DADS 6002 / CI 7301
Big Data Analytics

Data Ingestion

- Data Ingestion is the process to import external data into storages for data analytics.
- It is the first stage of the data science pipeline.
- Imported data can be structured data, semi-structured/unstructured data, or streaming data generated from log files or external events.

Sqoop

- It is designed to transfer data between a relational database like MySQL or Oracle into a Hadoop data store, including HDFS, Hive and Hbase.
- It is not designed to support continuous ingestion of data from multiple sources (streaming data).
- It automates most of the data transfer by reading the schema information directly from RDBMS then use MapReduce to import and export the data to and from Hadoop.

Importing from MySQL to HDFS

- When importing data from relational database like MySQL, Sqoop reads the source database to gather necessary metadata for the data being imported.
- Sqoop then submits a map-only Hadoop jobs to transfer the table data based on the captured metadata.
- This map-only job produces a set of serialized files which can be delimited text files, binary format (e.g. Avro) or sequenceFiles containing a copy of imported table or dataset.

Importing from MySQL to HDFS

- Default the files are saved as comma-separated files to directory on HDFS with a name that corresponds to the source table name.
- Let first create a simple database under MySQL.

```
# mysql –uroot –pcloudera
mysql > create database energydata;
mysql > use energydata;
mysql > create table avgprice_by_state (
              year INT NOT NULL,
```

Importing from MySQL to HDFS

```
    state VARCHAR(5) NOT NULL,
    sector  VARCHAR(255),
    residential  DECIMAL(10,2),
    industrial   DECIMAL(10,2),
    transportation      DECIMAL(10,2),
    other            DECIMAL(10, 2),
    total            DECIMAL(10,2) );
mysql > quit;
```

Importing from MySQL to HDFS

- Download the file from the MS Teams to the shared folder /home/cloudera/vbshare then copy it to the working directory or

  # wget https://github.com/bbengfort/hadoop-fundamentals/raw/data/avgprice_kwh_state.zip

  # unzip avgprice_kwh_state.zip

  # mysql –h localhost –uroot –pcloudera

  --local-infile=1

  mysql > load data local infile ‘/home/cloudera/avgprice_kwh_state.csv’ into table avgprice_by_state fields terminated by ‘,’ lines terminated by ‘\n’ ignore 1 lines;

  mysql > quit;

Importing from MySQL to HDFS

sqoop import --connect jdbc:mysql://localhost:3306/energydata
--username root --password cloudera --table avgprice_by_state --target-dir /user/cloudera/energydata -m 1

hadoop fs –cat /user/cloudera/energydata/part-m-00000

(-m 1 indicates that this job should use a single map task, so create just a single HDFS file)

Importing from MySQL to Hive

- Sqoop can import data from MySQL to Hive directly without importing the data into HDFS file and then load the file to Hive table.

  # sqoop import --connect jdbc:mysql : //localhost:3306/energydata --username root --password cloudera --table avgprice_by_state --hive-table avgprice --hive-import -m 1

  # hive

  hive > select * from avgprice;

Importing from MySQL to Hbase

- Sqoop allows importing data from a relational database to Hbase.

  # mysql –uroot –pcloudera

  mysql > create database country_db;

  mysql > use country_db;

  mysql > create table country_tbl

  (id int not null, country varchar(50), primary key (id));

Importing from MySQL to Hbase

mysql > insert into country_tbl values(1, ‘USA’);

mysql > insert into country_tbl values(2, ‘CANADA’);

mysql > insert into country_tbl values(3, ‘JAPAN’);

mysql > insert into country_tbl values(4, ‘ENGLAND’);

mysql > insert into country_tbl values(5, ‘THAILAND’);

mysql > select * from country_tbl;

mysql > quit;

Importing from MySQL to Hbase

sqoop import --connect jdbc:mysql://localhost:3306/country_db --username root --password cloudera --table country_tbl --hbase-table country --column-family country-cf --hbase-row-key id --hbase-create-table –m 1

hbase shell

hbase > scan ‘country’

Ingesting Streaming data with Flume

- Flume is designed to collect and ingest high volumes of data from multiple data streams into Hadoop.
- It is normally used to collect log data from multiple web servers.
- Flume can also be customized to transport massive quantities of event data from custom event sources e.g. network traffic data, social media generated data and sensor data.
- Flume is designed to maintain both fault-tolerance and scalability through its distributed architecture.

