Universidade de Aveiro

BillsDivider
Engenharia e Gestão de Serviços

Ana Ferreira (93301), João Gameiro (93097), Miguel Nogueira (93082),
Pedro Abreu (93240)

Departamento de Electrónica, Telecomunicações e Informática

22 de junho de 2022

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

O presente relatório visa descrever a resolução do Projeto desenvolvido no âmbito da unidade curricular de Engenharia e Gestão de serviços.

O código desenvolvido para o projeto encontra-se disponível em: https://github.com/EGS-BillsDivider.

O produto criado destina-se a grupos de pessoas, por exemplo estudantes, que estejam a partilhar uma casa e necessitem de dividir as contas de uma forma eficaz. Permite a criação de contas e associação de utilizadores a essas contas para que seja possível a divisão do valor a pagar entre os mesmos. No fundo o modo de operação do sistema consiste em:

- Um utilizador colocar uma conta na plataforma.
- Selecionar um grupo de utilizadores com quem a conta foi dividida.
- Notificar os utilizadores que têm uma conta para pagar.
- Notificar o utilizador da publicação quando a dívida foi saldada.

Para desenvolver o sistema foi seguida uma abordagem baseada em micro-serviços na qual se identificaram três serviços: o de notificações, de autenticação e de registo e gestão de pagamentos.

Capítulo 2

Serviços

2.1 WebApp

Este serviço faz agregação dos serviços que serão mencionados nas secções seguintes. É visto como compositor para oferecer ao utilizador uma interface web através da qual o mesmo possa interagir com o sistema.

Para desenvolvimento do frontend da aplicação foi usado *React*. Para o backend, que age como compositor interligando os diferentes serviços com o frontend, foi usado *Flask*. Finalmente, para a base de dados local foi usado *MySQL*.

2.2 Authentication

A documentação criada para este serviço encontra-se disponível em: https://app.swaggerhub.com/apis/miguelnogueira1234/AuthenticationAPI/1.0.0.

É um serviço que cria e regista as credenciais de um novo utilizador. Quando é necessário aplicar um serviço de autenticação numa aplicação, este serviço pode ser usado, para guardar os utilizadores e fazer as validações necessárias para autenticar ou autorizar o acesso a outros serviços.

Para desenvolvimento deste serviço, foi *Node.js* para criação da API e *MongoDB* para a base de dados.

Tem como endpoints:

- **POST /register** - Para criação de um utilizador.
- **POST /login** - Verifica a existência das credenciais do utilizador e devolve um token.
- **GET /verifyUser** - Para verificar se o token dado é válido.

No método */login* é gerada uma *Json Web Token*, obtida a partir do *id* gerado automaticamente no método */register*, para ser usada por outros serviços para verificação de utilizadores.

Essa verificação é feita através de uma chamada ao método */verifyUser* com o respetivo token a ser validado.

2.3 Bills

A documentação criada para este serviço encontra-se disponível em: https://app.swaggerhub.com/apis/JPCGameiro/BillsAPI/1.0.0.

É um serviço de gestão de dívidas/contas. Permite o registo, atualização e eliminação de pagamentos de contas entre utilizadores. Faculta também a visualização das dívidas presentes no sistema com a sua respetiva informação associada. No entanto como este serviço guarda ids de utilizadores, é necessário garantir segurança no acesso aos métodos do mesmo. Ou seja, para fazer pedidos a cada método, é necessário fornecer um token no cabeçalho do pedido que tem de ser válido para que a operação sobre o serviço de contas seja realizada. Caso o token não seja válido, a operação não é efetuada. A verificação da validade deste token é feita através de uma chamada ao método */verifyUser* da API de autenticação.

Para desenvolvimento deste serviço, foram usados *FastAPI* para criação da API e *MySQL* para a base de dados.

