SK Big Data Course 2019,
Big Data Architecture

Big Data Architecture

Designing Modern Streaming Data Applications

김영우 (Youngwoo Kim)
Hi-Tech DT 팀, Data Labs, ICT기술센터, SK 텔레콤

목차

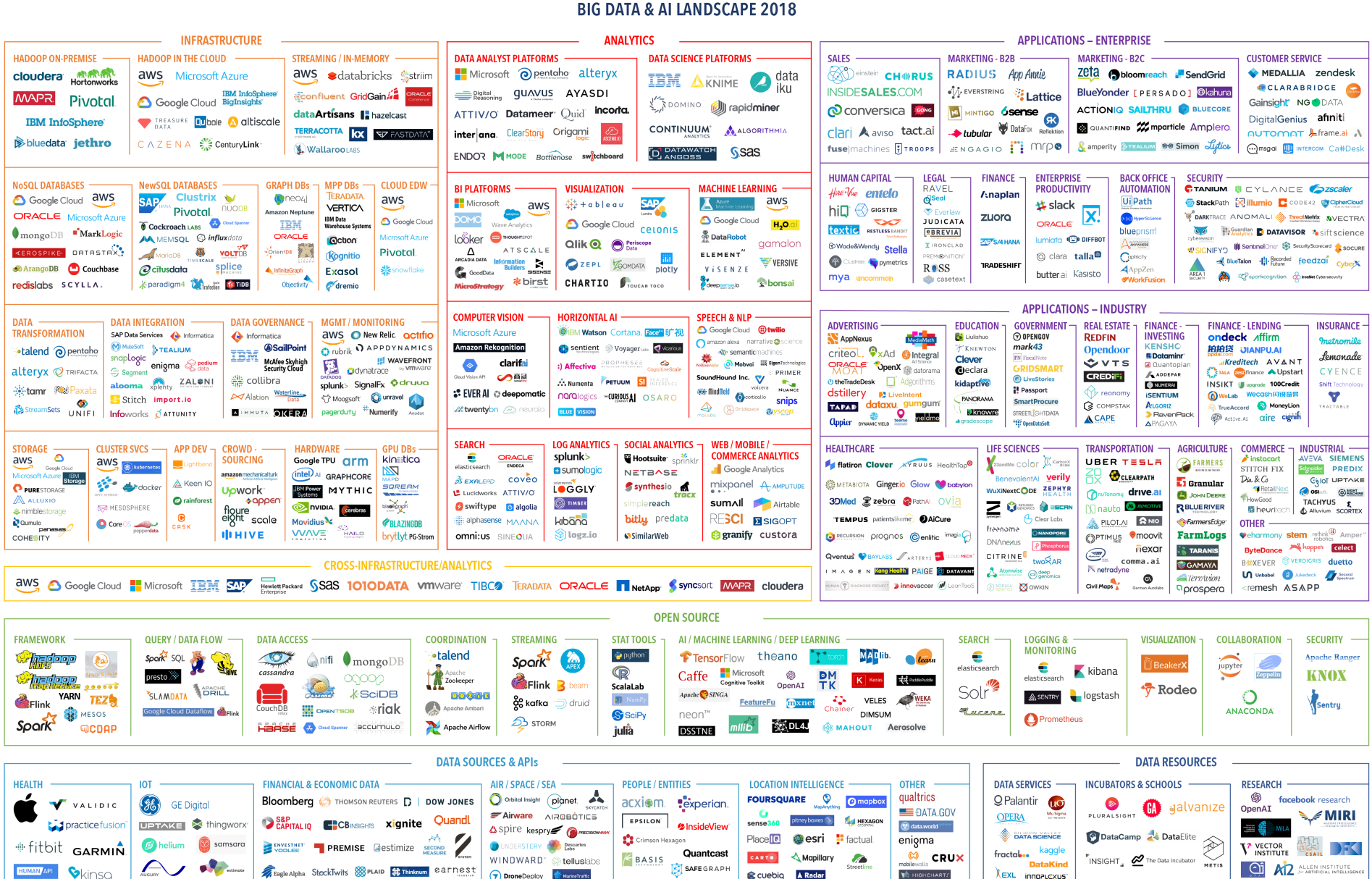
개요

- 스트리밍 데이터 처리 시스템을 구성하기 위해 필요한 구성 요소에 대하여 학습
- 실시간 데이터를 위한 오픈소스 프로젝트와 해당 프로젝트의 특징 학습
- 성공적인 데이터 응용 설계와 개발을 위한 고려사항
- 스트리밍 데이터 아키텍처 시나리오를 바탕으로 e2e 스트리밍 응용 개발 실습

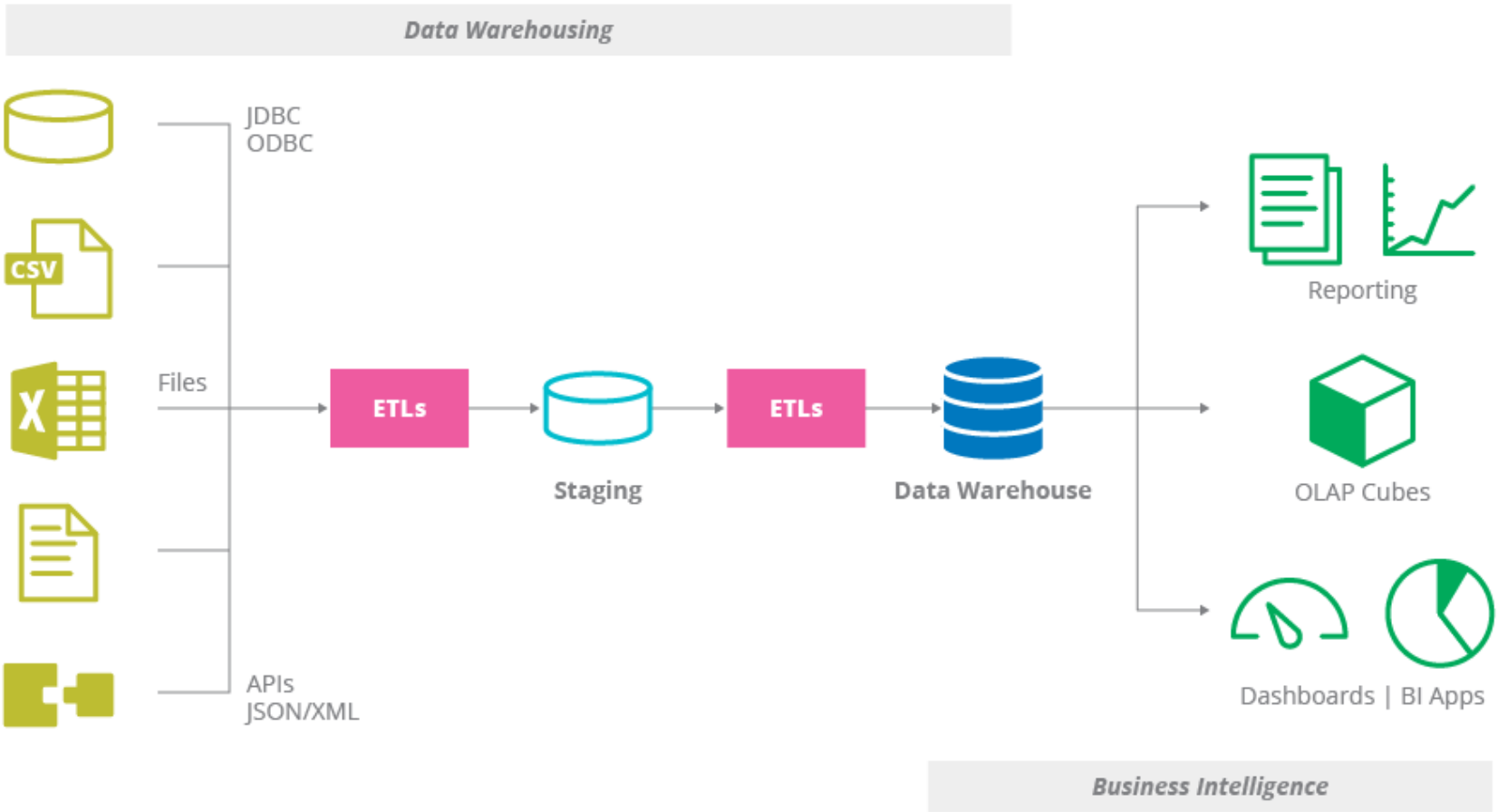
빅데이터 아키텍처

빅데이터?

Big Data & AI Landscape



Data Warehousing & Business Intelligence



Data Application?

데이터 애플리케이션

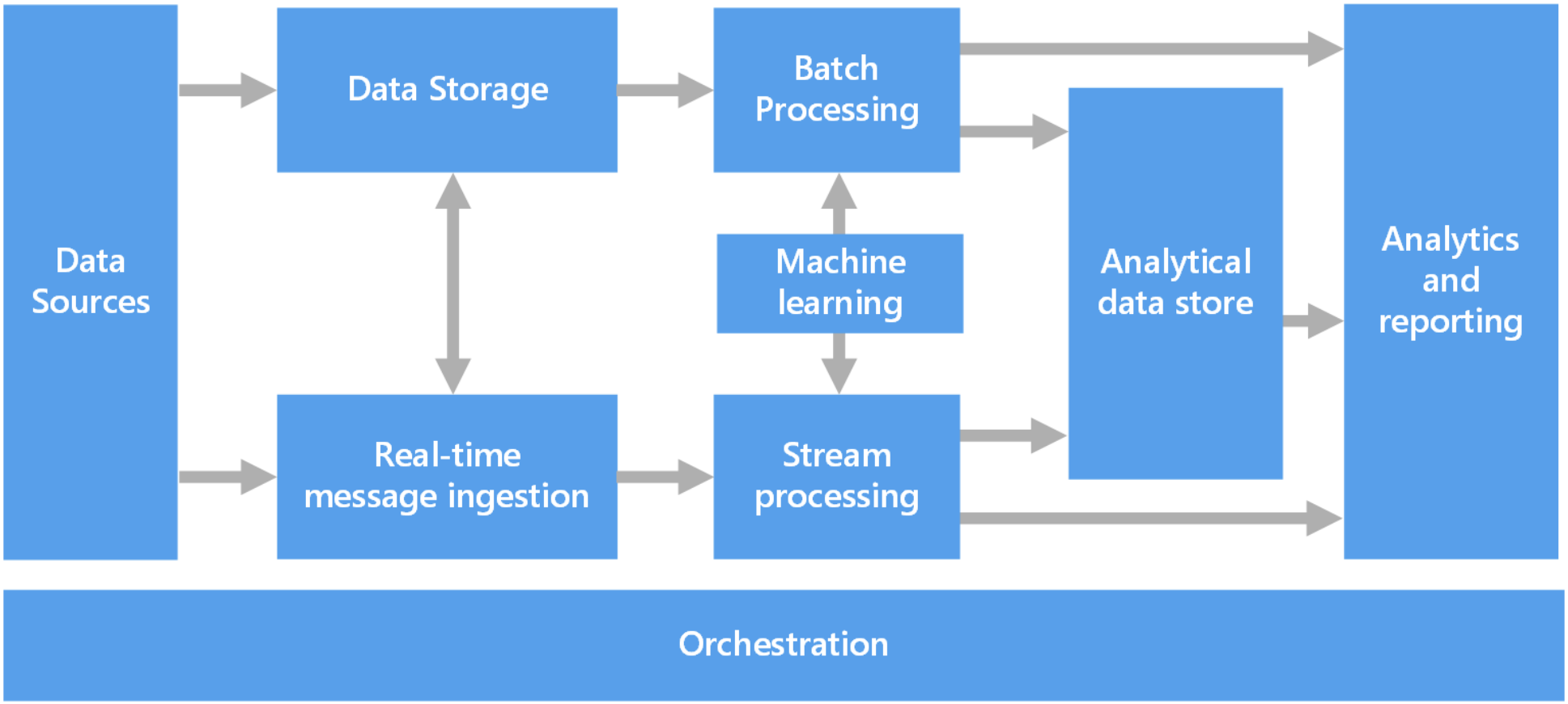
- 데이터 수집
- 데이터 저장
- 데이터 처리
- 데이터 분석 및 시각화

