Kafka Fundamentals

Your Instructors

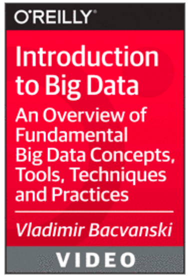
Petter Graff

- Architect, consultant, co-founder of SciSpike
- Main architect of Yaktor, a scalable event-driven, agent based rapid development platform
- Lives in Austin
- O'Reilly author:
  Design Patterns in Java

O'REILLY
Design Patterns in Java
Understand and Apply Analysis, Architecture, Design, and Language Patterns
Petter Graff
VIDEO

Dr. Vladimir Bacvanski

- Technologist, consultant, mentor, co-founder of SciSpike
- Passionate about software and data
- Lives in Silicon Valley, advises leading organizations worldwide
- O'Reilly author:
  Introduction to Big Data


O'REILLY
Introduction to Big Data
An Overview of Fundamental Big Data Concepts, Tools, Techniques and Practices
Vladimir Bacvanski
VIDEO

SciSpike

- Custom software development: focus on extremely fast development, web scale, cloud, Big Data applications
- Consultant, and mentor to many large organizations worldwide
- Node.js, Scala, Java…
- Big Data and NoSQL systems
- Custom training to leading firms worldwide

- http://www.scispike.com
- Custom software development
- Consulting, mentoring, training
- We are independent

Outline

- Introduction to Kafka
- Kafka core concepts
- Producing data to Kafka using the Producer API
- Consuming data from Kafka using the Consumer API
- Kafka Administration and Integration

What is Kafka?

- *"Kafka is a distributed, partitioned, replicated commit log service"*

- Publish-subscribe messaging system
- Fault tolerant
- Scalable
- Durable
- Apache project
- Originally developed by LinkedIn

Notable Users

- Cisco Systems
- Netflix
- PayPal
- Spotify
- Uber
- HubSpot
- Betfair
- Shopify

LinkedIn and Kafka's Origins

- Kafka created to solve a data pipeline problem at LinkedIn
- Legacy systems involved custom collectors for gathering system and application metrics based on polling
- In parallel, a separate system was created for collecting user activity tracking information
- *System monitoring* and *user activity tracking* systems could not use the same back-end service but the data collected by both shared many traits

- Correlation of data between the two systems was **highly desirable!**

Design Goals at LinkedIn

- Decouple the producers and consumers by using a push-pull model
- Provide persistence for messages to allow multiple consumers
- Optimize for high throughput of messages
- Allow for horizontal scaling as the data streams grow

Kafka is Born!

- Publish/subscribe system with an interface of a typical messaging system but a storage layer like a log aggregation system

- As of August 2015, LinkedIn produces over one trillion messages and a petabyte of data consumed daily!

- Kafka was released as open source in late 2010 and became an Apache project in 2011

