SOFTWARE DEFINED NETWORKS

Developing Kafka deployment and publisher/consumer for ReST based services

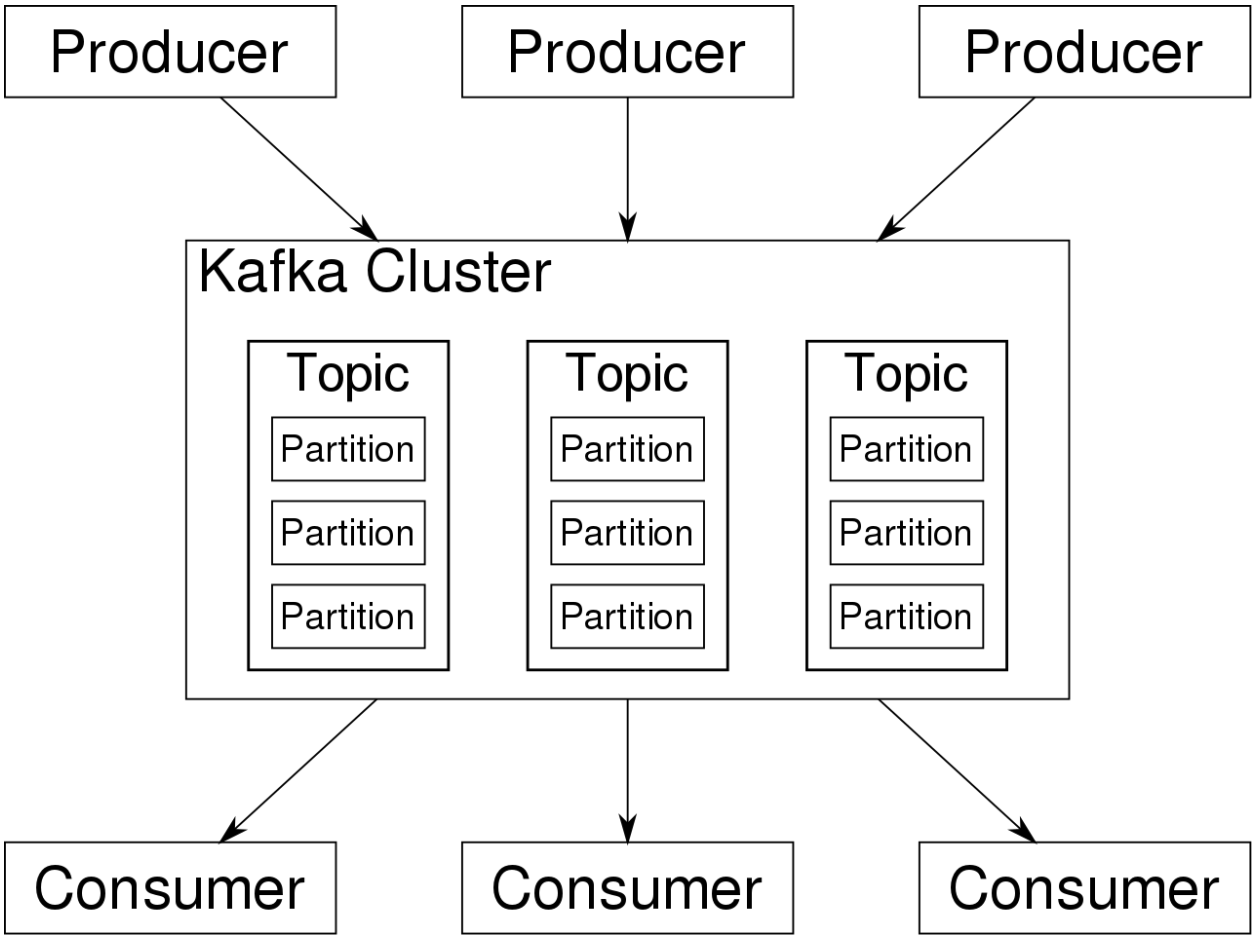
TANU BORDIA(IMT2016002) | VISHESH RUPARELIA(IMT2016006)
GITHUB: https://github.com/visheshruparelia/hackernewsKafka

Introduction

Apache Kafka is a community-based streaming event system that can handle trillions of events a day. Kafka is based on a distributed commit log abstraction, thought it was thought of as a messaging queue in the beginning. Kafka has grown rapidly from messaging queue to a full-fledged event streaming platform since it was developed and launched by LinkedIn in 2011.

