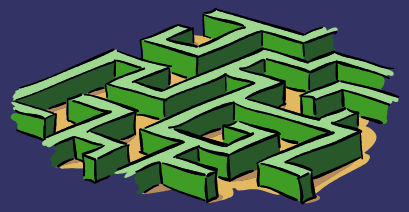
Apache Kafka

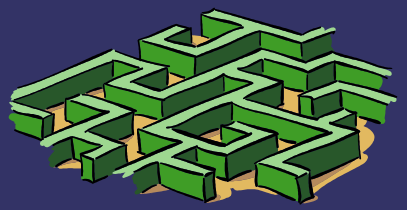
Obligatory bold claim:
Kafka Rocks! After this
presentation you will love it

Edward Capriolo @ m6d.com



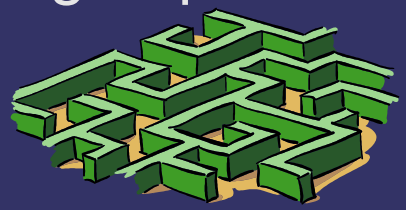
Facts and history

- Written in Java/scala
- Developed by Linked-in
- Recently accepted as Apache project



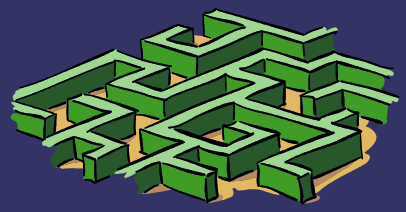
Overall design

- Distributed messages queuing system
- Designed for throughput not complicated acknowledgment stuff like JMS. Again... Kafka != JMS
- Simple producer consumer API
- Always persists messages to disk
  - Linear writes and reading
  - Don't worry still fast (sendfile support)
- Messages are retained for specified time period (then pruned)
- Clients state stores position in log files
  - Easy to replay messages
  - Easy to deliver messages to multiple consumer groups

