1. Para definir um volume no Docker Compose e persistir os dados do banco de dados PostgreSQL entre as execuções dos containers, é necessário adicionar a seguinte configuração no arquivo docker-compose.yml:

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

   Nesse exemplo, o serviço Python será escalado para três instâncias, permitindo processar mais mensagens em paralelo. Cada instância terá seu próprio container e será executada em seu próprio ambiente isolado.

2. Para configurar variáveis de ambiente no Docker Compose, especificamente para definir a senha do banco de dados PostgreSQL e a porta do servidor Nginx, pode-se adicionar a seguinte configuração ao serviço correspondente no arquivo docker-compose.yml:

   ```
   version: '3'
   services:
     db:
       image: postgres
       environment:
         - POSTGRES_PASSWORD=senha
     nginx:
       image: nginx
       ports:
         - 8080:80
       environment:
         - NGINX_PORT=8080
   ```

   Essa configuração indica que o serviço Python depende do serviço db (PostgreSQL) e garante que o PostgreSQL seja inicializado antes do Python.

3. Para criar uma rede personalizada no Docker Compose e permitir a comunicação entre os containers, pode-se adicionar a seguinte configuração no arquivo docker-compose.yml:

```
version: '3'
services:
  db:
    image: postgres
    networks:
      - mynetwork
  nginx:
    image: nginx
    networks:
      - mynetwork
networks:
  mynetwork:
```

Essa configuração indica que o serviço Python depende do serviço db (PostgreSQL) e garante que o PostgreSQL seja inicializado antes do Python.

4. Para configurar o container Nginx como um proxy reverso no Docker Compose e redirecionar o tráfego para diferentes serviços, pode-se adicionar a seguinte configuração ao serviço do Nginx no arquivo docker-compose.yml:

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

Essa configuração indica que o serviço Python depende do serviço db (PostgreSQL) e garante que o PostgreSQL seja inicializado antes do Python.

5. Para especificar dependências entre os serviços no Docker Compose e garantir que o banco de dados PostgreSQL esteja totalmente inicializado antes do Python iniciar, pode-se usar a opção "depends_on" no serviço Python no arquivo docker-compose.yml:

```
version: '3'
services:
  db:
    image: postgres
  python:
    build: .
```

```
    depends_on:
      - db
```

Essa configuração indica que o serviço Python depende do serviço db (PostgreSQL) e garante que o PostgreSQL seja inicializado antes do Python.

6. Para definir um volume compartilhado entre os containers Python e Redis no Docker Compose, pode-se adicionar a seguinte configuração aos serviços correspondentes no arquivo docker-compose.yml:

```
version: '3'
services:
  python:
    build: .
    volumes:
      - shared-data:/app/data
  redis:
    image: redis
    volumes:
      - shared-data:/data
volumes:
  shared-data:
```

Nesse exemplo, é criado um volume nomeado "shared-data" que é mapeado para os diretórios "/app/data" no container Python e "/data" no container Redis. Dessa forma, os dados da fila de mensagens implementada em Redis podem ser compartilhados entre os dois containers.

7. Para configurar o Redis para aceitar conexões apenas na rede interna do Docker Compose e não de fora, pode-se adicionar a seguinte configuração ao serviço Redis no arquivo docker-compose.yml:

```
version: '3'
services:
  redis:
    image: redis
    networks:
      - mynetwork
networks:
  mynetwork:
    internal: true
```

Ao adicionar a opção "internal: true" à rede "mynetwork", o container Redis só aceitará conexões de outros containers na mesma rede e não será acessível externamente.

8. Para limitar os recursos de CPU e memória do container Nginx no Docker Compose, pode-se usar as opções "cpus" e "mem_limit" no serviço correspondente no arquivo docker-compose.yml:

```
version: '3'
services:
  nginx:
    image: nginx
    cpus: 0.5
    mem_limit: 512m
```

Nesse exemplo, o container Nginx será limitado a utilizar no máximo 0,5 núcleos de CPU e 512 MB de memória.

9. Para configurar o container Python para se conectar ao Redis usando a variável de ambiente correta especificada no Docker Compose, pode-se adicionar a seguinte configuração ao serviço Python no arquivo docker-compose.yml:

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

Nesse exemplo, são definidas as variáveis de ambiente "REDIS_HOST" e "REDIS_PORT" com os valores adequados para a conexão com o Redis. O nome do serviço Redis no Docker Compose é usado como host ("redis") e a porta padrão do Redis é especificada como porta (6379).

10. Para escalar o container Python no Docker Compose e lidar com um maior volume de mensagens na fila implementada em Redis, pode-se usar a opção "scale" no comando "docker-compose up". Por exemplo:

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

Nesse exemplo, o serviço Python será escalado para três instâncias, permitindo processar mais mensagens em paralelo.

Cada instância terá seu próprio container e será executada em seu próprio ambiente isolado.