Tem como endpoints:

- **POST /bill** - Para criação de uma conta.
- **GET /bills** - Devolve todas as contas existentes no sistema.
- **GET /bill/{billId}** - Devolve a conta com Id igual a *billId*.
- **PUT /bill/{billId}** - Atualiza a conta com Id igual a *billId* com as novas propriedades dadas.
- **DELETE /bill/{billId}** - Apaga a conta com Id igual a *billId*.

2.4 Notifications

A documentação criada para este serviço encontra-se disponível em: https://app.swaggerhub.com/apis/AnaLu602/NotificationsAPI/1.0.0.

É um serviço para criação e envio de notificações para um dado grupo de utilizadores definido. Permite a criação de canais onde estão associados utilizadores. O envio de uma notificação é feito através do canal.

Para desenvolvimento deste serviço, foram usados *Node.js* para criação da API e *MySQL* para a base de dados.

Tal como a API anterior alguns destes métodos utilizam ids de utilizadores, o que significa que é necessário fornecer um token para garantir a autorização na execução dos métodos desta

API. Essa verificação é feita tal como referido na API das *Bills*, isto é, através da chamada ao método ***/verifyUser*** da API de autenticação.

Tem como endpoints:

- **POST /notification** - Para criação e envio de uma notificação para os canais escolhidos.
- **POST /channel** - Para criação de um canal para agrupar utilizadores.
- **PUT /channel** - Para atualização da descrição ou utilizadores de um canal.
- **DELETE /channel** - Para apagar um canal existente.

Capítulo 3

Arquitetura

A partição em micro-serviços levou à arquitetura representada na Figura 3.1, em que se identificam 3 serviços, o de autenticação, o de gestão de contas e o de notificações, que quando agregados em conjunto através do compositor, dão origem ao serviço de divisão de contas que pode ser acessível e utilizado pelos clientes. Cada micro-serviço contém a si associado uma base de dados o que permite a sua operação de forma independente do resto do sistema, ao invés de uma única base de dados partilhada por todos os serviços.

Esta arquitetura permite também uma reutilização dos serviços na medida em que os serviços de autenticação, gestão de pagamentos e de notificações podem ser reutilizados e integrados com outros serviços.