There are many arbitrary processes called producers that send key-valued to messages to Kafka, which it stores. Within different "topics," data can be partitioned into different "partitions." In a partition, messages are orders based on their offsets and are indexed and stored with timestamp. Other processes called consumers reads the messages from these partitions.

Brokers are clusters of one or more servers, Kafka runs on these. Partition of all topics distributed across the cluster nodes. Also, partitions are copied to multiple brokers. This architecture enables Kafka to deliver massive messaging streams in a tolerant manner.



Three major APIs used in our project were:

- **Producer API** – Permits an application to publish streams of records.
- **Consumer API** – Permits an application to subscribe to topics and processes streams of records.
- **Connector API** – Executes the reusable producer and consumer APIs that can link the topics to the existing applications.

Problem Statement

Developing Kafka deployment and publisher/consumer for ReST based services

1. Deploy kafka
2. Assume as application is exposing data using ReST services - for that any standard API available on open network can be used.
3. There is a publisher that calls this ReST api to read the data (ensure that the ReST APIs are returning data in JSON Format) and put it on kafka topic using Java APIs.
4. There are three consumers who are reading this data from this topic. i) one of them is printing it locally ii) Sending it using ReST apis to a remote applications. Iii) Storing the data in ElasticSerach DB.

Solution

Overview:

A producer script will pull new stories posted on https://news.ycombinator.com/ and publish it to the kafka bus under the topic name “story”. Three consumers pull data from this bus and process and store it accordingly.

Language: Python
Library Dependencies: haxor, kafka-python, flask, elasticsearch

Installing dependencies:

1. cd into the root directory of the project.
2. Run “pip3 install -r requirements.txt”
3. **Install Docker:**
    a. **https://docs.docker.com/v17.12/install/linux/docker-ce/ubuntu/#install-using-the-repository**

4. **Install Docker-compose:**
    a. **https://docs.docker.com/compose/install/**

Steps to run:

1. **Kafka, Zookeeper, ElasticSearch, Kibana:**
    a. cd into root directory of the project
    b. Run “sudo docker-compose up --no-recreate”

    Outcomes:

    - Kafka will run on port 9092
    - Zookeeper will run on port 2181
    - ElasticSearch will run on port 9200
    - Kibana will run on port 5601

2. **Producer:**
    a. cd into the root directory of the project
    b. Run “python3 producer.py”

    - **Outcome:** The script will keep publishing data to a topic named “story” in the kafka bus.

3. **Local Consumer:**
    a. cd into local_consumer folder
    b. Run “python3 consumer.py”

    - **Outcome:** The script will pull the data from the bus and append it to consumerdata.csv

4. **Remote Consumer:**
    a. cd into flask_app folder
    b. Run “python3 consumerflask.py”

    - **Outcome:** Webpage will be hosted on http://localhost:5000

5. **Elastic Search Consumer:**
    a. cd into elastic_consmer folder
    b. Run “python3 consumer.py”

- **Outcome:** Data will be indexed to ElasticSearch under index “hn_story” and can be viewed from Kibana console at http://localhost:5601

The following section will also cover results and observations:

Documentation:

1. **HackerNews API**: Exposes data continuously
   Documentation: https://github.com/HackerNews/API

```
while(True):
    story=hn.new_stories(limit=1)
    if(len(story)==0 or story[0].item_id<=latest_id):
        continue
    else:
        start=latest_id+1
        for i in range(start,story[0].item_id+1):
            curr_story=hn.get_item(i)
            if(curr_story.item_type!='story'):
                continue
            title=curr_story.title
            id=curr_story.item_id
            by=curr_story.by
            time=item.submission_time
            a=time.strftime("%m/%d/%Y, %H:%M:%S")
            data={'id':id,'title':title,'by':by,'time':a}
            latest_id=id
            print('publishing {}'.format(data))
            producer.send('story',value=data)
            print('\n')
    sleep(5)
```

   Will call “new_stories” method every 5 seconds.