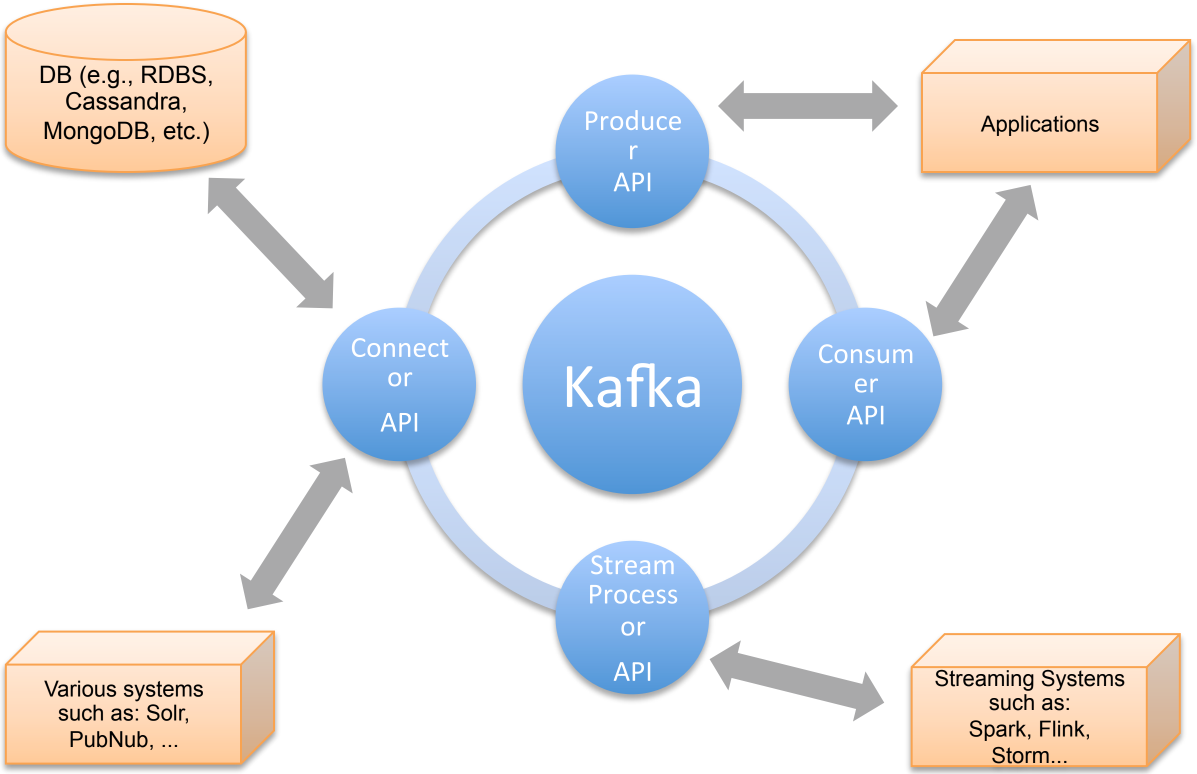
The Name

I thought that since Kafka was a system optimized for writing using a writer's name would make sense. I had taken a lot of lit classes in college and liked Franz Kafka. Plus the name sounded cool for an open source project.

So basically there is not much of a relationship.

Jay Kreps, Lead Developer at LinkedIn

Kafka API's


DB (e.g., RDBS, Cassandra, MongoDB, etc.)
Produce r API
Applications
Connect or API
Kafka
Consum er API
Stream Process or API
Various systems such as: Solr, PubNub, ...
Streaming Systems such as: Spark, Flink, Storm...

Use Cases

- Messaging
- Website Activity Tracking
- Metrics
- Log Aggregation
- Stream Processing
- Event Sourcing
- Commit Log

SciSpike

Website Activity Tracking

- The original use case for Kafka was to be able to rebuild user activity as a set of real-time publish-subscribe feeds

- Site activity such as page views, searches, or other user actions are published to central topics with one topic per activity type

- These feeds are then available for subscription for processing, monitoring, and loading into offline processing/reporting

- Activity tracking can be **very high volume** as many messages are generated for each user page view