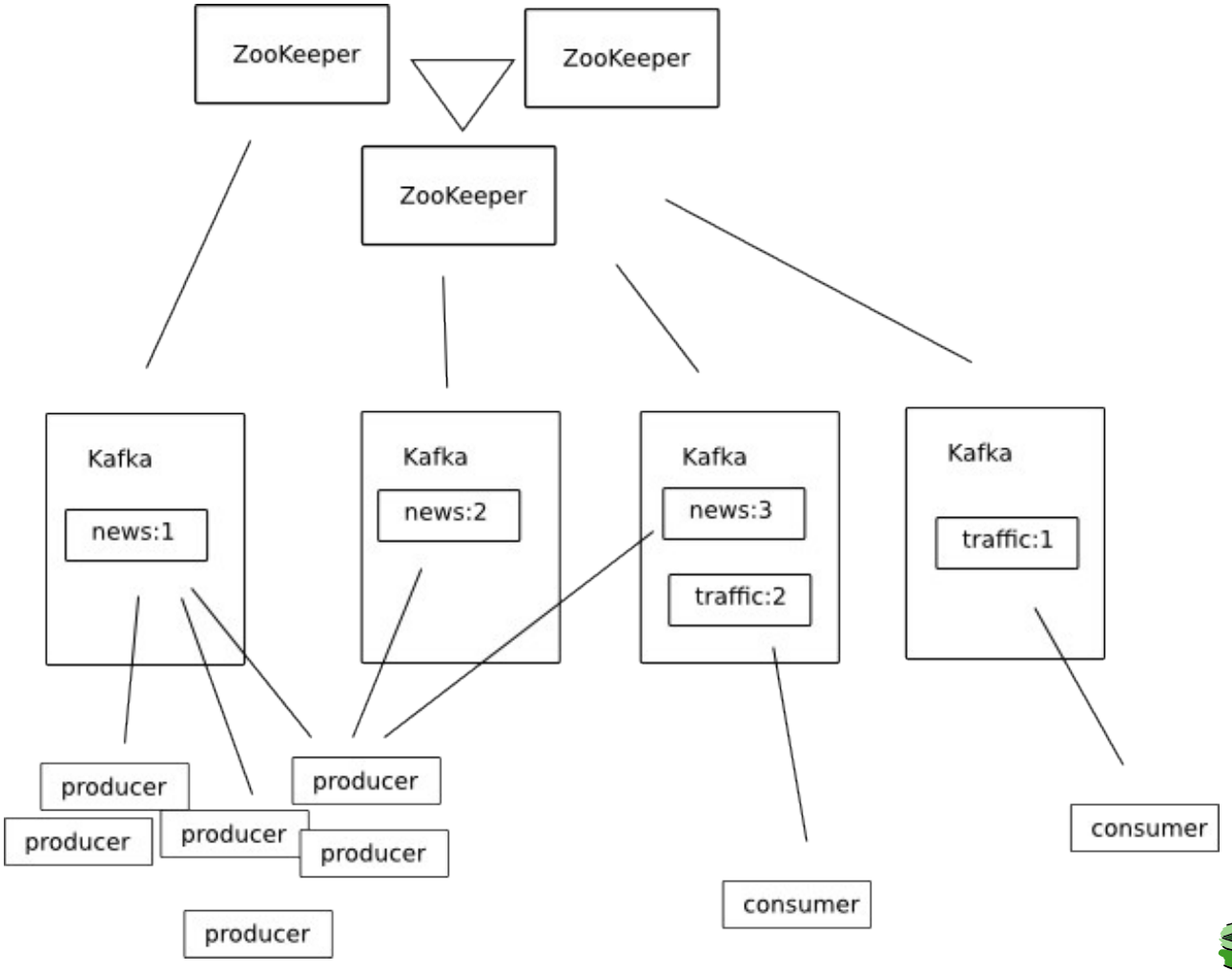


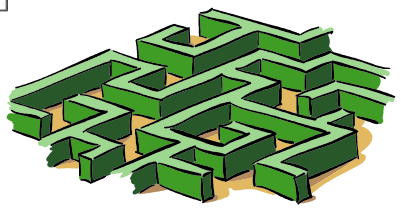
Key Concepts

- Message : Datum to send
- Topic : named place to send messages
- Broker: Kafka Server
- Partition: A logical division of a topic
- Producer: An API to send messages
- Consumer: An API to receive messages
- Consumer Group: a group of clients that will receive only one copy of a message from a topic (more later)

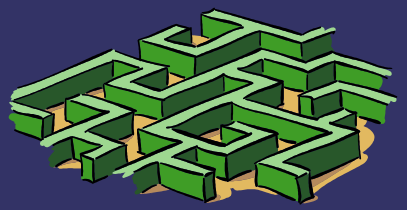


Design


ZooKeeper
ZooKeeper
ZooKeeper
Kafka
news:1
Kafka
news:2
Kafka
news:3
traffic:2
Kafka
traffic:1
producer
producer
producer
producer
producer
producer
consumer
consumer

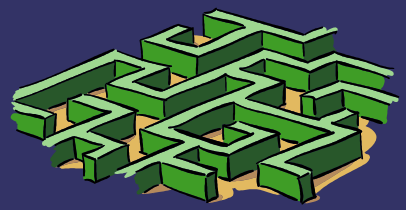


Demo 1 Producer Consumer Shell

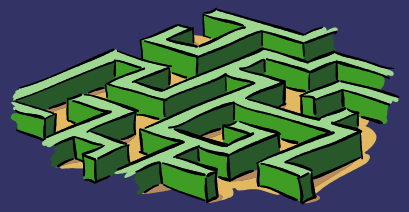


Startup the kafka/Broker components

- cd /home/edward/kafka/kafka/
- bin/zookeeper-server-start.sh config/zookeeper.properties &
- bin/kafka-server-start.sh config/server.properties
- bin/kafka-producer-shell.sh --server kafka://localhost:9092 --topic topic1
- bin/kafka-consumer-shell.sh --partitions 1 --props config/consumer.properties --topic topic1



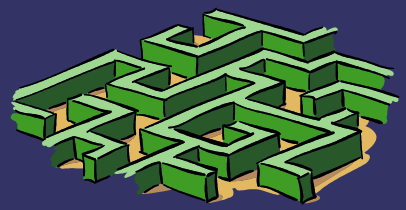
Demo 2: Produce with no consumer then reconnect



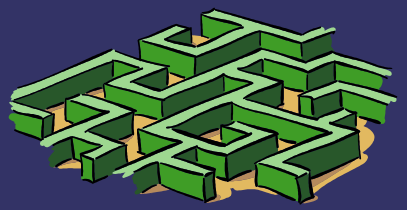
Stop all consumers produce messages

- Stop consumer1
- Produce messages
- Start consumer1

- Producing and consuming are not coupled
- Remember all messages are always persisted



Demo 3 multiple consumer groups

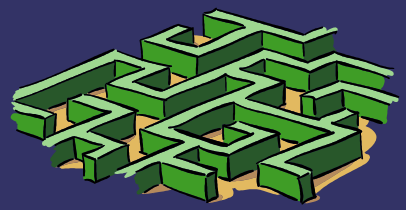


Start two consumers each with one consumer group

- bin/kafka-consumer-shell.sh --partitions 1 --props config/consumer-g1.properties --topic topic1
- bin/kafka-consumer-shell.sh --partitions 1 --props config/consumer-g2.properties --topic topic1

- #consumer group id groupid=test-consumer-group2

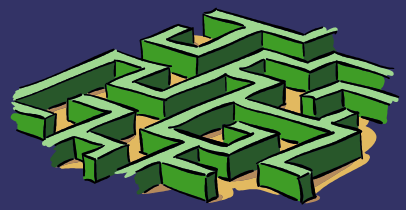
- Each message sent to topic gets delivered once to each group



Demo 4: Multiple clients on a consumer group

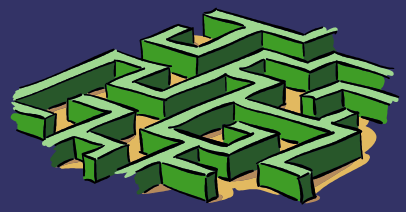
- bin/kafka-consumer-shell.sh --partitions 1 --props config/consumer-g2.properties --topic topic1
  (in a new shell)

- Each consumer group gets 1 copy of message
- messages are balanced across consumers in the group
  - Aka if consumer group has 3 consumers each get roughly 1/3rd of the messages (with random partitioning)



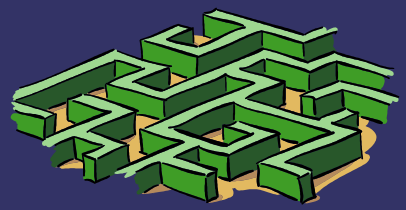
Processing kafka in a distributed fashion

- Each topic has N partitions
- We need 1-N consumers to process these messages
- Could write a class that implements Kafka Consumer interface and run this on N Nodes
- Enter IronCount (something I wrote)



Write handle message

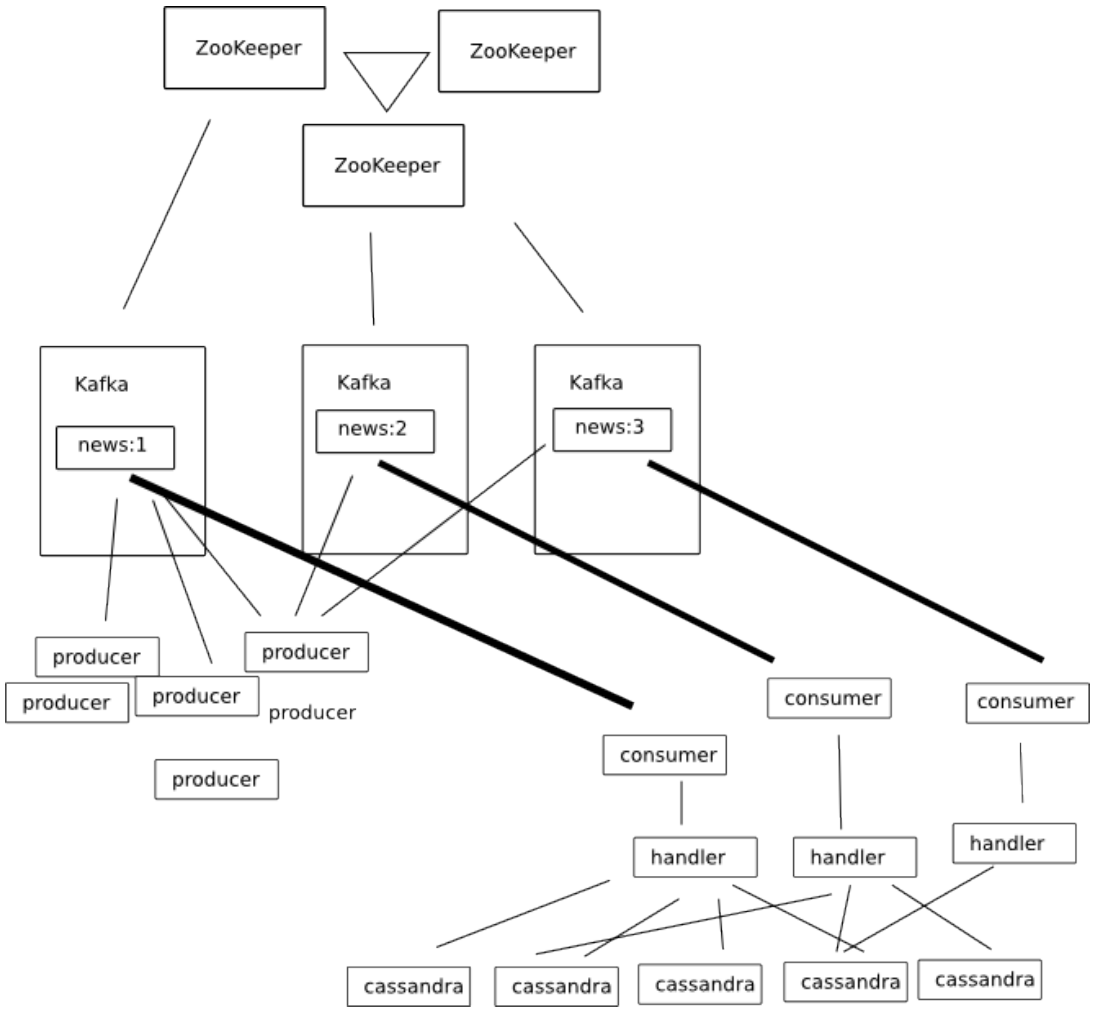
```
package com.jointhegrid.ironcount;

import kafka.message.Message;

public interface MessageHandler {
  public void setWorkload(Workload w);
  public void handleMessage(Message m) ;
  public void stop();
  public void setWorkerThread(WorkerThread wt) ;
}
```

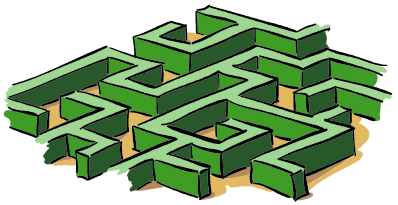


Real Time Count of city and state

```
public void handleMessage(Message m) {
    String [] parts = getMessage(m).split("\t");
    long time_in_ms=Long.parseLong( parts[1]);
    String city = parts[9];
    String state = parts[10];
    GregorianCalendar gc = new GregorianCalendar();
    gc.setTimeInMillis(time_in_ms);
    Mutator<String> mut =
HFactory.createMutator(keyspace, StringSerializer.get());
    HCounterColumn<String> hc =
HFactory.createCounterColumn(state+","+city, 1L);
    mut.addCounter( bucketByMinute.format(new Date()),
            w.properties.get("rts.cf"), hc);
    mut.execute();
  }
```

Aggregate To Cassandra





Wrap up

- Distributed design
- Designed for throughput not in memory per message semantics
- Survives fail overs of consumers and brokers
- Kafka partition is similar to a map/reduce partition
- Partial aggregations possible because the key of a message routes it to same consumer