Kubernetes e MiniKube

João Carneiro
Dep. de Informática
UBI
Covilhã, Portugal
joao.m.carneiro@ubi.pt 50938

João Pinto
Dep. de Informática
UBI
Covilhã, Portugal
joao.d.pinto@ubi.pt 49889

Leonor Moreira
Dep. de Informática
UBI
Covilhã, Portugal
brito.moreira@ubi.pt 50877

I. Palavras Chave

Microsserviços, Orquestração, Kubernetes, Docker, Node.js, PostgreSQL, Minikube.

II. Siglas e Acrónimos

Sigla/Acrónimo	Definição
REST	Representational State Transfer
NPM	Node Package Manager
ACID	Atomicity, Consistency, Isolation and Durability
ORM	Object-Relational Mapping

III. Abstrato

Este documento aprofunda o desenvolvimento de um sistema de gestão de tarefas, utilizando uma arquitetura baseada em microsserviços, e cumprindo os requisitos e funcionalidades da mesma. O sistema foi desenvolvido a pensar nos conceitos de escabilidade, flexibilidade e resiliência, construir serviços independentes para a autenticação do utilizador, gerir tarefas, utilizadores e notificações. Foi utilizado *Node.js* com *JavaScript* para o *backend*, *PostgreSQL* com *Sequelize* para a persistência de dados, e o *Docker* para a *containerização* dos serviços. Crucialmente, o *Kubernetes* (via *Minikube)*, orquestrou, localmente, e automatizou o *deployment* e a gestão dos serviços.

O projeto demonstra a aplicação prática dos princípios dos micrisserviços, realçando o quanto a *containerização* e orquestração de serviços é a solução mais robusta no desenvolvimento de sistemas escaláveis, distribuídos e disponíveis.

Em conjunto com a abordagem, a execução desta aplicação, também valida a eficiência de tecnologias modernas para criar *softwares* mais complexos, que podem, facilmente, responder às necessidaades da gestão de tarefas.

IV. Introdução

A. Âmbito e Enquadramento

Neste documento consta o relatório de projeto final, para a Unidade Curricular de Arquitetura e Desenvolvimento de Microsserviços. O tema do estudo laboral é o desenvolvimento de um sistema de gestão de tarefas com aequiteturas de microsserviços.

Atualmente, o software moderno exige características de escalabilidade, flexibilidade e resiliência, para tal, a aprendizagem do desenvolvimento de microserviços e as suas ferramentas, é crucial para profissionais na área de informática. Ao longo do desenvolvimento deste projeto, utilizamos ferramentas e aplicamos técnicas e conhecimento que foram aprendidas durante as aulas do semestre.

B. Objetivo

O objetivo deste projeto final, é o desenvolvimetno de um sistema para gestão de tarefas. O sistema deve ser composto por um conjunto de microsserverços para a gestão de diferentes serviços, os quais, a autenticação, gestão de utilizadores, gestão de tarefas e notificações. Para tal, devem ser aplicados os conhecimentos obtidos durante o semestre, assim como deve haver colaboração entre o grupo de colegas.

C. Abordagem

Para realizar o projeto e concluir todos os seus requisitos, utilizamos algumas aplicações, técnicas e abordagens. O *VS-Code* para editar o código fonte, o *Docker* para gerir os *containers*, onde constam os serviços e o *Kubernetes* para orquestrar os mesmos. Utilizamos a linguagem de programação *JavaScript* e o *package manager NodeJs* para desenvolver o *back-end* do projeto e *SQL* para gerir a base de dados. A abordagem do sistema é distribuída, pelo que a sua arquitetura é baseada em microsserviços, com recurso às técnicas de *containerização*, implementado com orquestração.

D. Organização do Documento

1) **Palavras Chave**
2) **Siglas e Acrónimos**
3) **Abstrato**
4) **Introdução**
5) **Conceitos da síntese**
6) **Engenharia de *Software***
7) **Execução e Desenvolvimento**
8) **Testes e Validação**
9) **Conclusões e Trabalho Futuro**
10) **Referências**

V. Conceitos da síntese

A. Introdução

Os seguintes conceitos fazem parte do tema desenvolvido neste trabalho de síntese.

B. *Arquitetura baseada em Microsserviços*

Uma abordagem de desenvolvimento de *software*, onde uma aplicação é construída como uma coleção de serviços pequenos, independentes e fracamente acoplados.

