

# **Aplicação para gestão do orquestrador Kubernetes**

Licenciatura em Engenharia Informática

João Martins Tendeiro

Miguel Francisco Lopes

Leiria, junho de 2025

# **Aplicação para gestão do orquestrador Kubernetes**

Licenciatura em Engenharia Informática

João Martins Tendeiro

Miguel Francisco Lopes

Trabalho Laboratorial nº2 (TL2) da unidade curricular de Laboratório de Tecnologias de  
Informação realizado sob a orientação do Professor Daniel Fuentes

Leiria, junho de 2025

## Resumo

O presente trabalho foi desenvolvido no âmbito da unidade curricular de Laboratório de Tecnologias de Informação e teve como principal objetivo a criação de uma aplicação gráfica que permita interagir com a API do Kubernetes, facilitando a gestão dos recursos de um cluster de forma centralizada e acessível.

O projeto envolveu, numa primeira fase, a análise comparativa de diferentes soluções de instalação de clusters Kubernetes, como o Minikube, Kind, MicroK8s e k3S. Após a seleção da solução mais adequada, foi implementado um cluster funcional composto por um nó de controlo (master) e dois nós de trabalho (workers), com documentação detalhada de todos os passos realizados.

A aplicação desenvolvida permite a autenticação por token e disponibiliza uma interface gráfica com funcionalidades para listar, criar e eliminar os principais objetos de um cluster Kubernetes, nomeadamente nodes, namespaces, pods, deployments, serviços e ingressos. Inclui ainda um dashboard com informação em tempo real sobre o estado do cluster, bem como o armazenamento de logins anteriores, facilitando sessões futuras.

Todos os testes realizados confirmaram o correto funcionamento das funcionalidades implementadas, validando o cumprimento dos objetivos propostos e demonstrando a utilidade prática da ferramenta como apoio à gestão de ambientes Kubernetes.

**Palavras-chave:** Kubernetes, orquestração, containers, cluster, API

# Abstract

This project was developed within the scope of the Information Technologies Laboratory course and had as its main objective the creation of a graphical application to interact with the Kubernetes API, facilitating centralized and user-friendly management of cluster resources.

The project began with a comparative analysis of different Kubernetes cluster installation solutions, such as Minikube, Kind, MicroK8s, and k3S. After selecting the most suitable option, a functional cluster was implemented, consisting of one control node (master) and two worker nodes, with detailed documentation of all the steps performed.

The developed application supports token-based authentication and provides a graphical interface with features to list, create, and delete the main objects of a Kubernetes cluster, namely nodes, namespaces, pods, deployments, services, and ingresses. It also includes a real-time dashboard displaying the cluster's current status, as well as storage of previous logins to facilitate future sessions.

All tests confirmed the correct functioning of the implemented features, validating the achievement of the proposed objectives and demonstrating the practical usefulness of the tool as support for managing Kubernetes environments.

**Keywords:** Kubernetes, orchestration, containers, cluster, API

# Índice

<b>Resumo</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Lista de Figuras</b>	<b>vi</b>
<b>Lista de Tabelas</b>	<b>vii</b>
<b>Lista de siglas e acrónimos</b>	<b>viii</b>
<b>1 Introdução</b>	<b>1</b>
<b>2 Conceitos</b>	<b>3</b>
2.1 Containers . . . . .	3
2.2 Image . . . . .	4
2.3 Docker . . . . .	4
2.4 Kubernetes . . . . .	4
2.5 Cluster . . . . .	5
2.6 Node . . . . .	6
2.7 Pod . . . . .	6
2.8 Deployment . . . . .	6
2.9 Service . . . . .	7
2.10 Ingress . . . . .	7
2.11 Namespace . . . . .	7
<b>3 Análise de Soluções Kubernetes</b>	<b>8</b>
3.1 K3s . . . . .	8
3.2 Minikube . . . . .	9
3.3 Kind (Kubernetes IN Docker) . . . . .	9

3.4	MicroK8s . . . . .	10
3.5	Comparação entre Minikube, Kind, MicroK8s e k3s . . . . .	11
3.6	Justificação da escolha da solução Kubernetes . . . . .	12
<b>4</b>	<b>Implementação da solução Kubernetes</b>	<b>14</b>
4.1	Requisitos de Hardware . . . . .	14
4.2	Instalação do node Master . . . . .	14
4.3	Instalação dos nodes Worker . . . . .	15
4.4	Validação do Cluster . . . . .	15
4.5	Criação do Token de acesso à API do K3s . . . . .	16
<b>5</b>	<b>Trabalho Desenvolvido</b>	<b>18</b>
5.1	Linguagem de Programação . . . . .	18
5.2	Arquitetura da Solução . . . . .	19
5.3	Base de Dados . . . . .	21
5.4	Lista de Endpoints . . . . .	23
5.5	Login . . . . .	26
5.6	Obtenção de dados da tabela . . . . .	28
5.7	Dashboard . . . . .	29
5.8	Nodes . . . . .	31
5.9	Namespaces . . . . .	31
5.9.1	Listar . . . . .	32
5.9.2	Criar . . . . .	32
5.9.3	Eliminar . . . . .	33
5.10	Pods . . . . .	33
5.10.1	Listar . . . . .	33
5.10.2	Criar . . . . .	34
5.10.3	Eliminar . . . . .	34

5.11 Deployments . . . . .	34
5.11.1 Listar . . . . .	35
5.11.2 Criar . . . . .	35
5.11.3 Eliminar . . . . .	36
5.12 Services/Ingress . . . . .	36
5.12.1 Listar . . . . .	37
5.12.2 Criar . . . . .	37
5.12.3 Eliminar . . . . .	38
<b>6 Testes</b>	<b>39</b>
6.1 Autenticação por Token . . . . .	39
6.2 Armazenamento de logins na base de dados . . . . .	41
<b>7 Análise crítica e proposta de melhorias</b>	<b>43</b>
<b>8 Conclusão</b>	<b>44</b>
<b>Bibliografia</b>	<b>45</b>
<b>Anexos</b>	<b>46</b>

## Lista de Figuras

1	Arquitetura de um container . . . . .	3
2	Arquitetura do docker . . . . .	4
3	Arquitetura de um cluster Kubernetes . . . . .	5
4	K3S . . . . .	8
5	Minikube . . . . .	9
6	Kind . . . . .	10
7	MicroK8s . . . . .	10
8	Diagrama Lógico . . . . .	21
9	Estrutura geral da Tabela . . . . .	22
10	Método EncryptToken . . . . .	22
11	Validação do login . . . . .	27
12	LoginForm . . . . .	28
13	Método para preencher os dados . . . . .	29
14	Dashboard . . . . .	30
15	<i>TabPage</i> Nodes . . . . .	31
16	<i>TabPage</i> Namespace . . . . .	32
17	<i>TabPage</i> Pods . . . . .	33
18	<i>TabPage</i> Deployments . . . . .	35
19	<i>TabPage</i> Services/Ingresss . . . . .	37
20	Criação do token . . . . .	40
21	Autenticação bem-sucedida através de token . . . . .	40
22	Token criptografado . . . . .	41
23	Histórico de logins . . . . .	42



## Lista de Tabelas

1	Comparação entre Minikube, Kind, MicroK8s e k3s . . . . .	12
2	Requisitos mínimos de hardware para K3s . . . . .	14
3	Especificações de <i>hardware</i> utilizadas nos nós do <i>cluster</i> . . . . .	19

## Lista de siglas e acrónimos

**AES** Advanced Encryption Standard. 22, 27

**API** Application Programming Interface. i, ii, 1, 8, 16, 23, 27, 29, 31–38, 43, 44

**ARM** Advanced RISC Machine. 8

**CD** Continuous Delivery. 9, 11

**CI** Continuous Integration. 9–11

**CPU** Central Processing Unit. 3, 8, 9, 12, 30

**DNS** Domain Name System. 11

**HA** High Availability. 8, 11

**HTTP** Hypertext Transfer Protocol. 7, 37, 38

**HTTPS** Hypertext Transfer Protocol Secure. 7, 16, 23, 27

**IoT** Internet of Things. 8, 11

**IP** Internet Protocol. 6, 7, 15, 21, 26–28, 30, 40, 41

**IT** Information Technology. 3

**IV** Initialization Vector. 22

**OS** Operating System. 3

**RAM** Random Access Memory. 9

**REST** Representational State Transfer. 43

**SQL** Structured Query Language. 8

**SQL** Structured Query Language. 19, 21

**VM** Virtual Machine. 9, 11, 12

# 1 Introdução

Atualmente, a gestão de aplicações em ambientes distribuídos representa um grande desafio, sobretudo pela necessidade de garantir escalabilidade, disponibilidade e resiliência. A utilização de contentores, através de ferramentas como o Docker, trouxe avanços significativos ao permitir o empacotamento e a execução de aplicações de forma isolada e portátil. No entanto, a gestão de múltiplos contentores em diferentes nós exige uma camada de orquestração eficiente.

O Kubernetes surge como uma solução para este problema, permitindo a orquestração automatizada de containers em larga escala, com funcionalidades como balanceamento de carga, monitorização, escalonamento automático e autorrecuperação.

O presente trabalho tem como objetivo o desenvolvimento de uma aplicação simples para auxiliar na gestão de clusters Kubernetes, proporcionando uma interface acessível para a visualização e controlo dos principais recursos do orquestrador.

A importância deste tema justifica-se pela crescente adoção de tecnologias baseadas em microserviços e computação em *cloud*, onde o Kubernetes se estabelece como padrão na gestão de aplicações distribuídas. A implementação de uma aplicação de gestão simplificada permite explorar, na prática, os conceitos fundamentais de orquestração de containers, bem como a interação com a API do Kubernetes e os seus principais componentes.

Entre os objetivos específicos, destacam-se:

- Implementar um cluster Kubernetes funcional com, pelo menos, um nó *master* e dois nós *worker*;
- Documentar detalhadamente os passos da instalação e configuração do cluster para permitir a sua replicação;
- Desenvolver uma aplicação que interaja com a API do Kubernetes;
- Criar um dashboard com informação em tempo real sobre o estado do cluster e a utilização

de recursos;

- Implementar funcionalidades para listar, criar e eliminar os principais objetos do Kubernetes, nomeadamente:
  - **Nodes**
  - **Namespaces**
  - **Pods**
  - **Deployments**
  - **Services e Ingress**
- Proporcionar uma interface de gestão simplificada que facilite operações básicas no cluster.

