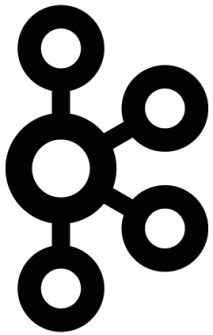
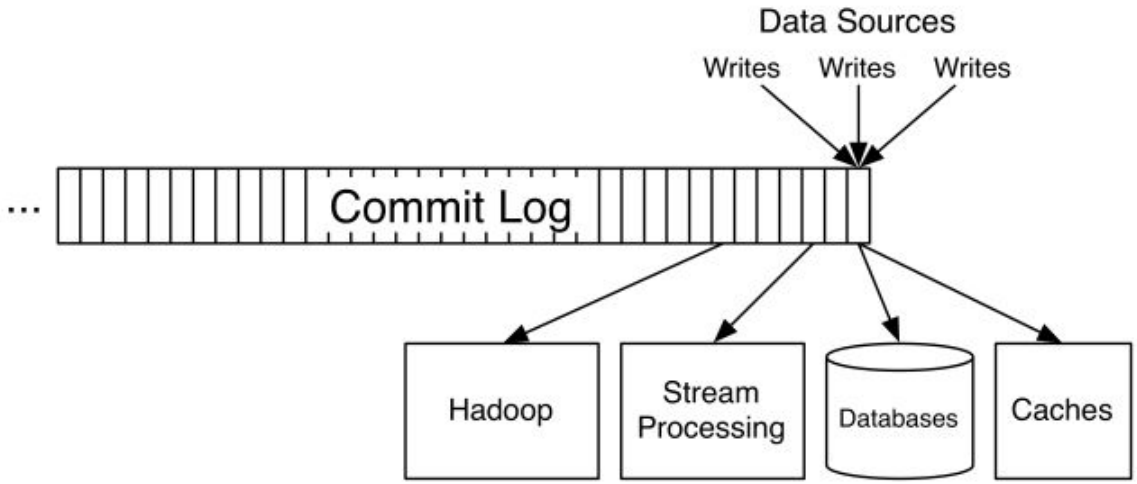
Kafka



distributed streaming platform

Apache Kafka is publish-subscribe messaging rethought as a distributed commit log.



Kafka characteristics

- Fast
  - A single Kafka broker can handle hundreds of megabytes of reads and writes per second from thousands of clients.
- Scalable
  - It can be elastically and transparently expanded without downtime
- Durable
  - Messages are persisted on disk and replicated within the cluster to prevent data loss
- Distributed by Design
- Replicated

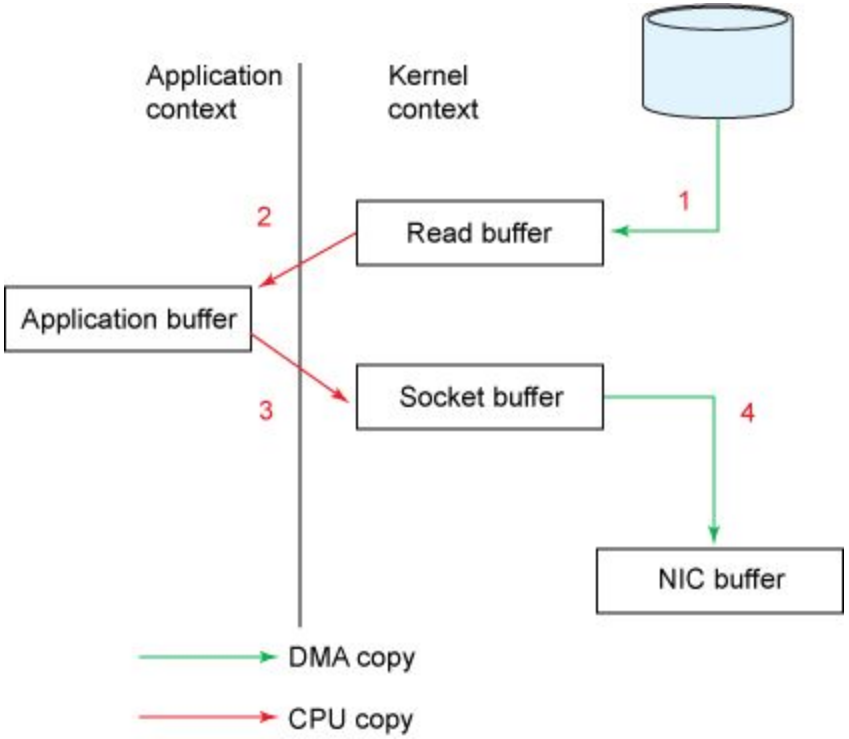
Why Kafka?