C. *Orquestração*

Refere-se a um padrão de integraçãoonde um serviços centreal, coordena e gere o fluxo de trabalho entre múltiplos microsserviços, para realizar uma operação de negócio complexa.

VI. Engenharia de *Software*

A. *Introdução*

B. *Engenharia de Software*

Antes de começar o desenvolvimento do projeto, analisámos quais os seus requisitos, e realizados um estudo de como seria a arquitetura do sistema distribuído. Nesta secção, exibimos as nossas conclusões no que toca à engenharia de *software* do sistema.

- **Requisitos funcionais:** Identificamos que os requisitos funcionais para este projeto seriam um sistema que gerisse tarefas, e um sistema com uma arquitetura baseada em microsserviços, para tal função.
- **Requisitos não funcionais:** Os requisitos não funcionais, consistem no uso do estilo de arquitetura ***REST***, comunicações assíncronas, *deployment* e *containerização*, com o *Kubernetes* e o *Docker*, respetivamente, e o uso de base de daods à nossa escolha (PostGreSQL).
- **Arquitetura do sistema:** A arquitetura que aqui expomos, é clássica de um sistema distribuído.

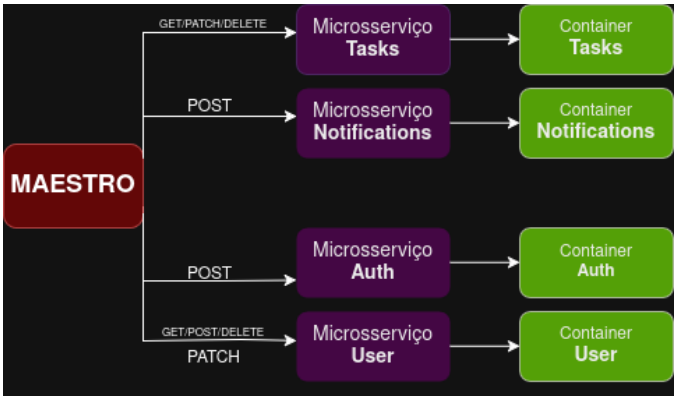

Fig. 1. Arquitetura

C. *Tecnologias Importantes*

- ***Node.js:*** O Node.js é útil para desenvolver um sistema com arquitetura baseada em microsserviços devido à sua natureza assíncrona, que permite gerir muitos pedidos em tempo-real, eficientemente. Além disto o seu vasto ecossistema de pacotes , acelera o desenvolvimento, já que utilizamos o
- ***JavaScript:*** O *JavaScript* unifica tecnologicamente o projeto, já que utilizamos o *Node.js*, o que facilita o desenvolvimento.
- ***PostgreSQL:*** A importânica do *PostgresSQL* neste projeto, reside na sua rebustz e extenso conjunto de funcionalidades que são ideais para a arquitetura do sistema. Entre estas funconalidades, encontra-se o suporte de diversos formatos de dados, e a sua conformidade ao conceito ***ACID***.
- ***Docker:*** O *Docker* permite encapsular os microsserviços em containers isolados. Ambos o *Docker* e o *Kubernetes* eram tecnologias requiridas para o desenvolvimento deste projeto.
- ***Kubernetes:*** Atua como um sistema operativo para os containers e é essencial para a gestão automatizada dos microsserviços e para a abordagem de orquestração.
- ***Sequelize***: Um ORM que suporta *PostgreSQL* que permite desenvolver modelos que representam tabelas, as suas associações (que definem as relações entre as tabelas) e métodos de operações.

D. *Conclusão*

Este projeto baseia-se na prática da utilização de tecnologias modernas para uma arquitetura baseada em microsserviços. A escolha do ***Node.js***, ***JavaScript*** no *backend*, assegura um ecossistema de desenvolvimento unificado e eficiente. O uso da base de dados ***PostgreSQL*** complementa a arquitetura, pois assegura a integridade e persistência de dados. As tecnologias *Docker* e *Kubernetes* complementam-se para um bom funcionamento do sistema, e são requisitos deste projeto pois representam o âmbito da prática da Unidade Curricular.

VII. Execução e Desenvolvimento

A. *Introdução*

Nesta secção vai ser elaborado os processos que foram percorridos para desenvolver o projeto. As etapas principais da execução foram, o desenvolvimento de cada serviço, as comunicações entre eles, a comunicação dos serviços com a base de dados, a *containerização* e, por fim, o *deployment* da aplicação.