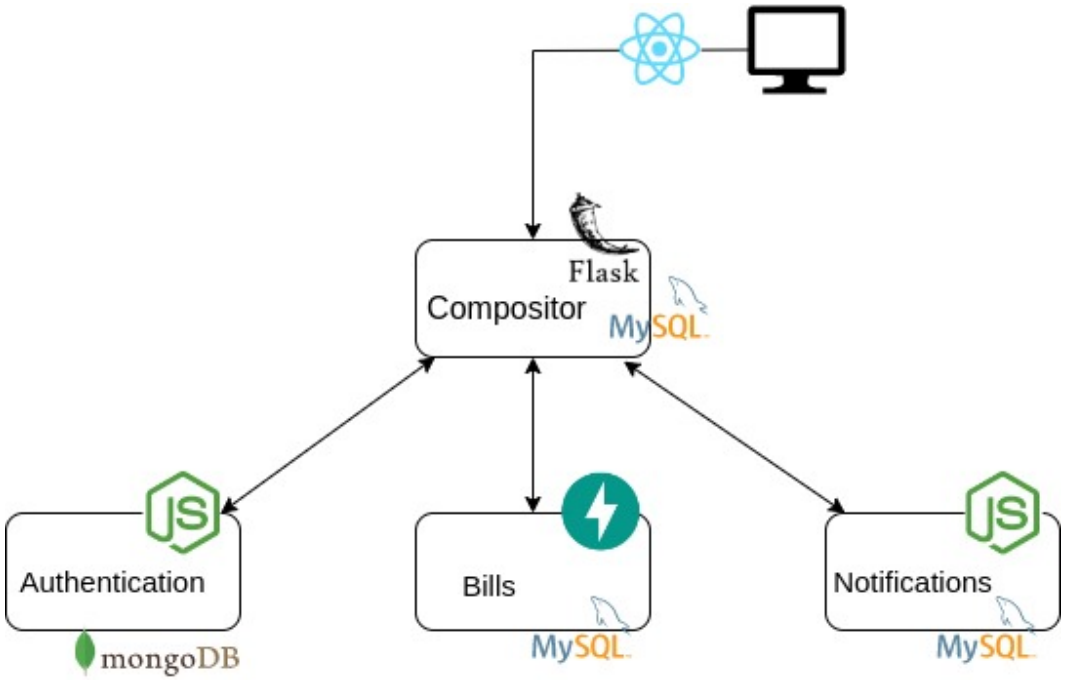

Figure 3.1: Arquitetura de software do sistema

Capítulo 4

Deployment

4.1 Webapp

O deployment deste serviço em *Kubernetes* é feito num único ficheiro, *deployment.yaml*. Encontra-se no repositório WebApp na pasta kube-deploy/ e é efetuado pela seguinte ordem:

- Criar um persistence volume claim (pvc)
- Criar deployment da base de dados que se baseia na imagem do Dockerfile.db que permite criar uma imagem baseada em *MySQL* na qual são criadas todas as tabelas necessárias. Expõe a porta 3306 e declara umn volume mount em *var/lib/mysql* associando-o ao *pvc* criado.
- Criar deployment da aplicação que se baseia na imagem do Dockerfile.app que é uma imagem baseada em python em que são copiados todos os ficheiros e instaladas todas as dependências para correr a aplicação. Contém 3 réplicas para redundância garantindo *high availability* do sistema.
- Criar deployment do nginx que se baseia na imagem do Dockerfile.nginx na qual são incluídas todas as configurações necessárias para o correto funcionamento do serviço. Contém 2 réplicas para redundância. Expõe a porta 80.
- Criar serviço da base de dados que expõe a porta 3306 associado ao deployment da base de dados.
- Criar serviço da app associado ao deployment da app que expõe o porto 8000 e mapeia para o porto 8000 das pods.
- Criar serviço do nginx associado ao deployment do nginx que expõe o porto 80 e mapeia para o porto 80 das pods.
- Configurar ingresso para o nginx associado ao serviço do *nginx* onde o host é "billsdivider.egs", para que o serviço esteja acessível para clientes exteriores ao cluster.

Na figura Figura 4.1 está um diagrama que representa o deployment feito.

