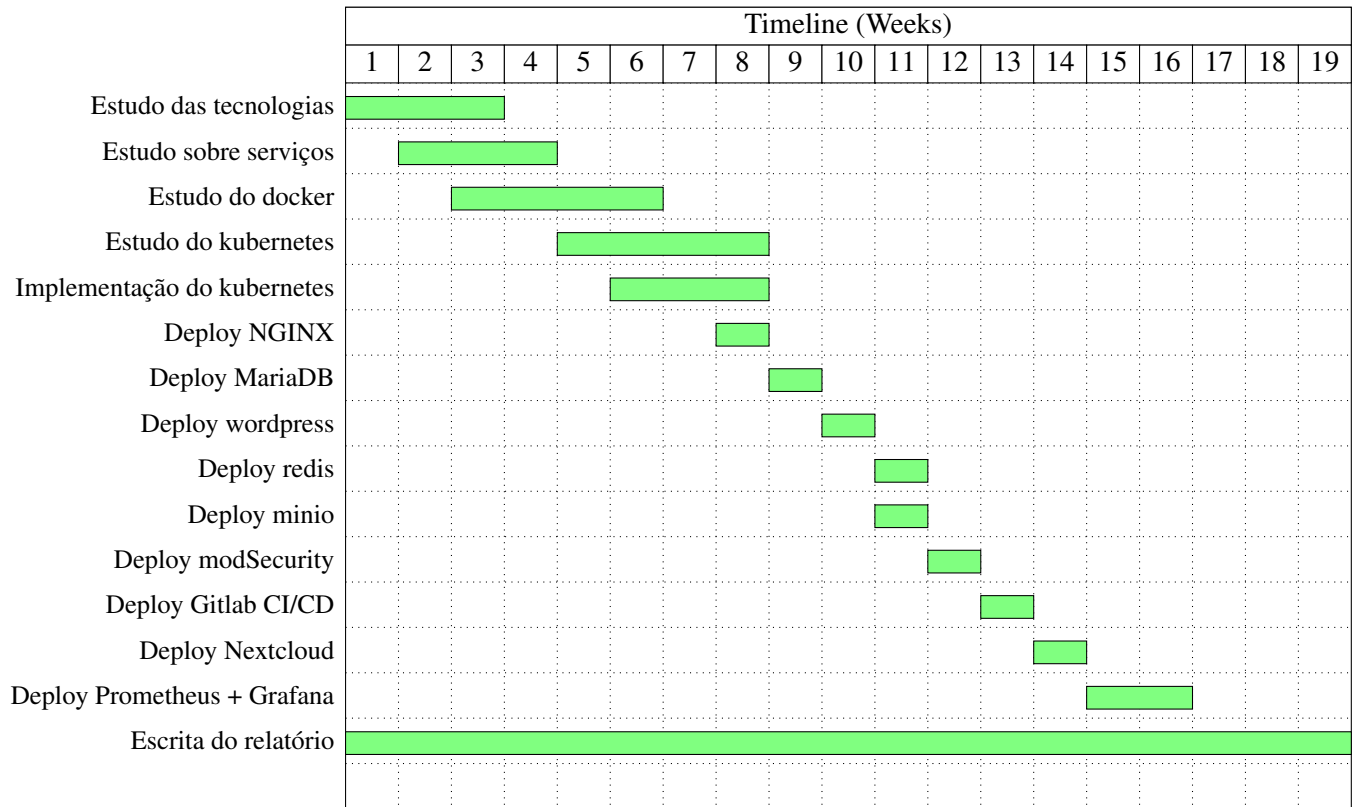


Figura 1.1: Plano de trabalho diagrama de gantt

Foi feita uma média de 40 horas de trabalho semanais, tendo estendido em algumas semanas e encurtado em outras.

## 2 VIRTUALIZAÇÃO

A virtualização é uma tecnologia que permite a criação de ambientes virtuais dentro de um sistema físico. Estes ambientes são isolados e operam como se fossem sistemas independentes, embora compartilhem os recursos físicos da máquina hospedeira. A virtualização começou a aparecer por volta do ano 1960-1970 e em 1974 Gerald J. Popek e Robert P. Goldberg definiram as condições necessárias para que um sistema fosse considerado "virtualizável" no seu artigo seminal "Formal Requirements for Virtualizable Third Generation Architectures". onde dizem que as três condições principais são:

- **Equivalência com o Hardware Nativo** - A arquitetura virtualizada deve ser capaz de executar programas de maneira que seja impossível para o utilizador detetar a diferença entre a execução na máquina virtual e na máquina física não virtualizada.
- **Eficiência de Controle** - O hypervisor (ou monitor de máquina virtual) deve ter controle completo sobre todas as operações que afetam o comportamento virtualizado. Isso inclui todas as instruções privilegiadas que normalmente só poderiam ser executadas pelo sistema operacional.
- **Eficiência de Desempenho** - A grande maioria das instruções do utilizador deve ser executada diretamente pelo hardware subjacente, sem intervenção do hypervisor. Isto é crucial para garantir a execução eficiente dos programas nas máquinas virtuais.

<p><b>Hypervisor:</b> Um hypervisor, também conhecido como monitor de máquina virtual (VMM - Virtual Machine Monitor), é um software que permite a execução de várias máquinas virtuais em um único hardware físico. A sua função principal é criar e gerir ambientes virtuais, isolando-os uns dos outros e do sistema operacional host.</p>
---

### 2.1 Tipos de virtualização

Existem dois tipos principais de virtualização, cada um com suas próprias características e casos de uso específicos:

#### **Virtualização de Hardware (ou Bare-Metal)**

A virtualização de hardware, também conhecida como virtualização bare-metal, ocorre quando um hypervisor é instalado diretamente no hardware físico da máquina, sem a necessidade de um sistema operacional hospedeiro. Esse tipo de virtualização oferece um desempenho superior, pois o hypervisor tem acesso direto aos recursos do hardware. É comumente utilizado em



ambientes de data centers e servidores, onde a eficiência e a capacidade de gerir recursos são fundamentais. Exemplos de hypervisor bare-metal incluem VMware ESXi, Microsoft Hyper-V, Xen, Proxmox...

### **Virtualização Baseada em Sistema Operacional**

A virtualização baseada em sistema operacional, também conhecida como virtualização hospedada, ocorre quando um hypervisor é executado sobre um sistema operacional hospedeiro. Nesse caso, o sistema operacional hospedeiro fornece recursos para o hypervisor, que, por sua vez, cria e gere as máquinas virtuais. Embora essa abordagem possa ter um desempenho ligeiramente inferior em comparação com a virtualização de hardware, oferece maior flexibilidade e é frequentemente usada em ambientes de desktop virtual (VDI) e desenvolvimento/teste. Exemplos de hypervisor baseados em sistema operacional incluem Oracle VirtualBox, VMware Workstation, Docker, Minikube...

### **Virtualização em container e Microserviços**

A virtualização em containers e microserviços representa uma abordagem mais leve e flexível, onde as aplicações são encapsuladas junto com suas dependências. Cada container compartilha o mesmo kernel do sistema operacional hospedeiro, resultando em elevada eficiência e rápida inicialização. Essa abordagem é ideal para ambientes de desenvolvimento e implementação ágil, permitindo escalabilidade e orquestração eficiente de microserviços. Exemplos populares de plataformas de containers incluem Docker, Kubernetes e OpenShift.

**Container:** Um contêiner é um pacote de software leve, independente, executável, isolado, portátil e escalável que inclui tudo o que é necessário para executar um pedaço de software, incluindo o código, tempo de execução, bibliotecas e ferramentas do sistema.

**Kernel:** O termo "kernel" refere-se ao núcleo do sistema operativo de um computador. Fornece recursos e gere os recursos utilizados pelos softwares e hardware.

## **2.2 Containers e a sua Orquestração**

### **2.2.1 Contêineres VS Virtualização**

Enquanto a virtualização tradicional utiliza máquinas virtuais para isolar sistemas operacionais inteiros, os contêineres isolam apenas o aplicativo e suas dependências, compartilhando o kernel do sistema operacional com o host. Isso resulta em tempos de inicialização mais rápidos, uma utilização de recursos menor e maior eficiência.

## 2.3 Docker e Kubernetes

- **DOCKER** - Docker é uma plataforma de que ajuda na implementação, escalonamento e gestão manual de containers. Fornece uma maneira fácil de criar, distribuir e executar aplicativos em containers. Cada container Docker é baseado em uma imagem que inclui o aplicativo e suas dependências.
- **KUBERNETES** - Kubernetes é uma plataforma de orquestração de containers que automatiza a implementação, o escalonamento, gestão e recuperação de aplicativos em containers. Facilita a operação eficiente de aplicativos em grande escala, gerindo automaticamente a distribuição de containers em um cluster, escalando conforme necessário entre os recursos disponíveis proporcionando resiliência, eficácia e disponibilidade a aplicativos baseados em containers. Conta com características como:
  - **Rollouts e Rollbacks Automáticos** - Aplica modificações e configurações nas aplicações e em caso de falha recupera estados anteriores automaticamente.
  - **Discovery e Load-balancing** - Distribui um nome DNS e um IP a cada conjunto de PODs e faz o balanceamento da carga entre eles.
  - **Storage Orchestration** - Utiliza sistemas de armazenamento Local, Cloud, remoto com iSCSI ou NFS.
  - **Self healing** - Reiniciar, substituir, reagendar, e desligar containers que não respondem ao health-check definido. Estes não são dispostos para o cliente enquanto não estiverem prontos a ser utilizados.
  - **Secret and configuration manager** - Aplica atualizações nas aplicações sem as reiniciar e sem expor conteúdo sensível.
  - **Automatic Bin packing** - Distribui containers automaticamente face as necessidades dos clientes assegurando a disponibilidade dos mesmos.
  - **Batch execution** - O kubernetes executa e gere as suas próprias tarefas sem intervenção direta do administrador. Estas tarefas são referentes a grandes volumes de dados (BATCH) e a compilação e teste de código (CI workloads).
  - **Horizontal scaling** - Faz o escalonamento automático de recursos que uma aplicação utiliza para garantir a disponibilidade da mesma. Este escalonamento baseia-se na multiplicação de containers que trabalham em conjunto para disponibilizar o mesmo serviço por forma a disponibiliza-lo para um maior número de utilizadores.
  - **IPV4/IPV6 dual-stack** - Alocação de ipv4 e ipv6 para pods e serviços.
  - **Designed for extensibility** - Capacidade de adição de novas ferramentas sem necessidade de modificar o código base.

<b>POD:</b> Grupo de um ou mais containers com um sistema de armazenamento e um sistema de rede partilhados.
--

**iSCSL** Internet Small Computer system interface é um protocolo de transporte em um rede tcp/ip que une um alvo de armazenamento a um dispositivo. Estabelece a conexão a um disco rígido remoto como se o mesmo estivesse conectado fisicamente.

**NFS** Network file system é um sistema de partilha de diretorias entre computadores que utilizam a mesma rede.

## 2.4 Vantagens da orquestração de containers

A orquestração de containers é um processo de automatização de implantação, escalabilidade e gestão de aplicações em containers.

- **Escalabilidade:** Tecnologia de scale out envolvem adicionar mais nodes a um sistema para distribuir a carga do mesmo. Em vez de se adicionar mais capacidade a uma máquina são adicionadas mais máquinas à infraestrutura que trabalham para um objetivo comum.
- **Resiliência:** ou tolerância a falhas garante a disponibilidade contínua das aplicações, mesmo em caso de falhas em alguns containers.
- **Gerenciamento Simplificado:** Automatiza tarefas como implementação, atualização e monitorização de aplicações em containers.

## 2.5 Ferramentas de virtualização

Existem várias ferramentas de virtualização disponíveis, cada uma com suas características e casos de uso específicos. Eis brevemente algumas delas:

1. **VMWare** - A família de produtos VMware inclui diversas soluções, cada uma com suas próprias características. Alguns dos produtos mais conhecidos são:
  - **VMware Workstation:** Uma solução para desktops que suporta a execução de várias máquinas virtuais em um único host. Pode ser utilizada para desenvolvimento, testes e demonstrações.
  - **VMware vSphere:** Uma plataforma para virtualização de data centers, proporcionando recursos avançados de gestão, segurança e escalabilidade.
2. **XEN** - Xen é uma plataforma de virtualização de código aberto que suporta máquinas virtuais (VMs) e containers. É conhecido pelo seu desempenho sólido e é amplamente utilizado em ambientes de cloud.
3. **ProxMox** - Proxmox Virtual Environment (Proxmox VE) é uma solução de virtualização de código aberto que combina virtualização de servidores com container. Utiliza o hypervisor KVM para máquinas virtuais e o LXC para containers, e oferece uma plataforma integrada de gestão.

