Tech Mahindra

Apache Kafka Developer L1

Distributed streaming platform

TLS

Rewards and Recognition

2012 ASTD BEST AWARD WINNER

training TOP 125 Winner - 2013

Tata Institute of Social Sciences CLO Awards India Best Virtual Learning Award

Agenda

- Introduction to Apache Kafka
- Configuring Kafka cluster
- Working with Kafka cluster
    - Single broker
    - Multi broker
- API

Tech Mahindra

Introduction to Apache Kafka

TLS

Rewards and Recognition

2012 ASTD BEST AWARD WINNER

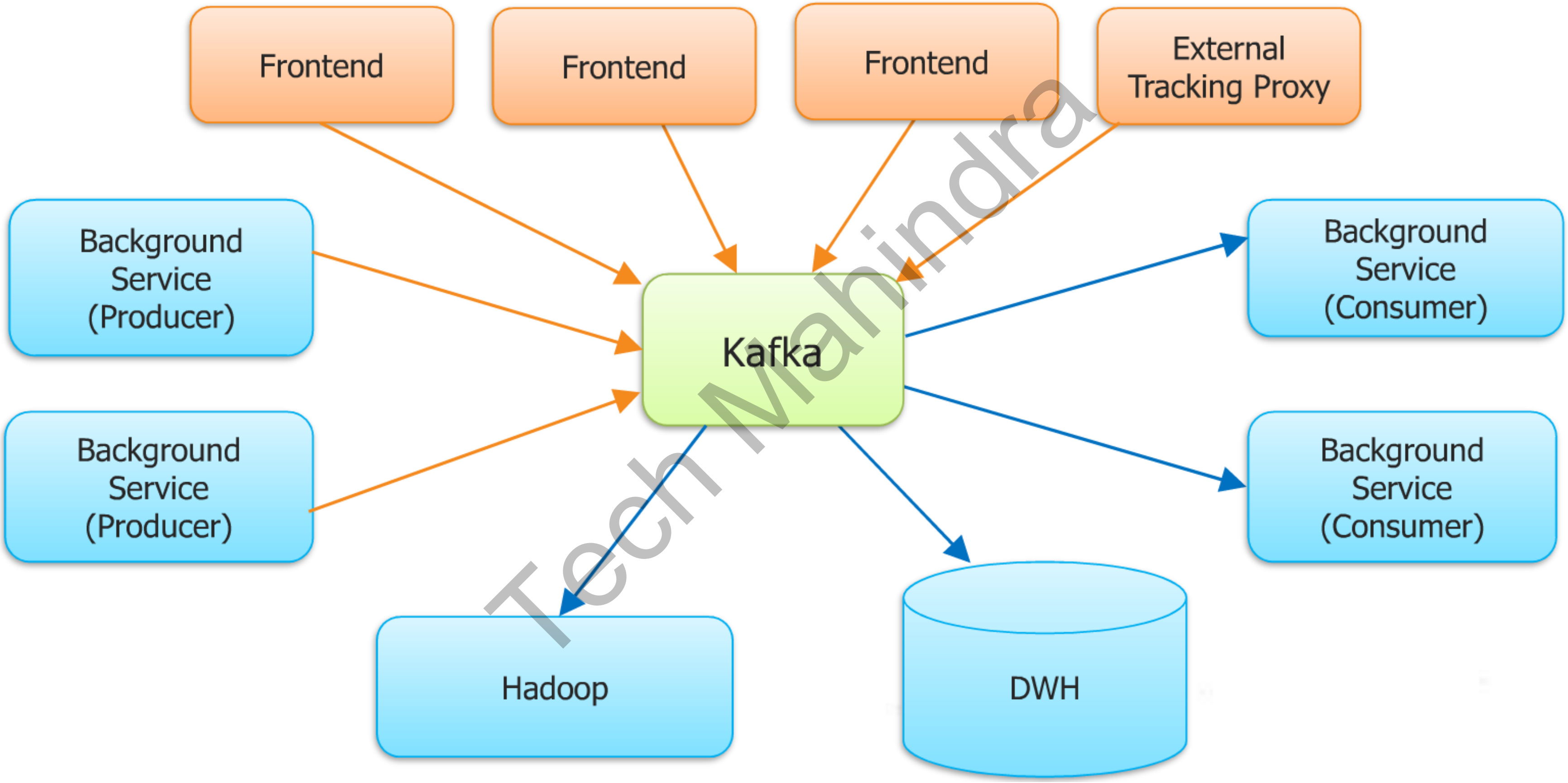
training TOP 125 Winner - 2013

Tata Institute of Social Sciences CLO Awards India Best Virtual Learning Award

Introduction

- **Apache Kafka™ is *a distributed streaming platform*.**
- A distributed publish-subscribe messaging system.
- Developed at LinkedIn Corporation
- Provides solution to handle all activity stream data
- Fully supported in Hadoop platform
- Partitions real time consumption across cluster of machines
- Provides a mechanism for parallel load into Hadoop

Apache Kafka Functional Overview


Frontend
Frontend
Frontend
External Tracking Proxy
Background Service (Producer)
Background Service (Producer)
Kafka
Background Service (Consumer)
Background Service (Consumer)
Hadoop
DWH

Need for Kafka

Feature	Description
High Throughput	Provides support for hundreds of thousands of messages with modest hardware
Scalability	Highly scalable distributed systems with no down time
Replication	Messages can be replicated across clusters, which provides support for multiple subscribers and also in case of failure balances the consumers
Durability	Provides support for persistence of messages to disk which can be further used for batch consumption
Stream Processing	Can be used along with real time streaming applications like Spark and Storm
Data Loss	Can ensure Zero Data loss with proper configurations

Kafka Core APIs

Producer API

- Allows an application to publish a stream of records to one or more Kafka topics.

Consumer API

- Allows an application to subscribe to one or more topics and process the stream of records produced to them.

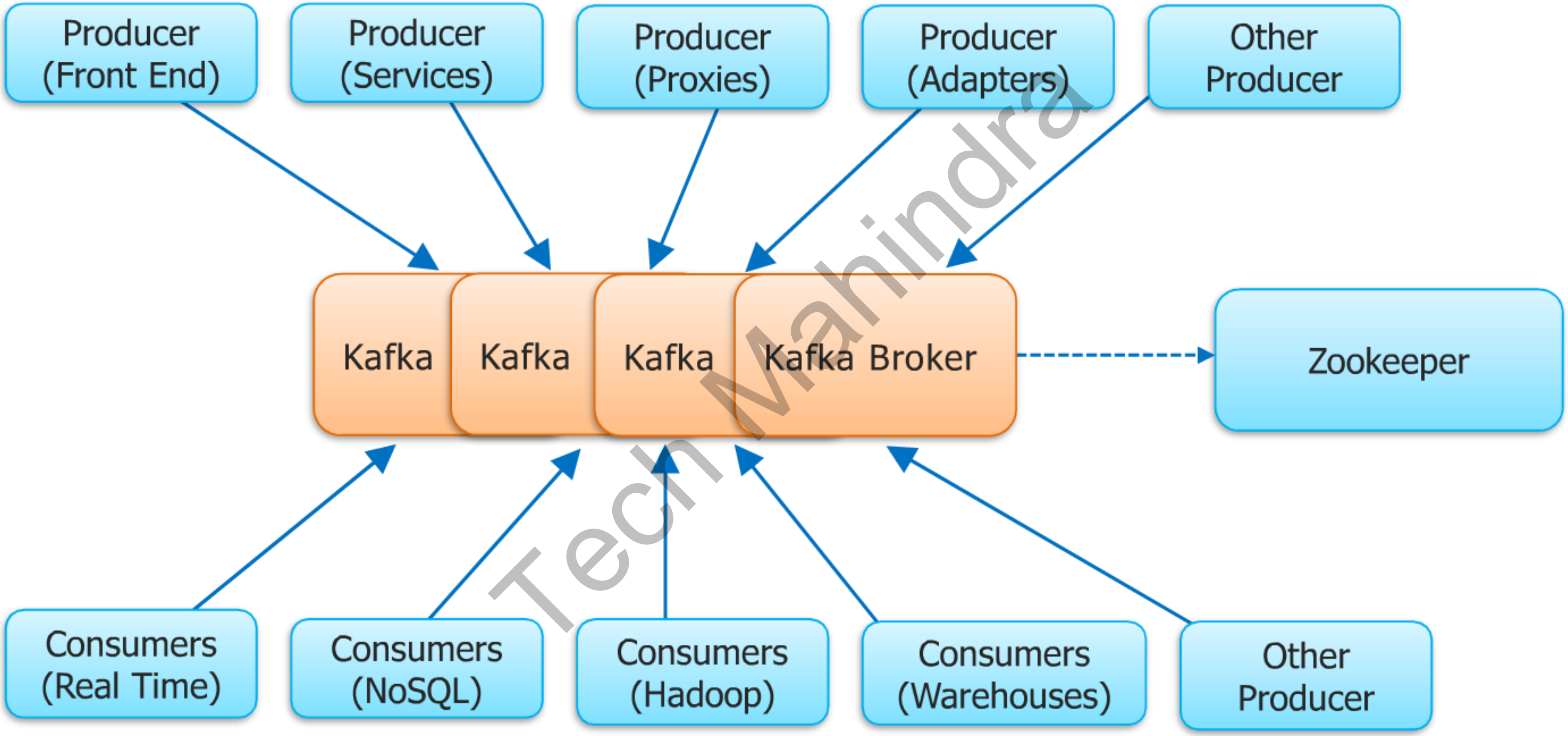
Streams API

- 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.

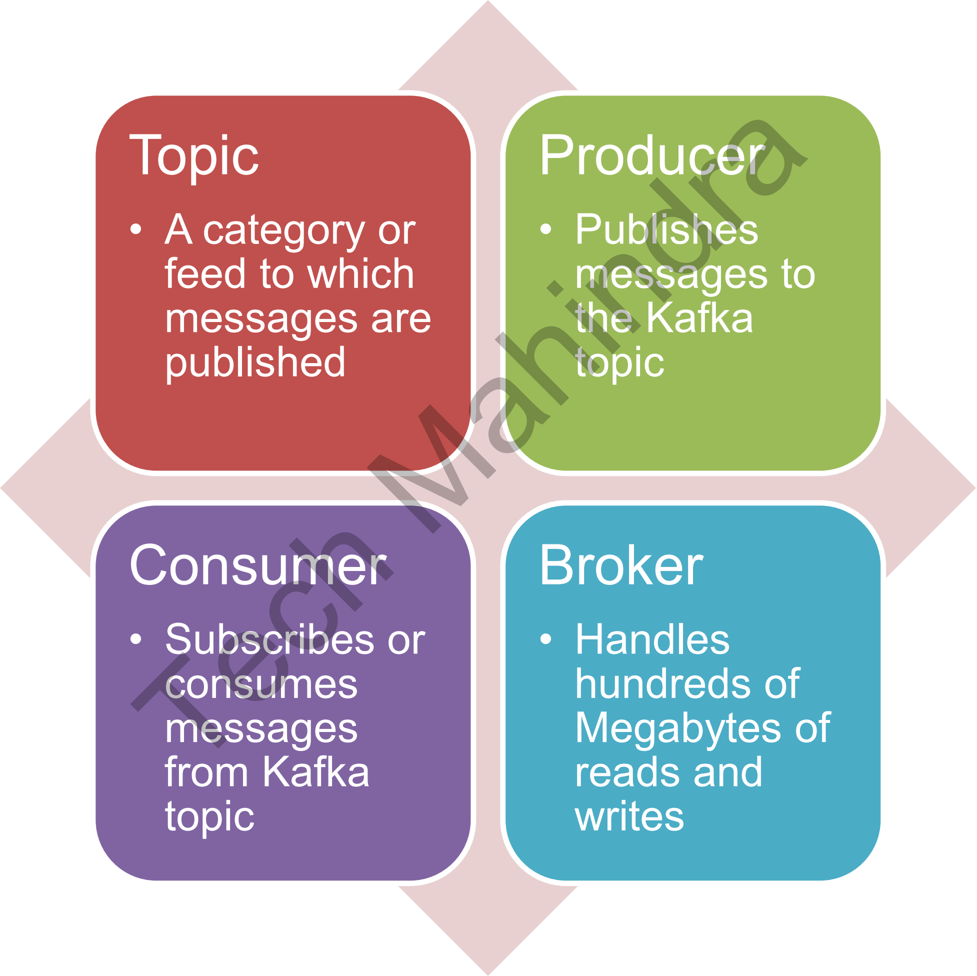
Connector API

- 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.

Kafka Architecture



Kafka Architecture Core Concepts



Topic

- A category or feed to which messages are published

Producer

- Publishes messages to the Kafka topic

Consumer

- Subscribes or consumes messages from Kafka topic

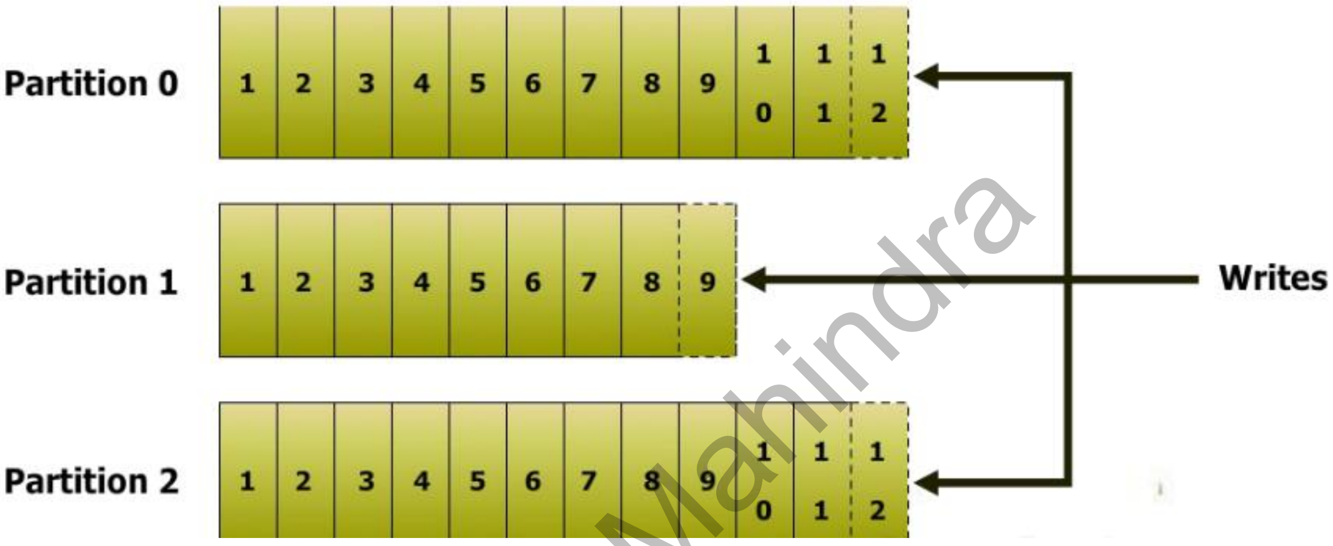
Broker

- Handles hundreds of Megabytes of reads and writes

Topic

- A topic is a category or feed name to which records are published.
- Topics in Kafka are always multi-subscriber; that is, a topic can have zero, one, or many consumers that subscribe to the data written to it.
- For each topic, the Kafka cluster maintains a partitioned log.
- Each partition basically contains an ordered, immutable sequence of messages where each message assigned a sequential ID number called offset.
- Writes to a partition are generally sequential thereby reducing the number of hard disk seeks
- Reading messages from partition can either be from the beginning and also can rewind or skip to any point in a partition by supplying an offset values

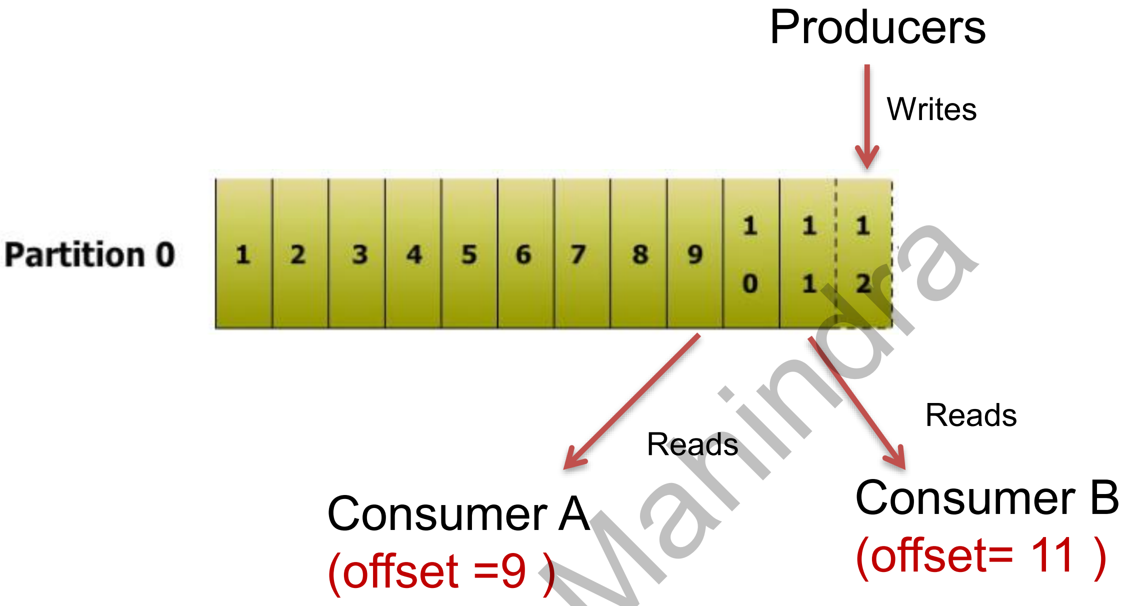
Topic



The Kafka cluster retains all published records—whether or not they have been consumed—using a configurable retention period.