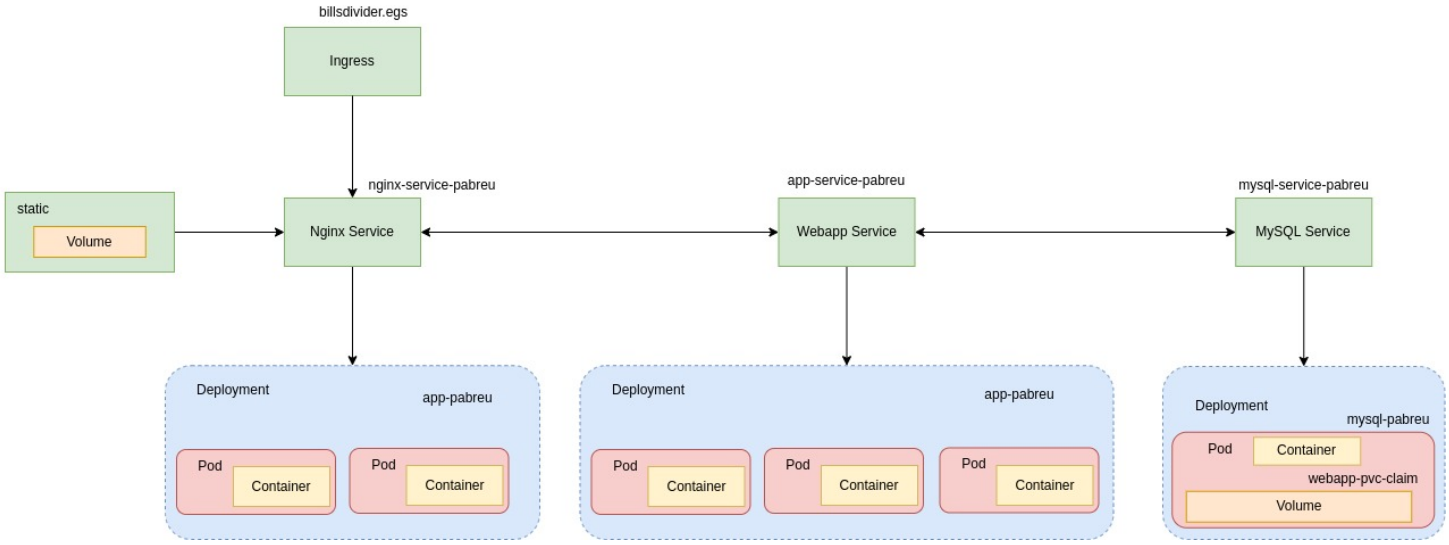

Figure 4.1: Diagrama do deployment da WebApp

4.2 Authentication

Para o deployment desta API, foram criados 3 serviços, um para a base de dados, um para a aplicação em si e um para o *nginx*. O *nginx* é um servidor http, reverse proxy que permite a partilha e utilização de dados em vários serviços diferentes. Para o caso do Ingresso, este é usado para disponibilizar o serviço para os utilizadores.

Na figura Figura 4.2 está um diagrama que representa o deployment feito.

