Documentation
“Beautonomy” mining data principal scheme

ABSTRACT: The application is aimed to obtain certain information about targets that originated from social networks

TECHNOLOGIES: Node, Redis, MySQL, Moleculer, S3 Cloud Object Storage, AWS, Puppeteer

1. Scheme legend

1.1. Flows (**bold** – material streams):

1. **Targets**
2. **L grab items**
3.1. **L grab items**
3.2. Proxies
4. **Raw data**
5. Update replay
6. **Dead tasks**
7. **Dead logs**
8. L resolve items (pointers)
9. F resolve items (pointers)
10. **Raw files**
11. **Processed data**

1.2. Software layers:

Service layer

– Initial Microservice
– Redis Microservice
– Grab Microservice (includes Grab Storage Subservice and Grab Dead Subservice)
– Resolve Microservice (includes Resolve Storage Subservice)
– Moleculer Broker
– Log Analyzer

Data layer

– Initial DB
– Redis
– S3 Cloud Object Storage
– MySQL

2. Scheme description

2.1. Data initialization

Input data coming from **Initial DB** (**1. Targets**) processed by **Initial Microservice** which produces *“Last In”* stream **2. L grab items**.

2.2. Scheduling

Then **Redis Microservice** pushes **2. L grab items** into the **Task Queue** which emits grabbing tasks on-demand (*“First out”* stream **3.1. F grab items**).

~~**Redis Microservice** also has **Proxy Storage** that is updated with the use counter.~~

according to the first revision (rev.1** on the scheme), **Proxy Storage** (updated with the use counter) will be processed inside **Moleculer Microservice** by **Transporter***

2.3. Data mining

Then **Redis Microservice** pops **3.1. F grab items** from **Task Queue** and **Moleculer Transporter** pops 3.2. Proxies from **Proxy Storage** into the **Grab Microservice**, which produces **4. Raw data** stream by its *Instances*. At the same time **Grab Microservice** update replay amount (**5. Update replay** stream) by pushing failed tasks into the **Task Queue**.

If the updated replay amount exceeds a certain number, these tasks are considered as dead (**6. Dead tasks**) and are sent to the **Grab Dead Subservice**, which logs them (**7. Dead logs** into the **S3 Cloud**) and store them in **Dead Queue**. Dead logs will be analyzed by **Log Analyzer**.

2.4. Raw data storing

Then **Grab Storage Subservice** processes **4. Raw data** stream and saves files to **S3 Cloud** (**10. Raw files**). At the same time **Redis Microservice** pushes pointers (*"Last in"* 8. L resolve items) from **Grab Storage Subservice**.

2.5. Raw data processing

Then **Redis Microservice** pops pointers (*"First Out"* 9. F resolve items) from **Task Queue** into the **Resolve Microservice**, which receives **11. Raw files** from **S3 Cloud** (through pointers) and produces **11. Processed data** stream by its *Instances*.

2.6. Processed data storing

Then **Resolve Storage Subservice** processes **11. Processed data** stream and saves records (**12. Result data**) to **MySQL DB**.

3. Application automation control

3.1. **Grab/Resolve Microservice** load

Load is distributed between *Instances* controlled by **Moleculer Broker**.

3.2. **Task Queue** load

The queue is self-controlled by definition.

3. Items structure description

Task Queue (**Dead Queue**) item general structure:

```
{
   task: grab,
   url: https://www.instagram.com/nasa/,
   replay: 0
},
{
   task: resolve,
   raw_guid: fnsu-df8f-fdaD-gre1
}
```

Proxy Storage item general structure:

```
{
   proxy: 127.0.0.0,
   times: 2
}
```