1. Como definir um volume no Docker Compose para persistir os dados do banco de dados PostgreSQL entre as execuções dos containers?
   –
   Podemos utilizar a seção "volumes" no arquivo docker-compose.yml. Como por exemplo:

```
version: '3'
services:
  db:
    image: postgres
    volumes:
      - db_data:/var/lib/postgresql/data

volumes:
  db_data:
```

Estamos definindo um serviço chamado "db" usando a imagem oficial do PostgreSQL. Depois estamos especificando um volume chamado "db_data" que será montado no diretório "/var/lib/postgresql/data" do container. Assim, os dados do banco de dados PostgreSQL serão armazenados no volume "db_data".

2. Como configurar variáveis de ambiente para especificar a senha do banco de dados PostgreSQL e a porta do servidor Nginx no Docker Compose?
   –
   Podemos utilizar a seção "environment" em cada serviço no arquivo docker-compose.yml. Como por exemplo:

```
version: '3'
services:
  db:
    image: postgres
    environment:
      - POSTGRES_PASSWORD=minha_senha_secreta

  nginx:
    image: nginx
    ports:
      - "80:80"
    environment:
      - NGINX_PORT=80
```

   Vamos definir a variável de ambiente `POSTGRES_PASSWORD` com o valor "minha_senha_secreta" para o serviço `db`. Ela será passada para o container do PostgreSQL como a senha do banco de dados. Vamos definir também a variável de ambiente `NGINX_PORT` com o valor "80" para o serviço `nginx`. Essa variável será passada para o container do Nginx e poderá ser usada dentro do container para configurar a porta do servidor.

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

–
Podemos utilizar a seção "networks" no arquivo docker-compose.yml. Como por exemplo:

```
version: '3'
services:
  app:
    build: .
    networks:
      - mynetwork

  db:
    image: postgres
    networks:
      - mynetwork

networks:
  mynetwork:
```

Foi criada uma rede personalizada chamada "mynetwork". Ambos os serviços "app" e "db" estão conectados a essa rede usando a chave `networks` em seus respectivos blocos de definição.
Os containers do serviço "app" e "db" serão conectados à mesma rede "mynetwork". Isso permite que eles se comuniquem entre si usando o nome do serviço como host, por exemplo, o container "app" poderá acessar o container "db" usando o hostname "db".

4. Como configurar o container Nginx para atuar como um proxy reverso para redirecionar o tráfego para diferentes serviços dentro do Docker Compose?
   –
   Necessita definir a configuração do Nginx em um arquivo de configuração personalizado e montá-lo como um volume no container. Como por exemplo:

```
version: '3'
services:
  nginx:
    image: nginx
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
```

No arquivo `docker-compose.yml`, adicione o serviço do Nginx e monte o arquivo de configuração personalizado como um volume. Estamos montando o arquivo `nginx.conf` no caminho `/etc/nginx/nginx.conf` dentro do container Nginx.

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?
   –
   Podemos usar a seção "depends_on" no arquivo docker-compose.yml. No entanto, é importante frizar que o Docker Compose não

espera a total inicialização do serviço dependente antes de iniciar o próximo serviço. O "depends_on" apenas controla a ordem de inicialização dos serviços.

6. Como definir um volume compartilhado entre os containers Python e Redis para armazenar os dados da fila de mensagens implementada em Redis?
   –
   `docker run --name meu-redis -v /caminho/completo/redis_data:/data redis`

   ```
   import redis

   r = redis.Redis(host='endereco_ip_do_redis', port=6379, db=0)
   ```

   Substitua "endereco_ip_do_redis" pelo endereço IP do contêiner Redis ou pelo IP do host, dependendo da configuração.

1. Crie um diretório no seu sistema host para ser usado como volume compartilhado, por exemplo, “redis_data”.
2. Ao iniciar o contêiner Redis, especifique o diretório a ser montado como um volume compartilhado usando o parâmetro `-v` ou `--volume`. Por exemplo.
3. No código Python, conecte-se ao servidor Redis usando o endereço IP do contêiner Redis. Use a biblioteca Redis para isso.

7. Como configurar o Redis para aceitar conexões de outros containers apenas na rede interna do Docker Compose e não de fora?
   –
   Pode definir o parâmetro "bind" no arquivo de configuração do Redis dentro do container. No entanto, no Docker Compose, todos os serviços dentro de uma rede têm acesso mútuo por padrão, portanto, não há necessidade de configurações adicionais para restringir as conexões apenas à rede interna do Docker Compose.
   *Um breve exemplo:

   1. Criar um arquivo `redis.config` na mesma pasta do arquivo `docker-compose.yml`.
   2. Depois colocar a configuração `bind 127.0.0.1` no arquivo `redis.conf`.
   3. No arquivo `docker-compose.yml`, precisa especificar o caminho para o arquivo `redis.config` no contêiner Redis usando o parâmetro `volumes.`
   4. Por último reinicie seus contêineres do Docker Compose para aplicar as alterações.

8. Como limitar os recursos de CPU e memória do container Nginx no Docker Compose?
   –
   Podemos adicionar as seguintes linhas no arquivo `docker-compose.yml`. Por exemplo:

   ```
   version: '3'
   ```

```
services:
  nginx:
    image: nginx
    ports:
      - 80:80
    cpus: 0.5
    mem_limit: 512m
```

O contêiner Nginx será limitado a usar no máximo 50% de um núcleo de CPU e 512MB de memória. Essas configurações garantem que o Docker limite os recursos disponíveis para o contêiner Nginx, controlando o uso de CPU e memória.

9. Como configurar o container Python para se conectar ao Redis usando a variável de ambiente correta especificada no Docker Compose?
   –
   Pode definir a variável de ambiente no serviço Python no arquivo docker-compose.yml. Por exemplo:

```
version: '3'
services:
  python:
    build: .
    environment:
      - REDIS_HOST=meu-redis
```

```
import os
import redis

redis_host = os.environ.get('REDIS_HOST', 'localhost')
r = redis.Redis(host=redis_host, port=6379, db=0)
```

   - Foi colocado a variável de ambiente `REDIS_HOST` com o valor `meu-redis`, que é o nome do serviço do contêiner Redis definido no mesmo arquivo `docker-compose.yml`.
   - No código Python, foi utilizado a biblioteca Redis para acessar a variável de ambiente para obter o valor do endereço IP ou host do Redis.
   - Foi utilizado `os.environ.get('REDIS_HOST', 'localhost')` para obter o valor da variável de ambiente `REDIS_HOST`. Caso a variável não esteja definida, será utilizado o valor padrão `'localhost'`.

10. Como escalar o container Python no Docker Compose para lidar com um maior volume de mensagens na fila implementada em Redis?
    –
    Podemos utilizar o comando `docker-compose up --scale` seguido pelo nome do serviço e o número de réplicas
    desejadas.
    Por exemplo:

```
docker-compose up --scale python=3
```