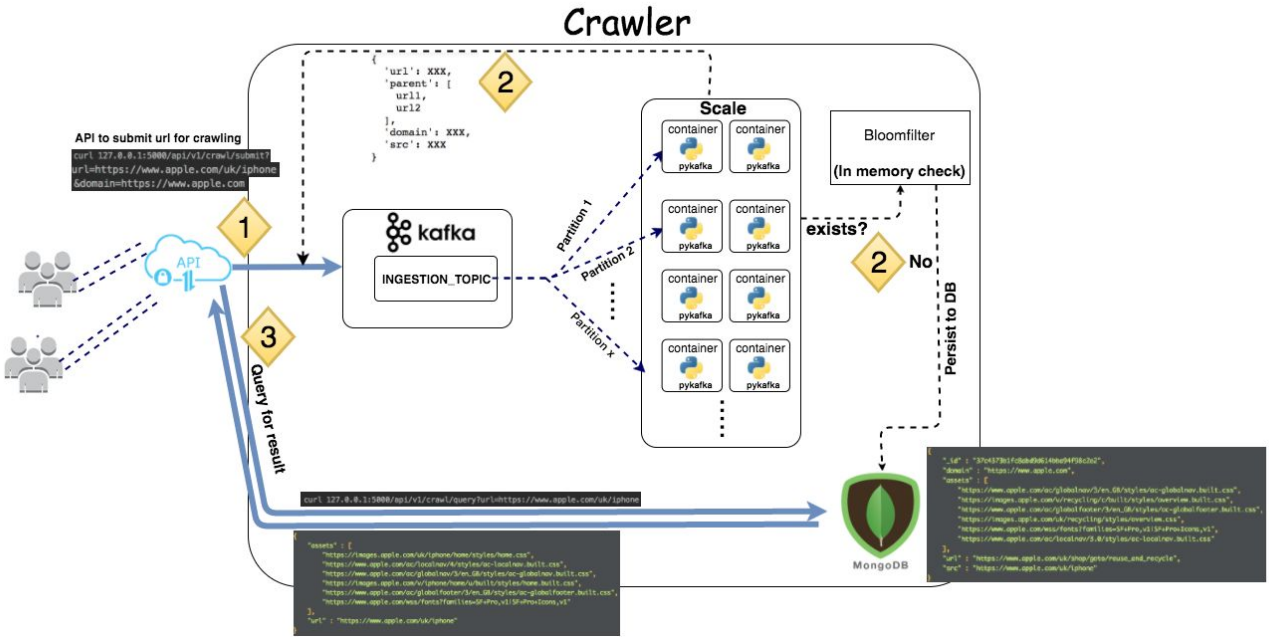
Architecture:

Following is an overview of my design solution for the requested crawler.



Algorithm design:

1. User submit the url to crawl using API
    - Ex: **curl 127.0.0.1:5000/api/v1/crawl/submit? url=https://www.apple.com/uk/iphone& domain=https://www.apple.com**
2. A json request pushed to kafka ingestion_topic.
3. As soon as it is ingested run.py starts consuming the url. Followings are steps:
    - Check if this url is already processed
    - Find all links in that url within our domain / Find all assets related to that url
    - Persist current url and its assets into db
    - For each links in the above links form a json like in step(2) and push to kafka to repeat crawling for those links
    - Since we store all the parents url for any new visited url we can control the level of crawling depth
    - The above procedure repeated until we reach to a predefined depth or all urls for that url gets crawled
    - This can run on one process or multiple process at the same time to scale it out(one of the reason behind choosing kafka)
4. User can see the list of assets at any time by generating following API call and passing url:
    - Ex: **curl 127.0.0.1:5000/api/v1/crawl/query? url=https://www.apple.com/uk/iphone**

Technologies:

Apache Kafka:

kafka used to ingest links using its ordering nature. Also by defining multiple partitions for a topic we can scale out crawling the links.

Advantages:

- Kafka is massively scalable. There is no limitation on the volume of input data.
- Using Kafka We can push different stream of data by defining different topics and partition them logically for different solutions.
- By defining multiple partitions for kaka topics, number of pykafka wrapper can be started and respond. This increase the level of parallelism.

pyKafka wrapper: I have written a wrapper around pykafka to simplify push(produce) and pull(consume) from kafka. It serializes/deserializes the message(json) in python.

Mongodb: A database is used to persist the results.

Flask: Flask web service is used to handle the API calls from the end user of the service by querying the outputs from mongo and send back the reply.

Python: Project code is written in python which is the language I am comfortable most.

Installation and Run:

Please follow the instruction to install and run the service.

Requirements:

- Fill database connection info in configuration.py
- Fill Kafka broker IP in configuration.py

- **Installation:**
  - clone repository
  - go to cloned folder
  - install: `sudo pip/pip3 install .`

- **Run it:**

1) Run consumer:
   a) Python run.py

   Note: This is a single thread consumer, you can run multiple instances of the consumer on different processes.
2) Start the web service:
   a) Python webservice.py

Deployment:

To deploy, the repo will be containerized and pushed to docker hub. On deployed node container will be pulled and run independently. To increase parallelism run multiple copy of the containers. Use docker swarm or marathon to start many containers.