Explicação Geral do Funcionamento do Sistema SafeCloud

O projeto **SafeCloud** foi desenvolvido com o objetivo de criar um sistema completo e seguro para o armazenamento e recuperação de ficheiros, distribuindo os dados de forma redundante e segura por diferentes bases de dados locais e remotas. Trata-se de uma aplicação com uma arquitetura moderna e modular, que envolve várias camadas — interface gráfica (frontend), lógica de aplicação (backend), armazenamento (bases de dados SQLite e MySQL), infraestrutura em containers (Docker e Kubernetes), e até mesmo segurança com certificados digitais emitidos por uma Autoridade de Certificação própria.

Do ponto de vista do utilizador, o sistema começa com uma interface web desenvolvida em HTML, CSS e Bootstrap, acessível através de páginas como index.html, login.html, register.html, dashboard.html, upload.html e download.html. Estas páginas foram desenhadas com uma estética moderna e responsiva, com foco na usabilidade. O utilizador pode registar-se, fazer login e, a partir do painel (dashboard), fazer upload e download de ficheiros com segurança.

Quando o utilizador faz upload de um ficheiro, esse ficheiro é dividido em várias partes (chunks), graças a um algoritmo de fragmentação implementado em Python. Este algoritmo calcula o tamanho de cada fragmento e aplica "padding" (preenchimento) quando necessário para garantir que todos os fragmentos têm tamanhos equilibrados. Cada fragmento é então armazenado em bases de dados diferentes: duas em SQLite locais (cloud1.db e cloud2.db) e duas em MySQL (cloud3 e cloud4), as quais correm em containers separados no Docker, orquestrados via docker-compose.

Além da fragmentação, o sistema também conta com uma funcionalidade de reconstrução, que permite ao utilizador descarregar (rebuild) o ficheiro original a partir dos fragmentos armazenados. A reconstrução pode ser feita diretamente no frontend através de um formulário em download.html, e o backend garante que apenas ficheiros válidos e previamente armazenados podem ser reconstruídos, evitando pedidos inválidos ou de ficheiros inexistentes.

A infraestrutura do sistema foi cuidadosamente construída para simular um ambiente de produção. A aplicação corre num container Docker com Python e Flask, enquanto as bases de dados MySQL estão em containers separados. Além disso, foi criado um cluster Kubernetes (em simulação) para organizar serviços como o NGINX, que está preparado para atuar como servidor web e proxy reverso, encaminhando pedidos dos utilizadores para os serviços internos.

A segurança é outro pilar central do projeto. Foi criada uma **Autoridade de Certificação (CA)** em Python, capaz de emitir, revogar e validar certificados digitais. Esses certificados garantem que os serviços e utilizadores autenticados são legítimos. Os certificados emitidos são armazenados numa base de dados própria, e podem ser visualizados e geridos através de um menu interativo. Uma VPN também foi configurada para simular uma rede segura, ligando os componentes internos do cluster e restringindo o acesso externo.

Integração Técnica e Académica: Como o Sistema Aplica os Conteúdos das Três Cadeiras

O sistema desenvolvido tem como principal objectivo fornecer uma solução segura, distribuída e eficiente para o armazenamento de ficheiros na cloud, com funcionalidades completas de gestão de utilizadores, partilha de ficheiros, controlo de segurança e reconstrução de dados. Este sistema foi concebido de raiz para integrar e aplicar conceitos fundamentais de **Criptografia Aplicada**, **Programação para a Internet** e **Sistemas Distribuídos e Segurança**.

A estrutura do sistema está dividida em várias camadas, com responsabilidades bem definidas. Na camada de interface, o **frontend** foi desenvolvido com HTML e CSS e oferece aos utilizadores um ambiente web intuitivo com páginas de login, registo, painel de controlo, upload e download de ficheiros. Este frontend é servido através de um servidor **NGINX** que actua como proxy reverso, permitindo que os pedidos dos utilizadores sejam encaminhados correctamente para o backend da aplicação.