빅데이터 애플리케이션(시스템) 패턴

- 데이터 생성
- 데이터 수집
- 데이터 저장 / 처리
- 데이터 분석 / 시각화
- + 데이터 파이프 라인

빅데이터 아키텍처: 구성요소

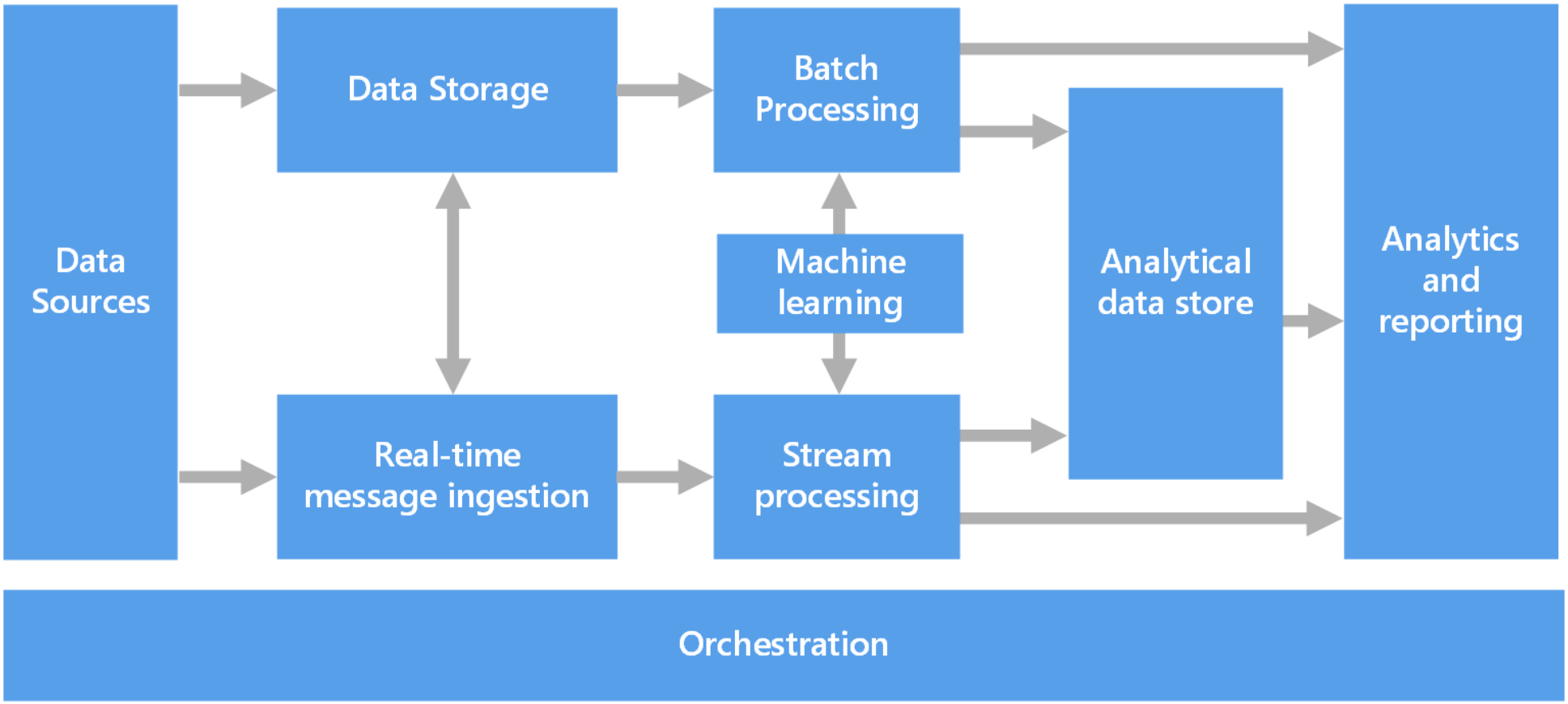
Components of a big data architecture [1]



1. https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/

스트리밍 데이터 응용을 위한 빅데이터 아키텍처

Components of the `Streaming Data Architecture`



`Big Data Architecture` - (minus) `Batch Processing` = ?

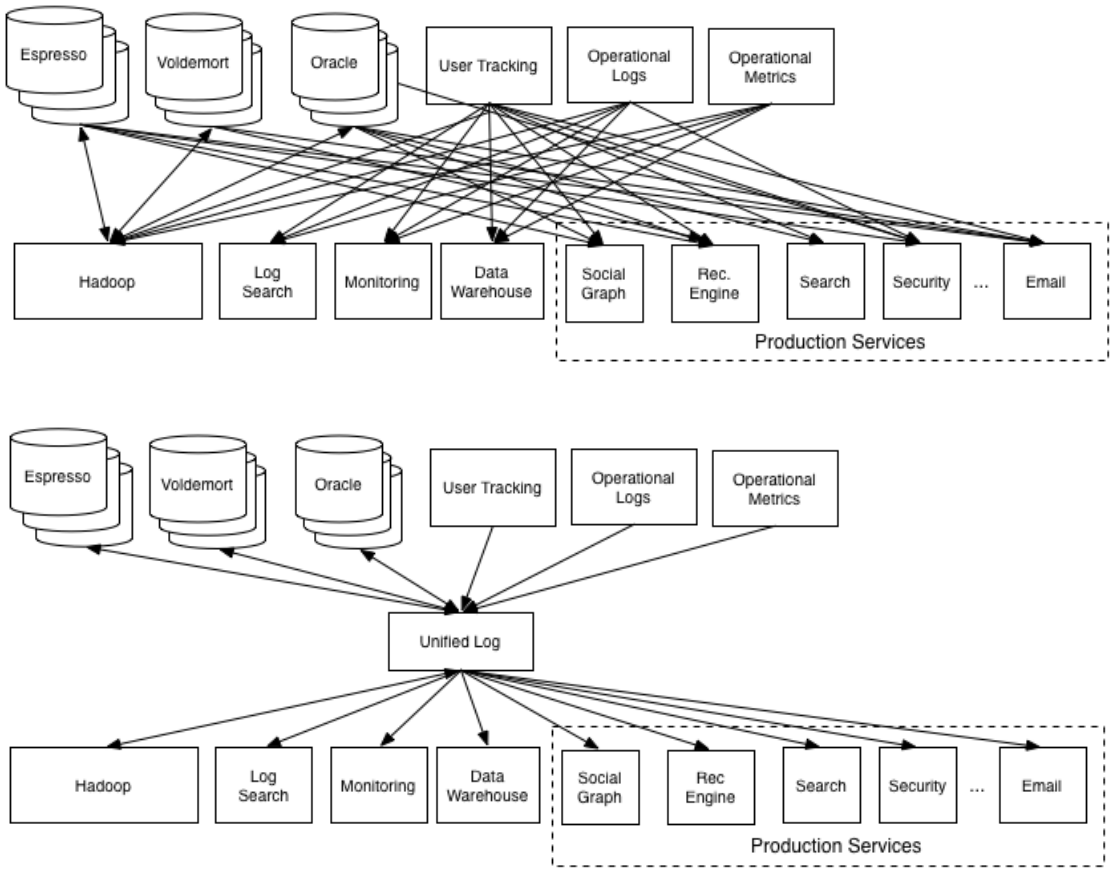
Streaming Data Platform

`Fast Data Architecture` [1]

- Data Sources
- Event Bus (a.k.a Event Hub, Message Broker)
    - Transport Layer
- Data Ingestion / Integration
- Data Storage
- Stream Processing
- Data Analytics, SQL, Dashboard, Search & etc.

1. Fast Data Architectures for Streaming Applications (2nd Edition)

Streaming Data Architecture: Bad and Good [1]



1. The Log: What every software engineer should know about real-time data's unifying abstraction

(Streaming) Data Sources

- Log files
- CDC (Transactions)
- Events (Equipments, Sensors...)
- APIs
- Agents
- Microservices
- .....

Streaming Data Platform: Event Hub

Message Broker?, Why?

> A message broker is an architectural pattern for message validation, transformation, and routing. It mediates communication among applications, minimizing the mutual awareness that applications should have of each other in order to be able to exchange messages, effectively implementing decoupling. [1]

- Event-driven Applications
- Asynchronous Processing

[1] https://en.wikipedia.org/wiki/Message_broker

Enterprise Event Hub

- a.k.a Enterprise Event Bus
- Immutable append-only data store

Goals for Enterprise Event Bus

- Buffering
- Reliable Storage
- Partitioning
- Replay
- High Throughput


엔터프라이즈 이벤트 버스 == 물류창고 ?

Key properties for Next-Generation Messaging

- High throughput
- Low latency
- Multi tenancy
- Consistency
- Ease of operations
- Resiliency
- Elasticity

Key properties for Next-Generation Messaging

Transmission

- Processing Guarantees (Messaging Semantics)
- Duplication
- Latency
- Ordering

Messaging semantics

- ▶️ message delivery semantics

At most once	At least once	Exactly once
Message pulled once	Message pulled one or more times; processed each time	Message pulled one or more times; processed once
May or may not be received	Receipt guaranteed	Receipt guaranteed
No duplicates	Likely duplicates	No duplicates
Possible missing data	No missing data	No missing data

Stream Processing

Requirements of Real-time stream processing [*]

- Supprting API, DSL, SQL
- Handling Delayed, Missing and Out-of-Order Data
- Integrate Stored and Streaming Data
- Guarantee Data Safety and Availability
- Partition and Scale Applications Automatically

* The 8 Requirements of Real-Time Stream Processing

(Analytical) Data Store

- **Application characteristics**
- Performance
- Efficiency
- Cost Efficiency
- Quality

One size fits all?

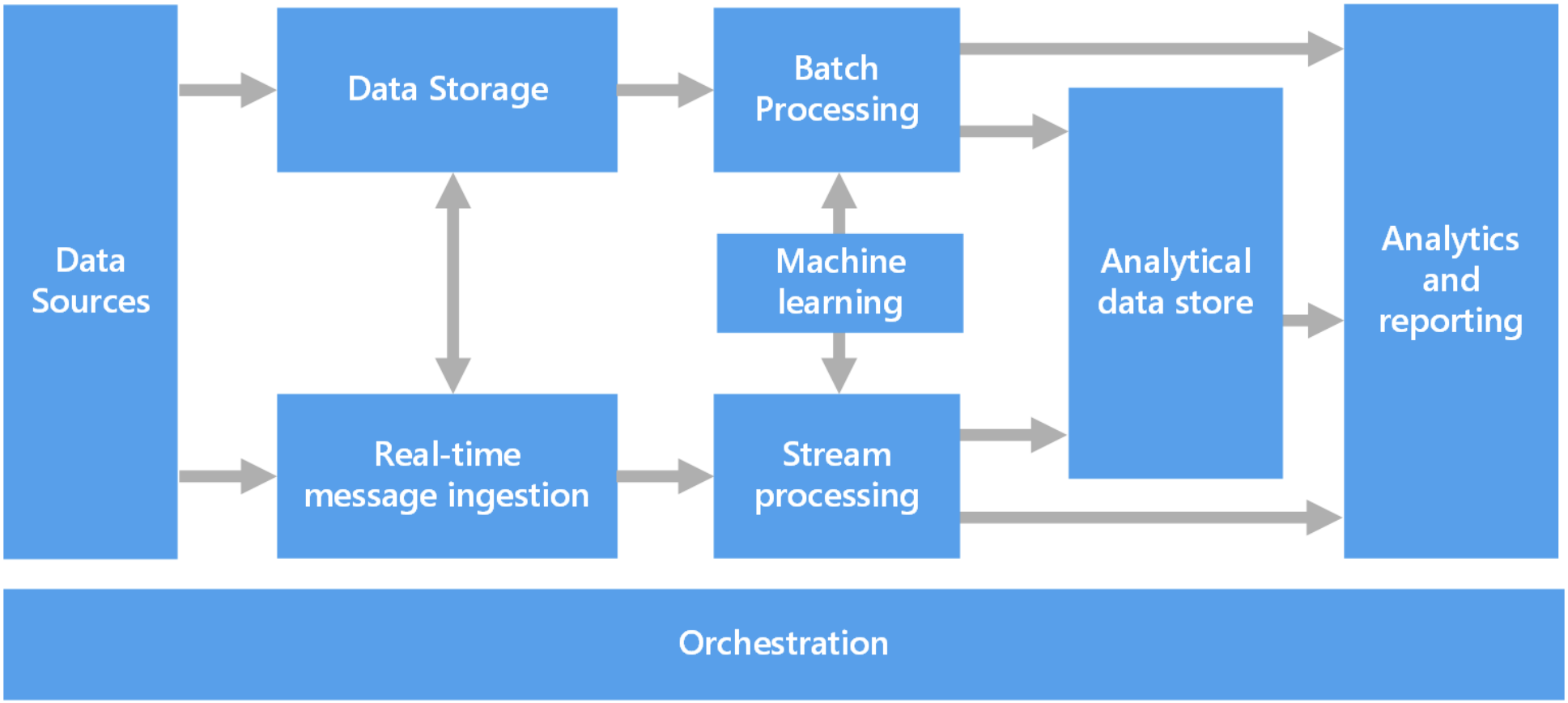
- RDBMS
- Distributed File System
- Object Storage
- Key/Value Store
- Search Engine
- Time Series Database
- OLAP
- Event Bus

Analytics / SQL / Dashboard & etc.

