1 - Abrir o arquivo “docker-compose.yml” e dentro do serviço do PostgreSQL, adicionar a seção “volumes”, onde vocÊ define a origem e o destino do volume

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

2 - Abri o arquivo “docker-compose.yml” e dentro dos serviços relevantes adicionar a seção “environment” para especificar as variáveis do ambiente.

```
    services:
  db:
    image: postgres
    environment:
      - POSTGRES_PASSWORD=minha_senha_secreta

  nginx:
    image: nginx
    ports:
      - "8080:80"
    environment:
      - NGINX_PORT=80
```

3 - Abrir p “docker-compose.yml” e no nível superior do arquivo “docker-compose.yml” adicionar a seção “networks” para definir a rede personalizada:

```
    version: '3.9'

    networks:
      minha-rede:
```

4 - Criar um novo arquivo chamado “nginx.config” no mesmo diretório onde está localizado o arquivo “docker-compose.yml”. No arquivo “nginx.config” adicionar as seguintes configurações:

```
        events {}

http {
  server {
    listen 80;

    location /service1/ {
      proxy_pass http://app1:5000/;
    }

    location /service2/ {
      proxy_pass http://app2:8000/;
    }
  }
}
```

e no arquivo “docker-compose.yml” adicionar o serviço do Nginx:

```
services:
  nginx:
    image: nginx
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    ports:
      - "80:80"
    depends_on:
      - app1
      - app2
```

5 -

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

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

6 - Usar a diretiva “volumes” no arquivo “docker-compose.yml”

```
     version: "3"
     services:
      python:
       build: .
        volumes:
      - mydata:/app/data
      # Resto da configuração do serviço Python

    redis:
    image: redis
    volumes:
      - mydata:/data
    # Resto da configuração do serviço Redis

     volumes:
     mydata:
```

7 -

Abrir o arquivo “redis.config” para adicionar:

$ docker exec -it <nome_do_seu_container_redis> bash

$ vi /etc/redis/redis.conf

Localizar a diretiva “bind” e certifica-se que está configurada para o enderço IP interno do container em vez de “0.0.0.0”:

bind 172.17.0.2

salvar e sair do arquivo “redis.config” e reiniciar o container Redis:

$ docker restart <nome_do_seu_container_redis>

8 - Usar as diretivas “cpu_limit” e “mem_limit” dentro do serviço Nginx no arquivo “docker-compose.yml”

```
version: "3"
 services:
   nginx:
    image: nginx
  ports:
    - 80:80
   cpu_limit: 0.5
   mem_limit: 512m
```

9 - Usar a diretiva “environment” para definir as variáveis de ambiente para o serviço python :

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

No código python ler as variáveis para obter as informações de conexão do Redis. Posso usar a biblioteca “os”

```
 import os
import redis

redis_host = os.getenv('REDIS_HOST', 'localhost')
redis_port = int(os.getenv('REDIS_PORT', '6379'))

# Conecte-se ao Redis usando as informações de conexão
r = redis.Redis(host=redis_host, port=redis_port)
# Use o objeto 'r' para realizar operações no Redis
```

10 - usar a diretiva “scale” no arquivo “docker-compose.yml”. A diretiva “scale ” permite que você especifique o número de réplicas do serviço

```
version: "3"
services:
  python:
    build: .
    environment:
      - REDIS_HOST=redis
      - REDIS_PORT=6379
    scale: 2
```