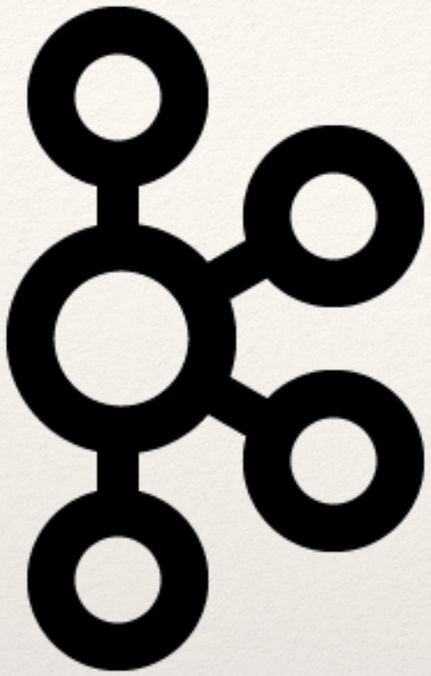


APACHE kafka™

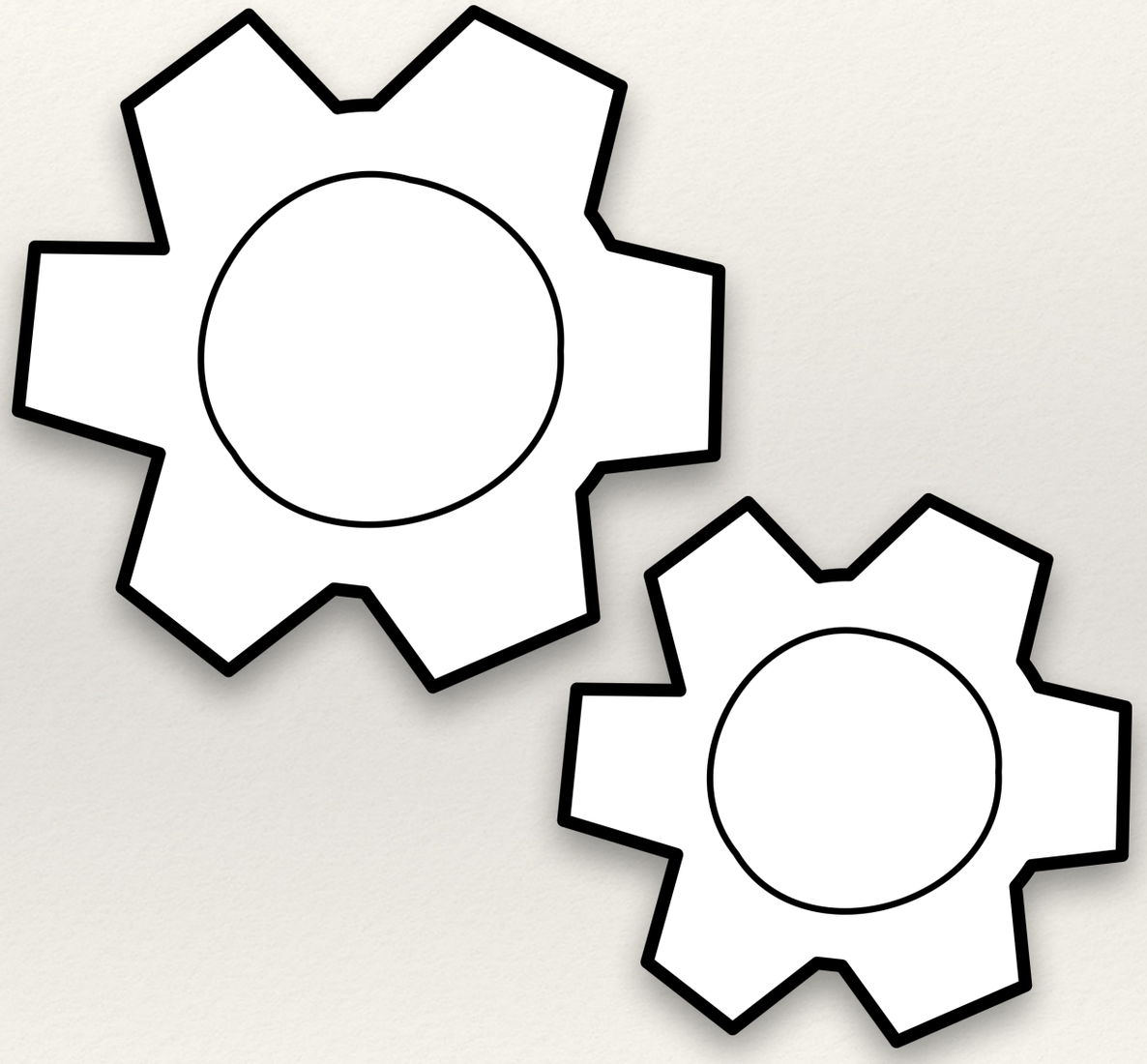
A distributed streaming platform

Directi

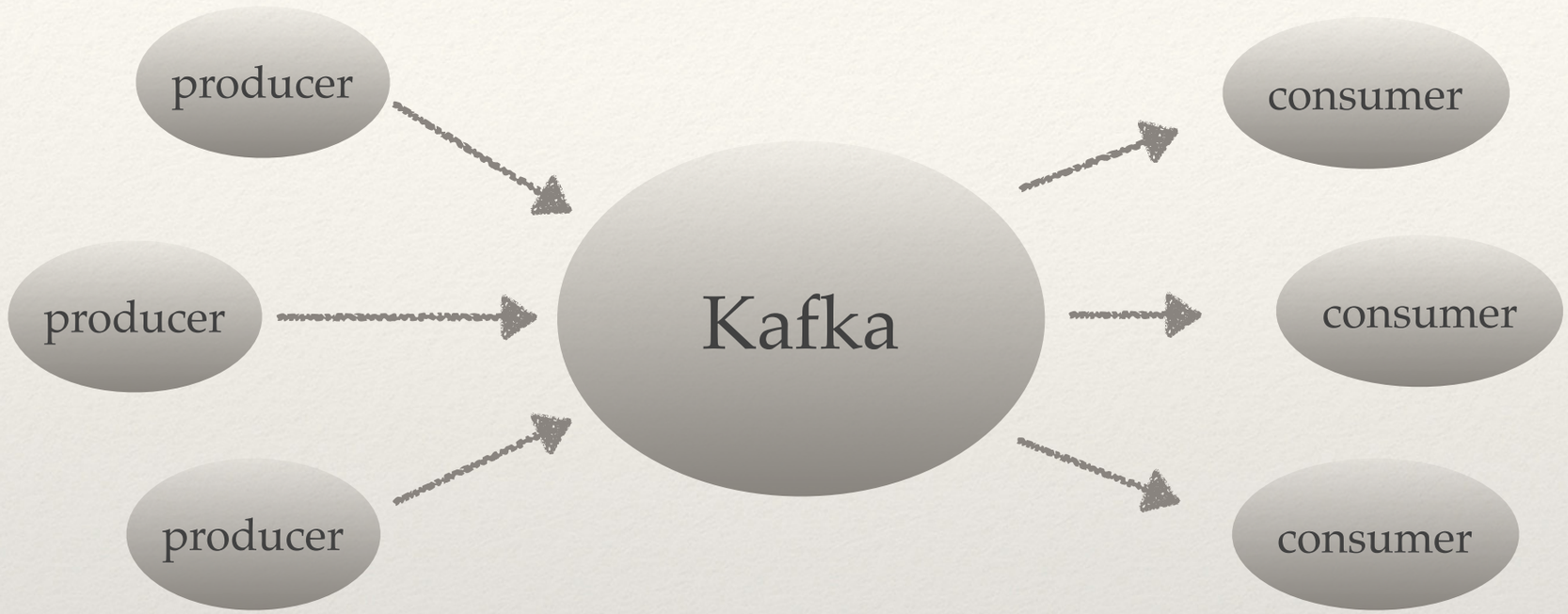
...

Computation Models

- Request Response Model
- Batch Processing Model
- Stream Model
  - ETL(Extract-Transform-Load)

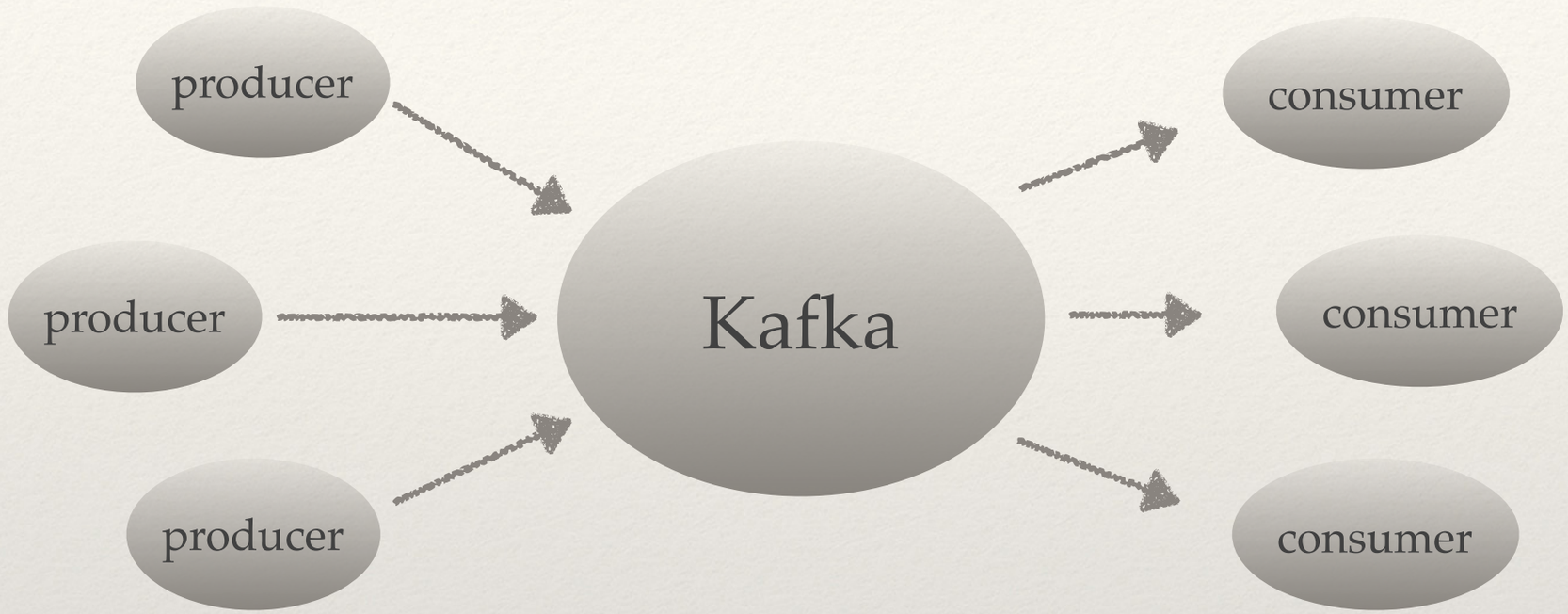


Introduction(What?)



- Apache Kafka is a **distributed** streaming platform
- Kafka lets you **publish** and **subscribe** to **streams** of records
- It lets you store streams of records in a **fault-tolerant** way.
- It lets you process streams of records as they occur.

Introduction(How?)



- Kafka is run as a cluster on one or more servers.
- The Kafka cluster stores streams of records in categories called topics.
- Each record consists of a key, a value, and a timestamp.
- Entities like producer, consumer, broker.

Application

- E-commerce
  - Analytics, Coupons, Campaigns, Fraud
- Event Sourcing
- Log aggregation
- Tracking

Entities in Kafka(Topic)

- Unit of address for related data
- Producer always produces data for an existing topic
- Consumer always consumes data by subscribing to any topic
- Similar to feeds.
- Multi-producer, multi-subscriber.(subscriber??)
- A partition-log is maintained for every topic.

Entities in Kafka(Partition)

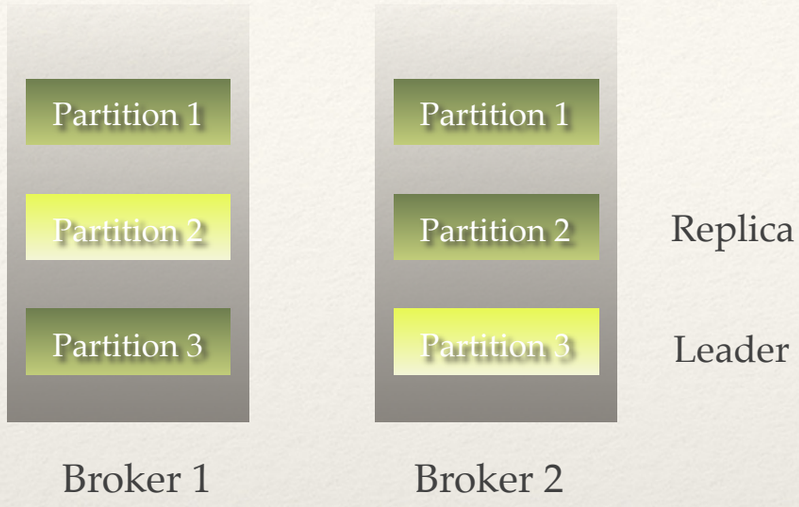
Partition 0	0(k_1,v,t)	1(k_2,v,t)	2(k_1,v,t)	3(k_3,v,t)
Partition 1	0(k_5,v,t)	1(k_6,v,t)	2(k_{11},v,t)	
Partition 2	0(k_7,v,t)	1(k_8,v,t)	2(k_9,v,t)	3(k_{10},v,t)

- Data Structure for storing topic's data.
- **Offset**(Key, Value, Timestamp)
- Data within a partition is **Ordered**, not in topic. Key is hashed to find out partition.
- Choose key wisely, so related data goes to a unique partition
  - Example: Generating payable after payment confirmation in e-commerce. Use order-id.
- Persistent data, independent of consumption, unless retention period in config expires.

Entities in Kafka(Offset)

- Offset is unique identity of every record within a partition.
- Offsets are maintained on per-consumer basis, so as to identify the progress of consumption.
- Offset is controlled by respective consumers so that they can even reprocess the data by reseting offset to older value.
- Auto-offset-commit(based on millis provided in config)
- Manual-offset-commit
  - On transactional processing
  - Per message commit
  - Batch commits

Entities in Kafka(Broker)



- Kafka cluster contains multiple nodes, each of them called brokers
- Brokers deals with partitions
- Every broker holds some partition(leader/replica)
- Replica ensures fault-tolerance
- Every partition is represented by leader copy, all read-writes go through leader.
- In case of leader failure, one of the replica is elected to be leader.

Entities in Kafka(Producer)

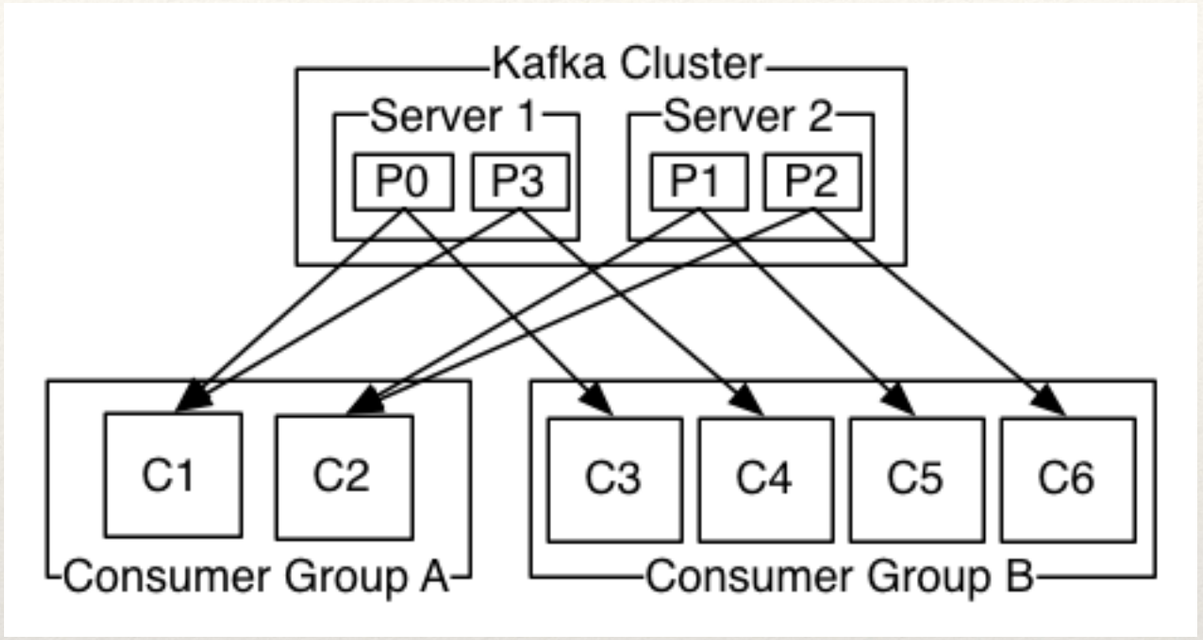
- Producers need to know about topic name to produce data
- They publish data(key, value) to a topic
- What happens internally?

- A (k,v) is directed to a **topic**.
- k is hashed to get destination **partition**
- Data is written to the **leader** of the partition.
- How parallelism is ensured in writes? / what about fault tolerance.

Entities in Kafka(Producer…)

- Every write is directed to leader broker for that partition.
- Every topic has multiple partition. We can get parallelism of degree to number of partitions.
- Even though data is produced from same producer node, key of the determines the partition and thus **parallelism**.
- After the writes at leader partition, it written back to the replica nodes(as Kafka stream itself)(details in a bit).

Entities in Kafka(Consumer)



- Consumers read data from a partition/ Consumer-Group reads data from a topic.
- A consumer group can contain just one consumer.
- Any partition can just be read by only one unique consumer in a consumer group.(Why???)
- So, this ensures parallelism on consumer end. Have more consumers in consumer group to increase the parallelism.
- What is the maximum number of effective consumers we can have for a given topic?

Kafka Design(Replication)

- Its configurable as replication factor on a topic basis.
- Unit of replication in kafka is partition
- Replica brokers consumes the data exactly as kafka model, ie pull model.
- When a node is called failed?
  - By **zookeeper(CAP theorm?)** heartbeat
  - Replication must not fall too far behind.

Kafka Design(Push vs Pull)

- Push has problems at broker end, as he controls the data push rate.
- Pull has intricacies at consumer, which needs producer to buffer.
- Kafka uses mix of push-pull model, where data is pushed to brokers and being pulled by consumer using the offset mechanisms.

Kafka Design(Message Delivery)

- Three models:
  - At most once
  - At least once
  - exactly once
- Generally at least once delivery is maintained.
- Consumer side guarantees, ensured by replication
- Producer side guarantees, every producer after writing a message to broker(remember push model applies here) wait for a success response
  - In case of non receipt, it resends the data to broker

Kafka Design(Message Delivery)

- This ensures at-least once delivery.
- In case message was delivered to broker and ack failed, we get duplicate message, thus at least once
- From **0.11 kafka(28 June,17)**
  - **Idempotent** operation is supported
  - Thus you can have exactly once mechanism.
  - How idempotent is supported… TCP sequence numbers

Kafka(Demo-Installation)

- Getting kafka

```
tar -xzf kafka_2.11-0.11.0.0.tgz
cd kafka_2.11-0.11.0.0
```

- Starting the zookeeper

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

Kafka(Demo-Starting server)

- Starting the server

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

- Creating a topic

```
bin/kafka-topics.sh --create --zookeeper localhost:
2181 --replication-factor 1 --partitions 1 --topic
test
bin/kafka-topics.sh --list --zookeeper localhost:
2181
```

Kafka(Demo-Sending messages)

- Sending messages

```
bin/kafka-console-producer.sh --broker-
list localhost:9092 --topic test
```

- Start a dummy consumer

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

Kafka(Demo-Setting cluster env)

- Setting multi-broker cluster(on same machine)

```
cp config/server.properties config/server-1.properties
cp config/server.properties config/server-2.properties
```

- Edit configs as

```
config/server-1.properties:
    broker.id=1
    listeners=PLAINTEXT://:9093
    log.dir=/tmp/kafka-logs-1
```

```
config/server-2.properties:
    broker.id=2
    listeners=PLAINTEXT://:9094
    log.dir=/tmp/kafka-logs-2
```

Kafka(Demo-Running multi-cluster env)

- Run both servers(zookeeper already running)

```
bin/kafka-server-start.sh config/server-1.properties &
bin/kafka-server-start.sh config/server-2.properties &
```

- Create topic with replication factor 3

```
bin/kafka-topics.sh --create --zookeeper localhost:2181
--replication-factor 3 --partitions 1 --topic my-
replicated-topic
```

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

Kafka(Demo-Testing fault tolerance)

- Lets kill the leader server

```
ps aux | grep server-1.properties
kill -9 pid
```

- Check the status of topic

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

References

- Documentation @ https://kafka.apache.org/

"If you shut up **truth** and bury it under the ground, it will but grow, and gather to itself such explosive power that the day it bursts through it will blow up everything in its way."

–EMILE ZOLA