- Data Applications
  - API
- Querying Data
  - SQL
- Notebooks
  - Jupyter Notebooks
  - Apache Zeppelin
- Data Discovery
  - metatron discovery
- OLAP
- ML / DL
- Search
- Dashboard

OSS for Streaming Data Platform

Components of a `Streaming Data Architecture` (again!)



Data Source & Data Acquisition

- Data Source
  ➡️ Gateway / DI
- Network
  ➡️ Gateway / DI
- Event Bus
  ➡️ .....

OSS for Data Source & Data Acquisition

- Apache Flume
- Apache Gobblin
- Kafka Connect
- Fluentd
- Logstash
- Data Integration tools...

OSS for Message Broker / Enterprise Event Hub

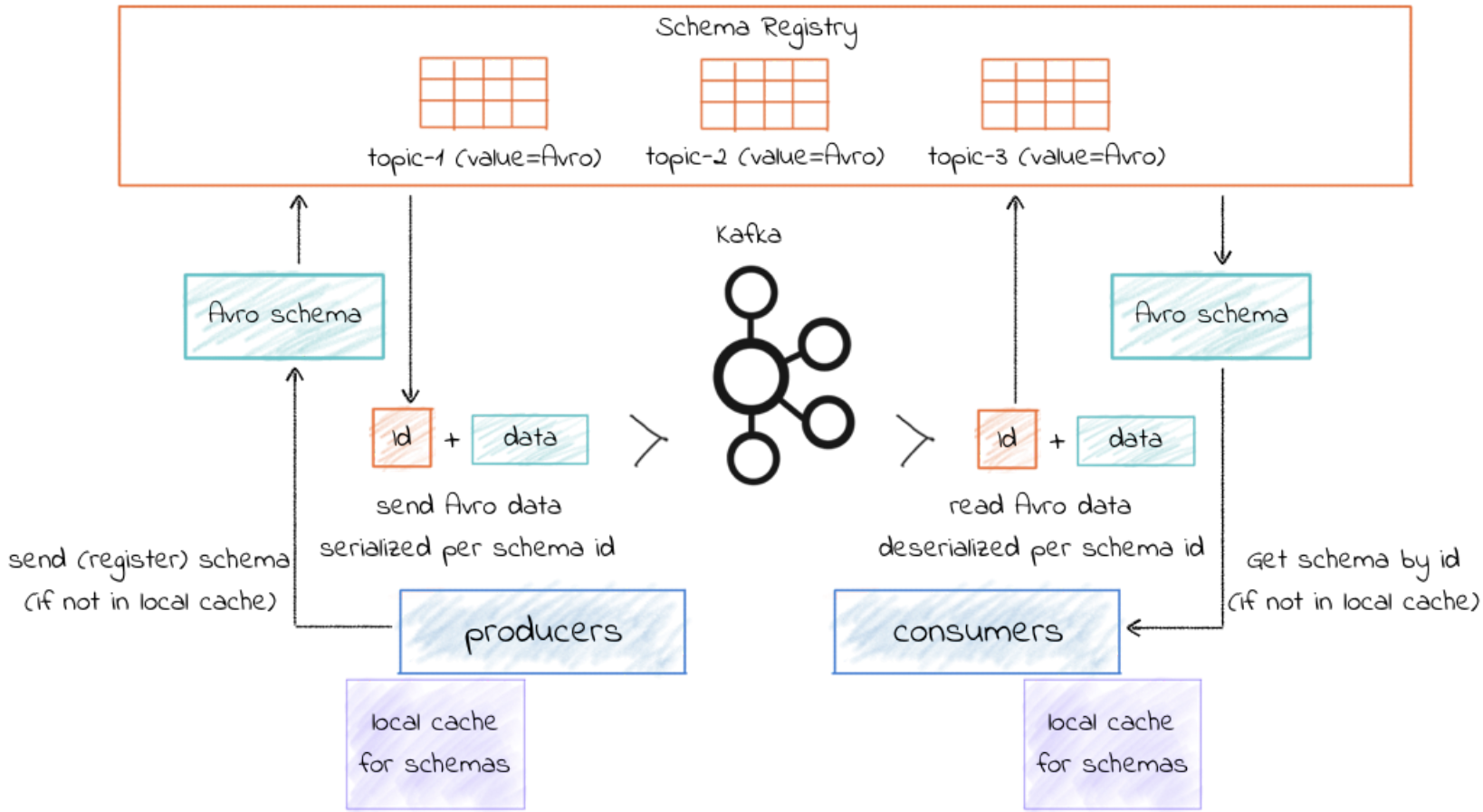
- Apache Kafka
- Apache Pulsar
- RabbitMQ
- Apache ActiveMQ
- Apache Qpid
- Apache RocketMQ
- ZeroMQ

Apache Kafka

- https://kafka.apache.org/
- High performance pub/sub messaging system
  - Broker
  - Streams
  - Connect
- Key terms
  - Message
  - Broker
  - Producer
  - Consumer
  - topic
  - topic partition
  - Consumer Group

Apache Kafka ❓ Confluent Platform ❓ 😕

Apache Kafka: Schema Management

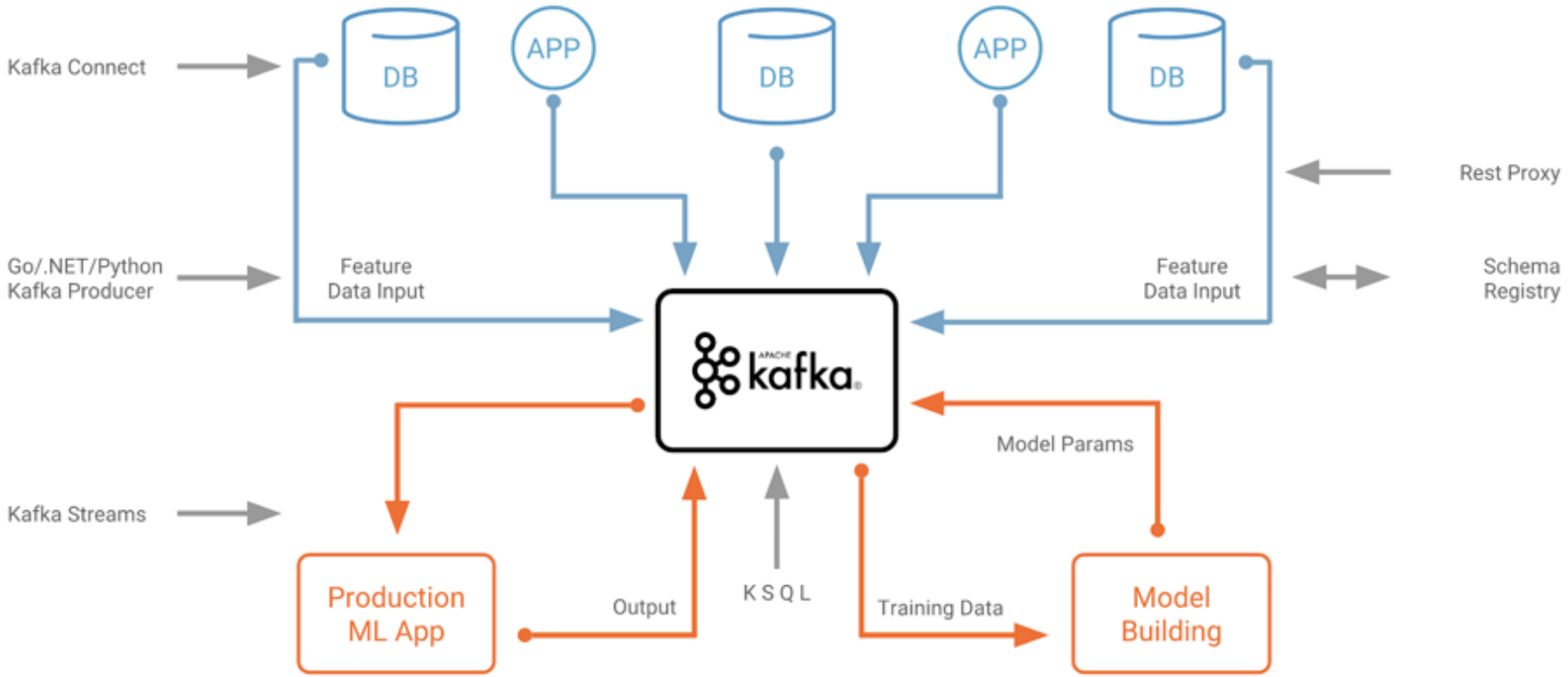


- Apache Kafka + Schema Registry
  - https://github.com/confluentinc/schema-registry
  - https://medium.com/@stephane.maarek/introduction-to-schemas-in-apache-kafka-with-the-confluent-schema-registry-3bf55e401321
- Why Avro For Kafka Data?
  - https://www.confluent.io/blog/avro-kafka-data/

Why binary format for event data (Avro or Protobuf)

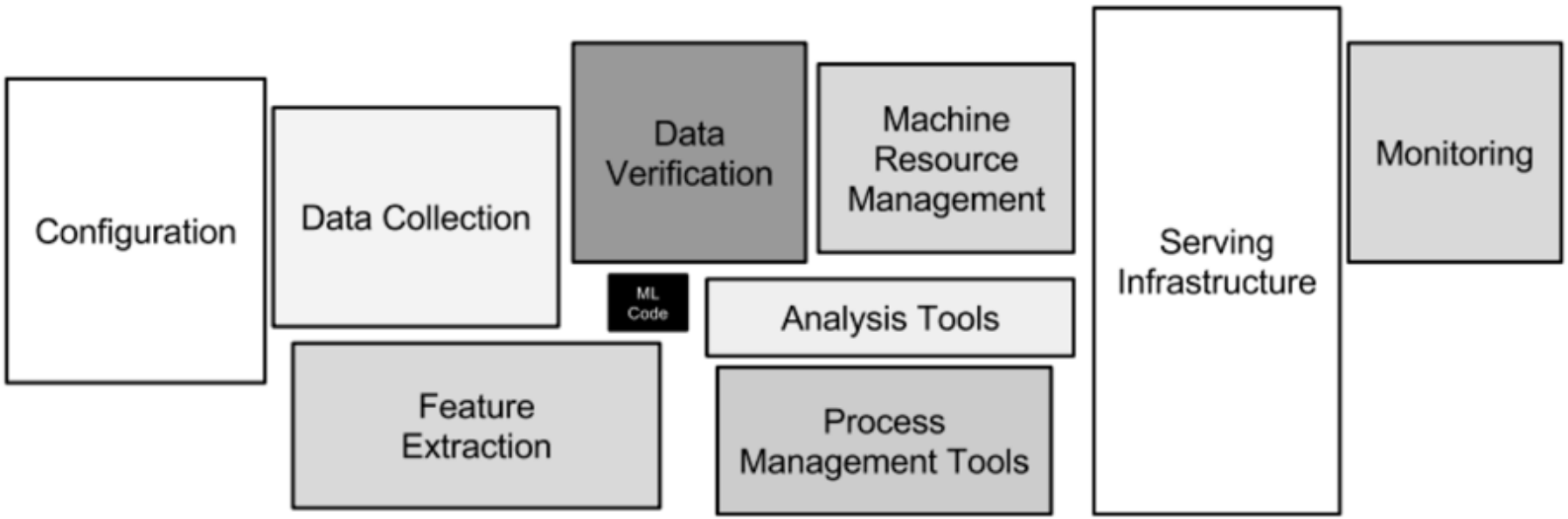
- Fast serialization / deserialization
- more compact messages
- have a schema of the data
- schema evolution
- .....
- Meta data and Data Quality

Machine learning and the Apache Kafka ecosystem



https://www.confluent.io/blog/using-apache-kafka-drive-cutting-edge-machine-learning

