Atividade 15

Sistemas Operacionais de Redes 2

Professor: Ricardo Duarte Taveira

Aluno: Erick Carneiro de Oliveira Rocha

1) Como definir um volume no Docker Compose para persistir os dados do banco de dados PostgreSQL entre as execuções dos containers?

Resposta: Ao escrever o docker-compose.yml, especificar o caminho do repositório do volume no container Postgres.

O código é o seguinte:

```
version: '3'
volumes:
  dados:
networks:
  banco:
  web:
  fila:
services:
  db:
    image: postgres:9.6
    volumes:
      # Volume dos dados
      - dados:/var/lib/postgresql/data
      # Scripts
      - ./scripts:/scripts
      - ./scripts/init.sql:/docker-entrypoint-initdb.d/init.sql
    networks:
      - banco
    environment:
      - POSTGRES_PASSWORD=password
```

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

Resposta: No arquivo docker-compose.yml, definir no container Postgres as seguintes linhas para denominar a senha a ser utilizada pelo serviço:

```
version: '3'
volumes:
  dados:
networks:
  banco:
  web:
  fila:
services:
  db:
    image: postgres:9.6
    volumes:
      # Volume dos dados
      - dados:/var/lib/postgresql/data
      # Scripts
      - ./scripts:/scripts
      - ./scripts/init.sql:/docker-entrypoint-initdb.d/init.sql
    networks:
      - banco
    environment:
      - POSTGRES_PASSWORD=password
```

Em seguida, no container Nginx escrever as seguintes linhas para especificar a porta a ser utilizada:

```
  frontend:
    image: nginx:1.13
    volumes:
      # Site
      - ./web:/usr/share/nginx/html/
      # Configuração do proxy reverso
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    ports:
      - 80:80
    networks:
      - web
    depends_on:
      - app
```

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

Resposta: No arquivo docker-compose.yml, definir as redes a serem utilizadas por cada serviço. Isso a feito por meio da declaração de networks.

```
version: '3'
volumes:
  dados:
networks:
  banco:
  web:
  fila:
```

```
services:
  db:
    image: postgres:9.6
    volumes:
      # Volume dos dados
      - dados:/var/lib/postgresql/data
      # Scripts
      - ./scripts:/scripts
      - ./scripts/init.sql:/docker-entrypoint-initdb.d/init.sql
    networks:
      - banco
    environment:
      - POSTGRES_PASSWORD=password

  frontend:
    image: nginx:1.13
    volumes:
      # Site
      - ./web:/usr/share/nginx/html/
      # Configuração do proxy reverso
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    ports:
      - 80:80
    networks:
      - web
    depends_on:
      - app

  app:
    image: python:3.6
    volumes:
      # Aplicação
      - ./app:/app
    working_dir: /app
    command: bash ./app.sh
    networks:
      - banco
      - web
      - fila
    depends_on:
      - db
      - queue
    environment:
      - DB_NAME=email_sender

  queue:
    image: redis:3.2
    networks:
      - fila
```

```
  worker:
    build: worker
    volumes:
      # Worker
      - ./worker:/worker
    working_dir: /worker
    command: worker.py
    networks:
      - fila
    depends_on:
      - queue
```

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

Resposta: Especificar os parâmetros de configuração próprios no arquivo nginx.conf dentro do diretório no qual o docker-compose utilizará para executar o container nginx. Tais parâmetros são os de roteamento de portas e endereços.

```
server {
    listen 80;
    server_name localhost;

    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root /usr/share/nginx/html;
    }

    location /api {
        proxy_pass http://app:8080/;
        proxy_http_version 1.1;
    }
}
```

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?

Resposta: Por meio da declaração de dependências escrita no arquivo docker-compose.yml, nas linhas especificadas como “depends_on:”. Nesse caso específico do Python depender do Postgres já estar instanciado, é na seguinte linha:

```
  app:
    image: python:3.6
    volumes:
      # Aplicação
      - ./app:/app
    working_dir: /app
    command: bash ./app.sh
    networks:
      - banco
      - web
      - fila
    depends_on:
      - db
      - queue
    environment:
      - DB_NAME=email_sender
```

