HELLO WORLD IN KAFKA USING PYTHON

Abraar Syed & Karthic Rao

Kafka Streaming Python

Hello World In Kafka Using Python

This is our first guest post here at Timber. If you're interested in writing for us, reach out on Twitter

This blog is for you if you've ever wondered:

1. What is Kakfa?
2. Why do I need a streaming/queueing/messaging system?
3. What are the benefits?
4. How does it fit with my current backend?

Just a disclaimer: we're a logging company here @ Timber. We'd love it if you tried out our product (it's seriously great!), but that's all we're going to advertise our product ... you guys came here to learn about Kafka and this guide won't disappoint.

What Is This Blog About?

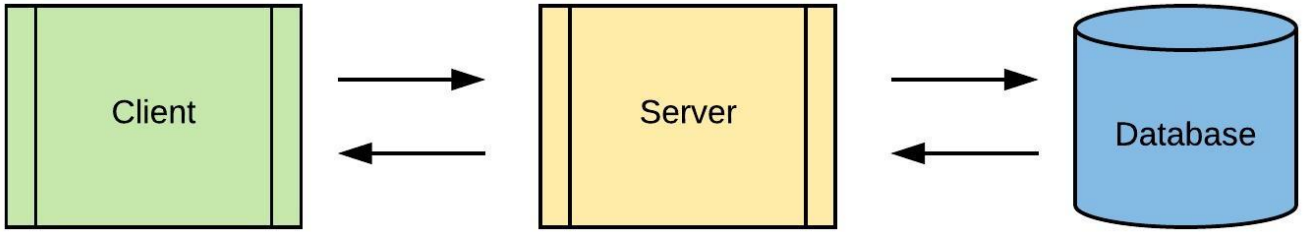
The title might have given it away, but we're going to show you what Kafka is, help you understand the need for a tool like Kafka, and then get started with it. We're believers that the best way to learn something is to do it, so get out your terminal and your favorite code editor and get ready.

What Is Kafka? Why Should One Use It?

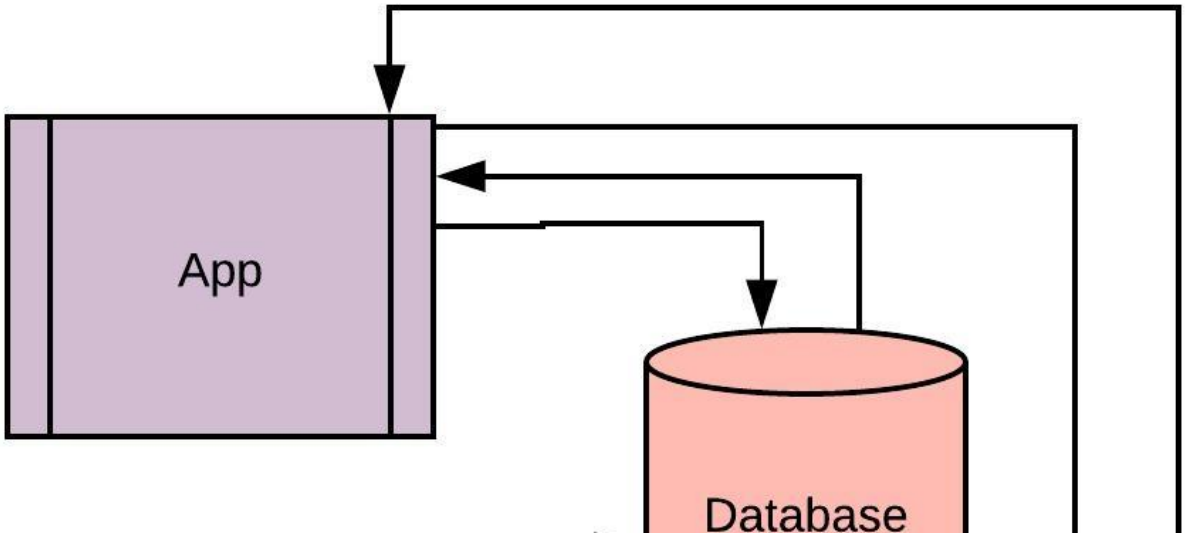
In short, Kafka is a distributed streaming platform.