Messaging

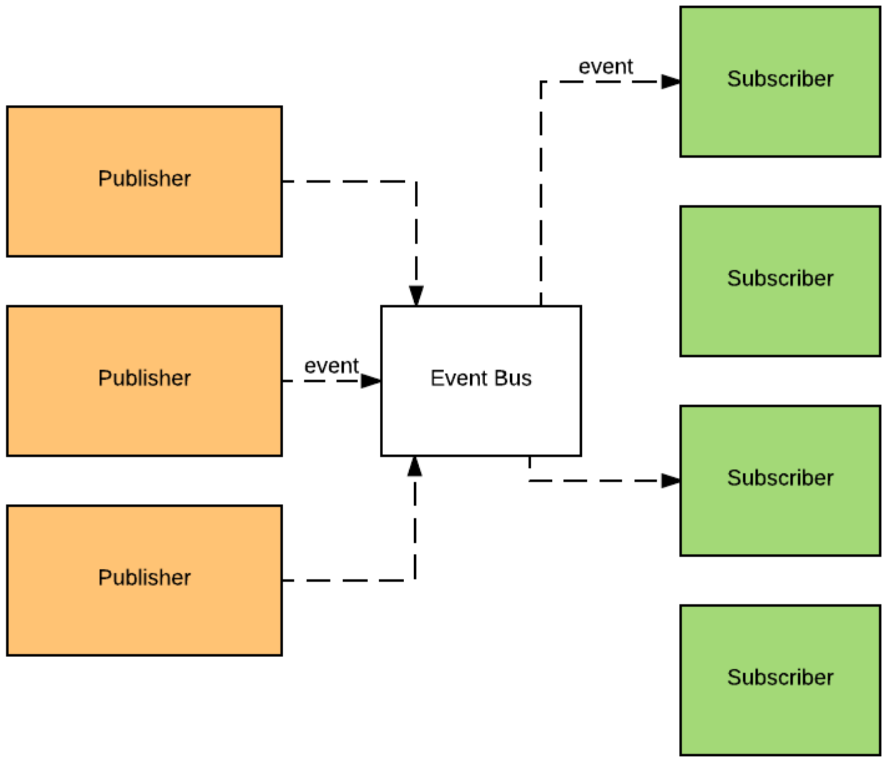
- Kafka can be a replacement for a more traditional message broker such as ActiveMQ or RabbitMQ
- Message broker scan be used to decouple processing from data producers, buffer unprocessed messages, etc.
- Kafka has replication, built-in partitioning, and fault-tolerance making it a good solution for large scale applications

Publish-Subscribe Pattern

- Sender (publisher) wants to send a piece of data (message)
- The message is not specifically directed to a receiver (subscriber)
- The sender classifies the message and the receiver subscribes to receive certain classes of messages
- Usually there is a central broker where messages are published

SciSpike

Publish-Subscribe Illustrated



Log Aggregation

- Log aggregation collects physical log files off servers and put them in a central place such as a file server of HDFS for processing
- Kafka abstracts away the details of files and gives a cleaner abstraction of log or event data as a stream of messages

Why Kafka?

- Multiple Producers
  - Clients can use many topics or the same topic
- Multiple Consumers
  - Multiple consumers can read any single stream of messages without interfering with each other
  - Some pub/sub systems only allow one consumer to read a message
- Disk-based Retention
  - Messages are committed to disk and stored with configurable retention rules

Why Kafka?

- Scalable
  - Flexible scalability was a design goal for Kafka from the beginning
  - Expansions can be performed while the cluster is online
  - Clusters can be configured with a replication factor

- High Performance
  - Producers, consumers, and brokers can all be scaled to handle very large message streams
  - Sub-second message latency to consumers

