Avaliação - 15

1) Como definir um volume no Docker Compose para persistir os dados do banco de dados PostgreSQL entre as execuções dos containers ?
Usando a chave “volumes” dentro do serviço do postgresQL no arquivo “docker-compose.yml”.

2) Como configurar variáveis de ambiente para especificar a senha do banco de dados PostgreSQL e a porta do servidor Nginx no Docker Compose ?
Pode usar a chave “enviroment” dentro do “docker-compose.yml” para definir as variáveis do ambiente e também para definir a porta do “nginx”, outra maneira de definir as variáveis do ambiente é criar um arquivo “.env” dentro do mesmo diretório do arquivo “docker-compose.yml”.

3) Como criar uma rede personalizada no Docker Compose para que os containers possam se comunicar entre si ?
Usando a chave “networks” dentro do arquivo “docker-compose.yml”, assim criando uma rede personalizada em que os containers fazem parte.

4) Como configurar o container Nginx para atuar como um proxy reverso para redirecionar o tráfego para diferentes serviços dentro do Docker Compose ?
Criando um arquivo de configuração para o nginx que definirá as regras de proxy reverso, por exemplo, definir um bloco "upstream" pro serviço que você quer redirecionar, depois configurar um bloco “server” com uma regra “location”, essa regra define um caminho e redireciona o tráfego para o serviço correspondente usando a diretiva “proxy_pass”.

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 ?
Usando a chave "depends_on" no arquivo “docker-compose.yml”, porém o uso do "depends_on" não garante que o serviço dependente esteja totalmente pronto e disponível, apenas que ele seja iniciado antes. Para garantir que o python espere a disponibilidade do banco de dados, é recomendável adicionar um trecho de código no seu aplicativo python que verifique a disponibilidade do banco de dados antes de iniciar as operações.

6) Como definir um volume compartilhado entre os containers Python e Redis para armazenar os dados da fila de mensagens implementada em Redis ?
Usando a chave “volumes” no serviço do redis e no serviço python no arquivo “docker-compose.yml”, o volume tem que estar definido na seção “volumes” fora do bloco de serviços para que ele possa ser referenciado em ambos os serviços.

7) Como configurar o Redis para aceitar conexões de outros containers apenas na rede interna do Docker Compose e não de fora ?
No arquivo “docker-compose.yml” definimos um nome para a rede interna do docker compose, adicionamos essa definição na seção networks do arquivo. Dentro do container redis, configuramos o redis para ouvir apenas no endereço IP local, para fazer isso adicionamos um arquivo de configuração personalizada para o redis. Por fim mapeamos o arquivo de configuração personalizada do redis para o container, e também adicionamos um “command” com o caminho do arquivo de configuração para iniciar o redis com a configuração personalizada.

8) Como limitar os recursos de CPU e memória do container Nginx no Docker Compose ?
Usando as chaves “cpu_shares”: para definir a quantidade de recursos de CPU que o container nginx deve receber, “cpus”: para limitar quanto o container nginx pode usar do núcleo de CPU e “mem_limit”: para definir um limite de memória para o container nginx, dentro do serviço nginx no arquivo “docker-compose.yml”.

9) Como configurar o container Python para se conectar ao Redis usando a variável de ambiente correta especificada no Docker Compose ?
Definindo a variável de ambiente no serviço python no arquivo “docker-compose.yml” e em seguida acessá-la no código python. No arquivo “docker-compose.yml” , adicionamos a variável de ambiente para a conexão Redis no serviço python, depois no código python usamos a biblioteca “os” para se conectar ao redis.

10) Como escalar o container Python no Docker Compose para lidar com um maior volume de mensagens na fila implementada em Redis ?
Usando o recurso de dimensionamento do docker compose, no arquivo “docker-compose.yml” definimos o número desejado de réplicas do serviço python usando a chave “scale”, depois executamos o comando “docker-compose up –scale python=" para iniciar as réplicas do serviço python , isso criará containers python em execução simultaneamente todos se conectando ao mesmo serviço redis. Cada instância do container python processará uma parte do volume de mensagens da fila, distribuindo a carga entre as réplicas.