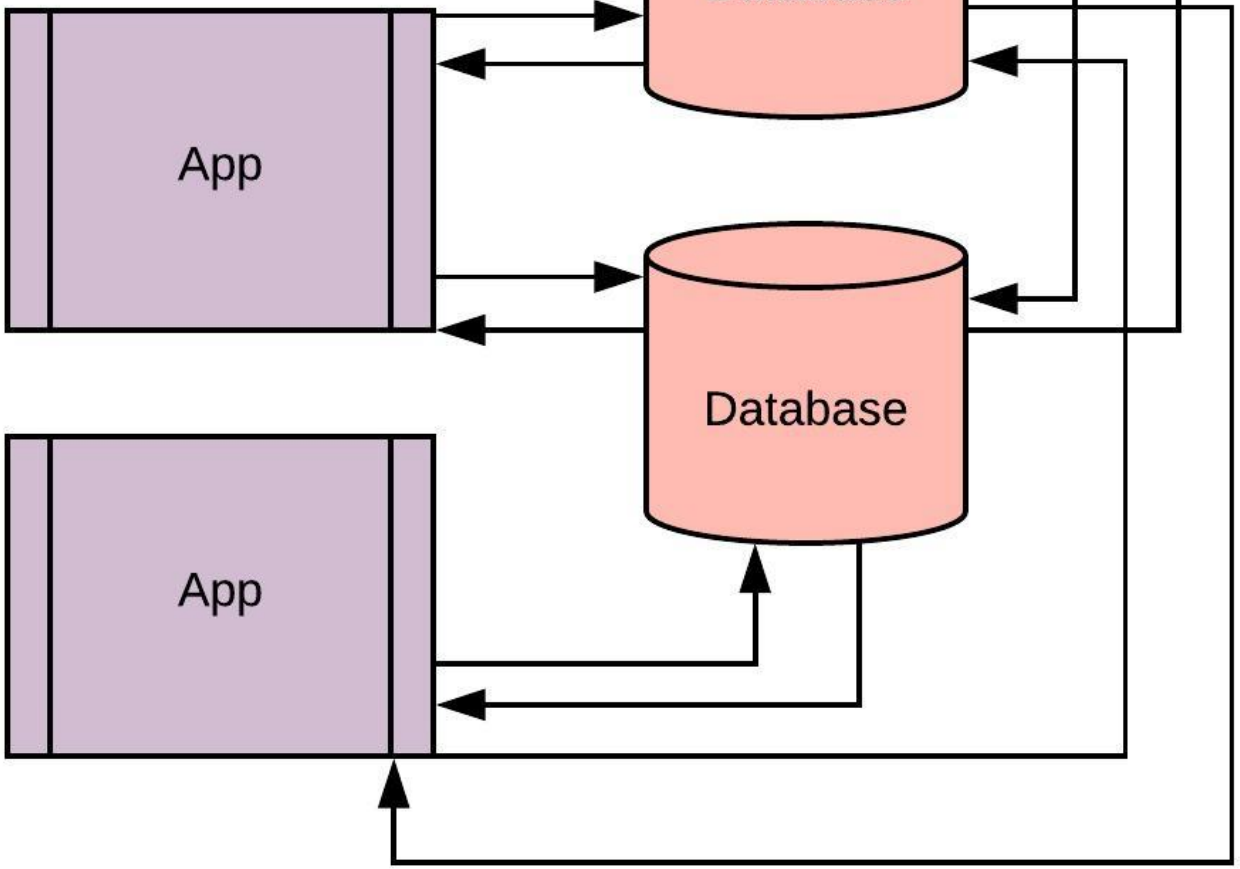
> *Oh wait! What does that even mean?*

Imagine that you have a simple web application which consists of an interactive UI, a web server, and a database.



You need to record all the events such as clicks, requests, impressions and searches that take place on your web application and store them for computation, reporting, and analysis, each of which is done by separate applications or services. A simple solution would be to store the data in your database and connect all other applications and services to your database.





This might look simple, but you're not finished. There are multiple challenges that can arise:

1. Events like clicks, requests, impressions and searches results in high-frequency interaction/requests or data flow to your web server and your primary database may not be equipped to scale seamlessly. This could introduce a high latency as more and more events pour into the server.
2. If you choose to store high-frequency data in database systems like SQL or MongoDB, it would be hard to introduce and reconstruct a new system or a

database on all of the historical data. You lose the flexibility to extend the capabilities of your system by introducing new technologies.