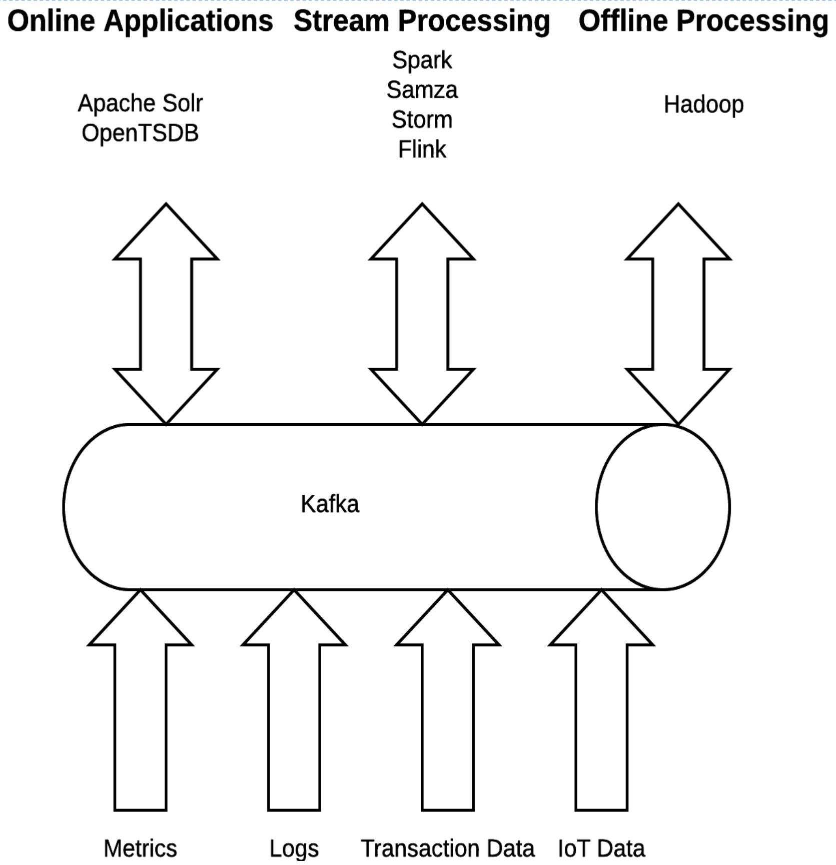
SciSpike

What Kafka Does NOT Do

- Not an end-user solution
  - You must write code for producers and consumers

- Not a replacement for all features of a messaging systems
  - There are no individual queues controlling the consumption
  - No point-to-point messaging

Kafka in the Big Data World



How Big is Big?

- Per day:
  - 800 billion messages
  - 175 TB of data
  - 650 TB of consumed data
- Per second:
  - 13 million messages
  - 2.75 GB of data
- Configuration
  - 1100 Kafka brokers
  - 60 clusters

$\Bigg\}$ Numbers from LinkedIn

Summary

- Apache Kafka is an open source, distributed, partitioned, and replicated commit-log based publish-subscribe messaging system
  - Scalable
  - High Performance
  - Multiple Consumers
  - Multiple Producers
  - Disk-based Retention