No backend, foi utilizado o framework **Flask (Python)**, responsável por toda a lógica da aplicação. Aqui são geridas as rotas HTTP, a autenticação, o processamento de ficheiros, a comunicação com as bases de dados e os sistemas de segurança. Quando o utilizador faz upload de um ficheiro, o backend divide esse ficheiro em quatro partes (chunks) com tamanhos uniformizados, utilizando padding quando necessário. Estes chunks podem ser **encriptados** para garantir a confidencialidade da informação antes de serem armazenados.

Os fragmentos resultantes são enviados para quatro bases de dados diferentes: duas utilizam **SQLite** (armazenadas localmente) e duas utilizam **MySQL** (em containers independentes). Estas bases de dados não comunicam entre si — **apenas o backend conhece as quatro localizações** e gere a escrita e leitura de dados em cada uma. Esta decisão arquitectural segue os princípios dos **sistemas distribuídos**, onde a redundância e a dispersão dos dados aumentam a segurança e a tolerância a falhas. A separação entre tipos de base de dados também demonstra flexibilidade na integração de múltiplas tecnologias.

Um dos componentes mais importantes do sistema é a **Autoridade de Certificação (CA)** criada em Python, que permite a emissão, revogação e validação de certificados digitais. Este sistema oferece uma interface interactiva para gerir certificados emitidos, que são armazenados numa base de dados dedicada. Estes certificados são fundamentais para garantir comunicações seguras entre serviços, especialmente entre os componentes que residem dentro do **cluster Kubernetes**. Este cluster foi configurado para alojar diferentes partes do sistema (como o backend, o frontend, o NGINX e os serviços de base de dados), utilizando um **ConfigMap** para parametrização e organização dos pods e serviços internos.

Para garantir uma ligação segura entre os serviços, foi também configurada uma **rede VPN**, que assegura que a comunicação entre o backend, as bases de dados e a autoridade de certificados seja protegida contra-ataques externos. A combinação da VPN com os certificados digitais garante a **autenticação mútua e a confidencialidade** dos dados em trânsito.

Na fase de download, o utilizador pode solicitar a reconstrução de um ficheiro previamente armazenado. O backend é responsável por recuperar os quatro chunks das bases de dados, reverter a encriptação, remover o padding e juntar os dados para formar novamente o ficheiro original. O sistema valida o sucesso da operação e disponibiliza o ficheiro final para o utilizador através da interface web.

Esta solução demonstra um forte alinhamento com os conteúdos curriculares das três disciplinas envolvidas:

Criptografia Aplicada

O sistema implementa encriptação simétrica para proteger os ficheiros antes do armazenamento, garantindo confidencialidade. Além disso, foi desenvolvida uma **Autoridade de Certificação (CA)** capaz de emitir, revogar e gerir certificados digitais, permitindo autenticação de serviços e comunicação segura entre componentes, com registo em base de dados.

Programação para a Internet

Foi construído um **frontend em HTML/CSS** e um **backend em Flask**, com rotas que permitem login, registo, upload e download de ficheiros. O backend gere a lógica de negócio, interage com as bases de dados e aplica segurança, refletindo boas práticas do desenvolvimento web moderno.