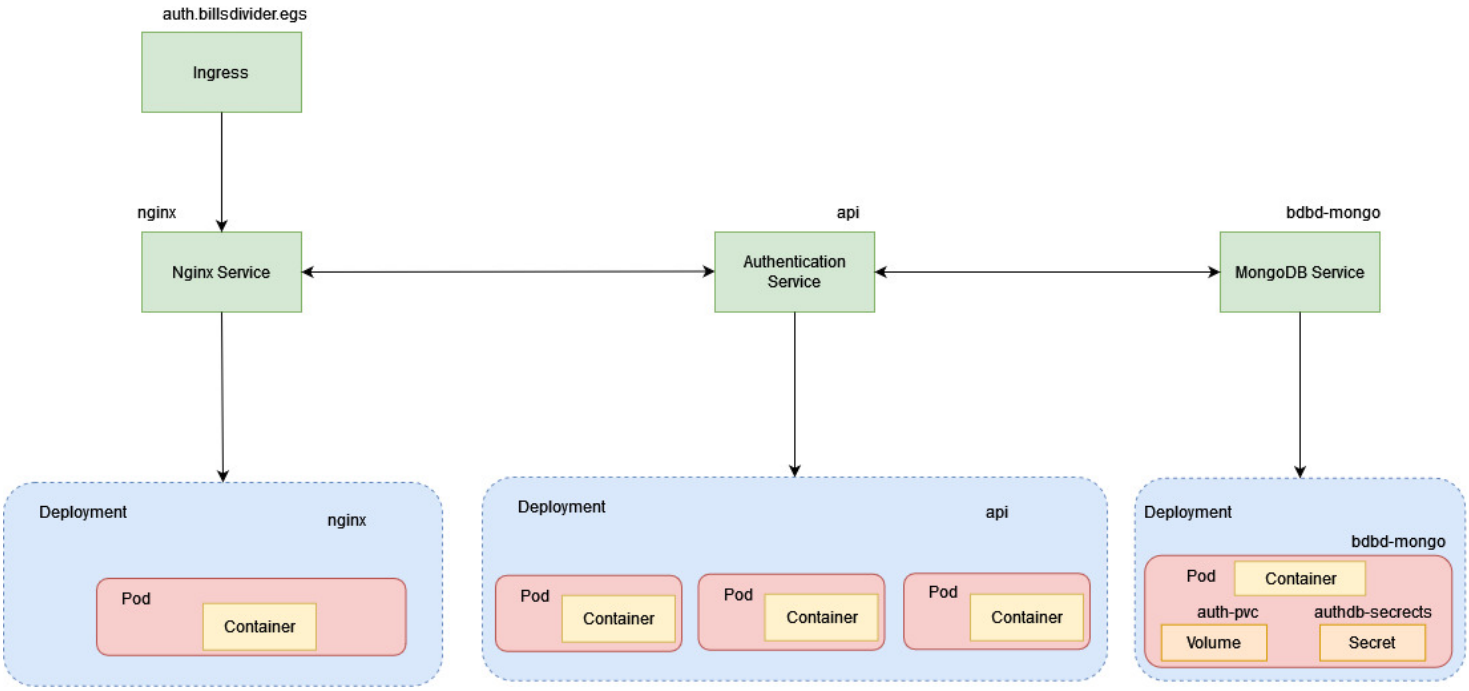

Figure 4.2: Diagrama do deployment da API de autenticação

MongoDB

Para o deploy da base de dados, primeiro foi necessário criar a imagem *registry.deti:5000/billsdivider/bdbd-mongo:17062022* e dar push para o registry da mesma. Esta imagem, é uma imagem de *mongo:4.4* com a porta 27017 exposta e onde vai o ficheiro que mapeia a base de dados e é criada a partir do ficheiro *Dockerfile.db*.

Para alocação de espaço para a base de dados, foi criado um Persistent Volume Claim, *auth-pvc*, de 500 Mi.

Para autenticação para acesso à base de dados, foi criado um Secret, authdb-secrets, onde as credenciais de acesso à base de dados se encontram codificadas em base 64.

Para o deploy da base de dados, foi criado um Deployment, com número de réplicas igual a 1, utilizando a imagem criada anteriormente e o volume. O Deployment contém um pod, com um container que expõe a porta 27017.

Por fim, para que seja possível aceder a esta base de dados por parte dos outros serviços presentes no mesmo namespace, foi criado um serviço na porta 27017, associado ao deploy criado anteriormente. O acesso a este serviço é feito através do nome *bdbd-mongo* e da porta 27017.

Os ficheiros usados no deployment da base de dados foram os seguintes:

- **deployment.yaml** - Para o deployment da base de dados e criação do persistent volume claim.
- **Dockerfile.db** - Para a criação da imagem que vai ser utilizada no deployment da base de dados.

Node.js

Para deploy da aplicação, primeiro foi necessário criar a imagem *registry.deti:5000/billsdivider/bdapi:06072022* e dar push para o registry da mesma. Esta imagem, é uma de *node:18* onde vão todos os ficheiros relacionados com a aplicação.

Para deploy da aplicação foram apenas criados um deployment usando a imagem criada do *Node.js* e um serviço associado a este. O acesso a este serviço é feito através do nome *api* e da porta 3000.

Os ficheiros usados no deployment da API foram os seguintes:

- **deployment.yaml** - Para o deployment da aplicação.
- **Dockerfile.api** - Para a criação da imagem de apoio ao deployment da API.

Nginx

Para deploy do Nginx, primeiro foi necessário criar a imagem *registry.deti:5000/billsdivider/nginx-proxy-bdapi:16062022* e dar push para o registry da mesma. Esta imagem, é uma de *nginx* onde vai o ficheiro de configuração.

O servidor web *Nginx* foi utilizado nesta parte do projeto para lidar com os acessos do utilizador à aplicação, assim como para servir os conteúdos estáticos utilizados na página de *login* e de *register*.

Os ficheiros usados no deployment do nginx foram os seguintes:

- **deployment.yaml** - Para o deployment do nginx.
- **nginx.conf** - Ficheiro com as configurações do nginx.
- **Dockerfile.nginx** - Para a criação da imagem de apoio ao deployment do nginx.