4. Docker - Docker é uma plataforma de containers que simplifica a implantação e a execução de aplicativos em containers. É conhecido pela sua leveza e rapidez na inicialização e na integração contínua.
5. Kubernetes - Kubernetes, frequentemente abreviado como K8s, é uma plataforma de orquestração de containers de código aberto e altamente utilizada. Automatiza a implantação, escalonamento e gestão de containers, facilitando a operação eficiente de aplicações em larga escala. O kubernetes oferece características como portabilidade, gestão de recursos, monitorização e logging integrado, grande variedade de pluggins, etc...
6. Docker Swarm - Docker Swarm é uma ferramenta nativa do Docker para orquestração de containers. Permite a criação e gestão de clusters de containers Docker.
7. Minikube - Minikube é uma ferramenta que facilita a execução de clusters Kubernetes locais, permitindo o desenvolvimento e teste de aplicações Kubernetes em um ambiente isolado. É designado como um cluster de 1 só node.

Tabela 2.1: Tabela de comparação de ferramentas de virtualização.

	<b>Vmware WS player</b>	<b>Virtualbox</b>	<b>Xen</b>	<b>Proxmox</b>	<b>Vmware vSphere</b>
<b>Learning curve</b>	Easy	Easy	Hard	Hard	Hard
<b>Performance</b>	Good	Standard	Top	Top	Top
<b>Cost</b>	Free	Free	Free	Free	Free
<b>Resource Usage</b>	Moderate	Moderate	Intense	Intense	Variable
<b>Security</b>	Limited	Moderate	Secure	Secure	Secure
<b>Use Case</b>	Test + develop	Test	Test	Develop	Develop

Tabela 2.2: Tabela de comparação de ferramentas de containers

	<b>Minikube</b>	<b>Docker Swarm</b>	<b>Kubernetes</b>
<b>Learning curve</b>	Easy	Easy	Moderate
<b>Performance/Features</b>	Standard	Good	Good
<b>Cost</b>	Free	Free	Free
<b>Resource Usage</b>	Low	Variable	Variable
<b>Security</b>	None	Variable	Secure/Variable
<b>Use Case</b>	Test	Test	Develop

## 3 SERVIÇOS DE REDE

### 3.1 DHCP

#### 3.1.1 Introdução

O protocolo DHCP é um protocolo utilizado para alocação dinâmica de parâmetros de rede como endereços IP, máscaras, default gateways, servidores dns etc...

#### 3.1.2 Componentes

- Cliente DHCP - Dispositivo que solicita a configuração de rede.
- Servidor DHCP - Dispositivo responsável por conceder parâmetros de rede
- Pool DHCP - Intervalo de endereços IP gerido por um servidor
- Relé DHCP - Quando os clientes e servidores não estão na mesma sub-rede são utilizados relés DHCP para encaminhar as solicitações e respostas DHCP

#### 3.1.3 Processo

- DHCP discover - O cliente enviar um pacote de *discover* para a rede solicitando um servidor DHCP disponível.
- DHCP Offer - O servidor DHCP disponível responde com uma oferta contendo um endereço disponível e outras configurações.
- DHCP Request - O cliente seleciona uma oferta das que recebeu e envia uma solicitação ao servidor que a produziu.
- DHCP Acknowledge - O servidor confirma a solicitação do cliente e envia uma confirmação com o endereço IP e outras configurações.

#### 3.1.4 Tempo de vida

Os endereços IP atribuídos pelo DHCP têm um tempo de vida limitado. Os clientes podem renovar ou libertar estes endereços antes do vencimento.

### 3.1.5 Segurança

O protocolo DHCP é suscetível a ataques, como servidores DHCP falsos, nos quais um dispositivo não autorizado assume o papel de um servidor legítimo DHCP, distribuindo endereços IP falsos. Isso pode impedir que o dispositivo acesse a rede, ou até mesmo atribuir um IP válido, mas distribuir-se como o *default gateway*. Como resultado, o dispositivo que solicita parâmetros de rede começa a enviar todos os pacotes diretamente para o servidor DHCP falso. Esse servidor DHCP falso pode analisar e interceptar esses pacotes, potencialmente resultando em roubo de informações antes de redirecioná-los para o destino correto. Esse tipo de ataque é comumente conhecido como ataque *man-in-the-middle*.

Para proteger contra esses ataques, é essencial implementar configurações apropriadas no DHCP. Os clientes devem ser configurados para ignorar ofertas DHCP não autorizadas, e a monitorização da rede deve estar em vigor para detetar atividades suspeitas. Além disso, a segmentação da rede é crucial, isolando partes críticas para limitar a exposição a possíveis ataques. Isso pode ser alcançado por meio de técnicas como autenticação de servidores DHCP, VLANs, políticas de acesso, etc. Por fim, o uso de *firewalls* ou sistemas de deteção e prevenção de intrusões (IDS/IPS) pode aprimorar a segurança geral.

### 3.1.6 DHCP no meu projeto

Sendo que o projeto a desenvolver no estágio se baseia num sistema de cluster Kubernetes e todos os serviços que poderão ser usados em conjunto com este dentro de uma empresa. Um servidor DHCP que, dinamicamente atribua ip's aos vários PODS e serviços gerados é uma boa prática a adotar.

## 3.2 DNS

### 3.2.1 Introdução

DNS é um protocolo essencial da Internet que converte nomes de domínio legíveis por humanos em endereços IP e vice-versa, facilita a comunicação entre computadores em uma rede distribuída. Em vez de depender exclusivamente de números IP para identificar servidores, dispositivos, e serviços o DNS oferece uma camada de abstração amigável, associando nomes de domínio, como *www.exemplo.com*, a endereços IP correspondentes.

### 3.2.2 Hierarquia

A hierarquia neste protocolo define a estrutura organizacional dos nomes de domínio e dos seus servidores associados.

- **Dominio Raiz** - O nível mais alto que contem os servidores raiz.

- **Domínios de primeiro nível** - domínios como o .com, .pt ...
- **Subdomínios** - criados pelos proprietários de domínio como o ISEC.

### 3.2.3 Processo

O processo neste protocolo delinea a sequência de etapas envolvidas na resolução de um nome de domínio para seu endereço IP correspondente dentro da nossa área.

- **Entrada do Usuário:** Primeiro digita-se o URL "www.isec.pt" no navegador web.
- **Cache DNS da Máquina Local:** O computador verifica a cache DNS local para ver se já possui o endereço IP para www.isec.pt armazenado. Se sim, pode saltar as próximas etapas e aceder ao site diretamente.
- **Resolver DNS Local:** Se o endereço IP não for encontrado na cache local, o computador envia uma consulta DNS para o seu resolver DNS local. Geralmente, isto é fornecido pelo Provedor de Serviços de Internet (ISP).
- **Cache do Resolver DNS:** O resolver DNS local verifica a sua cache para ver se já conhece o endereço IP para www.isec.pt. Se não conhecer prossegue para a próxima etapa.
- **Servidores DNS Raiz:** O resolver DNS local envia uma consulta a um dos 13 servidores DNS raiz. Esses servidores fornecem informações sobre os servidores DNS autoritários para domínios de primeiro nível (TLDs).
- **Servidores DNS de TLD (.pt):** O servidor DNS raiz responde com os endereços IP dos servidores DNS de TLD responsáveis pelo TLD ".pt" (domínio de topo de código de país de Portugal). O resolver DNS local então consulta um desses servidores TLD.
- **Servidores DNS Autoritários para isec.pt:** O servidor DNS de TLD responde com os endereços IP dos servidores DNS autoritários para "isec.pt."
- **Servidores DNS Específicos do Domínio:** O resolver DNS local consulta os servidores DNS autoritários para "isec.pt" para obter o endereço IP de www.isec.pt.
- **Resolução de Endereço IP:** Os servidores DNS autoritários respondem com o endereço IP associado a www.isec.pt.
- **Carregando o Site:** O computador recebe o endereço IP e utiliza-o para estabelecer uma conexão com o servidor web que hospeda www.isec.pt. Em seguida, o navegador faz o download do conteúdo da página web, e passa a poder visualizar o site.

### 3.2.4 Registos

Os registos DNS são entradas de um banco de dados que mapeiam nomes de domínio e/ou IP's para informações associadas como endereços IP.

- A (Address)
- AAAA (IPv8 Address)
- CNAME (Canonical name)
- MX (Mail Exchange)
- TXT (Text)
- NS (Name Server)

### **3.2.5 Segurança**

O Sistema de Nomes de Domínio (DNS) enfrenta vários desafios de segurança, incluindo tentativas de comprometer a integridade dos dados DNS. Os atacantes podem utilizar técnicas como envenenamento de cache, onde informações falsas são injetadas na cache DNS, levando ao redirecionamento de utilizadores para websites ou máquinas maliciosas que podem, então, roubar dados de forma despercebida. Os servidores DNS também são vulneráveis a ataques de Negação de Serviço Distribuído (DDoS), nos quais atores maliciosos inundam os servidores com um volume massivo de pedidos, tornando-os não responsivos a consultas legítimas. Além disso, o túnel DNS é uma técnica usada para contornar medidas de segurança encapsulando tráfego não relacionado com DNS dentro de pacotes DNS, permitindo que os atacantes obtenham dados sem serem detetados.

### **3.2.6 DNSSEC**

O DNSSEC, ou Extensões de Segurança do DNS, é uma tecnologia que adiciona uma camada adicional de segurança ao Sistema de Nomes de Domínio (DNS). O principal objetivo do DNSSEC é proteger contra ataques de manipulação e falsificação de dados DNS, garantindo a autenticidade e integridade das informações associadas aos nomes de domínio utilizando assinaturas digitais e pares de chaves criptografadas.

### **3.2.7 DNS no meu projeto**

Dentro do cluster kubernetes a ser implementado são disponibilizados vários serviços acessíveis através de IP's e ports. Para um cliente, aceder a estes serviços digitando os seus IP's pode ser pouco eficaz, utiliza-se então o DNS para disponibilizar IP's de serviços através de nomes.

## **3.3 VPN**

### **3.3.1 Introdução**

Uma VPN, ou Rede Privada Virtual, é uma tecnologia que proporciona a criação de uma ligação segura e cifrada através de uma rede pública. Esta ferramenta essencial no mundo digital



estabelece uma ponte virtual entre o dispositivo do utilizador e o servidor VPN, conferindo benefícios significativos em termos de segurança e privacidade.

### 3.3.2 Funcionamento

- **Túnel Criptografado:**  
Um "túnel" criptografado entre o dispositivo do utilizador e o servidor VPN. Este túnel assegura que todas as comunicações, sejam elas navegação na web, transferência de dados, ou qualquer outra atividade online, sejam protegidas por cifragem. Isto impede que terceiros indesejados tenham acesso ou possam monitorizar as informações transmitidas.
- **Endereço IP Oculto:**  
A capacidade de ocultar o endereço IP real do utilizador. Em vez de utilizar o seu próprio endereço IP, o utilizador passa a utilizar o endereço IP do servidor VPN. Esta medida proporciona anonimato online, uma vez que as suas atividades na Internet estão associadas ao endereço IP do servidor VPN, dificultando a identificação ou rastreamento por parte de terceiros.
- **Acesso a Recursos Privados:**  
As VPNs são amplamente utilizadas por empresas para fornecer aos colaboradores acesso seguro a recursos internos da rede, mesmo quando estão a trabalhar remotamente. Esta funcionalidade é particularmente valiosa para garantir a segurança das comunicações e a integridade dos dados, uma vez que todas as informações transmitidas através da VPN são protegidas pela cifragem.

### 3.3.3 Tipos

- **Remote Access VPN:**  
Usada para aceder a recursos de rede internos.
- **Site-to-Site VPN:**  
Conecta redes internas entre si, como por exemplo os vários departamentos de uma empresa.
- **Point-to-Point VPN:**  
Estabelece uma conexão segura entre dois dispositivos.
- **Application-Level VPN:**  
Utilizado por aplicações para redirecionar o tráfego das mesmas através de uma conexão VPN.
- **SSL VPN:**  
Utiliza os protocolos SSL/TLS para disponibilizar conexões seguras via browser. Uses the SSL/TLS protocol to provide secure connections via a web browser.