2. **Kafka Producer:** A Kafka client that publishes records to the Kafka cluster.
   The producer consists of a pool of buffer space that holds records that haven't yet been transmitted to the server as well as a background I/O thread that is responsible for turning these records into requests and transmitting them to the cluster.

   File location: producer.py

   Command to run: python3 producer.py

   It keeps pulling new stories posted on the HackerNews forum

```
from kafka import KafkaProducer

producer = KafkaProducer(bootstrap_servers=['localhost:9092'],
                         value_serializer=lambda x:
                          dumps(x).encode('utf-8'))
```

   When run, Kafka producer gets data from the HackerNews and sends it to the consumer.

```
producer.send('story',value=data)
```

3. **Kafka Consumers:** Consume records from a Kafka cluster.
   There are 3 consumer groups:

a. **CSVConsumer:** Consumes the data and stores it in a CSV file on the local system which gets updated continuously. (Local Consumer)

```
consumer = KafkaConsumer(
    'story',
     bootstrap_servers=['localhost:9092'],
     auto_offset_reset='earliest',
     enable_auto_commit=True,
     group_id='CSVConsumer',
     value_deserializer=lambda x: loads(x.decode('utf-8')))
```

File Location: local_consumer/consumer.py

Output Location: local_consumer/consumerdata.csv

Output Screenshot:

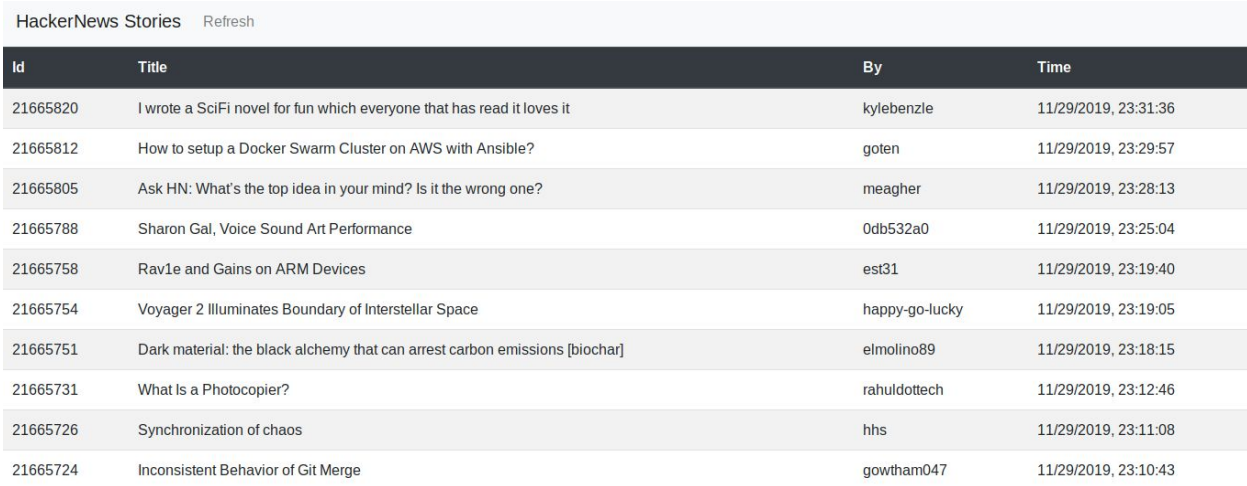
	A	B	C	D
1	id	title	by	time
2	21655958	Firefox Replay	nachtigall	11/28/2019, 15:29:47
3	21655951	Bill Gates-backed Heliogen targets 1,500°C solar thermal	nclzz	11/28/2019, 15:28:15
4	21655945	QuestDB – Fast open source relational time series database	beagle3	11/28/2019, 15:27:03
5	21655940	Use Tensor Flow Lite and a Particle Xenon to Build an ML-Gesture Wand	adunk	11/28/2019, 15:25:58
6	21655936	Test Automation Without Coding	dsheiko	11/28/2019, 15:25:22
7	21655926	Biological enzymes as source of hydrogen fuel	conse_lad	11/28/2019, 15:23:59
8	21655903	Replace Windows 7 with Linux	dagurp	11/28/2019, 15:19:40
9	21655897	How the FCC Helped Pave the Way for Predatory Prison Telecoms	vezycash	11/28/2019, 15:18:52
10	21655892	That Uplifting Tweet You Just Shared? A Russian Troll Sent It	edoloughlin	11/28/2019, 15:18:14
11	21655890	The blockers and catalysts of digital transformation	mooreds	11/28/2019, 15:17:11