3. What if you have data processing systems in place to process these events to gain deeper insights? Since these systems wouldn't be capable of handling high-frequency reads and you wouldn't have access to the true source of data, it is practically impossible to experiment with various data processing or machine learning algorithms on all of the data.
4. Each application can follow its own data format, which means that you will need systems for data transformations when there is the exchange of data across these applications.

All these problems can be better addressed by bringing a streaming platform like Kafka into the picture. A streaming platform is a system that can perform the following:

1. Store a huge amount of data that can be persistent, checksummed and replicated for fault tolerance
2. Process continuous flow of data (data streams) in real time across systems
3. Allow applications to publish data or data streams independently and agnostic to the application/service consuming it

> *Interesting! How different is it from traditional databases?*

Although Kafka can store persistent data, it is NOT a database.

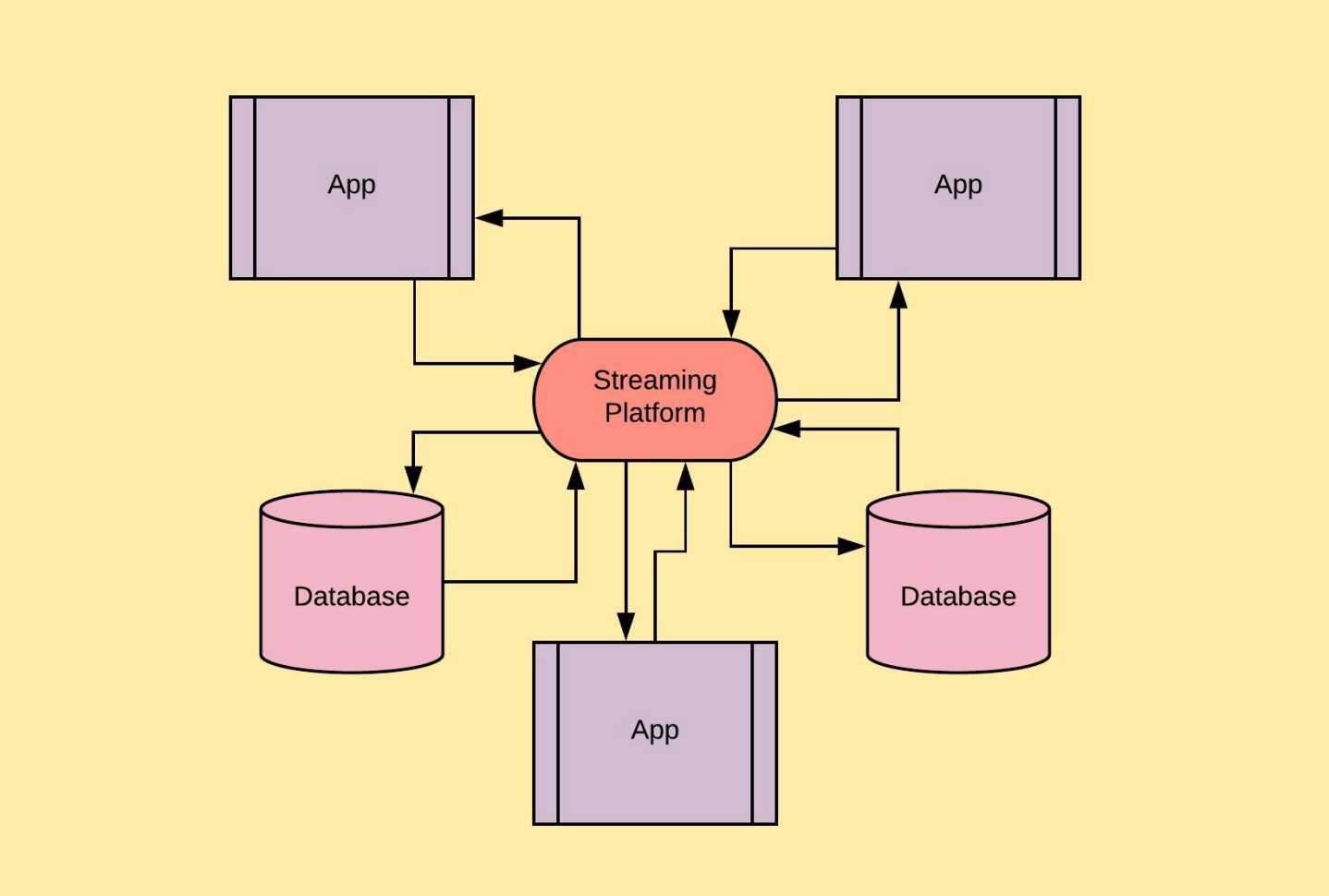
Kafka not only allows applications to push or pull a continuous flow of data, but it also deals with processing them to build and support real-time applications. This is different than performing CRUD operations on passive data or running queries on traditional databases.