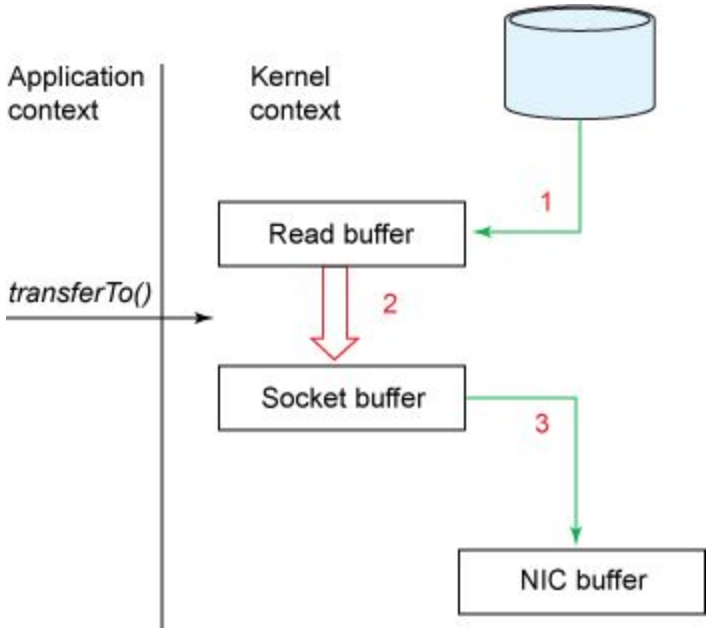
- Stands out for highly distributed messages.
- Consumer messages are ordered per partition.
- Mechanical sympathy - linear read and writes, pagecache, batching, batch compression
- Fast reads (efficient use of page cache) and fast writes (efficient transfer from page cache to network sockets - zero copy optimization --sendfile system call)

Kafka vs Rabbit

Kafka tuned to high throughput 100,000 mes/sec vs Rabbit 20,000 mes/sec	Rabbit: Tuned to complex route, supports point-to-point and pub/sub
Message state managed by consumer (Kafka), by Broker (RabbitMQ), producers ack configurable	Broker distributes messages to all available consumers, message can be redelivered if consumer fails
Ordering by partition (Kafka)	Better support to slower consumers
Messages are removed : retention policy (Kafka), on consumption (Rabbit)	Supports a lot of protocols
Kafka: one consumer (group) per partition	Web panels, monitoring tools
Kafka: allows batch processing and rereads	Clustering: Federated Exchanges/Queues

Zero Copy


Application context
Kernel context
1
2
Read buffer
Application buffer
3
Socket buffer
4
NIC buffer
DMA copy
CPU copy


Application context
Kernel context
1
Read buffer
transferTo()
2
Socket buffer
3
NIC buffer

Messaging terminology

- Data is called *message*
- *Producers* publish messages.
- Messages are stored in *topics*.
- Each Kafka server in a cluster is called *Broker*.
- Topics are *partitioned* and *replicated* into Brokers.
- *Consumers* consume messages from brokers..

Broker

Kafka brokers are stateless. This means that the consumer has to maintain how much it has consumed (in other queues brokers had to maintain the delivery state of every message)

This design has a big benefit, as consumer can deliberately rewind back to an old offset and re-consume data. This violates the common contract of a queue, but proves to be an essential feature for many consumers.

Each Kafka broker is coordinating with other Kafka brokers using ZooKeeper

Topic

Remove messages based on :

- *number of messages:* log.flush.interval
- *time:* log.default.flush.interval.ms,topic.flush.intervals.ms
- *size:* log.retention.size

Partition

- Ordered, immutable sequence of messages
- Each message is assigned unique offset
- Serves: Horizontal scaling, Parallel consumer reads (with consumption by partition based order)
- Partitioner may be default (based on hash of key) or custom (by requirement. e.g: user-id - if we need more processing based on user-id)

Producers - push

```
producer = new KafkaProducer<>(props);

producer.send(new ProducerRecord<>(topic, msg), callback);
```

- batching
- compression
- sync(Ack), *async* (auto batch - say 60k or 10ms)
- sequential writes - guaranteed order per partition
- load balancing by clients based on Partitioner

