**Event-driven systems with Apache Kafka**

**Summary**

Objective:

To inform people about Kafka, and to persuade them into realizing the benefits of using Kafka in production.

Evidence:

Personal anecdote, concrete facts, general examples

**Roadmap**

* Introduce problem
  + Personal anecdote at Zenreach
* Kafka is the solution
  + Overview of Kafka
  + Why not traditional messaging queues?
* In-depth discussion of architecture and impact
* Summary

**Problem**

* Worked at Zenreach last term
* Zenreach provides a method for brick and mortar businesses like restaurants to keep track of people who physically enter their business through wifi
* When a customer visits, they are detected by the wifi and a visit stat is produced
* Because there were 8000+ business using our platform, and each business had on average thousands of registered customers, there could be hundreds of thousands of visit stats being generated at one time during peak business hours
* We need a technology that can act as a giant buffer that can consume a high-volume of data and produce them in a streamlined manner for data processing
* Many services we have care about the visit stats data; one service might need and perform business logic, one service might need the data for training machine learning models, and one service could simply put each visit stat in a database.
* There could be some problem with our system, that requires us to re-process a sequence of visit stats; the visit stats need to be processed in the order in which they were registered; if a guy visits on the 3rd of the month and the 5th, we process the 3rd visit first; we need to be able to “go back in time”
* Need something that scales horizontally, given that our network of businesses and their customers will only expand with time

**Messaging Queues**

* Could we use a traditional messaging queue like RabbitMQ?
* Queue as in data structure, and message as in a packet of data; sounds like something we could use in this situation right?
* Use message queue when some system gets a lot of requests per second, and all requests must be processed by a task that is time consuming
* Ordering is not guaranteed; if we get the visit stat of a customer visiting at 5:00 before we get a different stat of him visiting at 3:00, then that doesn’t make sense.
* Once a message is delivered, it is removed by the queue; can’t replay messages
* Scalability; can we always scale messaging systems horizontally? Not always.

**Enter Kafka**

* Apache Kafka was developed by engineers at LinkedIn using Scala and Java, and became an open sourced Apache project in 2011
* Apache Kafka is similar to a publish-subscribed based messaging system that maintains logs of events in topics, which are dedicated storage spaces.
* According to the official documentation, Kafka was used to “[build] real-time streaming applications that transform or react to the streams of data.”
* Basically a distributed ledger of messages in sequential order
* Distributed meaning that Kafka can run on a group of computers in a shared network, that distribute computing workload; thus, if you need more computing power, you just need to add more computers to your existing cluster.
* Because of its distributed nature and implementation, Kafka can achieve linear scaling and near network speeds of event processing.

**Internal Architecture**

* A producer is an application that forwards messages to the Kafka
* Send data to Kafka as an array of bytes (called a message) through an API
* Messages are received by a consumer; a consumer decides what kind of data it wants from the Kafka and requests for that kind of data
* What are we sending the messages to? We are sending them to Kafka brokers, basically a server that acts as a message broker
* Since Kafka is distributed, there are multiple brokers arranged into clusters
* Broker: “An individual who arranges transactions between a buyer and seller”
* A topic is a unique name for a Kafka stream
* In our Zenreach example, there might be multiple producers producing messages to a Visit-Stats topic we have created
* A new message is pushed automatically to the end of a topic and assigned a unique offset id, and applications that are interested in these messages create Consumers that have their own pointers to a specific position in the log; they can read or *consume* messages at their own pace.
* One or more applications can use a Producer API to publish messages to various topics, and one or more applications can use a Consumer API to consume messages from various topics.

Apache Zookeeper is used to provide some coordination services for a distributed system

**Impact**

* Some companies that utilize Kafka in production include: LinkedIn, Twitter, Netflix, Uber, Goldman Sachs, PayPal, Airbnb, and Oracle

10 min so far