Projeto

Copiar pasta da aplicação para um container:

```
 sudo docker cp RPC negocioAPI:/home
```

Cria uma rede:

```
docker run --network minha-rede -d python:3
```

[Todo o SD deve estar dentro de uma mesma rede]

Criação dos containers:

```
docker run -itd --network rede-rpc --name negocioAPI python:3
```

Comandos para algumas instalações dentro dos containers:

```
pip install uvicorn
pip install fastapi
apt-get update
apt-get install nano
python3 -m pip install pymongo
 To update, run: pip install --upgrade pip
```

Execução dos containers base

negocioAPI

Startar o container

```
docker start negocioAPI
```

Executar o container

```
docker exec -it negocioAPI /bin/bash
```

Acessar o diretório

```
cd home/RPC/negocio
```

Definir PATH

```
export PYTHONPATH="/home/RPC"
```

Subir servidor de negocio

```
uvicorn negocioAPI:negocioAPI --reload --host 0.0.0.0 --port 
```

dadosAPI

Startar o container

```
docker start dadosAPI
```

Executar o container

```
docker exec -it dadosAPI /bin/bash
```

Acessar o diretório

```
cd home/RPC/dados
```

Definir PATH

```
export PYTHONPATH="/home/RPC"
```

Subir servidor de negocio

```
uvicorn dadosAPI:dadosAPI --reload --host 0.0.0.0 --port 8003
```

cliente

cd home/RPC/dados

Startar o container

```
docker start cliente
```

Executar o container

```
docker exec -it cliente /bin/bash
```

Acessar o diretório

```
cd home/RPC/cliente
```

Definir PATH

```
export PYTHONPATH="/home/RPC"
```

Para testar a conexão com a API: curl http://172.19.0.2:8001

- Testar dentro do container cliente

Pode-se apenas iniciar os containers mongo e depois acessar. Tem que iniciar o replica set.

Réplicas MongoDB → docker-compose

1. Criar docker-compose

```
version: '3'
services:
  mongo1:
    image: mongo
    command: mongod --replSet rs0
    ports:
      - 27017:27017
    volumes:
      - ./data/mongo1:/data/db
    networks:
      - rede-rpc

  mongo2:
    image: mongo
    command: mongod --replSet rs0
    ports:
      - 27018:27017
    volumes:
      - ./data/mongo2:/data/db
    networks:
```

```
      - rede-rpc

  mongo3:
    image: mongo
    command: mongod --replSet rs0
    ports:
      - 27019:27017
    volumes:
      - ./data/mongo3:/data/db
    networks:
      - rede-rpc

networks:
  rede-rpc:
    external: true
```

Caso já tenha subido o docker-compose.yml

```
docker-compose down
```

```
docker-compose up
```

Trocar nome do volume no docker compose antes de subir para poder inicializar outro replica set nesse container - talvez até mudar o nome dos bancos funciona.

2. Descobrir o IP de todos os banco de dados:

```
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAd
```

3. Acessar o primeiro container (será considerado como mestre).

```
docker exec -it nomedbc bash -c "mongosh"
```

3.1. Inicializar o replica-set adicionado todos os nós que deseja (bancos)

```
rs.initiate({
  _id: "rs0", // Nome do conjunto de réplicas - igual ao do d
  members: [
    { _id: 0, host: "172.19.0.10:27017" }, // Primeiro nó
    { _id: 1, host: "172.19.0.11:27017" }, // Segundo nó
    { _id: 2, host: "172.19.0.12:27017" }  // Terceiro nó
  ]
})
```

Dentro de uma mesma rede, se comunicam pela mesma porta (27017). Apenas expõe ao host (localhost) portas diferentes.

O **nó primário** no **conjunto de réplicas do MongoDB** é identificado pela propriedade "**stateStr**" com o valor "PRIMARY" (members) na lista de membros retornada pelo comando rs.status().

Testes de replicação - inserindo dados

Criar/selecionar dataset (se não existir, cria)

```
use meuBancoDeDados
```

Rápida inserção de dados:

```
db.minhaColecao.insertOne({ nome: "João", idade: 30 })
```

Visualizar dados do banco:

```
db.minhaColecao.find()
```

Acessar outros brancos e verificar dados

- Somente nó mestre pode alterar/ler.
- Nós secundários não podem ler/fazer alterações, logo é necessário habilitar se quiser fazer.

Habilitar temporariamente a opção slaveOk para permitir operações de leitura em nós secundários

```
rs.secondaryOk()
```

Alteração de IPs na aplicação

Inspecionar IP de qualquer container (trocar nome de acordo com o container)

```
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAd
```

Trocar os IPs nos container:

- **cliente** - Alterar os de negócios
  - cliente/RequisicoesCliente.py
- **negocio** - Alterar os de dados
  - negocio/RequisicoesNegocio.py
- **dados** - Alterar os do banco MongoDB
  - dados/MovimentacaoDados.py
  - Se necessário, alterar *database* e *collection*
    - dados/MongoConexao.py

💡 **Para acessar banco por meio do localhost, usar “localhost/porta/?replicaset”**

[Usar sempre a porta utilizada na inicialização do replica set]

```
from pymongo import MongoClient

# Cria uma conexão com o MongoDB
client = MongoClient('mongodb://localhost:27017/?replicaSet=r

# Seleciona o banco de dados 'mydatabase'
db = client['dbnovo']

# Imprime o nome do banco de dados selecionado
print(f"Conectado ao banco de dados: {db.name}")

# Insere um documento no banco de dados
doc = {"name": "Jenifer", "age": 22, "city": "Rio do Sul"}
result = db.colecao.insert_one(doc) # Colecao - nome da coleç

# Imprime o ID do documento inserido
print(f"Inseriu documento com ID: {result.inserted_id}")
```

💡 **Para acessar banco por meio do container, usar “IP-nó-banco/porta/?replicaset”:**

Usar sempre a porta utilizada na inicialização do replica set]

```
conexao = MongoConexao('mongodb://172.19.0.6:27017/?replicaSe
```

Execução da aplicação:

- Container: **cliente**
- Diretório:

```
/home/RPC/cliente
```

- Comando para execução:

```
python3 mainAPI.py
```

🔢 Replicações negocioAPI e dadosAPI

negocioAPI → 172.19.0.2:8001

negocioAPI2 → 172.19.0.3:8002

negocioAPI3 → 172.19.0.4:8003

dadosAPI → 172.19.0.5:8004

dadosAPI 2→ 172.19.0.6:8005

dadosAPI 3→ 172.19.0.7:8006

rpc_mongo1_1 →172.19.0.11:27017/27017

rpc_mongo2_1 →172.19.0.10:27018/27017

rpc_mongo3_1 →172.19.0.9:27019/27017

escalonadorAPI → 172.19.0.12:8007

Escalonador

Criar container para o escalonador

```
docker run -itd --network rede-rpc --name escalonadorAPI pyth
```

Acessar caminho da aplicação

```
cd /Documentos/BCC/AMBIENTE-SISTEMAS-DISTRIBUIDOS
```

Replicar docs. para o container

```
sudo docker cp RPC escalonadorAPI:/home
```

Acessar o container

```
 docker exec -it escalonadorAPI /bin/bash
```

```
cd /home/RPC/escalonador
```

```
export PYTHONPATH="/home/RPC"
```

```
uvicorn escalonadorAPI:escalonadorAPI --reload --host 0.0.0.0
```

RASCUNHOS DE FUNCIONAMENTO:

escalonadorAPI > 172.19.0.2:8001 **OK**

negocioAPI > 172.19.0.3:8002 **OK**

negocioAPI2 > 172.19.0.4:8003 **OK**

negocioAPI3 > 172.19.0.5:8004 **OK**

dadosAPI > 172.19.0.8:8005

dadosAPI2 > 172.19.0.7:8006

dadosAPI3 > 172.19.0.6:8007

cliente > 172.19.0.9

rpc_mongo1_1 > 172.19.0.10:27017

rpc_mongo2_1> 172.19.0.11:27017

rpc_mongo3_1 > 172.19.0.12:27017

```
negocios_disponiveis: List[Tuple[str, int]] = [
    ("172.19.0.3", 8002),
    ("172.19.0.4", 8003),
    ("172.19.0.5", 8004)
]

dados_disponiveis: List[Tuple[str, int]] = [
    ("172.19.0.8", 8005),
    ("172.19.0.7", 8006),
    ("172.19.0.6", 8007)
]

bancos_disponiveis: List[Tuple[str, int]] = [
    ("172.19.0.10", 27017),
    ("172.19.0.11", 27017),
    ("172.19.0.12", 27017)
]
```

Alterar:

- escalonadorAPI
- Todas as requisições… 3 em cada