- **MPLS VPN:**  
Solução de rede segura utilizado por empresas para criar conexões seguras.

### 3.3.4 Benefícios

- **Privacidade e Segurança:** A VPN protege as comunicações online, impedindo que terceiros acessem as informações sensíveis.
- **Anonimato Online:** Ao ocultar o endereço IP, a VPN proporciona um nível de anonimato adicional durante as atividades online.
- **Acesso a Conteúdo Restrito Geograficamente:** Permite contornar restrições geográficas e acessar conteúdo online bloqueado em determinadas regiões.
- **Conexões Seguras em Redes Públicas e Privadas:** Ideal para utilizar redes Wi-Fi públicas com segurança adicional, bem como para garantir a segurança em redes corporativas.

### 3.3.5 VPN no meu projeto

No que toca ao meu projeto alguns serviços podem ser comprometidos se forem disponibilizados publicamente, no entanto é importante que os colaboradores do mesmo sejam capazes de modificá-los. Para isso pode ser implementada uma VPN interna ao kubernetes que permite que utilizadores, simulando a sua conexão interna, possam acessar recursos disponibilizados numa rede privada.

## 3.4 Serviços de diretorias

### 3.4.1 Introdução

Um serviço de diretoria é um sistema em rede de armazenamento, organização, disponibilização e gestão de informações sobre recursos como dispositivos, utilizadores, grupos, políticas, etc...

### 3.4.2 Conceitos

- Recursos de rede - computadores, servidores, impressoras, aplicativos, arquivos de recursos...
- Organização hierárquica
- Atributos - Cada objeto de diretoria pode ter recursos associados como nome, email, senha...

### 3.4.3 Funcionalidades

- Autenticação

- Autorização
- Gestão de recursos
- Resolução de nomes
- Políticas de grupo

#### **3.4.4 Tipos**

- AD (active directory) - serviço da Microsoft como foco em ambientes windows
- LDAP (Lightweight Directory Access Protocol) - Protocolo padrão utilizado em vários serviços de diretoria para buscar informação e autenticar usuários.
- OpenLDAP - Implementação de código aberto do LDAP usado em sistemas UNIX.

### **3.5 SSL**

#### **3.5.1 Introdução**

Protocolo de segurança na Internet que fornece conexões seguras entre cliente e servidor. Previne intercepção e adulteração de dados.

#### **3.5.2 Funcionamento**

- Handshake - Troca de informação entre o cliente e o servidor para concordar em um conjunto comum de algoritmos criptográficos.
- Criptografia de dados - Dados transmitidos são criptografados usando as chaves partilhadas anteriormente.
- Integridade de dados - Utilização de uma *hash* para comprovar integridade dos dados.
- Autenticação de servidor - usando certificados o servidor também consegue garantir a sua autenticação.

#### **3.5.3 Aplicações**

- Navegadores WEB
- Email
- VPN
- Transações financeiras

### **3.5.4 SSL no meu projeto**

Dentro do kubernetes existem várias ferramentas e *binaries* que precisam de comunicar entre si. Qualquer dispositivo pode ter estas ferramentas e consequentemente poderia interligar-se aos meus dispositivos e adulterar o cluster. Para prevenir isso são utilizados certificados SSL para garantir que os dispositivos usados dentro do cluster são apenas os autorizados.

## 4 CONTAINER RUNTIME

Nesta secção vou falar um pouco sobre os vários componentes necessários para criar, disponibilizar e gerir uma aplicação dentro do meu cluster. Para isso primeiro vamos falar um pouco sobre estes componentes:

### 4.1 Configuration File

No kubernetes ficheiros de configuração são criados em editores de texto e usados para definir e gerir recursos. Estes ficheiros descrevem como os recursos que queremos criar são criados, atualizados e utilizados.

Nestes ficheiros são inseridos campos como:

- `apiVersion` : Especifica a versão do kubernetes.
- `kind` : Especifica o tipo de recurso a ser criado.
- `metadata` : Contem a informação sobre o recurso como o nome, labels, anotações, etc ...
- `Spec` : define o estado incluindo parâmetros adicionais ao tipo de recurso.
- `Status` : Define o estado atual do recurso (esta linha é normalmente inserida e atualizada pelo kubernetes).

Depois de criados os ficheiros de configuração, os recursos podem ser criados com estes ficheiros através do comando:

```
kubectl apply -f <filename.yaml>
```

Este comando pode ter algumas opções como:

- `-dry-run` : executa o ficheiro mas sem efetuar modificações ao cluster. Útil para testes.
- `-validate` : valida o ficheiro sem o aplicar.
- `-namespace (-n)` define o namespace onde se quer criar/atualizar o recurso.
- `-filename (-f)` : permite definir múltiplos ficheiros ou diretorias que contêm os ficheiros de configuração.

### 4.2 Imagens

O termo imagem em kubernetes refere-se aos containers pré-construídos por entidades e disponibilizados para uso pelo publico. Contêm o ambiente de execução de uma aplicação em conjunto

com o código e as suas dependências e configurações. Estas imagens podem ser aplicadas a aplicações editando os seus ficheiros de configuração:

```
spec :  
  containers :  
    - name: myapp  
      image: myregistry /myapp:latest
```

Listing 4.1: Exemplo de um ficheiro de configuração para utilizar uma imagem

Ou atualizar a imagem de um container em execução como por exemplo um deployment.

```
kubectl set image deployment/mydeployment  
mycontainer=myregistry /myapp:newversion
```

Listing 4.2: Utilização de uma imagem diretamente no cluster

## 4.3 Componentes

### 4.3.1 Deployment

No kubernetes as aplicações lançadas nos seus PODS por vezes precisam de outros componentes como serviços para comunicação dentro do cluster, volumes para guardar dados e muitos outros que iremos ver de seguida. Para a simplificação do processo o deployment é um conjunto de componentes que juntos disponibilizam uma aplicação pronta para utilização. Exemplo:

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: example-deployment  
spec:  
  replicas: 3  
  selector:  
    matchLabels:  
      app: example  
  template:  
    metadata:  
      labels:  
        app: example  
    spec:  
      containers:
```

```
- name: nginx  
  image: nginx:latest
```

Listing 4.3: Exemplo de um ficheiro de configuração de um deployment

Este deployment com o nome `nginx-deployment` especifica:

- **replicas: 3** - 3 replicas do POD Nginx.
- **selector: app: nginx** - usa a label `nginx` para saber que PODS gerir.
- **template: containers:** - especifica o nome do container e a imagem a utilizar.

Dentro destes deployments podem ser inseridas várias configurações de vários serviços desde que sejam separados com uma linha que contenha `--`.

### 4.3.2 Namespace

Um Namespace no kubernetes pode ser comparado a uma diretoria. É basicamente um novo sitio com um nome onde podem ser lançados serviços. Isto disponibiliza a separação de serviços com maior eficácia.

Para criar um namespace podemos utilizar o `kubectl`:

```
kubectl create namespace <NAMESPACE_NAME>
```

Listing 4.4: Exemplo da criação de um namespace diretamente

Ou utilizando um ficheiro de configuração:

```
apiVersion: v1  
kind: Namespace  
metadata:  
  name: example-namespace
```

Listing 4.5: Exemplo de um ficheiro de configuração de um namespace

### 4.3.3 Pod

Um POD é o objeto mais pequeno do kubernetes. Representa uma instância de um processo e encapsula o(s) containers, recursos de armazenamento, e o IP único de uma aplicação. Um POD pode conter mais do que um container dentro da mesma rede permitindo-os comunicar usando **localhost**.

```
apiVersion: v1  
kind: Pod
```

```
metadata:
  name: mypod
spec:
  containers:
  - name: mycontainer
    image: nginx:latest
```

Listing 4.6: Exemplo de um ficheiro de configuração de um pod

Este POD é criado com o nome **mypod** e corre um container **Nginx**.

É importante ter em conta que um POD é normalmente gerido por um deployment ou statefulSet que fornecem ferramentas de escalonamento, atualização, entre outras...

### 4.3.4 Secret

No kubernetes **Secrets** são utilizados para guardar informação sensível como credenciais.

Tipos:

- **Generic** - Valores arbitrários de pares Key-Value.
- **TLS** - Usado para guardar certificados TLS.
- **Docker-registry** - Credencias para aceder a registos do Docker.
- **Service account tokens** - Criados automaticamente pelo servidor API usado por contas de serviço.

```
kubectl create secret generic mysecret --from-literal=username=admin --from-literal=password=secret
```

Listing 4.7: Criação de um ficheiro secret

Criação de um **Secret** com o valor do Username e da Password

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mycontainer
    image: nginx:latest
    volumeMounts:
    - name: secret-volume
      mountPath: "/etc/secret-data"
```



```
volumes:
- name: secret-volume
  secret:
    secretName: mysecret
```

Listing 4.8: Aplicação de um ficheiro secret a um pod

### 4.3.5 Role-Based Access Control

Role-Based Access Control é uma ferramenta no kubernetes que permite controlar o acesso a recursos baseado em roles distribuídos pelos utilizadores.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: your-namespace
  name: secret-reader
rules:
- apiGroups: ["" ]
  resources: ["secrets"]
  verbs: ["get"]
```

Listing 4.9: Criação de um RBAC role

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: secret-reader-binding
  namespace: your-namespace
subjects:
- kind: User # or ServiceAccount for service accounts
  name: your-username
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

Listing 4.10: Criação de um RoleBinding

Com estas duas configurações asseguramos que o utilizador "**your-username**" tem a permissão para ler o **secret** no namespace com o nome "your-namespace".

### 4.3.6 ServiceAccount

Um serviceAccount é um serviço que pode ser usado por PODS dentro de um namespace definido para os mesmo se autenticarem e interagirem com a API kubernetes dependendo das permissões RBAC associadas a eles.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: secret-reader
  namespace: your-namespace
```

Listing 4.11: Criação de um ServiceAccount que age como conta pelos pods

### 4.3.7 ReplicaSet

O ReplicaSet é um objeto do kubernetes que garante um número especificado de réplicas de PODS em execução.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
  namespace: your-namespace
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: nginx:latest
```

Listing 4.12: Criação de um replicaset que garante sempre 3 replicas ativas de um POD com imagem nginx

É importante ter em conta que um ReplicaSet deve ser sempre gerado e mantido por um deployment para facilitar a sua atualização controlada.

### 4.3.8 Service

Serviços no kubernetes representam a comunicação entre conjuntos de PODS e outras interfaces.

Tipos:

- **ClusterIp**- Expõe o serviço apenas dentro do cluster.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Listing 4.13: Exemplo de ClusterIP

- **NodePort**- Expõe o serviço em um port específico em cada nó do cluster.

```
apiVersion: v1
kind: Service
metadata:
  name: my-nodeport-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: NodePort
```

Listing 4.14: Exemplo de NodePort

- **LoadBalancer**-Expor um serviço através de um Balanceador de carga externo.

```
apiVersion: v1
kind: Service
```

```

metadata:
  name: my-loadbalancer-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer

```

Listing 4.15: Exemplo de LoadBalancer

- **ExternalName**- Mapear um serviço para um nome DNS externo.

```

apiVersion: v1
kind: Service
metadata:
  name: my-external-service
spec:
  type: ExternalName
  externalName: example.com

```

Listing 4.16: Exemplo de ExternalName

### 4.3.9 Ingress

Um ingress no kubernetes é um recurso que gere o acesso externo a serviços dentro de um cluster. Este pode ser configurado para fornecer suporte a HTTPS por meio de certificados TLS fornecendo conexões seguras do exterior. O ingress também pode atuar como um ExternalName service dando nomes DNS aos serviços.

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  rules:
    - host: myapp.com
      http:
        paths:
          - path: /app
            pathType: Prefix

```

```

    backend:
      service:
        name: app-service
        port:
          number: 80
  tls:
    - hosts:
        - myapp.com
      secretName: myapp-tls-secret

```

Listing 4.17: Exemplo de um ficheiro de configuração para um ingress

### 4.3.10 Volumes

No contexto do Kubernetes, um volume pode ser comparado a uma diretoria compartilhada por todos os containers em execução dentro de um POD. É especificado e designado para facilitar a comunicação de dados entre os containers, além de assegurar a persistência desses dados, mesmo em situações de falha ou reinicialização.

```

apiVersion: v1
kind: Pod
metadata:
  name: volume-example
spec:
  containers:
    - name: nginx-container
      image: nginx
      volumeMounts:
        - name: shared-data
          mountPath: /usr/share/nginx/html
  volumes:
    - name: shared-data
      emptyDir: {}

```

Listing 4.18: Exemplo de um ficheiro de configuração para um volume

### 4.3.11 Persistent Volume Claim

Um persistentVolumeClaim no kubernetes age como um pedido de um utilizador para aceder/alocar armazenamento. Este armazenamento é atribuído a esse POD assegurando que está sempre disponível para o mesmo.

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi

```

Listing 4.19: Exemplo de um ficheiro de configuração para um PVC

### 4.3.12 Storage Class

Um storage Class, é um nível de abstração para alocação dinâmica de armazenamento. Permite aos administradores definirem classes de armazenamento com diferentes características como desempenho e espaço e permite também disponibilizar PVC's dinâmicos face a essas classes.

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2

```

Listing 4.20: Exemplo de um ficheiro de configuração para uma storage Class

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  storageClassName: fast
  accessModes:
    - ReadWriteOnce
  resources:
    requests:

```

```
storage: 1Gi
```

Listing 4.21: Utilização de um storage Class em um PVC

## 4.4 StatefulSet

No kubernetes um statefulSet é um nível de abstração mais alto que garante a ordem de operações de dados de aplicações em armazenamentos. Usado para aplicações com estados como bases de dados. Garante nomes previsíveis e estáveis a cada POD. E provisiona redes com identificadores estáveis. É comparado ao deployment mas com a diferença de ser utilizado para aplicações com estados.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  serviceName: "nginx"
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.21.1
          ports:
            - containerPort: 80
  volumeClaimTemplates:
    - metadata:
        name: data
      spec:
        accessModes: [ "ReadWriteOnce" ]
        storageClassName: "standard"
        resources:
          requests:
```

```
storage : 1Gi
```

Listing 4.22: Utilização de um storage Class em um PVC



## **5 SISTEMA DE ORQUESTRAÇÃO DE CONTAINERS**

Um sistema de orquestração de containers é uma plataforma que automatiza a gestão, implementação e escalabilidade de aplicações baseadas em containers. Projetados para simplificar e otimizar operações em ambientes distribuídos, esses sistemas oferecem diversas funcionalidades, tais como:

- Implementação distribuída de aplicações em grande escala.
- Dimensionamento automático do número de containers em resposta à carga de trabalho.
- Distribuição de tráfego para otimizar desempenho e disponibilidade.
- Reinicialização e substituição automáticas de containers com falhas.
- Gestão centralizada das configurações dos containers.

### **5.1 Sistema Distribuído (Cluster)**

Um sistema distribuído ou cluster é um conjunto de máquinas conectadas que trabalham em conjunto para alcançar um objetivo comum. No contexto do sistema de orquestração de containers, um cluster oferece recursos computacionais e de rede para os containers. Esse sistema possibilita escalabilidade pela adição de novas máquinas conforme necessário, confiança pela distribuição de recursos entre os nós para manter operações contínuas em caso de falha, e otimização da distribuição de recursos computacionais entre os containers.

### **5.2 Kubernetes**

O Kubernetes foi o sistema de orquestração de containers escolhido para aprofundamento e testes, sendo um projeto open-source com uma comunidade ativa. Amplamente utilizado na indústria, o Kubernetes oferece portabilidade entre nuvens públicas, privadas e ambientes físicos. Corrige automaticamente falhas, é escalável vertical e horizontalmente, suporta atualizações de aplicações sem interrupções e inclui recursos robustos de segurança, como RBAC (Role-Based Access Control), políticas de rede e isolamento entre PODS e Namespaces.

#### **5.2.1 Arquitetura do Kubernetes**

Um cluster Kubernetes pode adotar várias arquiteturas, incluindo:

- **Arquitetura Básica:** 1 Nó master responsável pela gestão do cluster e vários nodes executando containers.
- **Alta Disponibilidade:** Cluster com 3 ou mais master nodes configurados e múltiplos nós para a carga de trabalho.
- **Master-Slave Nodes:** Masters que também funcionam como worker nodes, envolvendo todos os nodes em funções de administração.
- **Lightweight Cluster:** Arquiteturas como K3s ou MicroK8s, adequadas para clusters com recursos limitados.

### 5.2.2 Kubernetes Control Plane

O Kubernetes Control Plane considerada a parte central do sistema, gere o cluster, e é composto por vários componentes, como:

- **ETCD:** É um armazenamento distribuído altamente disponível que mantém as configurações do cluster, o estado atual e outras informações importantes. Funciona como a base de dados do Kubernetes em si, garantindo consistência e integridade nos dados do cluster.
- **API Server:** Atua como a principal porta de entrada para o Control Plane do Kubernetes. Recebe comandos e consultas, permitindo que utilizadores, outros componentes e até mesmo aplicações interajam com o Kubernetes. É essencial para a comunicação eficiente no sistema do cluster.
- **Controller Manager:** Observa continuamente o estado do cluster e trabalha para garantir que este coincida com o estado desejado. Implementa controladores específicos para gerir recursos e garantir que as políticas definidas sejam aplicadas consistentemente.
- **Scheduler:** Avalia os recursos disponíveis no cluster e os requisitos de um POD para tomar decisões inteligentes sobre onde implantar esses POD's. O Scheduler contribui para a otimização da utilização de recursos e a distribuição equilibrada da carga de trabalho no cluster.
- **Kubelet:** Garante a execução adequada dos PODS no cluster. Recebe instruções do API Server, interage com o container runtime e relata o estado atual dos POD's de volta. Age como um agente essencial para a comunicação e execução eficaz dos containers.
- **Kube-Proxy:** Facilita a comunicação entre os POD's ao gerir regras de encaminhamento de tráfego e balanceamento de carga. Atua como um proxy de rede para garantir a conectividade adequada entre os diversos elementos do cluster. Sua função é crucial para manter a integridade e eficiência das comunicações no ambiente distribuído.

## **5.3 Gestão de Container**

### **5.3.1 Fases dos Pods e Suas Transições**

No Kubernetes, os POD's podem passar por diferentes fases durante seu ciclo de vida:

- Pending: Em criação, os containers ainda não foram iniciados.
- Running: Todos os containers do POD estão em execução.
- Succeeded: Todos os containers do POD foram executados com sucesso e encerrados.
- Failed: Pelo menos um container no POD terminou com um erro.
- Unknown: O estado do POD não pode ser obtido.

## **5.4 Kubernetes e o Container Runtime**

O Container Runtime é responsável por executar, manter e parar containers nos POD's do Kubernetes. Gere imagens, manipula isolamento de recursos e comunica com o Kubelet para coordenar a execução de POD's.

## 6 PROJETO

### 6.1 Descrição

Este projeto visa criar e configurar um cluster Kubernetes do zero por forma a desenvolver um ambiente de produção de micro serviços, proporcionando uma compreensão detalhada de cada etapa envolvida no processo. Note que todos os passos referidos neste capítulo estão disponíveis no repositório Kubernetes-the-hard-way e que aqui apenas foram explicados o que foi feito em cada um deles e não os comandos em si.

### 6.2 Pré-requisitos

Para este projeto é necessário o VirtualBox como ambiente de criação de máquinas virtuais e uma ferramenta chamada Vagrant para automatizar a criação das mesmas.

Neste ambiente a rede das máquinas virtuais vai ser 192.168.56.0/24 mas esta pode ser alterada no "vagrantfile" na linha 9.

A rede utilizada para atribuir endereços IP aos Pods é 10.244.0.0/16 e esta também pode ser alterada nos ficheiros (.md) na diretoria "docs" no parâmetro POD\_CIDR=10.244.0.0/16.

A rede utilizada para atribuir IP's aos serviços do cluster é 10.96.0.0/16 e para a alterar nos mesmos ficheiros (.md) na diretoria "docs" utiliza-se o parâmetro SERVICE\_CIDR=10.96.0.0/16

Adicionalmente no ficheiro coredns.yaml pode definir-se um novo endereço para o serviço DNS (este deve sempre terminar em .10)

### 6.3 Prvisionamento

Primeiramente clonar o repositório github : Kubernetes-the-hard-way De seguida na diretoria "vagrant" executar

```
vagrant up
```

Para aceder a estas máquinas virtuais criei sessões ssh no MobaXterm utilizando as chaves privadas e os IP's das máquinas virtuais criadas de forma a poder trocar entre as mesmas mais facilmente.

## 6.4 Instalação das ferramentas cliente

Para instalar as ferramentas cliente comecei por gerar um par de chaves no node master1 para possibilitar a conexão deste aos outros nodes. Depois de criar esta chave copiei-a e adicionei-a ao ficheiro `/.ssh/authorized_keys`. E testei a ligação ssh do master1 para todos os outros nós.

De seguida fiz download do ficheiro Kubectl e configurei-o como executável.

## 6.5 Comunicação no cluster com TLS

Para a criação de certificados e assinatura fidedigna dos mesmos, primeiro criei uma CA (certificate authority).

Nesta secção todos os certificados seguem a mesma metodologia:

1. Gerar a chave privada
2. Gerar o pedido de assinatura de certificado
3. Assinar o certificado

Estes pontos foram feitos para:

- **Certificado de Administrador** - Utilizado por administradores para autenticar-se no servidor da API do Kubernetes, permitindo a realização de tarefas administrativas como gestão de recursos, implementação de aplicações e configuração de definições do cluster.
- **Certificado de Proxy** - Utilizado pelo proxy do Kubernetes para autenticar-se no servidor da API do Kubernetes. O proxy do Kubernetes é um proxy de rede que mantém regras de rede nos nós. Este certificado é necessário para autenticar o proxy, possibilitando o acesso a informações do cluster e a capacidade de atualizar regras dinamicamente.
- **Certificado do Servidor da API** - Utilizado pelo servidor da API do Kubernetes para autenticar-se perante outros componentes e clientes. Outros componentes precisam verificar a identidade do servidor da API para garantir que estão a comunicar com o plano de controlo legítimo. Além disso, os clientes precisam garantir que estão a interagir com o servidor da API autorizado.
- **Certificado do ETCD** - Utilizado pelo servidor ETCD, um armazenamento distribuído de pares chave-valor para dados do cluster. O ETCD armazena dados críticos do cluster. A autenticação assegura que apenas entidades autorizadas podem comunicar com ele e efetuar alterações, mantendo a segurança dos dados do cluster.
- **Certificado do Par de Chaves do Gestor de Controladores + cliente** - Utilizado pelo gestor de controladores do Kubernetes para tarefas relacionadas com contas de serviço. Este certificado é utilizado para autenticar o gestor de controladores perante vários controladores.

- **Certificado do Kubelet** - Utilizado pelo kubelet nos nós de trabalho para autenticar-se no servidor da API do Kubernetes. Eles utilizam o certificado para comprovar a sua identidade ao servidor da API, permitindo-lhes juntar-se ao cluster e reportar o estado do nó de forma verídica.

A realização destes passos para cada um dos certificados descritos possibilita a comunicação segura e autenticação dentro do cluster. Contribuem para garantir que apenas entidades autorizadas podem interagir com os componente dentro do mesmo.

No final da criação deste certificados, os mesmo devem ser redistribuídos para os vários nodes correspondentes.

## 6.6 Ficheiros de configuração Kube

De seguida criei os ficheiros kubeconfig que são utilizados pelos clientes para aceder ao servidor API do Kubernetes. O utilizador administrador, gestor de controladores, scheduler e kube proxy atuam como clientes do servidor API do Kubernetes, utilizando esses ficheiros para comunicação. Componentes residentes nos nós principais, como o gestor de controladores e o scheduler, conseguem aceder diretamente ao servidor através da interface de loopback. Por outro lado, entidades fora do nó principal, como o utilizador administrador e o kube proxy, acedem ao servidor API através do balanceador de carga.

Depois de criados, redistribui os ficheiros de configuração do kube-proxy para os worker nodes e os ficheiros de configuração do utilizador admin, kube controller e scheduler para o os outros master nodes.

## 6.7 Encriptação de Dados

O próximo passo é criar um ficheiro de configuração de encriptação e a chave que utilizei para armazenar dados encriptados sobre o cluster, aplicações e passwords no armazenamento de dados ETCD. Isso proporciona uma forma de personalizar como a encriptação é tratada para determinados tipos de recursos. Depois de criado este ficheiro deve ser redistribuído para os outros masters.

## 6.8 Os binaries mais recentes do kubernetes

Os binários do Kubernetes são os ficheiros executáveis que constituem os componentes do Kubernetes que discutimos anteriormente (kube-apiserver, kube-controller-manager, kube-scheduler, kubelet, kube-proxy, etcd, kubectl). Manter estes binários do Kubernetes atualizados garante que posso beneficiar de todas as novas funcionalidades, atualizações de segurança, melhorias de compatibilidade e um suporte mais alargado pela comunidade.

Primeiro, naveguei até à página de lançamento dos binaries mais recentes do Kubernetes:: [kubernetes-latest-releases](#). Uma vez lá, localizei e selecionei a atualização "Downloads for v..." para a versão mais recente.

Copiei o link de download do ficheiro "kubernetes.tar.gz" e fiz o download do mesmo no master1. Após o download, extraí a pasta. Dentro da pasta extraída, executei o processo de instalação. Atenção que a diretoria do cliente já está extraída, mas a diretoria do servidor não está. Portanto, é necessário extrair a diretoria do servidor separadamente após instalar todos os binaries.

## 6.9 Componentes do Master

Nesta secção realizei o download da versão mais recente do ETCD. Esta versão pode ser encontrada em [ETCD-último-lançamento](#). Depois, extraí os binaries do ETCD e configurei o servidor ETCD criando a sua pasta e movendo os ficheiros de certificados e chaves para essa mesma pasta. Configurei este serviço com os IP's internos das máquinas do cluster. E por fim reiniciei o serviço.

## 6.10 Controll plane

Para instalar os componentes do controll plane (apiserver, kube-controller-manager, scheduler, kubectl) comecei fazer download dos mesmos, tornei-os executáveis e copiei-os para a diretoria bin. De seguida configurei o servidor API colocando as chaves na diretoria de dados do kubernetes. O endereço IP será utilizado para partilhar o servidor API com os outros nodes do cluster. O endereço IP do load balancer é utilizado como endpoint para o servidor API. Por fim criei o ficheiro de unidade de sistema do servidor API.

Para o controller manager movi o ficheiros de configuração do mesmo para o sitio correto e criei a unidade de sistema do controller manager.

Para o scheduler realizei o mesmo que o controller manager. Movi os ficheiros de configuração e criei o ficheiro de unidade de sistema.

No final reiniciei todos os sistemas modificados.

## 6.11 Load-Balancer

Nesta secção provisiono o node load balancer exterior para que fique na frente do servidor API do kubernetes.

O load balancer opera na camada 4 (TCP) o que significa que o tráfego passa pelo back-end dos servidores sem ser revisto e não interfere com o processo de TLS, esta interferência é deixada para o servidor API.

No terminal do load balancer:

- Ler os Ip's dos master nodes e do próprio load balancer para variáveis ambiente.
- Criar uma configuração HAProxy para escutar no port do servidor API no proprio host e distribuir os pedidos para os vários masters.
- Fazer um pedido http para verificar a versão do kubernetes verificando assim a distribuição a funcionar.

## 6.12 Worker nodes

Para fins de teste dos worker nodes, utilizei duas abordagens distintas para configuração. Inicialmente, optei por gerar certificados, tê-los assinados pela Autoridade de Certificação (CA) e depois transferi-los para os worker nodes. Subsequentemente, configurei o serviço kubelet para utilizar esses certificados. Para lidar com a expiração dos certificados, estes são renovados manualmente conforme necessário. No entanto, em um cluster com um número considerável de nós, esse processo de renovação manual torna-se impraticável. Portanto, o objetivo era capacitar os nós a gerir autonomamente os seus certificados. Para o segundo nó de trabalho, implementei uma abordagem de inicialização TLS, capacitando-o a criar, assinar, usar e renovar certificados de forma independente para fins de automação. No final, a configuração do kubeproxy nos nós seguiu o mesmo procedimento da abordagem inicial.

### 6.12.1 Worker 1:

No master1 gerei os certificados necessários para o kubelet: O Kubernetes utiliza um "modo de autorização especial" que autoriza especificamente os pedidos de API feitos pelo kubelet se este estiver autorizado e utilizar as credenciais que o identificam como pertencente ao grupo `system_nodes` com um determinado nome de utilizador.

Para gerar os ficheiros de configuração do kubelet, primeiro obti o endereço IP do load balancer (frente do servidor API) e coloquei-o numa variável ambiente. De seguida, gerei o ficheiro de configuração kube para o primeiro worker node.

Copiei os certificados, as chaves privadas e os ficheiros de configuração kube para os worker nodes.

No Worker1 fiz o download dos binaries do worker (kube-proxy e kubelet) criei as suas diretorias e instalei-os nas mesmas. De seguida configurei o kubelet copiando as chaves e as configurações para as diretorias corretas e protegendo-as. Criei uma variável ambiente com os intervalos de CIDR usados dentro do cluster outra com o endereço DNS do cluster e criei o ficheiro de configuração do kubelet, finalmente, criei o ficheiro de unidade de serviço do kubelet.

Para o kube proxy segui exatamente a mesma metodologia correspondente ao mesmo.

No final reiniciei ambos os serviços no worker1.



### **6.12.2 Worker 2:**

TLS bootstrapping permite que os nodes façam um auto gestão de forma automática dos pares de chaves e solicitações de assinatura de certificado. Posteriormente permite que os nodes enviem os certificados gerados para a CA e que obtenham os mesmo assinados. Usando esses certificados os nodes podem gerar automaticamente ficheiros de configuração kube e ingressar no cluster.

Para iniciar esta funcionalidade primeiro criei um token bootstrap que será usado para invocar a API de certificados e autorizei os workers a criarem, assinarem e renovarem os seus próprios pedidos de assinatura de certificados.

Utilizei a mesma metodologia do worker1 para instalar as binaries referentes ao worker2.

Só depois finalmente configurei o segundo worker para executar TLS bootstrap utilizando o token gerado anteriormente. Para isso criei o ficheiro de configuração de bootstrap com a informação sobre o token.

Por fim reiniciei os serviços do worker2.

Agora no master1 posso encontrar os certificados pendentes, aprova-los e verificar a lista de nodes que fazem parte do cluster.

## **6.13 Kubectl**

Para configurar o acesso remoto ao kubectl comecei por atribuir um servidor API (IP da frente do load balancer) a cada ficheiro de configuração kube do desse mesmo serviço que posteriormente criei.

## **6.14 Pod Networking**

Para administrar as redes que irão ligar os containers entre os vários nodes utilizei uma ferramenta chamada weave network como a minha CNI (container network interface).

## **6.15 RBAC permissions**

Permissões RBAC é aquilo que permite que o servidor API consiga conectar-se ao kubelet em cada node. Esta conexão é necessária para adquirir métricas, logs e executar comandos nos pods.

Para isso criamos um grupo que conta com uma regra de grupo do cluster que permite o acesso ao servidor API. Depois conectei o utilizador do api server ao grupo criado para este se autenticar com sucesso ao kubelet em cada worker node e vice-versa.

## 6.16 DNS

Nesta secção implementei o servidor dns suportado por uma ferramenta do kubernetes chamada de CoreDNS. Este servidor Dns é na verdade já implementado no cluster como um serviço (pod). De seguida criei outro pod com o serviço de BusyBox que me permite executar vários comandos de rede conhecidos como nslookup, dig... Estes comandos permitem-me testar a funcionalidade do CoreDNS.

## 6.17 Smoke Test

Nesta secção realizei alguns testes para verificar a funcionalidade do cluster como:

- Criar um secret generico
- Fazer print de um secret do etcd
- Criar um deployment para um servidor web nginx
- Aceder a aplicações remotamente utilizando port forwarding
- Recuperar e visualizar logs dos containers.
- Realizar comandos dentro de um container.

## 7 TESTES DO PROJETO

### 7.1 MariaDB

#### 7.1.1 O que é?

O MariaDB é um sistema de gestão de bases de dados relacionais (SGBDR) de código aberto altamente compatível com o MySQL.

#### 7.1.2 Como o implementei no meu cluster?

Após estudar este serviço, cheguei à conclusão de que a sua implementação no meu cluster envolveu a criação de vários componentes essenciais. Abaixo estão breves descrições de cada um deles:

- **MariaDB ConfigMap** - Este ConfigMap é utilizado para armazenar a configuração do MariaDB. Permite a fácil modificação das configurações sem a necessidade de alterar o ficheiro de configuração diretamente no StatefulSet.
- **MariaDB Persistent Volume** - Este Persistent Volume é usado para fornecer um armazenamento persistente de dados ao MariaDB. Mapeia um local de armazenamento físico, garantindo que os dados do MariaDB sejam guardados mesmo quando os pods são reiniciados.
- **MariaDB Persistent Volume Claim** - Este Persistent Volume Claim é uma solicitação para armazenamento persistente. O PVC associa-se ao PV e é utilizado pelo MariaDB para garantir o acesso ao armazenamento persistente.
- **MariaDB StorageClass** - Este StorageClass define as propriedades do armazenamento utilizado pelo PV. Especifica o tipo de provisionamento de armazenamento, como SSD ou HDD, e outras características importantes.
- **MariaDB StatefulSet** - O StatefulSet é como um deployment mas para serviços com estados como é o caso de bases de dados. Este é responsável por manter a identidade de cada pod MariaDB. Garante que mesmo durante a modificação da escala de cada instância, estas mantenham um identificador único e persistente.
- **MariaDB NodePort Service** - Este serviço expõe o MariaDB para fora do cluster, permitindo o acesso externo através de um número de porta fixo em todos os nós do cluster.

Depois da criação destes vários ficheiros yaml procedi à sua aplicação através do comando "kubectl apply -f" e para me certificar que este serviço está operacional utilizei o comando "mysql -u root -p -h <ipDeUmWorkerNode> -P <PortDoServicoMariaDB>".

```
vagrant@master-1:~$ mysql -u root -p -h 192.168.56.21 -P 32049
Enter password:
```

Figura 7.1: MariaDB Login

A password a ser utilizada é definida dentro do ficheiro de configuração do statefulset do mariadb.

```
env:
- name: MYSQL_ROOT_PASSWORD
  value: your-root-password
```

Figura 7.2: MariaDB Password

Este comando utiliza uma interface MySQL para visualizar e editar as bases de dados e seus conteúdos dentro do serviço mariaDB lançado.

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| nextcloud |
| performance_schema |
| sys |
| your-database-name |
+-----+
6 rows in set (0.02 sec)
```

Figura 7.3: MySQL interface

### 7.1.3 Dificuldades

Sendo este o primeiro deployment completo no kubernetes realizado por mim senti algumas dificuldades principalmente em perceber como utilizar os pv's, pvc's e storage class em conjunto com o deployment em si. O facto de não conseguir aceder a este serviço através de um browser fez com que também tivesse dificuldades em testa-lo pois, no inicio, pensava que era assim que se testava todos os serviços. Só mais tarde cheguei à conclusão que poderia utilizar a interface MySQL para aceder à mariaDB.

## 7.2 REDIS

### 7.2.1 O que é?

O Redis é uma base de dados em memória de código aberto, utilizado como banco de dados, cache e message broker. A sua estrutura de dados em memória e sua capacidade de processar operações rapidamente torna-o uma escolha popular para aplicações que requerem acesso rápido e eficiente aos dados.

### 7.2.2 Como o implementei no meu cluster?

Para aplicar este serviço no meu cluster utilizei os seguintes ficheiros de configuração:

- **REDIS Deployment** - O Deployment do Redis é responsável por gerir a criação e escalabilidade dos POD's Redis no cluster. Define o estado desejado e garante que o número correto de pods esteja em execução para atender à demanda.
- **REDIS Service** - O serviço do Redis expõe os POD's Redis para fora do cluster, permitindo o acesso externo. Utiliza um número de porta fixo em todos os nós do cluster, facilitando a comunicação com o Redis.
- **REDIS ClusterIP** - O ClusterIP é um serviço interno que fornece um IP estático para os POD's Redis dentro do cluster. Isso permite que outros POD's internos comuniquem com o Redis usando este IP interno fixo, proporcionando uma comunicação eficiente entre os serviços internos do Kubernetes.

Depois da sua criação e aplicação testei o serviço utilizando o comando "kubectl exec -it <NomePodDoRedis> -- sh -c 'redis-cli -h <NomeDoServiçoInternoRedis> -p <PortServiçoInternoRedis> PING'"Este comando basicamente envia para dentro do POD Redis que está a correr o comando redis-cli PING que é um comando da interface redis que devolve PONG se a mesma estiver ativa, o que demonstra a operacionalidade deste POD.

```
vagrant@master-1:~$ kubectl exec -it redis-deployment-65d8857849-qzjsj -- sh -c 'redis-cli -h redis-clusterip-service -p 6379 PING'
PONG
```

Figura 7.4: Redis PING

## 7.3 MinIO

### 7.3.1 O que é?

O MinIO é um serviço de armazenamento dados em objetos, compatível com a API S3 da Amazon. É frequentemente utilizado para armazenar grandes quantidades de dados não estruturados, como objetos e arquivos.

### 7.3.2 Como o implementei no meu cluster?

A implementação do MinIO no meu cluster envolveu a criação de vários componentes importantes:

- **MinIO Deployment** - O Deployment do MinIO é responsável por gerir a criação e escalabilidade dos POD's MinIO no cluster. Define o estado desejado e garante que o número correto de POD's esteja em execução para atender à demanda.
- **Minio NodePort Service** - O serviço NodePort do MinIO expõe os POD's MinIO para fora do cluster, permitindo o acesso externo através de um número de porta fixo em todos os nós do cluster. Este serviço é utilizado para interagir com o Minio e realizar operações de armazenamento de objetos.
- **Minio Persistent Volume** - O Persistent Volume do Minio é responsável por fornecer armazenamento persistente ao Minio. Mapeia um local de armazenamento físico, garantindo que os dados do Minio sejam retidos mesmo quando os POD's são reiniciados.
- **Minio Persistent Volume Claim (PVC)** - O Persistent Volume Claim do Minio é uma solicitação para armazenamento persistente. O PVC associa-se ao PV e é utilizado pelo Minio para garantir o acesso ao armazenamento persistente necessário.

Para fins de teste deste serviço utilizei o comando "kubectl exec -it <NomePodMinIO> -- sh -c 'mc alias set myminio http://<NomeServiçoMinIO>:<PortInternoServiçoMinIO> <MinioRootUserName> <MinioRootPassword> && mc ls myminio/'"

Os valores de Root Username e RootPassword são definidos no ficheiro de configuração do deployment MinIO.

Este comando configura um alias chamado "myminio" para conectar ao servidor MinIO. O comando "mc ls myminio/" lista os objetos no bucket padrão do MinIO associado ao alias.

```
env:
- name: MINIO_ROOT_USER
  value: your-access-key
- name: MINIO_ROOT_PASSWORD
  value: your-secret-key
```

Figura 7.5: MinIO Credentials

```
vagrant@master-1:~$ kubectl exec -it minio-deployment-5f95b5b489-b55wl -- sh -c 'mc alias set myminio http://minio-service:9000 your-access-key your-secret-key && mc ls myminio/'
Added myminio successfully
vagrant@master-1:~$ mc ls myminio/
```

Figura 7.6: MinIO Test

## 7.4 MariaDB VS MinIO VS REDIS

Apesar de estes vários serviços serem todos considerados bases de dados, não deixam de ser diferentes e ter *usecases* distintos. A mariaDB é um sistema de base de dados relacional que

armazena dados estruturados e relações complexas utilizada em aplicações que requerem uma estrutura de dados altamente organizada e utiliza volumes em memória física para guardar os dados. O MinIO é um sistema de base de dados que armazena objetos/buckets não estruturados, é altamente escalável, guarda grandes volumes de dados e é utilizado para armazenar arquivos, imagens, arquiteturas de armazenamento distribuído, entre outros. O REDIS é um tipo de base de dados não estruturados (Chave-valor), armazenados em memória, comumente utilizado para cache e armazenamento temporário.

### 7.4.1 Comparação Geral

Tecnologia	Desempenho	Escalabilidade	Dificuldade Configuração	Uso de Recursos
MariaDB	2º	2º	1º	3º
MinIO	3º	1º	2º	2º
REDIS	1º	3º	3º	1º

Tabela 7.1: Leaderboard de Avaliação Geral

Note que estas classificações são uma perspectiva pessoal destes serviços de bases de dados daí a complexidade de configuração da mariaDB estar em primeiro sendo que foi o primeiro serviço implantado por mim com um valor de conhecimentos mais baixo do que quando implementei o MinIO por exemplo.

## 7.5 Wordpress

### 7.5.1 O que é?

O WordPress é uma plataforma que gere conteúdo páginas php, amplamente utilizada para criar blogs e sites. Utiliza um sistema de base de dados para armazenar conteúdo, configurações e outros dados importantes.

### 7.5.2 Como o implementei no meu cluster?

A implementação do WordPress no meu cluster foi realizada de maneira simplificada, utilizando imagens Docker prontas disponíveis publicamente. Os seguintes comandos foram utilizados para criar o Deployment e expor o WordPress através de um serviço NodePort:

- **kubectl create deployment wordpress --image=wordpress:latest** - Este comando cria um Deployment chamado "wordpress" usando a imagem Docker oficial do WordPress disponível no repositório público. O Deployment gere a criação e escalabilidade dos pods WordPress no cluster.
- **kubectl expose deployment wordpress --type=NodePort --port=80** - Este comando expõe o serviço WordPress para fora do cluster através de um serviço NodePort. O serviço utiliza o número de porta 80 para permitir o acesso externo ao WordPress.

O WordPress, ao ser iniciado, automaticamente detecta a presença de uma base de dados e inicia o processo de configuração. Neste caso, a integração com o serviço MariaDB foi estabelecida devido à definição padrão do WordPress para procurar uma base de dados MySQL.

Essa integração é possível porque ambos o WordPress e a MariaDB seguem as práticas padrão de utilização de variáveis de ambiente para configurar a conexão com a base de dados. O WordPress deteta as variáveis de ambiente relacionadas ao MySQL/MariaDB e configura automaticamente a sua conexão.

### 7.5.3 Teste

Após a implementação acedi à pagina wordpress utilizando o serviço correspondente. Neste primeiro acesso a esse serviço é mostrada uma página de configuração inicial do wordpress.

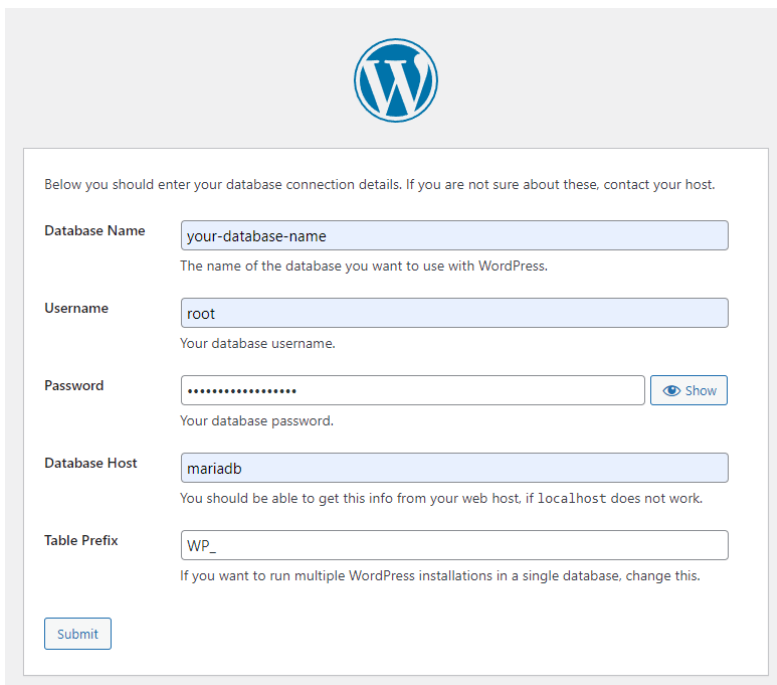
The image shows the WordPress installation configuration screen. At the top is the WordPress logo. Below it, a message says: "Below you should enter your database connection details. If you are not sure about these, contact your host." The form contains five input fields: "Database Name" with the placeholder "your-database-name", "Username" with the value "root", "Password" with masked characters and a "Show" button, "Database Host" with the value "mariadb", and "Table Prefix" with the value "WP\_". Each field has a descriptive subtitle. At the bottom left is a "Submit" button.

Figura 7.7: Wordpress Initial config

Nesta configuração escolhe-se uma base de dados dentro da MariaDB para utilizar, um utilizador com permissões para aceder à base de dados da mariaDB e a sua password correspondentes (definidos no deployment da mariaDB), o host da database é o nome do serviço que expõe a mariaDB, e o prefixo é um valor personalizável que irá ser colocado antes da tabela para utilização e identificação.

Após esta configuração inicial é inicializada uma configuração do site a criar.

Nesta configuração é definido o nome do site, o username para voltar a aceder à dashboard que pode modificar este site mais tarde, a password correspondente, e um mail.

No final da configuração basta fazer login e começar a criar!



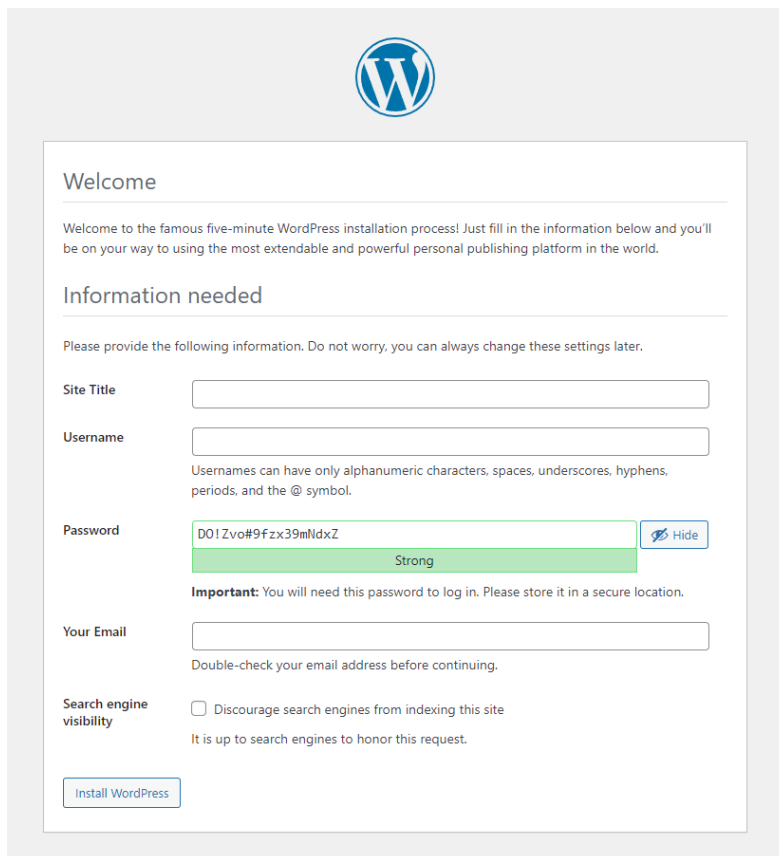


Figura 7.8: Wordpress Site config

## 7.6 ModSecurity

### 7.6.1 O que é?

O ModSecurity é uma firewall de aplicações web (WAF), desenvolvido para proteger aplicações web contra diversas ameaças, incluindo injeção de SQL, cross-site scripting (XSS), entre outros ataques. Atua como uma camada de segurança entre as aplicações web e a Internet, filtrando e monitorizando o tráfego HTTP.

### 7.6.2 Instalação Direta no Nó

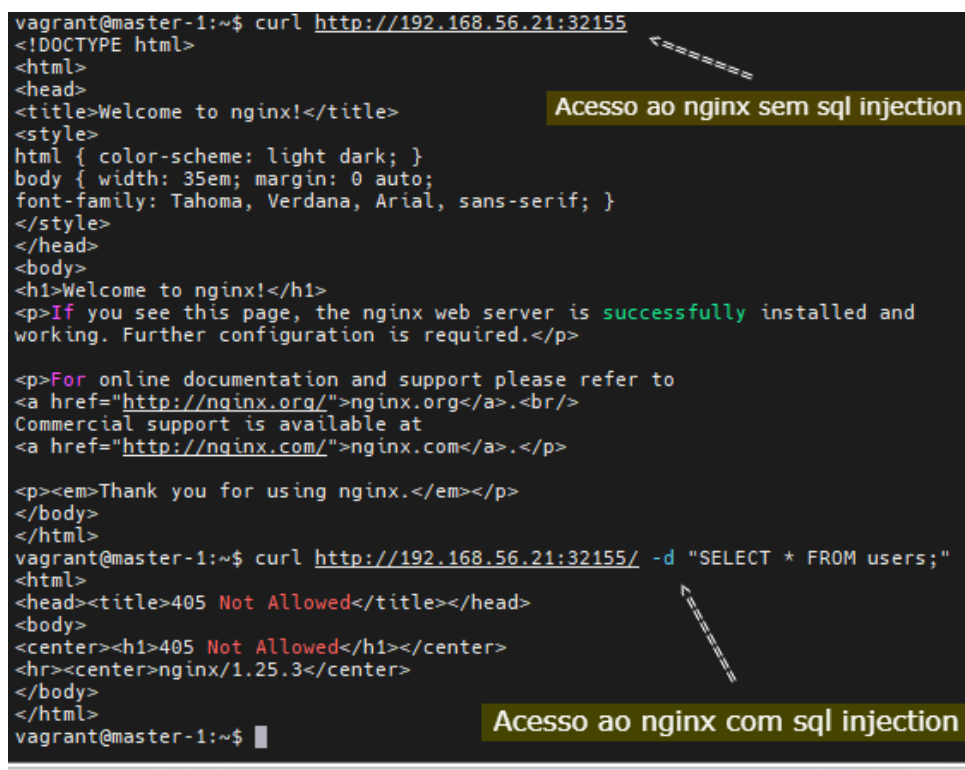
Para melhorar a segurança das aplicações em execução no meu cluster Kubernetes, optei por instalar e configurar o ModSecurity diretamente nos nós do cluster. A seguir estão os passos gerais que segui para realizar essa instalação:

- **Download e Compilação:** Baixei o código-fonte do ModSecurity do repositório SpiderLabs/ModSecurity-nginx e compilei-o no nó do Kubernetes. Os diretórios e arquivos resultantes estão listados abaixo:
- **Configuração:** Após a compilação, realizei a configuração do ModSecurity conforme necessário. O ModSecurity vem com um arquivo de utilização recomendado mas para uma

questão de testes utilizei um mais pequeno criado por mim aplicando-o posteriormente.

É importante notar que esta abordagem é uma escolha específica para o meu ambiente e requisitos de segurança. Para ambientes distintos pode ser necessário uma configuração de ModSecurity adequada.

Para testar esta nova 'firewall' decidi dar deploy de um serviço nginx que é de todos o mais simples e baseia-se numa página web básica pré-definida. Após dar deploy da mesma utilizei o comando "curl http://<IpDeUmWorkerNode>:<portDoServiçoNginx>/ -d "SELECT \* FROM users;" este comando acede ao site nginx mas com parâmetros do tipo SQL. Este método de acesso é considerado uma falha de segurança do tipo SQL injection, que, graças ao ModSecurity é bloqueada.



```
vagrant@master-1:~$ curl http://192.168.56.21:32155
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
vagrant@master-1:~$ curl http://192.168.56.21:32155/ -d "SELECT * FROM users;"
<html>
<head><title>405 Not Allowed</title></head>
<body>
<center><h1>405 Not Allowed</h1></center>
<hr><center>nginx/1.25.3</center>
</body>
</html>
vagrant@master-1:~$
```

Acesso ao nginx sem sql injection

Acesso ao nginx com sql injection

Figura 7.9: ModSecurity Test

## 7.7 Gitlab

### 7.7.1 O que é?

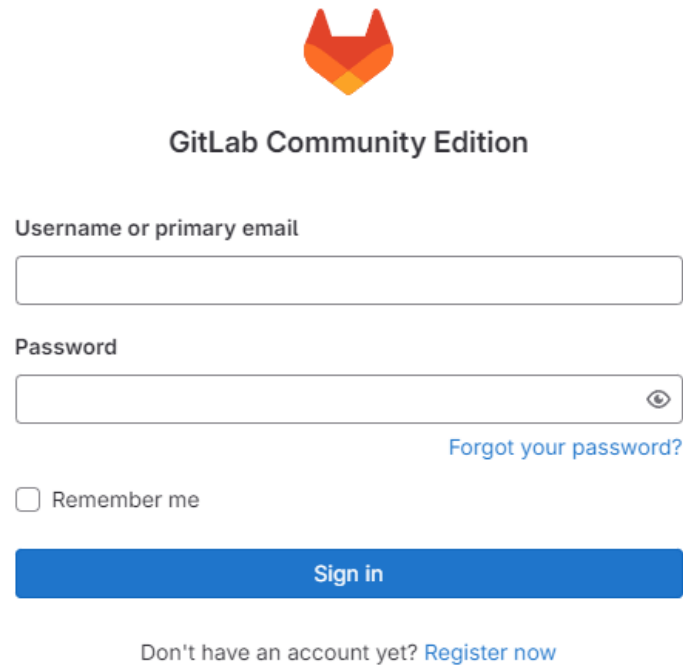
O GitLab é uma plataforma completa de DevOps que oferece um repositório Git, gestão de CI/CD (Integração Contínua/Entrega Contínua), controle de versão, sondagem de problemas, entre outros recursos. Permite a colaboração eficiente de uma equipa de desenvolvimento em projetos de software.

### 7.7.2 Implementação no Cluster Kubernetes

A implementação do GitLab no meu cluster Kubernetes envolveu a criação de vários componentes essenciais usando arquivos YAML. A seguir estão os YAMLs utilizados:

- **gitlab-deployment.yaml** - Este arquivo descreve o Deployment do GitLab, especificando detalhes como a imagem Docker a ser usada, as variáveis de ambiente necessárias e outras configurações específicas do GitLab.
- **gitlab-pvc.yaml** - O Persistent Volume Claim é utilizado para solicitar armazenamento persistente para o GitLab. Este arquivo YAML descreve as características do armazenamento necessário para o GitLab operar corretamente.
- **gitlab-service.yaml** - Este arquivo descreve o Service do GitLab, que expõe o Deployment. Define como o GitLab pode ser acessado externamente.
- **gitlab-volume.yaml** - Descreve um Volume persistente que será usado pelo GitLab para armazenar dados de forma persistente. Este volume é utilizado pelo PVC mencionado anteriormente.

Esses arquivos YAMLs foram aplicados no cluster Kubernetes usando o comando 'kubectl apply -f'. Após a aplicação bem-sucedida, o GitLab ficou disponível através de um browser utilizando o ip de um worker node do cluster e o PORT correspondente ao serviço criado.



The image shows the GitLab Community Edition login interface. At the top is the GitLab logo (a red and orange flame-like shape) and the text "GitLab Community Edition". Below this are two input fields: "Username or primary email" and "Password". The password field has a toggle icon (an eye) on the right. Below the password field is a link "Forgot your password?". There is a checkbox labeled "Remember me". At the bottom is a blue button labeled "Sign in". Below the button is a link "Don't have an account yet? Register now".

Figura 7.10: Gitlab Login

## 7.8 Nextcloud

### 7.8.1 O que é?

O Nextcloud é uma plataforma de colaboração em nuvem de código aberto que oferece serviços como armazenamento de arquivos, partilha, sincronização e colaboração. Fornece uma solução segura e escalável para armazenamento e gestão de dados na nuvem.

### 7.8.2 Como o implementei no meu cluster?

A implementação do Nextcloud no meu cluster Kubernetes envolveu a criação de vários componentes essenciais. A seguir estão breves descrições de cada um deles:

- **Nextcloud Deployment** - O Deployment do Nextcloud gere a criação e escalabilidade dos pods Nextcloud no cluster. Define o estado desejado e garante que o número correto de pods esteja em execução para atender à demanda. Nesta configuração também é definido onde é o banco de dados a ser utilizado pelo nextcloud, no meu caso foi utilizado novamente o mariaDB.
- **Nextcloud Persistent Volume** - O Persistent Volume do Nextcloud fornece armazenamento persistente para os dados do Nextcloud. Mapeia um local de armazenamento físico, garantindo que os dados sejam retidos mesmo durante reinicializações dos pods.
- **Nextcloud Persistent Volume Claim** - O PVC é uma solicitação para armazenamento

persistente associada ao PV. O Nextcloud utiliza o PVC para garantir acesso ao armazenamento persistente necessário.

- **Nextcloud Secret** - O Secret do Nextcloud armazena informações sensíveis, como senhas e chaves de acesso, de forma segura. É utilizado para configurar o acesso seguro aos serviços externos, como a base de dados.
- **Nextcloud Service** - O serviço Nextcloud expõe os pods Nextcloud para fora do cluster, permitindo o acesso externo.

Para testar este serviço basta aceder a um browser e aceder através do IP do worker node onde está a correr o gitlab e o PORT do serviço que expõe o nextcloud definido.

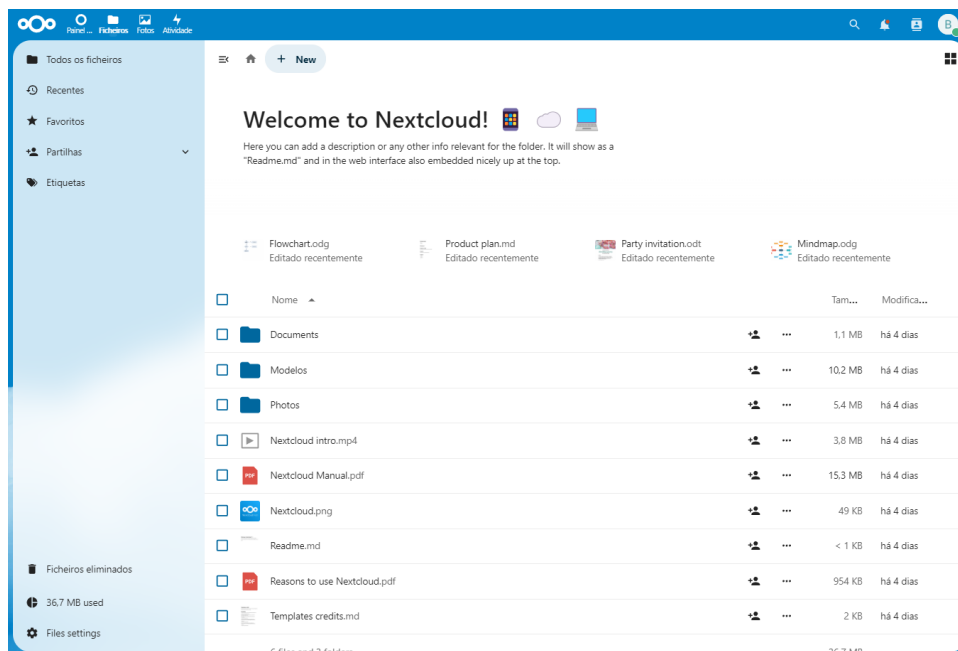


Figura 7.11: Nextcloud Test

## 7.9 Prometheus & Grafana

### 7.9.1 O que são?

Prometheus é um sistema de monitorização e alerta projetado para monitorizar serviços e aplicações em um ambiente container. Grafana, por outro lado, é uma plataforma de análise e visualização que funciona em conjunto com sistemas de monitorização, como o Prometheus.

### 7.9.2 Como implementei no meu cluster?

A implementação do Prometheus e Grafana no meu cluster Kubernetes foi realizada usando o Helm Chart fornecido pela comunidade. Aqui estão os passos gerais que segui para configurar a monitorização:

- **Pré-requisitos:** Primeiro instalei o Helm no meu ambiente local.
- **Setup:** Depois acedi ao site de monitorização de kubernetes do grafana. Lá dentro criei um conta com o meu email. Ativei essa mesma conta através do mail e fiz login. Uma vez lá dentro basta pressionar o botão "Get started in Grafana Cloud" que redireciona para a página de gestão de stacks grafana. Localizei a stack "Grafana" e pressionei o botão "Launch". Uma vez dentro da interface grafana basta carregar no botão "+ Connect data", seleccionar "kubernetes Monitoring" e "start sending data". Na primeira página "settings" alguns valores já estão preenchidos por default basta instalar as regras da dashboard, dos alertas e do "recording". Uma vez instaladas segui para o separador "Cluster configuration" onde preenchi os campos com a informação do meu cluster selecionei kubernetes, criei um novo token de acesso (esta token deve ser guardado). Depois de preencher as informações este separador irá gerar a configuração necessária a implementar no cluster, basta copiar e colar.
- **Configuração:** Depois de copiada a configuração gerada pela página anterior para o cluster o mesmo irá dar setup a todos os serviços necessários para a monitorização. De volta na página existe uma hiperligação no texto "cluster status" onde podemos verificar se os vários serviços foram corretamente inicializados.
- **Visualização:** De volta na home page basta seleccionar "Observability" e depois "Kubernetes" para visualizar o estado do cluster. Dentro desta página existem vários monitores de controle interessantes como a eficácia do cluster, o custo...

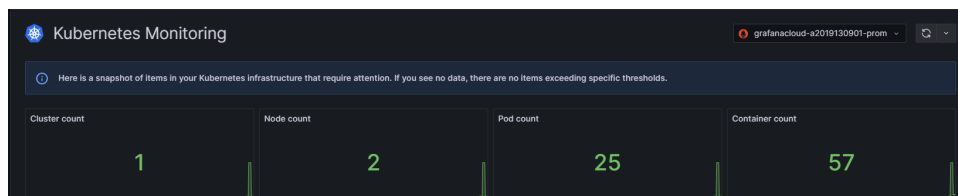


Figura 7.12: Grafana dashboard

## **8 CONCLUSÃO**

O estágio proporcionou uma experiência valiosa no desenvolvimento e implementação de micro-serviços e aplicações em cluster Kubernetes, contribuindo para o aprimoramento das minhas habilidades técnicas e a preparação para desafios futuros na área de redes e administração de sistemas.

## REFERÊNCIAS BIBLIOGRÁFICAS

- [1] T. with Nana, “Kubernetes tutorial for beginners,” <https://www.youtube.com/watch?v=X48VuDVv0do>, 2022, accessed on 2024-10-08. [Online]. Available: <https://www.youtube.com/watch?v=X48VuDVv0do>
- [2] KubeKloud, “Install kubernetes from scratch,” [https://www.youtube.com/playlist?list=PL2We04F3Y\\_41jYdadX55fdJplDvgNGENo](https://www.youtube.com/playlist?list=PL2We04F3Y_41jYdadX55fdJplDvgNGENo), 2021, accessed on 2024-11-15. [Online]. Available: [https://www.youtube.com/playlist?list=PL2We04F3Y\\_41jYdadX55fdJplDvgNGENo](https://www.youtube.com/playlist?list=PL2We04F3Y_41jYdadX55fdJplDvgNGENo)
- [3] OpenAI, “Chatgpt,” <https://chat.openai.com/>, 2020, accessed During Internship. [Online]. Available: <https://chat.openai.com/>
- [4] Docker, “Docker,” <https://docs.docker.com/get-started/>, 2013-2024, accessed During Internship. [Online]. Available: <https://docs.docker.com/get-started/>
- [5] G. J. P. . R. P. Goldberg, “Formal requirements for virtualizable third generation architectures,” <https://dl.acm.org/doi/10.1145/361011.361073>, 01 July 1974, accessed on 2024-10-17. [Online]. Available: <https://dl.acm.org/doi/10.1145/361011.361073>

(1) (2) (3) (4) (5)



## **ANEXOS**

## Anexo A - mariaDB yaml files

- apiVersion: v1 kind: ConfigMap metadata: name: mariadb-config data: my.cnf: | [mysqld]  
bind-address=0.0.0.0
- apiVersion: v1 kind: PersistentVolume metadata: name: mariadb-pv spec: capacity: storage:  
1Gi accessModes: - ReadWriteOnce hostPath: path: /path/on/your/host
- apiVersion: v1 kind: PersistentVolumeClaim metadata: name: mariadb-pvc spec: storage-  
ClassName: mariadb-storageclass accessModes: - ReadWriteOnce resources: requests: storage:  
1Gi
- apiVersion: v1 kind: Service metadata: name: mariadb-loadbalancer spec: selector: app:  
mariadb ports: - protocol: TCP port: 3306 targetPort: 3306 type: LoadBalancer
- apiVersion: v1 kind: Service metadata: name: mariadb-nodeport spec: selector: app: mariadb  
ports: - protocol: TCP port: 3306 targetPort: 3306 type: NodePort
- apiVersion: apps/v1 kind: StatefulSet metadata: name: mariadb spec: serviceName: "mari-  
adb" replicas: 1 selector: matchLabels: app: mariadb template: metadata: labels: app: mariadb  
spec: containers: - name: mariadb image: mariadb:latest env: - name: MYSQL\_ROOT\_PASSWORD  
value: your-root-password - name: MYSQL\_DATABASE value: your-database-name - name:  
MYSQL\_USER value: your-username - name: MYSQL\_PASSWORD value: your-user-  
password ports: - containerPort: 3306 volumeMounts: - name: mariadb-storage mountPath:  
/var/lib/mysql volumes: - name: mariadb-storage persistentVolumeClaim: claimName: mariadb-  
pvc — apiVersion: v1 kind: PersistentVolumeClaim metadata: name: mariadb-pvc spec:  
accessModes: - ReadWriteOnce resources: requests: storage: 1Gi — apiVersion: v1 kind:  
Service metadata: name: mariadb spec: selector: app: mariadb ports: - protocol: TCP port:  
3306 clusterIP: None
- apiVersion: storage.k8s.io/v1 kind: StorageClass metadata: name: default provisioner:  
kubernetes.io/no-provisioner

## Anexo B - REDIS yaml files

