News Classifier

High Level System Architecture

and

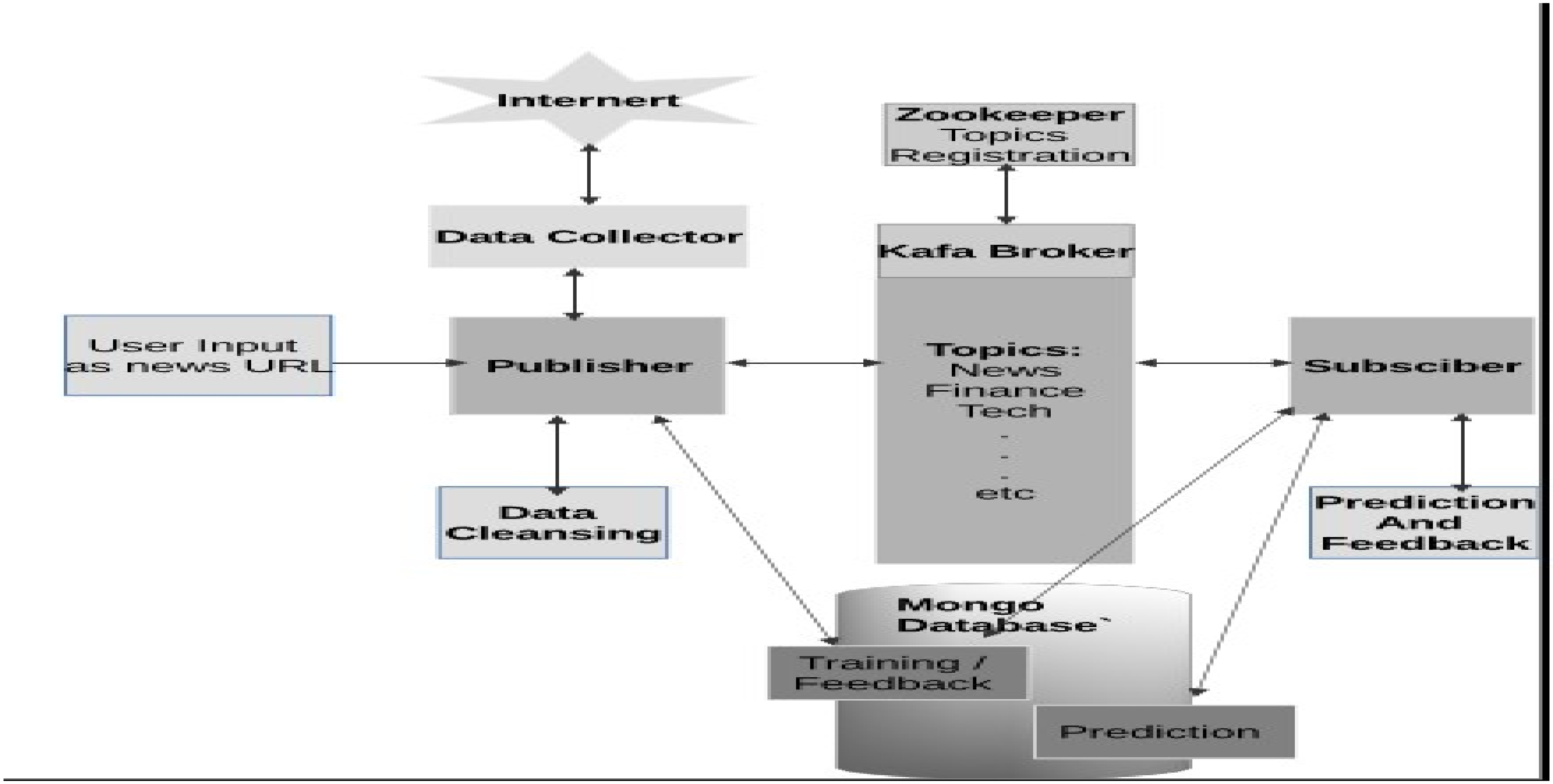
System Module Definition Document

Table of Contents

Introduction

This document contains the high level architecture for the News Classifier system. To describe the system at a high level, modules containing related low-level objects and functionality have been defined. The document then defines each module's responsibilities and inter-dependencies. Additionally, for completeness, each of the project requirements has been mapped to a particular module.

High Level Module Architecture Diagram



Module Type and Object Containment

The table below lists the system modules, their binary type, the objects that are contained within the module and a generalized bullet list of responsibilities for the module.

- All the modules are containerized and are of docker image format.
- Python programs and modules will have the .py extension
- MongoDB files will be stored with .json format.

Module Name	Binary Type	Objects Contained within module	Responsibilities
Kafka	Docker image	Kafka, zookeeper	Responsible for creating topics and streaming the messages for the topic to the broker.
Zookeeper	Docker image	Zookeeper	It will maintain the list of all active topics in the Kafka broker.
Producer	Docker image	Kafka, Spark	This program will fetch the data from freenewsapi, filter the required data from the response, connect to the kafka server, and stream messages to a specific topics in the json format.
Consumer	Docker image	Kafka, Spark	This program will retrieve the json messages from the kafka broker and will store the raw data into the mongo db
Database	Docker image	Mongo DB	The Mongo DB service that will start an instance of the Mongo server and will listen to the requests from the clients.
Spark-master	Docker image	Spark	
Spark-worker	**Docker image**	**Spark**	

Naming Convention: All the python programming naming conventions were followed. Also, the docker/docker-compose conventions were followed for all the docker images.