4.3 Bills

MySQL

Para deploy da base de dados, primeiro foi necessário criar a imagem *registry.deti:5000/billsdivider/billsdb:08062022* e dar push para o registry da mesma. Esta imagem, é uma imagem de *mysql:8.0.21* com a porta 3306 exposta e onde está contido um

ficheiro *init.sql* que está mapeado num diretório de inicialização do serviço *MySQL*, de modo a que no arranque do serviço, sejam criadas as tabelas necessárias para o correto funcionamento da aplicação.

Para alocação de espaço para a base de dados, foi criado um Persistent Volume Claim, noti-pvc-claim, de 500 Mi.

Para autenticação para acesso à base de dados, foi criado um Secret, *billsdb-secrets*, onde a password de acesso à base de dados se encontra codificada em base 64.

Para o deployment da base de dados, foi criado um Deployment, com número de réplicas igual a 3, utilizando a imagem, volume e segredo criados anteriormente. O Deployment contém 3 pods (por causa das réplicas), cada um com um container criado que expõe a porta 3306.

Por fim, para que seja possível aceder a esta base de dados por parte dos outros serviços no mesmo namespace, foi criado um serviço na porta 3306, associado ao deploy criado anteriormente. O acesso a este serviço é feito através do nome *noti-db* e da porta 3306. (*noti-db*) a esta base de dados

Os ficheiros usados no deployment foram os seguintes:

- **deployment-mysql.yaml** - Para o deployment da base de dados
- **deployment-mysql-pv.yaml** - Para a criação do persistent volume
- **secret-db.yaml** - Para a criação do segredo que contém a password de acesso à base de dados
- **Dockerfile.db** - Para a criação da imagem que vai ser utilizada no deployment da base de dados

Fast API

Para deploy da API, primeiro foi necessário criar a imagem *registry.deti:5000/billsdivider/billsapi:21062022* e dar push para o registry da mesma. Esta imagem, é uma imagem baseada em *python:3.8-alpine* com a porta 8000 exposta, na qual são copiados todos os ficheiros e instaladas todas as dependências necessárias para correr a API. Para além de todos os ficheiros que contém o código da aplicação, existe também um ficheiro *startup.sh* no qual se espera pela disponibilidade da base na porta 3306, para que a API não arranque sem a base de dados estar disponível.

Para passar as credenciais de acesso à base de dados foi criado um segredo do tipo *Opaque* através de um ficheiro que as contém que foi mapeado para um volume no diretório */tmp/secrets/*.

Para expôr a aplicação aos outros serviços presentes no mesmo namespace foi criado um serviço com o nome *bills-api*.

Em termos de replicação foram criadas três réplicas da aplicação para criar múltiplas instâncias do mesmo pod para oferecer redundância ao deployment.

Os ficheiros usados no deployment da API foram os seguintes:

- **deployment-api.yaml** - Para o deployment da aplicação
- **Dockerfile.api** - Para a criação da imagem de apoio ao deployment da API
- **create_secret.sh** - Script que permite criar o segredo que é utilizado na base de dados

A figura Figura 4.3 representa o um diagrama que representa o deployment feito.

