

3. 빅데이터 플랫폼



빅데이터 플랫폼 개요

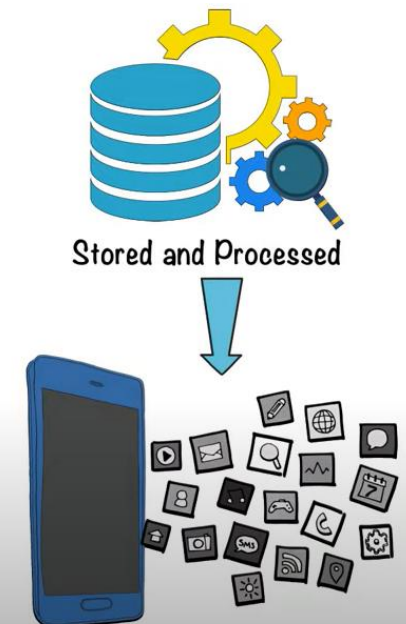
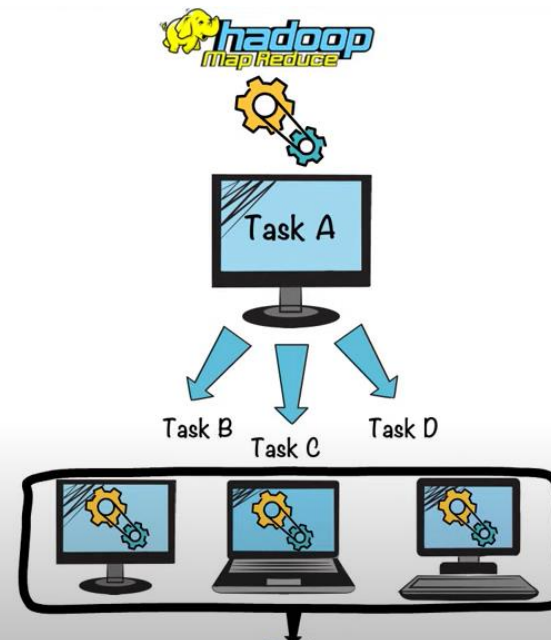
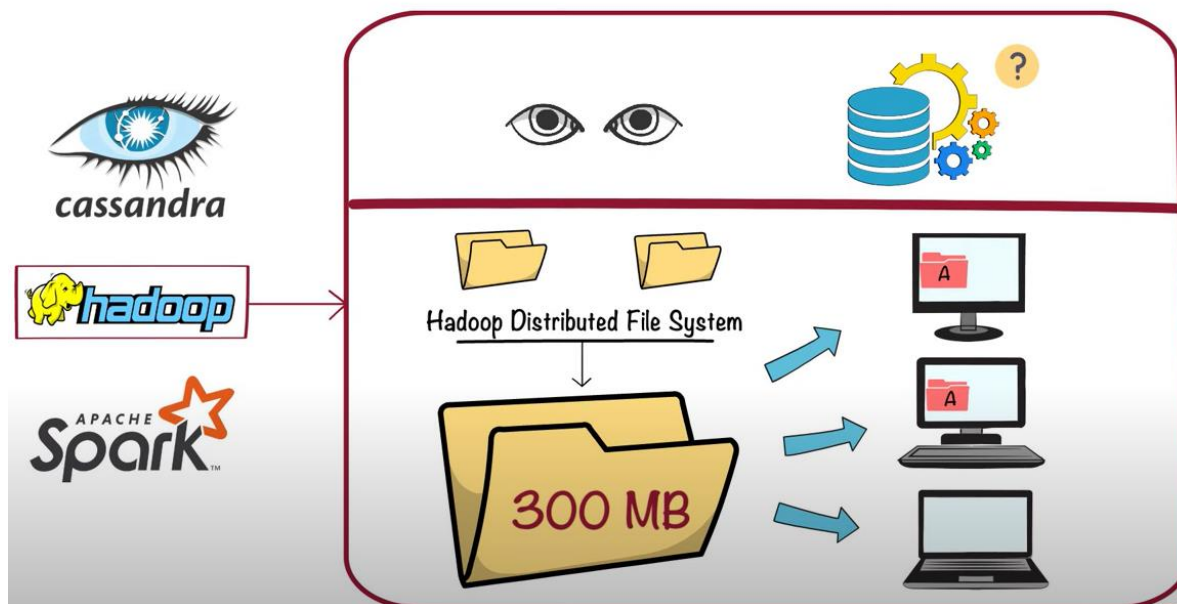
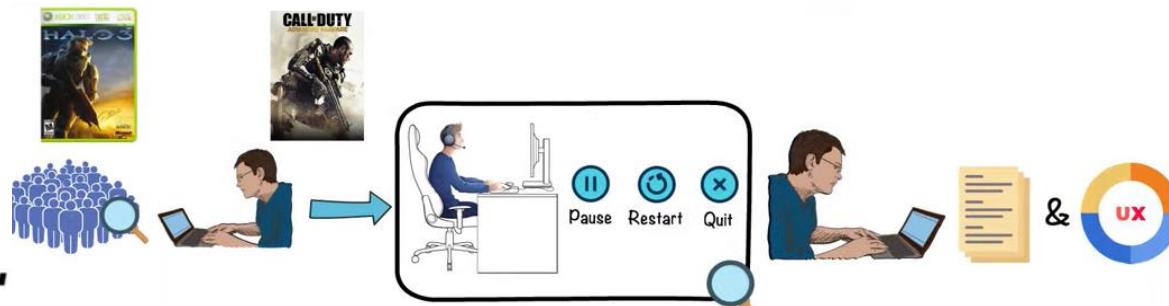
Let's have a look at the data generated per minute on the internet


"2.1Million"


"3.8Million"


"1.0Million"


"4.5Million"



데이터 형태

정형 데이터

- 형태가 있으며 연산 가능함. 주로 관계형 데이터베이스에 저장됨
- 데이터 수집 난이도가 낮음
- 내부 시스템인 경우가 대부분임
- 파일 형태의 스프레드시트라도 내부에 형식을 가지고 있어 처리가 쉬운 편임
- EX) 관계형 데이터베이스, 스프레드시트, CSV 등