For example, if the retention policy is set to two days, then for the two days after a record is published, it is available for consumption, after which it will be discarded to free up space.

Topic



Producers

Writes

Partition 0

Reads

Reads

Consumer A
(offset =9)

Consumer B
(offset= 11)

- Only metadata retained on a per-consumer basis is the offset or position of that consumer in the log. This offset is controlled by the consumer.
- Normally a consumer will advance its offset linearly as it reads records, but can consume records in any order it likes.
- For example a consumer can reset to an older offset to reprocess data from the past or skip ahead to the most recent record and start consuming from "now".

Partitions

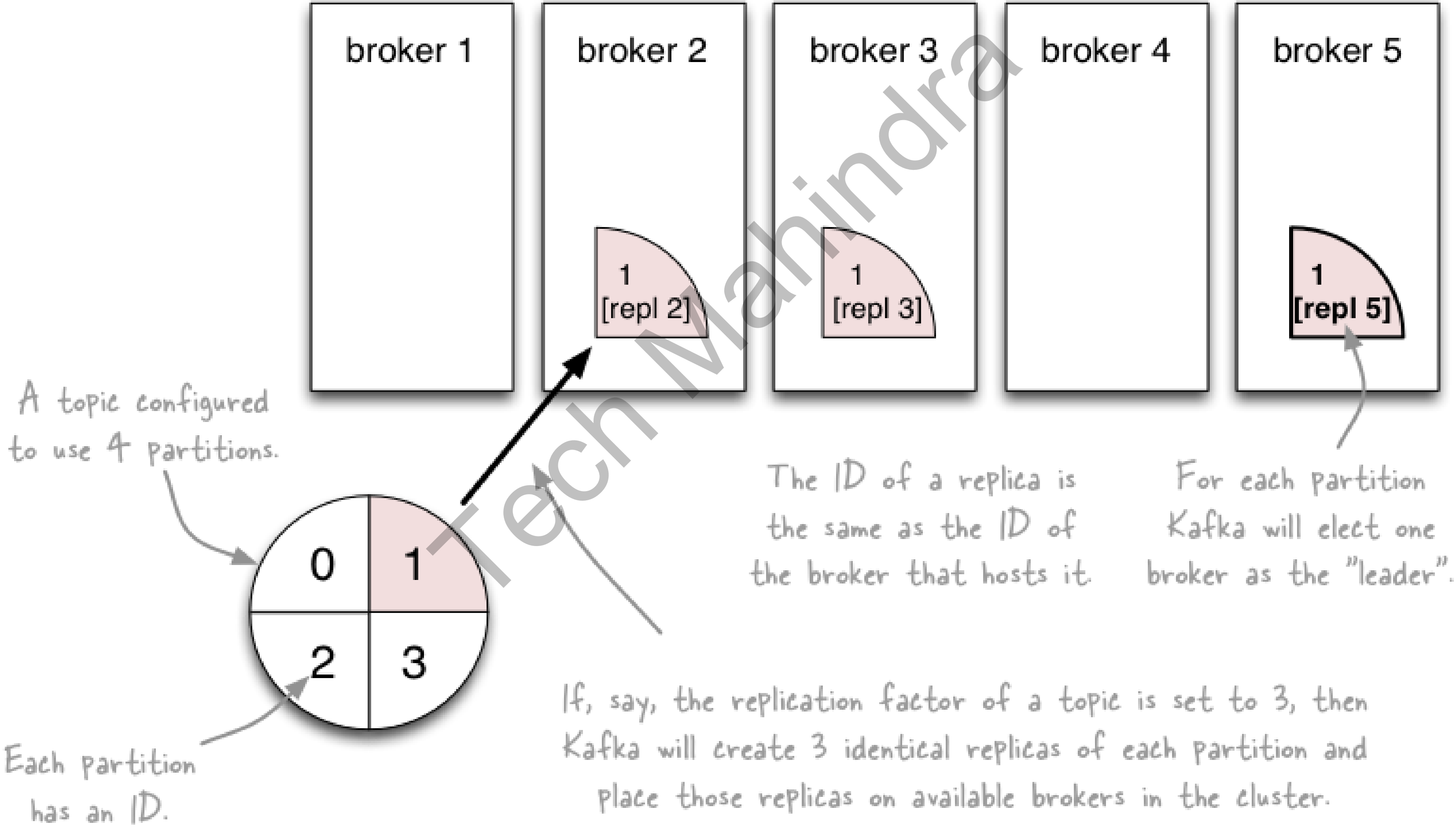
- The partitions in the log allow the log to scale beyond a size that will fit on a single server.
- Each individual partition must fit on the servers that host it.
- A topic may have many partitions so it can handle an arbitrary amount of data.
- Partitions act as the unit of parallelism.
- The partitions of the log are distributed over the servers in the Kafka cluster with each server handling data and requests for a share of the partitions.
- Each partition is replicated across a configurable number of servers for fault tolerance.

Partitions

- Each partition has one server which acts as the "leader" and zero or more servers which act as "followers".
- The leader handles all read and write requests for the partition while the followers passively replicate the leader. If the leader fails, one of the followers will automatically become the new leader.
- Each server acts as a leader for some of its partitions and a follower for others so load is well balanced within the cluster.

Topics, Partitions & Replicas

- The relationship between topics, partitions and replicas in Kafka



A topic configured to use 4 partitions.

Each partition has an ID.

The ID of a replica is the same as the ID of the broker that hosts it.

For each partition Kafka will elect one broker as the "leader".

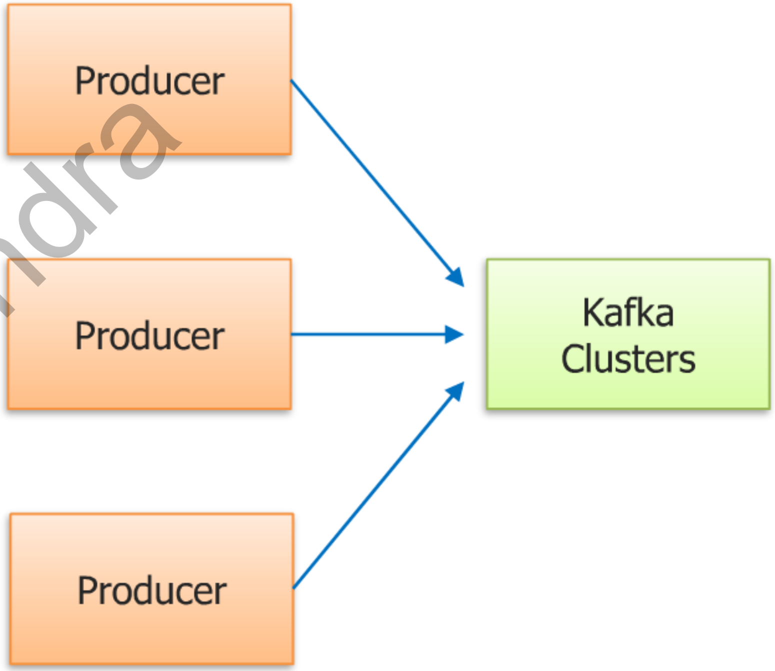
If, say, the replication factor of a topic is set to 3, then Kafka will create 3 identical replicas of each partition and place those replicas on available brokers in the cluster.

Topics, Partitions & Replicas

- The relationship between topics, partitions and replicas in Kafka
- When a topic is created in Kafka, it determines how each replica of a partition is mapped to a broker Kafka tries to spread the replicas across all brokers
- Messages are first sent to the first replica of a partition (i.e. to the current “leader” broker of that partition) before they are replicated to the remaining brokers
- Message producers may choose from different strategies for sending messages (e.g. synchronous mode, asynchronous mode)
- Producers discover the available brokers in a cluster and the number of partitions on each, by registering consumers in ZooKeeper

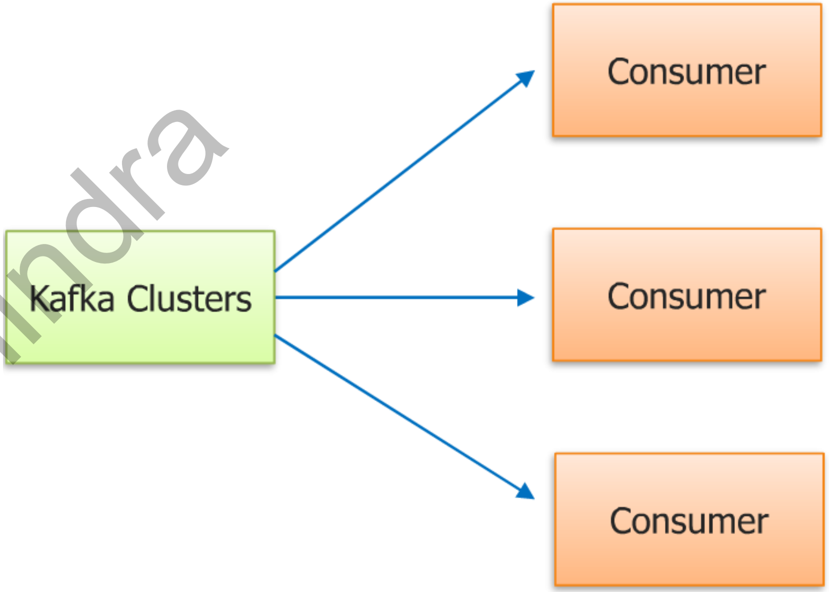
Producers

- Applications publishes messages to the topic in Kafka cluster.
- Can be of any kind like Front end, streaming etc.
- While writing messages , it is also possible to attach a key with the message.
- By attaching key the producers basically provide a guaranty that all messages with the same key will arrive in the same partition.
- Supports both async and sync modes.
- Publishes as much messages as fast as the broker in a cluster can handle.

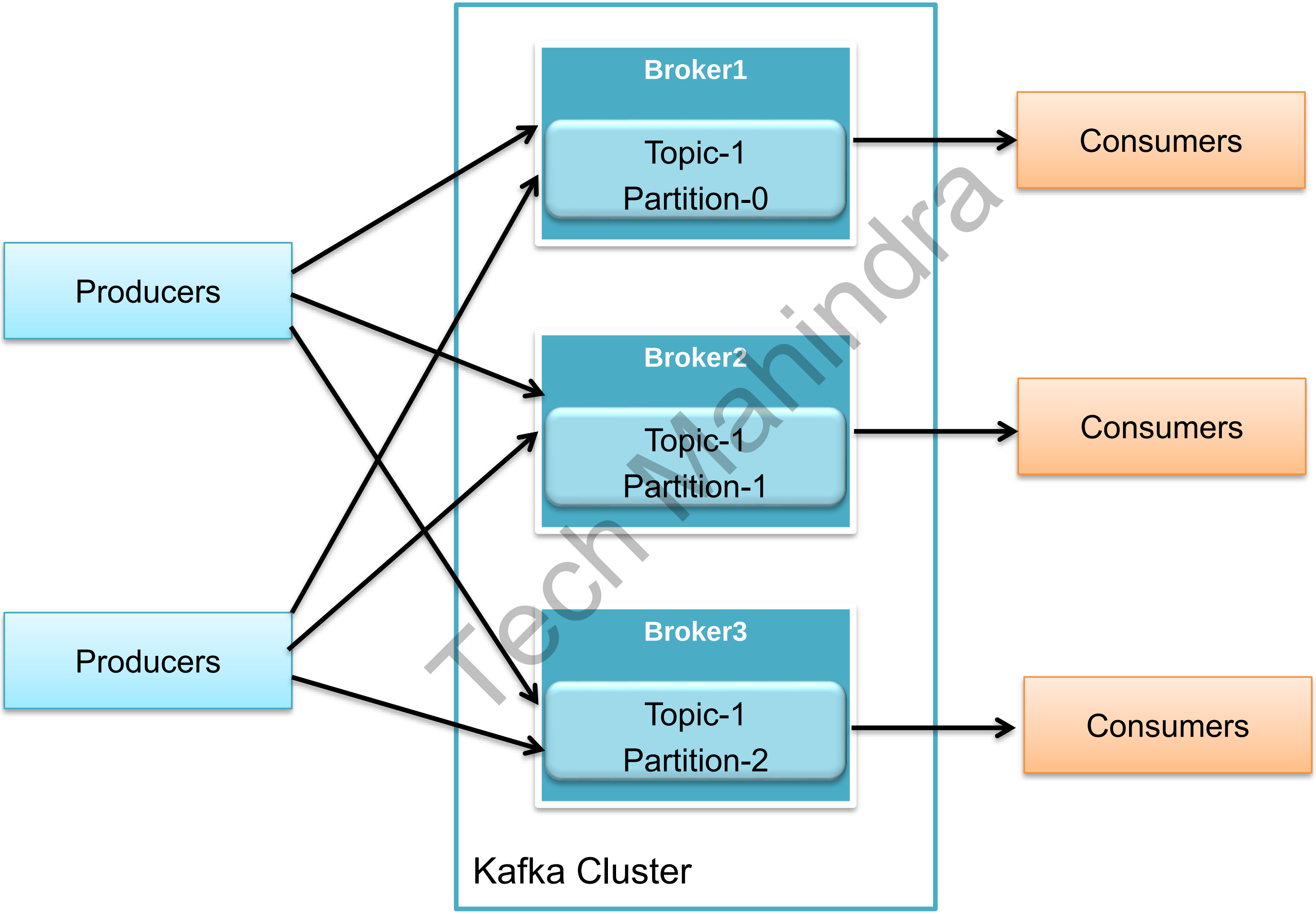


Consumers

- Applications subscribes and consumes messages from brokers in Kafka cluster.
- Can be any kind like real time consumers, NoSQL consumers etc.
- During consumption of messages from a topic a consumer group can be configured with multiple consumers.
- Each consumer of consumer group reads messages from a unique sub set of partitions in each topic they subscribe to
- Messages with same key arrives at same consumer
- Supports both Queuing and Publish-Subscribe
- Consumers have to maintain the number of messages consumed.