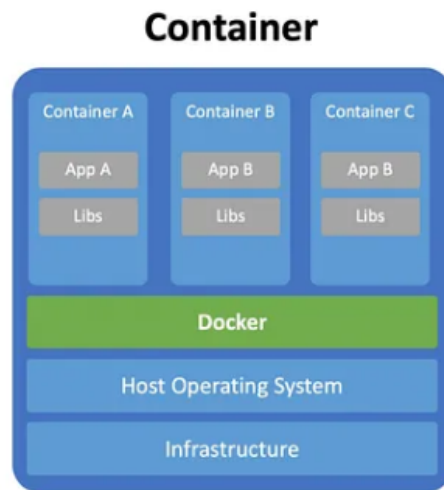
Este relatório está organizado em oito capítulos. O Capítulo 2 introduz os conceitos fundamentais relacionados com containers, Kubernetes e os seus principais componentes. O Capítulo 3 apresenta a análise comparativa entre diferentes soluções de instalação de clusters Kubernetes. O Capítulo 4 descreve o processo de implementação do cluster utilizado. No Capítulo 5 é detalhado o desenvolvimento da aplicação e as funcionalidades implementadas. O Capítulo 6 apresenta os testes realizados. O Capítulo 7 propõe melhorias com base numa análise crítica. Por fim, o Capítulo 8 encerra com as principais conclusões do trabalho.

## 2 Conceitos

Neste capítulo abordamos alguns conceitos relevantes para o projeto, como containers, image, docker, kubernetes, cluster, node, pod, deployment, service, ingress e namespace.

### 2.1 Containers

Os containers são unidades executáveis de software que agrupam o código de uma aplicação juntamente com as bibliotecas e dependências. Permitem que o código seja executado em qualquer ambiente informático, seja num computador pessoal, numa infraestrutura de IT tradicional ou na cloud.



**Figura 1:** Arquitetura de um container

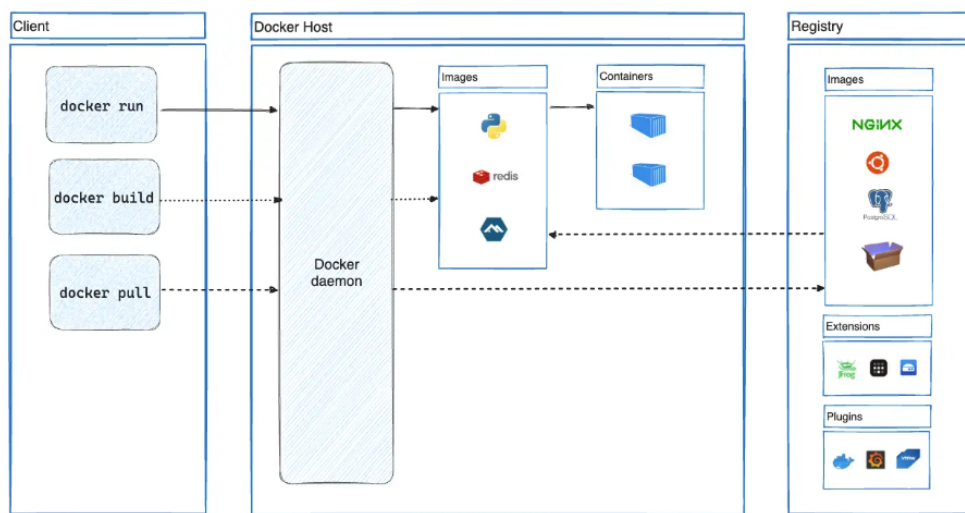
Os contentores tiram partido de uma forma de virtualização do sistema operativo (OS), na qual funcionalidades do kernel do OS (por exemplo, namespaces e cgroups no Linux) podem ser usadas para isolar processos e controlar a quantidade de CPU, memória e disco a que esses processos podem aceder.

## 2.2 Image

Uma imagem de container é um pacote binário que encapsula uma aplicação juntamente com todas as suas dependências de software. Estas imagens são executáveis autónomos que funcionam num ambiente de execução bem definido, sendo depois referenciadas para criar containers ou pods.

## 2.3 Docker

O Docker é uma plataforma open source para desenvolver, distribuir e executar aplicações. O Docker permite separar as aplicações da infraestrutura, possibilitando o acesso mais rápido ao software. Com o Docker, gere a infraestrutura da mesma forma que gere as aplicações. Ao tirar partido das metodologias do Docker para distribuir, testar e implementar código, é possível reduzir significativamente o tempo entre escrever o código e executá-lo.



**Figura 2:** Arquitetura do docker

## 2.4 Kubernetes

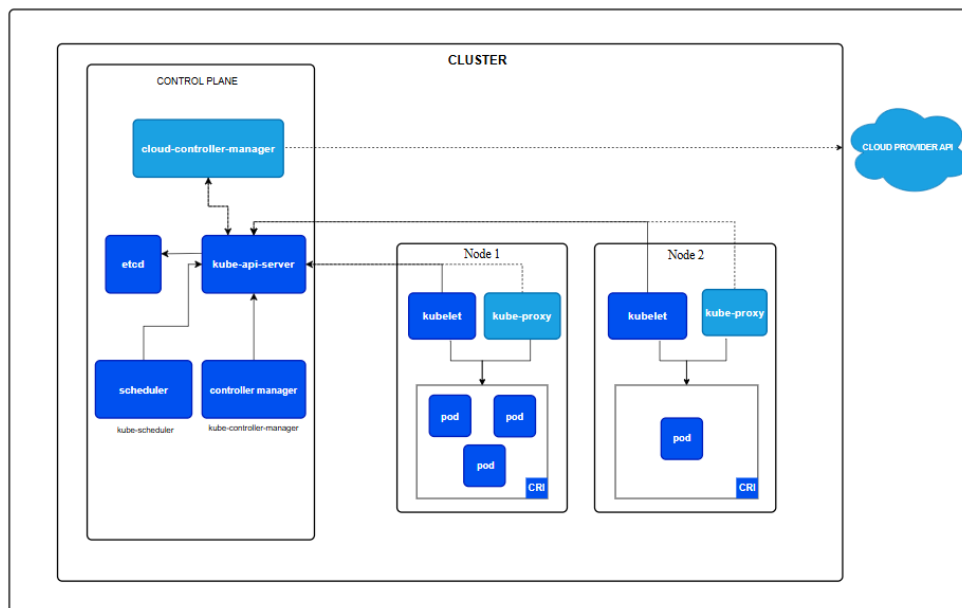
Kubernetes, é uma plataforma open source de orquestração de contentores, utilizada para agendar e automatizar o deployment, gestão e escalabilidade de aplicações em contentores.

Atualmente, o Kubernetes e o ecossistema mais amplo de tecnologias relacionadas com contêntores tornaram-se a base da infraestrutura moderna em cloud. Este ecossistema permite às organizações oferecer um ambiente de computação híbrido e *multicloud* altamente produtivo, capaz de executar tarefas complexas relacionadas com infraestrutura e operações. Além disso, suporta o desenvolvimento *cloud-native* ao possibilitar uma abordagem de construir uma vez e implementar em qualquer lugar para a criação de aplicações.

## 2.5 Cluster

Um cluster Kubernetes é composto por um plano de controlo (master node) e um conjunto de nós de trabalho (worker nodes), que executam aplicações containerizadas. O plano de controlo gere o estado do cluster, enquanto os nodes executam os Pods, que são as unidades onde correm as aplicações (**Fig. 3**).

Em ambientes de produção, tanto o plano de controlo como os nodes são distribuídos por várias máquinas, garantindo tolerância a falhas e alta disponibilidade.



**Figura 3:** Arquitetura de um cluster Kubernetes



## 2.6 Node

Um nó, em redes de comunicação, refere-se a qualquer dispositivo ou ponto que se liga a uma rede. Os nós desempenham um papel fundamental na transmissão, recepção e processamento de dados. Seja um computador, um router ou até uma impressora, todos os equipamentos de rede são considerados nós. Em termos simples, um nó é qualquer componente físico ou virtual que facilita a comunicação numa rede, sendo identificado por um endereço único, como um endereço IP.

## 2.7 Pod

Um pod é a unidade mínima de execução no Kubernetes. Um pod encapsula uma ou mais aplicações. Os pods são efêmeros por natureza; se um pod (ou o nó onde está a ser executado) falhar, o Kubernetes pode criar automaticamente uma nova réplica desse pod para continuar as operações. Os pods incluem um ou mais contentores (como contentores Docker).

Os pods também fornecem dependências de ambiente, incluindo volumes de armazenamento persistente (armazenamento permanente e disponível para todos os pods no cluster) e dados de configuração necessários para executar os contentores dentro do pod.

## 2.8 Deployment

Um Deployment no kubernetes é responsável por criar e atualizar pods. Define para o Kubernetes como deve criar e gerir esses pods no cluster.

O controlador do Deployment monitoriza continuamente as instâncias da aplicação e garante que, se algum node falhar ou for removido, as instâncias sejam recriadas noutro node disponível. Isto fornece um mecanismo de recuperação automática para manter a aplicação sempre ativa.

## 2.9 Service

Um Service em Kubernetes é uma forma de expor uma aplicação (executada em Pods) dentro do cluster através de um ponto de acesso único e estável na rede. Os Pods são efêmeros e mudam frequentemente. O Service abstrai essa variação, fornecendo um endereço estável para os clientes se ligarem, mesmo que os Pods atrás dele mudem. Permite que aplicações front-end descubram e comuniquem com os back-ends, sem se preocupar com os detalhes da infraestrutura ou nomes/IPs dinâmicos dos Pods.

Os Services permitem expor aplicações sem precisar modificar o código da aplicação.

## 2.10 Ingress

Um Ingress em Kubernetes é um recurso que permite expor serviços HTTP e HTTPS para fora do cluster, utilizando uma configuração que compreende os detalhes do protocolo *web*. Isso significa que o Ingress entende conceitos como nomes de domínio, caminhos de URL e regras de encaminhamento baseadas em HTTP, permitindo encaminhar o tráfego para diferentes serviços (backends) conforme definido pelo utilizador.

Através do Ingress, é possível controlar de forma centralizada o acesso a várias aplicações dentro do cluster, sem necessidade de expor cada serviço individualmente com um IP externo.

## 2.11 Namespace

Um Namespace em Kubernetes é um mecanismo que permite isolar e organizar recursos dentro de um mesmo cluster. Cada namespace funciona como um “espaço lógico” onde os nomes dos recursos (como Deployments, Services, Pods, etc.) precisam ser únicos apenas dentro desse namespace, permitindo reutilizar nomes em diferentes contextos.

## 3 Análise de Soluções Kubernetes

Neste capítulo, são analisadas e comparadas diferentes soluções de instalação de clusters Kubernetes, com o intuito de identificar suas principais características, requisitos de infraestrutura, níveis de complexidade e facilidade de utilização. Essa análise tem como objetivo fornecer uma base sólida de informações que auxilie na escolha da solução mais adequada, considerando o contexto e as necessidades específicas do ambiente de desenvolvimento.