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

Resposta: Dentro do arquivo docker-compose.yml determinar um volume, e nos containers do Python e do Redis especificar o direcionamento para o volume definido.

```
version: '3'
volumes:
  dados:
networks:
  banco:
  web:
  fila:

  app:
    image: python:3.6
    volumes:
      # Aplicação
      - ./app:/app
    working_dir: /app
    command: bash ./app.sh
    networks:
      - banco
      - web
      - fila
    depends_on:
      - db
      - queue
    environment:
      - DB_NAME=email_sender
```

```
  queue:
    image: redis:3.2
    networks:
      - fila
  worker:
    build: worker
    volumes:
      # Worker
      - ./worker:/worker
    working_dir: /worker
    command: worker.py
    networks:
      - fila
    depends_on:
      - queue
```

7) Como configurar o Redis para

aceitar conexões de outros containers apenas na rede interna do Docker Compose e não de fora?

Resposta: No arquivo docker-compose.yml especificar as redes para os containers por meio da definição networks, e na secção do container do Redis definir networks sendo essas as redes anteriormente especificadas.

```
version: '3'
volumes:
  dados:
networks:
  banco:
  web:
  fila:

  worker:
    build: worker
    volumes:
      # Worker
      - ./worker:/worker
    working_dir: /worker
    command: worker.py
    networks:
      - fila
    depends_on:
      - queue
```

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

Resposta: No arquivo docker-compose.yml definir no container Nginx as seguintes seções

```
  frontend:
    image: nginx:1.13
    deploy:
      resources:
        limits:
          cpus:
          mem_limit:
    volumes:
      # Site
      - ./web:/usr/share/nginx/html/
      # Configuração do proxy reverso
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    ports:
      - 80:80
    networks:
      - web
    depends_on:
      - app
```

A opção “cpus:” determina o limite de recursos de CPU a serem utilizados, já mem_limit determina o limite de recursos de memória.

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

Resposta: Dentro docker-compose.yml, no container Python definir a secção Environment.

```
  app:
    image: python:3.6
    volumes:
      # Aplicação
      - ./app:/app
    working_dir: /app
    command: bash ./app.sh
    networks:
      - banco
      - web
      - fila
    depends_on:
      - db
      - queue
    environment:
      - DB_NAME=email_sender
```

Já no arquivo do script Python que esse container vai executar, as seguintes linhas são responsáveis pelos parâmetros de conexão com o Redis.

```
import psycopg2
import redis
import json
import os
from bottle import Bottle, request


class Sender(Bottle):
    def __init__(self):
        super().__init__()
        self.route('/', method='POST', callback=self.send)

        redis_host = os.getenv('REDIS_HOST', 'queue')
        self.fila = redis.StrictRedis(host=redis_host, port=6379, db=0)

        db_host = os.getenv('DB_HOST', 'db')
        db_user = os.getenv('DB_USER', 'postgres')
        db_name = os.getenv('DB_NAME', 'email_sender')
        db_password = os.getenv('DB_PASSWORD', 'password')
        dsn = f'dbname={db_name} user={db_user} password={db_password} host={db_host}'
        self.conn = psycopg2.connect(dsn)

    def register_message(self, assunto, mensagem):
        SQL = 'INSERT INTO emails (assunto, mensagem) VALUES (%s, %s)'
        cur = self.conn.cursor()
        cur.execute(SQL, (assunto, mensagem))
        self.conn.commit()
        cur.close()

        msg = {'assunto': assunto, 'mensagem': mensagem}
        self.fila.rpush('sender', json.dumps(msg))

        print('Mensagem registrada !')

    def send(self):
        assunto = request.forms.get('assunto')
        mensagem = request.forms.get('mensagem')

        self.register_message(assunto, mensagem)
        return 'Mensagem enfileirada ! Assunto: {} Mensagem: {}'.format(
            assunto, mensagem
        )

if __name__ == '__main__':
    sender = Sender()
    sender.run(host='0.0.0.0', port=8080, debug=True)
```

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

Resposta: No arquivo docker-compose.yml definir no container Python a secção deploy com a propriedade réplicas com o número desejado para escalamento.

```
app:
  image: python:3.6
  volumes:
    # Aplicação
    - ./app:/app
  working_dir: /app
  command: bash ./app.sh
  deploy:
    replicas: 5
  networks:
    - banco
    - web
    - fila
  depends_on:
    - db
    - queue
  environment:
    - DB_NAME=email_sender
```

Neste caso, o container estará escalado para 5 instâncias.