NOME: Ícaro Ríon dos Santos Mendes

Pesquise e Responda as perguntas abaixo.
1) Como definir um volume no Docker Compose para persistir os dados do banco de dados PostgreSQL entre as execuções dos containers?

Adicionando a seção "volumes" ao seu arquivo docker-compose.yml, dentro de um serviço "db". O volume nomeado será criado e montado no caminho ao dar o comando docker-compose up.

2) Como configurar variáveis de ambiente para especificar a senha do banco de dados PostgreSQL e a porta do servidor Nginx no Docker Compose?

Usando a seção "environment" em cada serviço correspondente no arquivo docker-compose.yml. Então definimos a variável de ambiente com o valor que desejamos como senha: "POSTGRES_PASSWORD=senha_do_banco". O mesmo processo servirá para o serviço nginx, dentro da seção environment definirá a variável "NGINX_PORT=80", e depois a mapeamos para a porta 8080 na seção ports.

3) Como criar uma rede personalizada no Docker Compose para que os containers possam se comunicar entre si?

Adicionando a seção "networks" ao seu arquivo docker-compose.yml. A rede personalizada chamada “mynetwork” na seção “networks” será adicionada em ambos os serviços: “app” e “db”.

4) Como configurar o container Nginx para atuar como um proxy reverso para redirecionar o tráfego para diferentes serviços dentro do Docker Compose?

Criar um diretório chamado nginx no mesmo diretório do arquivo docker-compose.yml. Criar um arquivo de configuração chamado nginx.conf com o seguinte conteúdo:

```
events {}
        http {
                server { listen 80; location /app {
                        proxy_pass http://app:3000;
                        }
                location /api {
                proxy_pass http://api:5000;
                } } }
```

Adicionar o serviço do Nginx e monte o volume com o arquivo de configuração personalizado:

Adicionamos o serviço do Nginx ao seu arquivo docker-compose.yml.
Ele usa a imagem oficial do Nginx e mapeia a porta 80 do container para a porta 80 do host.
Especificamos o volume ./nginx/nginx.conf:/etc/nginx/nginx.conf

Na seção “depends_on”, incluímos os serviços app e api para garantir que o Nginx espere até que esses serviços estejam em execução antes de iniciar.

5) Como especificar dependências entre os serviços no Docker Compose para garantir que o banco de dados PostgreSQL esteja totalmente inicializado antes do Python iniciar?

Usar a seção “depends_on” no seu arquivo docker-compose.yml. Por exemplo:

```
version: '3'
services: db:
        image: postgres
        # Configurações do serviço de banco de dados

        python: build: ./python
        depends_on: - db
        # Configurações do serviço Python
```

O serviço python depende do serviço db, pois precisa que o banco de dados PostgreSQL esteja pronto antes de iniciar.

6) Como definir um volume compartilhado entre os containers Python e Redis para armazenar os dados da fila de mensagens implementada em Redis?

Utilizando os serviços: python e redis, na seção volumes, criamos um volume chamado queue-data que será compartilhado entre os dois serviços.
O volume é montado em caminhos diferentes para cada serviço:

Para o serviço python, o volume é montado em /app/queue-data, onde você pode armazenar os dados da fila de mensagens.
Para o serviço redis, o volume é montado em /data, que é o diretório padrão usado pelo Redis para armazenar dados.

Desse modo, tanto o serviço Python quanto o Redis poderão acessar e manipular os dados da fila de mensagens armazenados no volume queue-data.

7) Como configurar o Redis para
aceitar conexões de outros containers apenas na rede interna do Docker Compose e não de fora?

Ajustando as configurações do Redis para restringir as conexões a um endereço IP específico correspondente à rede interna do Docker Compose.
Ao remover a seção “ports” do serviço Redis, ele só estará acessível dentro da rede interna do Docker Compose.

8) Como limitar os recursos de CPU e memória do container Nginx no Docker Compose?

Usando as opções “cpus” e “mem_limit” no seu arquivo docker-compose.yml.

“cpus” permite especificar a quantidade de CPU alocada para o container. O valor '0.5' daria ao container Nginx acesso a metade do poder de processamento de uma CPU.

“mem_limit” permite definir o limite máximo de memória RAM que o container pode utilizar. No exemplo, 100m limitaria o container a 100 megabytes de memória.

Ao definir essas opções, você pode ajustar os valores de acordo com os recursos disponíveis no seu sistema e as necessidades do container Nginx.

9) Como configurar o container Python para se conectar ao Redis usando a variável de ambiente correta especificada no Docker Compose?

Passando a variável de ambiente para o container Python no arquivo docker-compose.yml.

```
version: '3'
services:
        python:
          build: ./python
          environment:
            - REDIS_HOST=redis
            - REDIS_PORT=6379

          redis:
          image: redis
```

Nesse exemplo, temos o serviço python que representa o container Python e o serviço redis que representa o container do Redis.

No serviço python, utilizamos a seção environment para definir as variáveis de ambiente “REDIS_HOST” e “REDIS_PORT”.

O valor “redis” indica que o hostname do Redis é o nome do serviço “redis” dentro da rede Docker Compose.
O valor 6379 especifica a porta em que o Redis está escutando.

Em seu código Python, você pode acessar essas variáveis de ambiente para configurar a conexão com o Redis. Por exemplo:

```
python
Copy code
import os
import redis
redis_host = os.environ.get('REDIS_HOST', 'localhost')
redis_port = int(os.environ.get('REDIS_PORT', 6379))

# Conexão com o Redis
r = redis.Redis(host=redis_host, port=redis_port)
```

Nesse exemplo, usamos o módulo os para acessar as variáveis de ambiente REDIS_HOST e REDIS_PORT definidas no Docker Compose.

10) Como escalar o container Python no Docker Compose para lidar com um maior volume de mensagens na fila implementada em Redis?

Usando a funcionalidade de escalabilidade do Docker Compose.
Exemplo:

```
version: '3'
services:
        python:
          build: ./python
          environment:
            - REDIS_HOST=redis
            - REDIS_PORT=6379
          scale: 3

  redis:
    image: redis
```

Nesse exemplo, temos o serviço python que representa o container Python e o serviço redis que representa o container do Redis.

No serviço python, adicionamos a propriedade scale para especificar o número de instâncias desejadas do container Python. No exemplo, scale: 3 indica que queremos três instâncias do container Python em execução.