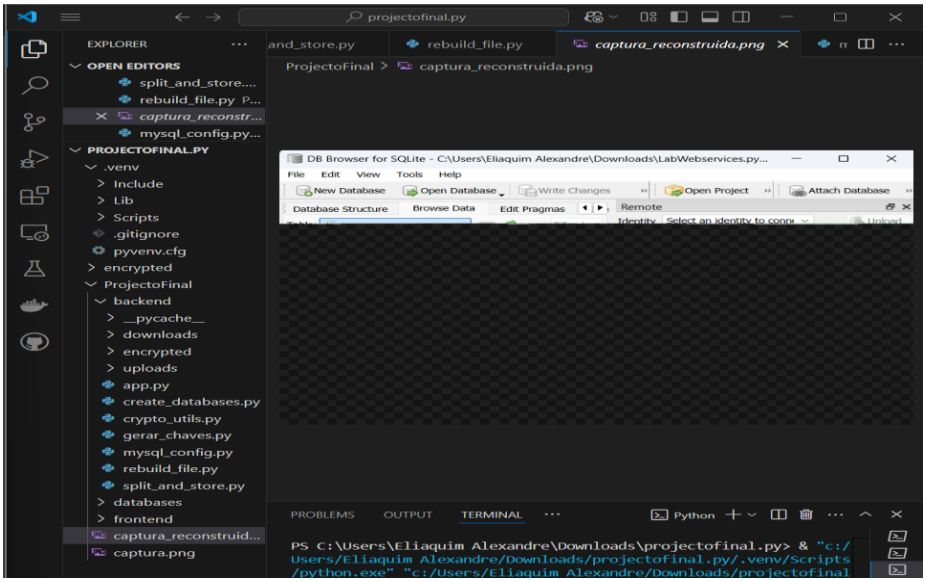
Sistemas Distribuídos e Segurança

A arquitectura usa **containers Docker** orquestrados por **Kubernetes**, com os dados distribuídos por quatro bases de dados independentes. Foi criada uma **VPN privada** para proteger as comunicações internas, e um **ConfigMap** organiza o cluster. A distribuição dos chunks e o isolamento dos serviços demonstram uma abordagem prática à computação distribuída e segura.

----------Imagens extras----------

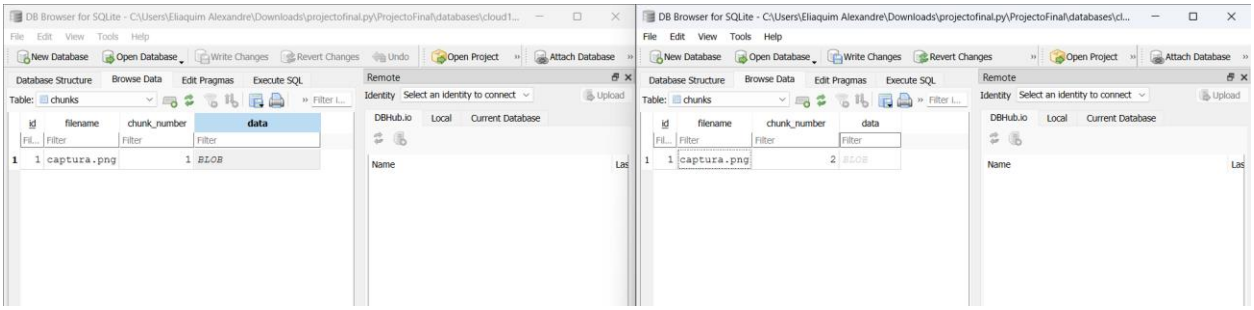
```
PS C:\Users\Eliaquim Alexandre\Downloads\projectofinal.py\ProjectoFinal> py backend/split_and_store.py captura.png
  Chunk 1 gravado em cloud1.db (SQLite)
  Chunk 2 gravado em cloud2.db (SQLite)
  Chunk 3 gravado em cloud3 (MySQL)
  Chunk 4 gravado em cloud4 (MySQL)
\ Ficheiro dividido e armazenado com sucesso!

PS C:\Users\Eliaquim Alexandre\Downloads\projectofinal.py\ProjectoFinal> py backend/rebuild_file.py
 SQLite (cloud1.db) - Chunk 1: 12983 bytes
 SQLite (cloud2.db) - Chunk 2: 12983 bytes
  MySQL (cloud3) - Chunk 3: 12983 bytes
  MySQL (cloud4) - Chunk 4: 12986 bytes

 Total chunks encontrados: 4

Ficheiro 'captura.png' reconstruído com sucesso em 'C:\Users\Eliaquim Alexandre\Downloads\projectofinal.py\ProjectoFinal\captura_reconstruida.png'!
PS C:\Users\Eliaquim Alexandre\Downloads\projectofinal.py\ProjectoFinal>
```

Teste realizado para validar o funcionamento do processo de divisão e reconstrução de ficheiros. O ficheiro captura.png foi dividido em quatro partes (chunks) e armazenado com sucesso em duas bases de dados SQLite e duas MySQL. Em seguida, foi reconstruído com sucesso como captura_reconstruida.png, confirmando que a lógica de fragmentação, armazenamento e recuperação estava a funcionar corretamente nesta fase de testes.