Tools – Denotes a generic set of utility functionality used by several sub-systems.

docker, docker-compose, zookeeper, kafka, mongodb, mongodb express, spark, python

Manager Dependencies

The table below lists the top-level system modules (a.k.a. Managers) and their interdependencies. The third column is a possible interface prototype to represent the needed module entry point for the identified dependency.

Module Name	Dependency	Interface needed
Zookeeper	None	Will run as a service in the docker container on Port 2181
Kafka	Zookeeper	Will connect to the zookeeper before initialization. This will also run as a service on Port 9092
Spark-master	None	Will run as a service on External Port 8080/Internal Port 7077
Spark-worker	Spark-master	Will run as a service on port 8081
Producer	Spark-master, Kafka, MongoDB	Will run as a service and run as a spark worker thread.
Consumer/ Prediction Service	Producer	Will run as a service and run as a spark worker thread.
Training Service	Producer	Will run as a service and run as a spark worker thread.
Feedback Service	Prediction Service, Training Service	Will run as a service and run as a spark worker thread.
MongoDB	None	Will run as a service on Port 27017

Module Requirements Mapping

The table below lists the system modules and a list of numbers. This correlation represents the module who “owns” the feature implementation. Duplicated or overlapping numbers means that the modules share the requirement, and both have the responsibility of ownership.

Module Name	Requirements Mapping
Producer	kafka, pathlib, time, bson, os, sys, requests
Consumer	kafka, json, time, pathlib, os, sys

Appendix A - Unicode Support

Data Sources

The Kafka server requires the data format to be UTF-8. The MongoDB will also store the data in the UTF-8 format only.

Mongo Drivers

Mongo drivers will be installed as part of the mongo image installation.

Appendix B – Working within Docker container boundaries

Explain the working:

Every service is running on a specific port in the container. Each service is either independent or has some dependents that it has to wait until they finish initialization, up and running. The docker-compose yaml file will contain the references to all the ports or URLs that a service has to refer and communicate.
Below is the sample:

```
producer:
    build: .
    environment:
      BROKER: kafka:9092
      DATA_BASE: mongo:27017
    command: bash -c "spark-submit --master spark://spark-master:7077 producer/send_data.py"
    depends_on:
      - spark-master
      - kafka
      - mongo
      # - mysql-db

 consumer:
    build: .
    environment:
      BROKER: kafka:9092
    command: "spark-submit --master spark://spark-master:7077 consumer/consumer.py "
    depends_on:
      - spark-master
      - kafka
      - producer
```

Benefits:

- Reduced cost of infrastructure operations
- Solution scalability on the microservice/function level
- Better security

- Instant replication of microservice
- Deploy anywhere
- Full portability between clouds and on-premises locations
- OS independent
- Fast deployment
- Lightweight
- Faster “ready to compute”

Limitations:

- Applications can’t run as fast as on a bare-metal server.
- Cross-platform compatibility is not supported for a docker container.
- Docker is good solution for applications that doesn’t require rich interfaces.
- Monitoring so many moving pieces within a dynamic, large-scale Docker environment is not so easy.
- Prior evaluation of the Docker-specific security risks is required to make sure you can handle them.

Appendix C - Programming Python with Kafka, zookeeper, spark

Producer:

Producer service will establish a connection to kafka service and stream the data using spark service. Kafka will register all the topics with zookeeper at the startup time(this is configured in docker-compose file).

For this code snippet as follows:

```
BROKER = os.getenv('BROKER', 'localhost:9092')
# TOPICS = ['news','tech']
TOPIC = 'news'
try:
    producer = KafkaProducer(bootstrap_servers=BROKER)
except Exception as e:
    print(f"ERROR --> {e}")
   sys.exit(1)
```

Using the above producer object, producer service will publish data with kafka broker. The data streaming will be in Json format.

```
producer.send(article["topic"],json.dumps(article,default=json_util.default).encode('utf-8'))
```

Consumer:

Consumer will subscribe with kafka service to consume the streamed data which producer published.

```
BROKER = os.getenv('BROKER', 'localhost:9092')
# TOPICS = ['news','tech']
TOPIC = 'news'
consumer = KafkaConsumer(
    TOPIC,
    bootstrap_servers=[BROKER],
    auto_offset_reset='earliest',
    enable_auto_commit=True,
        group_id='my-group')
```

Database:

Mongo db is used for storing the raw json data.

Appendix D – Training the ML Models, Data Cleansing, Feature Engineering

– To DO--

Appendix E – Data Processing using Apache Spark.

– To DO--

Appendix F—Applying NLTK algorithm for News article prediction.

– To Do--

2. Size

All the sizes of the docker images depends on the requirements for the image.

--To DO—

3. Performance

Docker Container

1.Method calls
2.Object creation
3.Conclusions

Python

1.Method calls
2.Object creation

3.Conclusions

Kafka/zookeeper

1.Method calls
2.Object creation
3.Conclusions

Spark

1.Method calls
2.Object creation
3.Conclusions

4. Other references:

5. Conclusion: