Web Scraper v Docker with Redis cache

1. Aplikační kód (app.py)

Do projektové složky se vloží funkční soubor `app.py`, který obsahuje:

- Flask webovou aplikaci (backend)
  logiku pro:
  - měření HTTP požadavků (GET, HEAD)
  - analýzu HTML a obrázků (BeautifulSoup)
  - komunikaci s Redis cache
- jednoduché HTML rozhraní vykreslované na serveru

Flask aplikace běží jako server dostupný na portu 5000.

2. Dockerfile - zabalení aplikace

Do stejné složky se vytvoří soubor **Dockerfile**.

Obsah Dockerfile:

```
FROM python:3.10-slim
WORKDIR /app
COPY app.py
RUN pip install --no-cache-dir flask requests beautifulsoup4 redis
CMD ["python", "app.py"]
```

Dockerfile:

1. vezme základní Python image
2. vytvoří pracovní adresář `/app`
3. zkopíruje aplikační kód
4. nainstaluje potřebné knihovny
5. definuje příkaz pro spuštění aplikace

Výsledkem je **Docker image**.

3. Build Docker image

Ve složce projektu se spustí:

```
docker build -t web-scraper .
```

Docker:

- přečte Dockerfile
- provede jednotlivé kroky
- vytvoří image s názvem `web-scraper`

Image je **připravený balíček,** který ale ještě neběží.

4. Spuštění aplikace bez Redis

Pro ověření funkčnosti lze image spustit samostatně `docker run -p 5000:5000 web-scraper`

- vytvoří se container
- aplikace běží na `http://localhost:5000`

5. Přidání Redis cache (multi-container řešení)

Protože projekt používá **Flask + Redis**, nepoužívá se `docker run`, ale `Docker Compose.`

7. docker-compose.yml

Do složky projektu se přidá soubor **docker-compose.yml**:

```
services:
 web:
   build: .
   ports:
     - "5000:5000"
   depends_on:
     - redis

 redis:
   image: redis:7
```

Význam:

- `web` - Flask aplikace
- `Redis` - Redis cache

Docker automaticky:

  - vytvoří síť
  - propojí kontejnery
  - zajistí správné pořadí spuštění

8. Spuštění celého projektu

Celý projekt se spustí jedním příkazem `docker-compose up --build`

Docker:

- znovu sestaví image aplikace
- spustí Redis container
- spustí Flask container
- aplikace je dostupná na portu 5000

9. Napojení Redis v app.py

V aplikaci se vytvoří Redis klient:

```
redis_client = redis.Redis(
   host="redis",
   port=6379,
   decode_responses=True
)
```

- **host není localhost**
- používá se název služby z `docker-compose.yml`
- hostname redis existuje pouze v Docker síti.

Redis slouží jako **cache vrstva** s TTL.

10. Chování aplikace

1. Uživatel zadá URL
2. Flask se nejprve zeptá Redis:
    - pokud data existují → vrátí se z cache
    - pokud ne → provede se scrapování
3. Výsledek se uloží do Redis s expirací
4. Uživatel dostane výsledek