Getting started with Apache Kafka in Python

Adnan Siddiqi · Follow
Published in Towards Data Science
9 min read · Jun 11, 2018

Listen Share More

Image Credit: linuxhint.com

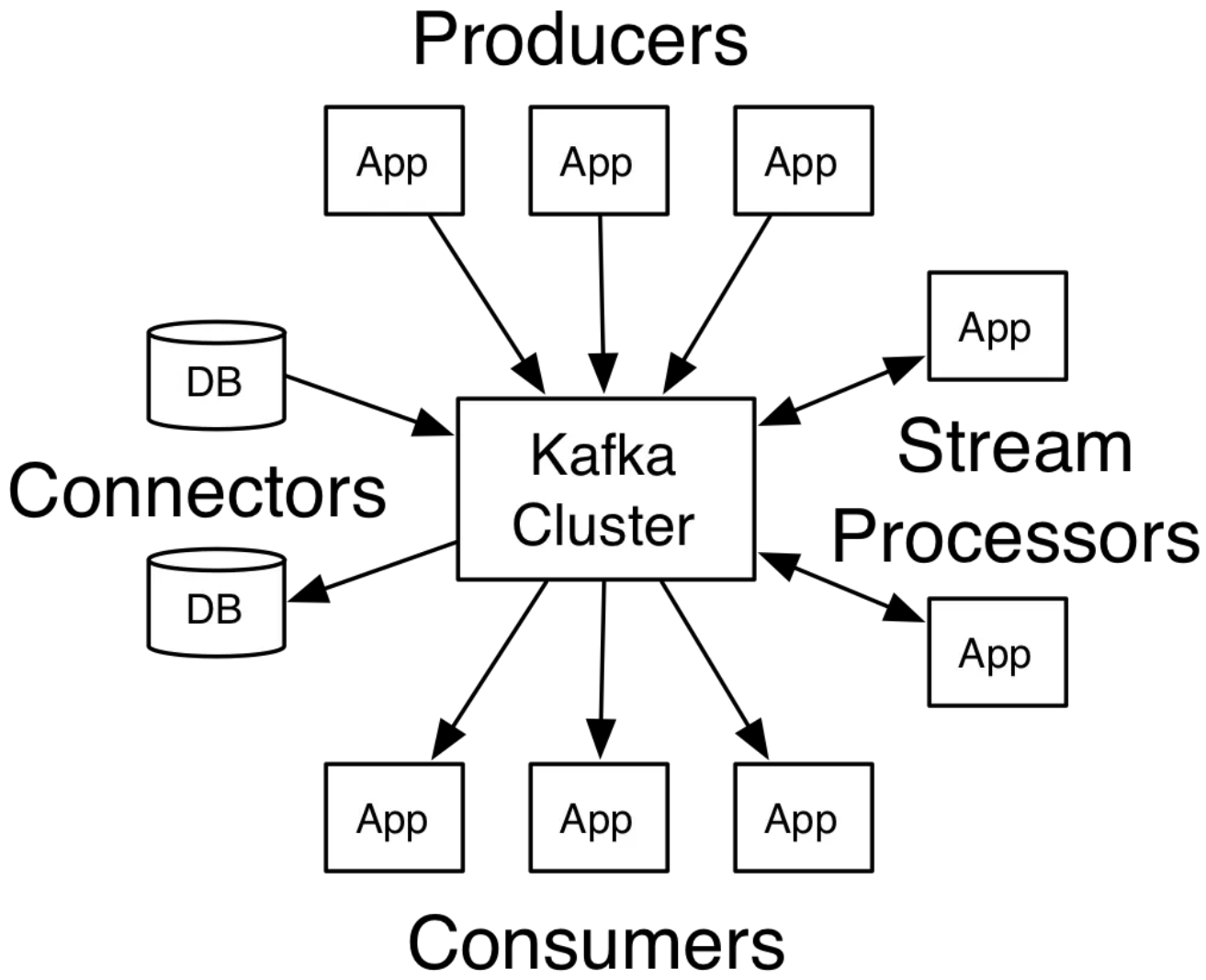
In this post, I am going to discuss Apache Kafka and how Python programmers can use it for building distributed systems.

What is Apache Kafka?

Apache Kafka is an open-source streaming platform that was initially built by LinkedIn. It was later handed over to Apache foundation and open sourced it in 2011.

According to **Wikipedia**:

Apache Kafka is an open-source stream-processing software platform developed by the Apache Software Foundation, written in Scala and Java. The project aims to provide a ***unified, high-throughput, low-latency*** *platform for handling* ***real-time data feeds****. Its storage layer is essentially a "massively scalable pub/sub message queue architected as a distributed transaction log,"[3] making it highly valuable for enterprise infrastructures to process streaming data. Additionally, Kafka connects to external systems (for data import/export) via Kafka Connect and provides Kafka Streams, a Java stream processing library.*



Credit: Official Website

Think of it is a big commit log where data is stored in sequence as it happens. The users of this log can just access and use it as per their requirement.