B. *Dependências de Desenvolvimento*

- **Ambiente de Execução do *Node.js***: Essencial para executar o código *backend* (Microsserviços e Maestro).
- ***Package Manager***: O **npm** é necessário para instalar e gerir as dependências das bibliotecas.
- **Controlo de Versão**: Dependemos do *Git* para gerir o histórico de verões do projeto e colaborar entre equipa.
- **CLI do Docker**: Indispensável aos requisitos do projeto, é necessário para criar a imagens *Docker* do microsserviços, e para gerir os *containers*, localmente.
- **CLI do Minikube**: Também indipensável para executar devidamente os requisitos do projeto. O Minikube é utilizado para iniciar e gerir o *cluster Kubernetes* local, permitindo executar testes de implantação e orquestração dos serviços. É um ambiente que simula a produção.

- **Cliente de Base de Dados**. O *pgAdmin* foi utilizado para interagir diretamente com a base de dados e, durante o desenvolvimento, visualizar e manipular dados.

C. *Dependências de Execução*

- ***Docker Engine***: Executa os *containers* dos microsserviços.
- ***Minikube***: O **Minikube** é um ambiente de desenvolvimento local, essencial para criar um *cluster Kubernetes*, onde os *containers* serão implantados.
- ***Cluster Kubernetes***: A dependência para executar testes robustos da implatação em produção.
- **Base de Dados**: Cada microsserviço que necessite de persistência de dados, necessita de uma instância de base de dados.
- **CLI do Minikube**: Também indipensável para executar devidamente os requisitos do projeto. O Minikube é utilizado para iniciar e gerir o *cluster Kubernetes* local, permitindo executar testes de implantação e orquestração dos serviços. É um ambiente que simula a produção.
- **Cliente de Base de Dados**. O *pgAdmin* foi utilizado para interagir diretamente com a base de dados e, durante o desenvolvimento, visualizar e manipular dados.

D. *Detalhes da Implementação dos Microsserviços*

Nesta secção, constam os detalhes das implementações que foram feitas no desenvoolvimento dos Microsserviços.

- **APIs RESTful**: Foi feita a implementação das **APIs RESTful**, definimos claramente os *endpoints*, - por exemplo, ***POST*** */login*, *GET* /utilizadores, ***DELETE*** /utilizador, ***PATCH*** /utilizador - utilizando os verbos ***HTTP*** corretos, e verificamos que as respotas em formato *JSON*, tinham códigos de *status* apropriados.
- **Lógica de Negócio**: Cada serviço contém uma lógica de negócio específica, o serviço ***Task***, gera a atualização, criação e exclusão de tarefas, o serviço ***Notifications*** é responsável pelo envio de um *email* de confirmação e pela notificação ao cliente, quando este faz operações com as tarefas, o serviço ***Auth***, tal como o nome indica, trata da autenticação do utilizador (*Login*, *Logout*), através de um token, e por fim, o serviço ***User***, que verifica a autenticação do cliente, e faz a gestão da criação, atualização e exclusão de utilizadores, assim como a gestão de quantos utilizadores estão autenticados no momento.
- **Conexão com a Base de Dados**: Esta implementação basea-se na utilização de ORM, como o ***Sequelize***, para extrair, legivelmente a interação direta com a base de dados, e a definição de modelos de dados.

E. *Detalhes de Implementação do Maestro*

- **Orquestração de Fluxo:** O Maestro faz chamadas sequenciais ou paralelas aos serviços necessários.
- **Comunicações Inter-Serviços:** As comunicações entre os serviços são feitas assincronamente, para evitar problemas de rede.

F. *Containerização com Docker*

Cada componente principal terá o seu próprio *DockerFile*, ou seja, cada serviço e o maestro.

Para o desenvolvimento local, implementamoss o ***docker-compose.yml***.

G. *Detalhes de Implementação da Orquestração com o Kubernetes (Minikube)*

- ***Deployment***: Para cad microsserviço, será criado um objeto ***Deployment*** do *Kubernetes*, que descreve como as instâncias dos *containers*, os *pods*, devem ser criadas.

H. *Detalhes de Implementação da Base de Dados*

Para implementar a base de dados, instalamos as dependências ***pg*** e ***pg-hstore*** através do ***npm***. Depois basta instânciar o *Sequelize* para conectar ao *ProsgreSQL*, e apartir daqui, definem-se os modelos para casa tabela da base de dados que será interagida. Com base nos ,modelos definidos, em ambiente de desenvolvimento, utilizamos o **sequelize.sync()** para criar automaticamente as tabelas na base de dados.