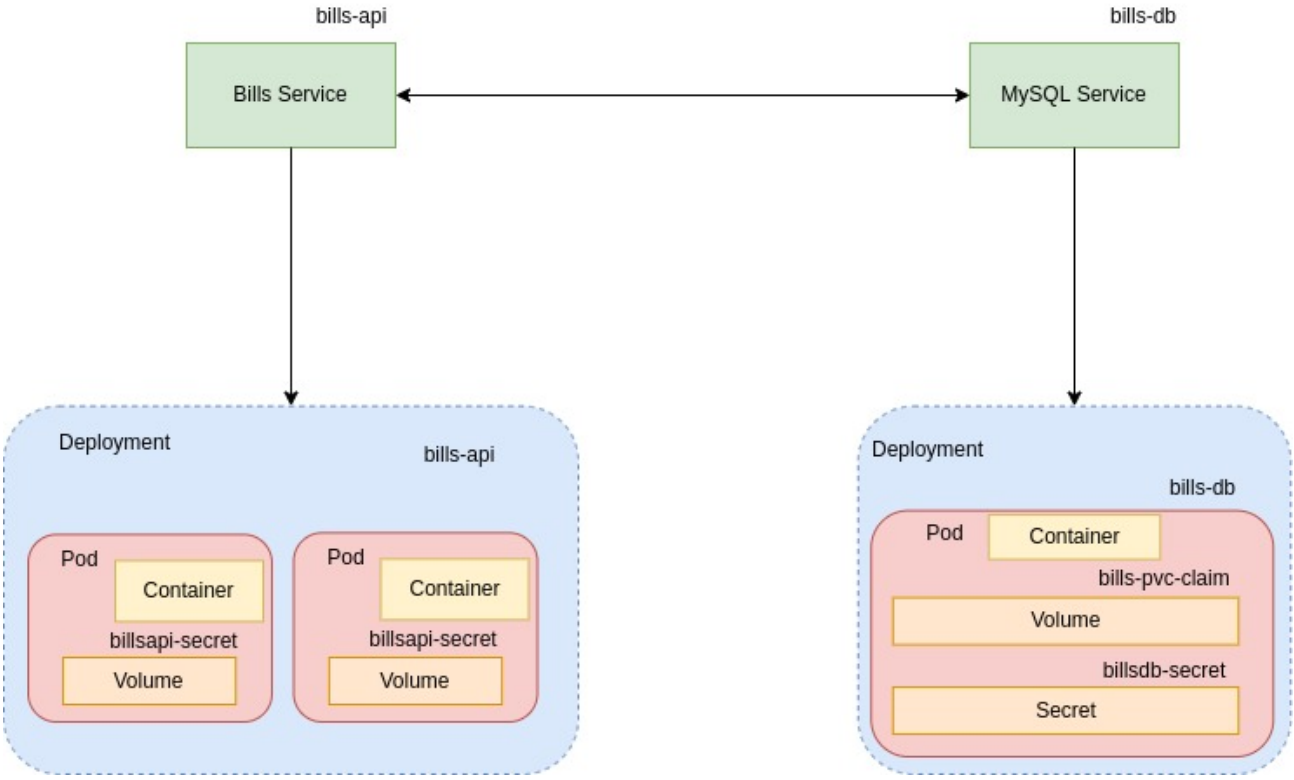

Figure 4.3: Diagrama do deployment da API de gestão de pagamentos

4.4 Notifications

Para deployment desta API, foram criados 2 serviços, um para a base de dados e um para a aplicação em si.

Na figura Figura 4.4 está um diagrama que representa o deployment feito.