Kafka Brokers



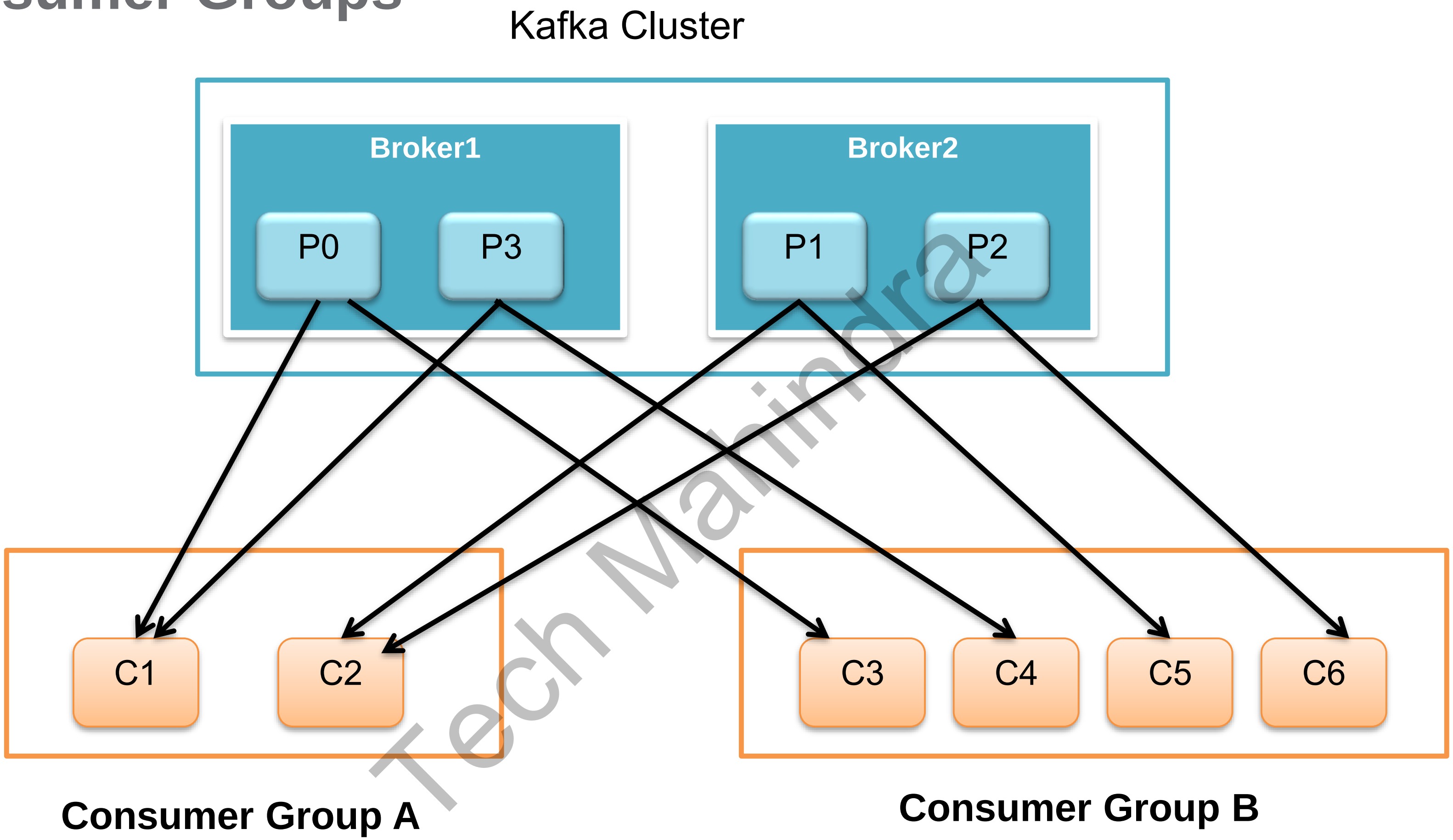
Kafka Brokers

- Kafka cluster basically comprised of one or more servers.
- Each of the servers in the cluster is called a broker.
- Handles hundreds of megabytes of writes from producers and reads from consumers.
- Retains all published messages irrespective of weather it is consumed or not.
- If retention is configured for ‘n’ days, then messages once published is available for consumptions for that days thereafter it is discarded.
- Works like a queue if consumer instances belong to same consumer group else works like publish-subscribe

Consumer Groups

- Consumers label themselves with a *consumer group* name, and each record published to a topic is delivered to one consumer instance within each subscribing consumer group.
- Consumer instances can be in separate processes or on separate machines.
- If all the consumer instances have the same consumer group, then the records will effectively be load balanced over the consumer instances.
- If all the consumer instances have different consumer groups, then each record will be broadcast to all the consumer processes.

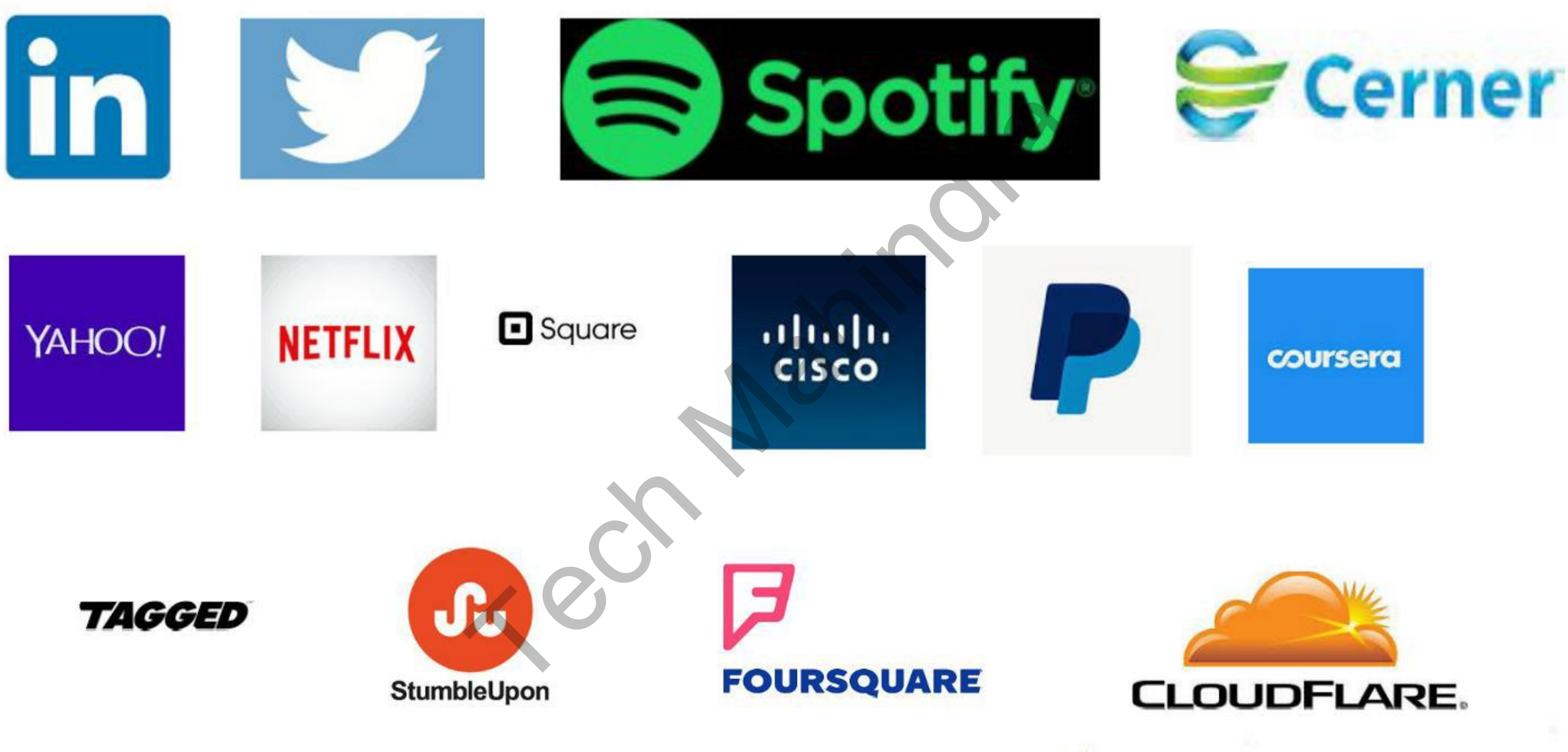
Consumer Groups



Consumer Groups

- The way consumption is implemented in Kafka is by dividing up the partitions in the log over the consumer instances so that each instance is the exclusive consumer of a "fair share" of partitions at any point in time.
- This process of maintaining membership in the group is handled by the Kafka protocol dynamically.
- If new instances join the group they will take over some partitions from other members of the group
- If an instance dies, its partitions will be distributed to the remaining instances.

Who Uses Kafka

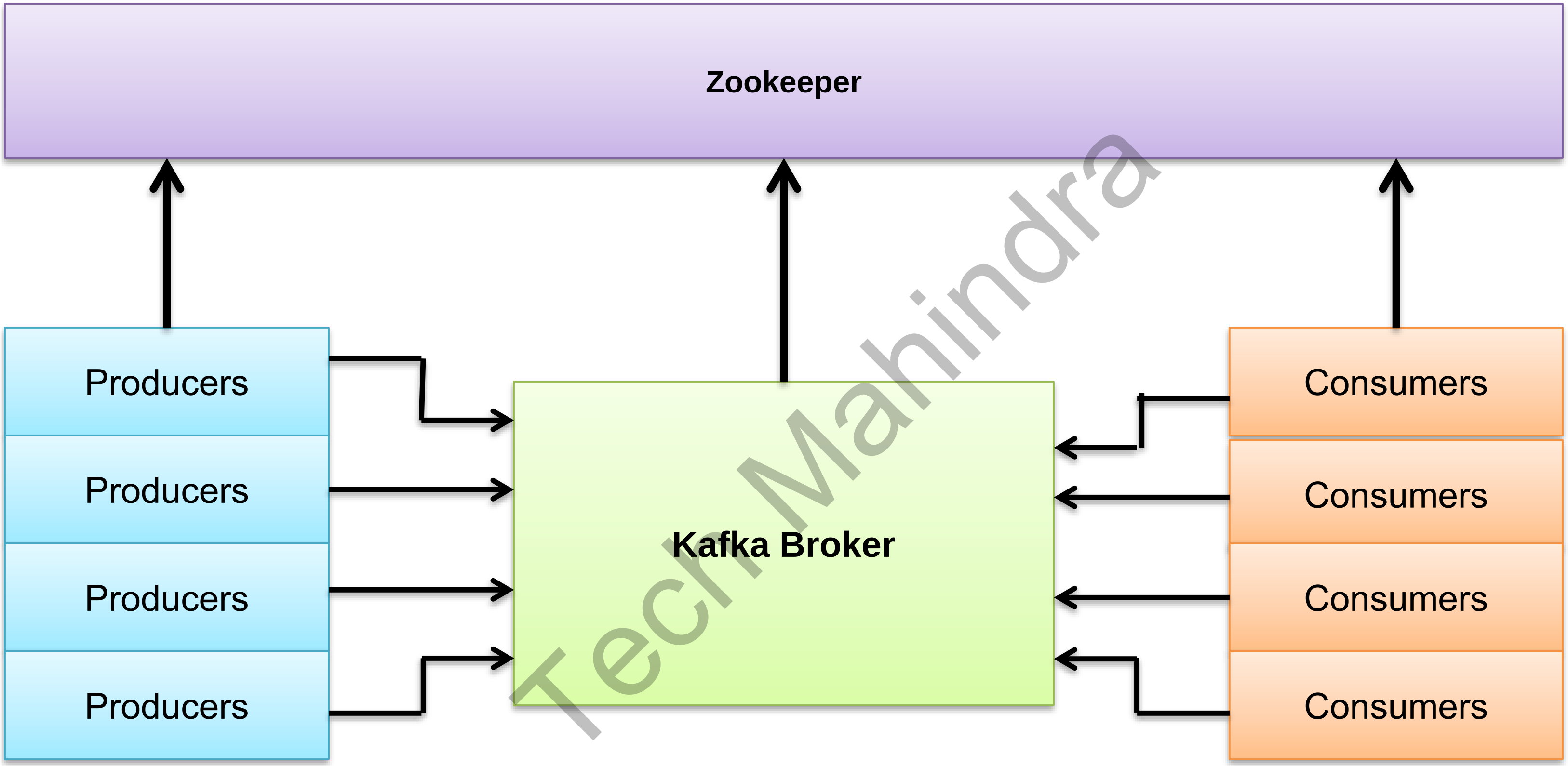


Setting up Kafka Cluster

Kafka Cluster

- A Kafka Cluster is generally fast, highly scalable messaging system.
- A publish-subscribe messaging system
- Can be used effectively in place of Java messaging system and advance messaging queuing protocol.
- Can be integrated with Hadoop Ecosystem
- Expanding of the cluster can be done with ease
- Effective for applications which involves large scale message processing.
- Easily handle hundreds of thousands of messages in second which makes it a high throughput messaging system.
- The cluster can be expanded with no downtime, making kafka highly scalable
- Messages are replicated which provides reliability and durability
- Fault tolerant

Kafka Cluster



Configurations

- The essential configurations are the following:

broker.id

log.dir

zookeeper.connect

Property	Description	default
zookeeper.connect	Zookeeper host string	
broker.id	The broker id for this server. If unset, a unique broker id will be generated. To avoid conflicts between zookeeper generated broker id's and user configured broker id's, generated broker ids start from reserved.broker.max.id + 1.	
Log.dir	The directory in which the log data is kept	/tmp/kafka-logs