Flume Data Flows

- Flume expresses the data ingestion pathway from origin to destination as a data flow.
- In a data flow, a unit of data or event (e.g. a single log statement) travels from a source to the next destination via a sequence of hops.
- A Flume agent is a single unit within a Flume data flow (actually a JVM process), through which events propagate.

Flume Agents

- A Flume agent consists of three configurable components:

1. Source
2. Channel
3. Sink

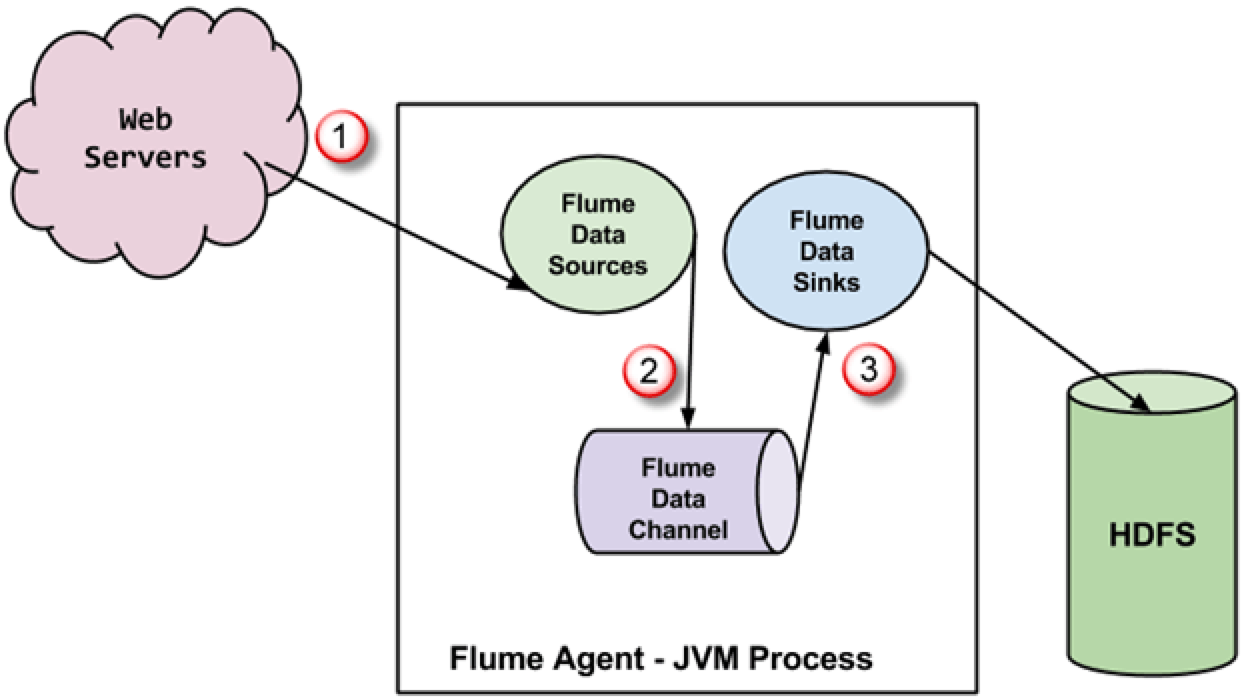
- A Flume source is configured to listen for and consume events from one or more external data sources.

Flume Agents

- Flume agent's source can accept events from an Apache access log via an exec source by running a Unix command, e.g.

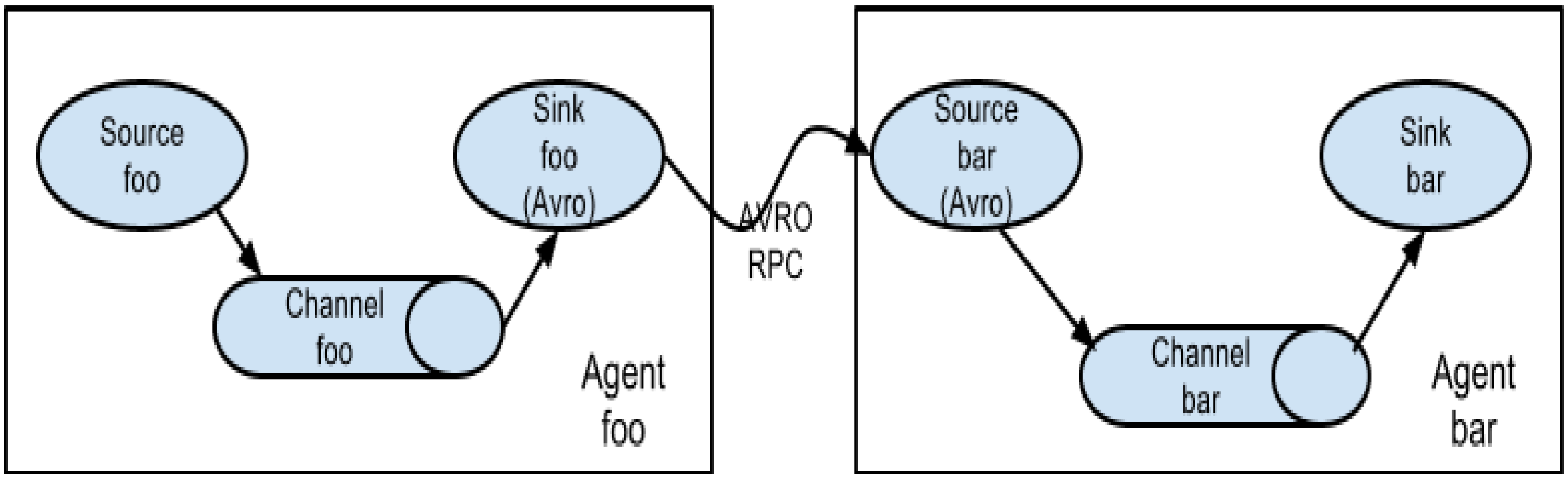
  tail –f /etc/httpd/logs/acess_log
- When the agent consumes one event, the source write it to a channel, which acts as a storage queue.
- Flume sinks read and remove events from the channel and forward them to their next hop (Flume agent) or final destination (Data stores)

Flume Data Flows



Simple Flume Data Flow

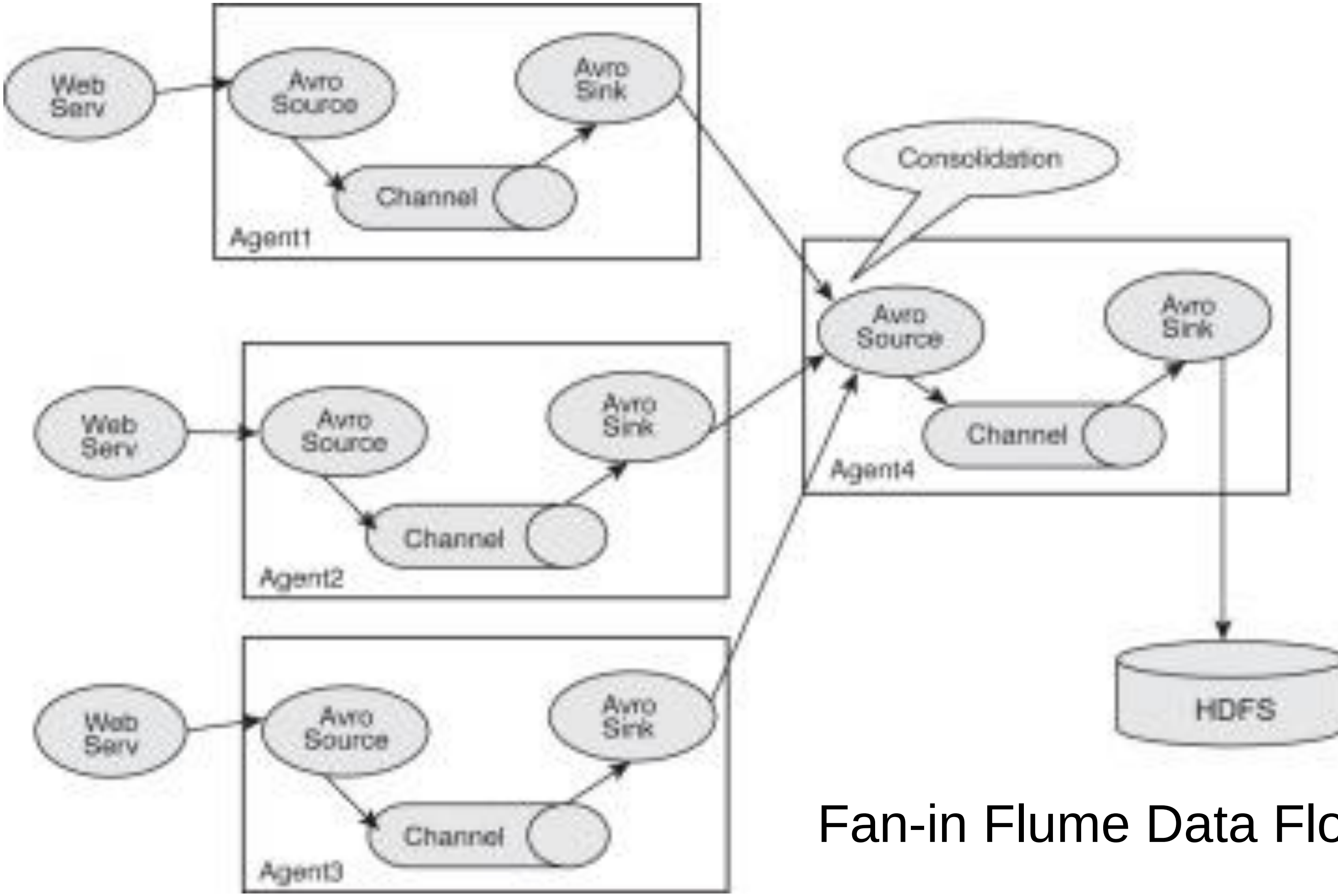
Flume Data Flows



Multi-agent Flume Data Flow

Two tier Agents provide better control over the rate of writes to the data stores

Flume Data Flows



Fan-in Flume Data Flow

Avro

- Avro is the data serialization system in Hadoop.

- It translates data like memory buffer, data structure or object state into binary or textual form that can be transported over network or stored in persistent data storage in a self describing file (Avro Data File).

- Avro stored serialized data along with the data schema in JSON format (Javascript Object Notation) in the Avro Data File.

- Avro uses Remote Procedure Calls (RPCs) to exchange schema between client and server (source and sink) during handshake.

Ingesting Product Impression Data

- Use Flume to consume the streaming user-interaction data generated by a hypothetical online store.
- Simulate an ecommerce impression log that records user interactions in the following JSON format:

{
 “sku” : “T9921-5”
 “timestamp” : 1453167527737
 “cid” : “51761”
 “action” : “add_cart”
 “ip” : “226.43.51.25”
}

Ingesting Product Impression Data

- The types of actions can include “view”, “click”, “add_cart”, “remove_cart”, and “purchase”.
- To create the necessary directories and HDFS, download and run the script as a user with sudo privileges.
- Download the script file from MS Teams or download it from the web.

wget https://raw.githubusercontent.com/bbengfort/hadoop-fundamentals/master/flume/setup.sh

Ingesting Product Impression Data

- Use nano editor to edit the flume_setup.sh as follow :

```
#!/bin/bash
hadoop fs -mkdir -p /user/cloudera/impressions/
hadoop fs -chmod 1777 /user/cloudera/impressions/

mkdir /tmp/impressions
chmod  777 /tmp/impressions
mkdir  /tmp/flume
chmod 1777 /tmp/flume
```

- Execute the setup file by
  # sh flume_setup.sh

Ingesting Product Impression Data

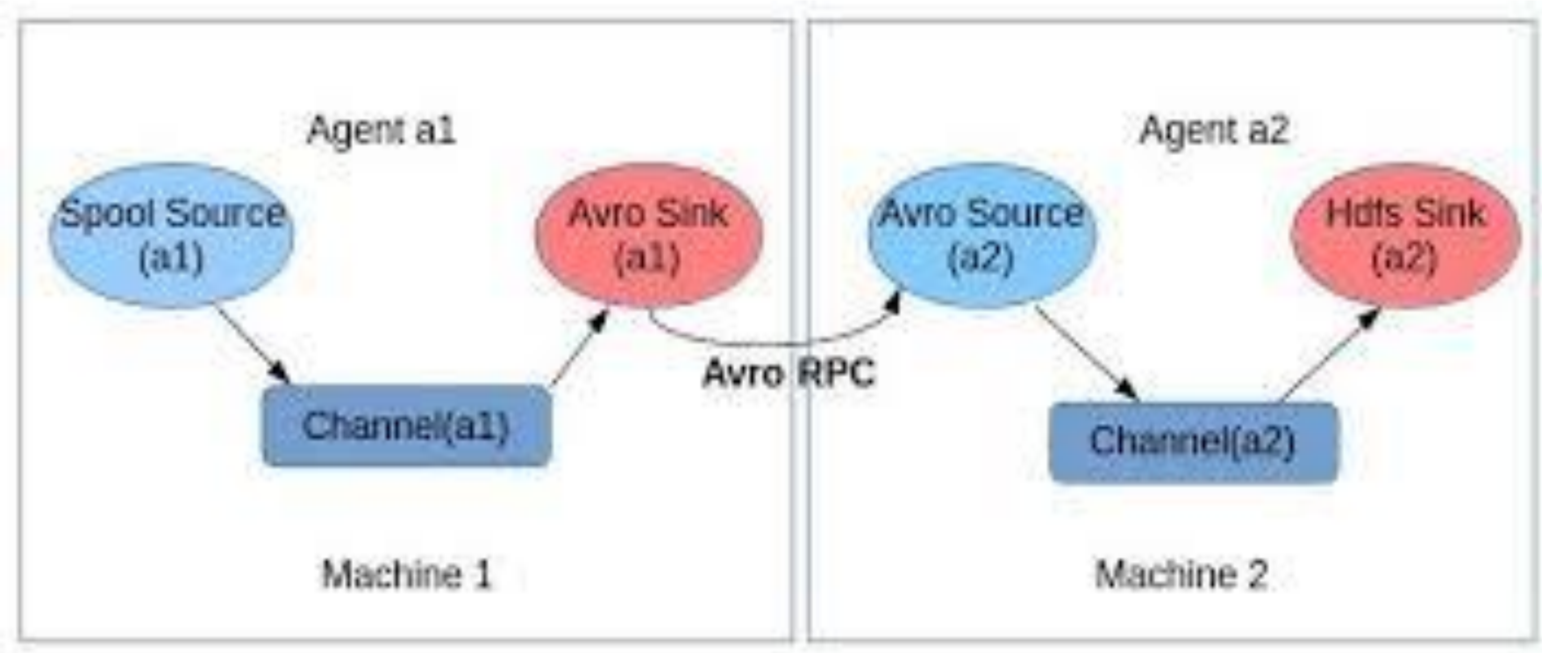
- Download a python program from MS Teams to the shared folder and edit it to create the log file named impressions.log at /tmp/impressions or download it from the web

  # wget : https://raw.githubusercontent.com/bbengfort/hadoop-fundamentals/master/flume/impression_tracker.py

- Use nano editor to edit the impression_tracker.py
  by inserting the first line into the file as follow :
  #!/usr/bin/env python

- Add execution privilege to the file then execute it as follow :
  # chmod +x impression_tracker.py
  # ./impression_tracker.py

Ingesting Product Impression Data



Source or Client Agent

Collector Agent

Download Examples of Configuration files

- Download the configuration files from MS Teams, one for client agent and one for collector agent or download them from the web and edit them accordingly.

  # wget https://raw.githubusercontent.com/bbengfort/hadoop-fundamentals/master/flume/client.conf

  # wget https://raw.githubusercontent.com/bbengfort/hadoop-fundamentals/master/flume/collector.conf

Configure Client Agent

```
# define spooling directory source :
client.sources=r1
client.sources.r1.channels=ch1
client.sources.r1.type=spooldir
client.sources.r1.spoolDir=/tmp/impressions
# define a file channel:
client.channels=ch1
client.channels.ch1.type=FILE
```

Configure Client Agent

```
# define an Avro sink:
client.sinks=k1
client.sinks.k1.type=avro
client.sinks.k1.hostname=localhost
client.sinks.k1.port=4141
client.sinks.k1.channel=ch1
```

Configure Collector Agent

```
# define an Avro source:
collector.sources=r1
collector.sources.r1.type=avro
collector.sources.r1.bind=0.0.0.0
collector.sources.r1.port=4141
collector.sources.r1.channels=ch1
```

Configure Collector Agent

```
# define a file channel using multiple disks for reliability
collector.channels=ch1
collector.channels.ch1.type=FILE
collector.channels.ch1.checkpointDir=/tmp/flume/checkpoint
collector.channels.ch1.dataDir=/tmp/flume/data

# define HDFS sinks to persist events as text
collector.sinks=k1
collector.sinks.k1.type=hdfs
collector.sinks.k1.channel=ch1
```

Configure Collector Agent

HDFS sink configuration

collector.sinks.k1.hdfs.path=/user/cloudera/impressions

collector.sinks.k1.hdfs.filePrefix=impressions

collector.sinks.k1.hdfs.fileSuffix=.log

collector.sinks.k1.hdfs.fileType=DataStream

collector.sinks.k1.hdfs.writeFormat=text

collector.sinks.k1.hdfs.batchSize=1000

Running Flume

flume-ng agent --name collector --conf . --conf-file ./collector.conf &

flume-ng agent --name client --conf . --conf-file ./client.conf &

After finish importing, check for the imported files in the target directory

hadoop fs –ls /user/cloudera/impressions

Use hadoop fs –cat /user/cloudera/impressions/…. to display one of the imported files.

Kafka

- A distributed streaming platform with three key capabilities :

1. Publish and subscribe to streams of records, similar to a message queue of enterprise messaging system.
2. Store streams of records in a fault-tolerant durable way.
3. Process streams of records as they occur.

Kafka

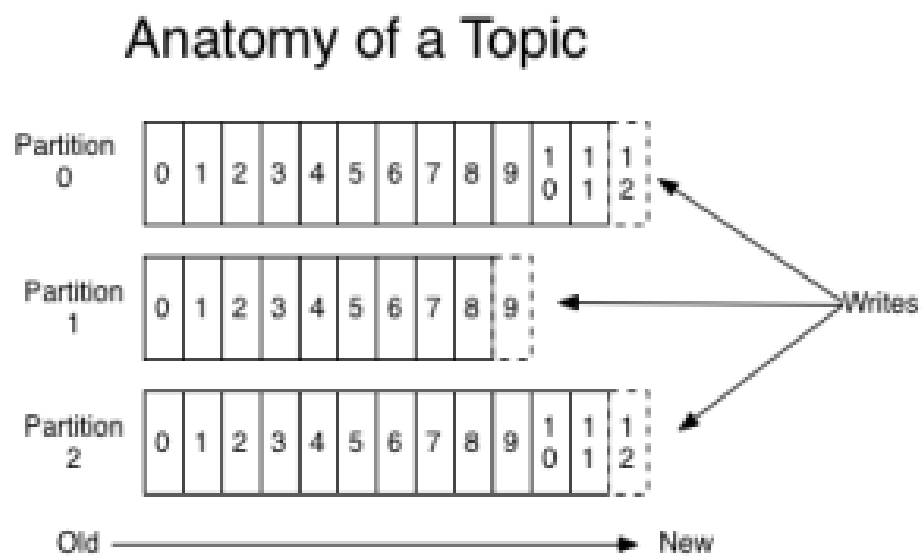
Concepts about Kafka

- Kafka is run as a cluster on one or more servers, each of which is called a broker.
- The Kafka cluster stores streams of records in categories called topics.
- Each record consists of a key, a value and a timestamp.
- Processes that publish messages to a Kafka topic are called producers.

Kafka

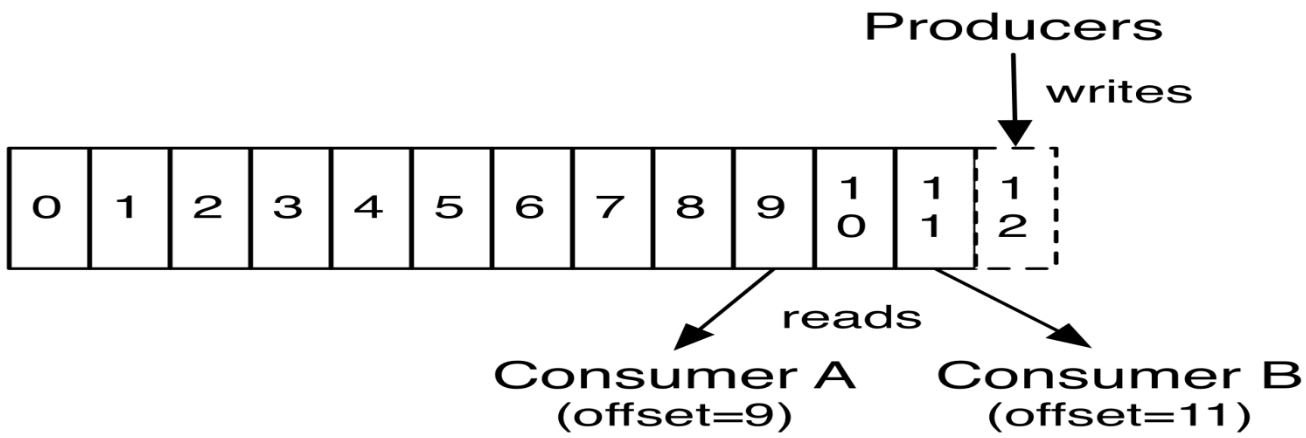
- Processes that subscribe to topics and process the feed of published messages are called consumers.
- The brokers and consumers use Zookeeper to get the state information and track message offsets.
- A topic can have many partitions (configurable). Every partition is mapped to a logical log file (a set of segment files of equal sizes). More partitions means more throughput.

Kafka


Anatomy of a Topic
Partition 0
0 1 2 3 4 5 6 7 8 9 10 11 12
Partition 1
0 1 2 3 4 5 6 7 8 9
Partition 2
0 1 2 3 4 5 6 7 8 9 10 11 12
Writes
Old
New

Kafka

- Offset is used to identify each message within the partition.



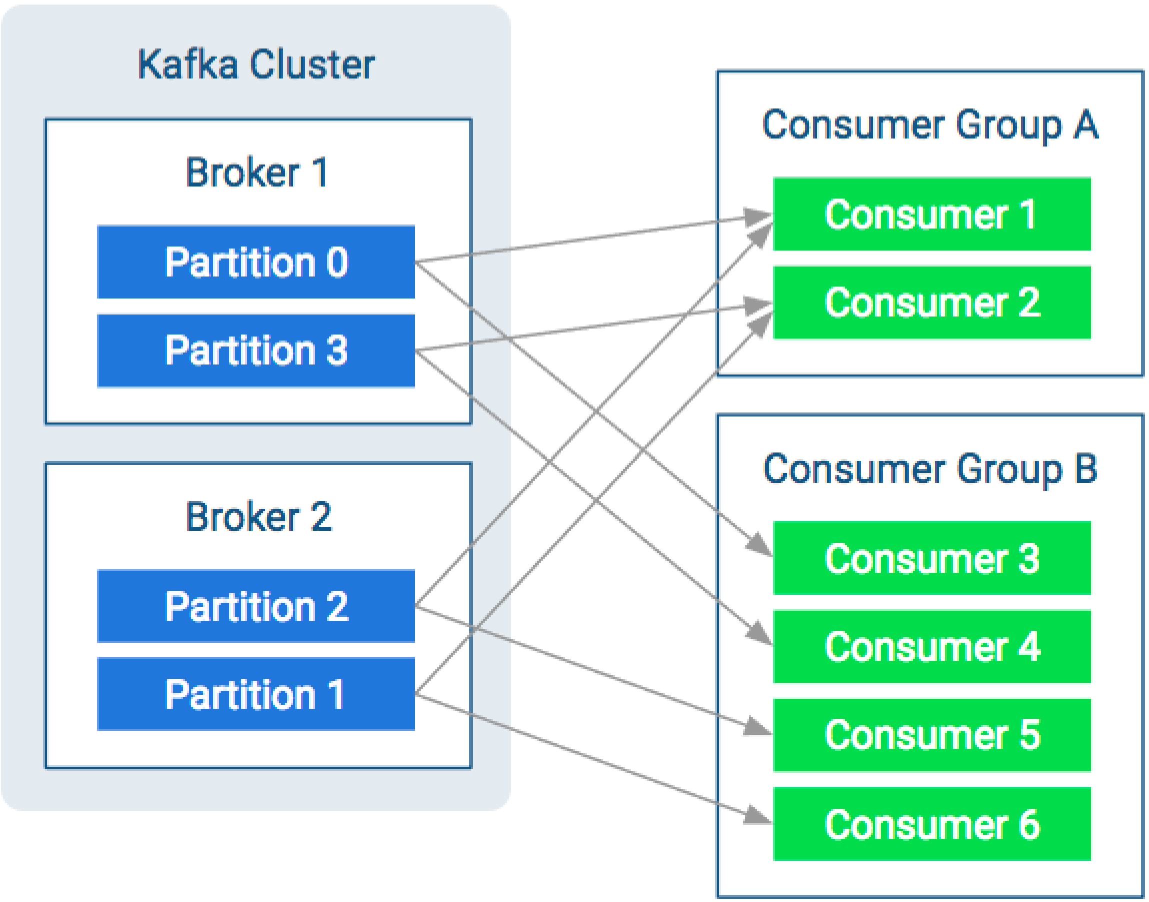
Kafka

- Each partition is optionally replicated across a configurable number of servers for fault tolerance.
- Each partition can be hosted on a different server.
- Producers publish data to the topics of their choice.
- The producer is responsible for choosing which record to assign to which partition within the topic. (This can be done in a round-robin fashion to balance load or based on keys in the record)

Kafka

- Consumers label themselves with a consumer group name.
- Each record published to a topic is delivered to one consumer instance within each subscribing consumer group.
- If all the consumer instances have the same consumer group, then the records will be 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.

Kafka


Kafka Cluster
Broker 1
Partition 0
Partition 3
Broker 2
Partition 2
Partition 1
Consumer Group A
Consumer 1
Consumer 2
Consumer Group B
Consumer 3
Consumer 4
Consumer 5
Consumer 6

Kafka

- To use Kafka to implement a queuing system, only one partition need to be used in order to maintain ordering of published messages. (since there is no ordering among messages put into different partitions)

Kafka

Advantages of Kafka

- Scalability, Kafka is easy to add a large number of consumers and more servers (brokers) can be added to clusters to achieve scalability.
- Message durability, massage fault tolerance is supported.