Apache Kafka
A Not Kafka-esque Experience

Joshua Ra'anan
June 28, 2018

What is Kafka?

Apache Kafka is *a distributed streaming platform*

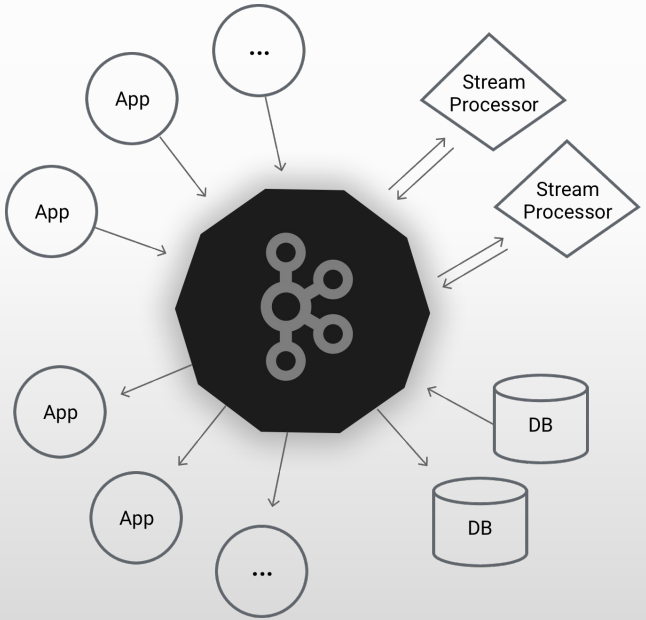


Image courtesy of “Apache Kafka.” *Apache Kafka*, kafka.apache.org/.

What does that mean?

A streaming platform has three key capabilities:

- **Publish and subscribe** to streams of records.
- **Store** streams of records.
- **Process** streams of records as they occur.

Kafka is generally used for two broad classes of applications:

- Real-time streaming data pipelines that get data between systems or applications
- Real-time streaming applications that transform or react to the streams of data

Why Kafka?

- Powerful
- Configurable
- *Reasonably* easy to set-up
- Provided in JAVA, but many client options
- Extensive documentation

Use Cases

Messaging - Kafka works well as a replacement for a more traditional message broker.

Website Activity Tracking - The original use case for Kafka was to be able to rebuild a user activity tracking pipeline as a set of real-time publish-subscribe feeds.

Metrics - Kafka is often used for operational monitoring data. This involves aggregating statistics from distributed applications to produce centralized feeds of operational data.

Log Aggregation - Kafka abstracts away the details of files and gives a cleaner abstraction of log or event data as a stream of messages.

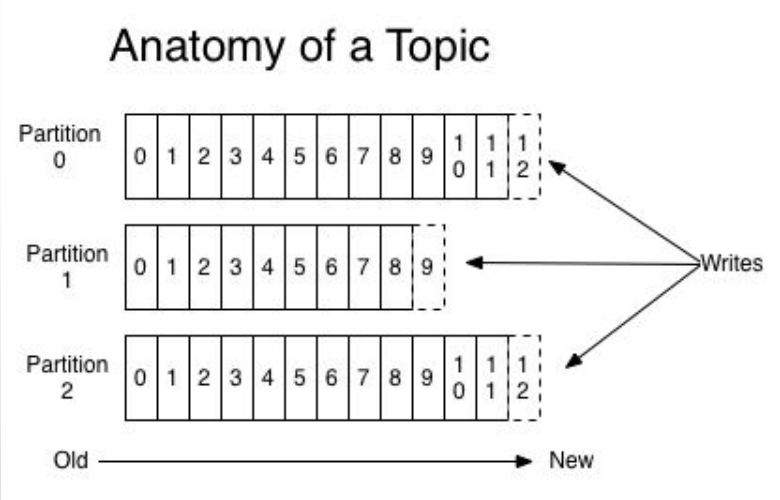
Stream Processing - Many users of Kafka process data in processing pipelines consisting of multiple stages, where raw input data is consumed from Kafka topics and then aggregated, enriched, or otherwise transformed into new topics for further consumption or follow-up processing.

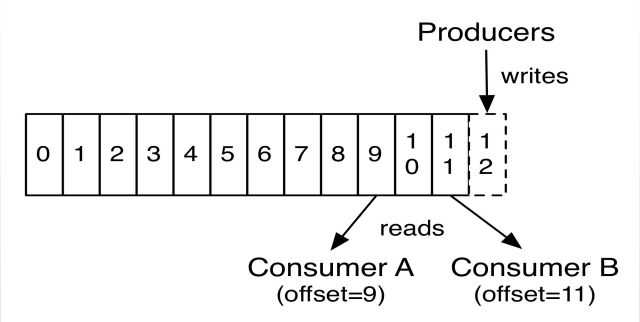
Event Sourcing - Event sourcing is a style of application design where state changes are logged as a time-ordered sequence of records. Kafka's support for very large stored log data makes it an excellent backend for an application built in this style.

Core Concepts

- Kafka is run as a cluster on one or more servers (aka brokers) that can span multiple datacenters.
- The Kafka cluster stores streams of *records* in categories called *topics*.
- Each record consists of a key, a value, and a timestamp.

Topics





Images courtesy of "Apache Kafka." *Apache Kafka*, kafka.apache.org/.

More Tools

Partitions - Kafka topics are divided into a number of partitions, which contains messages in an unchangeable sequence. Each message in a partition is assigned and identified by its uniqueoffset. This allows for multiple consumers to read from a topic in parallel.

Replication - In Kafka, replication is implemented at the partition level. The redundant unit of a topic partition is called a replica. Each partition usually has one or more replicas meaning that partitions contain messages that are replicated over a few Kafka brokers in the cluster.

ZooKeeper

Provides multiple features for distributed applications:

- Distributed management
- Self election
- Consensus building
- Synchronization

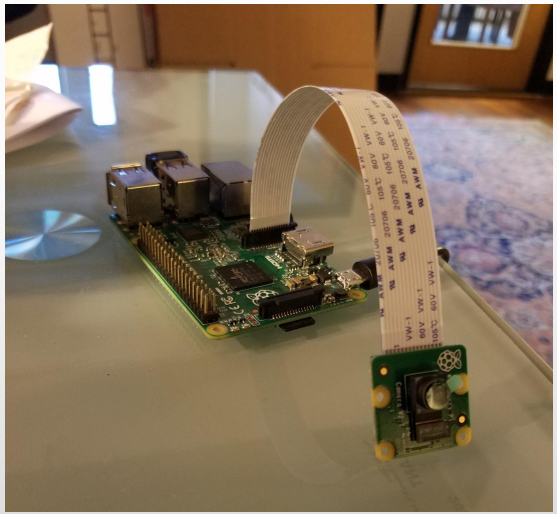
Quick Start

1. Start ZooKeeper
2. Start Kafaka Server on all Brokers
3. Create Topic
4. Create Producers (subscribed to Topic)
5. Create Consumers (subscribed to Topic)
6. Start communicating!

Four Core APIs

- The **Producer API** allows an application to publish a stream of records to one or more Kafka topics.
- The **Consumer API** allows an application to subscribe to one or more topics and process the stream of records produced to them.
- The **Streams API** allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
- The **Connector API** allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

DEMO TIME!



Demo Producer

```
from kafka import KafkaProducer

producer = KafkaProducer(bootstrap_servers=['10.42.0.1:9092', '10.42.0.21:9092'])

cmd = "raspistill -o /home/pi/image.jpg"

def image_emitter(image_file):
    # Take a picture and open it
    subprocess.call(cmd, shell=True)
    image = cv2.imread(image_file)
    # convert the image png
    jpeg = cv2.imencode('.png', image)[1].tostring()
    # Convert the image to bytes and send to kafka
    producer.send('topic', jpeg)
    time.sleep(5)

if __name__ == '__main__':
    while True:
        image_emitter('/home/pi/image.jpg')
```

Demo Consumer

```
from kafka import KafkaConsumer

consumer = KafkaConsumer(topic, group_id='view',
 bootstrap_servers=['10.42.0.1:9092'], consumer_timeout_ms=20000)
#Continuously listen to the connection and print messages as received
app = Flask(__name__)

@app.route('/')
def index():
    # return a multipart response
    return Response(kafkastream(),
                    mimetype='multipart/x-mixed-replace; boundary=frame')
def kafkastream():
    for msg in consumer:
        yield (b'--frame\r\n'
               b'Content-Type: image/png\r\n\r\n' + msg.value + b'\r\n\r\n')

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True)
```

Sources

- “Apache Kafka.” *Apache Kafka*, kafka.apache.org/.
- “Apache Kafka Message Streaming as a Service.” *CloudKarafka*, www.cloudkarafka.com/.