### 3.1 K3s

O k3s é uma distribuição Kubernetes leve, desenvolvida pela Rancher Labs, concebida para ser simples de instalar e otimizada para ambientes com recursos limitados, como *edge computing*, IoT, desenvolvimento e pequenas produções.



**Figura 4:** K3S

**Funcionamento:** Simplifica o Kubernetes ao combinar os principais componentes num único binário compacto, removendo funcionalidades não essenciais para reduzir o consumo de recursos. Pode ser instalado diretamente em sistemas Linux, incluindo arquiteturas ARM (como Raspberry Pi), e suporta clusters multi-nó com alta disponibilidade (HA).

**Principais funcionalidades:** Inclui um *runtime* de containers embutido, suporte a armazenamento leve com SQLite ou opções mais robustas como etcd, e rede simplificada. Suporta a API padrão do Kubernetes, garantindo compatibilidade com ferramentas e workloads habituais.

**Vantagens:** Muito leve e rápido a instalar, adequado para equipamentos com pouca memória e CPU, com suporte para produção em pequena e média escala.

**Desvantagens:** Funcionalidades avançadas são limitadas para manter a leveza, o que pode restringir a integração com certos addons e extensões.

## 3.2 Minikube

O Minikube é uma ferramenta popular que permite executar um cluster Kubernetes local, focado em ambientes de desenvolvimento. Cria uma máquina virtual (VM) ou utiliza um driver (como Docker, VirtualBox ou Hyper-V) para alojar o cluster.



**Figura 5:** Minikube

**Funcionamento:** O Minikube instala um nó Kubernetes completo dentro de uma VM/container, gerido localmente.

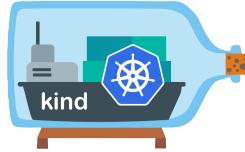
**Principais funcionalidades:** Suporte a clusters com múltiplos nós. Permite ativar facilmente addons como o Dashboard, Metrics Server, Ingress, entre outros. Suporta a simulação de ambientes com configurações específicas (CPU, RAM, armazenamento, etc.).

**Vantagens:** Simula de forma muito realista um ambiente de Kubernetes. Amplo suporte de documentação e comunidade.

**Desvantagens:** Utiliza mais recursos do sistema, especialmente memória RAM e CPU. Arranque mais lento em comparação com soluções baseadas apenas em containers.

## 3.3 Kind (Kubernetes IN Docker)

O Kind é uma solução mais leve e rápida, que cria clusters Kubernetes dentro de containers Docker. É amplamente utilizado em ambientes de integração contínua (CI/CD) devido à sua leveza e velocidade.



**Figura 6:** Kind

**Funcionamento:** Cada nó Kubernetes é, na prática, um container Docker.

**Principais funcionalidades:** Suporte a clusters multi-nó. Criação e destruição de clusters de forma rápida (ideal para testes automáticos). Integração fácil com pipelines CI (GitHub Actions, GitLab CI, Jenkins, etc.).

**Vantagens:** Extremamente leve e rápido a iniciar. Não necessita de hipervisores nem de máquinas virtuais, apenas de Docker.

**Desvantagens:** Algumas funcionalidades mais avançadas de Kubernetes podem ser limitadas (por exemplo, em redes complexas). Não adequado para ambientes de produção — apenas para desenvolvimento e testes.

### 3.4 MicroK8s

O MicroK8s, desenvolvido pela Canonical (empresa responsável pelo Ubuntu), é uma distribuição Kubernetes que oferece uma instalação compacta e otimizada, adequada tanto para desenvolvimento como para pequenas produções.



**Figura 7:** MicroK8s

**Funcionamento:** Instala um cluster Kubernetes diretamente no sistema operativo, através do snap (no Ubuntu/Linux). Em Windows e macOS, recorre a uma máquina virtual em segundo plano.

**Principais funcionalidades:** Instalação simplificada através de um único comando (snap install microk8s). Suporte nativo a addons como DNS, Ingress, Istio, Knative, Prometheus, entre outros. Permite a criação de clusters multi-nó com alta disponibilidade (HA).

**Vantagens:** Leve mas muito próximo de um ambiente de Kubernetes de produção. Permite utilização em produção para ambientes de pequena escala (IoT, *edge computing*, etc.). Atualizações automáticas através do Snap.

**Desvantagens:** A melhor experiência de utilização ocorre em sistemas Linux — em Windows e macOS pode exigir mais configurações. Algumas operações avançadas podem ser menos intuitivas para iniciantes.

### 3.5 Comparação entre Minikube, Kind, MicroK8s e k3s

Critério	Minikube	Kind	MicroK8s	k3s
Objetivo principal	Desenvolvimento local	Testes locais e CI/CD	Desenvolvimento e produção leve	<i>Edge computing</i> , IoT, desenvolvimento e produção leve
Execução	Máquina virtual ou <i>driver</i> Docker	Containers Docker	Instalação direta no sistema operativo (Snap)	Instalação direta no sistema operativo, binário compacto
Instalação	Muito fácil	Muito fácil (requer Docker)	Muito fácil (Linux)	Muito fácil, único binário, rápido
Consumo de recursos	Médio/Alto (depende da VM)	Muito baixo	Baixo/Médio	Muito baixo, otimizado para hardware limitado
Desempenho	Bom, mas dependente da VM	Muito bom (containers leves)	Excelente em Linux	Excelente em dispositivos com poucos recursos

<b>Critério</b>	<b>Minikube</b>	<b>Kind</b>	<b>MicroK8s</b>	<b>k3s</b>
Facilidade de utilização	Alta — interface simples, muitos drivers, ideal para iniciantes	Média — fácil para testes, mas configuração de multi-nó e rede pode ser complexa	Alta — comando ”microk8s” unificado, fácil ativar addons, especialmente em Linux	Média — configuração inicial simples, mas gestão de addons e rede pode requerer mais conhecimento
Suporte a clusters multi-nó	Sim, com múltiplos nós em VM/perfis locais (desenvolvimento)	Sim, <i>clusters</i> multi-nó usando containers Docker	Sim, suporta multi-nó (especialmente em Linux)	Sim, suporte robusto a multi-nó
Dependência de VMs	Sim (exceto modo Docker, que usa containers)	Não (usa apenas containers)	Sim no Windows/macOS	Não (exceto em Windows via VM)

**Tabela 1:** Comparação entre Minikube, Kind, MicroK8s e k3s

### 3.6 Justificação da escolha da solução Kubernetes

A escolha do k3s como solução Kubernetes para este projeto baseia-se em vários fatores que tornam mais adequado face às alternativas consideradas, conforme ilustrado na tabela 1.

Primeiramente, a leve estrutura do k3s destaca-se quando comparada com soluções como o Minikube, que apresenta um consumo significativamente superior de recursos devido à dependência de máquinas virtuais, tornando-o menos eficiente para ambientes com hardware limitado. O k3s utiliza um binário único e compacto que reduz o uso de CPU e memória, sendo ideal para equipamentos com recursos restritos.

Em comparação com o Kind, que também é uma solução leve, o k3s oferece um suporte mais robusto para *clusters* multi-nó e alta disponibilidade, características essenciais para a escalabilidade e fiabilidade em ambientes de produção de pequena a média escala. Enquanto o Kind está mais vocacionado para testes, já o k3s é adequado para ambientes reais, proporcionando maior

estabilidade e capacidades de rede mais completas.

Relativamente ao MicroK8s, apesar de oferecer uma experiência próxima do Kubernetes tradicional e ser adequado para pequenas produções, a sua instalação depende do snap em sistemas Linux e de máquinas virtuais em Windows e macOS, o que pode aumentar a complexidade e o consumo de recursos. O k3s apresenta maior portabilidade, suporta diversas arquiteturas e reduz a dependência de virtualização, facilitando a gestão e a implementação.

Desta forma, o k3s representa um equilíbrio entre desempenho, escalabilidade, facilidade de utilização e baixo consumo de recursos, justificando plenamente a sua escolha para este projeto em detrimento das outras soluções analisadas.



## 4 Implementação da solução Kubernetes

Nesta segunda parte do trabalho, procedeu-se à implementação da solução Kubernetes selecionada na secção anterior — o *k3s*.

São apresentados os requisitos mínimos para a implementação do ambiente Kubernetes. Em seguida, descrevem-se detalhadamente todos os passos realizados para a instalação e configuração do cluster, de modo a permitir que qualquer utilizador possa replicar o processo com sucesso.

### 4.1 Requisitos de Hardware

De acordo com a documentação oficial do *K3s*, os requisitos mínimos de hardware variam consoante o papel de cada nó no cluster. A Tabela 2 apresenta os valores mínimos recomendados para a instalação e funcionamento do ambiente Kubernetes baseado em *K3s*.

Tipo de Nó	CPU	Memória RAM
Servidor (master)	2 cores	2 GB
Agente (worker)	1 core	512 MB

**Tabela 2:** Requisitos mínimos de hardware para *K3s*

Estes valores correspondem ao mínimo necessário para a execução do *K3s*, podendo ser ajustados em função da carga de trabalho esperada. Para ambientes de produção, ou aplicações com maiores requisitos de desempenho, recomenda-se utilizar hardware com especificações superiores.

Relativamente ao sistema operativo, o *K3s* suporta distribuições baseadas em Linux, incluindo, entre outras, Ubuntu e Debian.

### 4.2 Instalação do node Master

A instalação do nó master consiste na execução de um script disponibilizado oficialmente pelo *K3s*, o qual automatiza todo o processo de configuração do servidor Kubernetes. Esta ins-

talação deve ser feita com permissões de root, garantindo que todos os componentes necessários são corretamente instalados.

```
# Autenticação como root
sudo su
# Download e instalação do K3s
curl -sfL https://get.k3s.io | sh -
```

### 4.3 Instalação dos nodes Worker

Nos nós worker, o primeiro passo consiste igualmente na autenticação como root e na obtenção do token de junção ao cluster, gerado automaticamente pelo master durante a instalação.

```
# Autenticação como root
sudo su
# Definir variáveis necessárias
K3S_URL=https://<ENDEREÇO-IP-DO-MASTER>:6443
K3S_TOKEN=<TOKEN-FORNECIDO-PELO-MASTER>
# Instalação do agente (worker)
curl -sfL https://get.k3s.io | K3S_URL K3S_TOKEN sh -
```

O endereço IP do master e o token devem ser substituídos pelos valores concretos do ambiente criado. O token encontra-se disponível no master no seguinte ficheiro:

```
/var/lib/rancher/k3s/server/node-token
```

### 4.4 Validação do Cluster

Após a instalação dos nós worker, a correta junção ao cluster pode ser validada através do comando seguinte, executado no nó master:

```
kubectl get nodes
```

Deverão ser listados todos os nós criados com o estado *Ready*.