I. *Procedimento de Instalação*

A instalação e configuração destes projeto envolve várias etapas. Com o auxílio do **npm**, instalamos os seguintes pacotes:

- **axios**
- **cookie-parser**
- **cors**
- **express**
- **jsonwebtoken**
- **nodemon**
- **nodemailer**
- **luxon**
- **node-cron**
- **pg**
- **pg-store**

```
# Dockerfile para o serviço de utilizadores
FROM node:20

WORKDIR /User/src

# Instala dependências do sistema
RUN apt-get update && apt-get install -y postgresql-client

# Copiar o package.json e instalar as dependências Node
COPY src/package*.json ./
RUN npm install

# Copiar o restante do código, incluindo o script de espera
COPY . .


# Expõe a porta do serviço de utilizadores
EXPOSE 5000

# Iniciar o serviço apenas após o PostgreSQL estar pronto
CMD ["npx", "nodemon", "userService.js", "-L"]
```

Fig. 2. Como implementamos o *DockerFile*

- **sequelize**
- **bcrypt**
- **bcryptjs**

Além disto, também foi necessário instalar *softwares* para executar variadas funções. A seguir, estão os *softwares* de pré-requisito deste projeto:

- **Git**
- **Node.js**
- **Docker Desktop** (Docker Engine e Docker Compose)
- **Minikube**
- **kubectl**

J. Conclusão

Ao longo da nossa discussão sobre a implementação, procedimentos e instalação deste projeto de gestão de tarefas, ficou claro que a sua construção é um reflexo das melhores práticas em arquiteturas de *software* distribuídas.

A implementação detalhada de cada microsserviço com *Node.js*, expondo *APIs RESTful* e a orquestração centralizada pelo serviço Maestro, formam a ”core” funcional do sistema. A escolha estratégica do *PostgreSQL* com *Sequelize* para a persistência de dados em cada serviço garante a integridade e escalabilidade dos dados.

O procedimento de instalação, meticulosamente linear, ressalta a importância de ferramentas como *Docker* e *Kubernetes* (via *Minikube*). O Docker permite a *containerização* de cada componente, assegurando ambientes de execução consistentes e portáveis. Por sua vez, o *Kubernetes* orquestra de forma inteligente estes *containers*, automatizando a implantação, a escala e a gestão da resiliência, essenciais para a robustez de um sistema de microsserviços.

Em suma, este projeto atende aos requisitos técnicos de um sistema distribuído moderno. A combinação de uma implementação modular e um processo de instalação bem definido não só simplifica o desenvolvimento e a manutenção, mas também prepara o terreno para um sistema altamente disponível e escalável, capaz de responder eficazmente às necessidades de gestão de tarefas.

VIII. Conclusões e Trabalho Futuro

A. Conclusões principais

Este projeto, validou o desenvolvimento bem-sucedido de um sistema de gestão de tarefas com arquitetura baseada em microsserviços, atendendo aos objetivos da unidade curricular. Criamos um sistema escalável com serviços independentes para autenticação, utilizadores, tarefas e notificações. A integração e o funcionamento de todas as componentes, usando *Node.js* com *JavaScript*, *PostgreSQL* e *Sequelize*, comprovam a robustez do sistema. A *containerização* com *Docker* e a orquestração com *Kubernetes* (via *Minikube*), possibilitaram implantar e gerir um ambiente distribuído eficiente. A principal conclusão é que a arquitetura baseada em microsserviços, aliada a ferramentas como Docker e Kubernetes, é uma solução poderosa para sistemas flexíveis e resilientes, validando a aplicação prática dos conhecimentos adquiridos.

B. Trabalho Futuro

- **O que ficou por fazer e porquê?** Não implementamos uma interface de utilizador (*frontend*), pois o foco era a arquitetura de *backend* e orquestração.
- **Em que outros casos ou situações ou cenários o trabalho aqui escrito pode ter aplicações interessantes?** A arquitetura de microsserviços e o uso de *Docker* e *Kubernetes* são aplicáveis em diversos cenários, como sistemas de *e-commerce*, plataformas de *streaming*, serviços bancários/financeiros, *IoT/Edge Computing* e *gaming online*. Estas tecnologias oferecem a escalabilidade, resiliência e modularidade necessárias para sistemas complexos e de alto desempenho.

Referências

References

[1] ADM-Lecture-05
[2] ADM-Lecture-06/07
[3] ADM-Lecture-08