> *That sounds convincing! But how does Kafka solve the above-mentioned challenges and why would one need a system like this?*

Kafka is a distributed platform and built for scale, which means it can handle sky-high frequency reads and writes & store huge volumes of data. It ensures that the data is always reliable. It also supports strong mechanisms for recovery from failures. Here are some of the key aspects of why one should be using Kafka:

1. Simplify The Backend Architecture

Look at how a complex architecture can be simplified and streamlined with the help of Kafka



2. Universal Pipeline Of Data

As you can see above, Kafka acts as a universal data pipeline across multiple applications and services. This gives us two advantages. The first one is data integration. We have all the data from different systems residing at a single place, making Kafka a true source of data. Any application can push data to this platform

which can later be pulled by another application. Secondly, Kafka makes it easy to exchange data between applications. Since we have all the data in one place, we

can standardize the data format that we will be using for the platform which can reduce our data transformations.

3. Connects To Existing Systems

Although Kafka allows you to have a standard data format, that does not mean the applications do not require data transformations. This allows us to reduce the overall number of data transformations in our architecture, but there may be cases when we still require transformations.

Consider connecting a legacy system to your architecture which does not know about Kafka: In such cases, Kafka offers a framework called Kafka Connect for us to connect to existing systems maintaining the universal data pipeline.

4. Process Data In Real-Time

A real-time application usually requires a continuous flow of data which can be processed immediately or within the current span of time with reduced latency. Kafka Streams make it possible to build, package and deploy applications without any need for separate stream processors or heavy and expensive infrastructure.

These features allow Kafka to become the true source of data for your architecture. This enables you to add new services and applications to your existing infrastructure

and allows you to rebuild existing databases or migrate from legacy systems with less effort.

Getting Started With Kafka

Installation

Installing Kafka is a fairly simple process. Just follow the given steps below:

1. Download the latest 1.1.0 release of Kafka
2. Un-tar the download using the following command: `tar -xzf kafka_2.11-1.1.0.tgz`
3. cd to Kafka directory to start working with it: `cd kafka_2.11-1.1.0`

Starting The Server

Kafka makes use of a tool called ZooKeeper which is a centralized service for a distributed environment like Kafka. It offers configuration service, synchronization service, and a naming registry for large distributed systems. You can read more about it here.

Thus, we need to first start the ZooKeeper server followed by the Kafka server. This can be achieved using the following commands:

```
# Start ZooKeeper Server
bin/zookeeper-server-start.sh config/zookeeper.properties

# Start Kafka Server
bin/kafka-server-start.sh config/server.properties
```

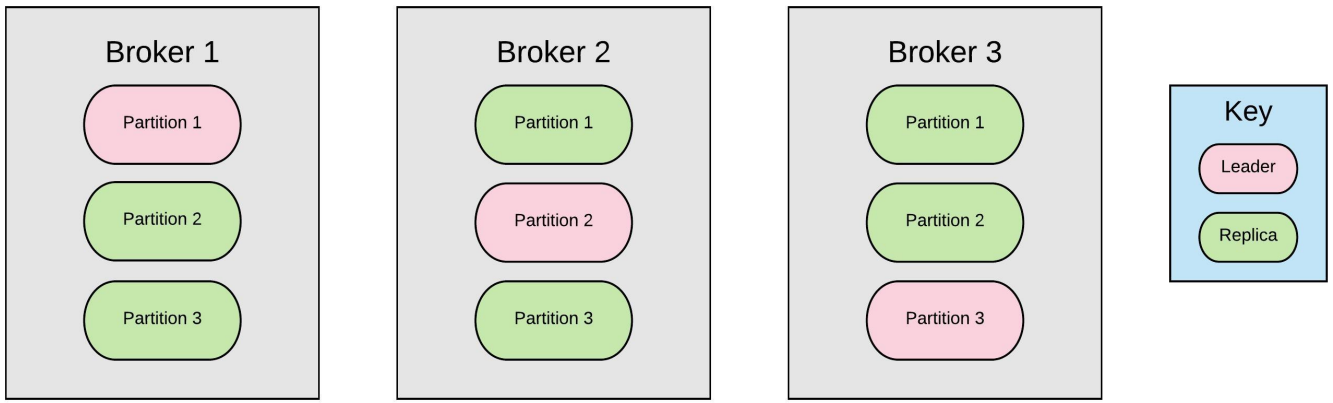
```
bin/kafka-server-start.sh config/server.properties
```

Understanding Kafka

Here is a quick introduction to some of the core concepts of Kafka architecture:

1. Kafka is run as a cluster on one or more servers
2. Kafka stores a stream of records in categories called `topics` . Each record consists of a key, value and a timestamp
3. Kafka works on the publish-subscribe pattern. It allows some of the applications to act as `producers` and publish the records to Kafka topics. Similarly, it allows some of the applications to act as `consumers` and subscribe to Kafka topics and process the records produced by it
4. Alongside, `Producer API` and `Consumer API` , Kafka also offers `Streams API` for an application to work as a stream processor and `Connector API` through which we can connect Kafka to other existing applications and data systems

Architecture



As you can see, Kafka topics are divided into partitions. These topics can be replicated across separate machines using brokers, which allows consumers to read from a topic in parallel.

Each of these brokers has partitions which are leaders and those that are replicas. This allows for an incredible level of fault tolerance through your system. When the system is functioning normally, all reads and writes to a topic go through the leader and the leader makes sure that all the other brokers are updated.

If a broker fails, the system can automatically reconfigure itself so a replica can take over as the new leader for that topic.

Creating Kafka Topics

Let us start by creating a `sample` Kafka topic with a single partition and replica. This can be done using the following command:

```
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitio
ns 1 --topic sample
```

Now, let us list down all of our Kafka topics to check if we have successfully created our `sample` topic. We can make use of the `list` command here:

```
bin/kafka-topics.sh --list --zookeeper localhost:2181
```

You can also make use of the `describe topics` command for more details on a particular Kafka topic:

```
bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic sample
```

Creating Producer And Consumer

Creating a producer and consumer can be a perfect `Hello, World!` example to learn Kafka but there are multiple ways through which we can achieve it. Some of them are listed below:

1. Command line client provided as default by Kafka
2. kafka-python
3. PyKafka
4. confluent-kafka

While these have their own set of advantages/disadvantages, we will be making use of `kafka-python` in this blog to achieve a simple producer and consumer setup in Kafka using python.

Kafka With Python

Before you get started with the following examples, ensure that you have `kafka-python` installed in your system:

```
pip install kafka-python
```

Kafka Consumer

Enter the following code snippet in a python shell:

```
from kafka import KafkaConsumer
consumer = KafkaConsumer('sample')
for message in consumer:
    print (message)
```

Kafka Producer

Now that we have a consumer listening to us, we should create a producer which generates messages that are published to Kafka and thereby consumed by our consumer created earlier:

```
from kafka import KafkaProducer
producer = KafkaProducer(bootstrap_servers='localhost:9092')
producer.send('sample', b'Hello, World!')
producer.send('sample', key=b'message-two', value=b'This is Kafka-Python')
```

You can now revisit the consumer shell to check if it has received the records sent from the producer through our Kafka setup.

Thus, a simple `Hello, World!` in Kafka using Python .

Final Remarks

Even though Kafka is a seriously powerful tool, there are some drawbacks, which is why we chose to go for a managed tool such as AWS Kinesis here at Timber. We've found that provisioning your own servers and digging into the nitty-gritty doesn't make as much sense when we're aiming for velocity. We're starting to reconsider that decision as we hit some of the limitations of Kinesis.