## 4.5 Criação do Token de acesso à API do K3s

Para permitir o acesso remoto à API do K3s, é necessário utilizar um token de autenticação associado a uma *ServiceAccount* com permissões adequadas. Este token permite interagir diretamente com o servidor da API do Kubernetes.

Antes de criar uma nova *ServiceAccount*, pode-se listar todas as *ServiceAccounts* existentes num dado namespace, usando o comando:

```
# Neste exemplo, o namespace usado é o default
kubectl get serviceaccounts -n default
```

Se não existir nenhuma *ServiceAccount* adequada ou se for preferível criar uma dedicada ao acesso externo, siga os passos abaixo:

```
# Criação de uma nova ServiceAccount
kubectl create serviceaccount <NOME-DA-SERVICEACCOUNT>

# Conceder permissões à ServiceAccount. Neste exemplo, permissões administrativas.
kubectl create clusterrolebinding api-user-binding
--clusterrole=cluster-admin
--serviceaccount=default:<NOME-DA-SERVICEACCOUNT>
```

Após criada e configurada a *ServiceAccount*, é possível gerar um token de acesso com os seguintes comandos:

```
# Token válido por omissão (1 hora, aproximadamente)
kubectl -n default create token <NOME-DA-SERVICEACCOUNT>

# Token válido por x horas
kubectl -n default create token <NOME-DA-SERVICEACCOUNT> --duration=<xh>
```

O valor gerado corresponde ao token necessário para a autenticação. Este token é inserido no cabeçalho das requisições HTTPS, a partir do campo *Authorization*, conforme o exemplo seguinte:

```
# Exemplo de chamada à API com curl
```

```
curl -k https://<ENDereco-IP-DO-MASTER>:6443/api  
--header "Authorization: Bearer <TOKEN>"
```

## 5 Trabalho Desenvolvido

Neste capítulo iremos abordar e explicar todas as funcionalidades implementadas no projeto. Começamos por referir as funcionalidades e de seguida mostramos, com recurso a imagens, a forma como foram implementadas e o seu funcionamento.

### Funcionalidades implementadas

- Um *dashboard* com as informações do *cluster* (recursos utilizados).
- *Nodes*: listar.
- *Namespaces*: listar, criar e eliminar.
- *Pods*: listar, criar e eliminar.
- *Deployments*: listar, criar e eliminar.
- *Services/Ingress*: listar, criar e eliminar.

### Funcionalidades adicionais implementadas

- Integração com uma base de dados para armazenamento dos dados de login.
- Implementação de autenticação por Token.

### 5.1 Linguagem de Programação

C# foi a linguagem de programação escolhida, por ser uma linguagem orientada a objetos que permite criar uma variedade de aplicações seguras e robustas, sendo fortemente utilizada para o desenvolvimento de aplicações tradicionais em Windows, *Web* e também para equipamentos móveis. Desenvolvida pela Microsoft faz parte da sua plataforma *.NET Framework*.

A plataforma **Visual Studio 2022**, foi a escolhida para desenvolver o nosso projeto, devido ao conjunto de ferramentas, interface amigável, suporte nativo à linguagem C# e integração com o *.NET Framework*, o que permite o desenvolvimento, análise e teste dos projetos. Ainda ofe-

rece uma excelente integração com base de dados, principalmente através da tecnologia *Entity Framework*, o que permite a comunicação eficiente com bases de dados relacionais, como *Structured Query Language (SQL) Server*. A criação da base de dados e as suas operações podem ser realizadas diretamente no Visual Studio 2022, tornando o processo mais prático e centralizado.

## 5.2 Arquitetura da Solução

A solução desenvolvida é constituída por três nós do *cluster* — tanto o *master* como os dois *workers* — configurados com as mesmas especificações de *hardware*, conforme descrito na Tabela 3.

Componente	Especificação
CPU	2 cores
Memória RAM	4 GB
Armazenamento	40 GB de disco
Sistema Operativo	Ubuntu 22.04 LTS

**Tabela 3:** Especificações de *hardware* utilizadas nos nós do *cluster*.

Estas especificações superam os requisitos mínimos recomendados pelo *K3s*, garantindo maior estabilidade e capacidade de resposta durante o funcionamento do *cluster*. O ambiente foi virtualizado recorrendo a plataforma VMware, o que permitiu uma gestão eficiente dos recursos e o isolamento dos nós do *cluster*.

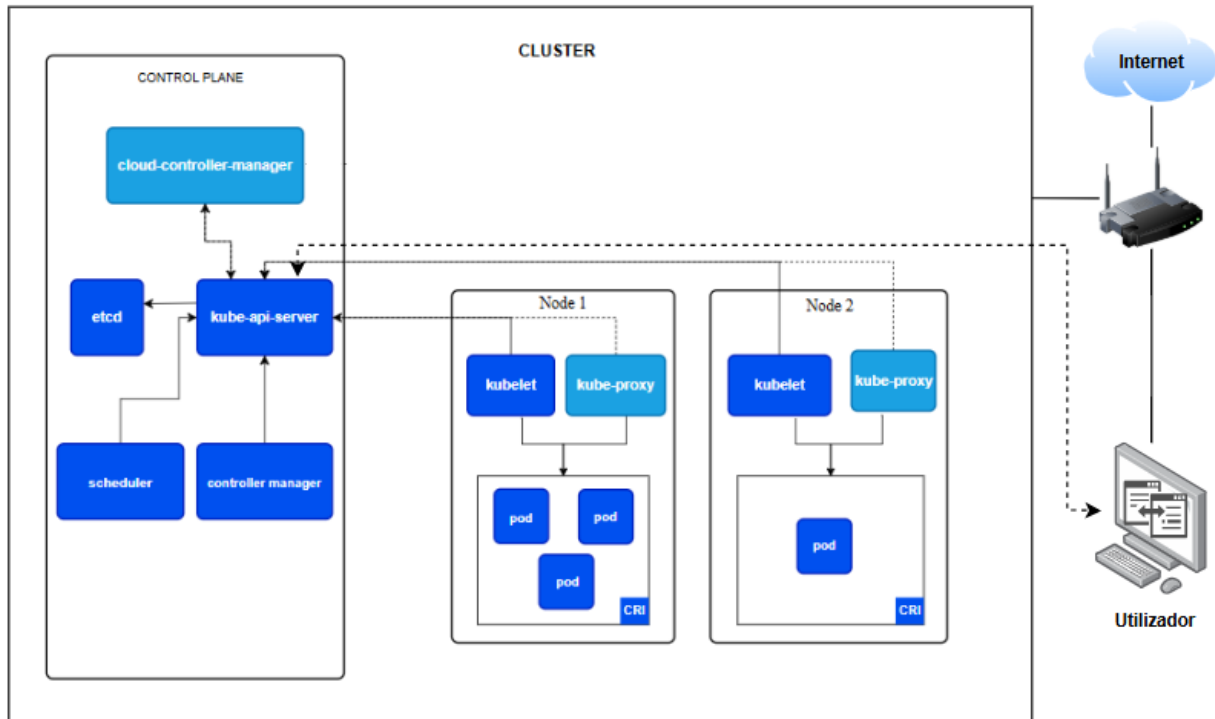
Os passos para a configuração da solução *Kubernetes* selecionada (*K3s*) no nosso *cluster* foram mencionados no Capítulo anterior, **Implementação da Solução Kubernetes**.

A aplicação para gestão do orquestrador Kubernetes foi desenvolvida utilizando o Form Designer da plataforma Visual Studio 2022, que permite testar e melhorar constantemente o visual da interface.

Foram criados **três forms** distintos para estruturar a aplicação de forma modular e organizada:

- **LoginForm**: Responsável exclusivamente pela autenticação do utilizador. Este *form* abstrai as demais funcionalidades, garantindo que o acesso às outras áreas da aplicação só seja permitido após uma autenticação bem-sucedida.
- **DashboardForm**: Exibe um painel com informações gerais do *cluster*, como os recursos utilizados, oferecendo uma visão consolidada do estado do sistema Kubernetes.
- **MainForm**: Contém as principais funcionalidades da aplicação, incluindo a gestão de *Nodes*, *Namespaces*, *Pods*, *Deployments* e *Services/Ingress* (listar, criar e eliminar). Este *form* utiliza um componente *TabControl* com várias *TabPage*, onde cada separador representa uma funcionalidade distinta, facilitando a navegação e a interação com as diferentes operações.

A interface gráfica foi projetada com foco na usabilidade e estética. O fundo da aplicação apresenta uma imagem em tons escuros com elementos gráficos inspirados no Kubernetes, promovendo uma identidade visual coerente. As caixas de texto possuem cores claras para facilitar a leitura, enquanto os botões utilizam cores vivas e contrastantes, garantindo uma navegação intuitiva e visualmente atrativa.



**Figura 8:** Diagrama Lógico

### 5.3 Base de Dados

A base de dados foi criada a partir da tecnologia *Entity Framework* do Visual Studio 2022, sendo uma base de dados *SQL Server LocalDB*. Foi criada com o propósito de guardar somente os dados de autenticação de vários MikroTiks numa tabela nomeada como "Table". Esta tabela (**figura 9**) contém 4 campos:

- **Id:** chave primária, preenchida automaticamente de forma sequencial;
- **Name:** nome para os dados de autenticação;
- **IpAddress:** endereço IP do Master;
- **Token:** Token para autenticação;



	Nome	Tipo de Dados	Permitir Nulos	Padrão
	Id	int	<input type="checkbox"/>	
	Name	nvarchar(100)	<input type="checkbox"/>	
	IpAddress	nvarchar(50)	<input type="checkbox"/>	
	Token	nvarchar(MAX)	<input type="checkbox"/>	
			<input type="checkbox"/>	

**Figura 9:** Estrutura geral da Tabela

Antes de ser armazenada na tabela, a palavra-passe (plainText) é encriptada através do método ”*Método EncryptToken*”, que utiliza o algoritmo *Advanced Encryption Standard (AES)* para encriptar o token com uma chave simétrica (key). O método ajusta a chave para garantir que tem 16 bytes e gera um vetor de inicialização (IV) aleatório para cada operação de encriptação, aumentando assim a segurança. O IV é armazenado no início do fluxo de dados encriptados, para poder ser reutilizado na desencriptação. O token é escrito num ”*CryptoStream*”, que aplica a encriptação utilizando o método ”*CreateEncryptor*” do AES. Por fim, o conteúdo encriptado, incluindo o IV, é convertido para Base64 e devolvido como uma cadeia de caracteres (*string*).

```
private string EncryptToken(string plainText, string key)
{
    using (var aes = Aes.Create())
    {
        aes.Key = Encoding.UTF8.GetBytes(key.PadRight(16).Substring(0, 16));
        aes.GenerateIV();

        using (var encryptor = aes.CreateEncryptor(aes.Key, aes.IV))
        using (var ms = new MemoryStream())
        {
            ms.Write(aes.IV, 0, aes.IV.Length);
            using (var cs = new CryptoStream(ms, encryptor, CryptoStreamMode.Write))
            using (var writer = new StreamWriter(cs))
            {
                writer.Write(plainText);
            }
            return Convert.ToBase64String(ms.ToArray());
        }
    }
}
```

**Figura 10:** Método EncryptToken

## 5.4 Lista de Endpoints

As ligações à API do Kubernetes são realizadas através do protocolo HTTPS, garantindo a comunicação segura entre a aplicação e o servidor. No entanto, como o certificado utilizado pelo cluster é auto assinado, o código ignora temporariamente a validação do certificado digital, permitindo o estabelecimento da conexão sem erros de segurança.

### Nodes

Pedido	Endpoint
Listar nodes	GET - /api/v1/nodes

### Namespaces

Pedido	Endpoint + Body (YAML)
Listar namespaces	GET - /api/v1/namespaces
Listar namespace específico	GET - /api/v1/namespaces/{namespace}
Criar namespace	POST - /api/v1/namespaces <pre> apiVersion: v1 kind: Namespace metadata:   name: nome-do-namespace           </pre>
Apagar namespace	DELETE - /api/v1/namespaces/{name}

### Pods

Pedido	Endpoint + Body (YAML)
Listar pods de um namespace	GET - /api/v1/namespaces/{namespace}/pods
Listar pod específico	GET - /api/v1/namespaces/{namespace}/pods/{pod}
Listar todos os pods	GET - /api/v1/pods

Pedido	Endpoint + Body (YAML)
Criar pod	POST - /api/v1/namespaces/{namespace}/pods <pre> apiVersion: v1 kind: Pod metadata:   name: httpd spec:   containers:     - name: httpd       image: httpd:latest       ports:         - containerPort: 80           </pre>
Apagar pod	DELETE - /api/v1/namespaces/{namespace}/pods/{name}

## Deployments

Pedido	Endpoint + Body (YAML)
Listar todos os deployments	GET - /apis/apps/v1/deployments
Listar deployments de um namespace	GET - /apis/apps/v1/namespaces/{namespace}/deployments
Listar deployment específico	GET - /apis/apps/v1/namespaces/{namespace}/deployments/{name}

Pedido	Endpoint + Body (YAML)
Criar deployment	POST - /apis/apps/v1/namespaces/{namespace}/deployments <pre> apiVersion: apps/v1 kind: Deployment metadata:   name: deployment-example spec:   replicas: 3   selector:     matchLabels:       app: nginx   template:     metadata:       labels:         app: nginx     spec:       containers:         - name: nginx           image: nginx:1.14           ports:             - containerPort: 80           </pre>
Apagar deployment	DELETE - /apis/apps/v1/namespaces/{namespace}/deployments/{name}

## Ingresses

Pedido	Endpoint + Body (YAML)
Listar todos os ingresses	GET - /apis/networking.k8s.io/v1/ingresses
Listar ingresses de um namespace	GET - /apis/networking.k8s.io/v1/namespaces/{namespace}/ingresses
Listar ingress específico	GET - /apis/networking.k8s.io/v1/namespaces/{namespace}/ingresses/{name}

Pedido	Endpoint + Body (YAML)
Criar ingress	POST - /apis/networking.k8s.io/v1/namespaces/{namespace}/ingresses
Apagar ingress	DELETE - /apis/networ- king.k8s.io/v1/namespaces/{namespace}/ingresses/{name}

## Services

Pedido	Endpoint + Body (YAML)
Listar todos os services	GET - /api/v1/services
Listar services de um namespace	GET - /api/v1/namespaces/{namespace}/services
Listar service específico	GET - /api/v1/namespaces/{namespace}/services/{name}
Criar service	POST - /api/v1/namespaces/{namespace}/services <pre> apiVersion: v1 metadata:   name: service-example spec:   ports:     - name: http       port: 80       targetPort: 80   selector:     app: nginx     type: LoadBalancer </pre>
Apagar service	DELETE - /api/v1/namespaces/{namespace}/services/{name}

## 5.5 Login

O processo de *login* foi implementado no formulário "LoginForm", o qual inclui três campos obrigatórios: nome de utilizador, endereço IP do cluster Kubernetes e *token* de acesso. Todos os campos devem ser preenchidos; caso contrário, é apresentada uma mensagem de erro informando o utilizador da obrigatoriedade dos mesmos.

A autenticação é realizada através de um pedido HTTPS ao endpoint `/api/v1/pods` da API do Kubernetes, utilizando o *token* fornecido no cabeçalho *Authorization: Bearer*. Se a resposta do servidor for bem-sucedida (código 200 OK), o formulário principal da aplicação (*DashboardForm*) é aberto, enquanto o formulário de login é ocultado. Nos casos em que o *token* está expirado ou não possui permissões suficientes (códigos 401), o sistema tenta localizar e remover automaticamente os dados correspondentes na base de dados, apresentando uma mensagem ao utilizador a informar que o token expirou e que os dados foram eliminados (**Figura 11**).

```
using (var client = new HttpClient(handler))
{
    client.DefaultRequestHeaders.Add("Authorization", $"Bearer {token}");
    HttpResponseMessage response = await client.GetAsync(apiUrl).ConfigureAwait(false);

    if (response.IsSuccessStatusCode)
    {
        return true;
    }

    if (response.StatusCode == HttpStatusCode.Unauthorized)
    {
        using (var context = new AppDbContext())
        {
            var tableToRemove = await context.Table
                .FirstOrDefaultAsync(d => d.Name == username && d.IpAddress == ipAddress && d.Token == token)
                .ConfigureAwait(false);

            if (tableToRemove != null)
            {
                context.Table.Remove(tableToRemove);
                await context.SaveChangesAsync().ConfigureAwait(false);

                this.Invoke(new Action(() =>
                {
                    LoadData();
                    MessageBox.Show("Expired token! Data has been removed from the database.", "Authentication Failed", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                }));
            }
            else
            {
                this.Invoke(new Action(() =>
                {
                    MessageBox.Show("Invalid or expired token.", "Authentication Failed", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                }));
            }
        }
    }

    return false;
}
```

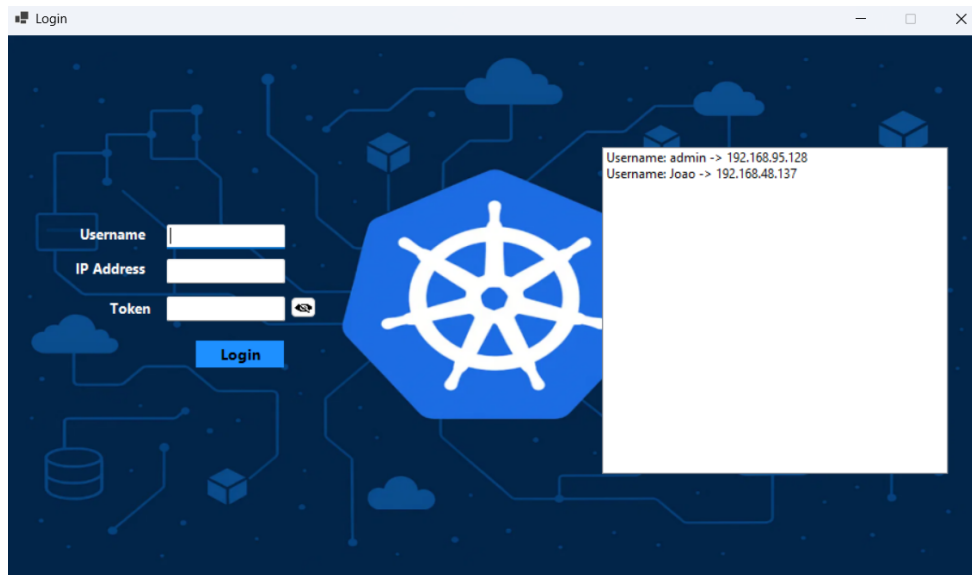
**Figura 11:** Validação do login

Uma vez validado o acesso, os dados de login são armazenados na base de dados, desde que ainda não exista um registo com a mesma combinação de nome de utilizador e endereço IP. Para garantir a unicidade dos nomes dos diferentes dados de login, o nome é ajustado dinamicamente com um sufixo numérico (ex.: *user\_1*, *user\_2*, etc.), caso existam conflitos com nomes já registados.

Antes do armazenamento, o *token* é encriptado através do algoritmo AES, utilizando uma chave simétrica definida pela aplicação. A função responsável por este processo, "SaveLogin-

Details”, verifica inicialmente se já existem os dados na base de dados. Se não existir, aplica a encriptação ao *token* e guarda os dados numa tabela chamada *Table*.

Além disso, o formulário inclui uma funcionalidade para visualizar ou ocultar o conteúdo do campo de *token*, através de um botão com o ícone de “olho”, permitindo ao utilizador verificar o valor inserido antes de confirmar o login.



**Figura 12:** LoginForm

## 5.6 Obtenção de dados da tabela

A tabela *Table* contém os dados de autenticações anteriores. Estes dados são carregados automaticamente quando a aplicação inicia e apresentados numa lista lateral no LoginForm, permitindo ao utilizador reutilizar acessos anteriores de forma rápida.

Todos os tokens são armazenados encriptados na base de dados. Durante o carregamento, o token é descriptado em memória e utilizado de forma transparente pela aplicação. A lista visível ao utilizador mostra apenas o nome dos dados e o endereço IP, ocultando o token por motivos de segurança.

Quando o utilizador selecciona um dos dados de login da lista, os campos do formulário de

autenticação são automaticamente preenchidos com as informações correspondentes (nome de utilizador, endereço IP e token), facilitando e agilizando o processo de login (**Figura 13**).