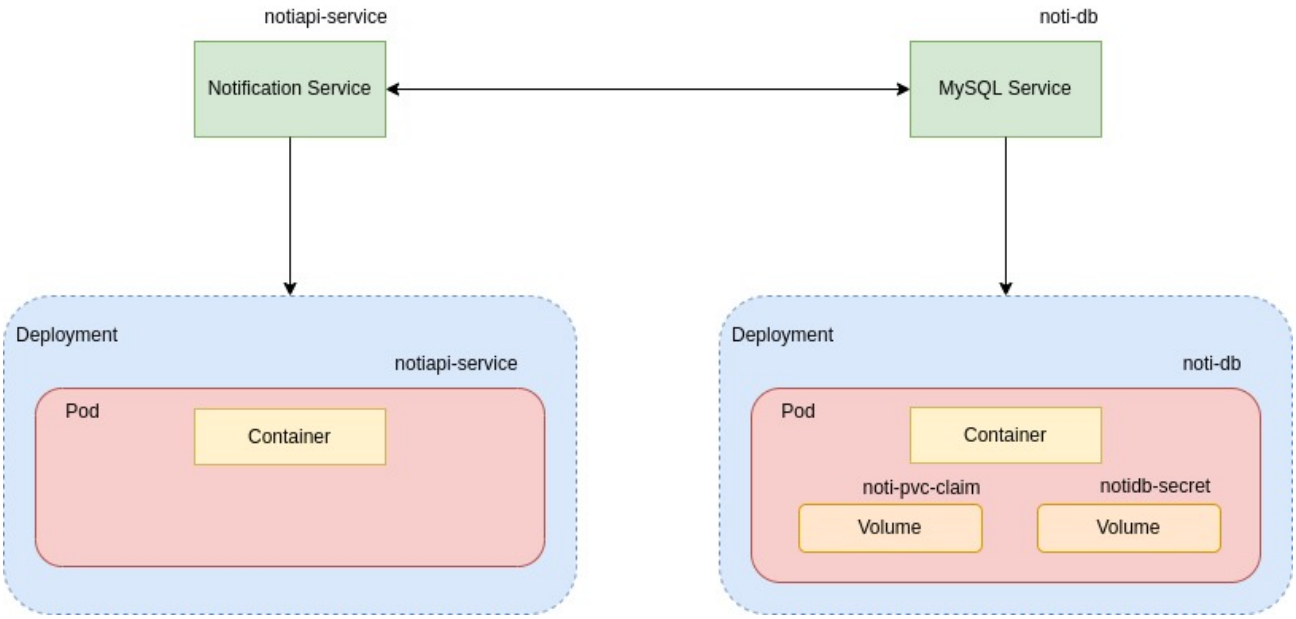

Figure 4.4: Diagrama do deployment da API de notificações

MySQL

Para deploy da base de dados, primeiro foi necessário criar a imagem *registry.deti:5000/billsdivider/notidb:19062022* e dar push para o registry da mesma. Esta imagem, é uma imagem de *mysql:8.0.21* com a porta 3306 exposta e onde vai o ficheiro que mapeia a base de dados e é criada a partir do ficheiro *Dockerfile.db*.

Para alocação de espaço para a base de dados, foi criado um Persistent Volume Claim, noti-pvc-claim, de 500 Mi.

Para autenticação para acesso à base de dados, foi criado um Secret, notidb-secret, onde a password de acesso à base de dados se encontra codificadas em base 64.

Para o deploy da base de dados, foi criado um Deployment, com número de réplicas igual a 1, utilizando a imagem criada anteriormente, o volume e o segredo. O Deployment contém um pod, com um container que expõe a porta 3306.

Por fim, para que seja possível aceder a esta base de dados por parte dos outros serviços presente no mesmo namespace, foi criado um serviço na porta 3306, associado ao deploy criado anteriormente. O acesso a este serviço é feito através do nome *noti-db* e da porta 3306.

Os ficheiros usados no deployment da base de dados foram os seguintes:

- **mysql-service.yaml** - Para o deployment da base de dados.
- **mysql-pvc.yaml** - Para a criação do persistent volume claim.

- **mysql-secret.yaml** - Para a criação do segredo que contém a password de acesso à base de dados.
- **Dockerfile.db** - Para a criação da imagem que vai ser utilizada no deployment da base de dados.

Node.js

Para deploy da aplicação, primeiro foi necessário criar a imagem *registry.deti:5000/billsdivider/notificationsapi:29062022* e dar push para o registry da mesma. Esta imagem, é uma de *node* com a porta 3000 exposta e onde vão todos os ficheiros relacionados com a aplicação.

Para deploy da aplicação foram apenas criados um deployment usando a imagem criada do *Node.js* e um serviço associado a este. O acesso a este serviço é feito através do nome *noti-api* e da porta 3000.

Os ficheiros usados no deployment da API foram os seguintes:

- **deployment-api.yaml** - Para o deployment da aplicação.
- **Dockerfile.api** - Para a criação da imagem de apoio ao deployment da API.

Capítulo 5

Divisão de trabalho

Consideramos que todos os elementos contribuíram de igual forma para a realização do projeto pelo que atribuímos 25% de participação a cada elemento do grupo.