But Not enough for the Machine learning platform

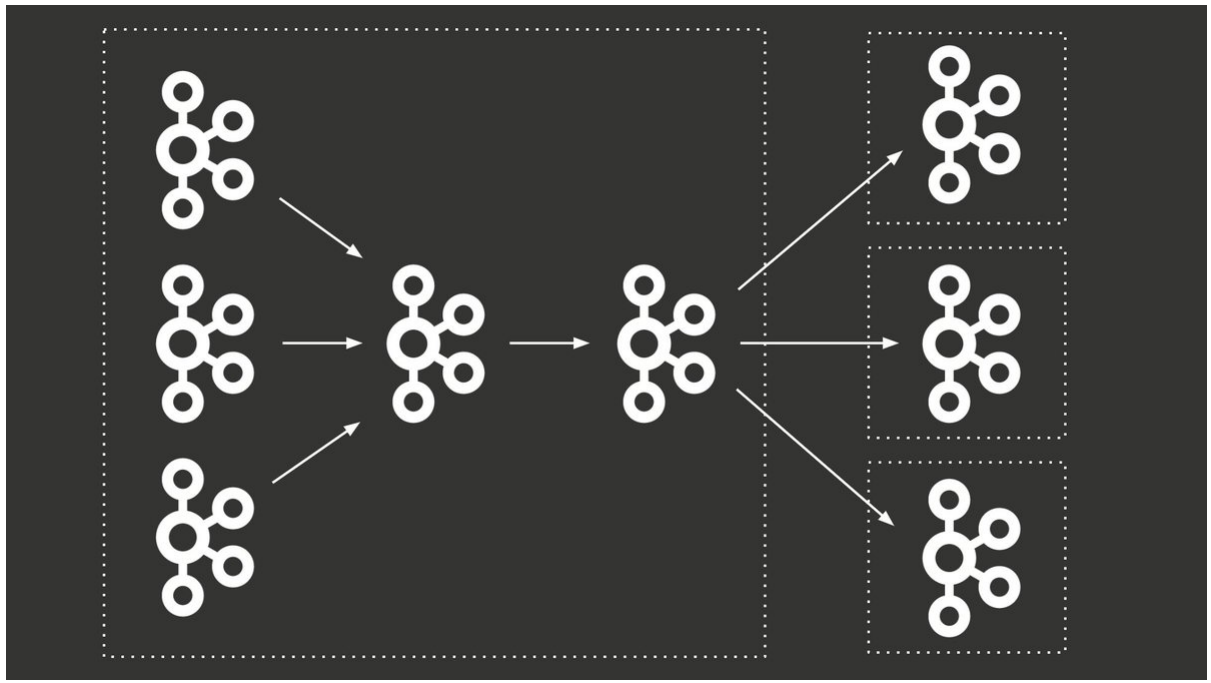


Scaling Apache Kafka

Tuning Kafka Broker, Producer and Consumer

- Async producer, acks, partitions, compression, Disk I/O ...

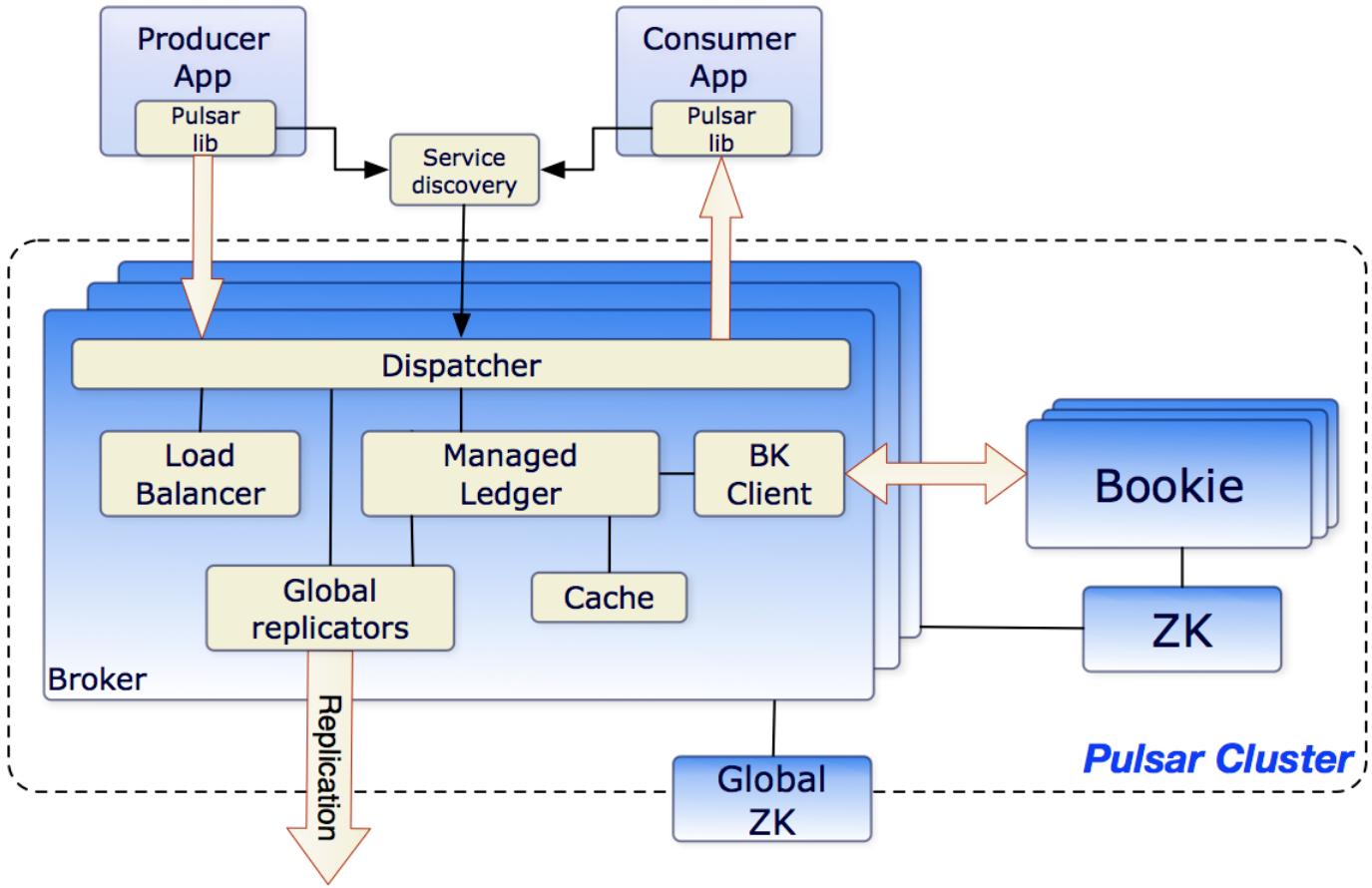
Tiered Kafka Cluster [1][2]



1. Kafka at Scale
2. Building Scalable and Extendable Data Pipeline for Call of Duty Games: Lessons Learned

Apache Pulsar

Two-Layer Architecture: Stateless broker layer + Stateful persistence layer



Apache Pulsar Architecture: Designing for Streaming Performance and Scalability

Components of Apache Pulsar

- Pulsar Cluster (one or more **brokers**)
- Apache Bookeeper
- Pulsar Functions
- Pulsar IO
- Pulsar SQL
- Schema Registry

Kafka vs Pulsar

OSS for Stream Processing

- Kafka client for Java, Scala, Python, Go, etc...
- Apache Spark
- Apache Flink
- Apache Samza
- Apache Storm
- Apache Heron
- Apache Samza
- Apache Beam
- Apache Bahir
- .....

OSS for Data Store

- RDBMS
- Distributed File System
  - Hadoop HDFS
- Object Storage
  - S3
  - Minio, Ceph
- KV Store
  - Apache HBase, Apache Cassandra
  - Redis, FoundationDB
- Search
  - Apache Solr, ElasticSearch
- OLAP
  - Apache Druid, Apache Pinot, Apache Kylin
- Event Bus
  - Apache Kafka, Apache Pulsar & MQs...
- etc.
  - CockroachDB, Apache Kudu...

File Format

File format for Data Store

- Distributed File System. ➡️ HDFS, QFS, & etc...
- Object Storage ➡️ S3 or S3 compatible Object Storage

Big Data File Formats

- Text (CSV, JSON)
- Apache Avro
- Apache ORC**
- Apache Parquet**

** Columnar storage ➡️ immutable data and analytics queries

OSS for Data Analytics / SQL / Dashboard & etc.

Data Analytics

- R
- Python, Java, Scala ...
- Jupyter Notebook, Apache Zeppelin
- ML/DL Libs.

SQL

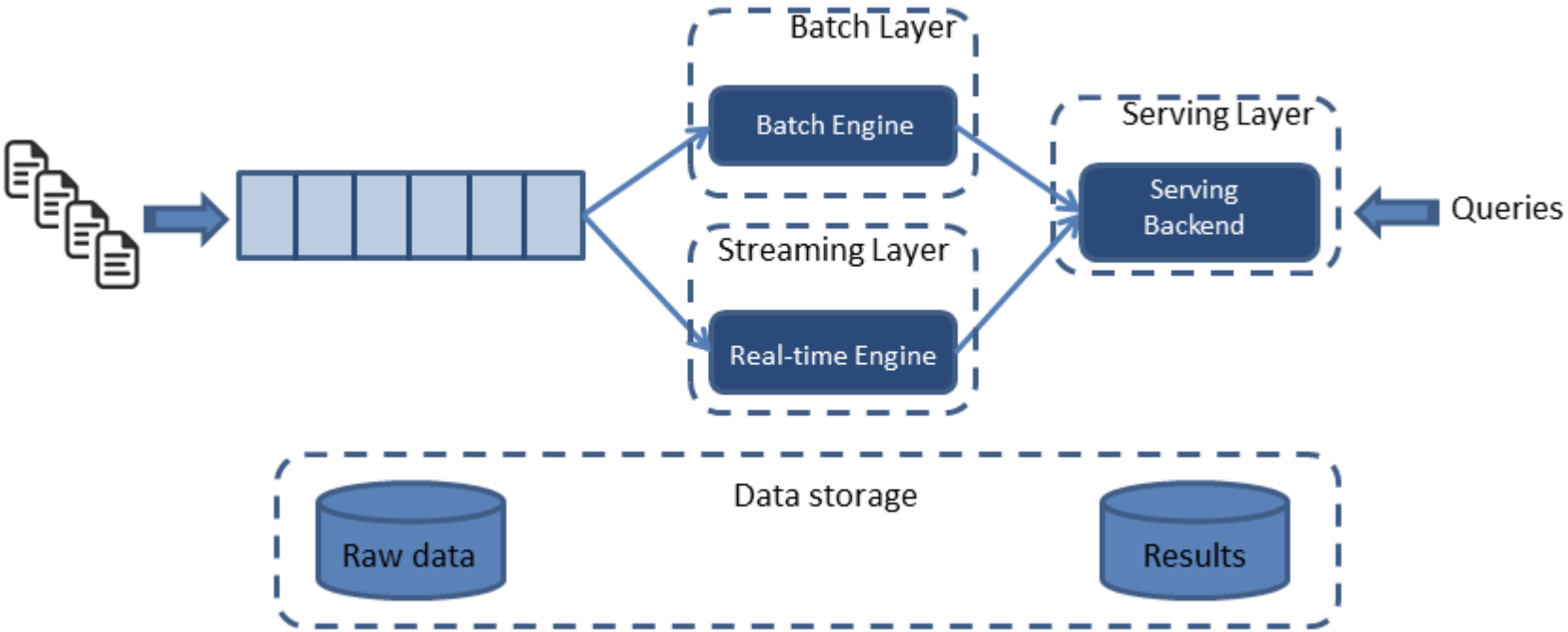
- .....

Etc.

- Search
  - ElasticSearch, Apache Solr
- Data Discovery tools
  - .....

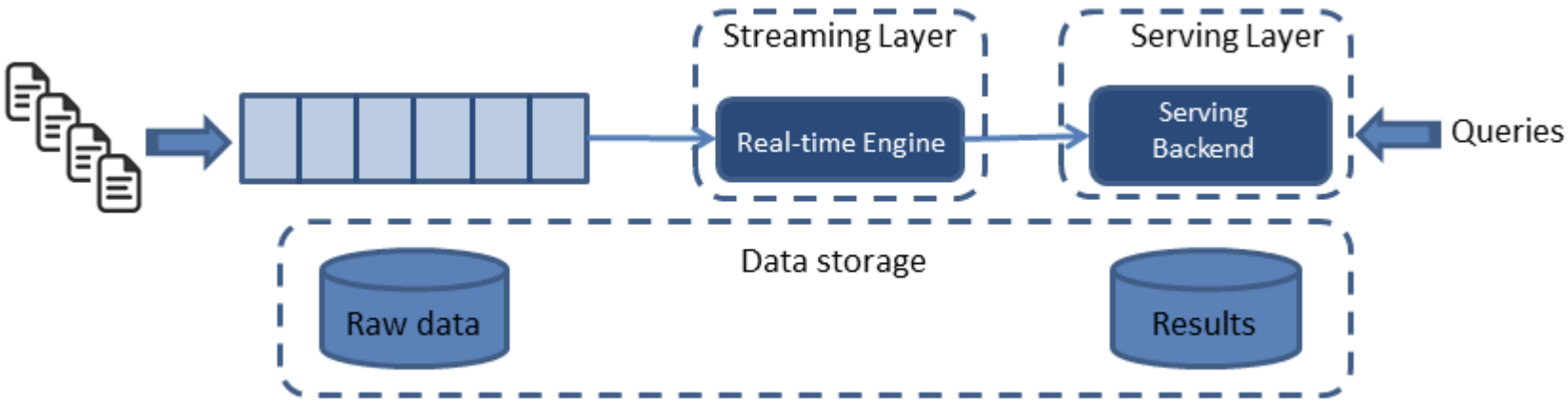
Lambda Arch. vs Kappa Arch.