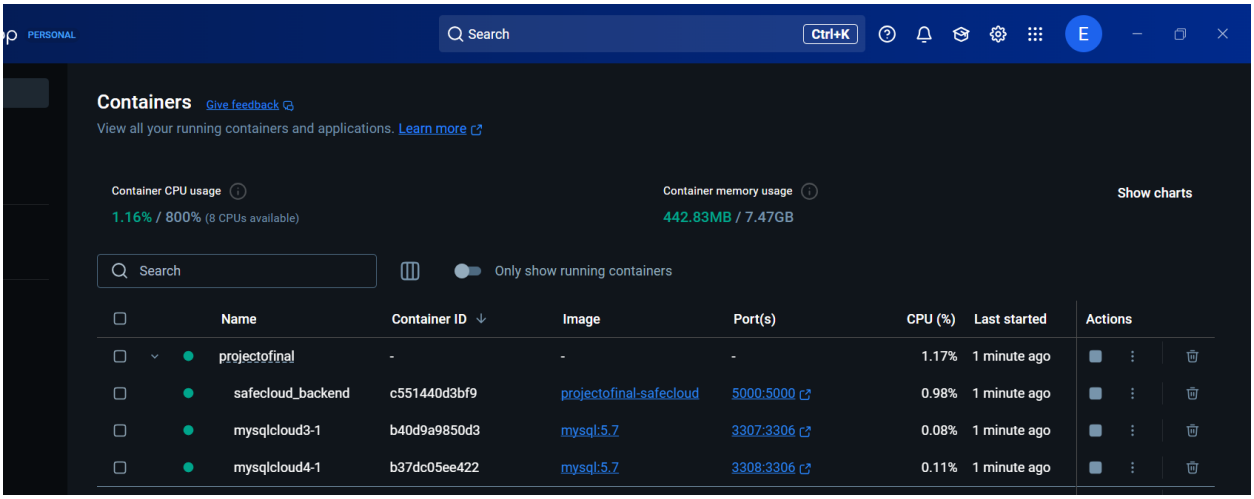


Nesta imagem, observa-se o ficheiro captura_reconstruida.png aberto no VS Code após uma tentativa de reconstrução do ficheiro original captura.png. O ficheiro apresenta-se corrompido (área preta), evidenciando que apenas parte dos dados foi recuperada com sucesso. Este erro deveu-se à indisponibilidade das bases de dados MySQL (cloud3 e cloud4), que não estavam a comunicar corretamente com o backend. As causas possíveis incluíam falhas nos containers, ausência das tabelas ou má configuração do ficheiro mysql_config.py.

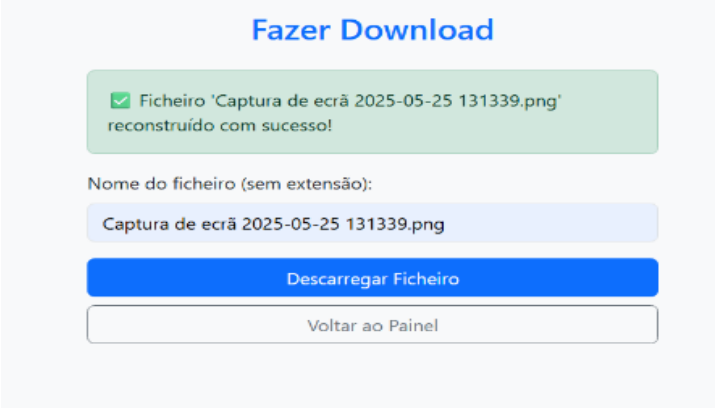
A resolução passou pela reconexão dos containers MySQL via Docker (confirmando portas e serviços ativos) e pela criação manual das tabelas chunks com a estrutura correta. Após estas correções, o backend passou a aceder aos quatro fragmentos (2 em SQLite e 2 em MySQL), permitindo a reconstrução completa e válida do ficheiro.



Visualização dos chunks 1 e 2 do ficheiro captura.png armazenados com sucesso nas bases de dados SQLite cloud1.db e cloud2.db.



visualização dos containers Docker em execução, incluindo o backend, dois serviços MySQL e o frontend via safecould_backend



Interface de download a confirmar a reconstrução bem-sucedida do ficheiro e disponibilizá-lo para descarregamento.