https://kafka.apache.org/documentation/#configuration

Kafka Cluster Setup

Single broker setup

Multiple broker setup

Kafka Single Broker Setup

Kafka-Multi Broker Setup

Testing Kafka Cluster

- To test the kafka cluster following scripts

Zookeeper-server-start.sh

Kafka-server-start.sh

Kafka-topics.sh

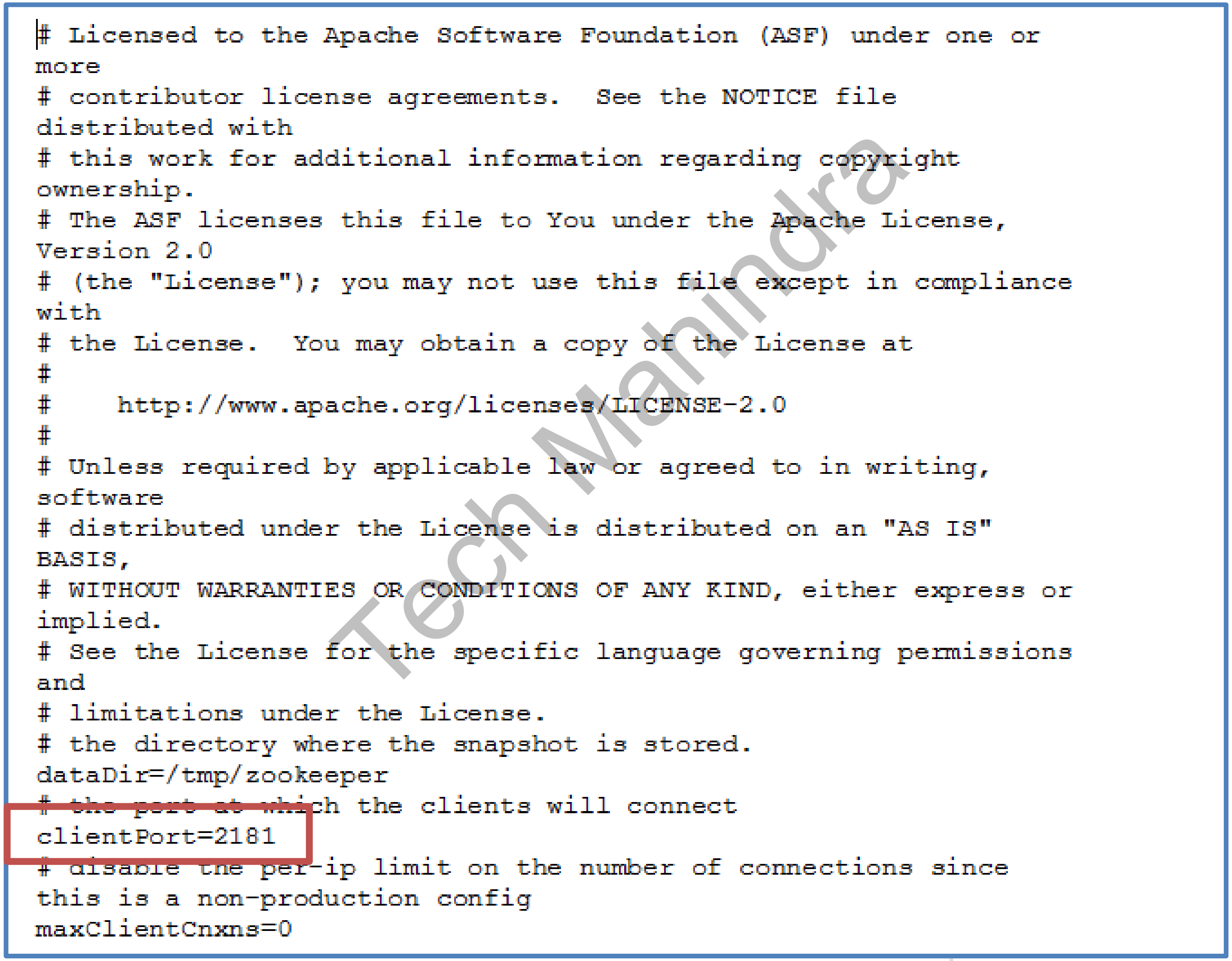
Kafka-console-producer.sh

Kafka-console-consumer.sh

Testing Kafka Cluster

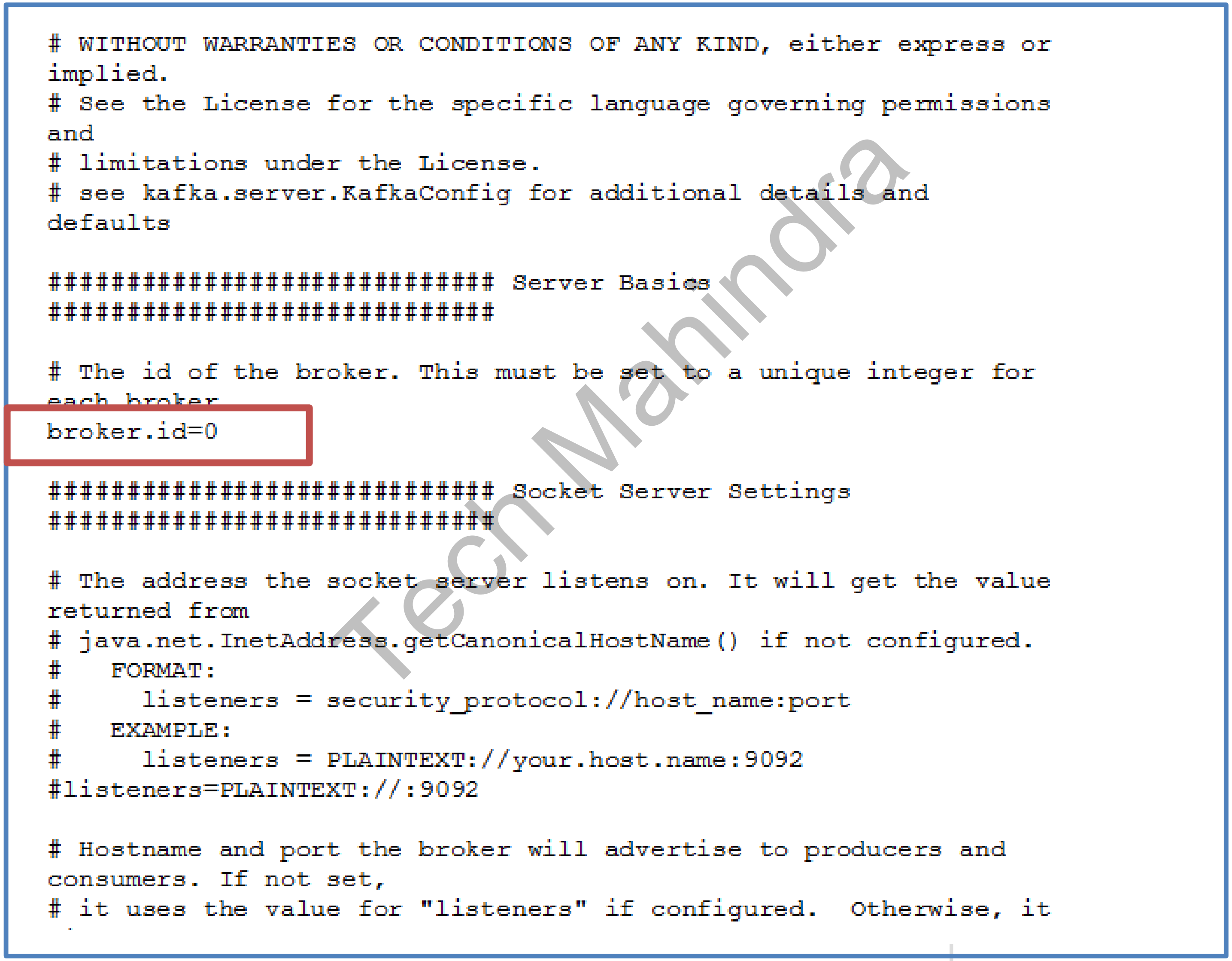
- Zookeeper-server-start.sh
  - Starts the zookeeper with the properties configured under config/zookeeper.properties
- Kafka-server-start.sh
  - Starts the zookeeper with the properties configured under config/server.properties
- Kafka-topics.sh
  - Used to create and list topics
- Kafka-console-producer.sh
  - Command line client to send messages to kafka cluster
- Kafka-console-consumer.sh
  - Command line client to consume messages from kafka cluster

Zookeeper. Properties File



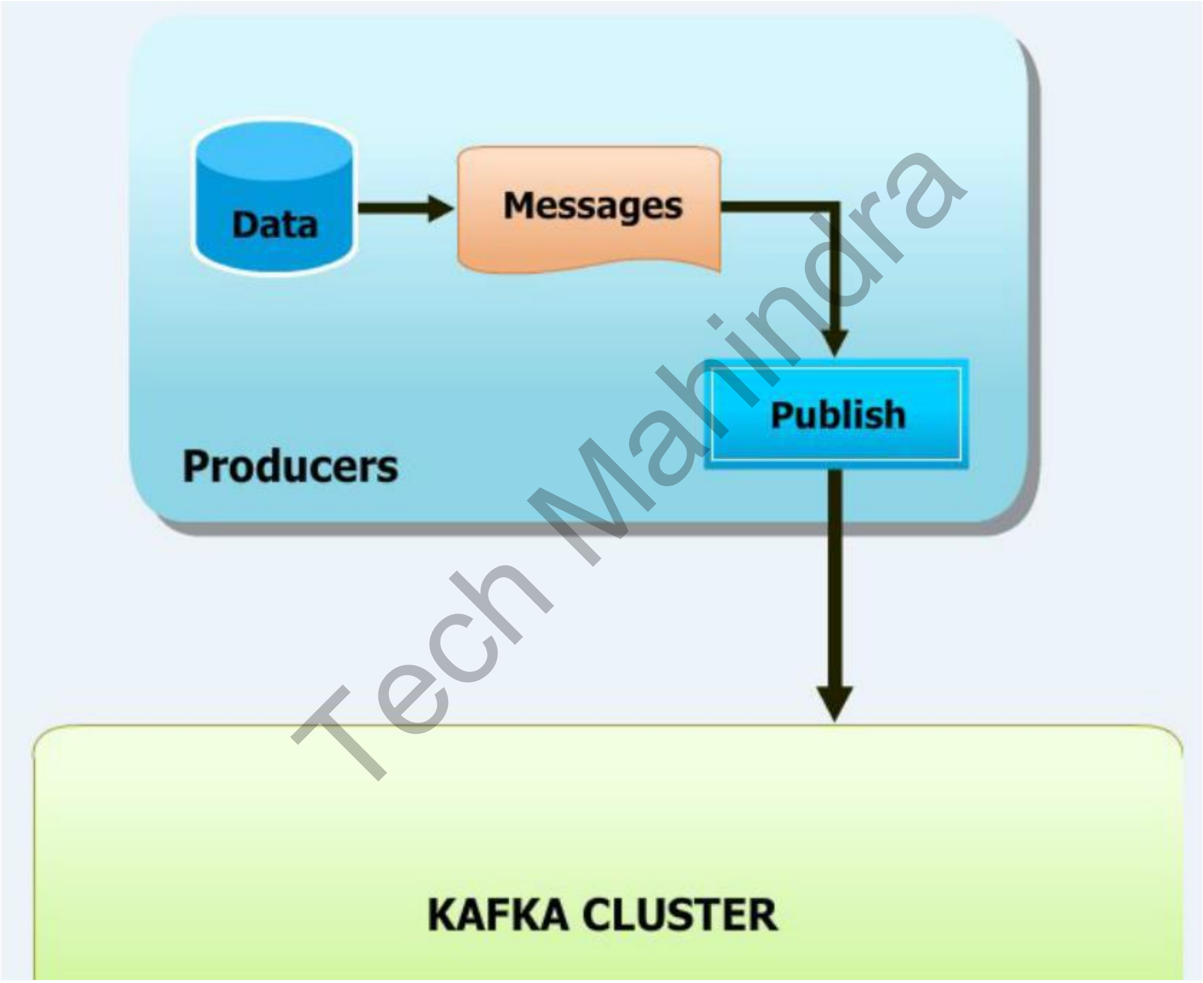
```
# Licensed to the Apache Software Foundation (ASF) under one or
more
# contributor license agreements.  See the NOTICE file
distributed with
# this work for additional information regarding copyright
ownership.
# The ASF licenses this file to You under the Apache License,
Version 2.0
# (the "License"); you may not use this file except in compliance
with
# the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
software
# distributed under the License is distributed on an "AS IS"
BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
# See the License for the specific language governing permissions
and
# limitations under the License.
# the directory where the snapshot is stored.
dataDir=/tmp/zookeeper
# the port at which the clients will connect
clientPort=2181
# disable the per-ip limit on the number of connections since
this is a non-production config
maxClientCnxns=0
```

Server. Properties File



```
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
# See the License for the specific language governing permissions
and
# limitations under the License.
# see kafka.server.KafkaConfig for additional details and
defaults

############################# Server Basics
#############################

# The id of the broker. This must be set to a unique integer for
each broker.
broker.id=0

############################# Socket Server Settings
#############################

# The address the socket server listens on. It will get the value
returned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#     listeners = security_protocol://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
#listeners=PLAINTEXT://:9092

# Hostname and port the broker will advertise to producers and
consumers. If not set,
# it uses the value for "listeners" if configured.  Otherwise, it
```

Kafka Producer - broker

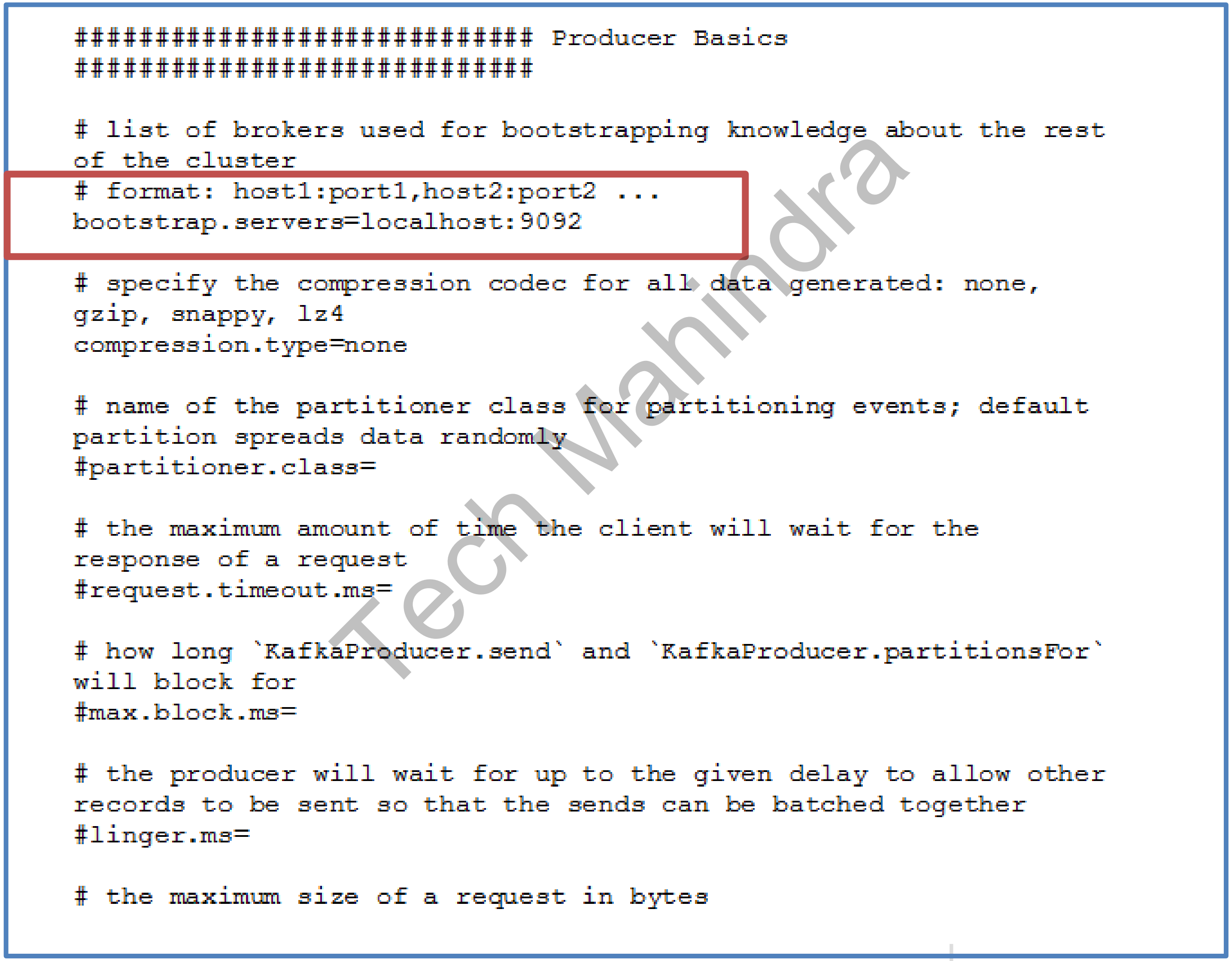


Producer and Consumer Configuration

Property	Description
bootstrap.servers	A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. This list should be in the form host1:port1,host2:port2,....
acks	The number of acknowledgments the producer requires the leader to have received before considering a request complete. This controls the durability of records that are sent. If set to zero then the producer will not wait for any acknowledgment from the server at all. If set to 1 mean the leader will write the record to its local log but will respond without awaiting full acknowledgement from all followers. If set to all means the leader will wait for the full set of in-sync replicas to acknowledge the record.
compression.type	The compression type for all data generated by the producer. The default is none (i.e. no compression). Valid values are none, gzip, snappy, or lz4
batch.size	The producer will attempt to batch records together into fewer requests whenever multiple records are being sent to the same partition. This helps performance on both the client and the server.

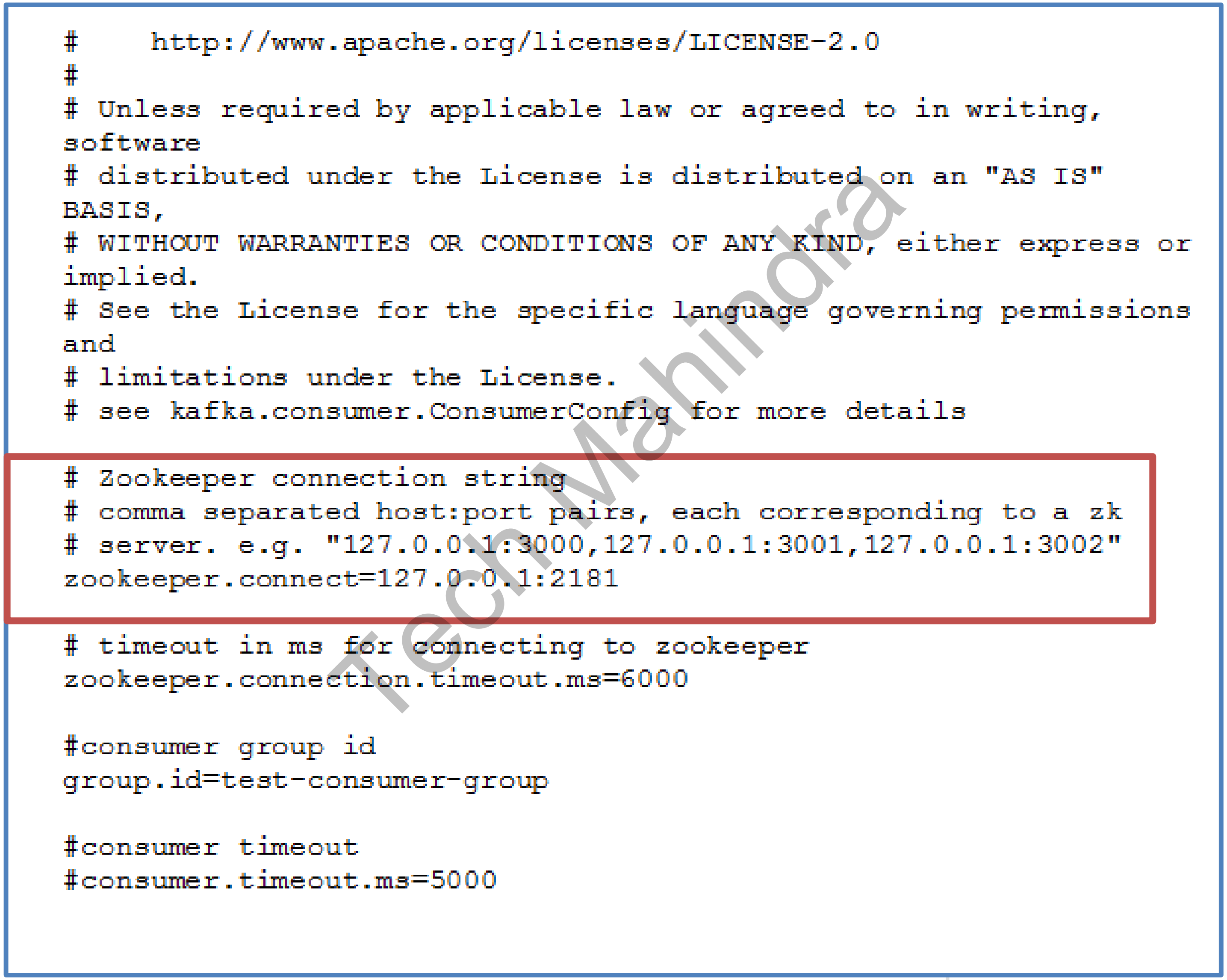
Property	Description
bootstrap.servers	A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. This list should be in the form host1:port1,host2:port2,....
fetch.min.bytes	The minimum amount of data the server should return for a fetch request. The default setting of 1 byte means that fetch requests are answered as soon as a single byte of data is available or the fetch request times out waiting for data to arrive.
group.id	A unique string that identifies the consumer group this consumer belongs to. This property is required if the consumer uses either the group management functionality by using subscribe(topic)
zookeeper.connect	Specifies the ZooKeeper connection string in the form hostname:port where host and port are the host and port of a ZooKeeper server. To allow connecting through other ZooKeeper nodes when that ZooKeeper machine is down you can also specify multiple hosts in the form hostname1:port1,hostname2:port2,hostname3:port3

Producer. Properties File



```
############################# Producer Basics
#############################

# list of brokers used for bootstrapping knowledge about the rest
of the cluster
# format: host1:port1,host2:port2 ...
bootstrap.servers=localhost:9092

# specify the compression codec for all data generated: none,
gzip, snappy, lz4
compression.type=none

# name of the partitioner class for partitioning events; default
partition spreads data randomly
#partitioner.class=

# the maximum amount of time the client will wait for the
response of a request
#request.timeout.ms=

# how long `KafkaProducer.send` and `KafkaProducer.partitionsFor`
will block for
#max.block.ms=

# the producer will wait for up to the given delay to allow other
records to be sent so that the sends can be batched together
#linger.ms=

# the maximum size of a request in bytes
```

Consumer. Properties File



```
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
software
# distributed under the License is distributed on an "AS IS"
BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
# See the License for the specific language governing permissions
and
# limitations under the License.
# see kafka.consumer.ConsumerConfig for more details

# Zookeeper connection string
# comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002"
zookeeper.connect=127.0.0.1:2181

# timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=6000

#consumer group id
group.id=test-consumer-group

#consumer timeout
#consumer.timeout.ms=5000
```

Kafka Operations

TLS

Rewards and Recognition

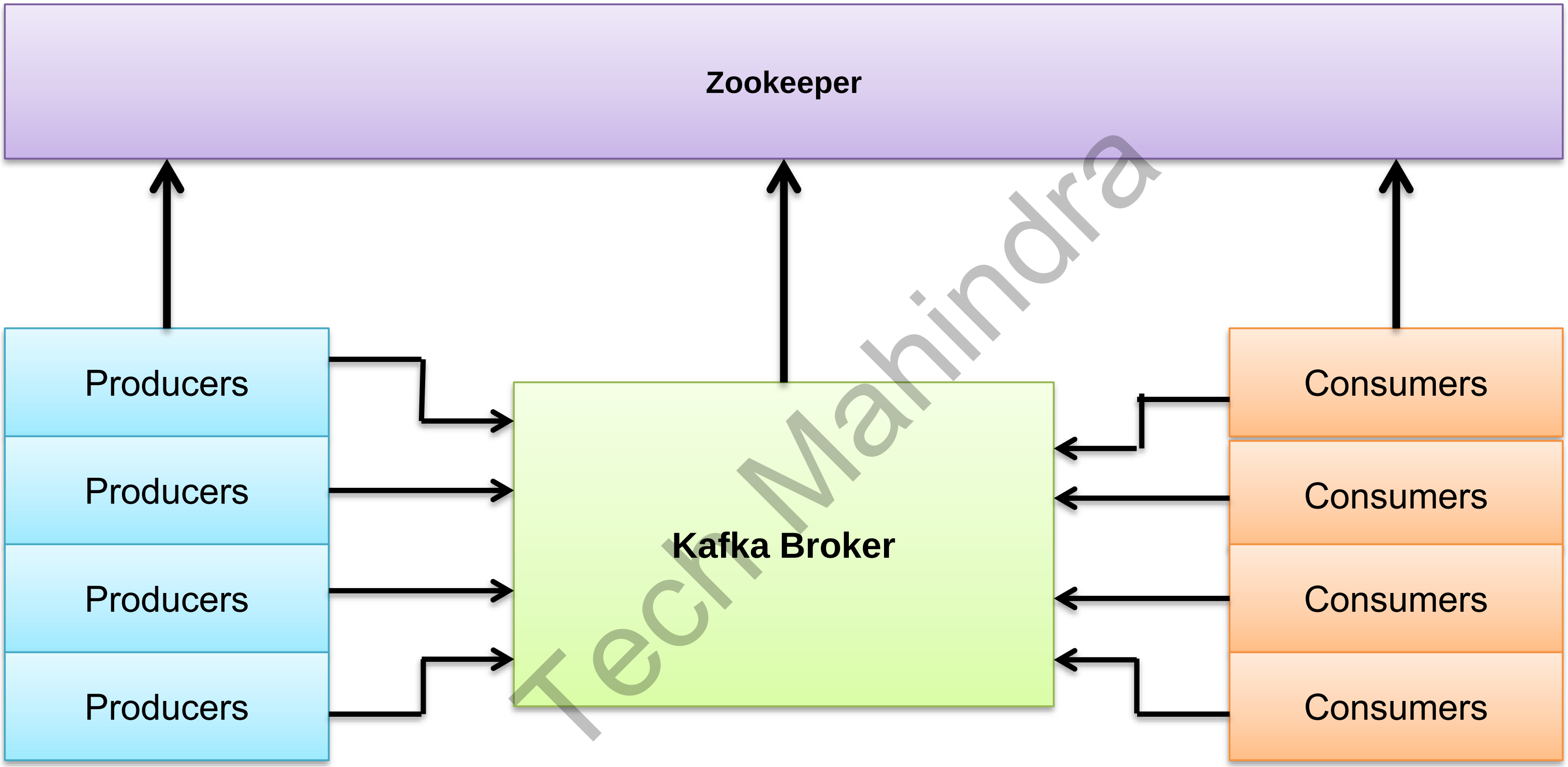
2012 ASTD BEST AWARD WINNER

training TOP 125 Winner - 2013

Tata Institute of Social Sciences CLO Awards India Best Virtual Learning Award

Single Broker Cluster

Single Broker Cluster



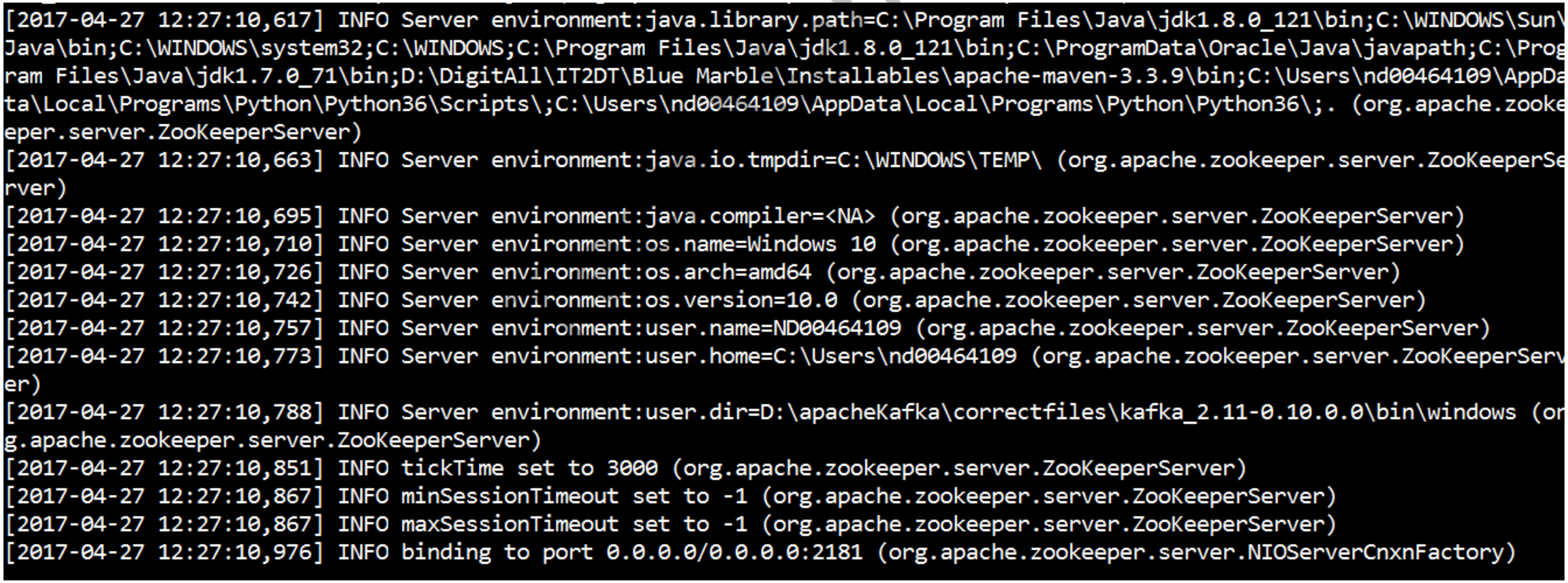
Starting Zookeeper

Linux

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

Windows

Bin/windows/zookeeper-server-start.bat config/zookeeper.properties



```
[2017-04-27 12:27:10,617] INFO Server environment:java.library.path=C:\Program Files\Java\jdk1.8.0_121\bin;C:\WINDOWS\Sun\
Java\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Java\jdk1.8.0_121\bin;C:\ProgramData\Oracle\Java\javapath;C:\Prog
ram Files\Java\jdk1.7.0_71\bin;D:\DigitAll\IT2DT\Blue Marble\Installables\apache-maven-3.3.9\bin;C:\Users\nd00464109\AppDa
ta\Local\Programs\Python\Python36\Scripts\;C:\Users\nd00464109\AppData\Local\Programs\Python\Python36\;. (org.apache.zooke
eper.server.ZooKeeperServer)
[2017-04-27 12:27:10,663] INFO Server environment:java.io.tmpdir=C:\WINDOWS\TEMP\ (org.apache.zookeeper.server.ZooKeeperSe
rver)
[2017-04-27 12:27:10,695] INFO Server environment:java.compiler=<NA> (org.apache.zookeeper.server.ZooKeeperServer)
[2017-04-27 12:27:10,710] INFO Server environment:os.name=Windows 10 (org.apache.zookeeper.server.ZooKeeperServer)
[2017-04-27 12:27:10,726] INFO Server environment:os.arch=amd64 (org.apache.zookeeper.server.ZooKeeperServer)
[2017-04-27 12:27:10,742] INFO Server environment:os.version=10.0 (org.apache.zookeeper.server.ZooKeeperServer)
[2017-04-27 12:27:10,757] INFO Server environment:user.name=ND00464109 (org.apache.zookeeper.server.ZooKeeperServer)
[2017-04-27 12:27:10,773] INFO Server environment:user.home=C:\Users\nd00464109 (org.apache.zookeeper.server.ZooKeeperServ
er)
[2017-04-27 12:27:10,788] INFO Server environment:user.dir=D:\apacheKafka\correctfiles\kafka_2.11-0.10.0.0\bin\windows (or
g.apache.zookeeper.server.ZooKeeperServer)
[2017-04-27 12:27:10,851] INFO tickTime set to 3000 (org.apache.zookeeper.server.ZooKeeperServer)
[2017-04-27 12:27:10,867] INFO minSessionTimeout set to -1 (org.apache.zookeeper.server.ZooKeeperServer)
[2017-04-27 12:27:10,867] INFO maxSessionTimeout set to -1 (org.apache.zookeeper.server.ZooKeeperServer)
[2017-04-27 12:27:10,976] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)
```

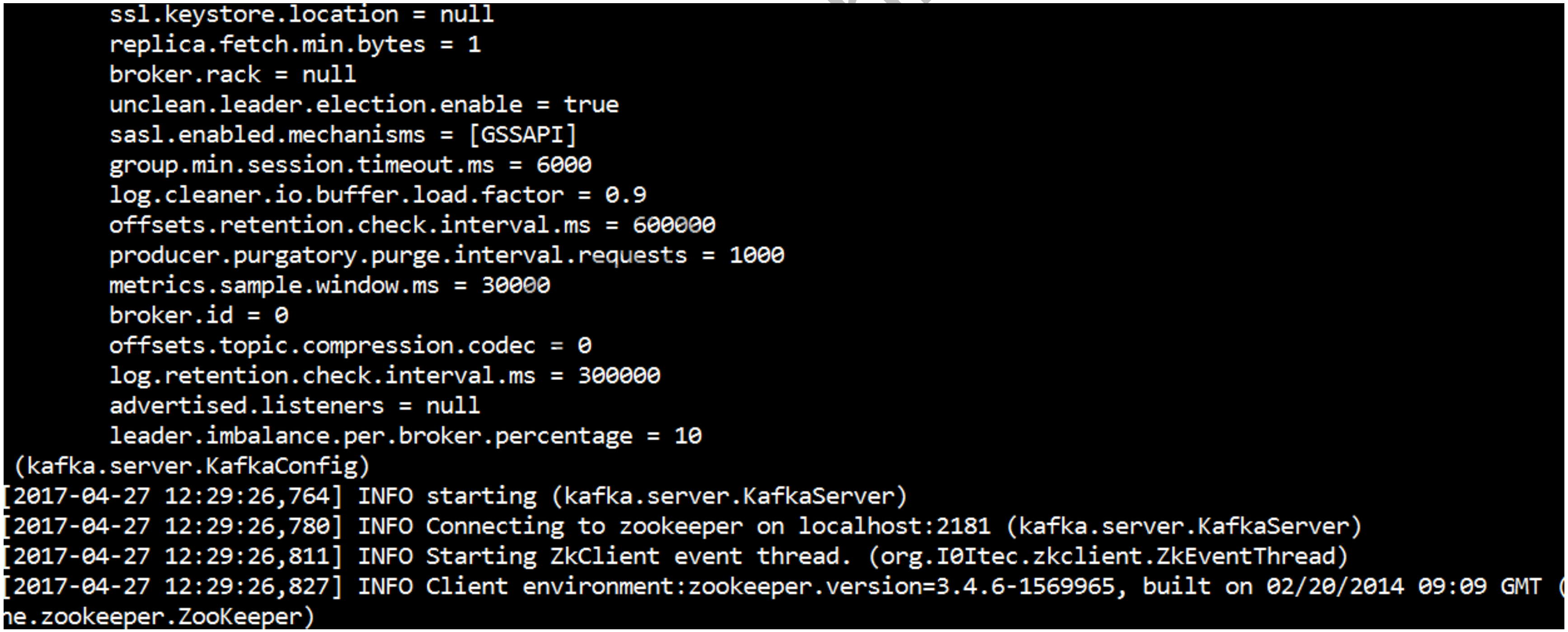
Starting kafka server

Linux

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

Windows

bin/windows/kafka-server-start.bat config/server.properties



```
        ssl.keystore.location = null
        replica.fetch.min.bytes = 1
        broker.rack = null
        unclean.leader.election.enable = true
        sasl.enabled.mechanisms = [GSSAPI]
        group.min.session.timeout.ms = 6000
        log.cleaner.io.buffer.load.factor = 0.9
        offsets.retention.check.interval.ms = 600000
        producer.purgatory.purge.interval.requests = 1000
        metrics.sample.window.ms = 30000
        broker.id = 0
        offsets.topic.compression.codec = 0
        log.retention.check.interval.ms = 300000
        advertised.listeners = null
        leader.imbalance.per.broker.percentage = 10
 (kafka.server.KafkaConfig)
[2017-04-27 12:29:26,764] INFO starting (kafka.server.KafkaServer)
[2017-04-27 12:29:26,780] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2017-04-27 12:29:26,811] INFO Starting ZkClient event thread. (org.I0Itec.zkclient.ZkEventThread)
[2017-04-27 12:29:26,827] INFO Client environment:zookeeper.version=3.4.6-1569965, built on 02/20/2014 09:09 GMT (
ne.zookeeper.ZooKeeper)
```

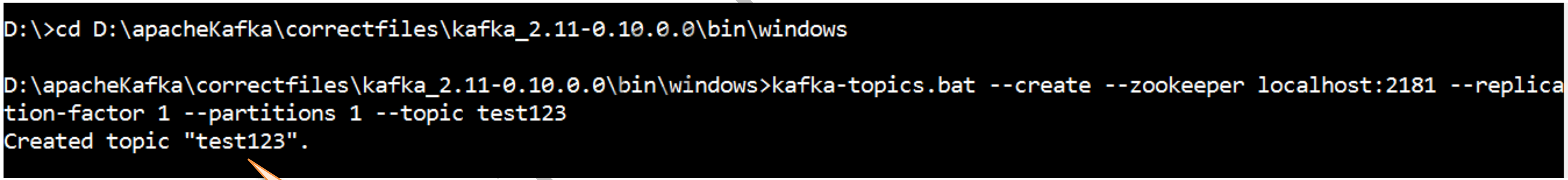
Create a topic

Linux

kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test123

Windows

kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test123



```
D:\>cd D:\apacheKafka\correctfiles\kafka_2.11-0.10.0.0\bin\windows

D:\apacheKafka\correctfiles\kafka_2.11-0.10.0.0\bin\windows>kafka-topics.bat --create --zookeeper localhost:2181 --replica
tion-factor 1 --partitions 1 --topic test123
Created topic "test123".
```

Topic created

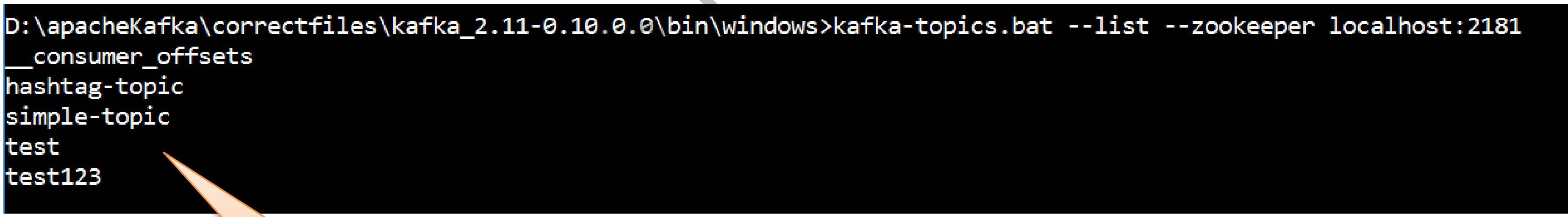
List topics

Linux

kafka-topics.sh --list --zookeeper localhost:2181

Windows

kafka-topics.bat --list --zookeeper localhost:2181



```
D:\apacheKafka\correctfiles\kafka_2.11-0.10.0.0\bin\windows>kafka-topics.bat --list --zookeeper localhost:2181
__consumer_offsets
hashtag-topic
simple-topic
test
test123
```

List of Topics

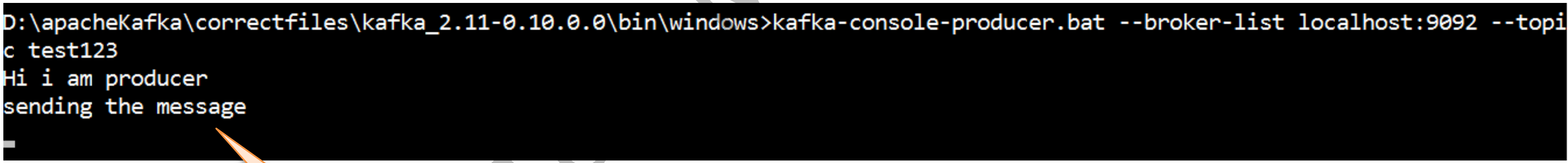
Start Producer

Linux

bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test123

Windows

bin/windows/kafka-console-producer.bat --broker-list localhost:9092 --topic test123



```
D:\apacheKafka\correctfiles\kafka_2.11-0.10.0.0\bin\windows>kafka-console-producer.bat --broker-list localhost:9092 --topi
c test123
Hi i am producer
sending the message
_
```

Producer sending the messages

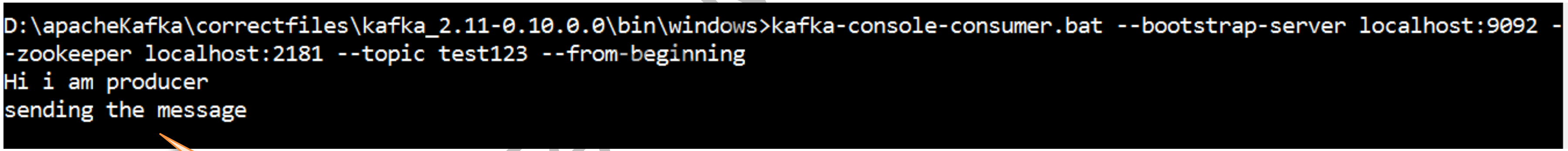
Start Consumer

Linux

bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --zookeeper localhost:2181 --topic test123 --from-beginning

Windows

Bin/windows/kafka-console-consumer.bat --bootstrap-server localhost:9092 --zookeeper localhost:2181 --topic test123 --from-beginning



```
D:\apacheKafka\correctfiles\kafka_2.11-0.10.0.0\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:9092 -
-zookeeper localhost:2181 --topic test123 --from-beginning
Hi i am producer
sending the message
```

Consumer consuming the messages

Tech Mahindra

Multi Broker Cluster

TLS

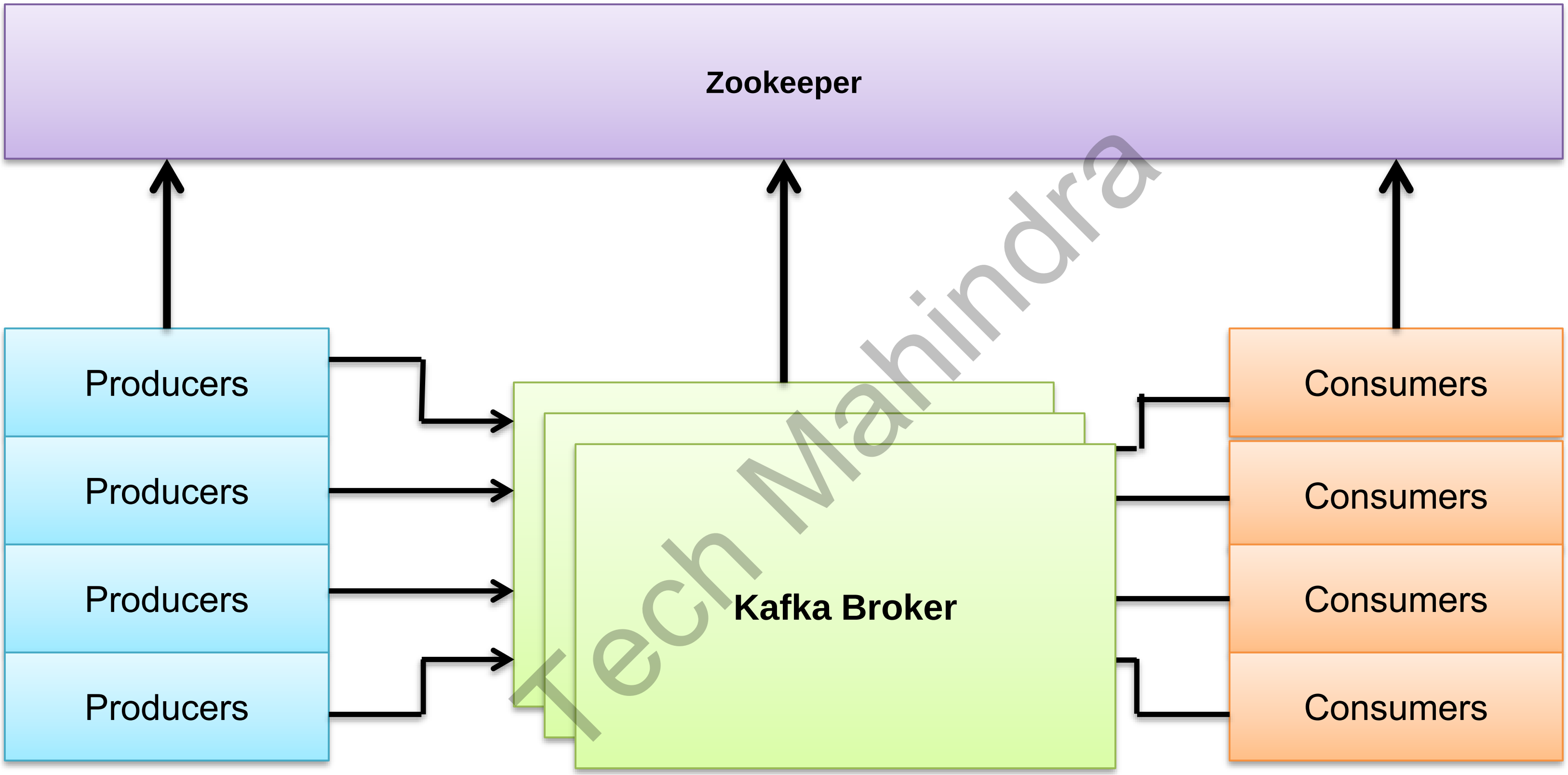
Rewards and Recognition

2012 ASTD BEST AWARD WINNER

training TOP 125 Winner - 2013

Tata Institute of Social Sciences CLO Awards India Best Virtual Learning Award

Multi Broker Cluster



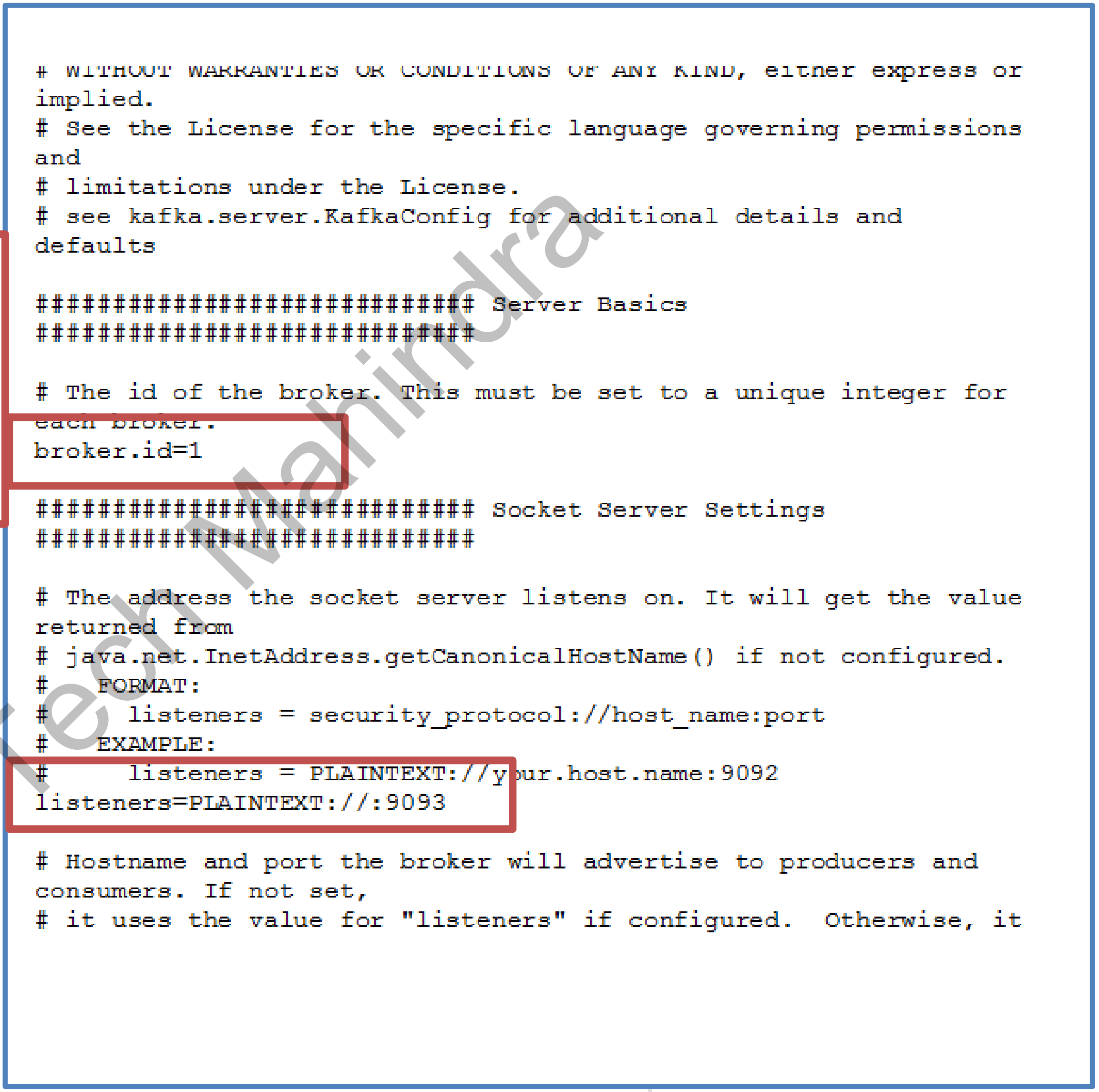
Starting other Kafka broker

Change the following properties

broker.id =1

listeners=PLAINTEXT://:9093

log.dirs=/tmp/kafka-logs-1



```
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
# See the License for the specific language governing permissions
and
# limitations under the License.
# see kafka.server.KafkaConfig for additional details and
defaults

############################# Server Basics
#############################

# The id of the broker. This must be set to a unique integer for
each broker.
broker.id=1

############################# Socket Server Settings
#############################

# The address the socket server listens on. It will get the value
returned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#     listeners = security_protocol://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://:9093

# Hostname and port the broker will advertise to producers and
consumers. If not set,
# it uses the value for "listeners" if configured.  Otherwise, it
```

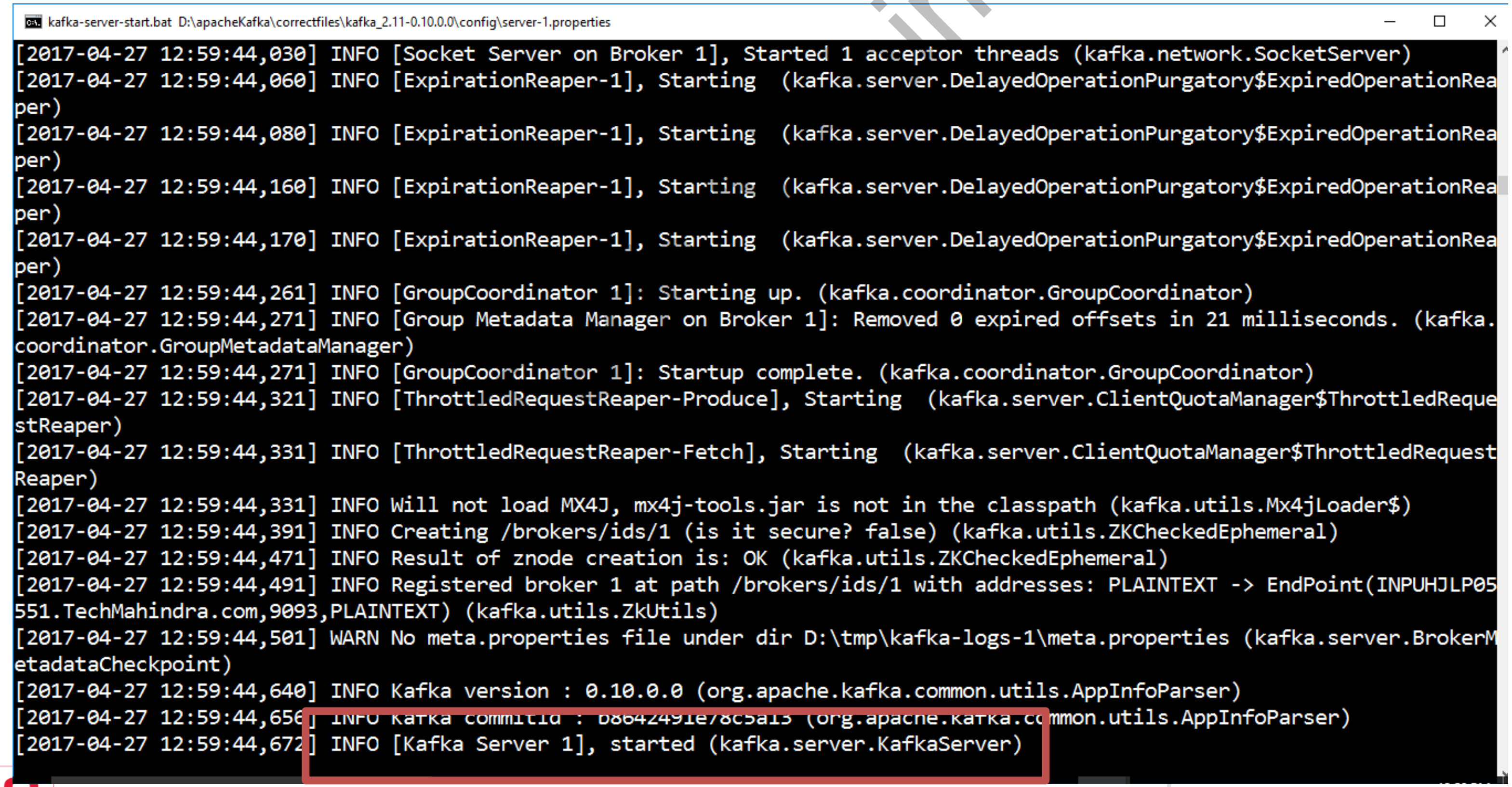
Starting other Kafka broker

Linux

bin/kafka-server-start.sh config/server-1.properties

Windows

bin/windows/kafka-server-start.bat config/server-1.properties


kafka-server-start.bat D:\apacheKafka\correctfiles\kafka_2.11-0.10.0.0\config\server-1.properties

```
[2017-04-27 12:59:44,030] INFO [Socket Server on Broker 1], Started 1 acceptor threads (kafka.network.SocketServer)
[2017-04-27 12:59:44,060] INFO [ExpirationReaper-1], Starting  (kafka.server.DelayedOperationPurgatory$ExpiredOperationRea
per)
[2017-04-27 12:59:44,080] INFO [ExpirationReaper-1], Starting  (kafka.server.DelayedOperationPurgatory$ExpiredOperationRea
per)
[2017-04-27 12:59:44,160] INFO [ExpirationReaper-1], Starting  (kafka.server.DelayedOperationPurgatory$ExpiredOperationRea
per)
[2017-04-27 12:59:44,170] INFO [ExpirationReaper-1], Starting  (kafka.server.DelayedOperationPurgatory$ExpiredOperationRea
per)
[2017-04-27 12:59:44,261] INFO [GroupCoordinator 1]: Starting up. (kafka.coordinator.GroupCoordinator)
[2017-04-27 12:59:44,271] INFO [Group Metadata Manager on Broker 1]: Removed 0 expired offsets in 21 milliseconds. (kafka.
coordinator.GroupMetadataManager)
[2017-04-27 12:59:44,271] INFO [GroupCoordinator 1]: Startup complete. (kafka.coordinator.GroupCoordinator)
[2017-04-27 12:59:44,321] INFO [ThrottledRequestReaper-Produce], Starting  (kafka.server.ClientQuotaManager$ThrottledReque
stReaper)
[2017-04-27 12:59:44,331] INFO [ThrottledRequestReaper-Fetch], Starting  (kafka.server.ClientQuotaManager$ThrottledRequest
Reaper)
[2017-04-27 12:59:44,331] INFO Will not load MX4J, mx4j-tools.jar is not in the classpath (kafka.utils.Mx4jLoader$)
[2017-04-27 12:59:44,391] INFO Creating /brokers/ids/1 (is it secure? false) (kafka.utils.ZKCheckedEphemeral)
[2017-04-27 12:59:44,471] INFO Result of znode creation is: OK (kafka.utils.ZKCheckedEphemeral)
[2017-04-27 12:59:44,491] INFO Registered broker 1 at path /brokers/ids/1 with addresses: PLAINTEXT -> EndPoint(INPUHJLP05
551.TechMahindra.com,9093,PLAINTEXT) (kafka.utils.ZkUtils)
[2017-04-27 12:59:44,501] WARN No meta.properties file under dir D:\tmp\kafka-logs-1\meta.properties (kafka.server.BrokerM
etadataCheckpoint)
[2017-04-27 12:59:44,640] INFO Kafka version : 0.10.0.0 (org.apache.kafka.common.utils.AppInfoParser)
[2017-04-27 12:59:44,656] INFO Kafka commitId : b8642491e78c5a13 (org.apache.kafka.common.utils.AppInfoParser)
[2017-04-27 12:59:44,672] INFO [Kafka Server 1], started (kafka.server.KafkaServer)
```

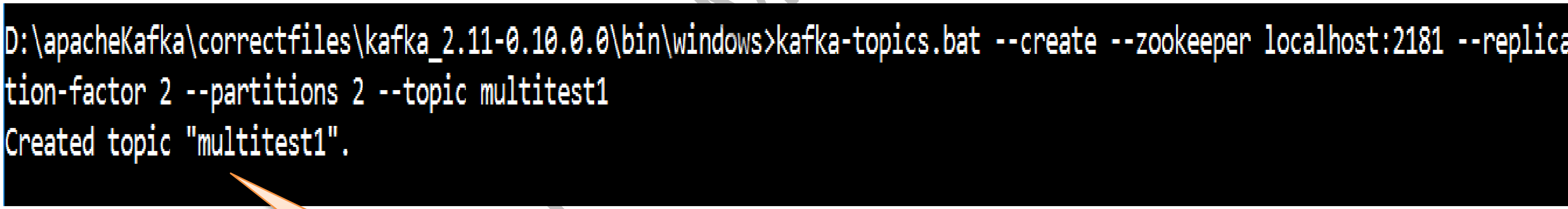
Create a topic

Linux

kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 2 --partitions 2 --topic multitest1

Windows

kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 2 --partitions 2 --topic multitest1



```
D:\apacheKafka\correctfiles\kafka_2.11-0.10.0.0\bin\windows>kafka-topics.bat --create --zookeeper localhost:2181 --replica
tion-factor 2 --partitions 2 --topic multitest1
Created topic "multitest1".
```

Topic created for multi broker cluster

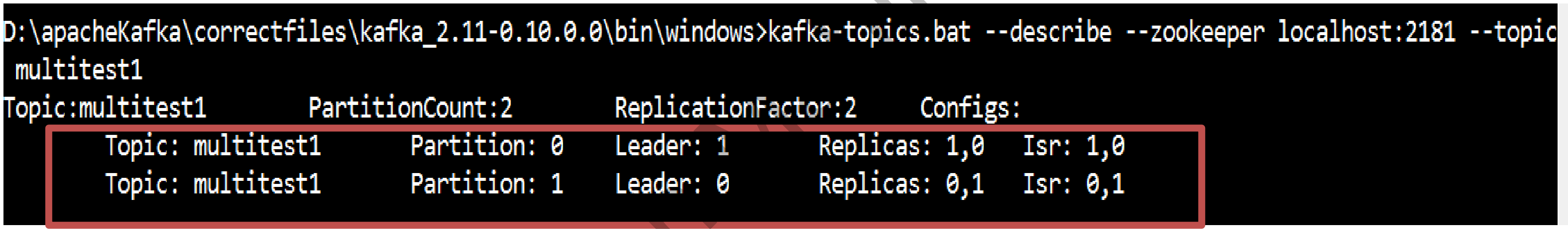
Describe the Multi broker cluster

Linux

kafka-topics.sh --describe --zookeeper localhost:2181 --topic multitest1

Windows

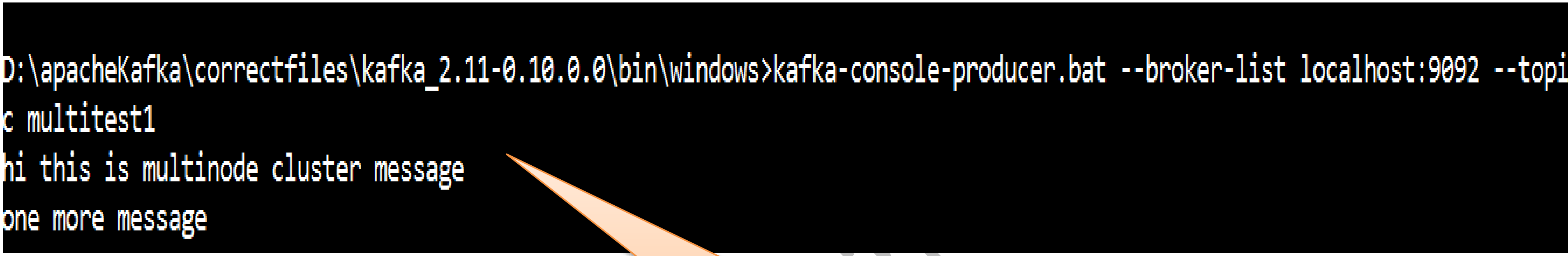
kafka-topics.bat --describe --zookeeper localhost:2181 --topic multitest1



```
D:\apacheKafka\correctfiles\kafka_2.11-0.10.0.0\bin\windows>kafka-topics.bat --describe --zookeeper localhost:2181 --topic
 multitest1
Topic:multitest1        PartitionCount:2        ReplicationFactor:2     Configs:
        Topic: multitest1       Partition: 0    Leader: 1       Replicas: 1,0   Isr: 1,0
        Topic: multitest1       Partition: 1    Leader: 0       Replicas: 0,1   Isr: 0,1
```

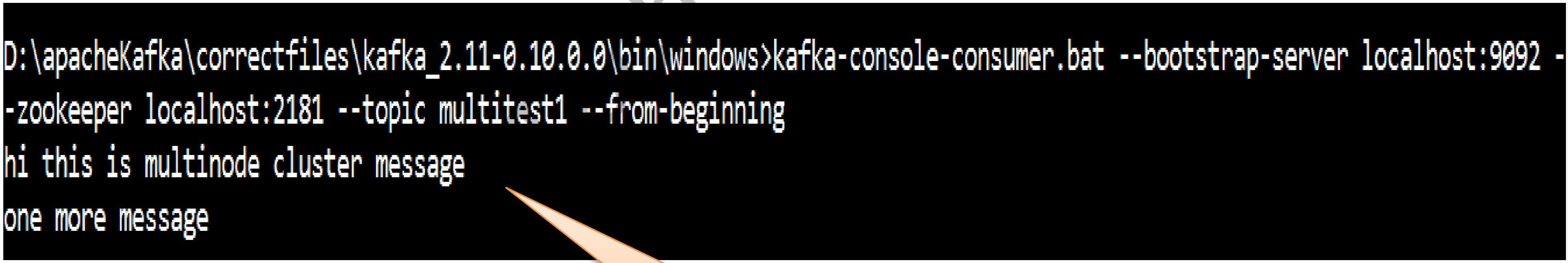
- "leader" is the node responsible for all reads and writes for the given partition.
- Each node will be the leader for a randomly selected portion of the partitions. "replicas" is the list of nodes that replicate the log for this partition regardless of whether they are the leader or even if they are currently alive.
- "isr" is the set of "in-sync" replicas. This is the subset of the replicas list that is currently alive and caught-up to the leader.

Sending messages in multi broker cluster



```
D:\apacheKafka\correctfiles\kafka_2.11-0.10.0.0\bin\windows>kafka-console-producer.bat --broker-list localhost:9092 --topi
c multitest1
hi this is multinode cluster message
one more message
```

Producer sending messages

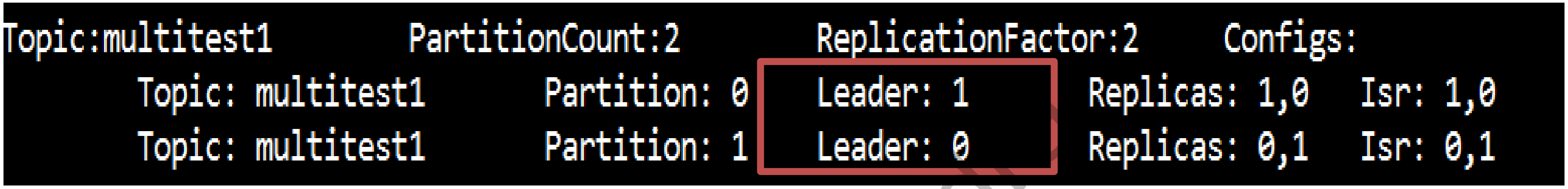


```
D:\apacheKafka\correctfiles\kafka_2.11-0.10.0.0\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:9092 -
-zookeeper localhost:2181 --topic multitest1 --from-beginning
hi this is multinode cluster message
one more message
```

Consumer consuming messages

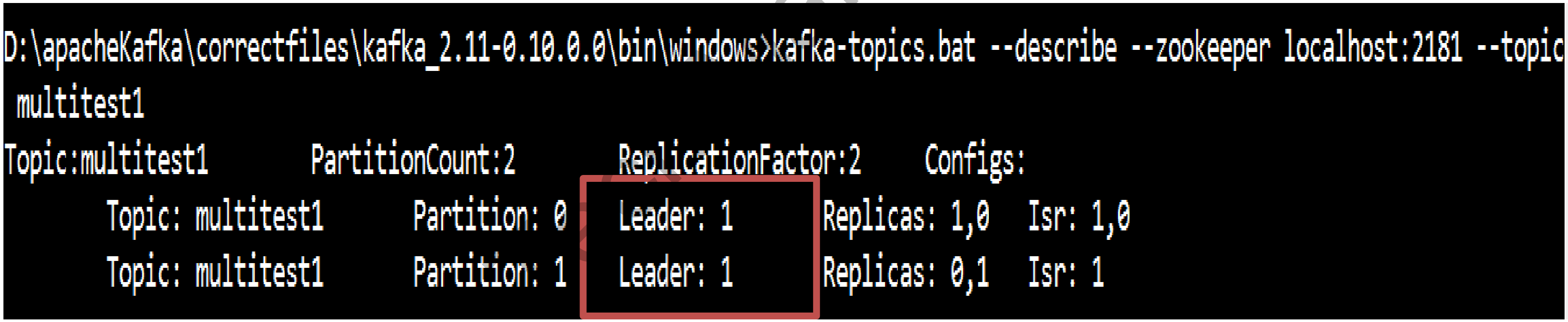
Test fault tolerance

before broker 0 shutdown



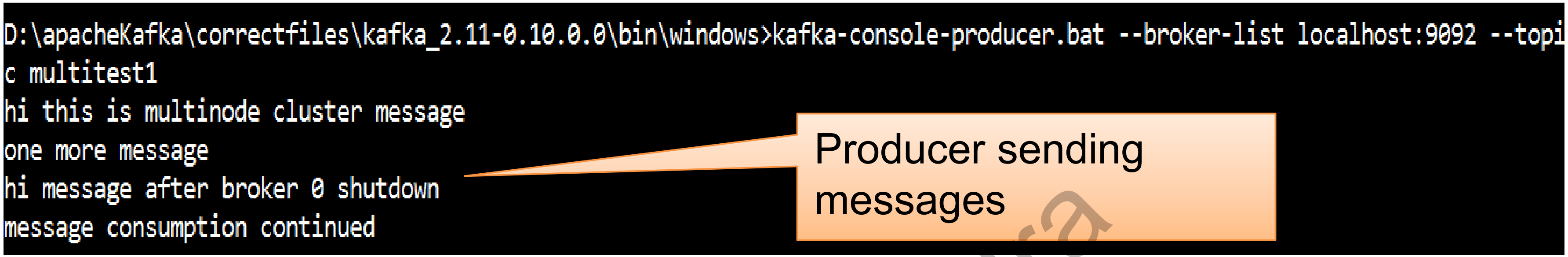
```
Topic:multitest1        PartitionCount:2        ReplicationFactor:2     Configs:
        Topic: multitest1       Partition: 0    Leader: 1       Replicas: 1,0   Isr: 1,0
        Topic: multitest1       Partition: 1    Leader: 0       Replicas: 0,1   Isr: 0,1
```

- After shutdown of one of the Kafka broker



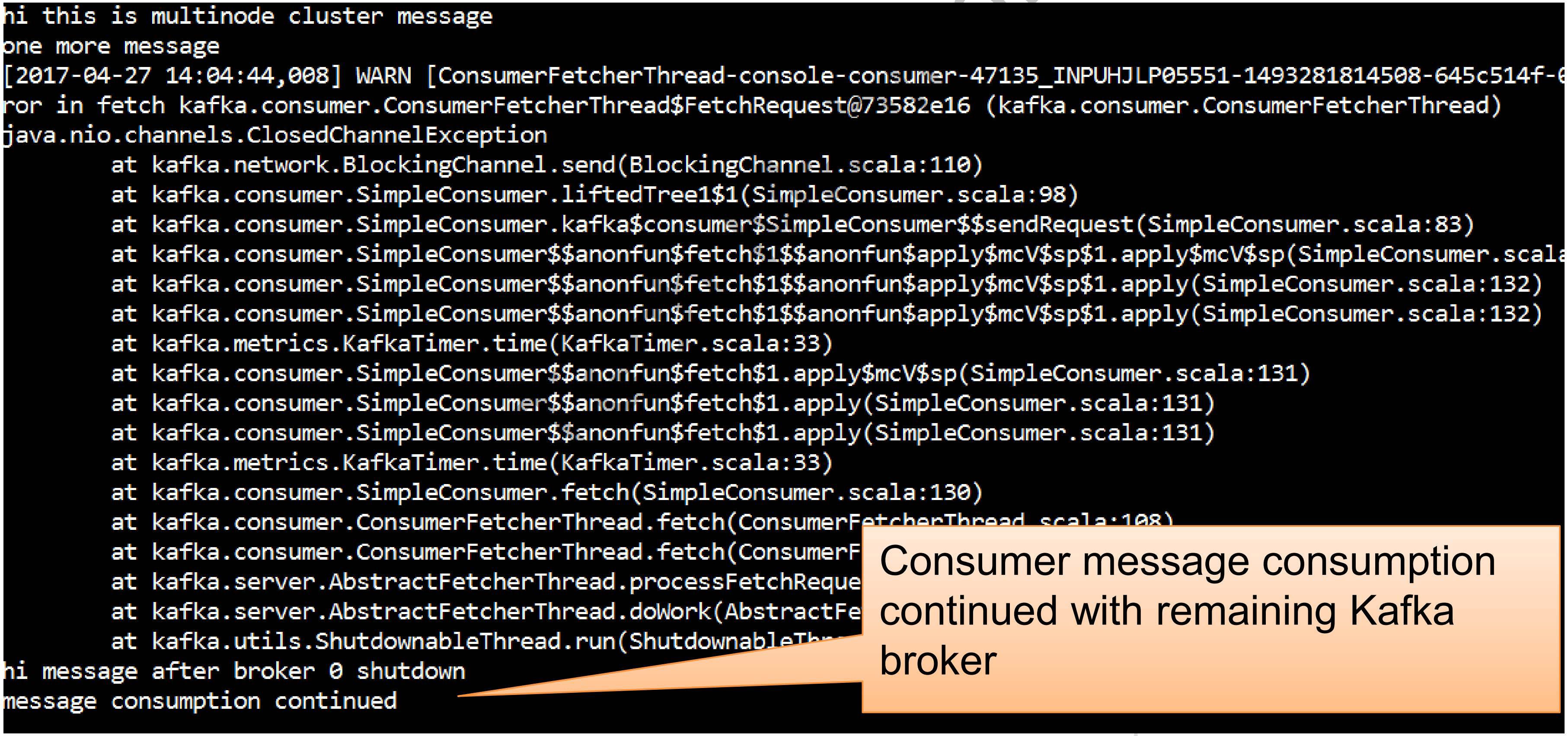
```
D:\apacheKafka\correctfiles\kafka_2.11-0.10.0.0\bin\windows>kafka-topics.bat --describe --zookeeper localhost:2181 --topic
 multitest1
Topic:multitest1        PartitionCount:2        ReplicationFactor:2     Configs:
        Topic: multitest1       Partition: 0    Leader: 1       Replicas: 1,0   Isr: 1,0
        Topic: multitest1       Partition: 1    Leader: 1       Replicas: 0,1   Isr: 1
```

Sending messages continued



```
D:\apacheKafka\correctfiles\kafka_2.11-0.10.0.0\bin\windows>kafka-console-producer.bat --broker-list localhost:9092 --topi
c multitest1
hi this is multinode cluster message
one more message
hi message after broker 0 shutdown
message consumption continued
```

Producer sending messages



```
hi this is multinode cluster message
one more message
[2017-04-27 14:04:44,008] WARN [ConsumerFetcherThread-console-consumer-47135_INPUHJLP05551-1493281814508-645c514f-0
ror in fetch kafka.consumer.ConsumerFetcherThread$FetchRequest@73582e16 (kafka.consumer.ConsumerFetcherThread)
java.nio.channels.ClosedChannelException
        at kafka.network.BlockingChannel.send(BlockingChannel.scala:110)
        at kafka.consumer.SimpleConsumer.liftedTree1$1(SimpleConsumer.scala:98)
        at kafka.consumer.SimpleConsumer.kafka$consumer$SimpleConsumer$$sendRequest(SimpleConsumer.scala:83)
        at kafka.consumer.SimpleConsumer$$anonfun$fetch$1$$anonfun$apply$mcV$sp$1.apply$mcV$sp(SimpleConsumer.scala
        at kafka.consumer.SimpleConsumer$$anonfun$fetch$1$$anonfun$apply$mcV$sp$1.apply(SimpleConsumer.scala:132)
        at kafka.consumer.SimpleConsumer$$anonfun$fetch$1$$anonfun$apply$mcV$sp$1.apply(SimpleConsumer.scala:132)
        at kafka.metrics.KafkaTimer.time(KafkaTimer.scala:33)
        at kafka.consumer.SimpleConsumer$$anonfun$fetch$1.apply$mcV$sp(SimpleConsumer.scala:131)
        at kafka.consumer.SimpleConsumer$$anonfun$fetch$1.apply(SimpleConsumer.scala:131)
        at kafka.consumer.SimpleConsumer$$anonfun$fetch$1.apply(SimpleConsumer.scala:131)
        at kafka.metrics.KafkaTimer.time(KafkaTimer.scala:33)
        at kafka.consumer.SimpleConsumer.fetch(SimpleConsumer.scala:130)
        at kafka.consumer.ConsumerFetcherThread.fetch(ConsumerFetcherThread.scala:108)
        at kafka.consumer.ConsumerFetcherThread.fetch(ConsumerF
        at kafka.server.AbstractFetcherThread.processFetchReque
        at kafka.server.AbstractFetcherThread.doWork(AbstractFe
        at kafka.utils.ShutdownableThread.run(ShutdownableThr
hi message after broker 0 shutdown
message consumption continued
```

Consumer message consumption continued with remaining Kafka broker

TLS

APIs

TLS

Kafka APIs

Producer API

- allows an application to publish a stream of records to one or more Kafka topics.

Consumer API

- allows an application to subscribe to one or more topics and process the stream of records produced to them.

Streams API

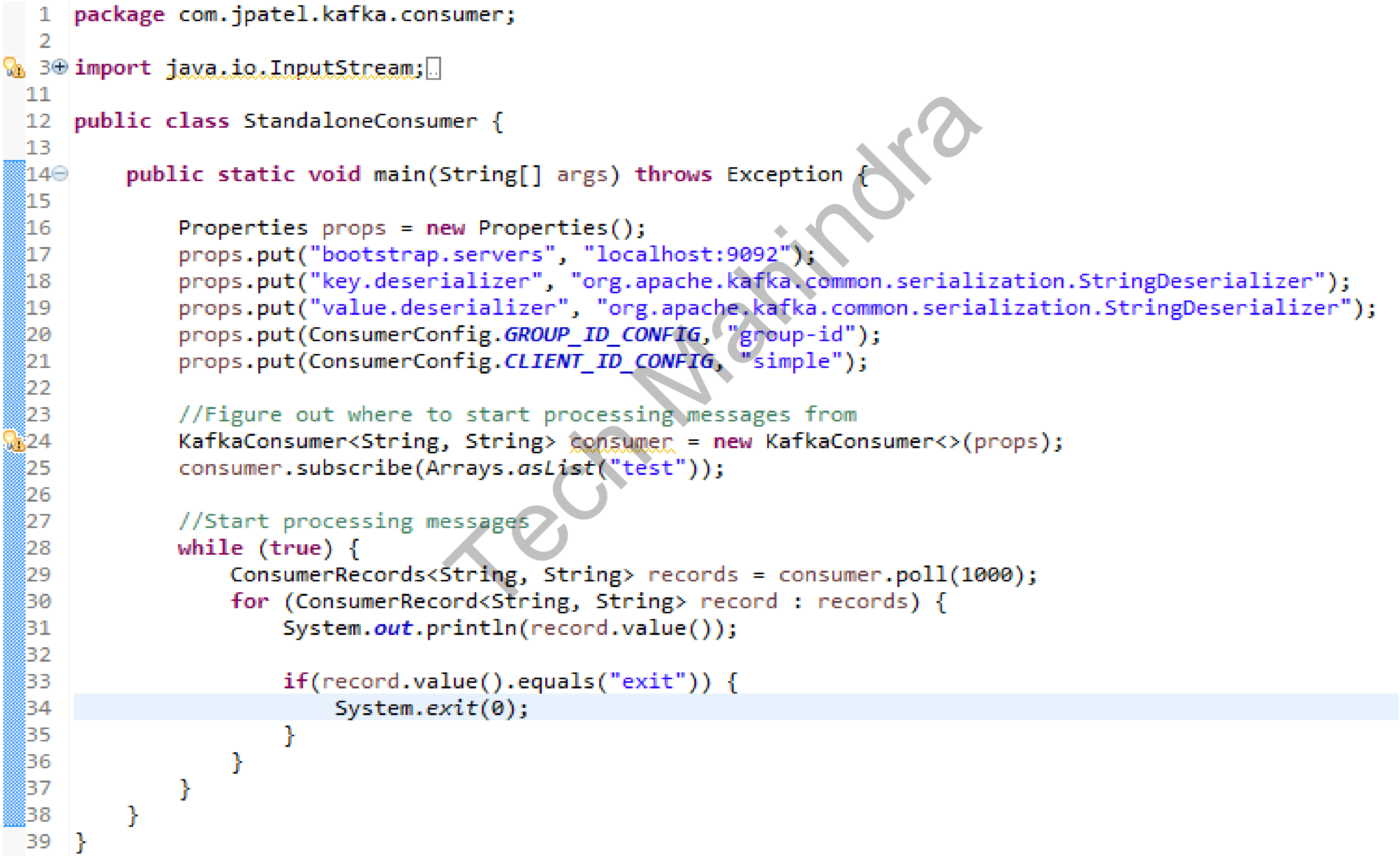
- allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.

Connector API

- 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.

Kafka APIs

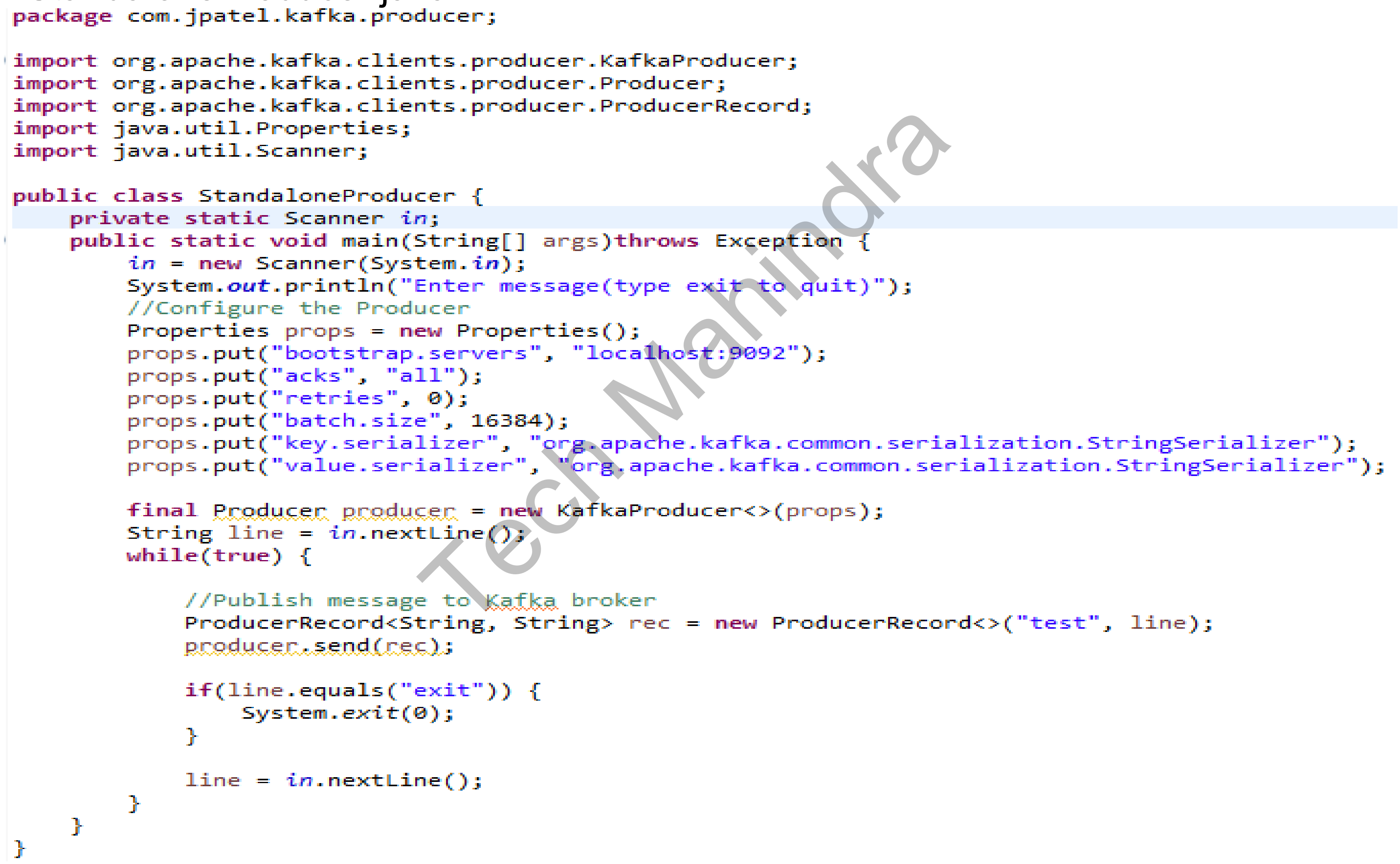
- StandaloneConsumer.java



```java
package com.jpatel.kafka.consumer;

import java.io.InputStream;

public class StandaloneConsumer {

    public static void main(String[] args) throws Exception {

        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092");
        props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "group-id");
        props.put(ConsumerConfig.CLIENT_ID_CONFIG, "simple");

        //Figure out where to start processing messages from
        KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
        consumer.subscribe(Arrays.asList("test"));

        //Start processing messages
        while (true) {
            ConsumerRecords<String, String> records = consumer.poll(1000);
            for (ConsumerRecord<String, String> record : records) {
                System.out.println(record.value());

                if(record.value().equals("exit")) {
                    System.exit(0);
                }
            }
        }
    }
}
```

Kafka APIs

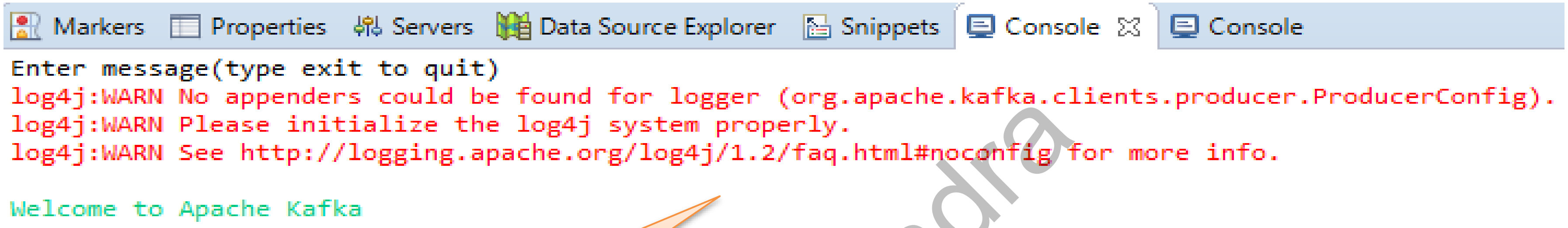
- StandaloneProducer.java



```java
package com.jpatel.kafka.producer;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;
import java.util.Properties;
import java.util.Scanner;

public class StandaloneProducer {
    private static Scanner in;
    public static void main(String[] args)throws Exception {
        in = new Scanner(System.in);
        System.out.println("Enter message(type exit to quit)");
        //Configure the Producer
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092");
        props.put("acks", "all");
        props.put("retries", 0);
        props.put("batch.size", 16384);
        props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        final Producer producer = new KafkaProducer<>(props);
        String line = in.nextLine();
        while(true) {

            //Publish message to Kafka broker
            ProducerRecord<String, String> rec = new ProducerRecord<>("test", line);
            producer.send(rec);

            if(line.equals("exit")) {
                System.exit(0);
            }

            line = in.nextLine();
        }
    }
}
```

Kafka APIs

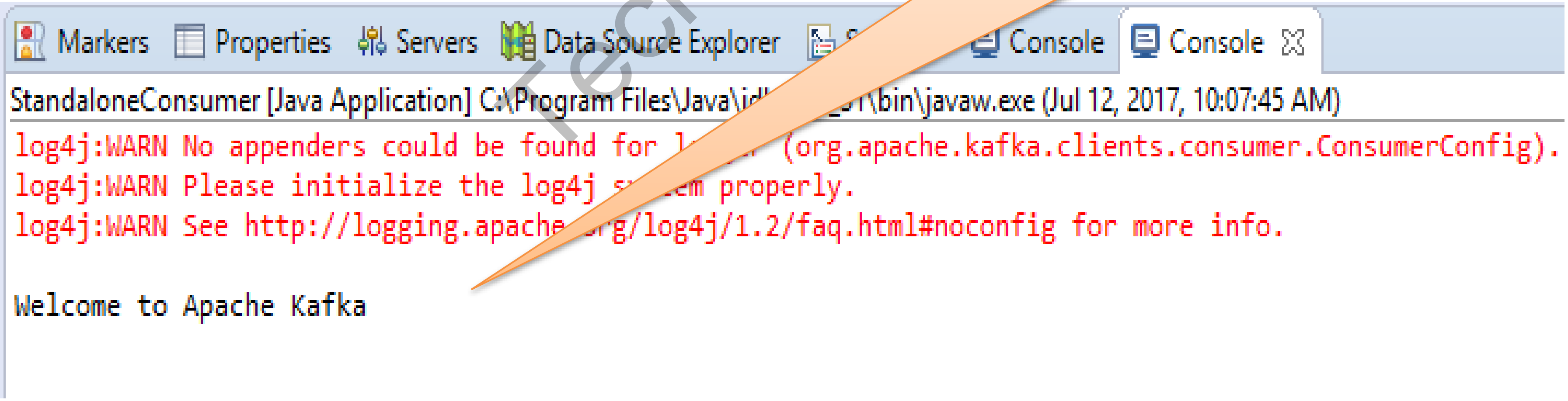
- Output:



```
Enter message(type exit to quit)
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.producer.ProducerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

Welcome to Apache Kafka
```

Producer sending messages

Consumer consuming messages



```
StandaloneConsumer [Java Application] C:\Program Files\Java\jd[illegible]\bin\javaw.exe (Jul 12, 2017, 10:07:45 AM)
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.consumer.ConsumerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

Welcome to Apache Kafka
```