Lambda Architecture



Lambda Arch. vs Kappa Arch.

Kappa Architecture



Stream analytics @ Public Cloud

- AWS
  - https://aws.amazon.com/ko/streaming-data/
- MS Azure
  - https://azure.microsoft.com/en-in/services/stream-analytics/
- GCP
  - https://cloud.google.com/solutions/big-data/stream-analytics/
- .....

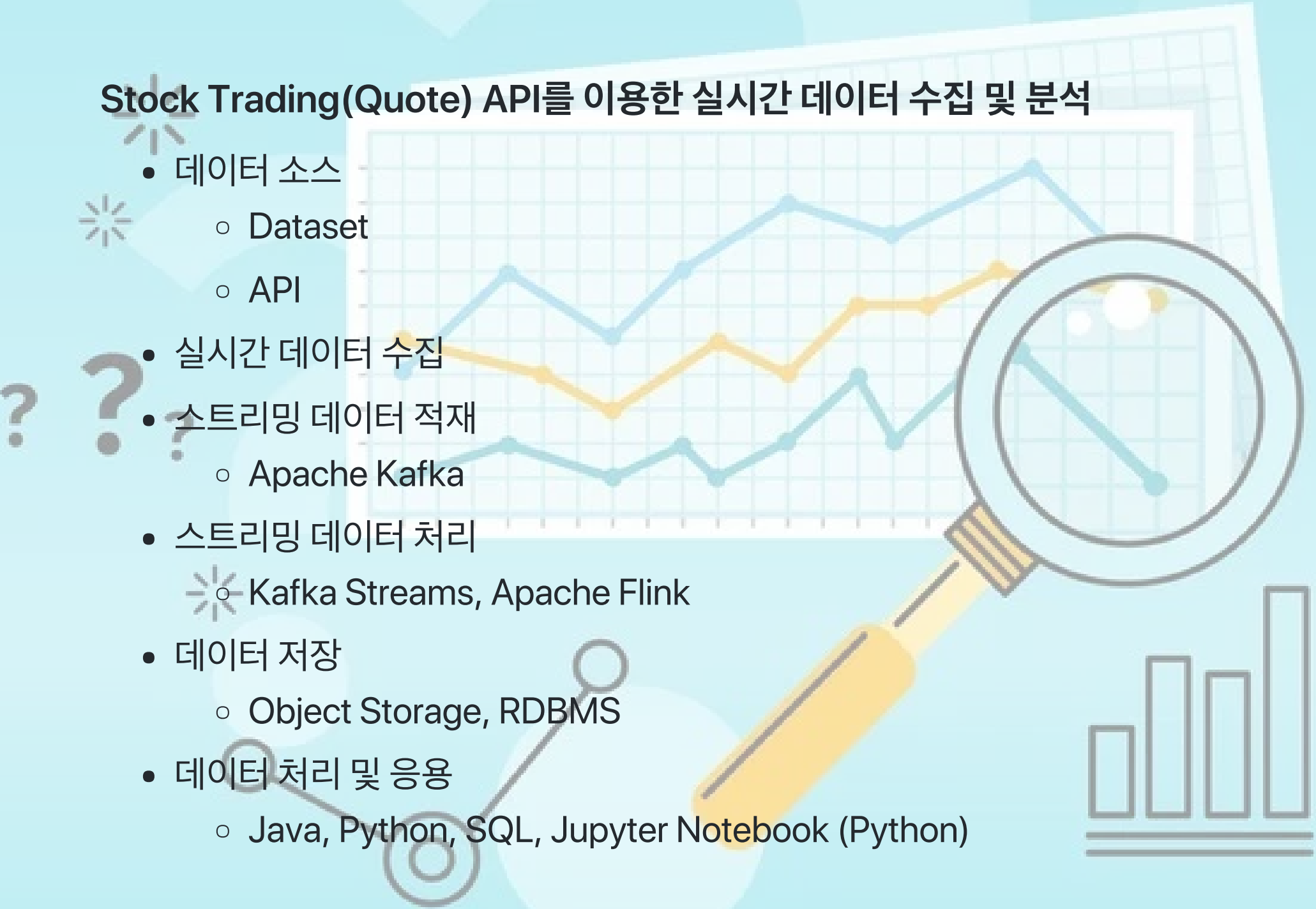
Big Data Architecture

Designing Modern Streaming Data Applications

Hands-on Labs

Stock Trading(Quote) API를 이용한 실시간 데이터 수집 및 분석

- 데이터 소스
  - Dataset
  - API
- 실시간 데이터 수집
- 스트리밍 데이터 적재
  - Apache Kafka
- 스트리밍 데이터 처리
  - Kafka Streams, Apache Flink
- 데이터 저장
  - Object Storage, RDBMS
- 데이터 처리 및 응용
  - Java, Python, SQL, Jupyter Notebook (Python)



Prerequisites

- Git
  - https://git-scm.com/downloads
- Docker & Docker Compose
  - https://www.docker.com/products/docker-desktop
- Java SE Development Kit 8 (a.k.a Java 8 / JDK 8)
  - https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- 텍스트 에디터, e.g., VIM, notepad++
- (선택) 선호하는 Java IDE, e.g., Eclipse, IntelliJ
- (선택) Python 3.x

OS 환경

Linux or MacOS

- RHEL7/CentOS7, Ubuntu 18.04+
- MacOS 10.x.y

Windows

Windows + Virtual Machine(via VirtualBox)

Sanity check

Git

```
$ git --version
git version 2.19.2
```

Docker & Docker compose

```
$ docker --version
Docker version 18.09.0, build 4d60db4
$ docker-compose --version
docker-compose version 1.23.2, build 1110ad01
```

Java8 (JDK8)

```
$ java -version
java version "1.8.0_192"
Java(TM) SE Runtime Environment (build 1.8.0_192-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.192-b12, mixed mode)
```

(선택) Maven

```
$ mvn -version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-11T01:41:47+09:00)
Maven home: /usr/local/Cellar/maven@3.3/3.3.9/libexec
Java version: 1.8.0_192, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_192.jdk/Contents/Home/jre
Default locale: ko_KR, platform encoding: UTF-8
OS name: "mac os x", version: "10.14.1", arch: "x86_64", family: "mac"
```

(선택) python

```
$ python --version
Python 3.4.0
```

Misc.

IP Address

```
$ ip addr show
$ ipconfig getifaddr en0
```

실습을 위한 데이터와 파일

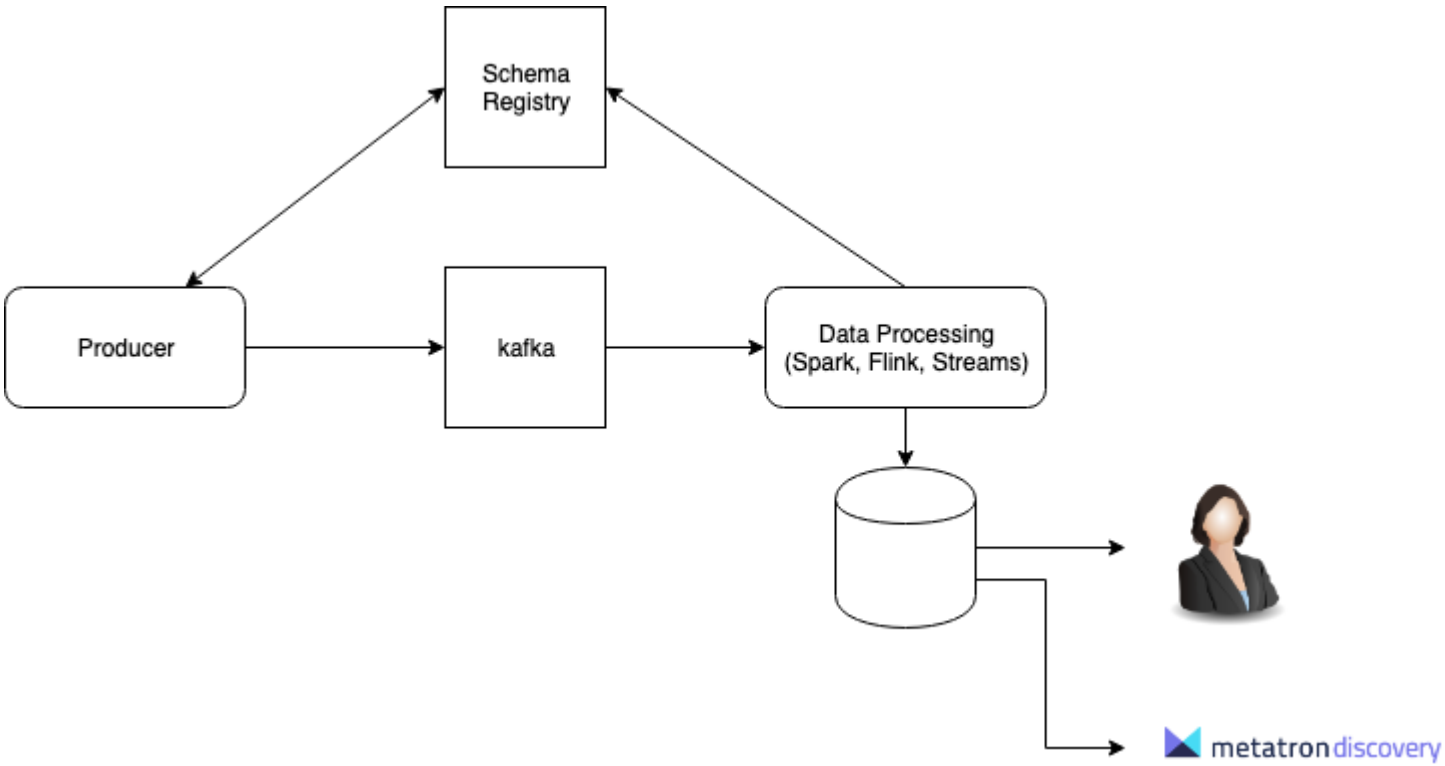
https://github.com/youngwookim/skbdc-bda-2019

```
$ mkdir -p /path/to/workspace
$ cd /path/to/workspace
$ git clone https://github.com/youngwookim/skbdc-bda-2019.git
$ cd skbdc-bda-2019
$ ls -als
```

or

Download https://github.com/youngwookim/skbdc-bda-2019/archive/master.zip or
https://github.com/youngwookim/skbdc-bda-2019/archive/master.tar.gz

시스템 구성


Schema Registry
Producer
kafka
Data Processing (Spark, Flink, Streams)
metatron discovery

상세 데이터 흐름

Source

- Kafka Client (Producer)
- Avro messages

Event Hub (Apache Kafka)

Stream Processing

- Apache Flink
- Kafka Streams

Data Store

- Log Data Store (Event Bus)
- RDBMS
- Object Storage (Minio)

Data processing and analytics

- Python (jupyter notebook)
- SQL (Hive, Presto, etc...)