```
apiVersion: apps/v1 kind: Deployment metadata: name: redis-deployment spec: replicas: 1
selector: matchLabels: app: redis template: metadata: labels: app: redis spec: containers: -
name: redis image: redis:latest ports: - containerPort: 6379 — apiVersion: v1 kind: Service
metadata: name: redis-service spec: selector: app: redis ports: - protocol: TCP port: 6379
targetPort: 6379 type: NodePort — apiVersion: v1 kind: Service metadata: name: redis-
clusterip-service spec: selector: app: redis ports: - protocol: TCP port: 6379 targetPort: 6379
type: ClusterIP
```

```
— apiVersion: v1 kind: Service metadata: name: redis-nodeport spec: selector: app: redis ports:
- protocol: TCP port: 6379 targetPort: 6379 type: NodePort
```

```
— apiVersion: v1 kind: Service metadata: name: redis-service spec: selector: app: redis ports: -
protocol: TCP port: 6379 targetPort: 6379
```

## Anexo C - minIO yaml files

```
apiVersion: apps/v1 kind: Deployment metadata: name: minio-deployment spec: replicas: 1
selector: matchLabels: app: minio template: metadata: labels: app: minio spec: containers: -
name: minio image: minio/minio:latest args: - server - /data env: - name: MINIO_ROOT_USER
value: your-access-key - name: MINIO_ROOT_PASSWORD value: your-secret-key ports:
- containerPort: 9000 — apiVersion: v1 kind: Service metadata: name: minio-service spec:
selector: app: minio ports: - protocol: TCP port: 9000 targetPort: 9000 type: NodePort —
apiVersion: v1 kind: PersistentVolumeClaim metadata: name: minio-pvc spec: accessModes: -
ReadWriteOnce resources: requests: storage: 5Gi — apiVersion: v1 kind: PersistentVolume
metadata: name: minio-pv spec: capacity: storage: 5Gi accessModes: - ReadWriteOnce
hostPath: path: /path/on/host — apiVersion: v1 kind: PersistentVolumeClaim metadata: name:
minio-pvc spec: accessModes: - ReadWriteOnce resources: requests: storage: 10Gi
```

## Anexo D - gitlab yaml files

```
apiVersion: apps/v1 kind: Deployment metadata: name: gitlab spec: replicas: 1 selector:
matchLabels: app: gitlab template: metadata: labels: app: gitlab spec: containers: - name:
gitlab image: gitlab/gitlab-ce:latest ports: - containerPort: 80 volumes: - name: gitlab-persistent-
storage persistentVolumeClaim: claimName: gitlab-pvc — apiVersion: v1 kind: Persistent-
VolumeClaim metadata: name: gitlab-pvc spec: accessModes: - ReadWriteOnce resources:
requests: storage: 1Gi — apiVersion: v1 kind: Service metadata: name: gitlab spec: selector:
app: gitlab ports: - protocol: TCP port: 8166 targetPort: 80 type: NodePort — apiVersion: v1
kind: PersistentVolume metadata: name: gitlab-pv spec: capacity: storage: 1Gi accessModes: -
ReadWriteOnce hostPath: path: "/path/on/host"
```

## Anexo E - nextcloud yaml files

```
apiVersion: apps/v1 kind: Deployment metadata: name: nextcloud spec: replicas: 1 selector:
matchLabels: app: nextcloud template: metadata: labels: app: nextcloud spec: containers: -
name: nextcloud image: nextcloud:latest ports: - containerPort: 80 env: - name: MYSQL_HOST
value: "mariadb- name: MYSQL_DATABASE value: "nextcloud- name: MYSQL_USER value:
"root- name: MYSQL_PASSWORD valueFrom: secretKeyRef: name: nextcloud-mysql-secret
key: password volumeMounts: - name: nextcloud-storage mountPath: /var/www/html volumes:
- name: nextcloud-storage persistentVolumeClaim: claimName: nextcloud-pvc — apiVersion:
v1 kind: PersistentVolume metadata: name: nextcloud-pv spec: capacity: storage: 1Gi ac-
cessModes: - ReadWriteOnce hostPath: path: "/path/to/your/storage— apiVersion: v1 kind:
PersistentVolumeClaim metadata: name: nextcloud-pvc spec: accessModes: - ReadWriteOnce
resources: requests: storage: 1Gi — apiVersion: v1 kind: Secret metadata: name: nextcloud-
mysql-secret type: Opaque data: password: eW91ci1yb290LXBhc3N3b3Jk — apiVersion:
v1 kind: Service metadata: name: nextcloud-service spec: selector: app: nextcloud ports: -
protocol: TCP port: 7555 targetPort: 80 type: NodePort
```

## Anexo F - Prometheus + Grafana yaml files

apiVersion: v1 kind: ConfigMap metadata: name: prometheus-config namespace: monitoring  
data: prometheus.yml: | global: scrape\_interval: 15s

scrape\_configs: - job\_name: 'kubernetes-apisservers' kubernetes\_sd\_configs: - role: endpoints  
scheme: https tls\_config: ca\_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt bearer\_token\_file: /var/run/secrets/kubernetes.io/serviceaccount/token relabel\_configs: - source\_labels: [\_\_meta\_kubernetes\_namespace, \_\_meta\_kubernetes\_service\_name, \_\_meta\_kubernetes\_endpoint\_port\_name] keepregex: default;kubernetes;https

- job\_name: 'kubernetes-nodes' scheme: https tls\_config: ca\_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt bearer\_token\_file: /var/run/secrets/kubernetes.io/serviceaccount/token kubernetes\_sd\_configs: - role: node relabel\_configs: - action: labelmap regex: \_\_meta\_kubernetes\_node\_label\_(.+)

- job\_name: 'kubernetes-pods' kubernetes\_sd\_configs: - role: pod relabel\_configs: - source\_labels: [\_\_meta\_kubernetes\_pod\_annotation\_prometheus\_io\_scrape] action: keep regex: true - source\_labels: [\_\_meta\_kubernetes\_pod\_annotation\_prometheus\_io\_path] action: replace target\_label: \_\_metrics\_path\_\_ regex: (.+) - source\_labels: [\_\_address\_\_, \_\_meta\_kubernetes\_pod\_annotation\_prometheus\_io\_port] action: replace regex: ([:]+)?(?!:);(+) replacement: 1:2 target\_label: \_\_address\_\_ - action: labelmap regex: \_\_meta\_kubernetes\_pod\_label\_(.+)

- job\_name: 'kubernetes-cadvisor' kubernetes\_sd\_configs: - role: node relabel\_configs: - source\_labels: [\_\_meta\_kubernetes\_node\_name] action: replace target\_label: node - action: labelmap regex: \_\_meta\_kubernetes\_pod\_label\_(.+)

— apiVersion: apps/v1 kind: Deployment metadata: name: prometheus namespace: monitoring  
spec: replicas: 1 selector: matchLabels: app: prometheus template: metadata: labels: app: prometheus spec: containers: - name: prometheus image: prom/prometheus:latest volumeMounts: - name: prometheus-config mountPath: /etc/prometheus ports: - containerPort: 9090 volumes: - name: prometheus-config configMap: name: prometheus-config

— apiVersion: v1 kind: Service metadata: name: prometheus namespace: monitoring spec: selector: app: prometheus ports: - protocol: TCP port: 9090 targetPort: 9090 type: NodePort

— apiVersion: apps/v1 kind: Deployment metadata: name: grafana namespace: monitoring  
spec: replicas: 1 selector: matchLabels: app: grafana template: metadata: labels: app: grafana spec: containers: - name: grafana image: grafana/grafana:latest ports: - containerPort: 3000 —  
apiVersion: v1 kind: Service metadata: name: grafana namespace: monitoring spec: selector: app: grafana ports: - protocol: TCP port: 8544 targetPort: 3000 type: NodePort



**Instituto Superior  
de Engenharia**

Politécnico de Coimbra