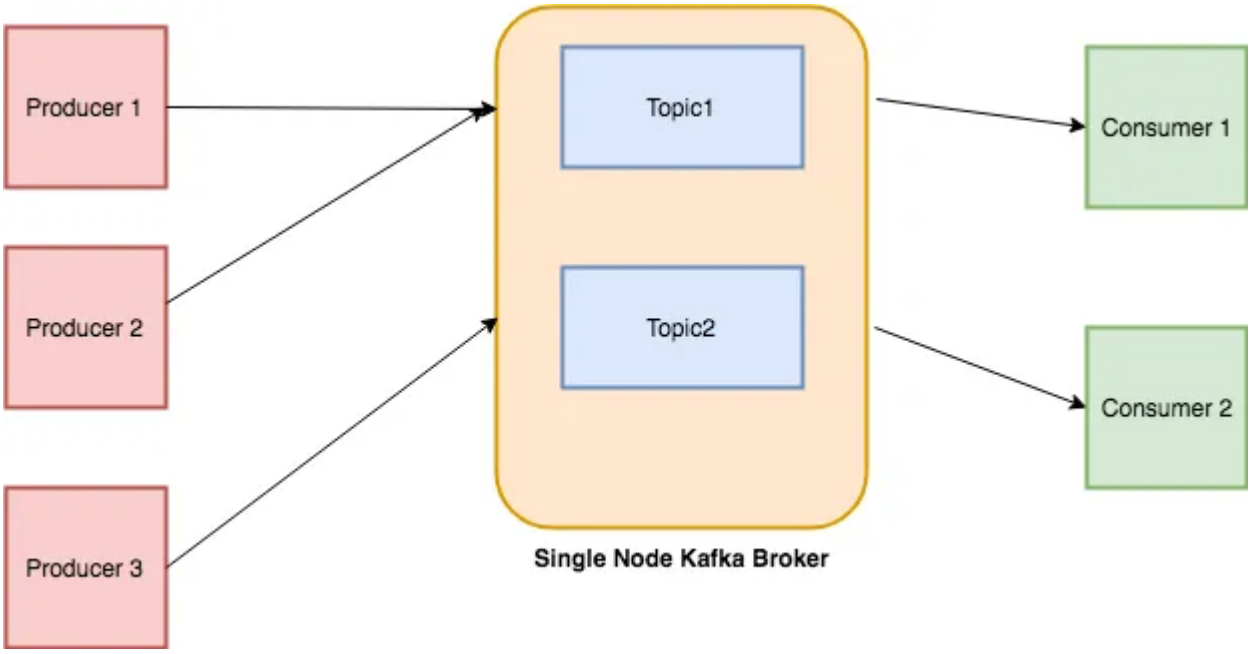
Kafka Use Cases

Uses of Kafka are multiple. Here are a few use-cases that could help you to figure out its usage.

- **Activity Monitoring:**- Kafka can be used for activity monitoring. The activity could belong to a website or physical sensors and devices. Producers can publish raw data from data sources that later can be used to find trends and pattern.
- **Messaging:**- Kafka can be used as a message broker among services. If you are implementing a microservice architecture, you can have a microservice as a producer and another as a consumer. For instance, you have a microservice that is responsible to create new accounts and other for sending email to users about account creation.
- **Log Aggregation:**- You can use Kafka to collect logs from different systems and store in a centralized system for further processing.
- **ETL:**- Kafka has a feature of almost real-time streaming thus you can come up with an ETL based on your need.
- **Database:**- Based on things I mentioned above, you may say that Kafka also acts as a database. Not a typical databases that have a feature of querying the data as per need, what I meant that you can keep data in Kafka as long as you want without consuming it.

Kafka Concepts

Let’s discuss core Kafka concepts.



Topics

Every message that is feed into the system must be part of some topic. The topic is nothing but a stream of records. The messages are stored in key-value format. Each message is assigned a sequence, called *Offset*. The output of one message could be an input of the other for further processing.

Producers

Producers are the apps responsible to publish data into Kafka system. They publish data on the topic of their choice.

Consumers

The messages published into topics are then utilized by *Consumers* apps. A consumer gets subscribed to the topic of its choice and consumes data.

Broker

Every instance of Kafka that is responsible for message exchange is called a Broker. Kafka can be used as a stand-alone machine or a part of a cluster.

I try to explain the whole thing with a simple example, there is a warehouse or godown of a restaurant where all the raw material is dumped like rice, vegetables etc. The restaurant serves different kinds of dishes: Chinese, Desi, Italian etc. The chefs of each cuisine can refer to the warehouse, pick the desire things and make things. There is a possibility that the stuff made by the raw material can later be used by all departments'

chefs, for instance, some secret sauce that is used in ALL kind of dishes. Here, the warehouse is a *broker*, vendors of goods are the *producers*, the goods and the secret sauce made by chefs are *topics* while chefs are *consumers*. My analogy might sound funny and inaccurate but at least it'd have helped you to understand the entire thing :-)

Setting up and Running

The easiest way to install Kafka is to download binaries and run it. Since it's based on JVM languages like Scala and Java, you must make sure that you are using Java 7 or greater.

Kafka is available in two different flavors: One by **Apache foundation** and other by **Confluent** as a **package**. For this tutorial, I will go with the one provided by Apache foundation. By the way, Confluent was founded by the **original developers** of Kafka.

Starting Zookeeper

Kafka relies on Zookeeper, in order to make it run we will have to run Zookeeper first.

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

it will display lots of text on the screen, if see the following it means it's up properly.

```
2018-06-10 06:36:15,023] INFO maxSessionTimeout set to -1
(org.apache.zookeeper.server.ZooKeeperServer)
[2018-06-10 06:36:15,044] INFO binding to port 0.0.0.0/0.0.0.0:2181
(org.apache.zookeeper.server.NIOServerCnxnFactory)
```

Starting Kafka Server

Next, we have to start Kafka broker server:

```
bin/kafka-server-start.sh config/server.properties
```

And if you see the following text on the console it means it's up.

```
2018-06-10 06:38:44,477] INFO Kafka commitId : fdcf75ea326b8e07
(org.apache.kafka.common.utils.AppInfoParser)
```

```
[2018-06-10 06:38:44,478] INFO [KafkaServer id=0] started
(kafka.server.KafkaServer)
```

Create Topics

Messages are published in topics. Use this command to create a new topic.

```
➜  kafka_2.11-1.1.0 bin/kafka-topics.sh --create --zookeeper
localhost:2181 --replication-factor 1 --partitions 1 --topic test
Created topic "test".
```

You can also list all available topics by running the following command.

```
➜  kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper
localhost:2181
test
```

As you see, it prints, `test` .

Sending Messages

Next, we have to send messages, *producers* are used for that purpose. Let's initiate a producer.

```
➜  kafka_2.11-1.1.0 bin/kafka-console-producer.sh --broker-list
localhost:9092 --topic test
>Hello
>World
```

You start the console based producer interface which runs on the port `9092` by default. `--topic` allows you to set the topic in which the messages will be published. In our case the topic is `test`

It shows you a `>` prompt and you can input whatever you want.

Messages are stored locally on your disk. You can learn about the path of it by checking the value of `log.dirs` in `config/server.properties` file. By default they are set to `/tmp/kafka-logs/`

If you list this folder you will find a folder with name `test-0`. Upon listing it you will find 3 files: `00000000000000000000.index 00000000000000000000.log 00000000000000000000.timeindex`

If you open `00000000000000000000.log` in an editor then it shows something like:

```
^@^@^@^@^@^@^@^@^@^@=^@^@^@^@^BÐØR^V^@^@^@^@^@^@^@^Acça<9a>o^@^@^A
cça<9a>oÿÿÿÿÿÿÿÿÿÿÿÿ^@^@^@^A^V^@^@^@^A
Hello^@^@^@^@^@^@^@^A^@^@^@=^@^@^@^@^BÉJ^B-
^@^@^@^@^@^@^@^@^Acça<9f>^?^@^@^Acça<9f>^?
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ^@^@^@^A^V^@^@^@^A
World^@
~
```

Looks like the encoded data or delimiter separated, I am not sure. If someone knows this format then do let me know.

Anyways, Kafka provides a utility that lets you examine each incoming message.

```
➜  kafka_2.11-1.1.0 bin/kafka-run-class.sh
kafka.tools.DumpLogSegments --deep-iteration --print-data-log --files
/tmp/kafka-logs/test-0/00000000000000000000.log
Dumping /tmp/kafka-logs/test-0/00000000000000000000.log
Starting offset: 0
offset: 0 position: 0 CreateTime: 1528595323503 isvalid: true keysize:
-1 valuesize: 5 magic: 2 compresscodec: NONE producerId: -1
producerEpoch: -1 sequence: -1 isTransactional: false headerKeys: []
payload: Hello
offset: 1 position: 73 CreateTime: 1528595324799 isvalid: true
keysize: -1 valuesize: 5 magic: 2 compresscodec: NONE producerId: -1
producerEpoch: -1 sequence: -1 isTransactional: false headerKeys: []
payload: World
```

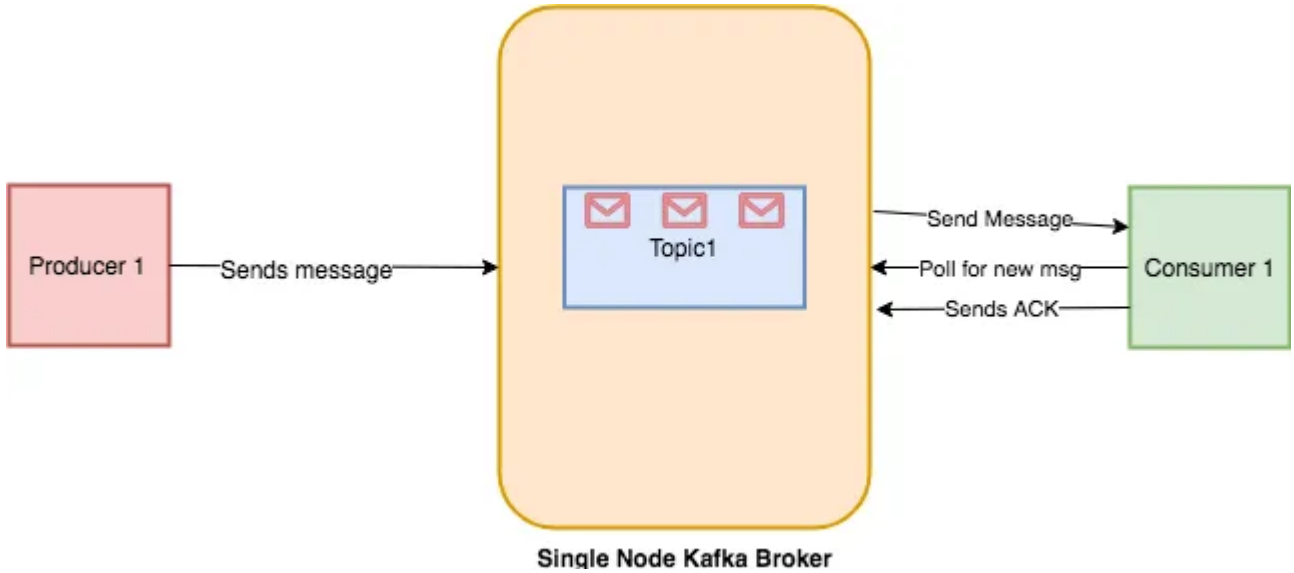
You can see the message with other details like `offset`, `position` and `CreateTime` etc.

Consuming Messages

Messages that are stored should be consumed too. Let's started a console based consumer.

```
➜  kafka_2.11-1.1.0 bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic test --from-beginning
```

If you run, it will dump all the messages from the beginning till now. If you are just interested to consume the messages after running the consumer then you can just omit `--from-beginning` switch it and run. The reason it does not show the old messages because the offset is updated once the consumer sends an ACK to the Kafka broker about processing messages. You can see the workflow below.



Accessing Kafka in Python

There are multiple Python libraries available for usage:

- **Kafka-Python** — An open-source community-based library.
- **PyKafka** — This library is maintained by Parsly and it's claimed to be a Pythonic API. Unlike Kafka-Python you can't create dynamic topics.
- **Confluent Python Kafka**:- It is offered by Confluent as a thin wrapper around **librdkafka**, hence it's performance is better than the two.

For this post, we will be using the open-source *Kafka-Python*.

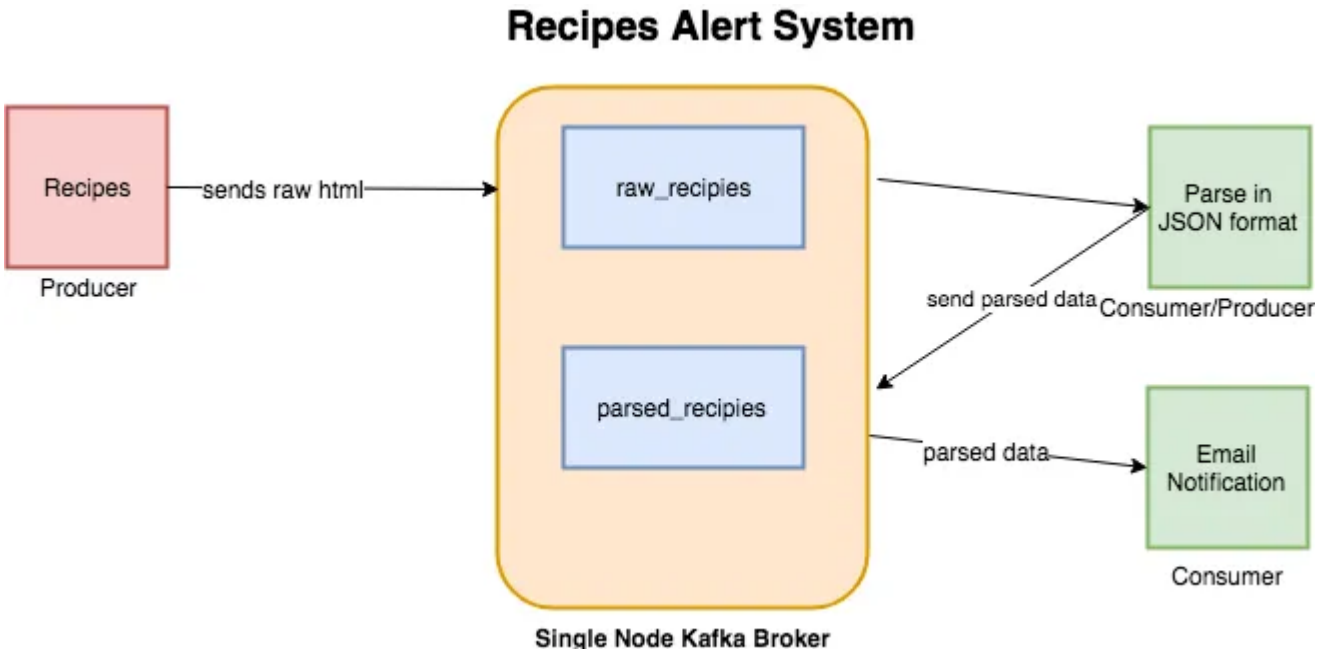
Recipes Alert System in Kafka

In the last **post** about Elasticsearch, I scraped Allrecipes data. In this post, I am going to use the same scraper as a data source. The system we are going to build is an alert system which will send notification about the recipes if it meets the certain threshold of the calories. There will be two topics:

- **raw_recipes:**- It will be storing the raw HTML of each recipe. The idea is to use this topic as the main source of our data that later can be processed and transformed as per need.
- **parsed_recipes:**- As the name suggests, this will be parsed data of each recipe in JSON format.

The length of Kafka topic name should not exceed **249**.

A typical workflow will look like below:



Install `kafka-python` via `pip`

```
pip install kafka-python
```

Raw recipe producer

The first program we are going to write is the producer. It will access Allrecpies.com and fetch the raw HTML and store in **raw_recipes** topic.

```python
def fetch_raw(recipe_url):
    html = None
    print('Processing..{}'.format(recipe_url))
    try:
        r = requests.get(recipe_url, headers=headers)
        if r.status_code == 200:
            html = r.text
    except Exception as ex:
        print('Exception while accessing raw html')
        print(str(ex))
    finally:
        return html.strip()


def get_recipes():
    recipies = []
    salad_url = 'https://www.allrecipes.com/recipes/96/salad/'
    url = 'https://www.allrecipes.com/recipes/96/salad/'
    print('Accessing list')

    try:
        r = requests.get(url, headers=headers)
        if r.status_code == 200:
            html = r.text
            soup = BeautifulSoup(html, 'lxml')
            links = soup.select('.fixed-recipe-card__h3 a')
            idx = 0
            for link in links:

                sleep(2)
                recipe = fetch_raw(link['href'])
                recipies.append(recipe)
                idx += 1
                if idx > 2:
                    break
    except Exception as ex:
        print('Exception in get_recipes')
        print(str(ex))
    finally:
        return recipies
```

producer-raw-recipies.py hosted with ❤ by GitHub view raw

This code snippet will extract markup of each recipe and return in `list` format.

Next, we to create a producer object. Before we proceed further, we will make changes in `config/server.properties` file. We have to set `advertised.listeners` to `PLAINTEXT://localhost:9092` otherwise you could experience the following error:

```
Error encountered when producing to broker b'adnans-mbp':9092. Retrying.
```

We will now add two methods: `connect_kafka_producer()` that will give you an instance of Kafka producer and `publish_message()` that will just dump the raw HTML of individual recipes.

```
def publish_message(producer_instance, topic_name, key, value):
    try:
        key_bytes = bytes(key, encoding='utf-8')
        value_bytes = bytes(value, encoding='utf-8')
        producer_instance.send(topic_name, key=key_bytes, value=value_bytes)
```



```
        print(str(ex))


def connect_kafka_producer():
    _producer = None
    try:
        _producer = KafkaProducer(bootstrap_servers=['localhost:9092'], api_version=(0, 10))
    except Exception as ex:
        print('Exception while connecting Kafka')
        print(str(ex))
    finally:
        return _producer
```

producer-raw-recipies.py hosted with ❤ by GitHub view raw

The `__main__` will look like below:

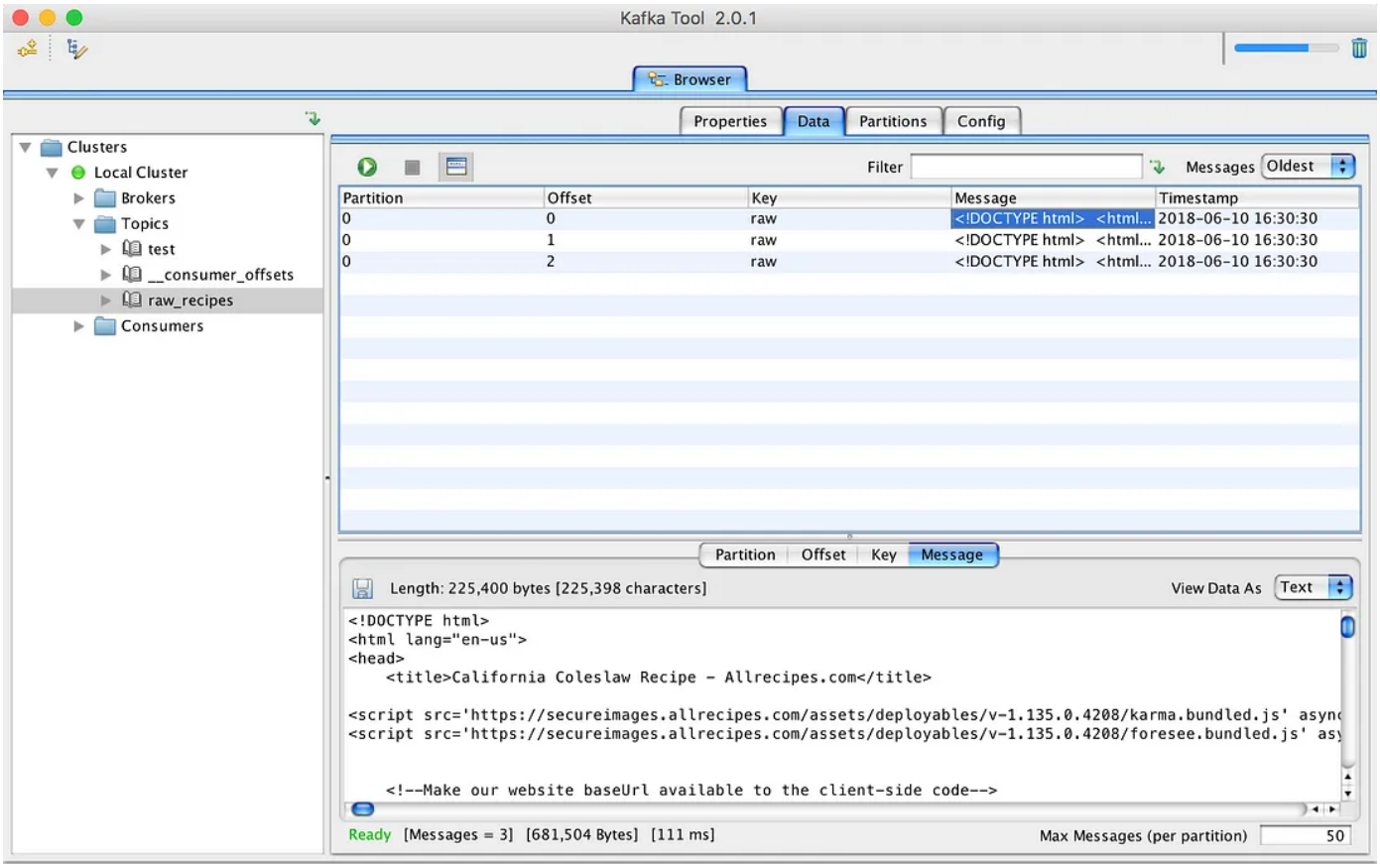
```
if __name__ == '__main__':
    headers = {
```

```
        'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181
Safari/537.36',
        'Pragma': 'no-cache'
    }

all_recipes = get_recipes()
    if len(all_recipes) > 0:
        kafka_producer = connect_kafka_producer()
        for recipe in all_recipes:
            publish_message(kafka_producer, 'raw_recipes', 'raw',
recipe.strip())
        if kafka_producer is not None:
            kafka_producer.close()
```

If it runs well, it shows the following output:

```
/anaconda3/anaconda/bin/python /Development/DataScience/Kafka/kafka-
recipie-alert/producer-raw-recipies.py
Accessing list
Processing..https://www.allrecipes.com/recipe/20762/california-
coleslaw/
Processing..https://www.allrecipes.com/recipe/8584/holiday-chicken-
salad/
Processing..https://www.allrecipes.com/recipe/80867/cran-broccoli-
salad/
Message published successfully.
Message published successfully.
Message published successfully.

Process finished with exit code 0
```

I am using a GUI tool, named as **Kafka Tool** to browse recently published messages. It is available for OSX, Windows and Linux.



KafkaToolKit in action

Recipe Parser

The next script we are going to write will serve as both consumer and producer. First it will consume data from `raw_recipes` topic, parse and transform data into JSON and then will publish it in `parsed_recipes` topic. Below is the code that will fetch HTML data from `raw_recipes` topic, parse and then feed into `parsed_recipes` topic.

```
import json
from time import sleep

from bs4 import BeautifulSoup
from kafka import KafkaConsumer, KafkaProducer


def publish_message(producer_instance, topic_name, key, value):
    try:
        key_bytes = bytes(key, encoding='utf-8')
        value_bytes = bytes(value, encoding='utf-8')
        producer_instance.send(topic_name, key=key_bytes, value=value_bytes)
        producer_instance.flush()
        print('Message published successfully.')
    except Exception as ex:
        print('Exception in publishing message')
        print(str(ex))


def connect_kafka_producer():
    _producer = None
    try:
        _producer = KafkaProducer(bootstrap_servers=['localhost:9092'], api_version=(0, 10))
    except Exception as ex:
        print('Exception while connecting Kafka')
        print(str(ex))
    finally:
        return _producer


def parse(markup):
    title = '-'
    submit_by = '-'
    description = '-'
    calories = 0
    ingredients = []
    rec = {}

    try:

        soup = BeautifulSoup(markup, 'lxml')
        # title
        title_section = soup.select('.recipe-summary__h1')
        # submitter
        submitter_section = soup.select('.submitter__name')
```

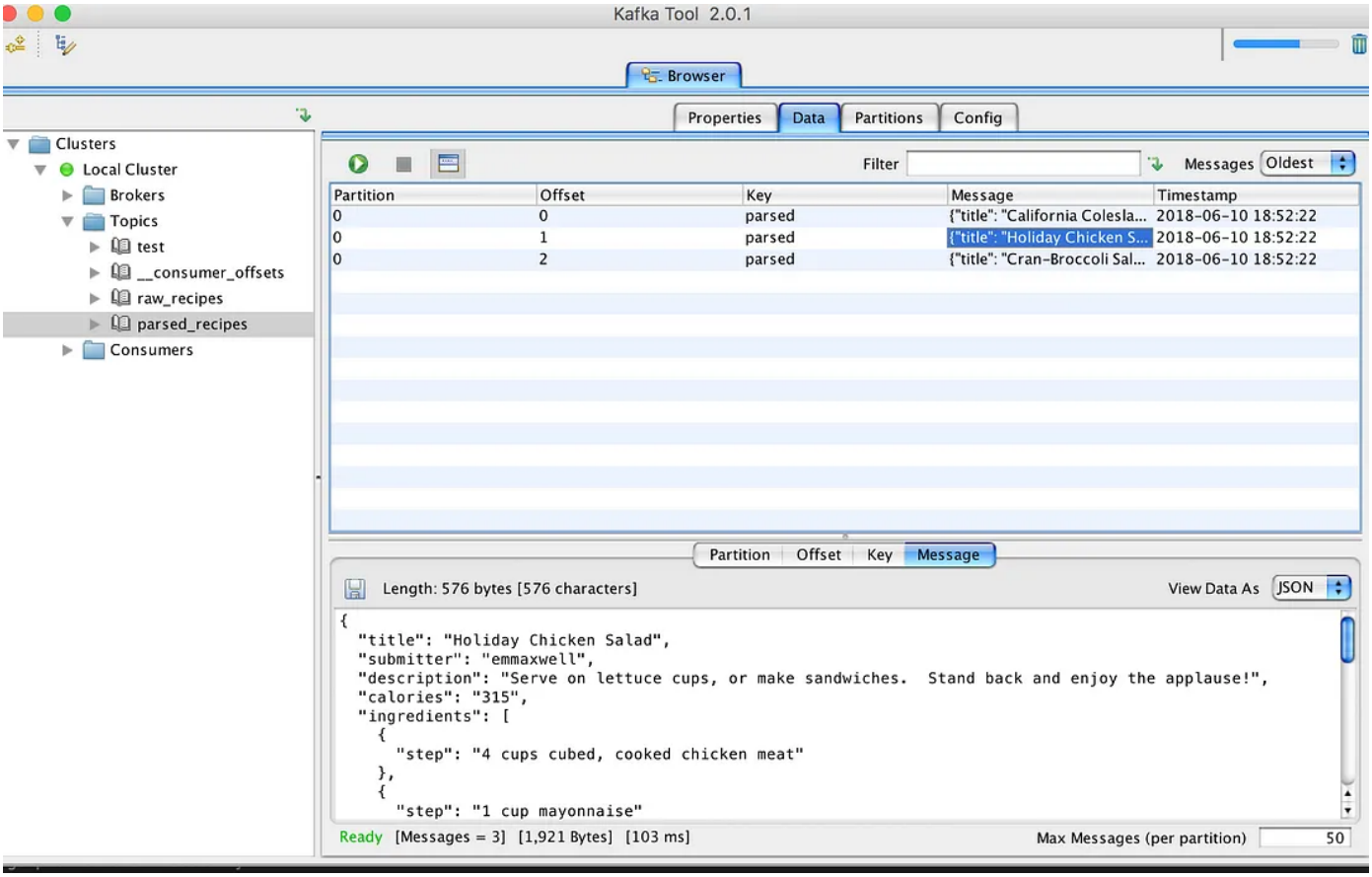
```
        submitter_section = soup.select('.submitter__name')
        # description
        description_section = soup.select('.submitter__description')
        # ingredients
        ingredients_section = soup.select('.recipe-ingred_txt')

        # calories
        calories_section = soup.select('.calorie-count')
        if calories_section:
            calories = calories_section[0].text.replace('cals', '').strip()

        if ingredients_section:
            for ingredient in ingredients_section:
                ingredient_text = ingredient.text.strip()
                if 'Add all ingredients to list' not in ingredient_text and ingredient_text != ''
                    ingredients.append({'step': ingredient.text.strip()})

        if description_section:
            description = description_section[0].text.strip().replace('"', '')

        if submitter_section:
            submit_by = submitter_section[0].text.strip()

        if title_section:
            title = title_section[0].text

        rec = {'title': title, 'submitter': submit_by, 'description': description, 'calories': ca
               'ingredients': ingredients}

    except Exception as ex:
        print('Exception while parsing')
        print(str(ex))
    finally:
        return json.dumps(rec)


if __name__ == '__main__':
    print('Running Consumer..')
    parsed_records = []
    topic_name = 'raw_recipes'
    parsed_topic_name = 'parsed_recipes'

    consumer = KafkaConsumer(topic_name, auto_offset_reset='earliest',
                             bootstrap_servers=['localhost:9092'], api_version=(0, 10), consumer_
    for msg in consumer:
```

```
            html = msg.value
            result = parse(html)
            parsed_records.append(result)
        consumer.close()
        sleep(5)

        if len(parsed_records) > 0:
            print('Publishing records..')
            producer = connect_kafka_producer()
            for rec in parsed_records:
                publish_message(producer, parsed_topic_name, 'parsed', rec)
```

producer_consumer_parse_recipes.py hosted with ❤ by GitHub view raw

`KafkaConsumer` accepts a few parameters beside the topic name and host address. By providing `auto_offset_reset='earliest'` you are telling Kafka to return messages from the beginning. The parameter `consumer_timeout_ms` helps the consumer to disconnect after the certain period of time. Once disconnected, you can close the consumer stream by calling `consumer.close()`

After this, I am using same routines to connect producers and publish parsed data in the new topic. *KafaTool* browser gives glad tidings about newly stored messages.



So far so good. We stored recipes in both raw and JSON format for later use. Next, we have to write a consumer that will connect with `parsed_recipes` topic and generate alert if certain `calories` critera meets.

The JSON is decoded and then check the calories count, a notification is issued once the criteria meet.

Conclusion

Kafka is a scalable, fault-tolerant, publish-subscribe messaging system that enables you to build distributed applications. Due to its **high performance and efficiency**, it's getting popular among companies that are producing loads of data from various external sources and want to provide real-time findings from it. I have just covered the

gist of it. Do explore the docs and existing implementation and it will help you to understand how it could be the best fit for your next system.

The code is available on **Github**.

This article originally published here.

Click here to subscribe my newsletter for future posts.

Big Data | Kafka | Python | Distributed Systems | Data Science

Follow

Written by Adnan Siddiqi

2.9K Followers · Writer for Towards Data Science

Pakistani | Husband | Father | Software Consultant | Developer | blogger. I occasionally try to make stuff with code. http://adnansiddiqi.me

More from Adnan Siddiqi and Towards Data Science

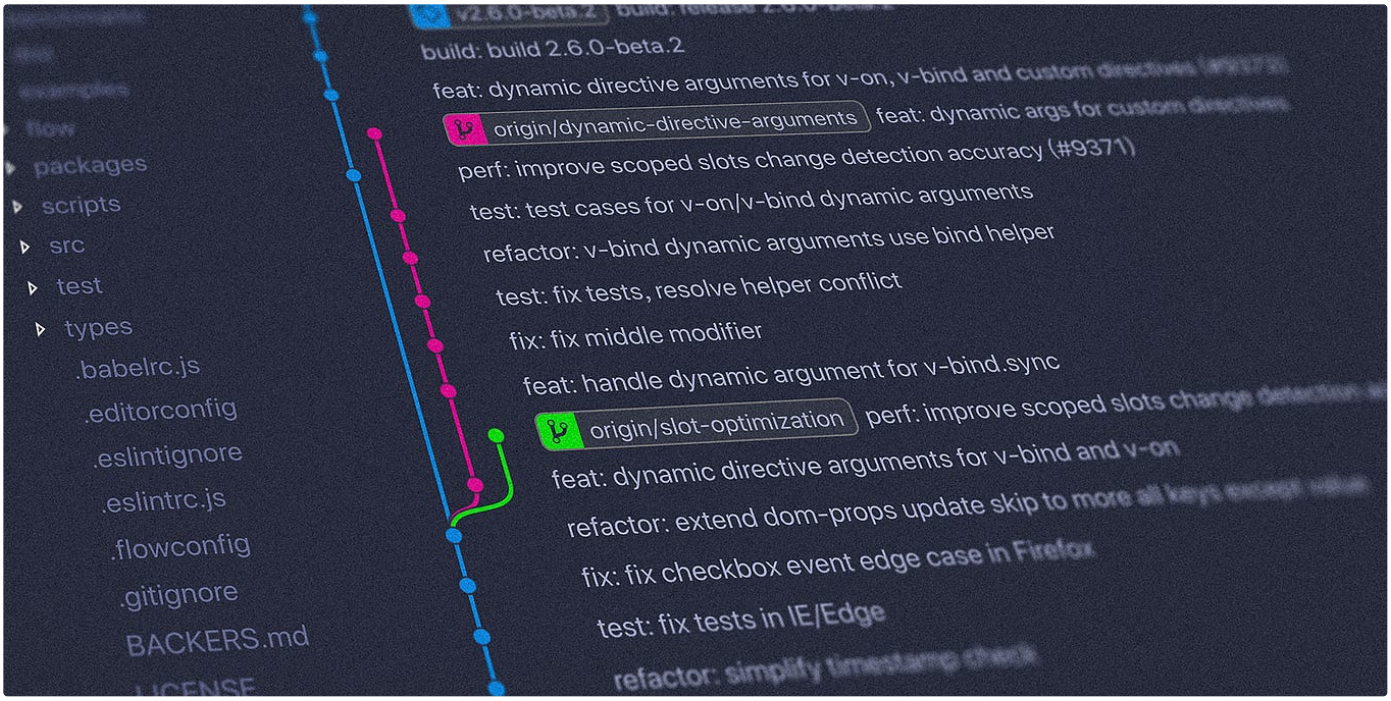
Adnan Siddiqi in Towards Data Science

Create your first ETL Pipeline in Apache Spark and Python

In this post, I am going to discuss Apache Spark and how you can create simple but robust ETL pipelines in it. You will learn how Spark...

8 min read · Jun 9, 2019

789 5



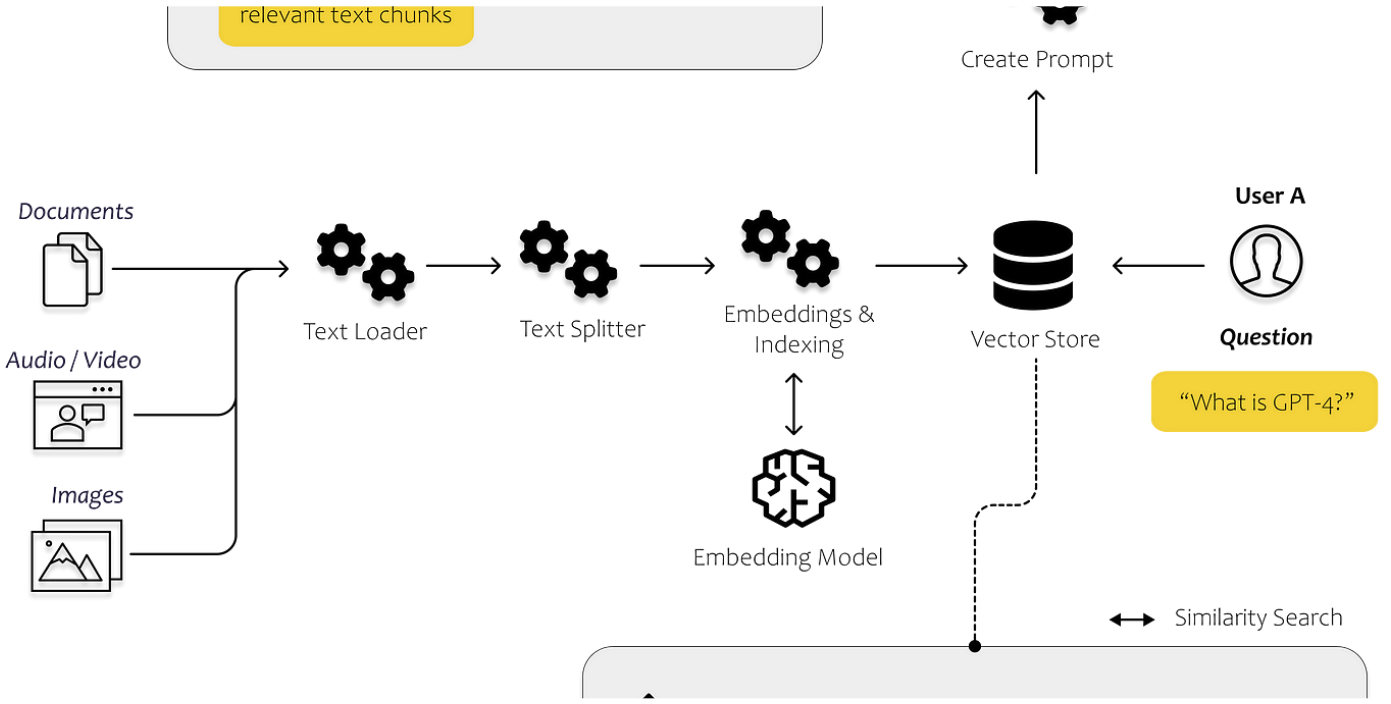
Miriam Santos in Towards Data Science

Pandas 2.0: A Game-Changer for Data Scientists?

The Top 5 Features for Efficient Data Manipulation

7 min read · Jun 27

1.98K 24



Dominik Polzer in Towards Data Science

All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt templates

✨ · 26 min read · Jun 22

3.3K 29

elasticsearch

Adnan Siddiqi in Towards Data Science

Getting started with Elasticsearch in Python

12 min read · Jun 2, 2018

2.6K 10

See all from Adnan Siddiqi

See all from Towards Data Science

Recommended from Medium

Thomas Reid in Level Up Coding

Streaming change data capture (CDC) data between databases using Kafka

This article will briefly introduce Kafka, how to connect database sources to it using the Kafka SQL client ksqlDB and create and...

21 min read · Mar 26

86

Sai Parvathaneni

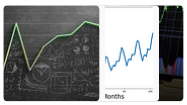
Kafka: Multi-Node Cluster Simplified

Introduction to Kafka, explained like you are 10.

· 6 min read · Apr 7

4

Lists



Predictive Modeling w/ Python

18 stories · 176 saves

Coding & Development

11 stories · 77 saves

New_Reading_List

174 stories · 40 saves

Practical Guides to Machine Learning

10 stories · 194 saves

Wei-Meng Lee in Towards Data Science

Using Apache Kafka for Data Streaming

Learn how to install and use Kafka to send and receive messages

· 6 min read · Mar 1

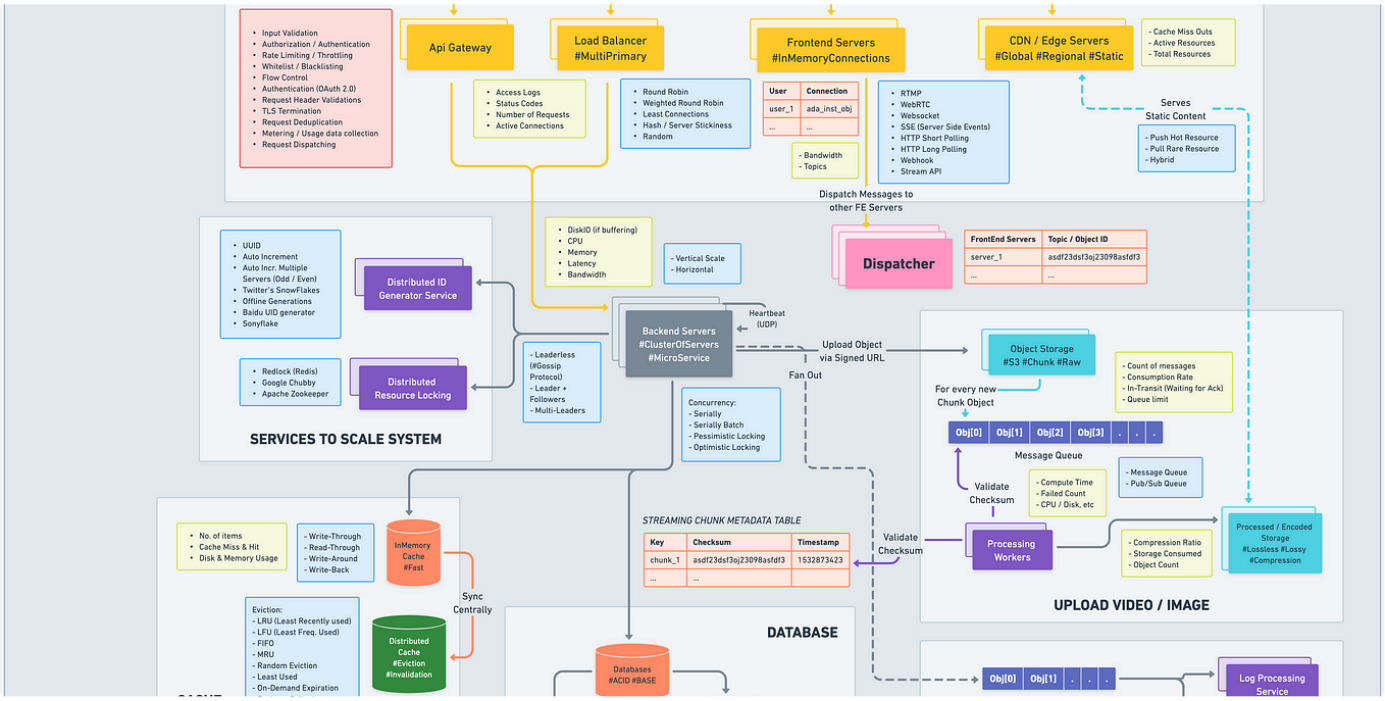
Mori

Fastest way to implement an ETL pipeline

with Airflow and Docker

✦ · 8 min read · Mar 15

215



Love Sharma in ByteByteGo System Design Alliance

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However, understanding the key concepts and components can make the...

✦ · 9 min read · Apr 20

6.8K 54

DataGeeks

Apache Airflow, A must-know orchestration tool for Data engineers.

Apache Airflow is an orchestration tool developed by Airbnb and later given to the open-source community. Today, it is the most beloved...

· 9 min read · Feb 8

179 2