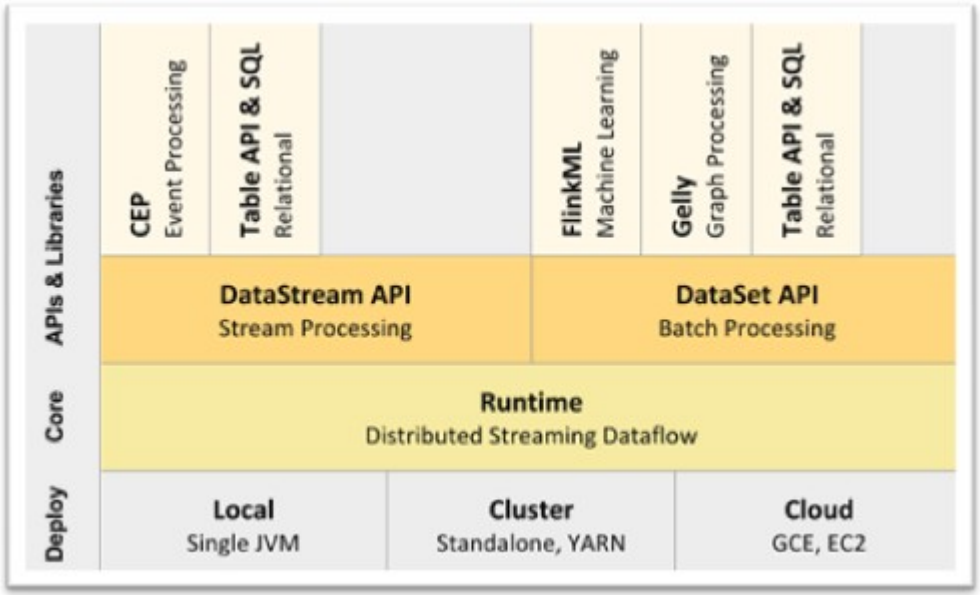
소프트웨어 스택

- Apache Zookeeper 3.4.9
- Apache Kafka 2.0.0
- Kafka Schema Registry 5.0.0 (confluent)
- Kafka Schema Registry UI 0.9.4
- Apache Flink 1.8.0
- Presto 302
- Minio
- Apache Avro 1.8.2
- JupyterLab
- MySQL
- Tools
  - kadmin, https://github.com/BetterCloud/kadmin
  - Kafka Manager, https://github.com/yahoo/kafka-manager

Apache Flink

> Apache Flink is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams. Flink has been designed to run in all common cluster environments, perform computations at in-memory speed and at any scale.

Architecture



Apache Flink: Programs and Dataflows

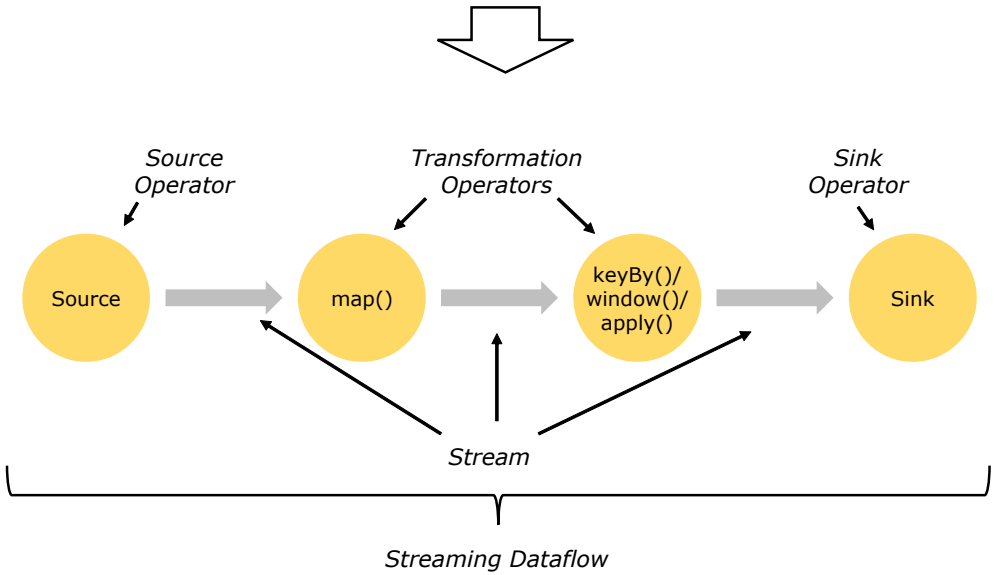
```
DataStream<String> lines = env.addSource(
                                new FlinkKafkaConsumer<>(…));

DataStream<Event> events = lines.map((line) -> parse(line));

DataStream<Statistics> stats = events
        .keyBy("id")
        .timeWindow(Time.seconds(10))
        .apply(new MyWindowAggregationFunction());

stats.addSink(new RollingSink(path));
```

Source

Transformation

Transformation

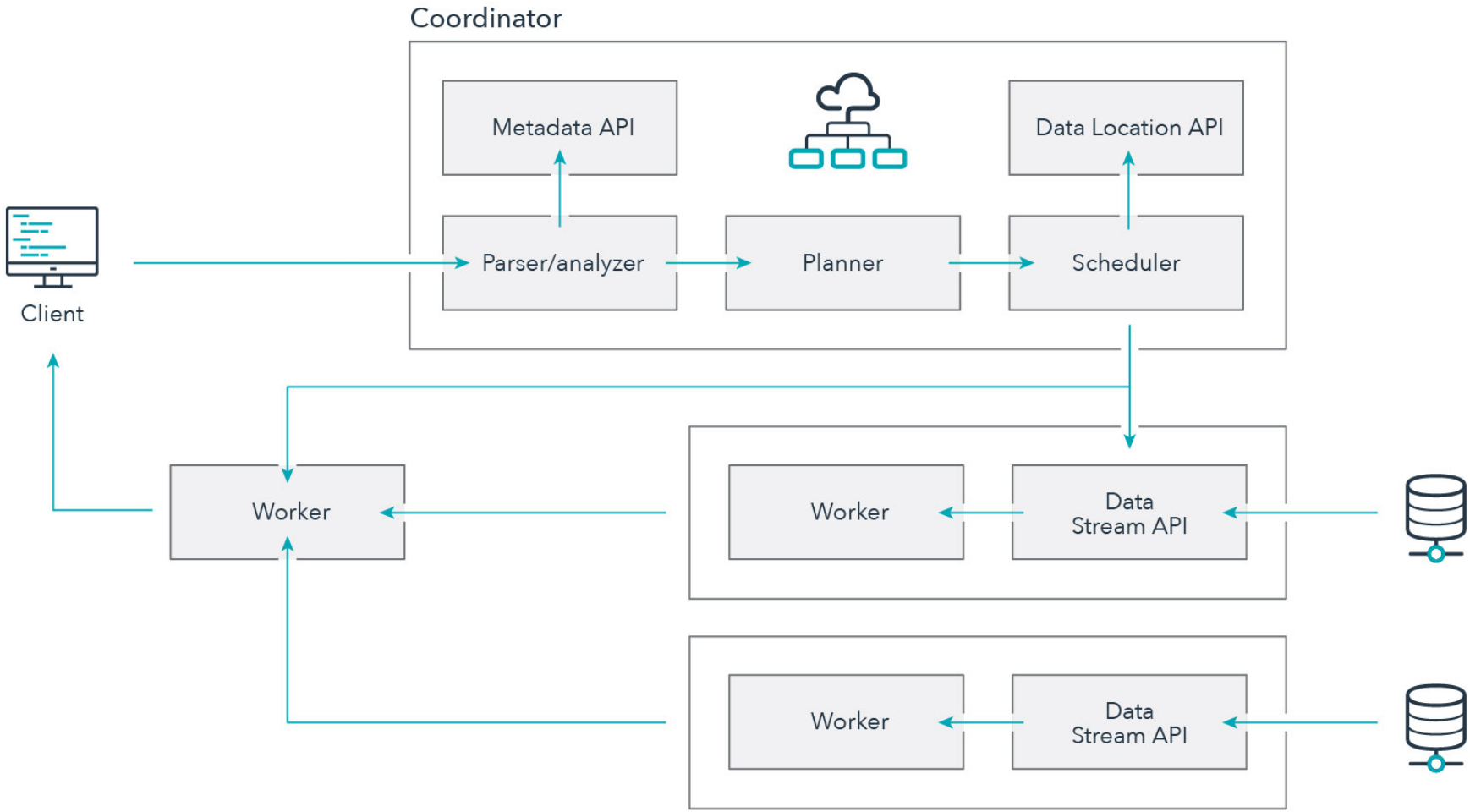
Sink



Presto (a.k.a PrestoSQL)

> Presto is an open source distributed SQL query engine for running interactive analytic queries against data sources of all sizes ranging from gigabytes to petabytes.

- http://prestosql.io



Metatron Discovery

- Intro. Metatron Discovery
    - https://www.youtube.com/channel/UC5ldHK8qBiN9zVgD7SvO41g
- Demo
    - https://discovery.metatron.app/

Apache Spark?

Stream Data Platform 개발 환경

Docker 컨테이너

호스트 IP 주소 찾기

```
$ ipconfig getifaddr en0
192.168.1.4
```

Docker 컨테이너 실행

```
# Stop all running containers
$ docker stop $(docker ps -aq)

# Delete all containers
$ docker rm $(docker ps -a -q)

# Delete all images
$ docker rmi $(docker images -q)

$ docker network ls
$ docker network prune
```

```
$ cd labs/docker
$ export DOCKER_HOST_IP=192.168.1.4
$ cat docker-compose.yml

$ docker-compose up -d
$ docker-compose ps
```

mc (Minio client) 실행

```
$ docker pull minio/mc
$ docker run --net docker_default -it --entrypoint=/bin/sh minio/mc
```

```
/ # mc

(snip)
```

Edit `~/.mc/config.json` :

labs/minio/mc/config.json 참고

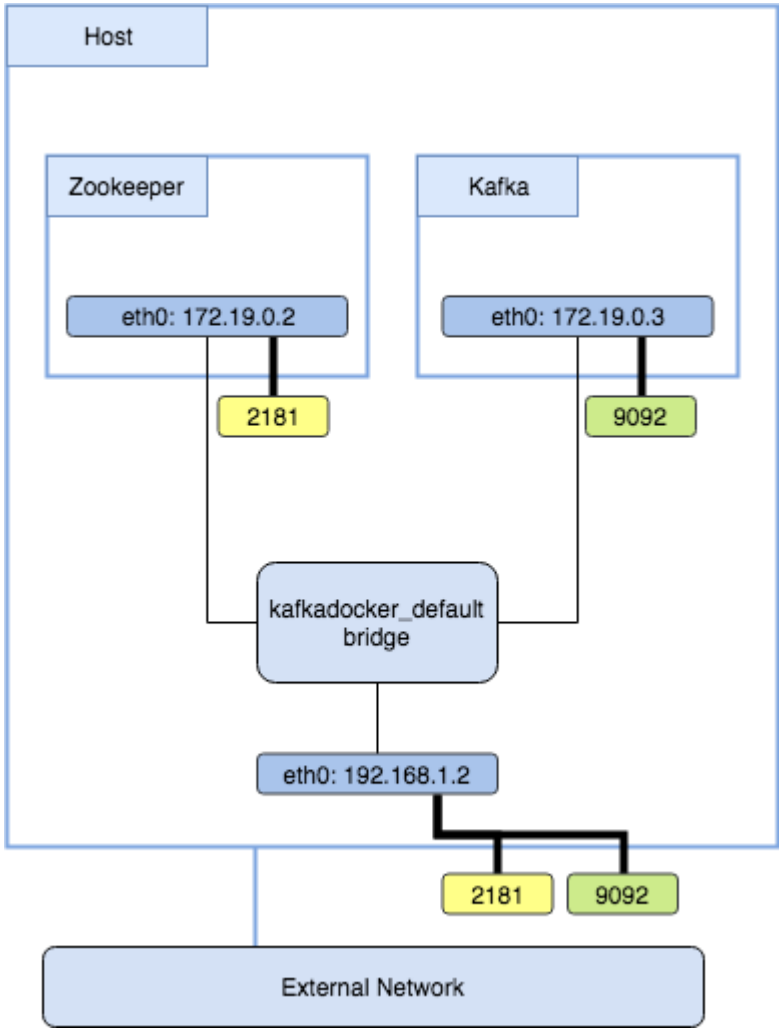
```
/ # vi ~/.mc/config.json
(snip)
                "local": {
                        "url": "http://minio:9000",
                        "accessKey": "V42FCGRVMK24JJ8DHUYG",
                        "secretKey": "bKhWxVF3kQoLY9kFmt91l+tDrEoZjqnWXzY9Eza",
                        "api": "S3v4",
                        "lookup": "auto"
                },
(snip)
```

```
/ # mc ls local/
[2019-06-24 05:34:07 UTC]      0B customer-data-json/
[2019-06-24 05:34:07 UTC]      0B customer-data-text/
[2019-06-24 05:34:07 UTC]      0B data/
/ #
```

Services

- Single Zookeeper: $DOCKER_HOST_IP:2181
- Single Kafka: $DOCKER_HOST_IP:9092
- Kafka Schema Registry: $DOCKER_HOST_IP:18081
- Kafka Schema Registry UI: $DOCKER_HOST_IP:8001
- Presto: $DOCKER_HOST_IP:8080
- Minio: $DOCKER_HOST_IP:9000

Docker Network



> Networking of (Kafka) docker compose, https://github.com/wurstmeister/kafka-docker/wiki/Connectivity

Sanity check: Apache Kafka

- `create` a topic

```
$ export KAFKA_BROKER=$(docker ps --filter name=kafka1 --format={{.ID}})
$ docker exec -t -i "$KAFKA_BROKER" \
kafka-topics --create --topic foo --partitions 1 --replication-factor 1 \
--if-not-exists --zookeeper zoo1:2181

$ docker exec -t -i "$KAFKA_BROKER" \
kafka-topics --create --topic hello --partitions 4 --replication-factor 1 \
--if-not-exists --zookeeper zoo1:2181

$ docker exec -t -i "$KAFKA_BROKER" \
kafka-topics --create --topic world --partitions 8 --replication-factor 1 \
--if-not-exists --zookeeper zoo1:2181
```

```
$ docker exec -t -i "$KAFKA_BROKER" \
kafka-topics --describe --topic foo --zookeeper zoo1:2181

$ docker exec -t -i "$KAFKA_BROKER" \
kafka-topics --describe --topic hello --zookeeper zoo1:2181

$ docker exec -t -i "$KAFKA_BROKER" \
kafka-topics --describe --topic world --zookeeper zoo1:2181
```

- `delete` a topic

```
$ export KAFKA_BROKER=$(docker ps --filter name=kafka1 --format={{.ID}})
$ docker exec -t -i "$KAFKA_BROKER" \
kafka-topics --zookeeper zoo1:2181 --delete --topic topicName
```

- `list` topic(s)

```
$ export KAFKA_BROKER=$(docker ps --filter name=kafka1 --format={{.ID}})
$ docker exec -t -i "$KAFKA_BROKER" \
kafka-topics --zookeeper zoo1:2181 --list
```

- etc...

`bin` scripts for Kafka

```
$ export KAFKA_BROKER=$(docker ps --filter name=kafka1 --format={{.ID}})
$ docker exec -t -i "$KAFKA_BROKER" bash -l
```

```
# dpkg -L confluent-kafka-2.11

# ls -als /usr/bin/kafka*
```

Test for Kafka producer & consumer

- `producer`

```
$ docker exec -t -i "$KAFKA_BROKER" \
bash -c "seq 100 | kafka-console-producer --request-required-acks 1 \
--broker-list kafka1:9092 --topic foo && echo 'Produced 100 messages.'"

>>>>> ...... >>>>>>Produced 100 messages.
```

- `consumer`

```
$ docker exec -t -i "$KAFKA_BROKER" \
kafka-console-consumer --bootstrap-server kafka1:9092 --topic foo \
--from-beginning --max-messages 100

1
2
3

......

99
100
```

kadmin: Web UI for Kafka producer/consumer

- https://github.com/BetterCloud/kadmin
- Running kadmin (on localhost)

```
$ cd /path/to/workspace
$ git clone https://github.com/BetterCloud/kadmin.git
$ cd kadmin
$ cd dist
$ cp ../application.properties .
```

- Edit Kadmin conf:

```
$ vi application.properties

server.port=9090
```

- Running kadmin:

```
$ java -jar shared-kafka-admin-micro-*.jar --spring.profiles.active=kadmin,local

......
...... : Tomcat started on port(s): 9090 (http)
...... : Started Application in 5.27 seconds (JVM running for 5.877)
```

Sanity check using kadmin

- Browse `kadmin` web, http://localhost:9090/kadmin/
- Create Kafka topics

```
$ export KAFKA_BROKER=$(docker ps --filter name=kafka1 --format={{.ID}})

# test_topic1 topic

$ docker exec -t -i "$KAFKA_BROKER" \
kafka-topics --create --topic test_topic1 --partitions 4 --replication-factor 1 \
--if-not-exists --zookeeper zoo1:2181

# eventcall topic

$ docker exec -t -i "$KAFKA_BROKER" \
kafka-topics --create --topic eventcall --partitions 4 --replication-factor 1 \
--if-not-exists --zookeeper zoo1:2181
```

- Basic producer: `String` -> `String`
- Avro producer: `JSON` -> `AVRO` -> `ByteArray`
  - Schema Registry UI: http://localhost:8001
- `EventCall` Avro Schema
  - eventcall.avsc
- EventCall message
  - eventcall.json

Kafka Manager

> A tool for managing Apache Kafka.

- https://github.com/yahoo/kafka-manager

Browse kafka-manager web:

- http://localhost:19000

How to add 'Local' cluster:

- Cluster > `Add Cluster`
- Cluster Name: `DEV`
- Cluster Zookeeper Hosts: `zoo1:2181`
- `Save`

데이터 소스

실습을 위한 데이터

NASDAQ symbols

- https://datahub.io/core/nasdaq-listings

nasdaq-listed

- https://datahub.io/core/nasdaq-listings/r/nasdaq-listed.csv

Field Name	Order	Type (Format)	Description
Symbol	1	string	
Company Name	2	string	
Security Name	3	string	
Market Category	4	string	
Test Issue	5	string	
Financial Status	6	string	
Round Lot Size	7	number	

nasdaq-listed-symbols

- https://datahub.io/core/nasdaq-listings/r/nasdaq-listed-symbols.csv

Field Name	Order	Type (Format)	Description
Symbol	1	string	
Company Name	2	string	

Stock Quote API

IEX Cloud API

- https://iexcloud.io/docs/api/
- IEX Cloud is a platform that makes financial data and services accessible to everyone.
- iextrading4j, https://github.com/WojciechZankowski/iextrading4j

E.g., Quote Request:

```
final IEXCloudClient iexTradingClient = IEXTradingClient.create(IEXTradingApiVersion.IEX_CLOUD_BETA_SANDBOX,
                new IEXCloudTokenBuilder()
                        .withPublishableToken("Tpk_18dfe6cebb4f41ffb219b9680f9acaf2")
                        .withSecretToken("Tsk_3eedff6f5c284e1a8b9bc16c54dd1af3")
                        .build());
final Quote quote = iexTradingClient.executeRequest(new QuoteRequestBuilder()
        .withSymbol("AAPL")
        .build());
System.out.println(quote);
```

> Quote{symbol=AAPL, companyName=Apple, Inc., primaryExchange=null, sector=null, calculationPrice=close, open=208.3, openTime=1565030786525, close=200.18, closeTime=1618408074295, high=207.68, low=199.32, latestPrice=198.97, latestSource=Close, latestTime=June 19, 2019, latestUpdate=1629414771181, latestVolume=21938828, iexRealtimePrice=200.09, iexRealtimeSize=104, iexLastUpdated=1579260527164, delayedPrice=198.86, delayedPriceTime=1592686867485, extendedPrice=198.9, extendedChange=0.54, extendedChangePercent=0.00277, extendedPriceTime=1614799731031, previousClose=203.47, change=-0.59, changePercent=-0.00296, iexMarketPercent=0.0236840881544615, iexVolume=496646, avgTotalVolume=31079954, iexBidPrice=0, iexBidSize=0, iexAskPrice=0, iexAskSize=0, marketCap=922850939808, peRatio=16.8, week52High=241.05, week52Low=143, ytdChange=0.2633361185170389, bidPrice=null, bidSize=null, askPrice=null, askSize=null}

IEX Cloud API, `Quote`

주식 시세

- https://iexcloud.io/docs/api/#quote
- https://github.com/WojciechZankowski/iextrading4j/blob/master/iextrading4j-api/src/main/java/pl/zankowski/iextrading4j/api/stocks/Quote.java

Test for IEX Trading API

```
$ cd labs/iextrading-test
$ ./mvnw clean package
$ ./mvnw exec:java -Dexec.mainClass="com.example.IexTradingTest"
```

Kafka Message Generator

Create `iextrading` Kafka topic for `Quote` data:

```
$ export KAFKA_BROKER=$(docker ps --filter name=kafka1 --format={{.ID}})
$ docker exec -t -i "$KAFKA_BROKER" \
kafka-topics --create --topic iextrading --partitions 4 --replication-factor 1 \
--if-not-exists --zookeeper zoo1:2181
```

Avro schema for IEX Cloud 'Quote':

- IEX Cloud `Quote` Avro 스키마 등록(선택)
  - labs/avro/iextrading.avsc

Running Kafka producer:

```
$ cd labs/kafka-message-gen
$ ./mvnw clean package
$ java -jar target/kafka-message-gen-1.0.0.jar
```

확인?

- (STDOUT) logs from 'kafka-message-gen'
- (console) Avro consumer
- (web) kadmin

Data Processing

Apache Flink Streaming Applicaltion

- Setup Flink local cluster
  - https://ci.apache.org/projects/flink/flink-docs-stable/tutorials/local_setup.html
- Download Flink tarball:
  - https://flink.apache.org/downloads.html#apache-flink-180

```
$ cd labs/flink-local
$ wget http://mirror.apache-kr.org/flink/flink-1.8.0/flink-1.8.0-bin-scala_2.11.tgz
```

Starting & Stopping Flink local cluster

```
$ bin/start-cluster.sh  # Start Flink
$ ps aux | grep flink
......

$ bin/stop-cluster.sh  # Stop Flink
```

> For Windows: https://ci.apache.org/projects/flink/flink-docs-release-1.8/tutorials/flink_on_windows.html

Flink + S3 (hadoop)

Edit Flink filnk-conf.yaml file

```
$ tar xvfz ...
$ cd flink-[VERSION]
$ vi conf/flink-conf.yaml
```

```
# Minio(S3)
state.backend: filesystem
s3.endpoint: http://127.0.0.1:9000
s3.path-style: true
s3.access-key: V42FCGRVMK24JJ8DHUYG
s3.secret-key: bKhWxVF3kQoLY9kFmt91l+tDrEoZjqnWXzY9Eza
```

Copy required jars into 'lib' directory

- flink-s3-fs-hadoop

```
$ cp opt/flink-s3-fs-hadoop-1.8.0.jar lib/
```

- flink-shaded-hadoop
  - flink-shaded-hadoop-2-uber-2.8.3-7.0.jar

Restart Flink local cluster

```

```

Apache Flink (batch) example

Source code for Apache Flink (batch) example:

- https://ci.apache.org/projects/flink/flink-docs-release-1.8/dev/batch/examples.html#word-count

Dataset for example:

- http://www.gutenberg.org/ebooks/4300

Create Minio(S3) bucket via Minio WebUI:

- http://127.0.0.1:9000

Creating required buckets:

- `test`
- `flink` (for checkpoints)
- `iextrading` (for Stock quote data)

Create 'iextrading' bucket

```
# mc mb local/iextrading
# mc mb local/flink
# mc mb local/test
```

Running Flink (batch) example

```
$ cd labs/docker/flink-local
$ cd flink-[VERSION]
$ bin/flink run examples/batch/WordCount.jar \
--input s3://data/gutenberg/4300-0.txt \
--output s3://test/wordcount/output
```

Flink Dashboard:

- http://localhost:8081

Verify example' output

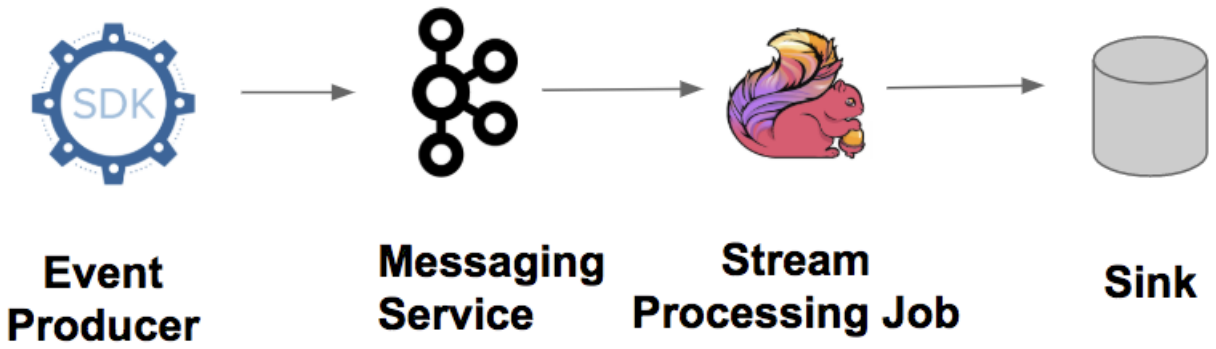
- verify the result in using Minio Web: http://localhost:9000
- verify the result(text file) in using Minio mc:

```
# mc cat local/test/wordcount/output
```

Runnig Flink-Kafka streaming application

데이터 흐름

- Trading(Quote) API -> Kafka Producer -> Kafka -> Flink Streaming -> Minio(S3)



Event Producer	Messaging Service	Stream Processing Job	Sink
kafka-message-gen	Apache Kafka	Flink Streaming Application	Minio (S3 Compatible)

Builing Flink Kafka application

```
$ cd /path/to/workspace/skbdc-bda-2019/
$ cd labs/flink-kafka-streaming
$ ./mvnw clean package
```

Running Flink streaming app on local cluster

```
$ cd labs/flink-local
$ cd flink-[VERSION]
$ bin/flink run ../../flink-kafka-streaming/target/flink-kafka-streaming-1.0.0.jar \
--input-topic iextrading \
--output-path s3://iextrading/filtered/ \
--bootstrap.servers localhost:9092 \
--schema-registry-url http://localhost:18081 \
--group.id cgrp1
```

Flink Logs & outputs

- Logs & STDOUT:
  - log/flink-...log
  - log/flink-...out
- Apache Flink Dashboard:
  - http://localhost:8081
- Output file(object) from Flink Streaming Application

```
# mc cat local/iextrading/filtered/YYYY-MM-DD/[part file]

E.g.,

# mc cat local/iextrading/filtered/2019-06-18/part-0-0
```

Kafka Streams 'filter' Application

- Kafka `iextrading_filtered` topic -> Kafka Streams (filter) -> Kafka `iextrading_filtered`

Create 'iextrding_filtered' Kafka topic:

```
$ export KAFKA_BROKER=$(docker ps --filter name=kafka1 --format={{.ID}})
$ docker exec -t -i "$KAFKA_BROKER" \
  kafka-topics --create --topic iextrading_filtered \
  --partitions 4 --replication-factor 1 \
  --if-not-exists --zookeeper zoo1:2181
```

Create 'iextrading_filtered-value' schema

- http://localhost:8001

Building & running Kafka Streams application

```
$ cd labs/kafka-streams-bda
$ ./mvnw clean package
$ java -cp target/kafka-streams-bda-1.0.0-standalone.jar com.example.KafkaStreamsApp
```

Verify events from 'iextrading_filtered' topic

```
$ export KAFKA_SR=$(docker ps --filter name=kafka-schema-registry --format={{.ID}})
$ docker exec -t -i "$KAFKA_SR" \
  kafka-avro-console-consumer --topic iextrading_filtered --bootstrap-server kafka1:19092 \
  --property schema.registry.url="http://kafka-schema-registry:8081"
```

Stream Processing with Apache Spark

Running Spark docker container

```
$ docker run --rm -it --name spark1 --net docker_default \
  -v "$PWD":/data -p 4040:4040 gettyimages/spark bash -l
#
```

Spark shell with S3 support

Copy core-site.xml into container:

```
$ export SPARK=$(docker ps --filter name=spark1 --format={{.ID}})
$ docker cp core-site.xml "$SPARK":/usr/hadoop-3.0.0/etc/hadoop/core-site.xml
```

```
# export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
# export SPARK_HOME=/usr/spark-2.4.1
# export PATH=$PATH:$SPARK_HOME/bin
# export HADOOP_HOME=/usr/hadoop-3.0.0
# export PATH=$PATH:$HADOOP_HOME/bin
# export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native
# export HADOOP_OPTIONAL_TOOLS="hadoop-aws"
# export SPARK_DIST_CLASSPATH=$(hadoop classpath)
```

```
# cd /usr/spark-2.4.1
# bin/spark-shell --master local[4]
```

Reading a text file from Minio(S3)

```
scala> val b1 = sc.textFile("s3a://data/gutenberg/4300-0.txt")
scala> b1.collect().foreach(println)
```

Word Count

```
scala> val textFile = sc.textFile("s3a://data/gutenberg/4300-0.txt")
scala> val counts = textFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_ +
scala> counts.saveAsTextFile("s3a://test/wc/result")
```

Spark Streaming - Kafka Integration: Processing Kafka messages

Building Spark straming application

```
$ cd labs/spark-kafka-streaming
$ mvn clean package
```

Running spark application

```
# export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
# export SPARK_HOME=/usr/spark-2.4.1
# export PATH=$PATH:$SPARK_HOME/bin
# export HADOOP_HOME=/usr/hadoop-3.0.0
# export PATH=$PATH:$HADOOP_HOME/bin
# export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native
# export HADOOP_OPTIONAL_TOOLS="hadoop-aws"
# export SPARK_DIST_CLASSPATH=$(hadoop classpath)
```

```
# cd $SPARK_HOME
# bin/spark-submit --master local[4] spark-kafka-streaming-1.0.0-jar-with-dependencies.jar
```

Browse http://localhost:4040 for web UI.

Spark Structured Streaming - Minio(S3) Integration

TBD

See https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

Streaming Data Integration

- Apache Kafka - 'iextrading_filtered' topic -> Kafka Streams -> RDBMS(MySQL)

Running MySQL server

```
$ docker run -d --name mysql1 --network docker_default -p 3306:3306 \
  -e MYSQL_ROOT_PASSWORD=mypasswd -d mysql:5.7
```

Running MySQL CLI

```
$ docker run -it --network docker_default --rm mysql mysql \
  -hmysql1 -uroot -pmypasswd
```

Schema for `iextrading_filtered` table

Column Name	Data type	Description
ts	timestamp	
symbol	varchar(10)	
company_name	varchar(100)	
low	double	
high	double	
latest_update	double	
latest_price	double	

DDL for MySQL

```
mysql> CREATE DATABASE stock;

mysql> USE stock;

mysql> CREATE TABLE iextrading_filtered
(
  ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP PRIMARY KEY,
  symbol varchar(10),
  company_name varchar(100),
  low double,
  high double,
  latest_update double,
  latest_price double
);
```

Building Kafka Streams application

```
$ cd /path/to/labs/kafka-streams-proc/
$ ./mvnw clean package
```

Running Kafka Streams application

```
$ java -cp target/kafka-streams-proc-1.0.0-standalone.jar \
com.example.KafkaStreamsProcessor
```

Querying MySQL table `iextrading_filtered`

```
$ docker run -it --network docker_default --rm mysql mysql \
  -hmysql1 -uroot -pmypasswd
```

```
mysql> USE stock;
mysql> SELECT * FROM iextrading_filtered limit 10;
```

Data Analytics / SQL / Dashboard

SQL

Querying the wordcount result in using Presto

Presto is a distributed SQL query engine designed to query large data sets distributed over one or more heterogeneous data sources.

Querying data from Object Storage

Make `warehouse` bucket for presto-hive connector:

```
# mc mb local/warehouse
```

Running Presto CLI:

```
$ docker exec -it presto bash -l

# presto-cli

or

$ docker exec -it presto presto-cli
```

```
presto> SHOW CATALOGS;
  Catalog
-----------
 blackhole
 jmx
 memory
 minio
 system
 tpcds
 tpch

presto> SHOW SCHEMAS FROM minio;
       Schema
 --------------------
  information_schema

presto> CREATE SCHEMA minio.default;

# create 'skbdc' schema...
presto> CREATE SCHEMA minio.skbdc;
```

```
presto> SELECT * FROM system.metadata.table_properties;

presto> CREATE TABLE minio.default.gutenberg (txt varchar)
        WITH (
          format = 'TEXTFILE',
          external_location = 's3a://data/gutenberg/'
        );

presto> CREATE TABLE minio.default.wordcount (line varchar)
        WITH (
          format = 'TEXTFILE',
          external_location = 's3a://test/wordcount/'
        );

presto> SELECT split(line, ' ')[1] as word, split(line, ' ')[2] as cnt
        FROM minio.default.wordcount;
```

```
presto> CREATE VIEW minio.default.v_wordcount
        AS SELECT split(line, ' ')[1] AS word, split(line, ' ')[2] AS cnt
           FROM minio.default.wordcount;

presto> CREATE TABLE minio.skbdc.wordcount_top100
        AS SELECT * from minio.default.v_wordcount ORDER BY cnt DESC LIMIT 100;

presto> SELECT * FROM minio.skbdc.wordcount_top100;

presto> SELECT count(1) FROM minio.default.v_wordcount;

presto> SELECT * FROM minio.default.v_wordcount;
```

Presto table for filtered `Quote` data

```
presto> SELECT * FROM system.metadata.table_properties;

presto> CREATE TABLE minio.default.iextrading_filtered (
          uts double,
          symbol varchar(10),
          high double,
          low double,
          latest_price double,
          latest_update varchar(100) )
        WITH (
          format = 'TEXTFILE',
          external_location = 's3a://iextrading/filtered/YYYY-MM-DD/'
        );

presto> SELECT * FROM minio.default.iextrading_filtered;

presto> CREATE TABLE minio.skbdc.stock_2019
        AS SELECT from_unixtime(uts) AS ts, symbol, latest_price
           FROM minio.default.iextrading_filtered;
```

Data Analytics

JupyterLab

- JupyterLab
  - JupyterLab is the next-generation web-based user interface for Project Jupyter.
- Python modules... boto3, pymysql, pandas, etc.

Refs.

- https://github.com/jupyterlab/jupyterlab
- https://github.com/youngwookim/my-docker-stacks

Running JupyterLab via Docker:

```
$ cd labs
$ docker run --rm --network docker_default --user root -p 8888:8888 \
-e GRANT_SUDO=yes -e JUPYTER_ENABLE_LAB=yes \
-v "$PWD":/home/jovyan youngwookim/my-datascience-notebook:latest

......

Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://(be1ee3cbaf72 or 127.0.0.1):8888/?token=42a319245ef11fc8b5fbae2480fd3b3da557489b05f4
```

http://localhost:8888/?token=42a319245ef11fc8b5fbae2480fd3b3da557489b05f4f357

Crunching S3 data with Jupyter Notebook

- labs/notebooks/minio-s3-python-pandas.ipynb

Visualization MySQL data in Jupyter Notebook

- Plotting data using python(pandas, pymysql, matplotlib) on Jupyter Notebook:
    - labs/notebooks/kafka-mysql-pandas-plot.ipynb

Dashboard

Apache Superset (incubating)

- https://superset.incubator.apache.org/
- Apache Superset (incubating) is a modern, enterprise-ready business intelligence web application
- How Superset and Druid Power Real-Time Analytics at Airbnb | DataEngConf SF '17

Superset Docker 컨테이너 실행

```
$ docker run --network docker_default -d --name superset \
  -p 8088:8088 tylerfowler/superset
```

Browse http://localhost:8088

Login with a default username and password of:

```
username: admin
password: superset
```

데이터 소스

```
presto://[HOST_IP]:8080/minio

e.g., presto://presto:8080/minio
```

Metatron Discovery

Metatron Discovery Docker 컨테이너 실행

- https://metatron.app/download/installation-guide-docker/

```
$ cd labs
$ docker run  -i -d --rm -m 6G --network docker_default \
  -p 8180:8180 \
  --name metatron-discovery metatronapp/discovery:latest
```

Browse metatron discovery:

- http://localhost:8180

Login with a default username and password of:

```
username: admin
password: admin
```

참고

- https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/
- https://cdn.oreillystatic.com/en/assets/1/event/278/Architecting a next-generation data platform Presentation 3.pdf
- https://www.slideshare.net/arunkejariwal/designing-modern-streaming-data-applications-115037555
- https://blog.newrelic.com/engineering/kafka-best-practices/
- https://community.hortonworks.com/articles/80813/kafka-best-practices-1.html
- https://www.slideshare.net/JayKreps1/i-32858698
- https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying
- https://www.slideshare.net/hadooparchbook/architectural-considerations-for-hadoop-applications
- https://www.youtube.com/watch?v=oCW5y4_8uGU
- https://www.youtube.com/watch?v=W_Sp4jo1ACg
- https://ci.apache.org/projects/flink/flink-docs-release-1.8/
- https://prestosql.io/
- https://min.io/
- https://github.com/jupyterlab/jupyterlab
- https://jack-vanlightly.com/blog/2018/10/2/understanding-how-apache-pulsar-works

Image Credits

- https://mattturck.com/bigdata2018/
- https://www.thoughtworks.com/insights/blog/agile-data-warehousing-and-business-intelligence-action
- http://insight360.com/big-data/data-stores/
- https://en.wikipedia.org/wiki/Apache_Kafka
- https://www.talend.com/blog/2017/08/28/lambda-kappa-real-time-big-data-architectures
- https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781786466228/1/ch01lvl1sec8/architecture
- https://www.starburstdata.com/learn-presto/reference-architectures/
- https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/
- https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying
- https://cbmpress.com/toronto/아마존-캐나다-대형급-물류창고-허브-gta에-건설-발표/
- https://www.memsql.com/blog/exactly-once-semantics-with-apache-kafka/
- https://www.talend.com/blog/2017/08/28/lambda-kappa-real-time-big-data-architectures/
- https://www.oreilly.com/ideas/applying-the-kappa-architecture-in-the-telco-industry
- https://www.starburstdata.com/learn-presto/reference-architectures/
- https://medium.com/netflix-techblog/keystone-real-time-stream-processing-platform-a3ee651812a

FIN