SciSpike

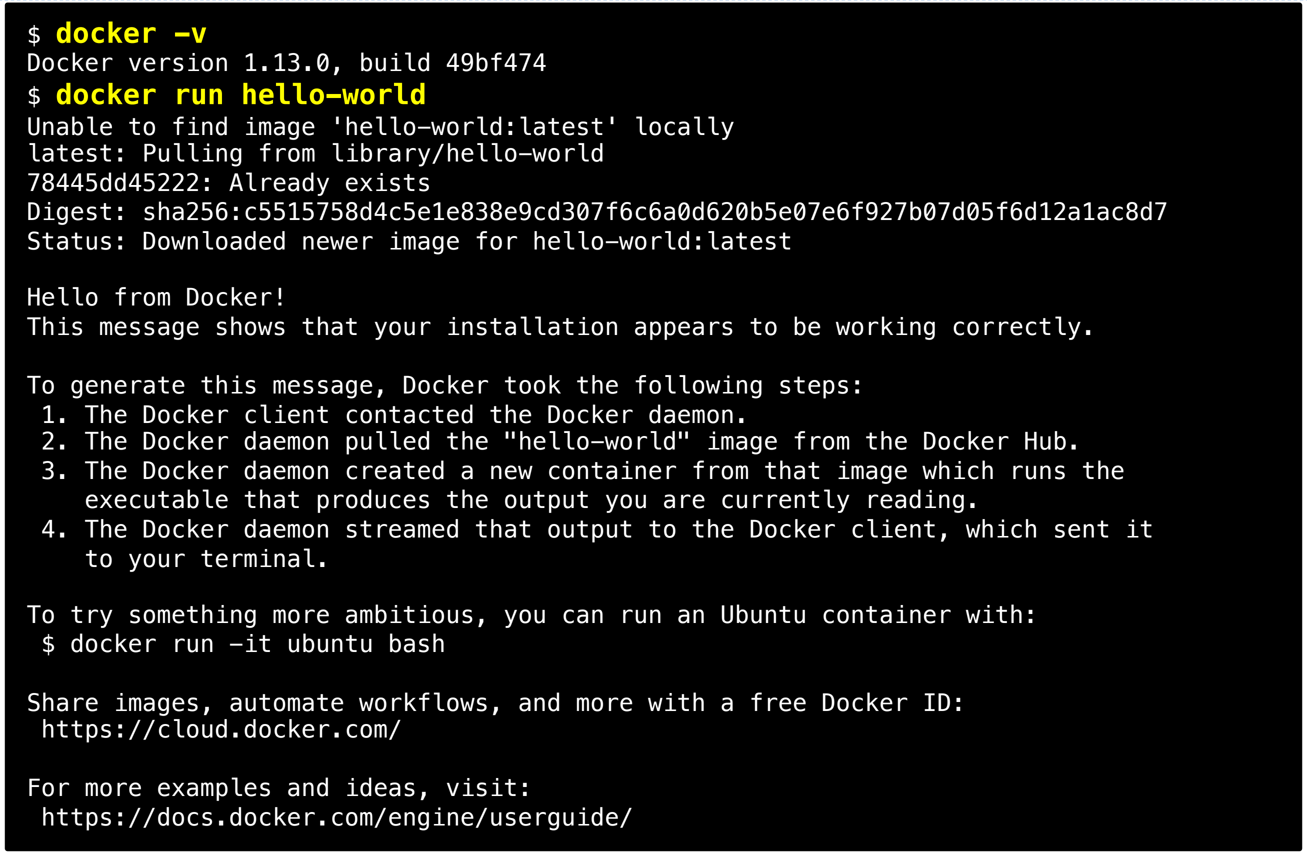
Start of Exercise

- Do you have Docker installed?

- We will be using Docker to run Kafka
- If you have not done so already, please install Docker
  - https://docs.docker.com/engine/installation/

- Check if Docker works by running

Check installation of Docker



```
$ docker -v
Docker version 1.13.0, build 49bf474
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
78445dd45222: Already exists
Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

docker-compose.yml

```
version: '2'
services:
    zookeeper:
        image: zookeeper:3.4.9
    kafka:
        image: wurstmeister/kafka:0.10.1.1
        environment:
            HOSTNAME_COMMAND: "echo $HOSTNAME"
            KAFKA_ADVERTISED_PORT: 9092
            KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
        depends_on:
            - zookeeper
```

Run Zookeeper

Run Kafka

SciSpike

Essential Kafka CLI Commands (used in exercise)

- Kafka can be configured via the command
- Key commands
  - kafka-topics.sh
    - Allows us to manipulate and view topics
  - kafka-console-producer.sh
    - Allows us to produce data from using stdin
  - kafka-console-consumer.sh
    - Allows us to consume messages from the console
- We're running all tools in docker, so we have to 'reach into' the docker instance, hence our commands look like this:
  - docker-compose exec kafka /opt/kafka_2.11-0.10.1.1/bin/CMD
  - You may want to alias these commands to reduce typing or simply run a bash shell inside the docker image
    - docker-compose exec kafka /bin/bash

SciSpike

Lab

- In this lab we'll simply ensure that we can run Kafka
- We'll run Kafka using Docker
  - Easy configuration
  - Runs Kafka and Zookeeper
  - Flexible setup that allows us to setup a cluster of machines for later exercises
- The lab simply:

1. Starts the docker images using docker-compose
2. Runs a simple producer
3. Runs a simple consumer
4. Allows you to type messages in the producer and see them consumed by the consumer