Consumers - pull

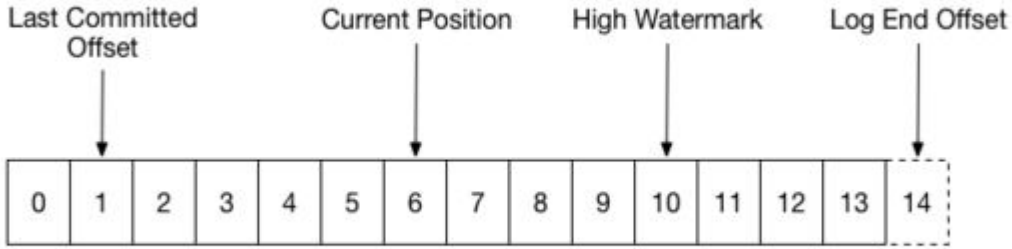


Figure 2: The Consumer's Position in the Log

```
consumer = new KafkaConsumer<>(props);
consumer.subscribe(topics);
ConsumerRecords<String, JsonNode> records = consumer.poll(timeout)
```

- Queue of consumers (consumer group)
- Position based on offset, controlled by consumer and persisted at intervals into topic - __consumer_offsets
- can rewind offset
- Guaranteed order per partition
- More partitions enables better parallel reads

Zookeeper: coordinator between the broker and consumer

- **Sequential Consistency** - Updates from a client will be applied in the order that they were sent.
- **Atomicity** - Updates either succeed or fail. No partial results.
- **Single System Image** - A client will see the same view of the service regardless of the server that it connects to.
- **Reliability** - Once an update has been applied, it will persist from that time forward until a client overwrites the update.
- **Timeliness** - The clients view of the system is guaranteed to be up-to-date within a certain time bound.

Distribution & Replication

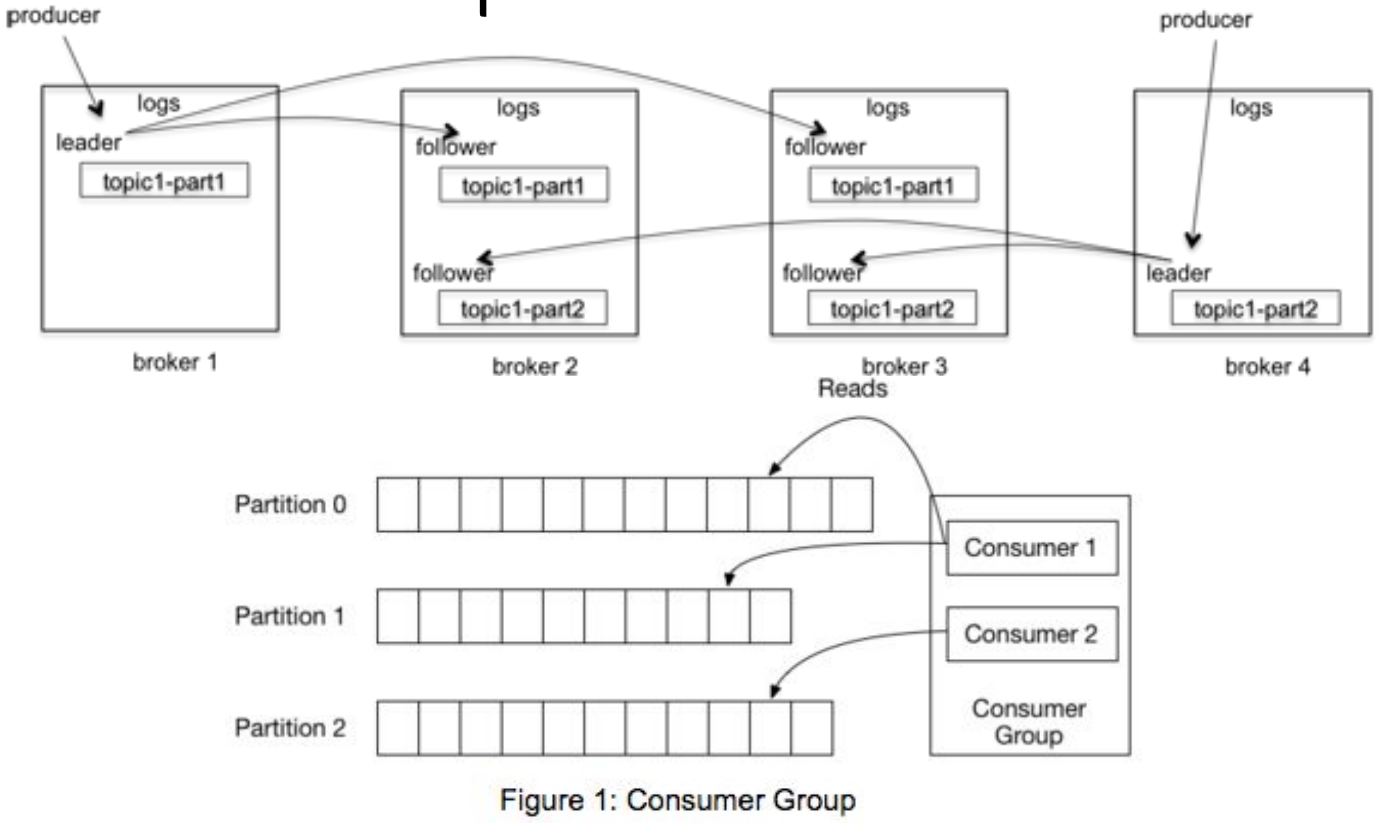


Figure 1: Consumer Group

Usecases

- Messaging (comparable with ActiveMQ, RabbitMQ)
- Website activity tracking
- Operational monitoring
- Log Aggregation
- Stream processing (along with Storm/Samza)
- Event sourcing
- Commit log (comparable with BookKeeper)

KEEP CALM IT IS DEMO TIME

Zookeeper

Download current stable version of Zookeer 3.4.10
wget https://apache.mivzakim.net/zookeeper/current/zookeeper-3.4.10.tar.gz

Extract
tar -zxf zookeeper-3.4.10.tar.gz
cp zookeeper-3.4.10/conf/zoo_sample.cfg zookeeper-3.4.10/conf/zoo.cfg
change dataDir value in zookeeper-3.4.10/conf/zoo.cfg

Start
sudo zookeeper-3.4.10/bin/zkServer.sh start

Or
sudo zookeeper-3.4.10/bin/zkServer.sh start zookeeper-3.4.10/conf/zoo.cfg
verify zookeeper installation
tailf zookeeper.out
Or
zookeeper-3.4.10/bin/zkCli.sh

Kafka cluster

Download current stable version of Kafka 0.11.0.1
wget http://apache.mivzakim.net/kafka/0.11.0.1/kafka_2.11-0.11.0.1.tgz
extract
tar -zxf kafka_2.11-0.11.0.1.tgz
Start Kafka
./kafka_2.11-0.11.0.1/bin/kafka-server-start.sh ./kafka_2.11-0.11.0.1/config/server.properties

Change server.properties
broker.id=1
port=9092
log.dir=/tmp/kafka-logs-1

CLI

Topic

./kafka_2.11-0.11.0.1/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 10 --topic test
cd /home/lev/dev/kafka-workshop/kafka_2.11-0.11.0.1/bin
./kafka-topics.sh --list --zookeeper=localhost:2181

Consumer / Producer

bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test

Zookeeper

sudo zookeeper-3.4.10/bin/zkCli.sh
$ ls /brokers/ids # Gives the list of active brokers
$ ls /brokers/topics #Gives the list of topics
$ get /brokers/ids/0 #Gives more detailed information of the broker id '0'

Kafka manager

https://github.com/yahoo/kafka-manager/releases
Download sources
Install sbt
#./sbt clean dist
Unzip .../kafka-manager-1.3.3.14_zip/target/universal/kafka-manager-1.3.3.14.zip
Chmod +x bin/kafka-monitor
bin/kafka-monitor

Open http://localhost:9000
Add new cluster

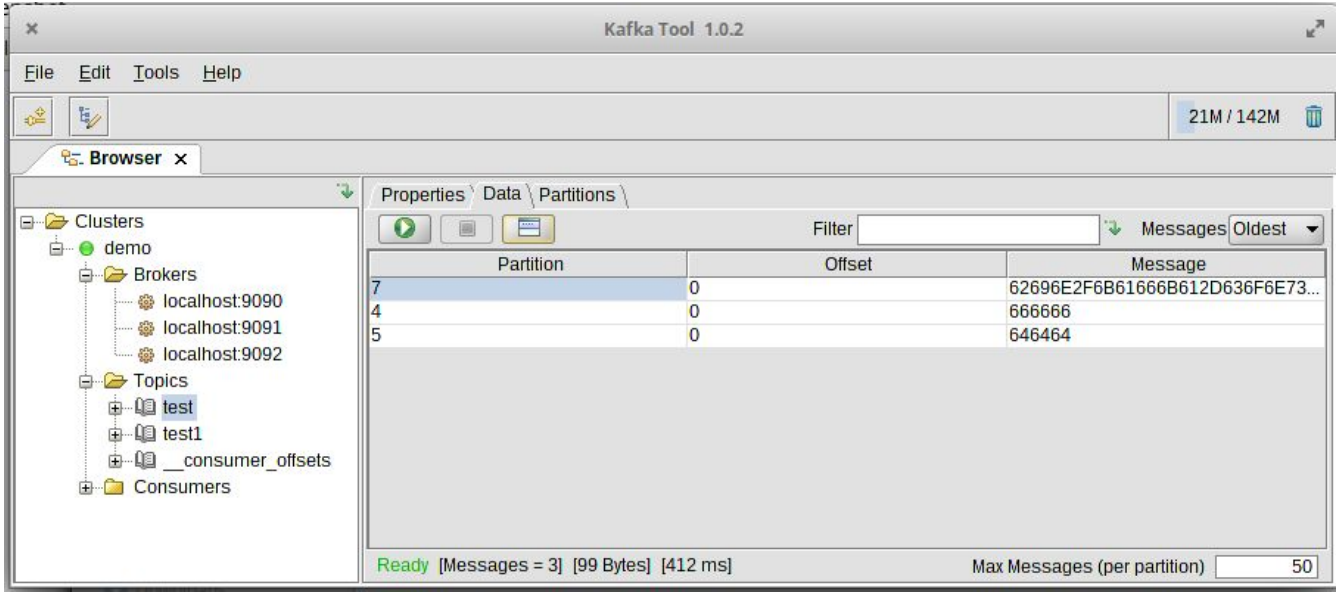
Application.conf edit kafka-manager.zkhosts="localhost:2181"

Kafka tool

http://www.kafkatool.com/download/kafkatool.sh

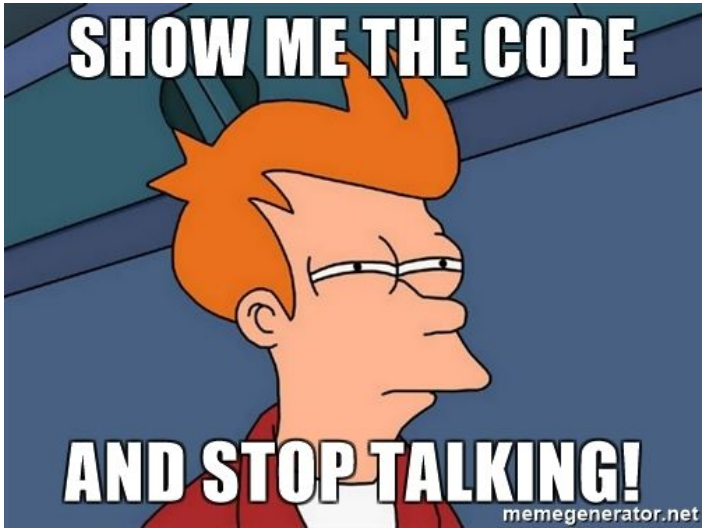
sh ./kafkatool.sh

kafkatool/kafkatool

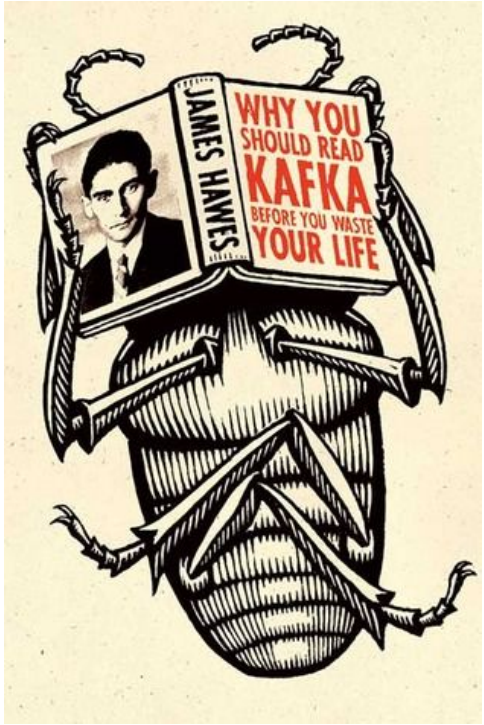


APIs

1. The **Producer API** allows an application to publish a stream of records to one or more Kafka topics.
2. The **Consumer API** allows an application to subscribe to one or more topics and process the stream of records produced to them.
3. The **Connector API** (from version 9) 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.
4. The **Streams API** (from from version 10) 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.



git clone https://github.com/levplotkin/kafka-introduction-workshop.git


JAMES HAWES
WHY YOU SHOULD READ KAFKA BEFORE YOU WASTE YOUR LIFE

Fin