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (listBox1.SelectedIndex >= 0 && listBox1.SelectedIndex < devices.Count)
    {
        var selecionado = devices[listBox1.SelectedIndex];

        textBoxUsername.Text = selecionado.Name;
        textBoxIpAddress.Text = selecionado.IpAddress;
        textBoxToken.Text = selecionado.Token;
    }
}
```

**Figura 13:** Método para preencher os dados

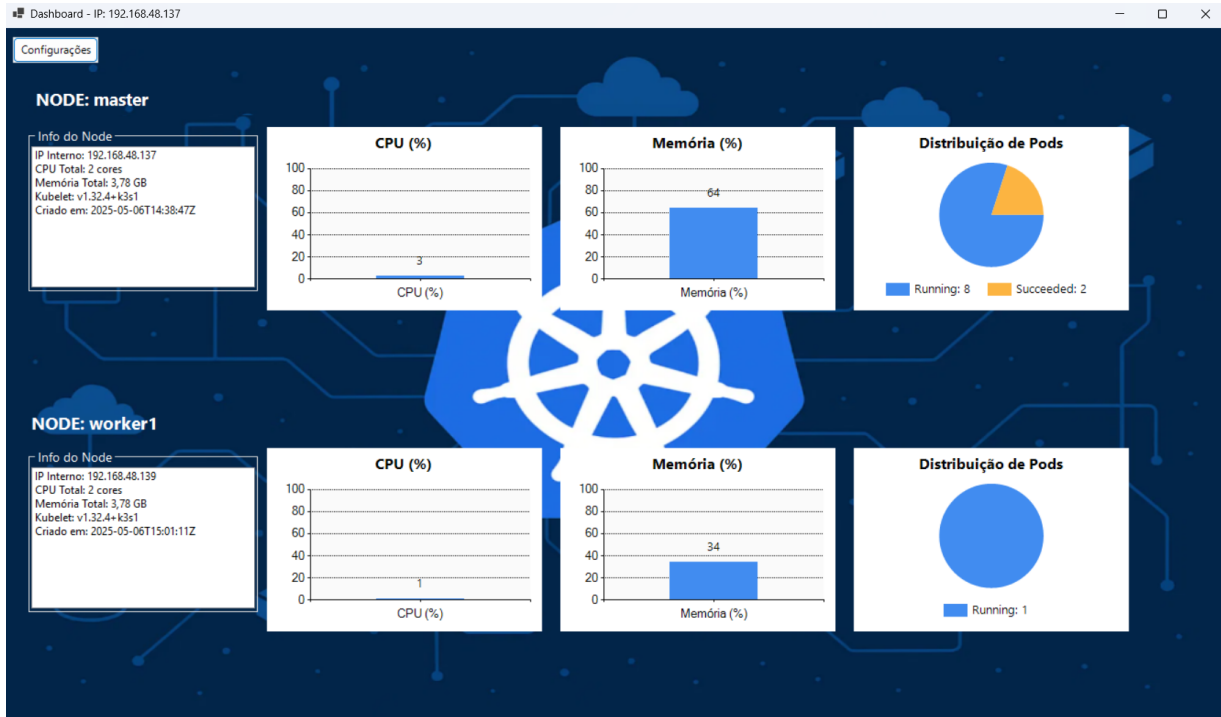
## 5.7 Dashboard

O dashboard apresentado no "DashboardForm" foi construído inteiramente com código C#, sem o uso do *Designer* do Visual Studio. A interface gráfica é gerada dinamicamente durante a execução da aplicação, utilizando controlo manual das posições, tamanhos e estilos dos elementos gráficos.

O objetivo do dashboard é apresentar, de forma visual e resumida, o estado atual de todos os *nodes* do cluster Kubernetes autenticado. Para isso, são realizados periodicamente (a cada 2 segundos) pedidos GET aos seguintes endpoints da API do Kubernetes:

- GET - **/api/v1/nodes**: para obter a informação geral de cada node;
- GET - **/apis/metrics.k8s.io/v1beta1/nodes**: para obter métricas de uso de CPU e memória;
- GET - **/api/v1/pods**: para agrupar os pods por node e estado.





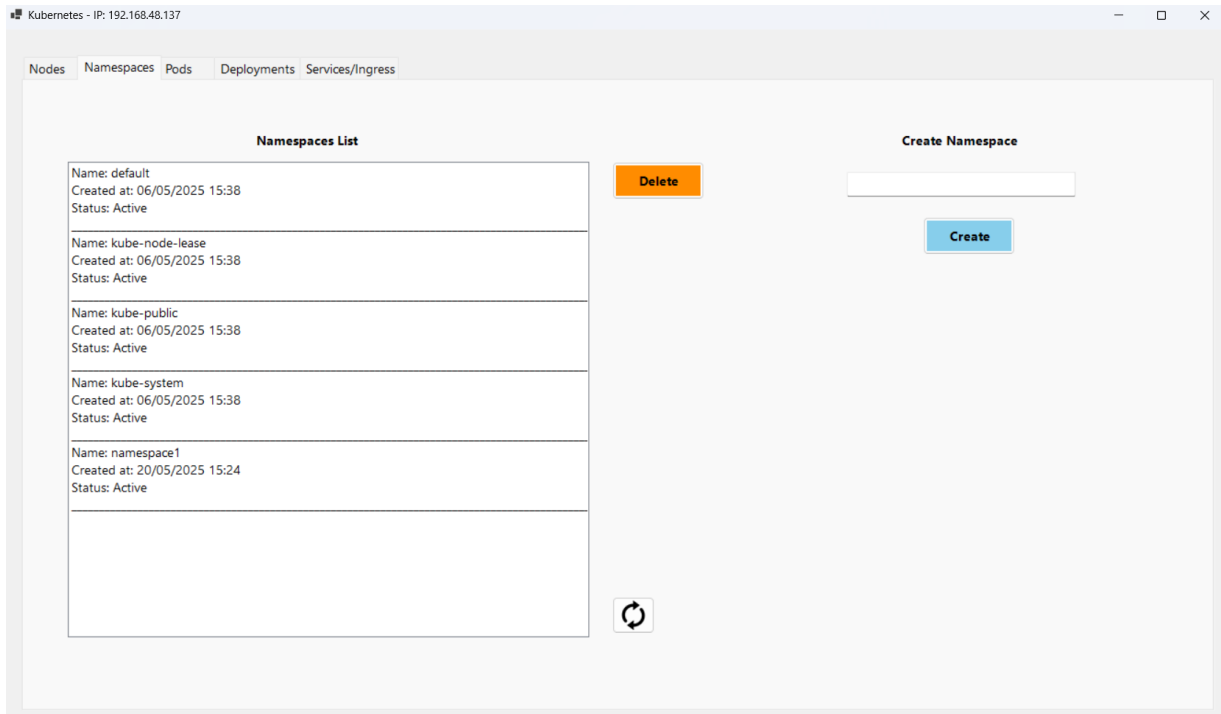
**Figura 14:** Dashboard

Cada node é representado por um painel independente com os seguintes elementos:

- **Informações do node:** IP, número total de CPUs, memória total, versão do kubelet e data de criação do node.
- **Gráficos de barra** com a percentagem atual de utilização de CPU e de memória.
- **Gráfico circular (pie chart)** com a distribuição dos pods do node, de acordo com o seu estado (*Running*, *Succeeded*).

Todos os elementos são construídos manualmente através da classe *Panel*, sendo alinhados de forma centralizada num *FlowLayoutPanel*, que permite o *scroll* vertical em caso de muitos nodes. O botão “Configurações” no topo do formulário permite aceder ao formulário de gestão de recursos adicionais (MainForm), sendo o ”DashboardForm” temporariamente ocultado.





**Figura 16:** *TabPage* Namespace

### 5.9.1 Listar

A lista de namespaces é apresentada numa *listbox* da secção "Namespaces List", sendo gerada automaticamente através de um pedido GET ao endpoint `/api/v1/namespaces` da API do Kubernetes. A resposta obtida é processada para extrair os principais dados de cada namespace, como o nome, a data de criação e o estado atual.

Em todas as *listboxes* existe um botão para atualizar as listas.

### 5.9.2 Criar

Para criar um novo namespace, o utilizador introduz o nome no campo de texto da secção "Create Namespace" e clica no botão "Create". Em seguida, é enviado um pedido POST ao endpoint `/api/v1/namespaces` da API do Kubernetes.

### 5.9.3 Eliminar

O utilizador pode eliminar namespaces existentes no cluster Kubernetes. Para isso, deve seleccionar o namespace pretendido directamente na *listBox* e clicar no botão "Delete". O sistema valida se a linha seleccionada corresponde ao nome do namespace e, caso seja válida, envia um pedido DELETE ao endpoint `/api/v1/namespaces/nome` da API do Kubernetes.

## 5.10 Pods

Na *TabPage* "Pods", o utilizador tem a possibilidade de **listar**, **criar** e **eliminar** pods no cluster Kubernetes.

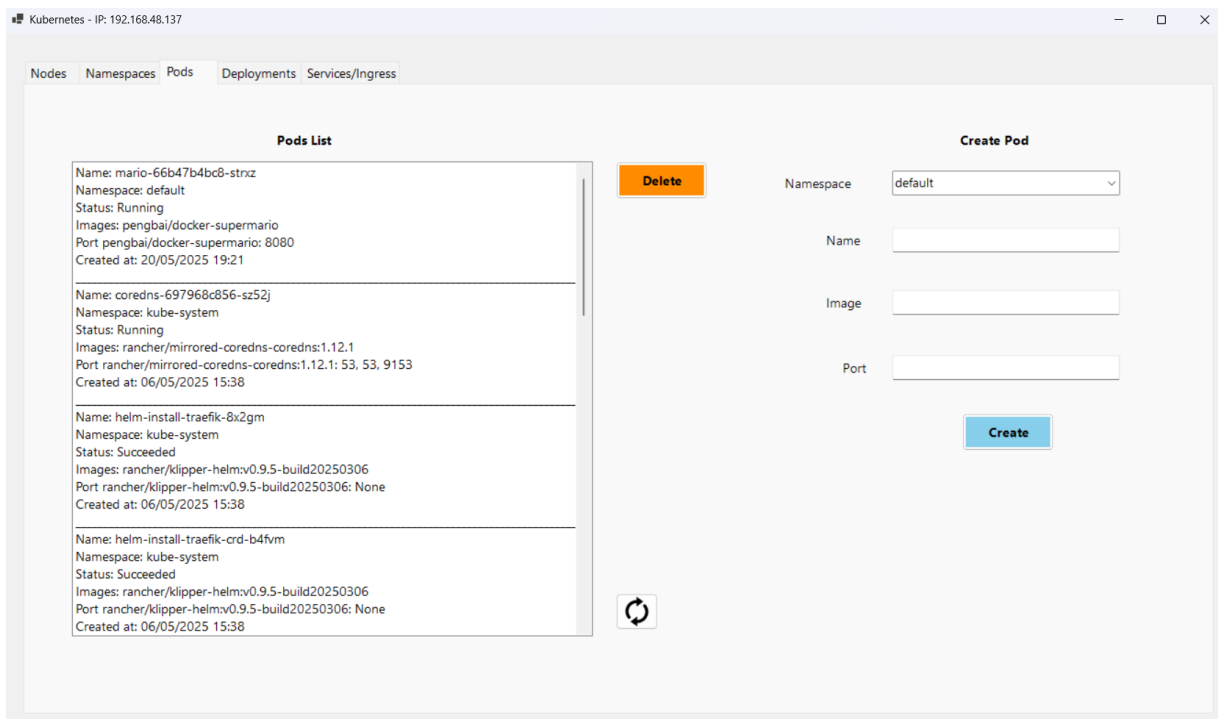


Figura 17: *TabPage* Pods

#### 5.10.1 Listar

A lista de pods é apresentada na *listBox* da secção "Pods List", sendo gerada automaticamente através de um pedido GET ao endpoint `/api/v1/pods` da API do Kubernetes. A resposta

obtida é processada para extrair os principais dados de cada pod, nomeadamente o nome, o namespace, o estado atual, a data de criação, as imagens utilizadas e as respetivas portas expostas.

### 5.10.2 Criar

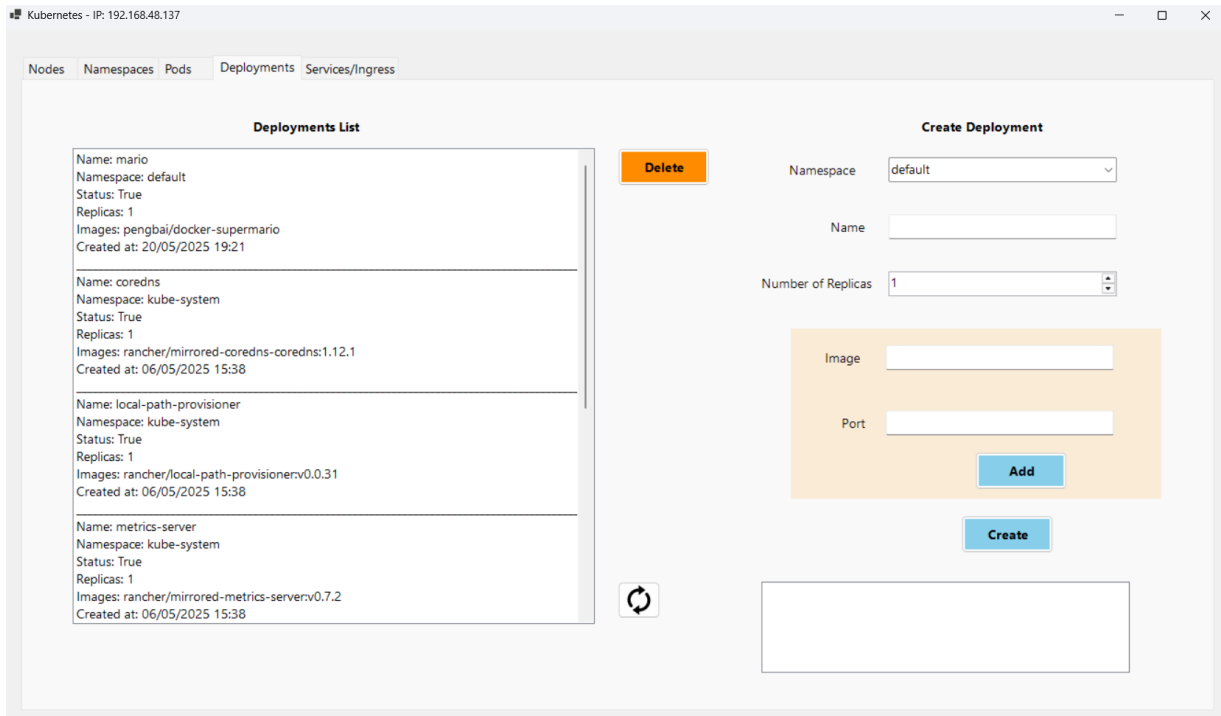
Para criar um novo pod, o utilizador deve preencher os campos com o nome do pod, a imagem a utilizar, o namespace (selecionado através da combobox) e, opcionalmente, a porta. Após clicar no botão "Create", é feito um pedido à API do Kubernetes para lançar o pod no namespace indicado. Caso a operação seja bem-sucedida, o novo pod é criado e a *listBox* é atualizada automaticamente.

### 5.10.3 Eliminar

Para eliminar um pod, o utilizador deve selecionar, na *listBox*, a linha correspondente ao nome do pod que pretende remover e clicar no botão "Delete". A aplicação verifica se a seleção é válida e identifica o namespace associado ao pod. Em seguida, é feito um pedido à API do Kubernetes para eliminar o pod no namespace correspondente. Caso a operação seja concluída com sucesso, o pod é removido do cluster e a *listBox* é atualizada automaticamente.

## 5.11 Deployments

Na *TabPage* "Deployments", o utilizador tem a possibilidade de **listar**, **criar** e **eliminar** deployments no cluster Kubernetes.



**Figura 18:** *TabPage* Deployments

### 5.11.1 Listar

A lista de deployments é apresentada na *listBox* da secção "Deployments List", sendo gerada automaticamente através de um pedido GET ao endpoint `/apis/apps/v1/deployments` da API do Kubernetes. A resposta obtida é processada para extrair os principais dados de cada deployment, como o nome, o namespace, o estado, o número de réplicas definidas, as imagens utilizadas e a data de criação.

### 5.11.2 Criar

Para criar um deployment, o utilizador deve preencher os campos com o nome do deployment, o namespace de destino e o número de réplicas desejado. De seguida, pode adicionar um ou mais containers à lista, indicando a imagem e a porta correspondente, através do botão "Add". Cada container adicionado é apresentado na *listbox* abaixo do formulário a laranja. Após adicionar todos os containers necessários, o utilizador clica no botão "Create", o que origina um

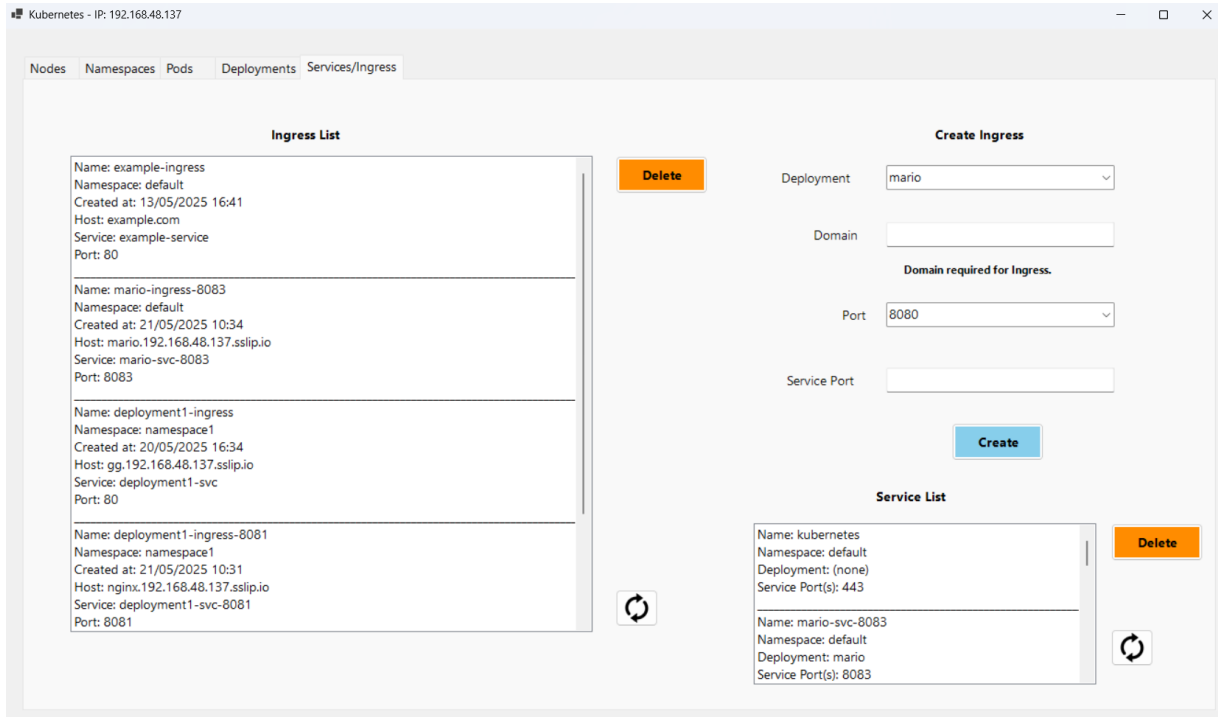
pedido POST ao endpoint `/apis/apps/v1/namespaces/namespace/deployments` da API do Kubernetes, com os dados introduzidos. Se a operação for bem-sucedida, o deployment é criado e a interface é atualizada automaticamente. Esta funcionalidade permite criar deployments com múltiplos containers de forma simples.

### 5.11.3 Eliminar

Para eliminar um deployment, o utilizador deve seleccionar, na *listbox* da secção "Deployments List", a linha correspondente ao nome do deployment que pretende remover. Após validação da seleção, a aplicação identifica o namespace associado e envia um pedido DELETE ao endpoint `/apis/apps/v1/namespaces/namespace/deployments/deployment` da API do Kubernetes. Se a operação for concluída com sucesso, o deployment é removido do cluster e a lista é atualizada automaticamente.

## 5.12 Services/Ingress

Na *TabPage* "Services/Ingress", o utilizador tem a possibilidade de **listar, criar e eliminar** services/ingress no cluster Kubernetes.



**Figura 19:** *TabPage Services/Ingresss*

### 5.12.1 Listar

A lista de ingressos é apresentada na *listBox* da secção "Ingress List", sendo gerada automaticamente através de um pedido GET ao endpoint `/apis/networking.k8s.io/v1/ingresses` da API do Kubernetes. A resposta é processada para extrair os principais dados de cada ingresso, como o nome, o namespace, a data de criação, o domínio (host), o serviço associado e a respetiva porta.

### 5.12.2 Criar

Na *TabPage* "Services/Ingress", o utilizador pode criar um service e, opcionalmente, um ingress associado a um deployment existente. Para isso, deve seleccionar o deployment desejado, indicar a porta do container e a porta que será exposta no serviço. Se pretender expor a aplicação, pode também preencher o campo do domínio.

Ao clicar no botão "Create", a aplicação envia um pedido HTTP do tipo POST para o end-



point **/api/v1/namespaces/namespace/services**, com os dados do serviço em formato YAML. Caso o domínio esteja preenchido, é enviado um segundo pedido POST para o endpoint **/apis/-networking.k8s.io/v1/namespaces/namespace/ingresses**, criando assim um Ingress associado ao serviço.

### 5.12.3 Eliminar

Na secção "Services/Ingress", o utilizador pode eliminar tanto um recurso do tipo service como um recurso do tipo ingress. Para remover um service, basta seleccionar a linha correspondente ao seu nome na lista "Service List" e clicar no botão "Delete". É então enviado um pedido HTTP do tipo DELETE para o endpoint **/api/v1/namespaces/namespace/services/serviceName** da API do Kubernetes.

No caso dos ingresses, o processo é semelhante, mas ao eliminar um Ingress, o serviço associado também é removido automaticamente. Após a confirmação do utilizador, são enviados dois pedidos DELETE: um para o endpoint **/apis/-networking.k8s.io/v1/namespaces/namespace/ingresses/ingressName** e outro para **/api/v1/namespaces/namespace/services/serviceName**.

## 6 Testes

Neste capítulo são descritos dois testes realizados para validar o correto funcionamento das funcionalidades adicionais implementadas no projeto, autenticação por token e no armazenamento dos logins anteriores na base de dados. Para cada uma destas funcionalidades, são apresentados os objetivos dos testes, os passos executados e os resultados obtidos, de forma a assegurar que os requisitos definidos foram devidamente cumpridos.

Os restantes testes, referentes às demais funcionalidades implementadas, encontram-se demonstrados no vídeo em anexo.

### 6.1 Autenticação por Token

#### Objetivos do teste

- Verificar se a geração de tokens de acesso ao cluster é realizada com sucesso através do `kubectl`.
- Confirmar que a aplicação aceita o token como forma válida de autenticação.
- Garantir que, após a inserção do token, o utilizador consegue aceder à aplicação sem introduzir credenciais adicionais.

O teste teve início com a criação de um token de autenticação através do seguinte comando executado no terminal do node master:

```
sudo kubectl create token default
```

Este comando gerou um token, como ilustrado na **figura 20**. O token foi então copiado e inserido manualmente na interface de login da aplicação, substituindo a necessidade de introdução de utilizador e palavra-passe.

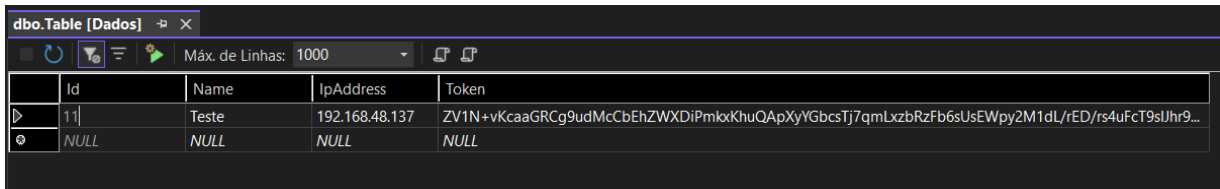


## 6.2 Armazenamento de logins na base de dados

### Objetivos do teste

- Verificar se os dados de login (nome de utilizador e IP) são corretamente armazenados após cada sessão, incluindo o token, se este é armazenado de forma criptográfica.
- Confirmar que os logins anteriores são apresentados ao utilizador nas sessões seguintes.
- Assegurar que os dados guardados são persistentes e reutilizáveis em futuras autenticações.

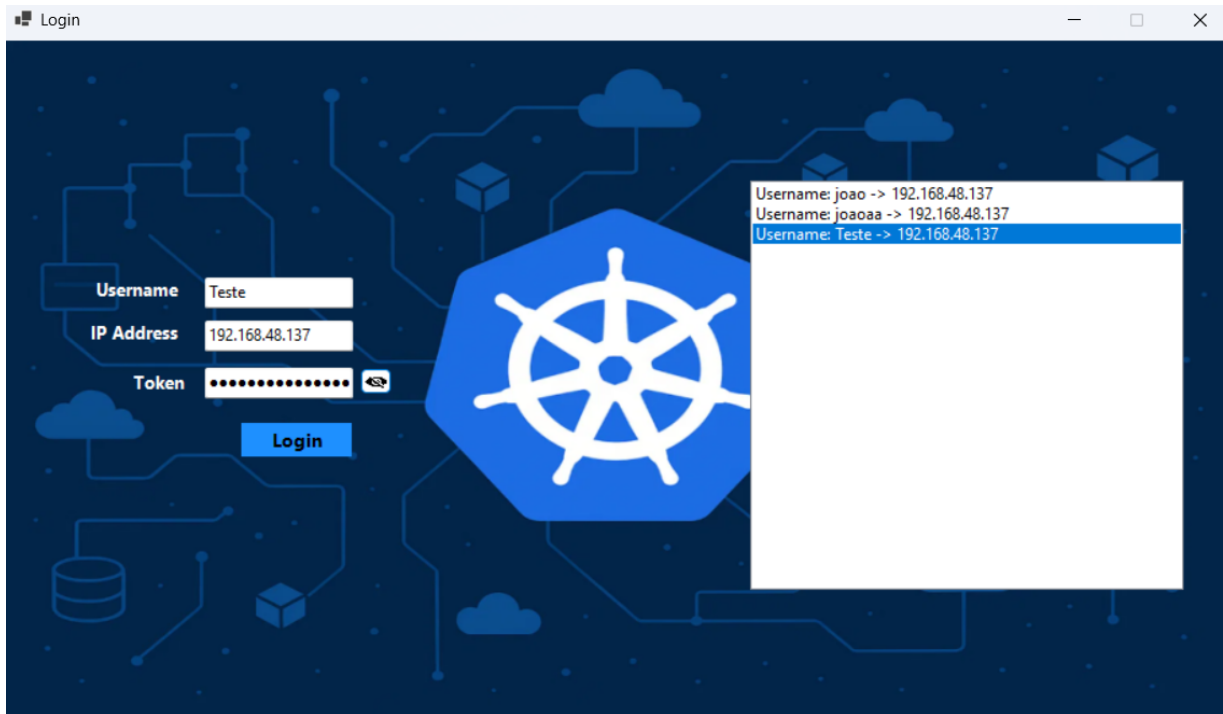
Para testar esta funcionalidade, foram realizados múltiplos logins na aplicação utilizando diferentes nomes de utilizador. Após cada autenticação, verificou-se que o nome do utilizador, o respetivo endereço IP e o token de autenticação eram armazenados localmente. Confirmou-se que o token era guardado de forma criptográfica (**fig 22**), garantindo a segurança dos dados sensíveis.



Id	Name	IpAddress	Token
11	Teste	192.168.48.137	ZV1N+vKcaaGRCg9udMcCbEhZWXDiPmkxKhuQApXyYGbcsTj7qmLxzbRzFb6sUsEWpy2M1dL/rED/rs4uFcT9sIjhr9...
NULL	NULL	NULL	NULL

**Figura 22:** Token criptografado

Na abertura seguinte da aplicação, foi possível observar a presença desses registos numa interface dedicada ao histórico de logins (**fig 23**), permitindo a sua reutilização. O que confirmou que os dados de login são corretamente guardados.



**Figura 23:** Histórico de logins

## 7 Análise crítica e proposta de melhorias

De forma geral, o desenvolvimento deste trabalho decorreu de forma satisfatória, tendo sido possível alcançar os objetivos inicialmente propostos. Todas as funcionalidades planeadas foram implementadas com sucesso, nomeadamente a autenticação por token, a visualização e gestão de recursos do cluster Kubernetes (nodes, namespaces, pods, deployments, services e ingresses), bem como o armazenamento dos logins anteriores. Este processo permitiu não só consolidar conhecimentos previamente adquiridos, como também explorar novas abordagens e ferramentas relacionadas com a orquestração de containers e comunicação com APIs REST.

Apesar dos resultados positivos, é importante reconhecer que existem sempre oportunidades de melhoria. Nesse sentido, alguns dos próximos passos que se poderiam considerar incluem a implementação de novas funcionalidades, como a edição de recursos existentes, o aperfeiçoamento do tratamento de erros e validações na interface, e uma eventual adaptação da aplicação para funcionar em sistemas operativos além do Windows, uma vez que atualmente está limitada a esse ambiente.

## 8 Conclusão

Concluindo, o desenvolvimento da aplicação para gestão de clusters Kubernetes foi bem-sucedido, tendo alcançado os objetivos propostos. A ferramenta desenvolvida permite ao utilizador interagir com os principais recursos do cluster de forma simples e intuitiva, suportando operações como a criação, listagem e eliminação de namespaces, pods, deployments, services e ingressos, bem como a autenticação através de token e o registo de logins anteriores.

Todos os testes realizados confirmaram o funcionamento adequado das funcionalidades implementadas, demonstrando a fiabilidade da aplicação na comunicação com a API do Kubernetes. O trabalho cumpriu integralmente a proposta inicial, resultando numa solução prática e eficaz para a gestão de ambientes Kubernetes em contexto local.

## Bibliografia

- [1] IBM. What is Kubernetes?, 2024, <https://www.ibm.com/think/topics/kubernetes>.
- [2] Dockerdocs. What is Docker?, 2025, <https://docs.docker.com/get-started/docker-overview/>
- [3] Tailscale. What are Nodes?, 2025, <https://tailscale.com/learn/what-are-nodes>
- [4] Vmware. What are Kubernetes Pods?, 2025, <https://www.vmware.com/topics/kubernetes-pods>
- [5] IBM. What are containers?, 2024, <https://www.ibm.com/think/topics/containers>
- [6] kubernetes. Service, 2025, <https://kubernetes.io/docs/concepts/services-networking/service/>
- [7] kubernetes. Images, 2025, <https://kubernetes.io/docs/concepts/containers/images/>
- [8] kubernetes. Kubernetes Deployments, 2025, <https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/>
- [9] kubernetes. Cluster Architecture, 2025, <https://kubernetes.io/docs/concepts/architecture/>
- [10] kubernetes. Ingress, 2025, <https://kubernetes.io/docs/concepts/services-networking/ingress/>
- [11] kubernetes. Namespaces, 2025, <https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>
- [12] K3S-Doc. Quick-Start Guide, 2025, <https://docs.k3s.io/quick-start>
- [13] Minikube. Minikube start, 2025, <https://minikube.sigs.k8s.io/docs/start/?arch=%2Fwindows%2Fx86-64%2Fstable%2F.exe+download>
- [14] Kind. Kind documentation, 2025, <https://kind.sigs.k8s.io/>
- [15] Microk8s. MicroK8s documentation, 2025, <https://microk8s.io/>
- [16] ChatGPT. 2025, <https://chatgpt.com/>



## **Anexos**

Em anexo, encontra-se um vídeo com a demonstração dos testes realizados para a validação das funcionalidades implementadas.

Vídeo: <https://www.youtube.com/watch?v=UdofIDfRhA>