b. **FlaskConsumer:** It is a flask app which consumes data from the bus under the group name “FlaskConsumer” and publishes the data to localhost:5000. Two threads run, one is the main flask app and the other one keeps pulling data from the bus.

File Location: flask_app/consumerflask.py

Output: Can be accessed at http://localhost:5000, refresh page every time to see new records.

Output Screenshot:



HackerNews Stories Refresh

Id	Title	By	Time
21665820	I wrote a SciFi novel for fun which everyone that has read it loves it	kylebenzle	11/29/2019, 23:31:36
21665812	How to setup a Docker Swarm Cluster on AWS with Ansible?	goten	11/29/2019, 23:29:57
21665805	Ask HN: What's the top idea in your mind? Is it the wrong one?	meagher	11/29/2019, 23:28:13
21665788	Sharon Gal, Voice Sound Art Performance	0db532a0	11/29/2019, 23:25:04
21665758	Rav1e and Gains on ARM Devices	est31	11/29/2019, 23:19:40
21665754	Voyager 2 Illuminates Boundary of Interstellar Space	happy-go-lucky	11/29/2019, 23:19:05
21665751	Dark material: the black alchemy that can arrest carbon emissions [biochar]	elmolino89	11/29/2019, 23:18:15
21665731	What Is a Photocopier?	rahuldottech	11/29/2019, 23:12:46
21665726	Synchronization of chaos	hhs	11/29/2019, 23:11:08
21665724	Inconsistent Behavior of Git Merge	gowtham047	11/29/2019, 23:10:43

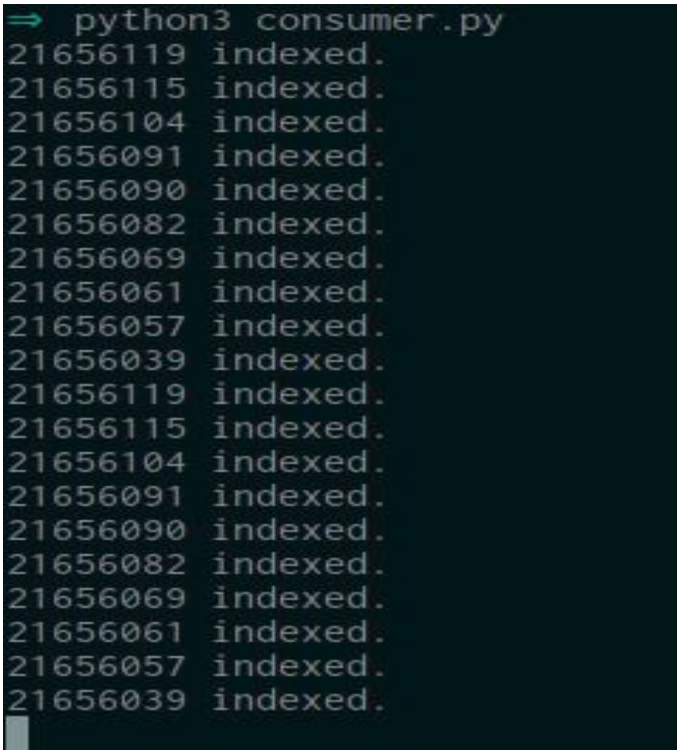
c. **ElasticSearch Consumer:**

Using “elasticsearch” module available in python, the script consumes data from the bus as the group “ElasticConsumer” and indexes it to the ElasticSearch DB instance running on localhost:9200 under the index name “hn_story” and doc type “test” with the id of the data being the id of the HN story.

A kibana instance connects with this elastic search instance and displays the data at localhost:5601.

NOTE: When opening kibana for the first time, an index named “hn_story” needs to be created.

File Location: elastic_consumer/consumer.py



Kibana Output: