Table of Contents

Apache Kafka

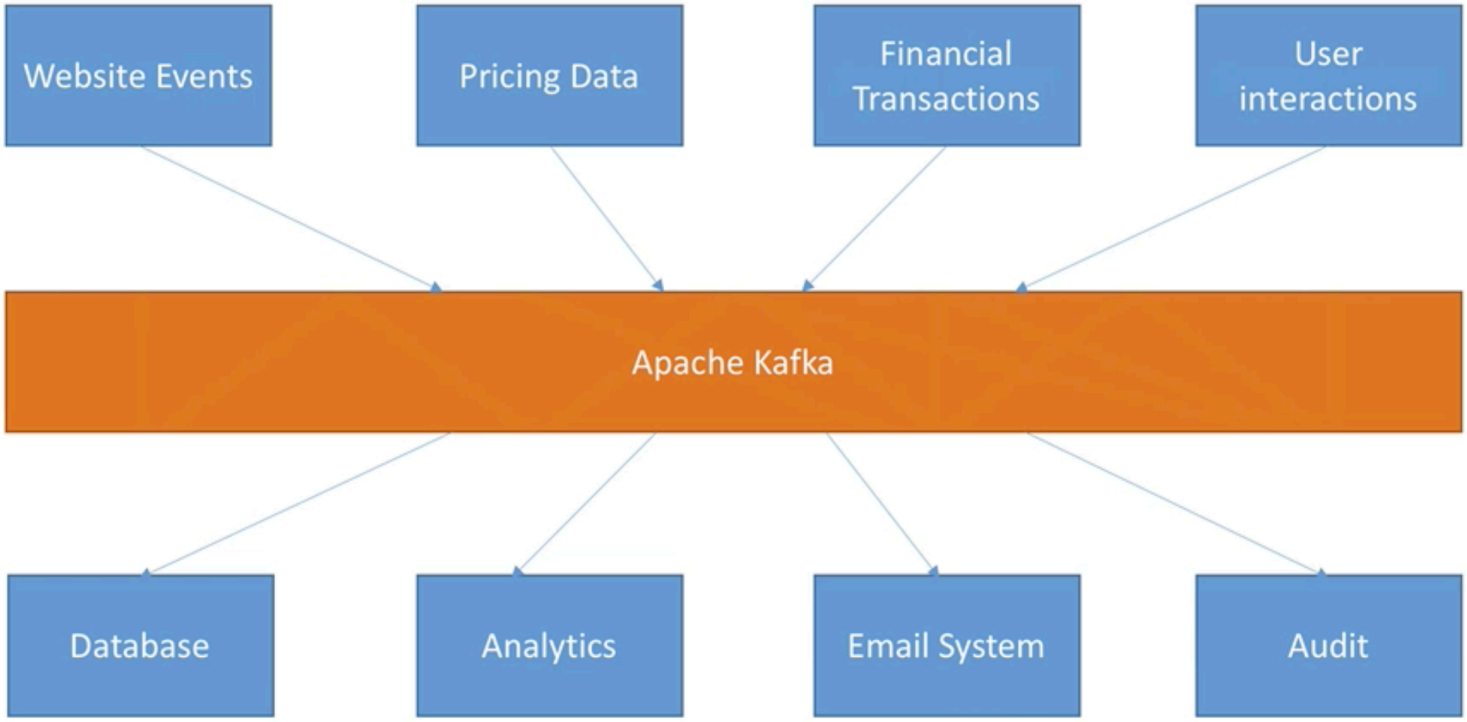
Version

Author	Version	Last update	Comment
luwenwu@cn.ibm.com	V1.0	04/11/18	Create the document

Introduction to Kafka and the ecosystem

Why Apache Kafka

Why Apache Kafka: Decoupling of data streams



Why Apache Kafka

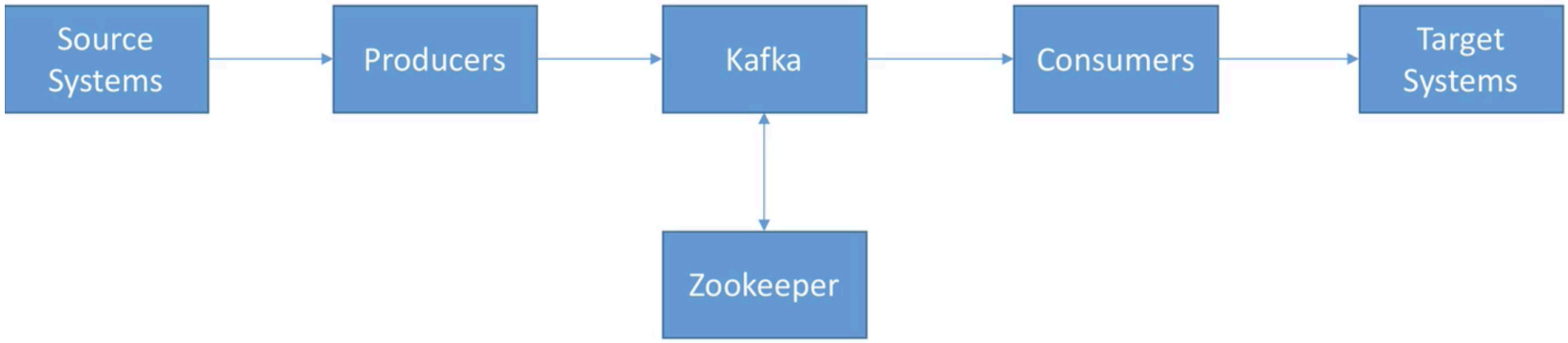
- Distributed, resilient architecture, fault tolerant
- Horizontal scalability
- High performance (latency of less than 10ms) – real time
- Used by the 2000+ firms:
  - LinkedIn
  - Netflix
  - AirBnB
  - Yahoo
  - Walmart

Apache Kafka: Use cases

- Messaging System
- Activity Tracking
- Gather metrics from many different locations
- Application Logs gathering
- Stream processing (with the Kafka Streams API or Spark for example)
- De-coupling of system dependencies
- Integration with Spark, Flink, Storm, Hadoop, and many other Big Data technologies

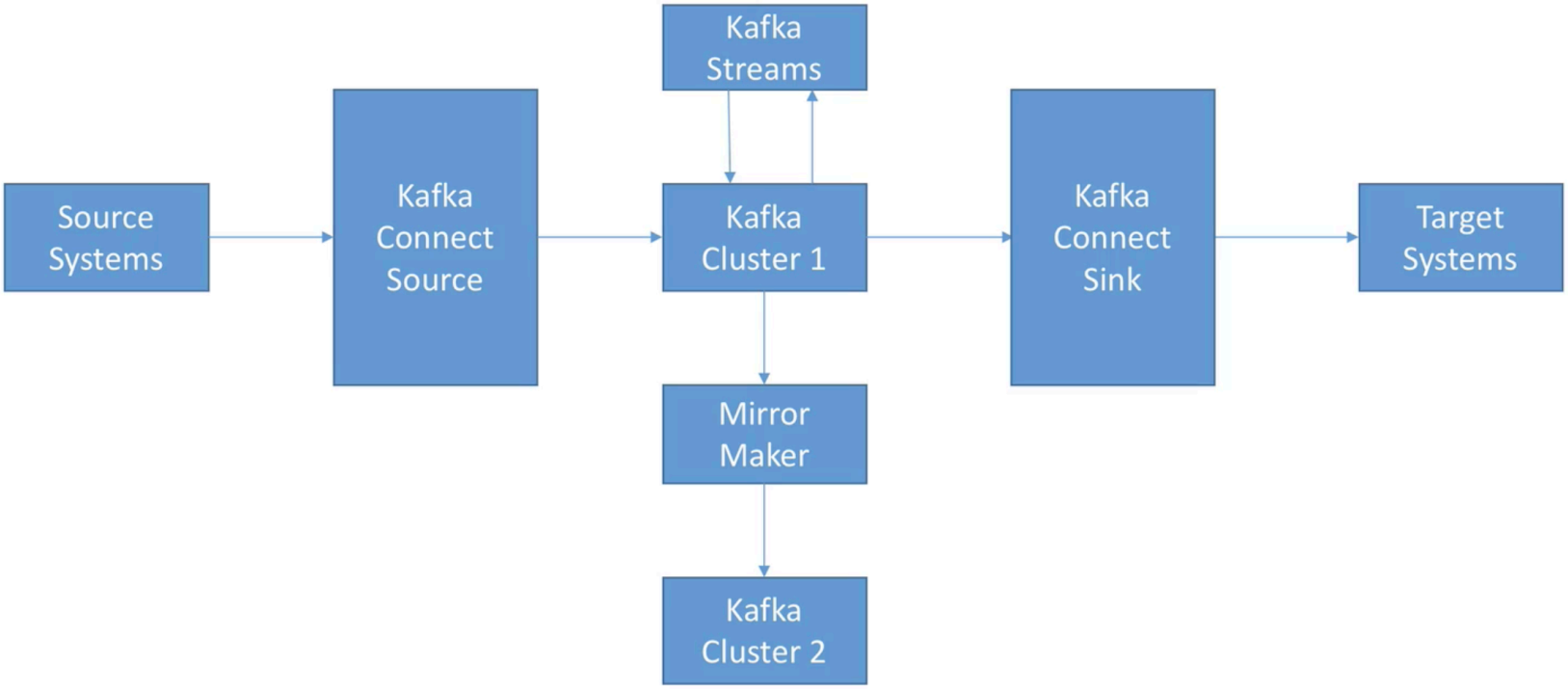
Kafka core API

Kafka Ecosystem
Kafka Core



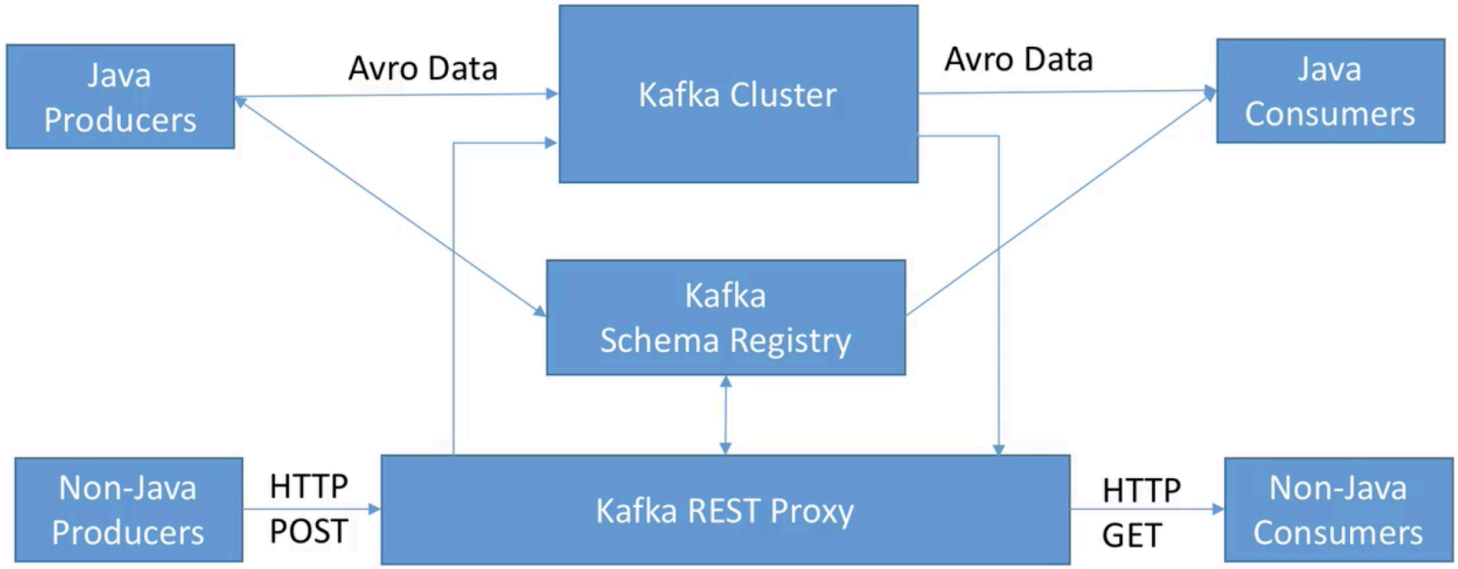
Kafka extended API

Kafka Ecosystem
Kafka Extended API



Confluent components Schema Registry and REST Proxy

Kafka Ecosystem: Confluent Components Schema Registry and REST Proxy



Kafka Ecosystem: Administration and Monitoring Tools

- Topics UI (Landoop): View the topics content
- Schema UI (Landoop): Explore the Schema registry
- Connect UI (Landoop): Create and monitor Connect tasks
- Kafka Manager (Yahoo): Overall Kafka Cluster Management
- Burrow (LinkedIn): Kafka Consumer Lag Checking
- Exhibitor (Netflix): Zookeeper Configuration, Monitoring, Backup
- Kafka Monitor (LinkedIn): Cluster health monitoring
- Kafka Tools (LinkedIn): Broker and topics administration tasks simplified
- Kafkat (Airbnb): More broker and topics administration tasks simplified
- JMX Dump: Dump JMX metrics from Brokers
- Control Centre / Auto Data Balancer / Replicator (Confluent): Paid tools

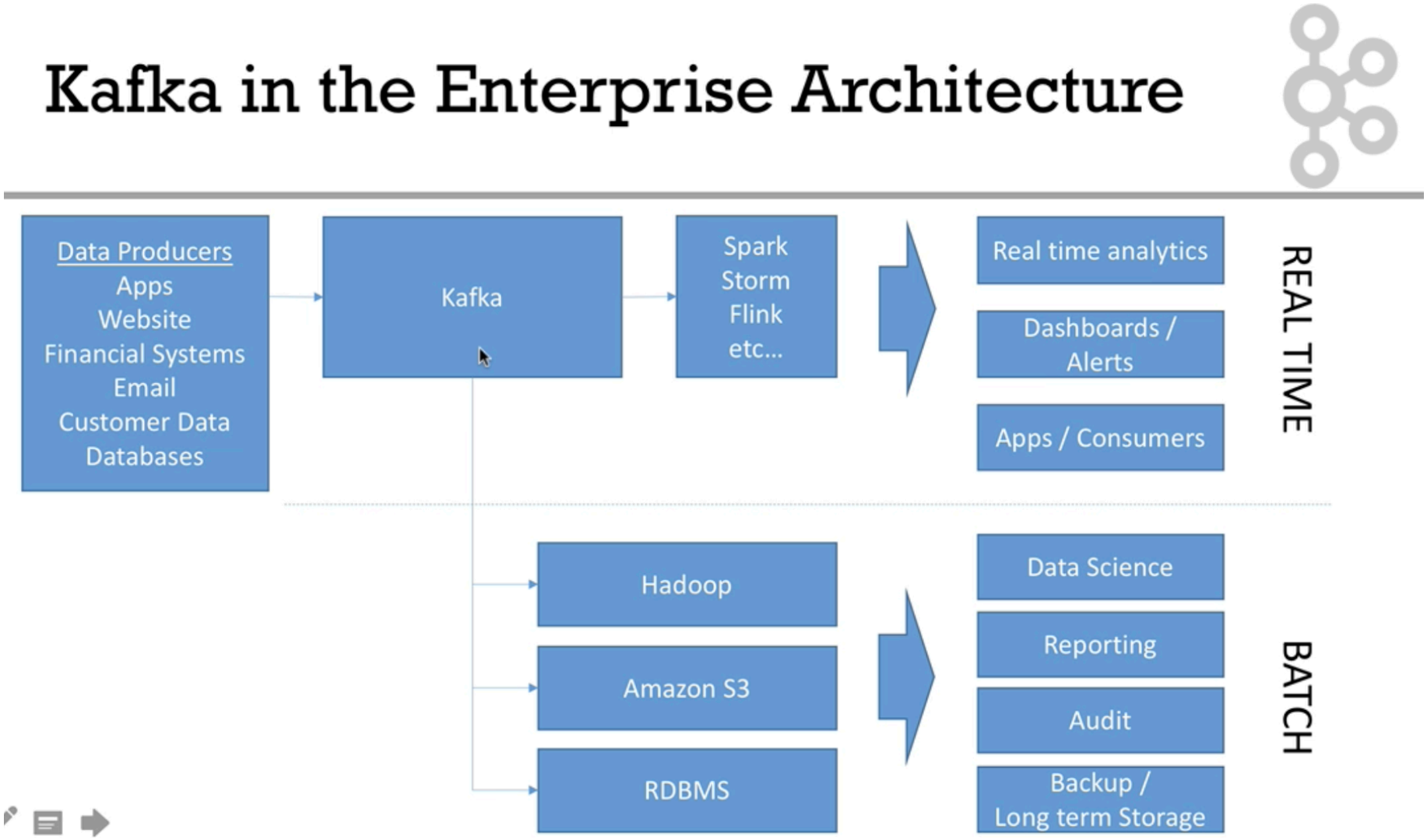
Course series

Apache Kafka Series

1. **Kafka For Beginners:** get a strong base for Kafka, basic operations, write your first producers and consumers
2. Kafka Connect API: Understand how to import / export data to / from Kafka
3. Kafka Streams API: Learn how to process and transform data within Kafka
4. Kafka Cluster Setup & Administration: Get a deep understanding of how Kafka & Zookeeper works, how to Setup Kafka and various administration tasks in AWS.
5. Confluent Components: REST Proxy and Schema Registry
6. Kafka Security: Setup Kafka security in a Cluster and Integrate your applications with Kafka Security

1 for all; 2, 3 for developers; 4 for administrators

Kafka in Enterprise Architecture



Quiz

Question 1:

Kafka is...

- ○ A new communication protocol
- ◉ A middle layer to decouple your real time data pipelines
- ○ A database

Question 2:

Kafka is not good for

- ○ Data Streaming
- ◉ Running computations such as data aggregations
- ○ De-coupling your data pipelines
- ○ Storing data in a distributed system

Question 3:

Kafka doesn't have the following capability

- ○ distributed system
- ○ horizontal scaling
- ○ fault tolerance
- ○ high performance
- ◉ SQL semantics

Question 4:

Kafka is most of the time used for

- (●) real-time streaming
- ( ) batch processing

How to ask question

How to Ask a Question in the Q&A

1. **Search** to see if the question has already been asked
2. Reference the Lecture number
3. Take screenshots
4. Copy the part of the log that has the error
5. Explain in details what you did, and what the error is.
6. Be as explicit as possible. I can't read your mind nor see your pc

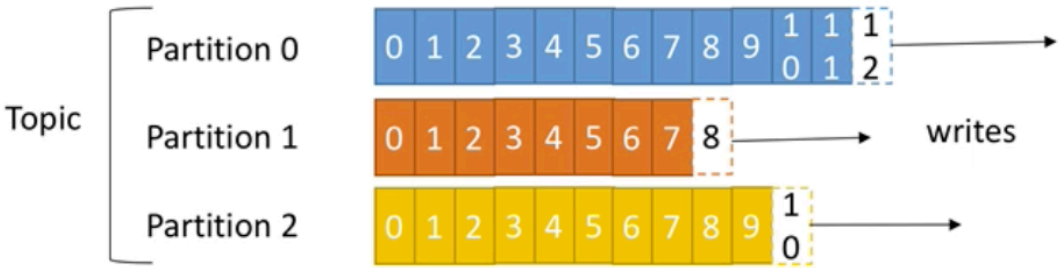
7. Wait. I'm a human, and I will try to reply in time

Apache Kafka core concepts

Topics and Partitions

Topics and partitions

- Topics: a particular stream of data
  - Similar to a table in a database (without all the constraints)
  - You can have as many topics as you want
  - A topic is identified by its name
- Topics are split in partitions
  - Each partition is ordered
  - Each message within a partition gets an incremental id, called offset



Topics and partitions

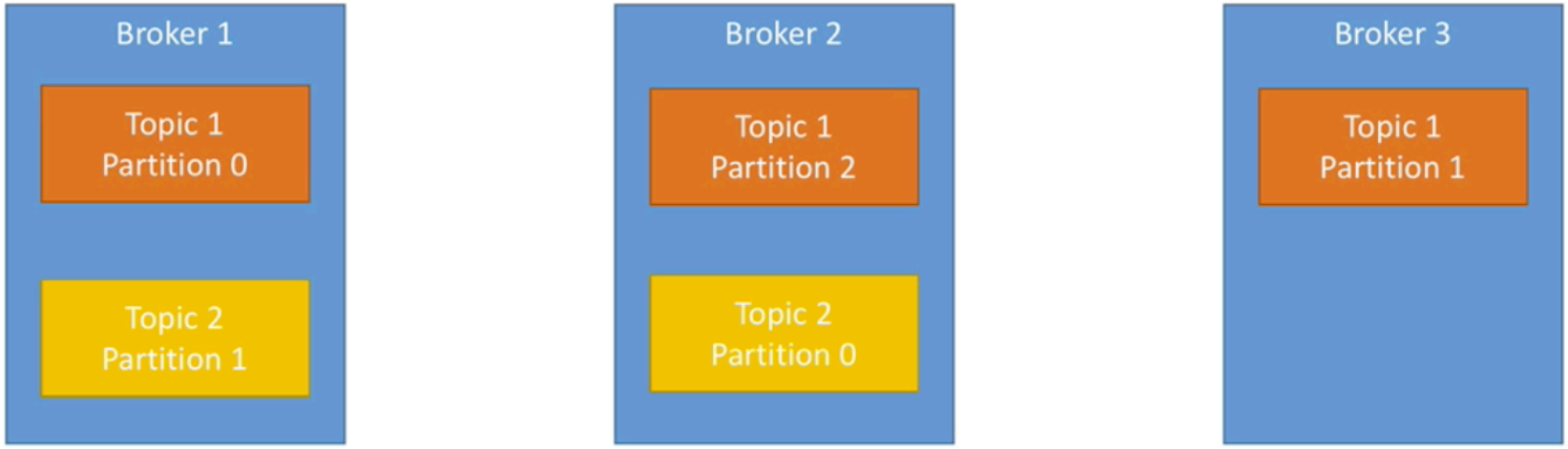
- Offset only have a meaning for a specific partition.
  - E.g. offset 3 in partition doesn't represent the same data as offset 3 in partition 1
- Order is guaranteed only within a partition (not across partitions)
- Data is kept only for a limited time (default is two weeks)
- Once the data is written to a partition, it can't be changed (immutability)
- Data is assigned randomly to a partition unless a key is provided (more on this later)
- You can have as many partitions per topic as you want

Brokers

- A Kafka cluster is composed of multiple brokers (servers)
- Each broker is identified with its ID (integer)
- Each broker contains certain topic partitions
- After connecting to any broker (called a bootstrap broker), you will be connected to the entire cluster
- A good number to get started is 3 brokers, but some big clusters have over 100 brokers

Brokers and topics

- Example of 2 topics (3 partitions and 2 partitions)

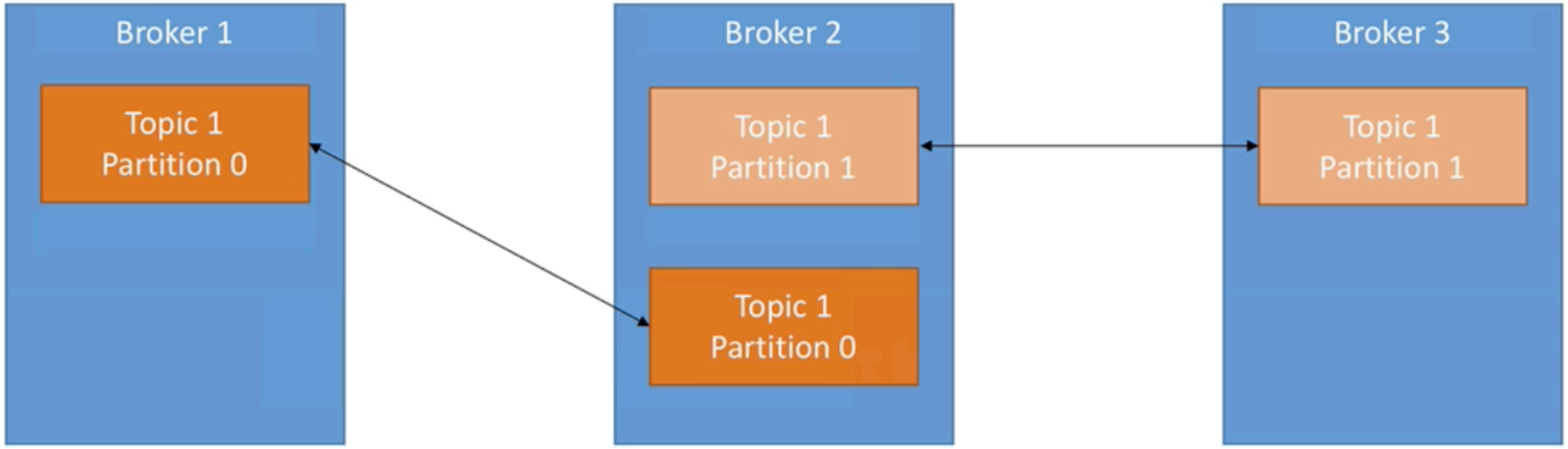


- Data is distributed and Broker 3 doesn't have any Topic 2 data

Topic replication factor

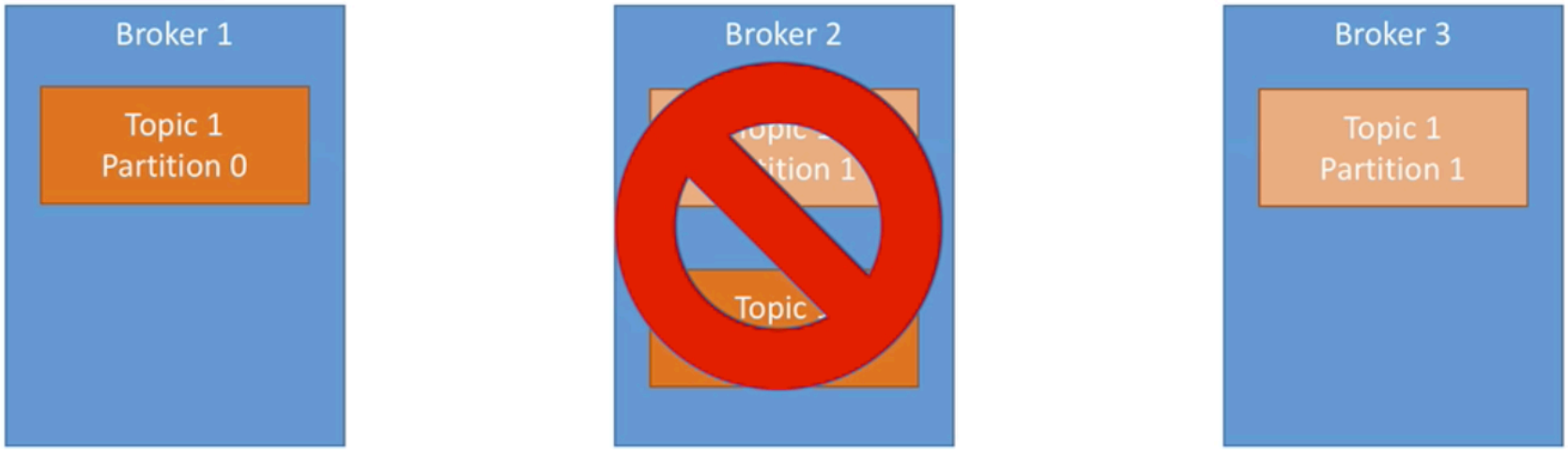
Topic replication factor

- Topics should have a replication factor > 1 (usually between 2 and 3)
- This way if a broker is down, another broker can serve the data
- Example: Topic with 2 partitions and replication factor of 2



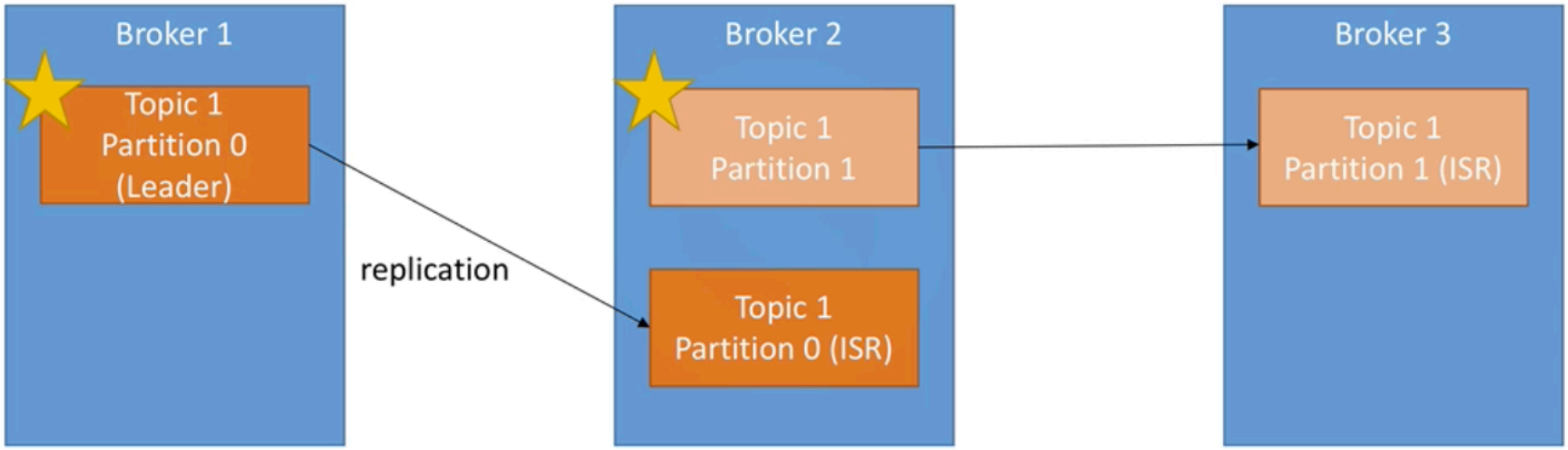
Topic replication factor

- Example: we lost Broker 2
- Result: Broker 1 and 3 can still serve the data



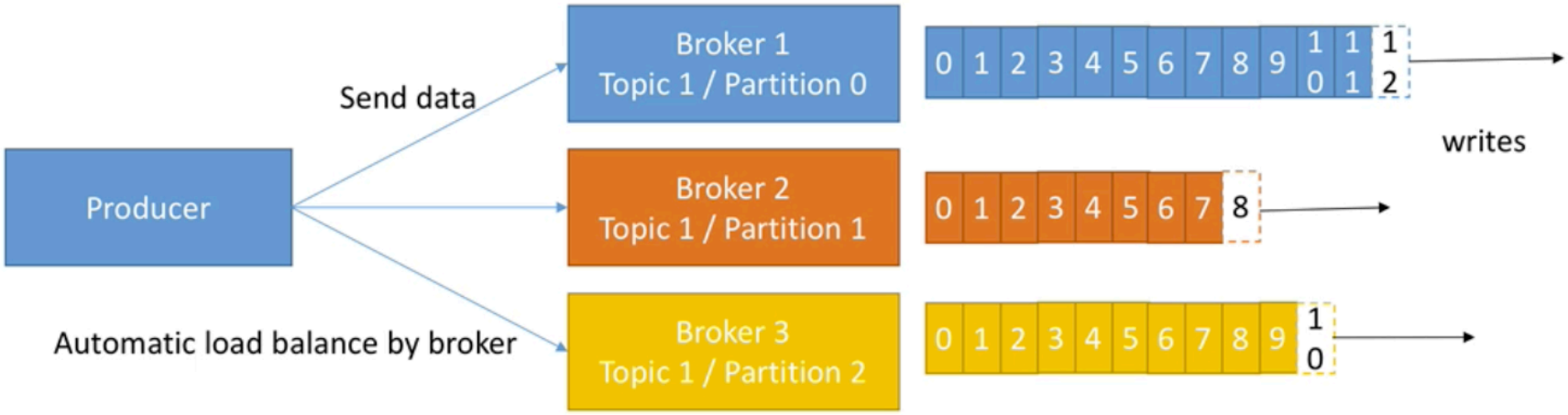
Concept of Leader for a partition

- **At any time only 1 broker can be a leader for a given partition**
- **Only that leader can receive and serve data for a partition**
- The other brokers will synchronize the data
- There each partition has: one leader, and multiple ISR (in-sync replica)



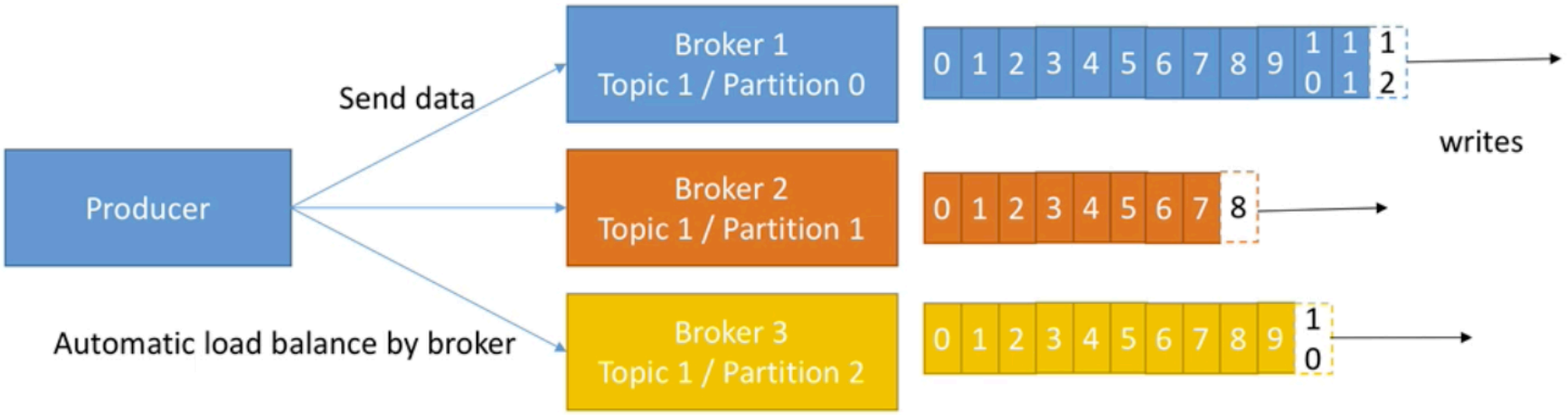
Producers

- Producers write data to topics.
- **They only have to specify the topic name and one broker to connect to, and Kafka will automatically take care of routing the data to the right brokers**



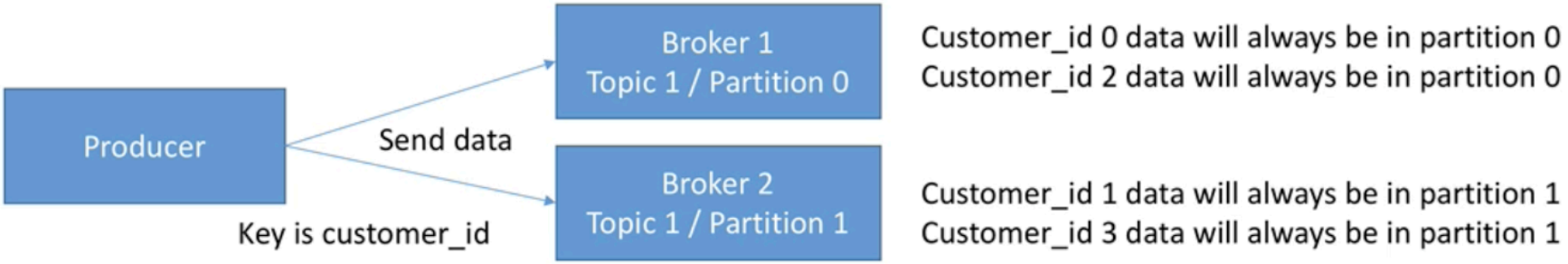
Producers

- Producers can choose to receive acknowledgment of data writes:
  - Acks=0: Producer won't wait for acknowledgment (possible data loss)
  - Acks=1: Producer will wait for leader acknowledgment (limited data loss)
  - Acks=all: Leader + replicas acknowledgment (no data loss)



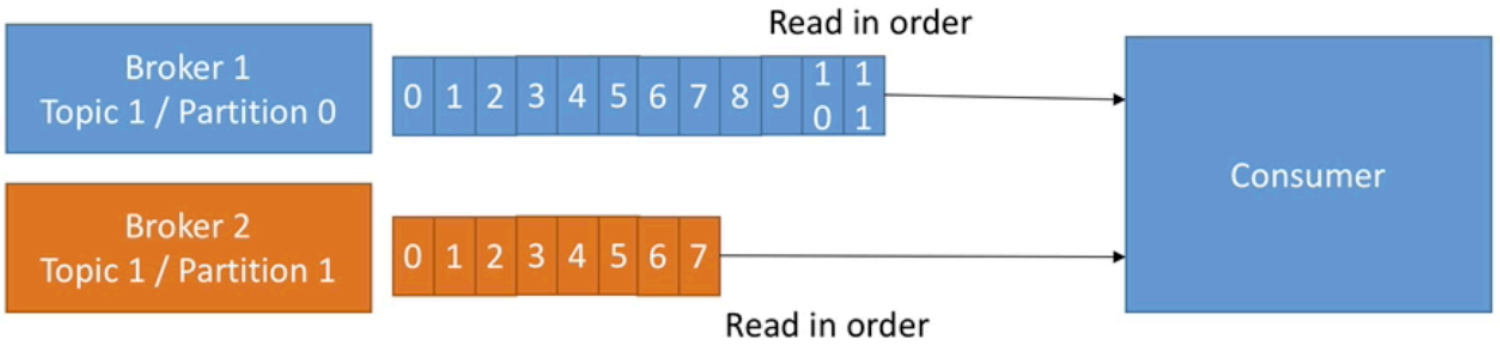
Producers: Message keys

- Producers can choose to send a key with the message
- If a key is sent, then the producer has the guarantee that all messages for that key will always go to the same partition
- This enables to guarantee ordering for a specific key



Consumers

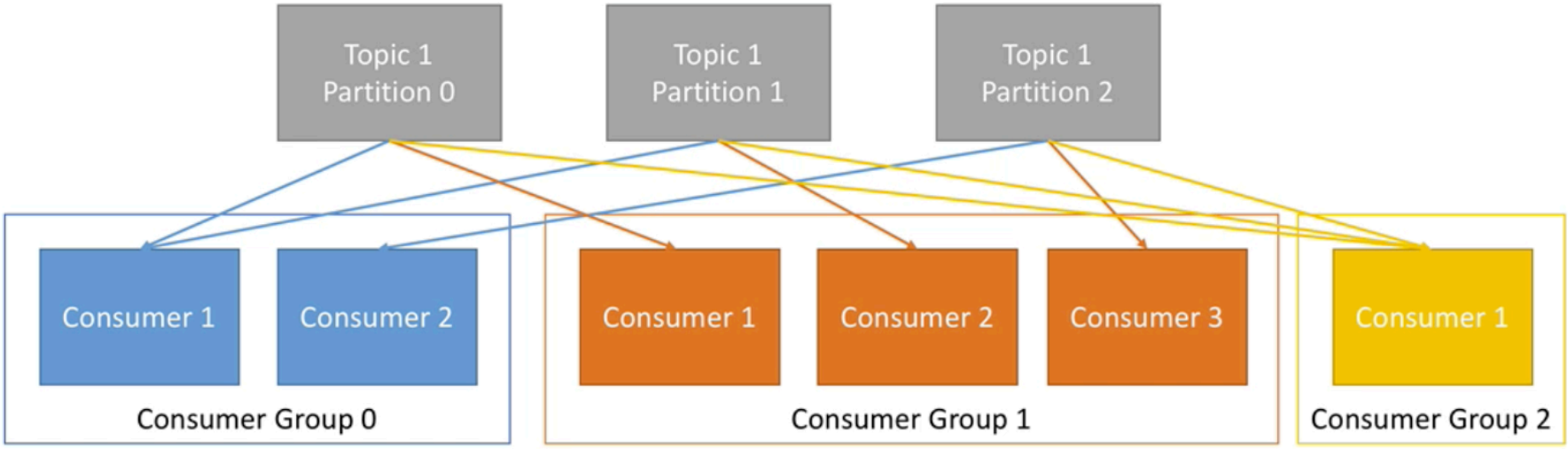
- Consumers read data from a topic
- **They only have to specify the topic name and one broker to connect to, and Kafka will automatically take care of pulling the data from the right brokers**
- Data is read in order **for each partitions**



Consumer groups

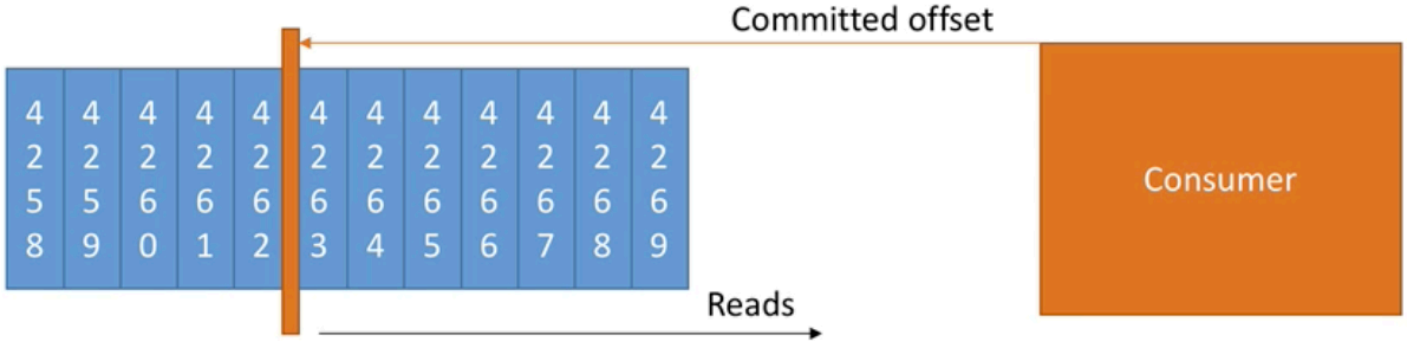
Consumer Groups

- Consumers read data in consumer groups
- Each consumer within a group reads from exclusive partitions
- You cannot have more consumers than partitions (otherwise some will be inactive)



Consumer Offsets

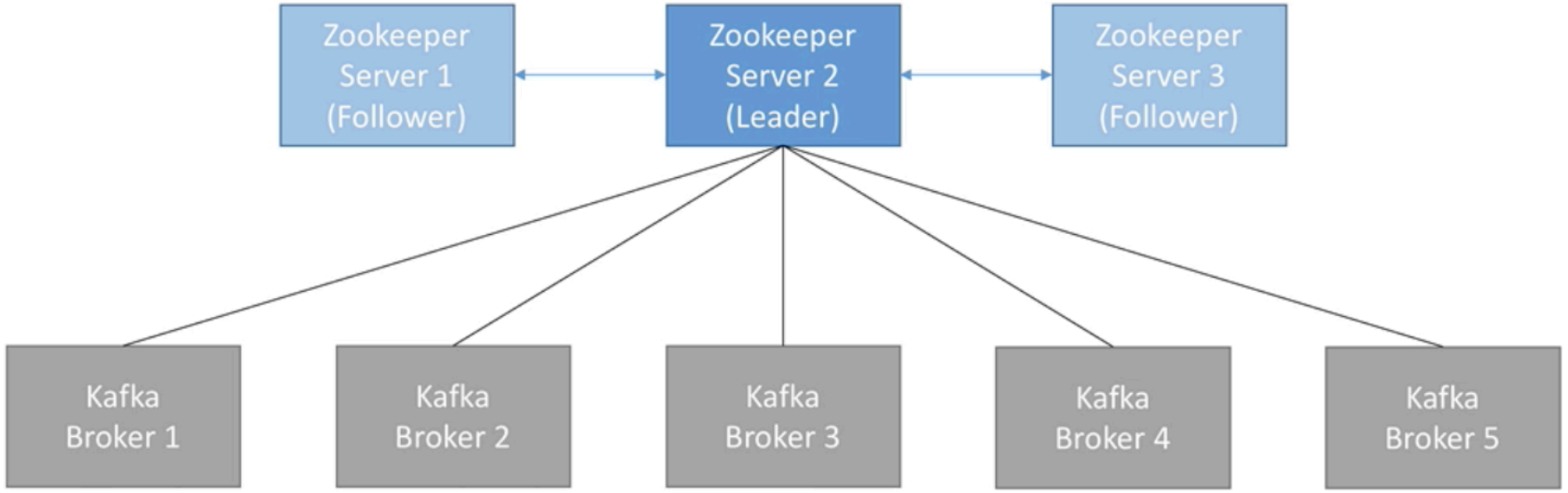
- Kafka stores the offsets at which a consumer group has been reading
- The offsets commit live in a Kafka topic named "__consumer_offsets"
- When a consumer has processed data received some Kafka, it should be committing the offsets
- If a consumer process dies, it will be able to read back from where it left off thanks to consumer offsets!



Zookeeper

- Zookeeper manages brokers (keeps a list of them)
- Zookeeper helps in performing leader election for partitions
- Zookeeper sends notifications to Kafka in case of changes (e.g. new topic, broker dies, broker comes up, delete topics, etc.…)
- **Kafka can't work without Zookeeper**
- Zookeeper usually operates in an odd quorum (cluster) of servers (3, 5, 7)
- Zookeeper has a leader, the rest of the servers are followers

Zookeeper


Zookeeper Server 1 (Follower)
Zookeeper Server 2 (Leader)
Zookeeper Server 3 (Follower)
Kafka Broker 1
Kafka Broker 2
Kafka Broker 3
Kafka Broker 4
Kafka Broker 5

Kafka Guarantees

- Messages are appended to a topic-partition in the order they are sent
- Consumers read messages in the order stored in a topic-partition
- With a replication factor of N, producers and consumers can tolerate up to N-1 brokers being down
- This is why a replication factor of 3 is a good idea:
  - Allows for one broker to be taken down for maintenance
  - Allows for another broker to be taken down unexpectedly
- As long as the number of partitions remains constant for a topic (no new partitions), the same key will always go to the same partition

Delivery semantics for consumers

- As learned before, consumers choose when to commit offsets.

- At most once: offsets are committed as soon as the message is received. If the processing goes wrong, the message will be lost (it won't be read again).
- At least once: offsets are committed after the message is processed. If the processing goes wrong, the message will be read again. This can result in duplicate processing of messages. Make sure your processing is **idempotent** (i.e. processing again the messages won't impact your systems)
- Exactly once: Very difficult to achieve / needs strong engineering.

Bottom line: most often you should use at least once processing and ensure your transformations / processing are idempotent

Quiz

Question 1:

Kafka topics...

- ○ always have 1 partition
- ◉ can have as many partitions as desired

Question 2:

Offsets only at the level of

- ○ the topic
- ◉ the topic-partition

Question 3:

Once sent to a topic, a message can be modified

- ○ true
- ◉ false

Question 4:

Brokers are identified by

- ○ A name (string)
- ◉ An ID (number)

Question 5:

Every brokers

- ○ contain all the topics and all the partitions
- ◉ contain only a subset of the topics and the partitions

Question 6:

If a topic has a replication factor of 3

- (●) Each partition will live on 3 different brokers
- ( ) Each partition will live on 2 different brokers
- ( ) Each partition will live on 4 different brokers

Question 7:

If a topic has a replication factor of 3, what maximum number of brokers can be stopped without impacting the cluster?

- ○ 1
- ◉ 2
- ○ 3

Question 8:

Each partition can only have 1 leader, and multiple replicas

- (●) true
- ( ) false

Good job!

Very important: you only need to connect to one broker (any broker) and just provide the topic name you want to write to. Kafka will route your data to the appropriate brokers and partitions for you!

Question 9:

To produce data to a topic, a producer must provide the Kafka client with...

- (●) any broker from the cluster and the topic name
- ( ) any broker from the cluster and the topic name and the partitions list
- ( ) all the brokers from the cluster and the topic name
- ( ) the list of brokers that have the data, the topic name, and the partitions list

Question 10:

To get acknowledgement of writes to only the leader, we need to use the config...

- (●) acks=1
- ( ) acks=0
- ( ) acks=all

Question 11:

To read data from a topic, the following configuration is needed for the consumers

- (●) any brokers to connect to, and the topic name
- ( ) all brokers of the cluster, and the topic name
- ( ) any brokers, and the list of topic partitions

Question 12:

Two consumers that have the same group.id (consumer group id) will read from mutually exclusive partitions

- (●) true
- ( ) false

Question 13:

As of Kafka Version >= 0.9.0
Zookeeper is not used

- ○ to perform a leader election
- ○ to maintain the list of all the brokers in the cluster
- ○ to send notifications to Kafka in case of any changes
- ◉ to maintain consumer offsets

Install Docker and Docker compose

Install Docker

I have installed Docker and Docker compose on my Linux server. So I'd skip this step. But I'd show the repo here for reference purpose:

```
[root@sbybz3137 ~]# cat /etc/yum.repos.d/docker-ce.repo
[docker-ce-stable]
name=Docker CE Stable - $basearch
baseurl=https://download.docker.com/linux/centos/7/$basearch/stable
enabled=1
gpgcheck=1
gpgkey=https://download.docker.com/linux/centos/gpg
```

Kafka Docker for development

Kafka, Zookeeper, Schema Registry, Kafka-Connect, Landoop Tools, 20+ connectors
https://github.com/Landoop/fast-data-dev

Start Kafka

```
docker run --rm --net=host -e ADV_HOST=sbybz3137.sby.ibm.com landoop/fast-data-dev
```

```
[root@sbybz3137 ~]# docker run --rm --net=host landoop/fast-data-dev
Unable to find image 'landoop/fast-data-dev:latest' locally
latest: Pulling from landoop/fast-data-dev
ff3a5c916c92: Already exists
0a6f3946d366: Pull complete
38cbac368ac8: Pull complete
940e9a6559b3: Pull complete
2533ff68afb1: Pull complete
```

```
5310643b7010: Pull complete
4a0bf0f7b38c: Pull complete
256a5053f0c5: Pull complete
df59bd94bdd4: Pull complete
6aac547c4d90: Pull complete
1818c7d8b194: Pull complete
8297d3821b7b: Pull complete
27f5c49688ee: Pull complete
3a13574f4ddd: Pull complete
13d13524515b: Pull complete
dd4dddc9dc97: Pull complete
633bee4809e9: Pull complete
a6842c7eb2cf: Pull complete
22ae19fc8a44: Pull complete
c58290624b9c: Pull complete
09afeafbcca3: Pull complete
30033ace6359: Pull complete
14951d7d8ded: Pull complete
6797a96a7092: Pull complete
5e958485d959: Pull complete
6e77d9fc1e81: Pull complete
99046468067f: Pull complete
def900f2c3f5: Pull complete
e3298ece27a7: Pull complete
46b35ea8b707: Pull complete
498af3d85513: Pull complete
c29a64ba4098: Pull complete
d5267660efab: Pull complete
068d760d616b: Pull complete
12c2d93a4a0c: Pull complete
Digest: sha256:041358122a185144261c95cb8abfb38d47461d585e1672ed58d3ce9800634629
```

```
Status: Downloaded newer image for landoop/fast-data-dev:latest
Starting services.
This is landoop's fast-data-dev. Kafka 0.11.0.1, Confluent OSS 3.3.1.
You may visit http://localhost:3030 in about a minute.
2018-04-12 20:57:06,795 CRIT Supervisor running as root (no user in config file)
2018-04-12 20:57:06,795 INFO Included extra file "/etc/supervisord.d/01-fast-data.conf" during parsing
2018-04-12 20:57:06,795 INFO Included extra file "/etc/supervisord.d/99-supervisord-sample-data.conf" during parsing
2018-04-12 20:57:06,797 INFO supervisord started with pid 7
2018-04-12 20:57:07,800 INFO spawned: 'sample-data' with pid 99
2018-04-12 20:57:07,802 INFO spawned: 'zookeeper' with pid 100
2018-04-12 20:57:07,805 INFO spawned: 'caddy' with pid 101
2018-04-12 20:57:07,808 INFO spawned: 'broker' with pid 102
2018-04-12 20:57:07,811 INFO spawned: 'smoke-tests' with pid 103
2018-04-12 20:57:07,817 INFO spawned: 'connect-distributed' with pid 106
2018-04-12 20:57:07,819 INFO spawned: 'logs-to-kafka' with pid 110
2018-04-12 20:57:07,821 INFO spawned: 'schema-registry' with pid 111
2018-04-12 20:57:07,823 INFO spawned: 'rest-proxy' with pid 116
2018-04-12 20:57:09,464 INFO success: sample-data entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
2018-04-12 20:57:09,464 INFO success: zookeeper entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
2018-04-12 20:57:09,464 INFO success: caddy entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
2018-04-12 20:57:09,464 INFO success: broker entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
2018-04-12 20:57:09,464 INFO success: smoke-tests entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
2018-04-12 20:57:09,464 INFO success: connect-distributed entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
2018-04-12 20:57:09,464 INFO success: logs-to-kafka entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
2018-04-12 20:57:09,464 INFO success: schema-registry entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
2018-04-12 20:57:09,464 INFO success: rest-proxy entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
2018-04-12 20:59:32,743 INFO exited: sample-data (exit status 0; expected)
2018-04-12 21:00:08,651 INFO exited: logs-to-kafka (exit status 0; expected)
2018-04-12 21:02:12,065 INFO exited: smoke-tests (exit status 0; expected)
```

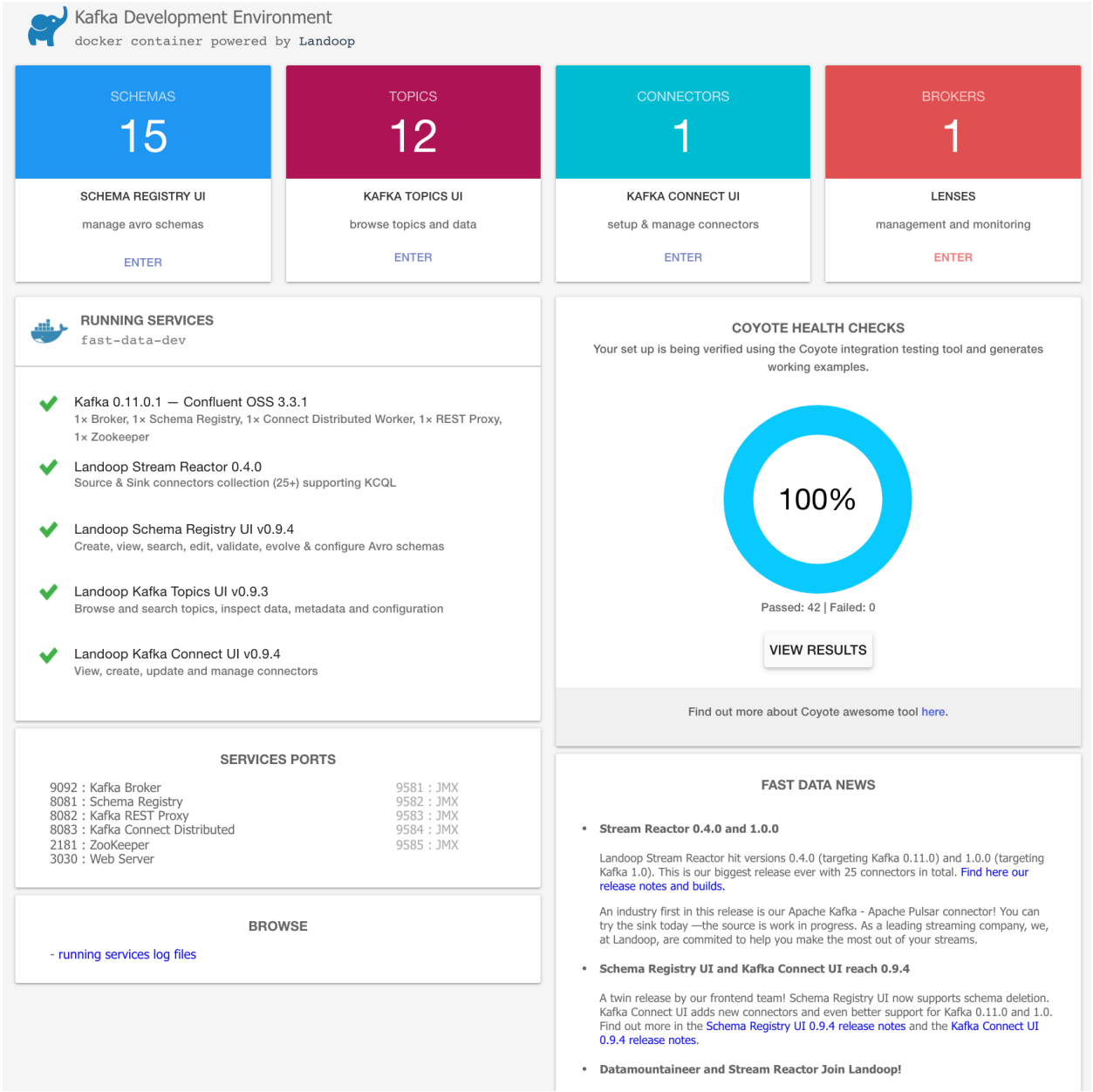
Or other options:
If you want to have the services remotely accessible, then you need to pass in your machine's IP address or hostname that other machines can use to access it:

docker run --rm --net=host landoop/fast-data-dev

docker run --rm --net=host -e ADV_HOST=<IP> landoop/fast-data-dev

docker run --rm --net=host -e ADV_HOST=sbybz3137.sby.ibm.com landoop/fast-data-dev

http://sbybz3137.sby.ibm.com:3030/


Kafka Development Environment
docker container powered by Landoop
SCHEMAS
15
SCHEMA REGISTRY UI
manage avro schemas
ENTER
TOPICS
12
KAFKA TOPICS UI
browse topics and data
ENTER
CONNECTORS
1
KAFKA CONNECT UI
setup & manage connectors
ENTER
BROKERS
1
LENSES
management and monitoring
ENTER
RUNNING SERVICES
fast-data-dev
Kafka 0.11.0.1 — Confluent OSS 3.3.1
1× Broker, 1× Schema Registry, 1× Connect Distributed Worker, 1× REST Proxy, 1× Zookeeper
Landoop Stream Reactor 0.4.0
Source & Sink connectors collection (25+) supporting KCQL
Landoop Schema Registry UI v0.9.4
Create, view, search, edit, validate, evolve & configure Avro schemas
Landoop Kafka Topics UI v0.9.3
Browse and search topics, inspect data, metadata and configuration
Landoop Kafka Connect UI v0.9.4
View, create, update and manage connectors
COYOTE HEALTH CHECKS
Your set up is being verified using the Coyote integration testing tool and generates working examples.
100%
Passed: 42 | Failed: 0
VIEW RESULTS
Find out more about Coyote awesome tool here.
SERVICES PORTS
9092 : Kafka Broker
8081 : Schema Registry
8082 : Kafka REST Proxy
8083 : Kafka Connect Distributed
2181 : ZooKeeper
3030 : Web Server
9581 : JMX
9582 : JMX
9583 : JMX
9584 : JMX
9585 : JMX
BROWSE
- running services log files
FAST DATA NEWS
Stream Reactor 0.4.0 and 1.0.0
Landoop Stream Reactor hit versions 0.4.0 (targeting Kafka 0.11.0) and 1.0.0 (targeting Kafka 1.0). This is our biggest release ever with 25 connectors in total. Find here our release notes and builds.
An industry first in this release is our Apache Kafka - Apache Pulsar connector! You can try the sink today —the source is work in progress. As a leading streaming company, we, at Landoop, are commited to help you make the most out of your streams.
Schema Registry UI and Kafka Connect UI reach 0.9.4
A twin release by our frontend team! Schema Registry UI now supports schema deletion. Kafka Connect UI adds new connectors and even better support for Kafka 0.11.0 and 1.0. Find out more in the Schema Registry UI 0.9.4 release notes and the Kafka Connect UI 0.9.4 release notes.
Datamountaineer and Stream Reactor Join Landoop!

Apache Kafka: Hands-on practice

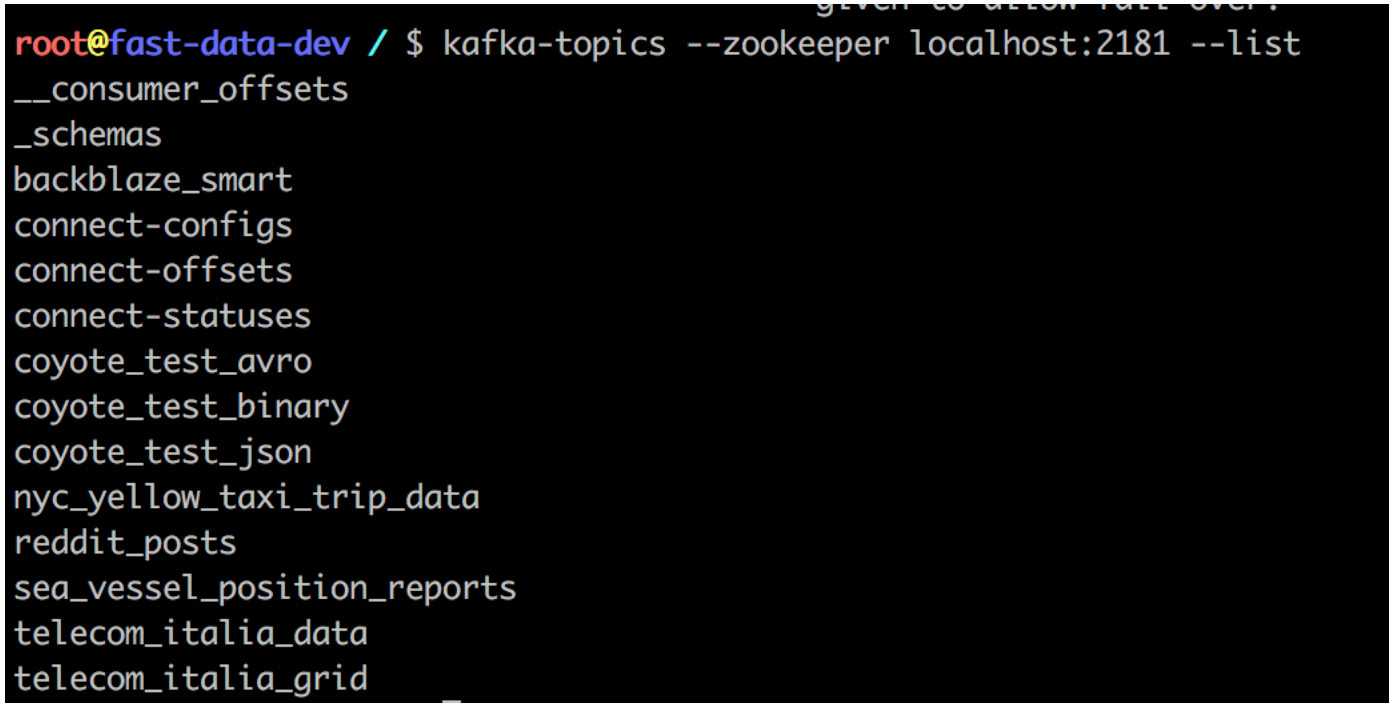
Topic operations: create, list, delete, describe

Kafka cli tool

docker run -it --rm --net=host landoop/fast-data-dev bash

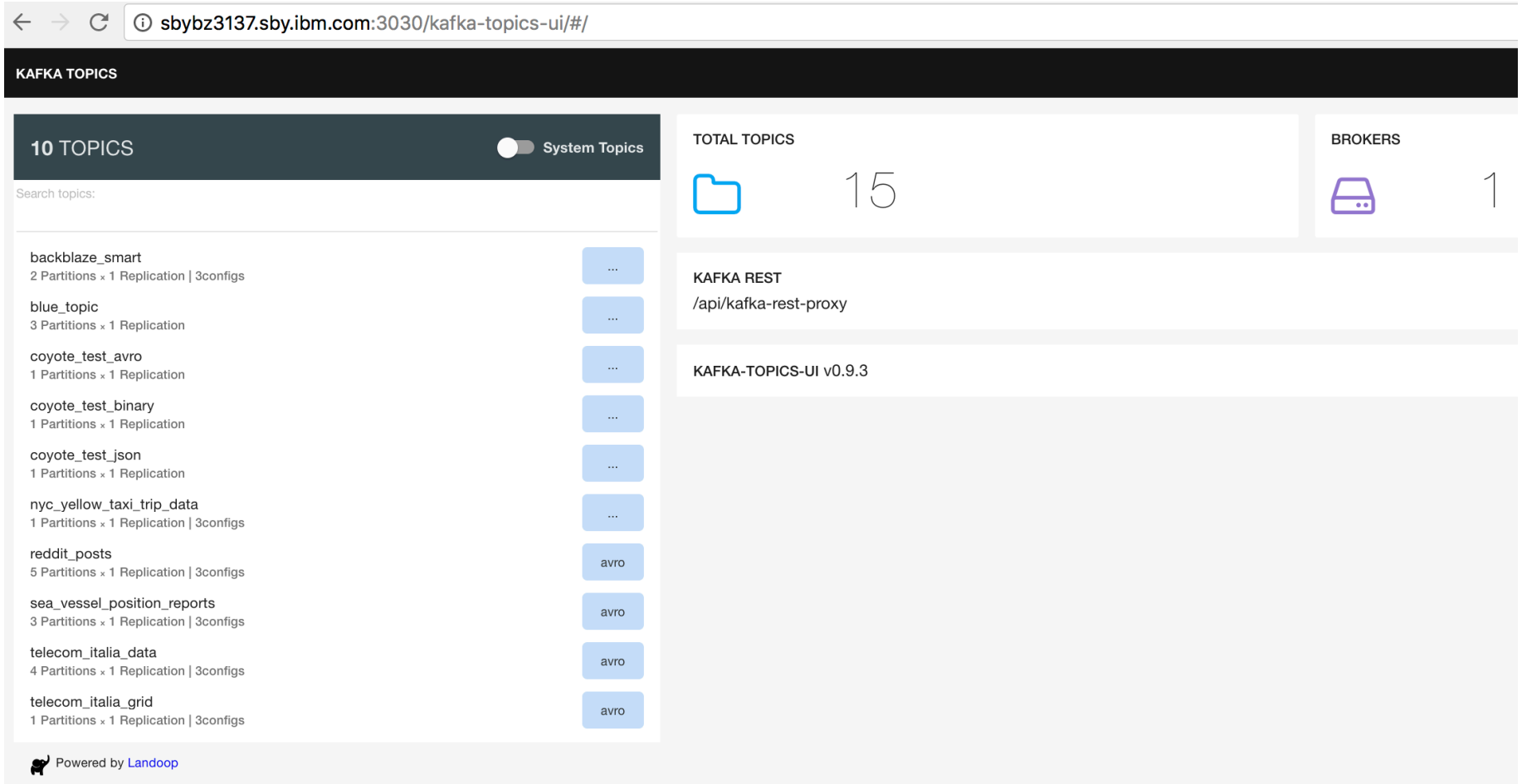
```
[root@sbybz3137 ~]# docker run -it --rm --net=host landoop/fast-data-dev bash
root@fast-data-dev / $ kafka-
kafka-acls                      kafka-console-producer          kafka-preferred-replica-election  kafka-rest-start              kafka-simple-consumer-shell
kafka-avro-console-consumer     kafka-consumer-groups           kafka-producer-perf-test          kafka-rest-stop               kafka-streams-application-reset
kafka-avro-console-producer     kafka-consumer-offset-checker   kafka-reassign-partitions         kafka-rest-stop-service       kafka-topics
kafka-broker-api-versions       kafka-consumer-perf-test        kafka-replay-log-producer         kafka-run-class               kafka-verifiable-consumer
kafka-configs                   kafka-delete-records            kafka-replica-verification        kafka-server-start            kafka-verifiable-producer
kafka-console-consumer          kafka-mirror-maker              kafka-rest-run-class              kafka-server-stop
```

kafka-topics --zookeeper localhost:2181 –list



kafka-topics --zookeeper localhost:2181 --create --topic blue_topic --partitions 3 --replication-factor 1

WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is best to use either, but not both.
Created topic "blue_topic".


sbybz3137.sby.ibm.com:3030/kafka-topics-ui/#/
KAFKA TOPICS
10 TOPICS
System Topics
Search topics:
backblaze_smart
2 Partitions x 1 Replication | 3configs
...
blue_topic
3 Partitions x 1 Replication
...
coyote_test_avro
1 Partitions x 1 Replication
...
coyote_test_binary
1 Partitions x 1 Replication
...
coyote_test_json
1 Partitions x 1 Replication
...
nyc_yellow_taxi_trip_data
1 Partitions x 1 Replication | 3configs
...
reddit_posts
5 Partitions x 1 Replication | 3configs
avro
sea_vessel_position_reports
3 Partitions x 1 Replication | 3configs
avro
telecom_italia_data
4 Partitions x 1 Replication | 3configs
avro
telecom_italia_grid
1 Partitions x 1 Replication | 3configs
avro
Powered by Landoop
TOTAL TOPICS
15
BROKERS
1
KAFKA REST
/api/kafka-rest-proxy
KAFKA-TOPICS-UI v0.9.3

kafka-topics --zookeeper localhost:2181 –list

```
root@fast-data-dev / $ kafka-topics --zookeeper localhost:2181 --list
__consumer_offsets
_schemas
backblaze_smart
blue_topic
connect-configs
connect-offsets
connect-statuses
coyote_test_avro
coyote_test_binary
coyote_test_json
nyc_yellow_taxi_trip_data
reddit_posts
sea_vessel_position_reports
telecom_italia_data
telecom_italia_grid
```

kafka-topics --zookeeper localhost:2181 --describe --topic blue_topic

```
root@fast-data-dev / $ kafka-topics --zookeeper localhost:2181  --describe --topic blue_topic
Topic:blue_topic        PartitionCount:3        ReplicationFactor:1     Configs:
        Topic: blue_topic       Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: blue_topic       Partition: 1    Leader: 0       Replicas: 0     Isr: 0
        Topic: blue_topic       Partition: 2    Leader: 0       Replicas: 0     Isr: 0
```

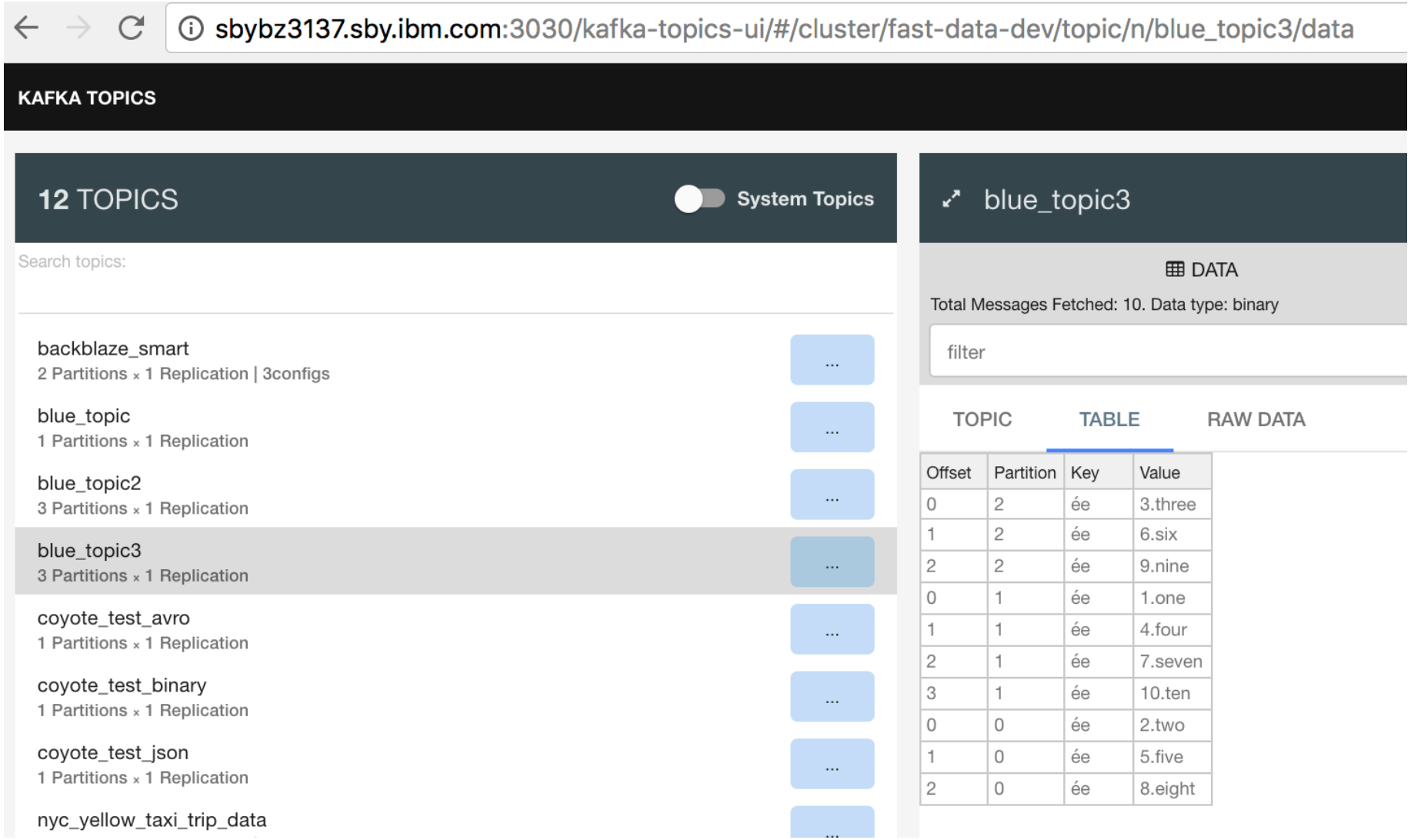
kafka-topics --zookeeper localhost:2181 --delete --topic blue_topic

```
root@fast-data-dev / $ kafka-topics --zookeeper localhost:2181  --delete --topic blue_topic
Topic blue_topic is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
```

Another topic test

kaftopics --zookeeper localhost:2181 --create --topic blue_topic3 --partitions 3 --replication-factor 1
kafka-console-producer --broker-list localhost:9092 --topic blue_topic3

```
>^Croot@fast-data-dev / $ kaftopics --zookeeper localhost:2181  --create --topic blue_topic3 --partitions 3 --replication-factor 1
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is best to use either, but not both.
Created topic "blue_topic3".
root@fast-data-dev / $ kafka-console-producer --broker-list localhost:9092 --topic blue_topic3
>1.one
>2.two
>3.three
>4.four
>5.five
>6.six
>7.seven
>8.eight
>9.nine
>10.ten
```



Offset	Partition	Key	Value
0	2	ée	3.three
1	2	ée	6.six
2	2	ée	9.nine
0	1	ée	1.one
1	1	ée	4.four
2	1	ée	7.seven
3	1	ée	10.ten
0	0	ée	2.two
1	0	ée	5.five
2	0	ée	8.eight

Partitions

Number of Partitions: 3
Replication Factor: 1

Partition	Replica	Replica Broker	Is Leader	Is in-sync
0	0	0	true	true
1	0	0	true	true
2	0	0	true	true

Comparison of the topic data

blue_topic3

DATA

Total Messages Fetched: 10. Data type: binary

filter

TOPIC TABLE RAW DATA

Offset	Partition	Key	Value
0	2	ée	3.three
1	2	ée	6.six
2	2	ée	9.nine
0	1	ée	1.one
1	1	ée	4.four
2	1	ée	7.seven
3	1	ée	10.ten
0	0	ée	2.two
1	0	ée	5.five
2	0	ée	8.eight

blue_topic3

DATA

Total Messages Fetched: 10. Data type: binary

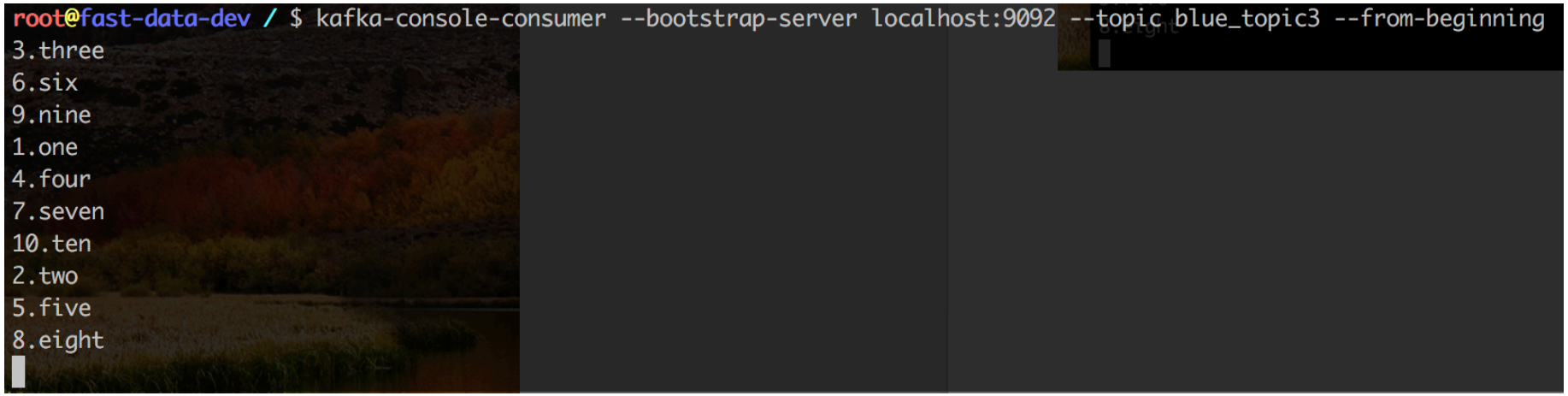
filter

TOPIC TABLE RAW DATA


```
[
  {
    "topic": "blue_topic3",
    "key": "ée",
    "value": "3.three",
    "partition": 2,
    "offset": 0
  },
  {
    "topic": "blue_topic3",
    "key": "ée",
    "value": "6.six",
    "partition": 2,
    "offset": 1
  },
  {
    "topic": "blue_topic3",
    "key": "ée",
    "value": "9.nine",
    "partition": 2,
    "offset": 2
  },
  {
    "topic": "blue_topic3",
    "key": "ée",
    "value": "1.one",
    "partition": 1,
    "offset": 0
  },
  {
    "topic": "blue_topic3",
    "key": "ée",
    "value": "4.four",
    "partition": 1,
    "offset": 1
  },
  {
    "topic": "blue_topic3",
    "key": "ée",
    "value": "7.seven",
    "partition": 1,
    "offset": 2
  },
  {
    "topic": "blue_topic3",
    "key": "ée",
    "value": "10.ten",
    "partition": 1,
    "offset": 3
  },
  {
    "topic": "blue_topic3",
    "key": "ée",
    "value": "2.two",
    "partition": 0,
    "offset": 0
  },
  {
    "topic": "blue_topic3",
    "key": "ée",
    "value": "5.five",
    "partition": 0,
    "offset": 1
  },
  {
    "topic": "blue_topic3",
    "key": "ée",
    "value": "8.eight",
    "partition": 0,
    "offset": 2
  }
]
```

Consume topic

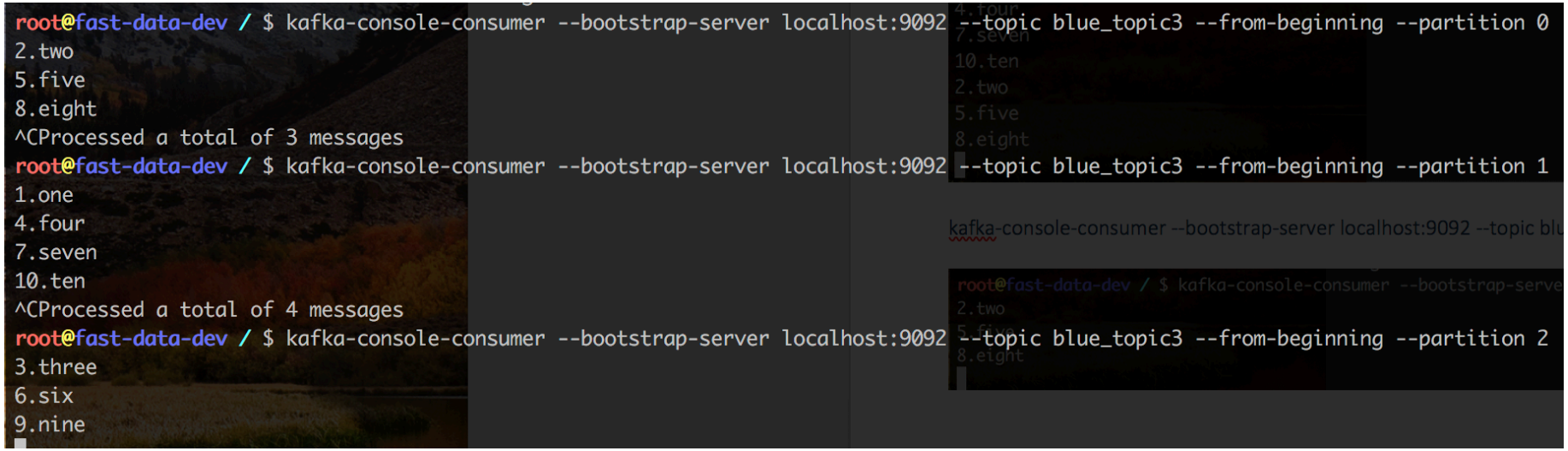
kafka-console-consumer --bootstrap-server localhost:9092 --topic blue_topic3 --from-beginning



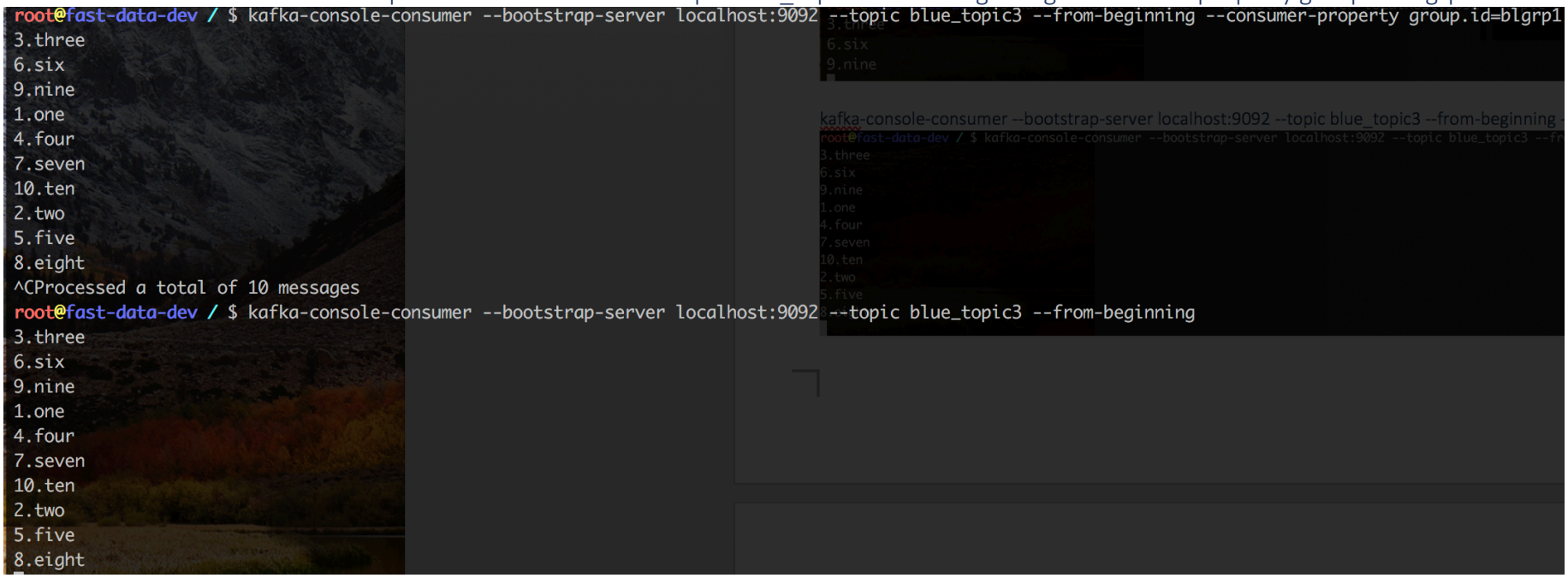
kafka-console-consumer --bootstrap-server localhost:9092 --topic blue_topic3 --from-beginning --partition 0

kafka-console-consumer --bootstrap-server localhost:9092 --topic blue_topic3 --from-beginning --partition 1

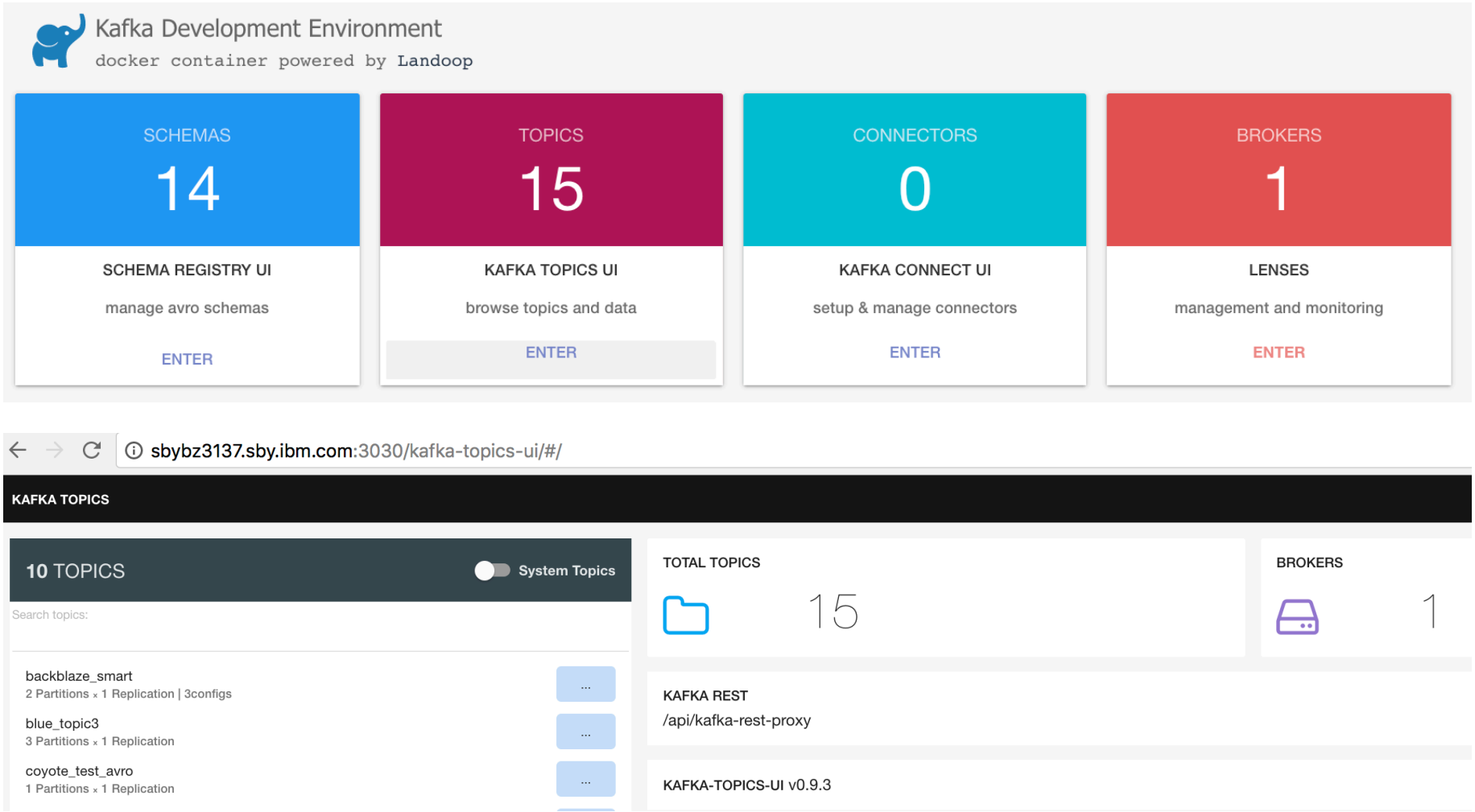
kafka-console-consumer --bootstrap-server localhost:9092 --topic blue_topic3 --from-beginning --partition 2



kafka-console-consumer --bootstrap-server localhost:9092 --topic blue_topic3 --from-beginning --consumer-property group.id=blgrp1



Kafka Topics UI



Writing your own producer

Kafka API

http://kafka.apache.org/documentation/#producerapi

To use the producer, you can use the following maven dependency:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>1.1.0</version>
</dependency>
```

To use the consumer, you can use the following maven dependency:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>1.1.0</version>
</dependency>
```

To use Kafka Streams you can use the following maven dependency:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-streams</artifactId>
    <version>1.1.0</version>
</dependency>
```

Akka Streams

TODO: