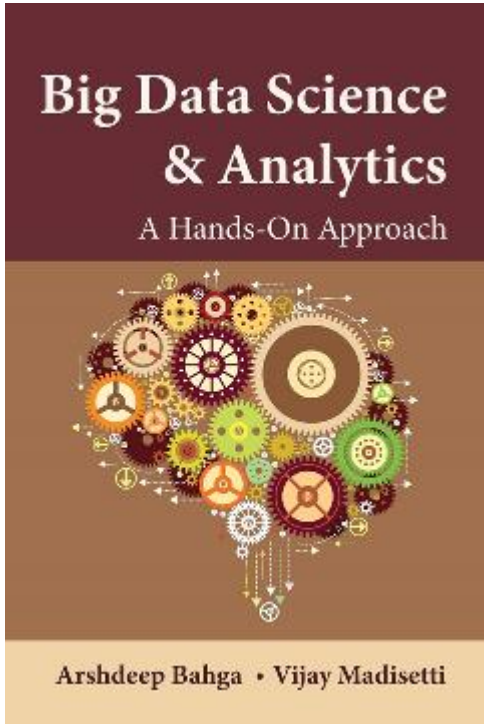
Data Acquisition

Prof. Gheith Abandah

Reference

- Chapter 5: **Data Acquisition**



- Arshdeep Bahga and Vijay Madisetti, **Big Data Science and Analytics: A Hands-On Approach**, 2019.
  - Web site: http://www.hands-on-books-series.com/

Outline

- Introduction
- Publish - Subscribe Messaging Frameworks
- Big Data Collection Systems
- Messaging Queues
- Custom Connectors

Introduction

- **Need to collect** data from various data sources:
  - **into** a distributed file system or a NoSQL database for **batch analysis** of data,
  - or **to** connect the data sources to stream or in-memory processing frameworks for **real-time analysis** of data.

Data Source Types

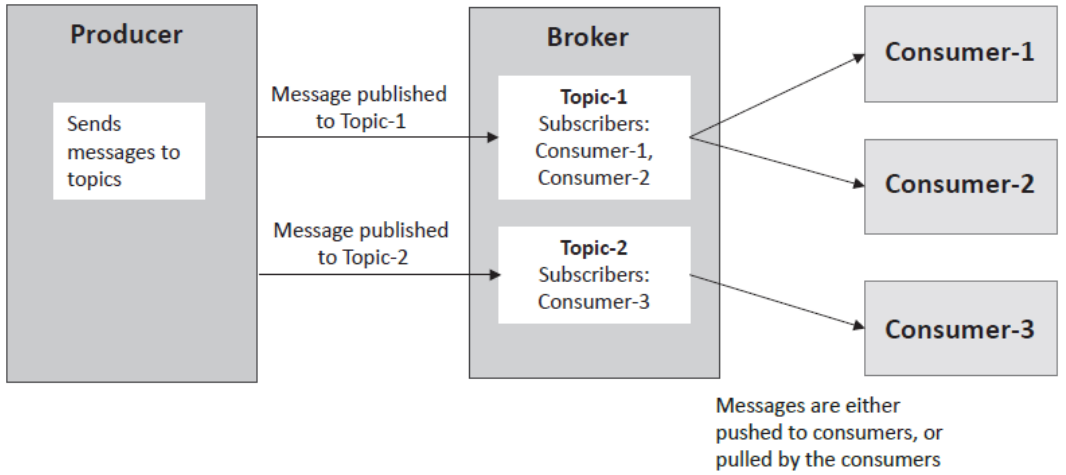
1. **Batch data sources**
   - Files
   - Logs
   - Relational databases
2. **Real-time data sources**
   - Machines generating sensor data
   - Internet of Things (IoT) systems sending real-time data
   - Social media feeds
   - Stock market feeds

Outline

- Introduction
- Publish - Subscribe Messaging Frameworks
- Big Data Collection Systems
- Messaging Queues
- Custom Connectors

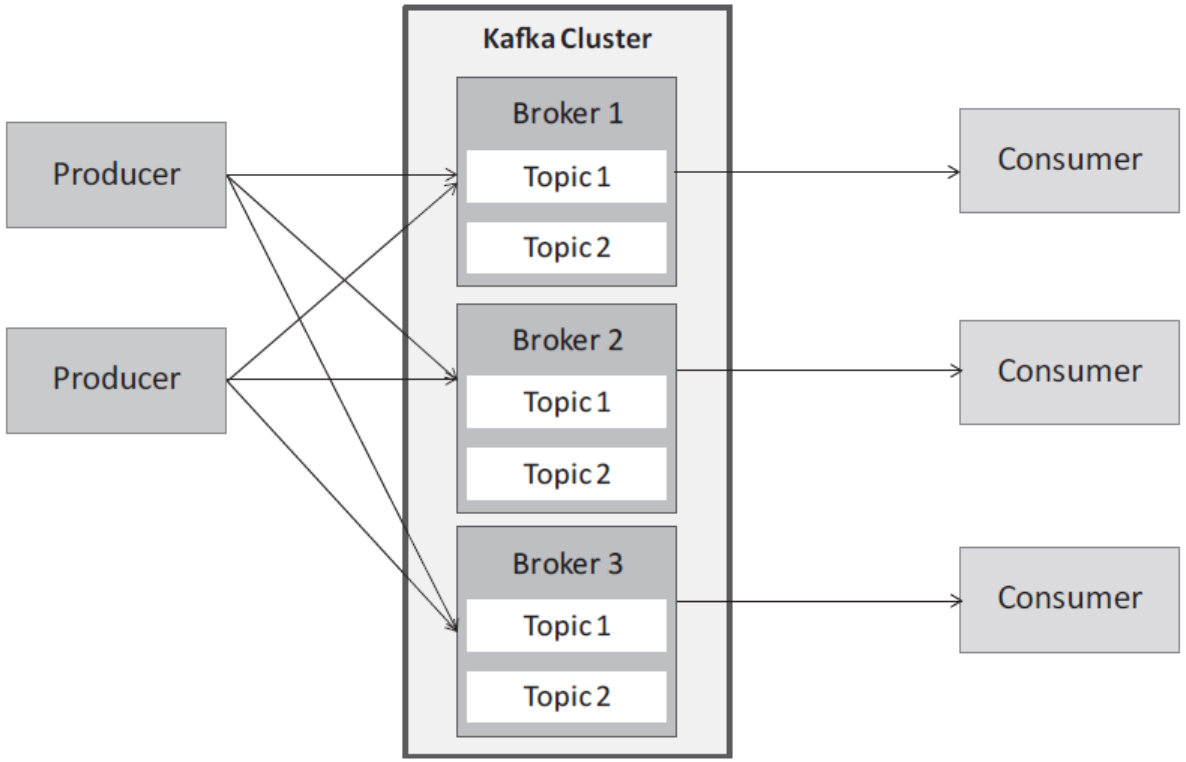
Publish - Subscribe Messaging Frameworks

- Suitable for **high velocity data**
- Feature **low overhead** and **low latency**
- Data can be **pushed** or **pulled** by the consumers
- Publish-subscribe messaging frameworks
  - Apache Kafka
  - Amazon Kinesis



Apache Kafka Components

- **Topic**: is a user-defined category to which messages are published.
- **Producer**: is a component that publishes messages to one or more topics.
- **Consumer**: is a component that subscribes to one or more topics and processes the messages.
- **Broker**: is a component that manages the topics and handles the persistence, partitioning, and replication of the data.
- A Kafka cluster can have **multiple Kafka Brokers** (or servers), with each Broker managing multiple topics.



Kafka Producer for sending messages

```
import time
from datetime import datetime
from kafka.client import KafkaClient
from kafka.producer import Producer

client = KafkaClient("localhost:6667")
producer = Producer(client)

while True:
    ts=time.time()
    timestamp = datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
    data = "This is a test string generated at: " + str(timestamp)
    producer.send_messages('topic-1', data)
    time.sleep(1)
```

Kafka Consumer

```
from kafka.client import KafkaClient
from kafka.consumer import Consumer

client = KafkaClient("localhost:6667")
consumer = Consumer(client, "topic-1")

for message in consumer:
    # Print message object
    print(message)
    # Print only message value
    print(message.message.value)
```

Outline

- Introduction
- Publish - Subscribe Messaging Frameworks
- Big Data Collection Systems
- Messaging Queues
- Custom Connectors

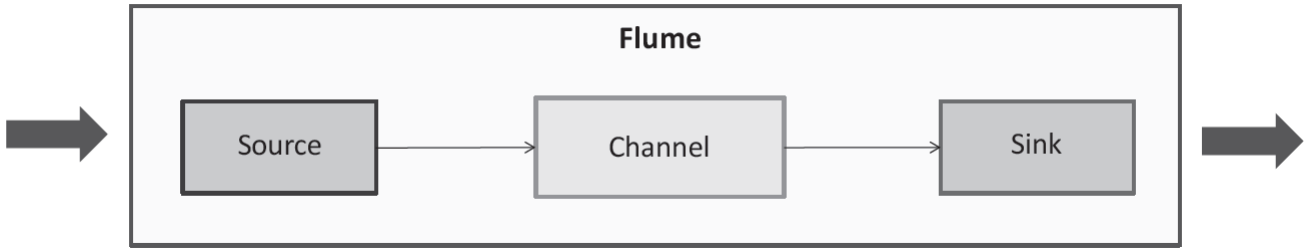
Big Data Collection Systems

- Big data collection systems allow **collecting, aggregating and moving data**:
  - **From** various sources
    - Server logs
    - Databases
    - Social media
    - Streaming sensor data from Internet of Things devices
  - **Into** a centralized data store
    - Distributed file system
    - NoSQL database

Example Frameworks

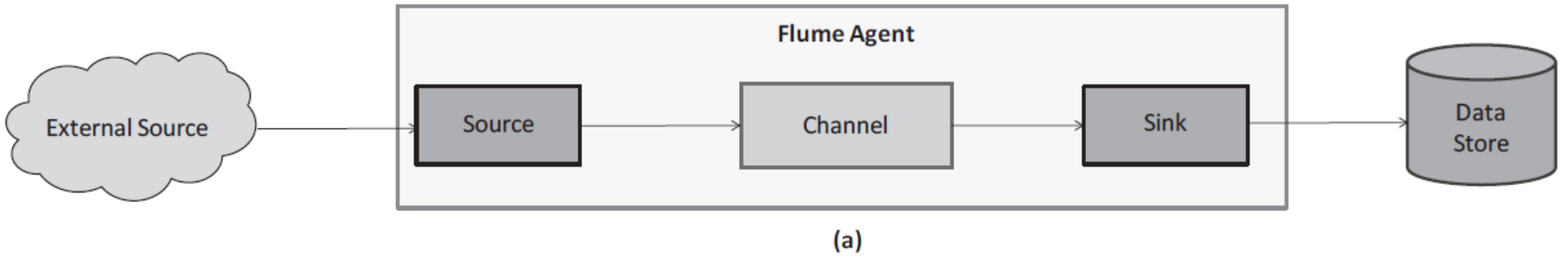
- Apache Flume
- Apache Sqoop

Apache Flume

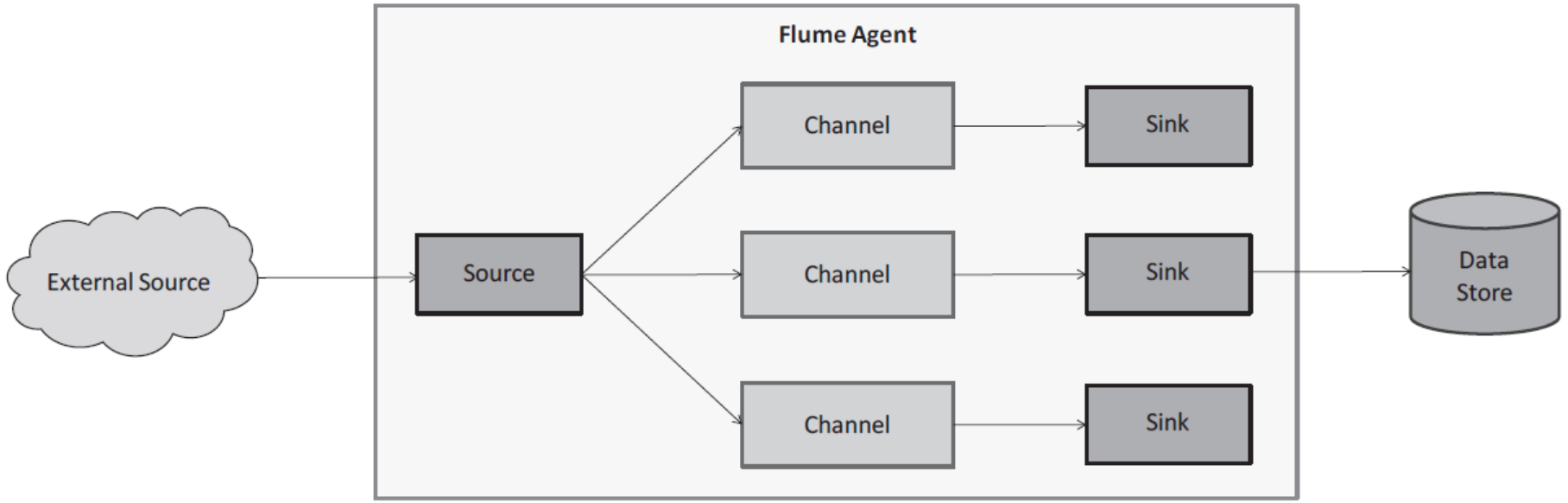


- **Distributed**, **reliable**, and **available** system for collecting, aggregating, and moving large amounts of data from different data sources into a centralized data store.
- **Source**: receives or polls for data from external sources.
- **Channel**: a source transmits data to one or more channels.
- **Sink**: drains data from a channel to a data store.
- **Agent**: collection of sources, channels and sinks.
- **Event**: is a unit of data flow having a payload and an optional set of attributes. Flume sources consume events generated by external sources.

Flume data flow examples



(a)



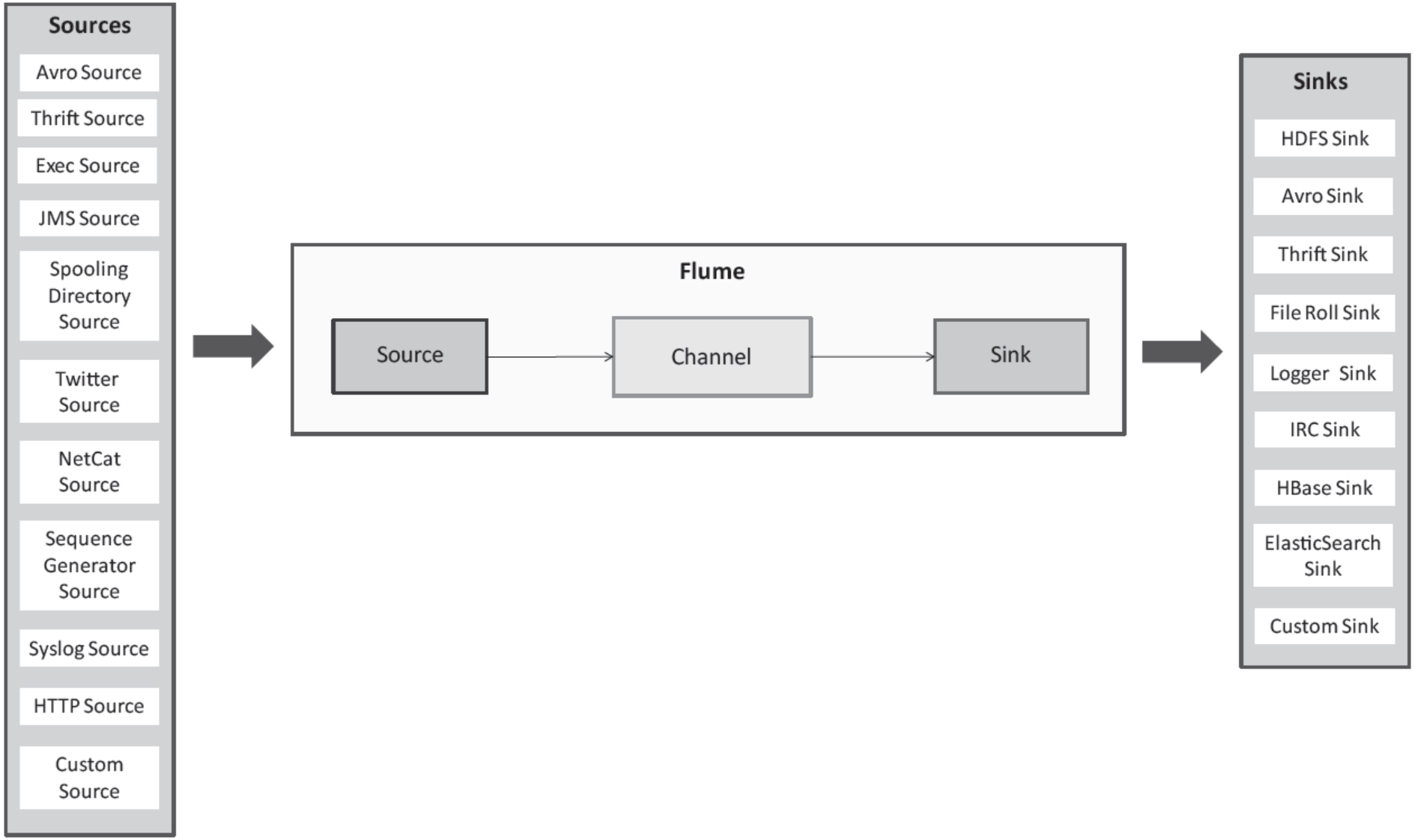
(b)

Generic definition of a Flume agent

```
<agent name>.sources = <source-1> <source-2> ... <source-N>
<agent name>.channels = <channel-1> <channel-2> ... <channel-N>
<agent name>.sinks = <sink-1> <sink-2> ... <sink-N>
# Define sources
<agent name>.sources.<source-1>.type = <source type>
:
# Define sinks
<agent name>.sinks.<sink-1>.type = <sink type>
:
# Define channels
myagent.channels.<channel-1>.type = <channel type>
:
# Bind the sources and sinks to the channels
myagent.sources.<source-1>.channels = <channel-1>
myagent.sinks.<sink-1>.channel = <channel-1>
:
```

The configuration file lists the sources, channels and sinks for the agent. Then defines each source, channel and sink. Finally, the binds sources, channels and sinks.

Apache Flume Sources and Sinks



Example Flume Sources

- **Spooling Directory Source**: ingests files such as log files. A spool directory is setup on the disk from where the Spooling Directory source ingests the files.

```
myagent.sources = source1
myagent.sources.source1.type = spooldir
myagent.sources.source1.spoolDir = /var/log/apache/flumeSpool
```

- **NetCat Source**: listens to a specific port to which the data is written by a NetCat client, which is a simple Unix utility that uses TCP or UDP protocol.

```
myagent.sources = source1
myagent.sources.source1.type = netcat
myagent.sources.source1.bind = 0.0.0.0
myagent.sources.source1.port = 6666
```

Example Flume Sink

- **HDFS Sink**: The Hadoop Distributed File System (HDFS) Sink drains events from a channel to HDFS.

```
myagent.sinks = sink1
myagent.sinks.sink1.type = hdfs
myagent.sinks.sink1.hdfs.fileType = DataStream
myagent.sinks.sink1.hdfs.path = /flume/events
```

Outline

- Introduction
- Publish - Subscribe Messaging Frameworks
- Big Data Collection Systems
- Messaging Queues
- Custom Connectors

Messaging Queues

- Messaging queues are useful for **push-pull** messaging where the **producers push** data to the queues, and the **consumers pull** the data from the queues.
- The producers and consumers do **not need to be aware of each other**.
- Messaging queues allow **decoupling** of producers of data from the consumers.
- Message queuing systems are based on
  - Advanced Message Queuing Protocol (AMQP)
  - ZeroMQ Message Transfer Protocol (ZMTP)

Example Frameworks

- RabbitMQ
- ZeroMQ
- RestMQ
- Amazon SQS

RabbitMQ

- RabbitMQ implements the **AMQP**.
- AMQP clients can either be **producers** or **consumers**.
- The clients can **communicate** with each other **through brokers**.
- The **producers publish messages** to the exchanges, which then distribute the messages to queues.
- AMQP brokers provide four types of **exchanges**:
  - **Direct exchange** (for point-to-point messaging)
  - **Fanout exchange** (for multicast messaging)
  - **Topic exchange** (for publish-subscribe messaging)
  - **Header exchange** (that uses header attributes for making routing decisions)

Outline

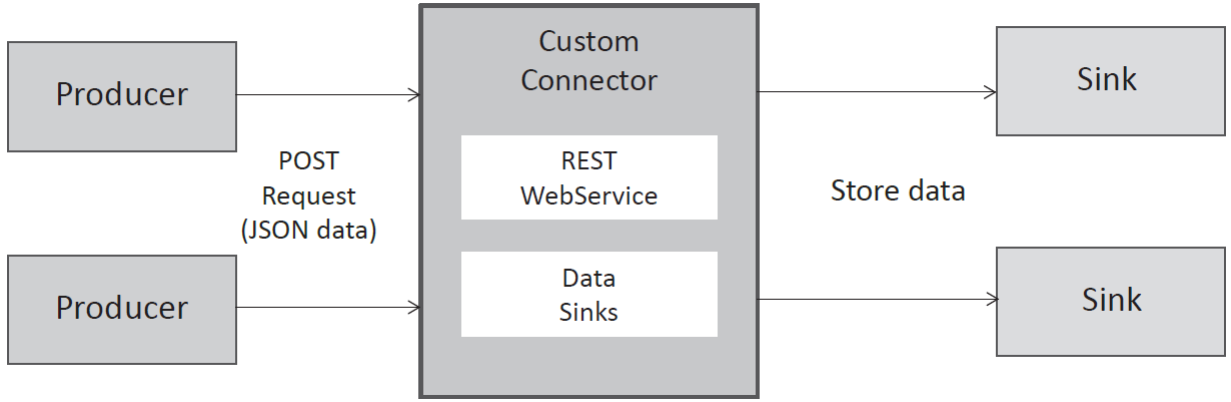
- Introduction
- Publish - Subscribe Messaging Frameworks
- Big Data Collection Systems
- Messaging Queues
- Custom Connectors

Custom Connectors

- Custom connectors and web services for acquiring data from data producers can be developed **to meet the application requirements**.
- Example Frameworks
  - REST-based Connectors
  - WebSocket-based Connectors
  - MQTT-based Connectors
  - Amazon IoT
  - Azure IoT Hub

REST-based Connectors

- **Representational state transfer** (**REST**) is a software architectural style which uses **web services** with resources in a **textual representation**, **stateless** protocol, and predefined operations.
- Producers **publish** data to the connector using **HTTP POST** requests which contain the data payload.
- REST-based connector enables **any client that can make HTTP requests** to send data to the connector.



Summary

- Introduction
- Publish - Subscribe Messaging Frameworks
- Big Data Collection Systems
- Messaging Queues
- Custom Connectors