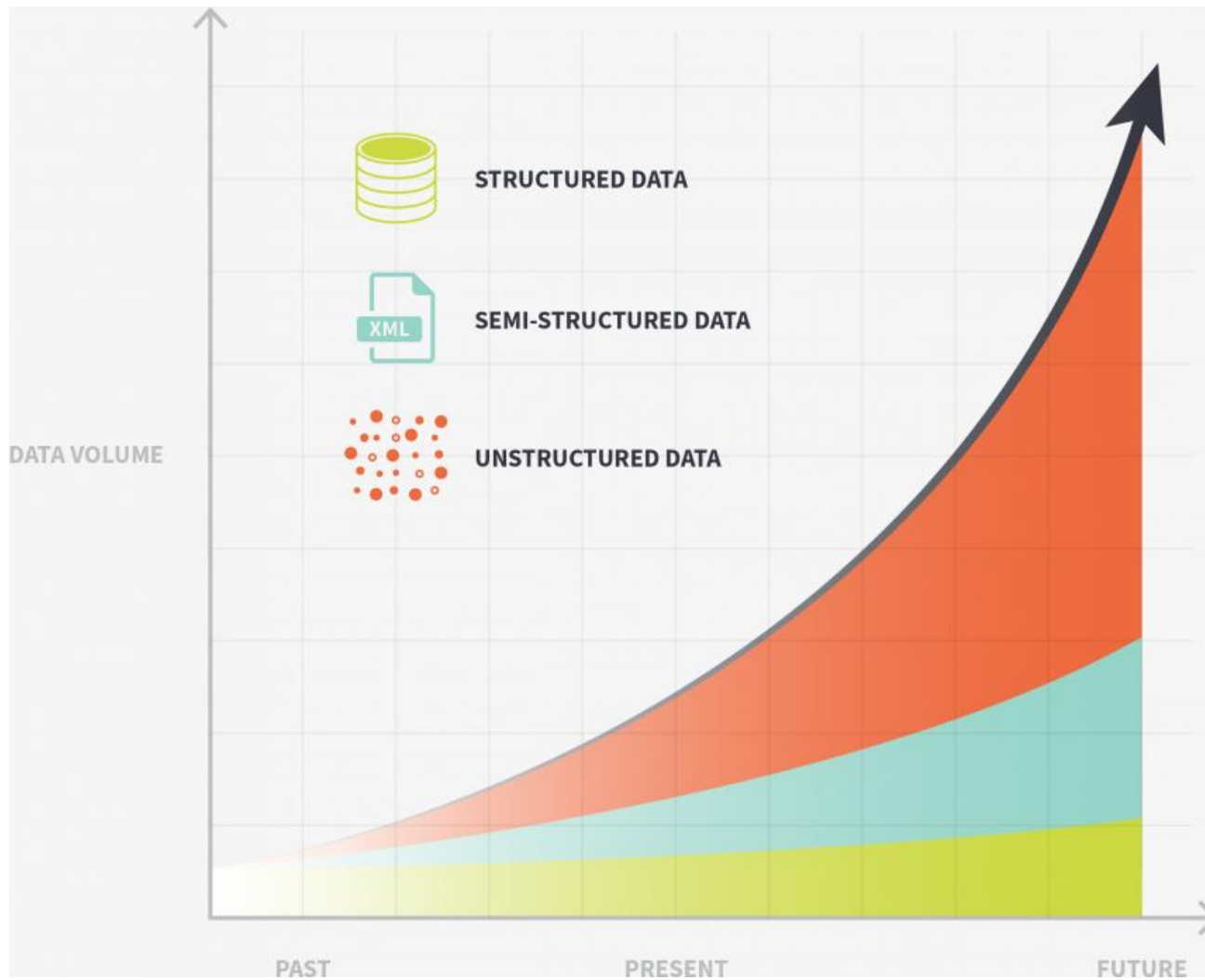
반정형 데이터

- 형태(스키마, 메타데이터)가 있으며 연산이 불가능
- 주로 파일에 저장됨
- 데이터 수집 난이도가 중간
- 보통 API 형태로 제공되기 때문에 데이터 처리기술이 요구됨
- EX) XML, HTML, 로그형태(웹로그, 센서데이터), Machine Data 등

비정형 데이터

- 형태가 없으며, 연산이 불가능함
- 주로 NoSQL에 저장됨
- 데이터 수집 난이도가 높음
- 텍스트 마이닝 혹은 파일일 경우 파일을 데이터 형태로 파싱해야 하기 때문에 수집 데이터 처리가 어려움
- EX) 소셜데이터(트위터, 페이스북), 이메일, 보고서

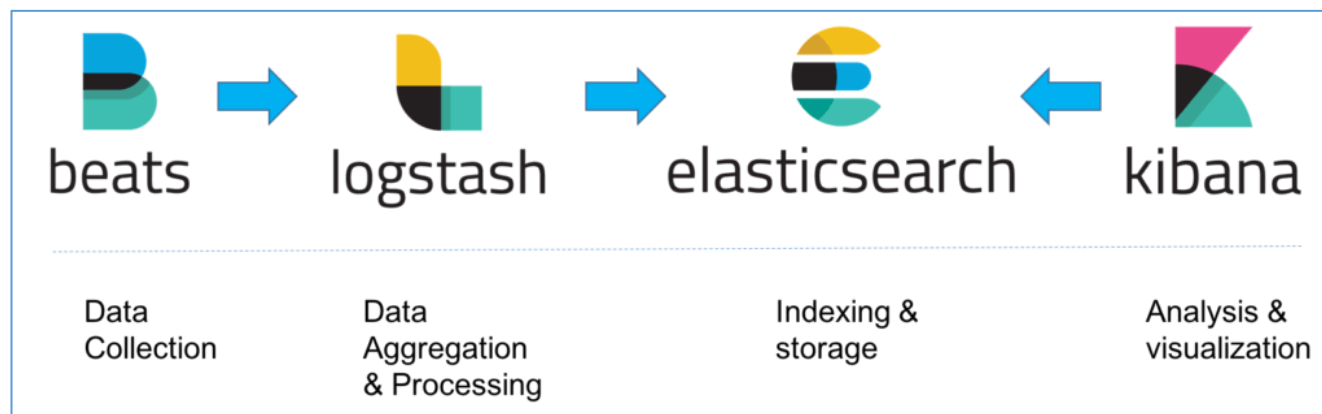
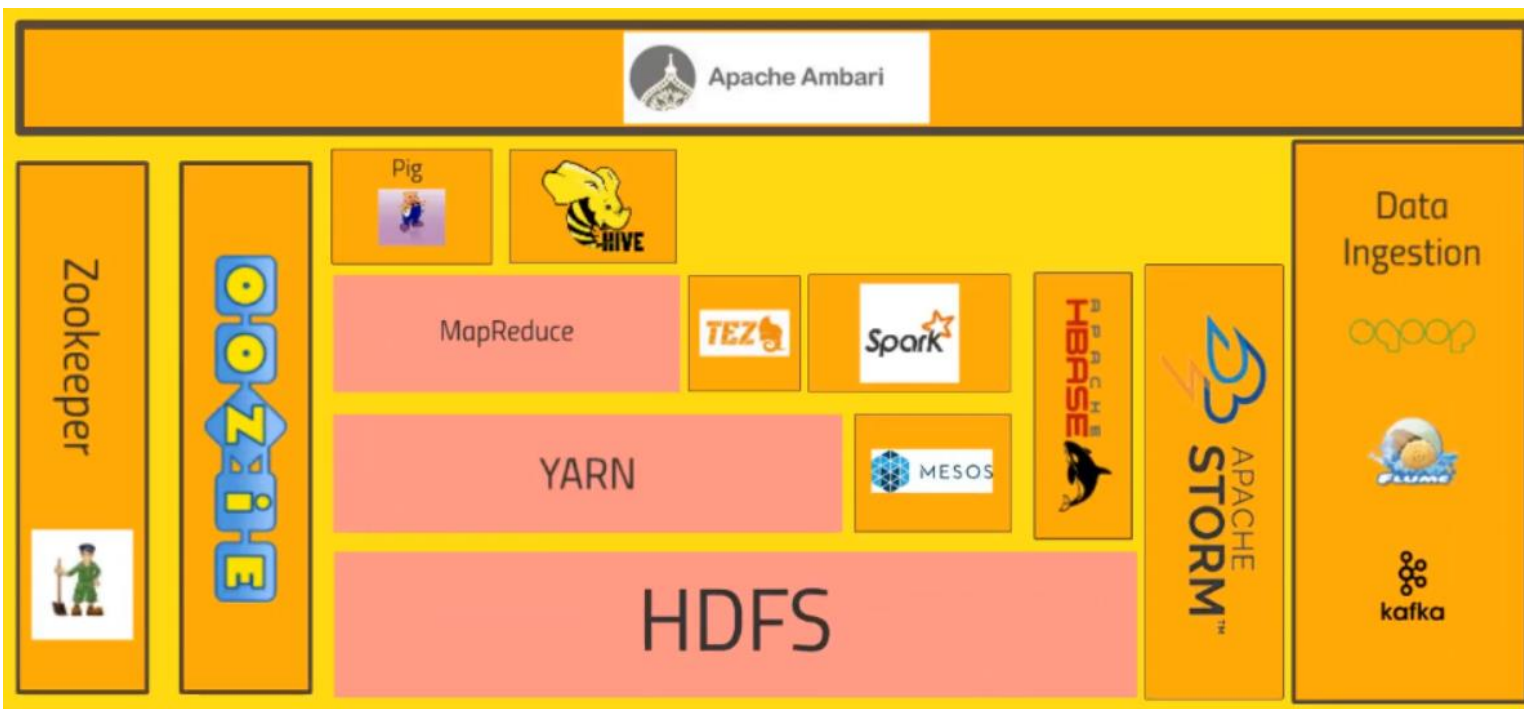
비정형 데이터



NoSQL



빅데이터 프로세싱 솔루션



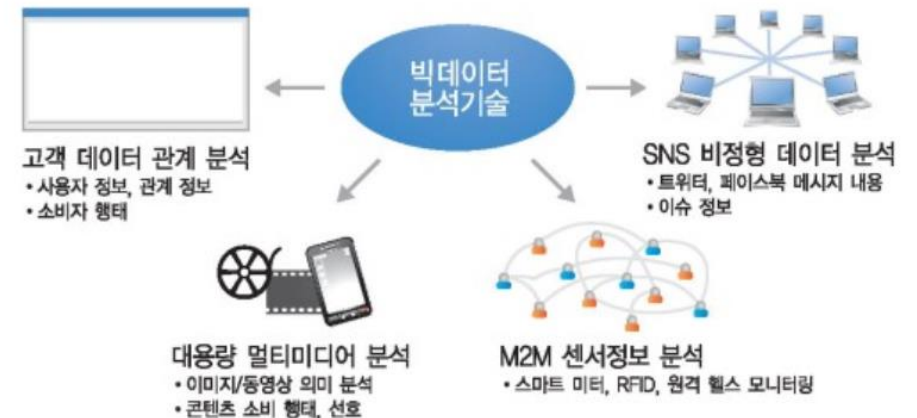
빅데이터 기술

■ 빅데이터 저장기술

- 다양하고 많은 양의 빅데이터를 저장하고 관리하는 기술이 필수
- 대표적인 빅데이터 저장기술 : 하둡(Hadoop), NoSQL(Not Only SQL)
- 하둡(Hadoop) : 대용량 데이터를 분산 처리할 수 있는 자바 기반의 오픈 소스 프레임워크
- NoSQL(Not Only SQL) : 관계형데이터베이스의 일관성 특징보다는 가용성과 확장성에 중점을 둔 데이터베이스

■ 빅데이터 분석기술

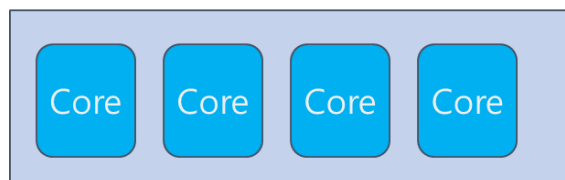
- 텍스트 마이닝, 오피니언 마이닝, 소셜 네트워크 분석, 패턴인식, 머신러닝, 딥러닝, 자연어 처리 기술 활용
- 텍스트 마이닝(text mining) : 텍스트 데이터에서 자연어 처리 기술을 기반해 가치 있는 정보를 추출하고 가공
- 오피니언 마이닝(opinion mining) : SNS, 블로그 게시물 등에서 사용자들의 의견을 수집하여 제품과 서비스에 대한 감성을 파악하거나 유용한 정보로 재가공하는 기술
- 소셜 네트워크 분석(social network analysis) : 소셜 네트워크상에서의 영향력인 사람/데이터 등 객체 간의 관계/특성을 분석하고 시각화하는 기법



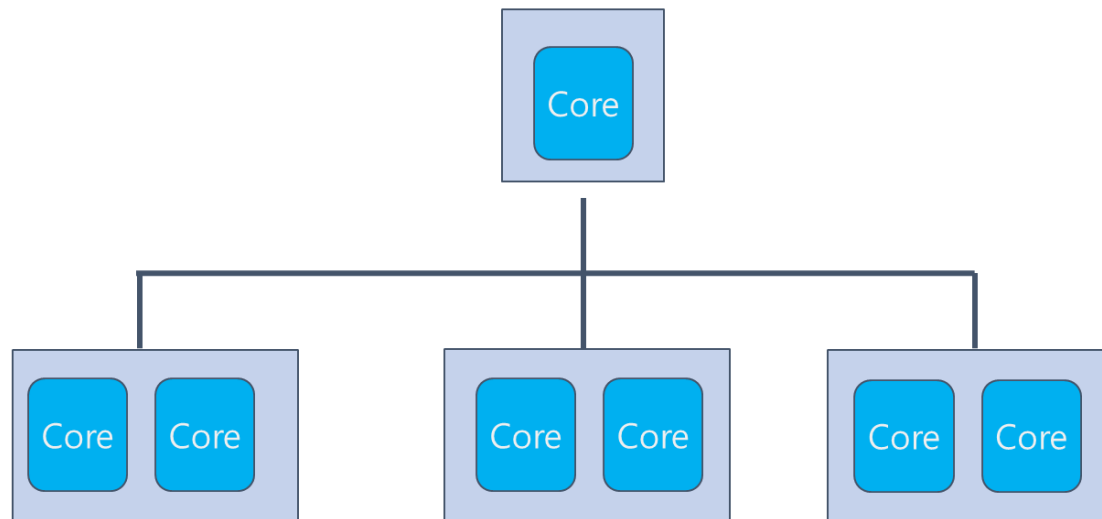
분산시스템

- 로컬 프로세스는 단일 시스템의 컴퓨팅 리소스를(예: 32GB) 사용합니다.
- 더 큰 데이터셋은 SQL 데이터베이스를 사용하여 스토리지를 사용하거나, 여러대의 컴퓨터로 구성된 분산시스템을 사용해야 합니다.
- 분산 프로세스는 네트워크를 통해 연결된 여러 머신의 컴퓨팅 리소스를 액세스 할 수 있습니다.
- 단일 머신을 Scale Up 하는 것보다 여러대의 머신으로 확장(Scale Out)하는 것이 더 쉽습니다.
- 분산시스템에서는 한 대의 시스템에 장애가 발생해도 전체 네트워크가 계속 작동 할 수 있습니다.

Local



Distributed



하둡(Hadoop)



Hadoop Architecture

Application Layer



Other
Applications

Resource Management
Layer

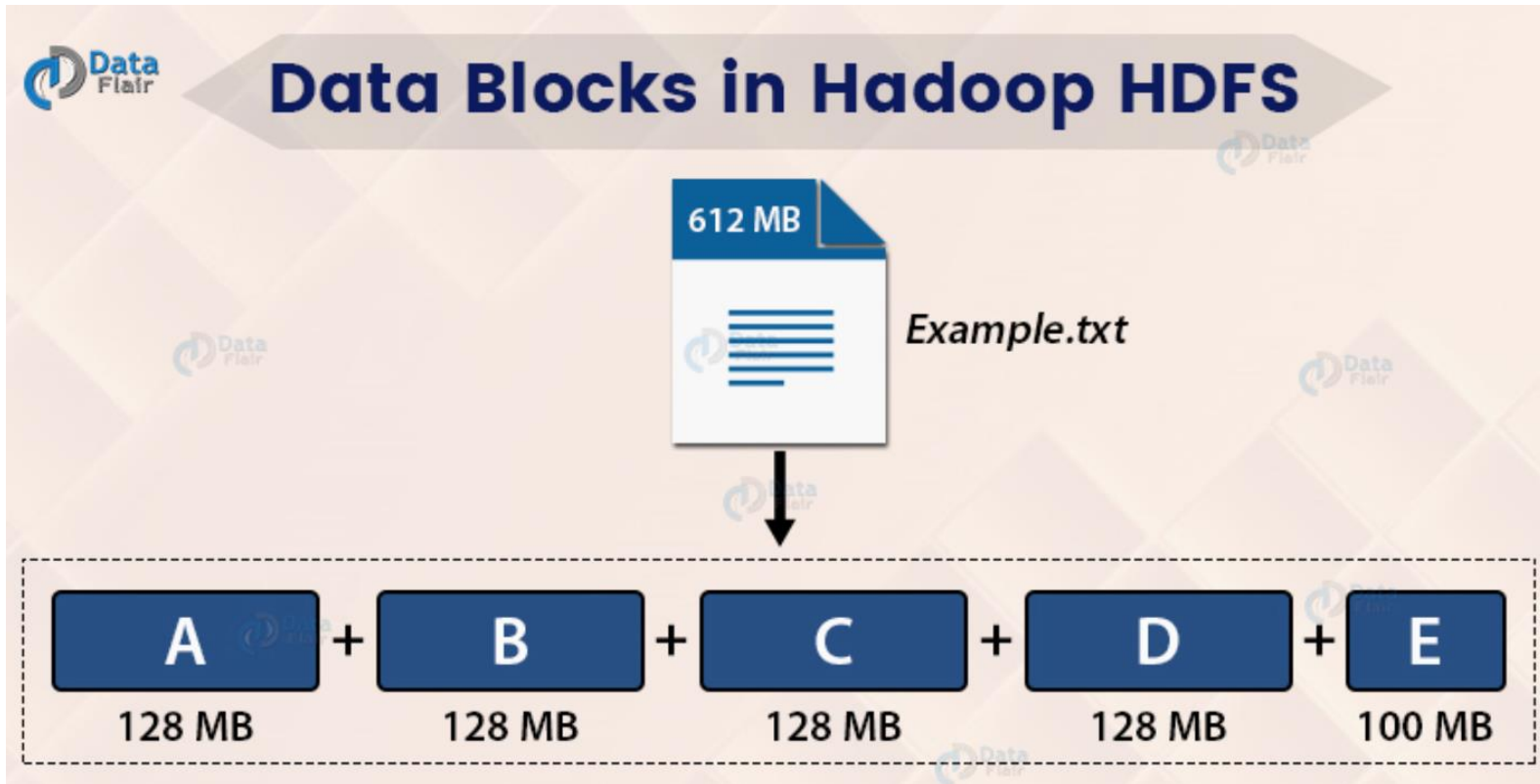


Storage Layer



하둡 HDFS(Hadoop Distributed File System)

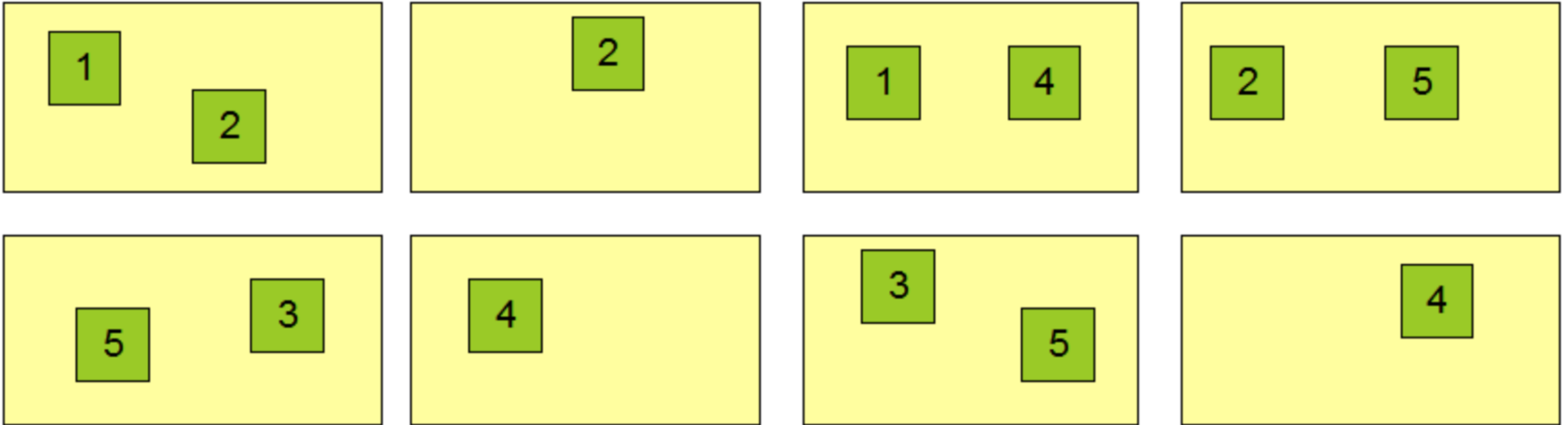
- HDFS는 파일을 분산 저장하기 위해서 먼저 파일의 메타 데이터와 콘텐츠 데이터를 분리합니다.
- 메타 데이터 : 파일의 접근 권한, 생성일, 수정일, 네임 스페이스 등 파일에 대해 설명하는 정보
- 콘텐츠 데이터 : 실제 파일에 저장된 데이터
- 파일의 메타데이터는 네임 노드에 저장되며 콘텐츠 데이터는 블록 단위로 쪼개져서 데이터 노드에 저장 됩니다.



하둡 HDFS 데이터 노드

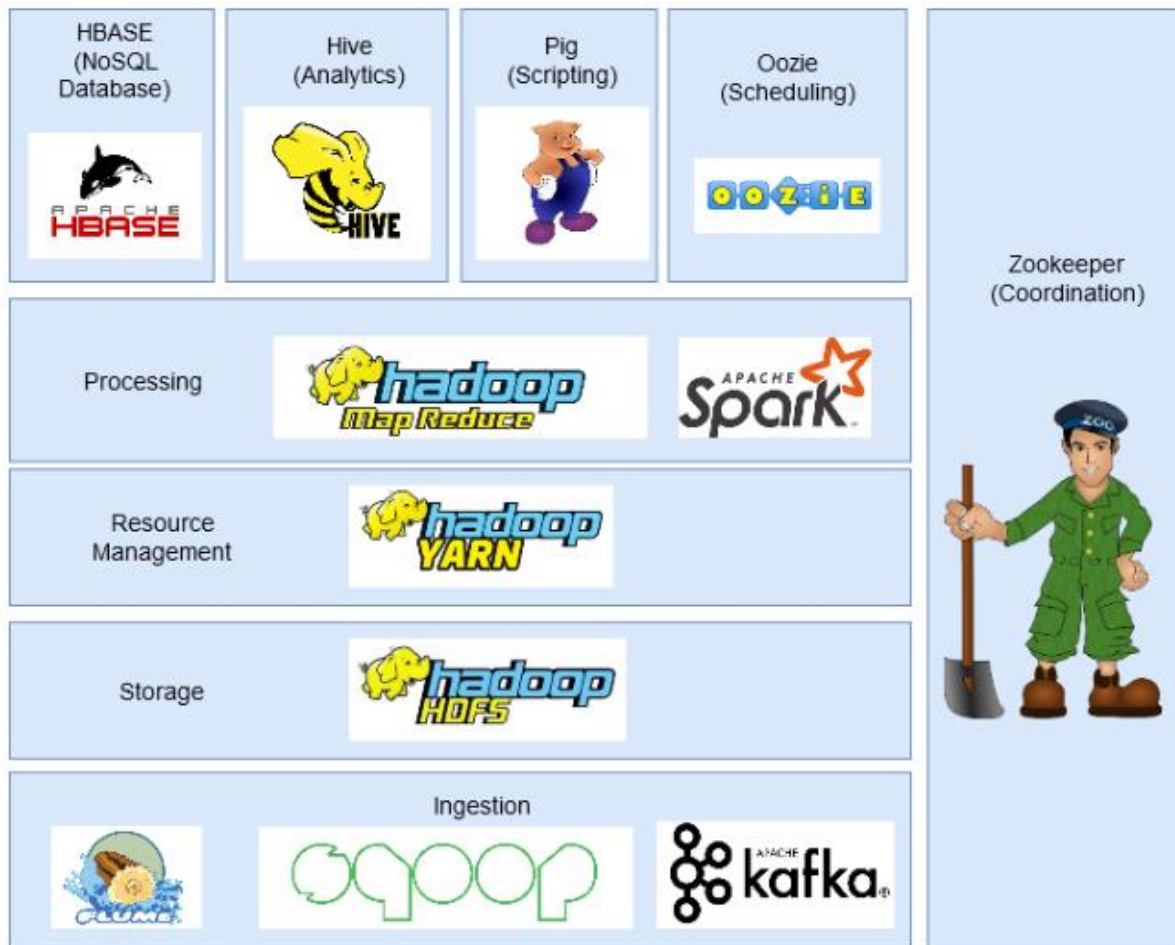
- 파일의 콘텐츠 데이터는 블록 단위로 나뉘며, 블록의 기본 크기는 128MB이며 최소 3개의 복사본을 생성하여 분산 저장합니다.
- 데이터 노드는 그 중 하나의 복사본을 저장하는 것입니다.

Datanodes

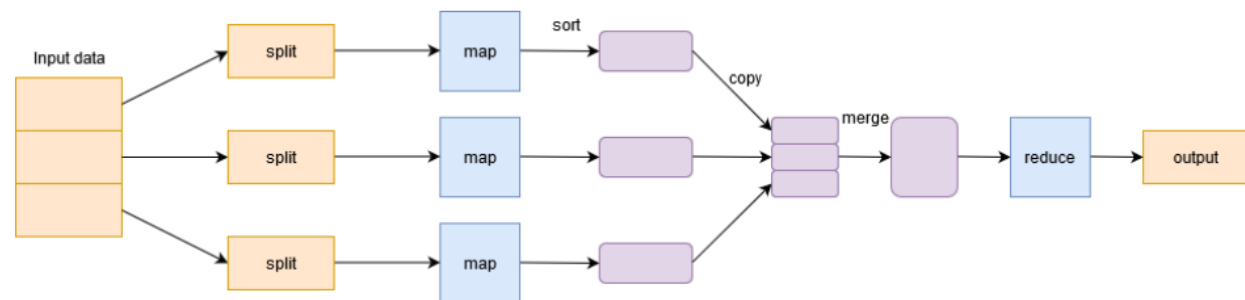


하둡 에코시스템

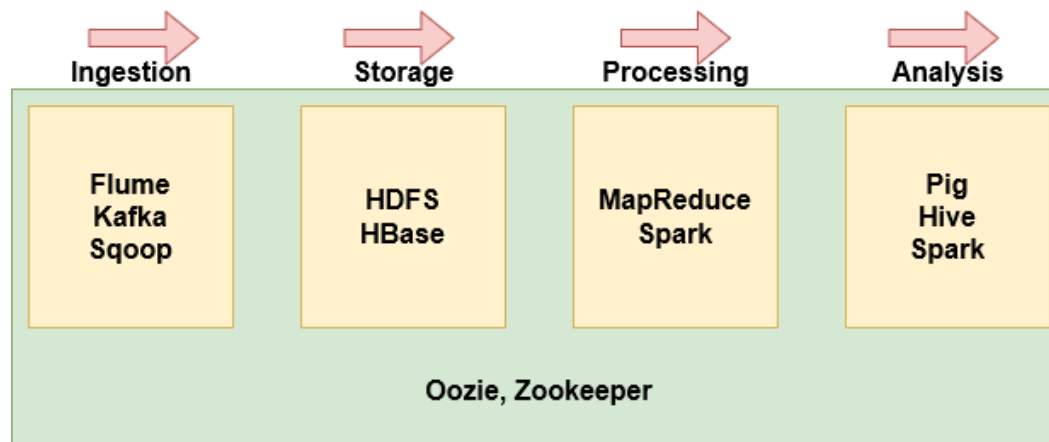
하둡 에코시스템



MapReduce 알고리즘

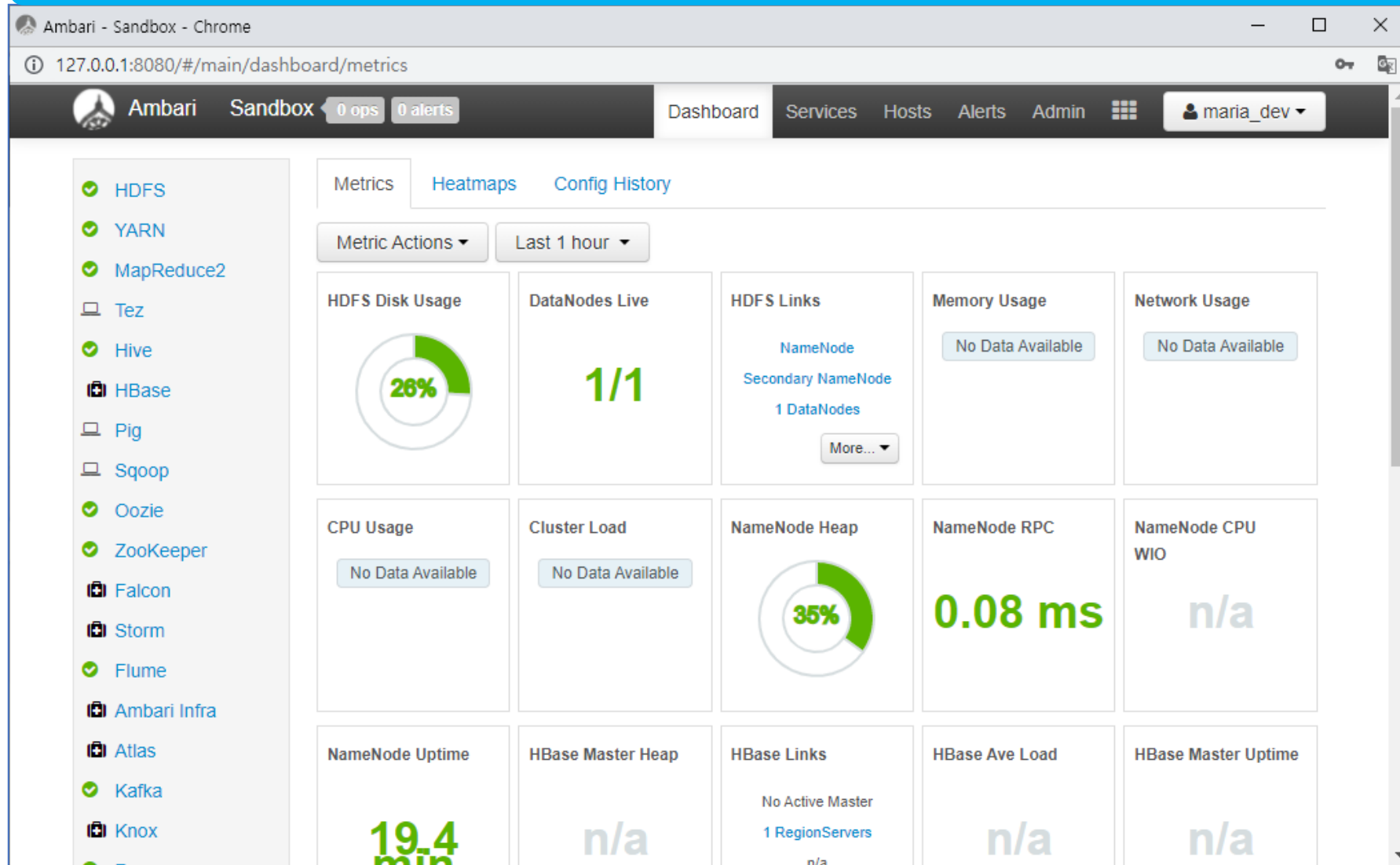


빅데이터 프로세싱 단계



HDP(Hortonworks Data Platform)

<https://www.cloudera.com/downloads/hortonworks-sandbox.html>



빅데이터 분산 프로세싱 엔진 스파크(Spark)

spark.apache.org



Download Libraries Documentation Examples Community Developers

Apache Software Foundation

Unified engine for large-scale data analytics

GET STARTED

Python

SQL

Run now

Installing with 'pip'

```
$ pip install pyspark
$ pyspark
```

QuickStart

```
df = spark.read.json("logs.json")
df.where("age > 21").select("name.first").show()
```

What is Apache Spark™?

Apache Spark™ is a multi-language engine for executing data engineering, data

Simple.
Fast.
Scalable.
Unified.



Batch/streaming data

Unify the processing of your data in batches and real-time streaming, using your preferred language: Python, SQL, Scala, Java or R.



Data science at scale

Perform Exploratory Data Analysis (EDA) on petabyte-scale data without having to resort to downsampling



SQL analytics

Execute fast, distributed ANSI SQL queries for dashboarding and ad-hoc reporting. Runs faster than most data warehouses.



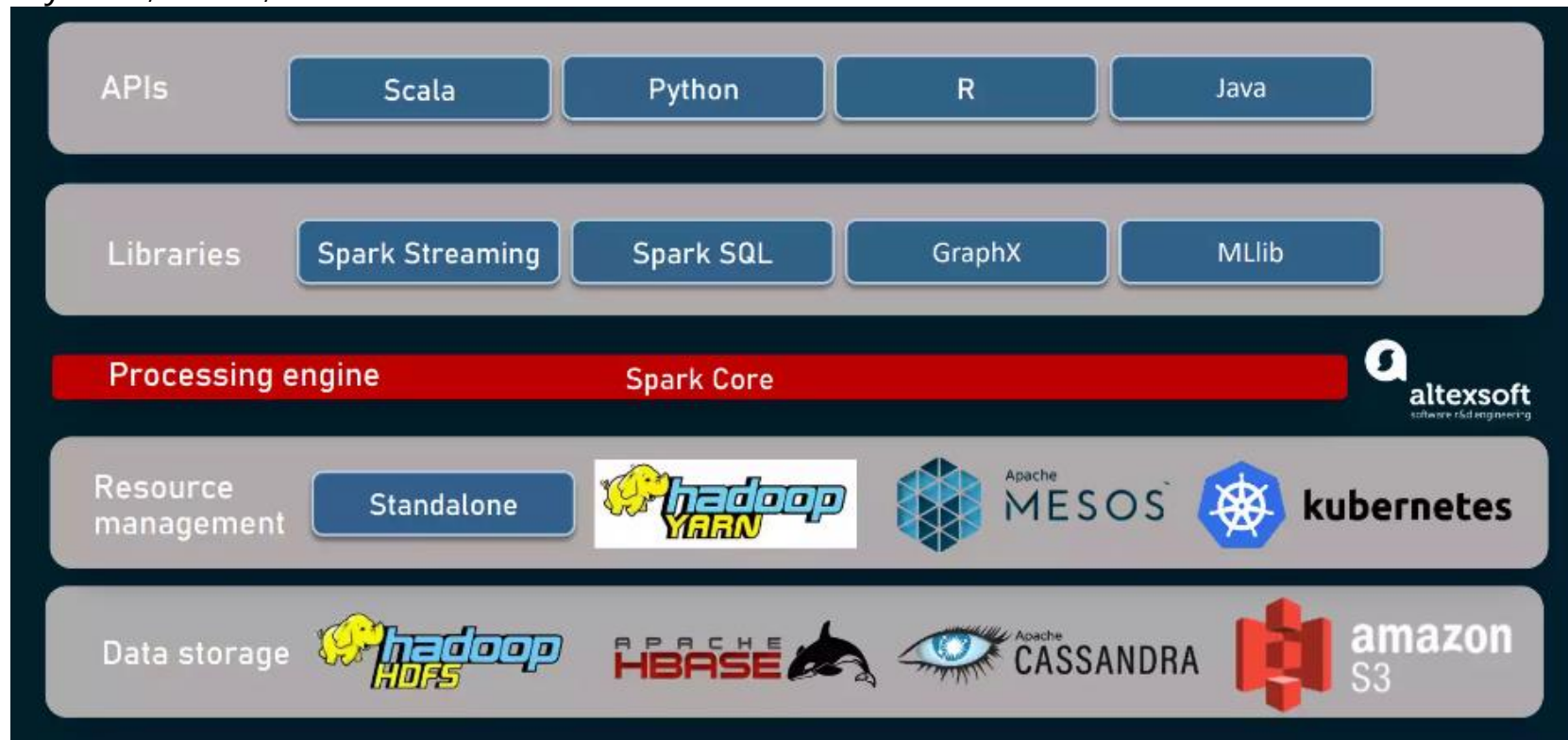
Machine learning

Train machine learning algorithms on a laptop and use the same code to scale to fault-tolerant clusters of thousands of machines.

Spark 특징

- 인메모리(In-Memory) 컴퓨팅, Disk 기반도 가능
- RDD(Resilient Distributed DataSet) 데이터모델, 빠른 데이터 프로세싱(In-Memory Cached RDD, Up to 100x faster)
- 실시간(Real-Time) Stream Processing
- 다양한 개발언어(Scala, Python, Java, R, SQL) 지원, 개발자 친화적인 수많은 API 제공, 2~10x Less Code
- 사용자의 데이터 처리 명령을 방향성 비순환 그래프(Directed Acyclic Graph, DAG)로 스케줄링
- Hadoop(HDFS, YARN, HBase 등)과 유연한 연계
- 대화형 질의를 위한 Interactive Shell : Python, Scala, R 인터프리터
- 하나의 애플리케이션에서 배치, SQL 쿼리, 스트리밍, 머신러닝과 같은 다양한 작업을 하나의 워크플로우로 결합 가능

참고 : <https://www.itworld.co.kr/news/147556>



<https://www.altexsoft.com/blog/hadoop-vs-spark/>

Hadoop vs Spark

- 대부분의 Hadoop 배포판에 Spark가 포함되어 있으며, Hadoop vs Spark 비교는 다소 부적절합니다.
- Spark는 인메모리 데이터 엔진을 통해 특정 상황에서 맵리듀스보다 100배 더 빠르게 작업을 수행하고, 개발자 친화적인 API 제공 이점 덕분에 빅데이터 처리 분야에서 Hadoop MapReduce 패러다임을 추월하여 가장 유력한 프레임워크로 부상 하였습니다.

	Hadoop	Spark
What is it?	Open-source framework for distributed data storage and processing	Open-source framework for in-memory distributed data processing and app development
Initial release	2006	2014
Supported languages	Java	Scala, Java, Python, R
Processing methods	Batch processing, using hard discs to read/write data	Batch and micro-batch processing in RAM
Built-in capabilities	<ul style="list-style-type: none">✓ File system (HDFS)✓ Resource management (Yarn)✓ Processing engine (MapReduce)	<ul style="list-style-type: none">✓ Processing engine (Spark Core)✓ Near real-time processing (Spark Streaming)✓ Structured data processing (Spark SQL)✓ Graph data management (GraphX)✓ ML library (MLlib)

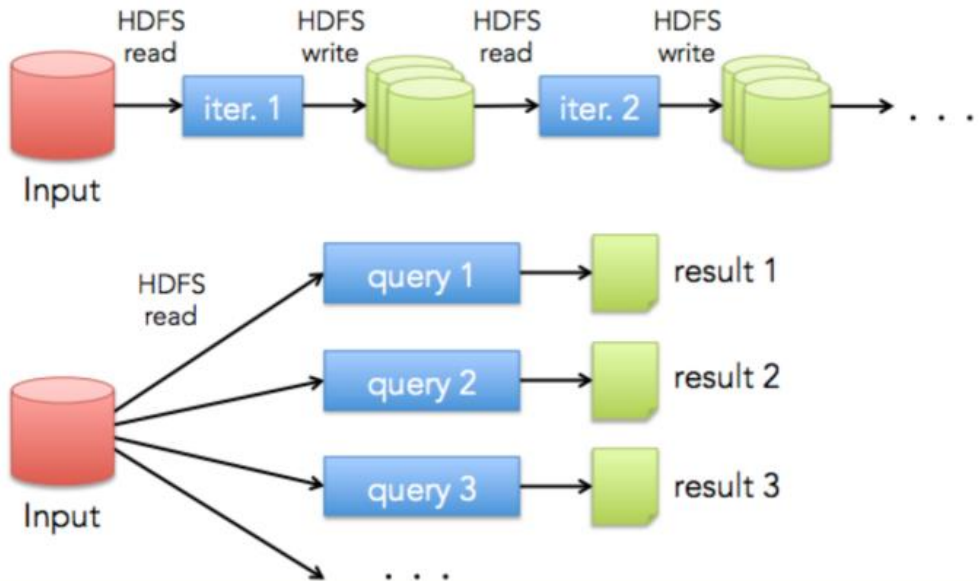
	Hadoop	Spark
Best fit for	Delay-tolerant processing tasks, involving huge datasets	Almost instant processing of live data and quick analytics app development
Real-life use cases	<ul style="list-style-type: none">✓ Enterprise archived data processing✓ Sentiment analysis✓ Predictive maintenance✓ Log files analysis	<ul style="list-style-type: none">✓ Fraud detection✓ Telematics analytics✓ User behavior analysis✓ Near real-time recommender systems✓ Stock market trends prediction✓ Risk management

<https://www.altexsoft.com/blog/hadoop-vs-spark/>

Hadoop vs Spark

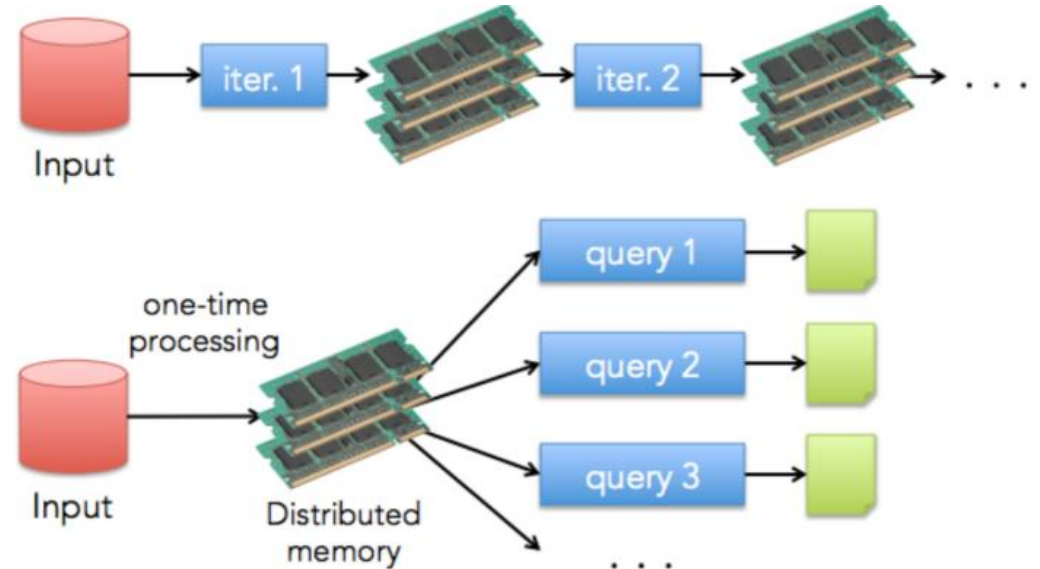
- Hadoop MapReduce는 각 map, reduce 이후에 대부분의 데이터를 디스크(HDFS)에 저장합니다.
- Spark는 각 변환 후 대부분의 데이터를 메모리에 보관합니다.
- Spark는 인메모리 기반의 처리로 Hadoop MapReduce 대비 10~100배 빠른 속도를 제공합니다.
- Spark는 머신러닝, 그래프처리 등 빅데이터 분석을 위한 통합 컴포넌트를 제공합니다.

■ Hadoop



Slow due to data replication and disk I/O

■ Spark



10-100× faster than network and disk

Colab에서 PySpark 사용하는 방법



spark_in_colab.ipynb

■ 방법 #1

```
!pip install pyspark py4j
```

■ 방법 #2

```
# !apt-get install openjdk-8-jdk-headless -qq  
!wget -q !wget -q https://downloads.apache.org/spark/spark-3.3.2/spark-3.3.2-bin-hadoop3.tgz  
!tar -xf spark-3.3.2-bin-hadoop3.tgz  
!pip install -q findspark
```

```
import os  
import findspark
```

```
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"  
os.environ["SPARK_HOME"] = "/content/spark-3.3.2-bin-hadoop3"
```

```
findspark.init()  
findspark.find()
```

참조 : <https://medium.com/@TheITspace/running-pyspark-on-google-colab-2552435972b3>

PySpark DataFrame 실습



spark_dataframe.ipynb

■ 데이터 파일 : people.json

```
{"name":"Michael"}
```

```
{"name":"Andy", "age":30}
```

```
{"name":"Justin", "age":19}
```

■ Creating a DataFrame

```
[2] from pyspark.sql import SparkSession
```

```
[3] # May take a little while on a local computer  
spark = SparkSession.builder.appName("Basics").getOrCreate()
```

```
[4] df = spark.read.json('people.json')
```

PySpark DataFrame 실습

■ Showing the data

```
[5] # Note how data is missing!  
df.show()
```

```
+-----+-----+  
|  age|   name|  
+-----+-----+  
| null|Michael|  
|   30|   Andy|  
|   19|  Justin|  
+-----+-----+
```

```
[6] df.printSchema()
```

```
root  
 |-- age: long (nullable = true)  
 |-- name: string (nullable = true)
```

```
[7] df.columns
```

```
['age', 'name']
```

```
[8] df.describe()
```

```
DataFrame[summary: string, age: string, name: string]
```


PySpark DataFrame 실습

■ Infer schema

```
[9] from pyspark.sql.types import StructField,StringType,IntegerType,StructType
```

```
[10] data_schema = [StructField("age", IntegerType(), True),StructField("name", StringType(), True)]
```

```
[11] final_struct = StructType(fields=data_schema)
```

```
[12] df = spark.read.json('people.json', schema=final_struct)
```

```
[13] df.printSchema()
```

```
root
 |-- age: integer (nullable = true)
 |-- name: string (nullable = true)
```

PySpark DataFrame 실습

Grabbing the data

```
[14] df['age']
```

```
Column<'age'>
```

```
[15] type(df['age'])
```

```
pyspark.sql.column.Column
```

```
[16] df.select('age')
```

```
DataFrame[age: int]
```

```
[17] type(df.select('age'))
```

```
pyspark.sql.dataframe.DataFrame
```

```
[18] df.select('age').show()
```

```
+-----+  
|  age |  
+-----+  
| null |  
|   30 |  
|   19 |  
+-----+
```

PySpark DataFrame 실습

■ Multiple Columns

```
[20] df.select(['age', 'name'])
```

```
DataFrame[age: int, name: string]
```

```
[21] df.select(['age', 'name']).show()
```

```
+-----+-----+
| age | name |
+-----+-----+
| null | Michael |
| 30 | Andy |
| 19 | Justin |
+-----+-----+
```

■ Creating new columns

```
[22] # Adding a new column with a simple copy
df.withColumn('newage', df['age']).show()
```

```
+-----+-----+-----+
| age | name | newage |
+-----+-----+-----+
| null | Michael | null |
| 30 | Andy | 30 |
| 19 | Justin | 19 |
+-----+-----+-----+
```

PySpark DataFrame 실습

■ More complicated operations to create new columns

```
[25] df.withColumn('doubleage',df['age']*2).show()
```

age	name	doubleage
null	Michael	null
30	Andy	60
19	Justin	38

```
[26] df.withColumn('add_one_age',df['age']+1).show()
```

age	name	add_one_age
null	Michael	null
30	Andy	31
19	Justin	20

■ PySpark 튜토리얼

<https://sparkbyexamples.com/pyspark-tutorial/>

데이터 웨어하우스(DW)의 한계

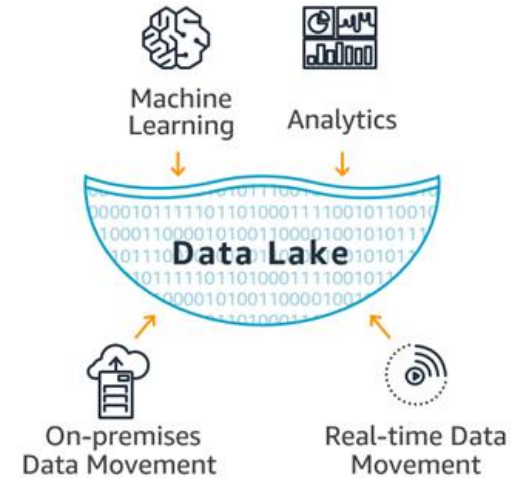
■ 데이터 웨어하우스의 한계

- SaaS 활용이 폭발적으로 증가하면서 수집 데이터의 다양성과 종류가 크게 증가(비정형, 반정형 데이터, 실시간 스트리밍)
- SaaS API에 사용되는 반정형 데이터(JSON, Avro, Protocol Buffer)를 처리하지 못함
- 비정형 데이터(binary, image, video, audio data)를 처리하지 못함
- 구조화된 데이터의 빈번한 스키마 변경에 대응이 어려움
- 데이터웨어하우스에 제공하는 SQL엔진과 저장 프로시저만 사용해야 하는 제약이 있음
- 스토리지와 처리영역이 결합되어 있어 확장성과 유연성에 제약이 큼
- 배치 중심 처리 방식으로 스트림 데이터가 처리가 어려움



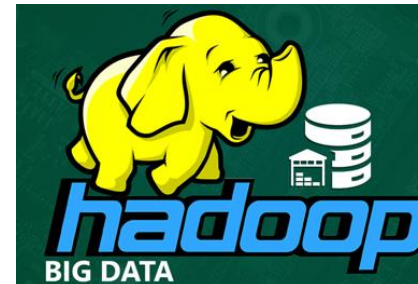
■ 데이터 레이크

- 방대한 양의 원시 데이터를 보관하는 storage repository로, 특정 주제나 목적 중심이 아닌 통합되지 않은 데이터 집합
- 과거 10년 동안 Hadoop이 데이터 레이크에 대한 실질적인 표준이었음



■ Hadoop의 단점

- 매우 복잡한 시스템
- 비즈니스 사용자가 비정형 형태를 이해하고 활용하기에 쉽지 않은 시스템
- 개발자도 잘 사용하려면 많은 노력이 필요
- 스토리지와 컴퓨팅이 분리되어 있지 않음
- 시스템 확장을 위한 하드웨어 추가/변경에 수개월이 소요될 수 있음



퍼블릭 클라우드 활용

하둡의 장점은 살리고 단점들을 보완해서 유연성을 훨씬 더 가져다 주는 솔루션이 클라우드와 함께 등장함

- 온디맨드 스토리지, 컴퓨팅 리소스 프로비저닝, 사용량 기반 요금 지불 모델을 갖춘 퍼블릭 클라우드의 등장으로 데이터 레이크 설계가 Hadoop의 한계를 뛰어 넘음
- 퍼블릭 클라우드를 통해 데이터 레이크는 설계 및 확장성 측면에서 더 많은 유연성을 포함하고 필요한 지원(support)의 양을 대폭 줄이는 동시에 비용 효율성을 높임
- 퍼블릭 클라우드의 출현은 분석 데이터 시스템에 관한 모든 것을 바꾸어 놓음
- AWS EMR은 관리형 서비스로 제공되며 AWS에서 Hadoop 및 Spark 작업을 실행할 수 있음

Elastic resources

Modularity(Storage and compute are separate)

Pay per use

Cloud turns CAPEX into OPEX

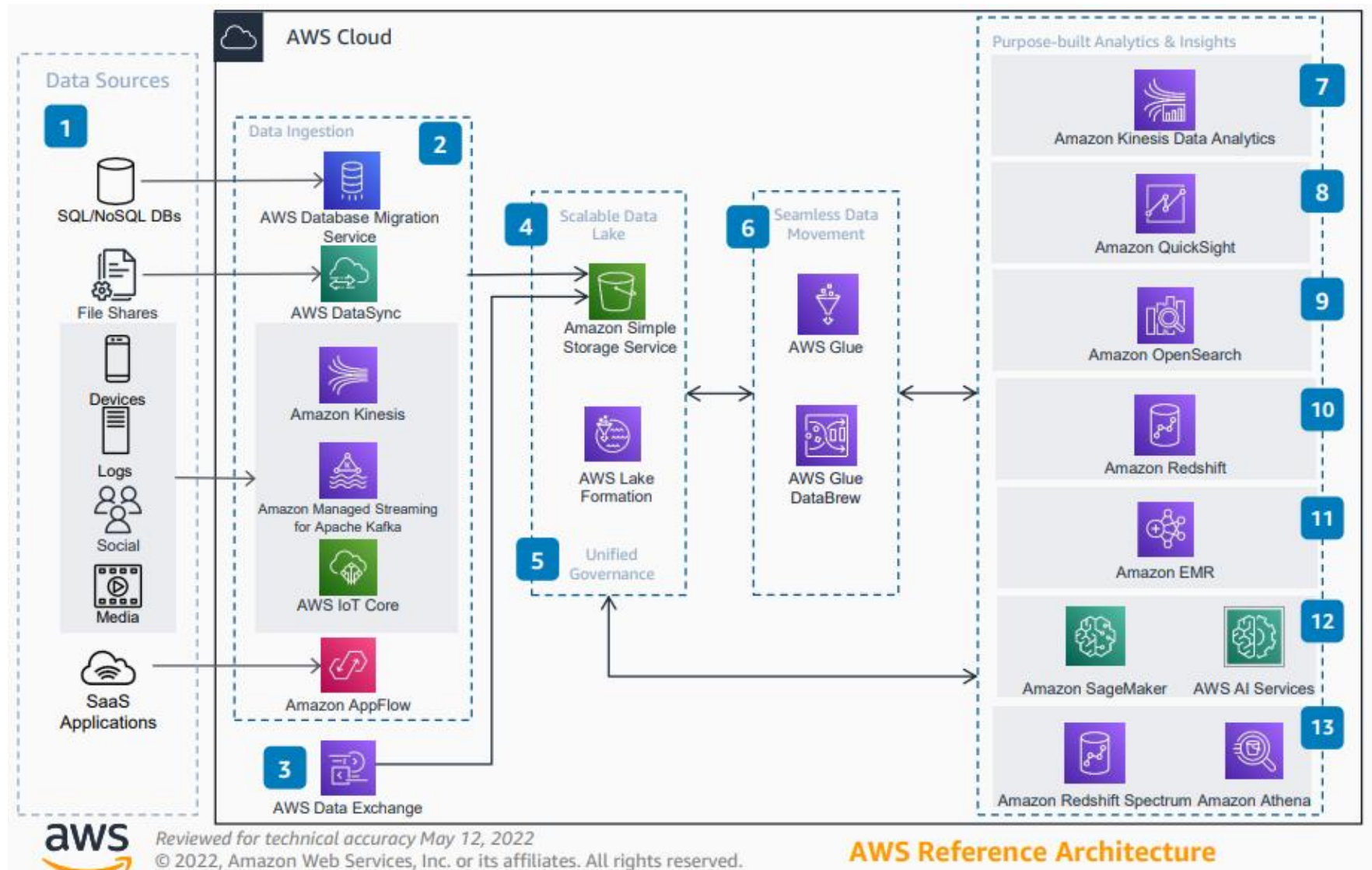
Managed services are the norm

Instant availability

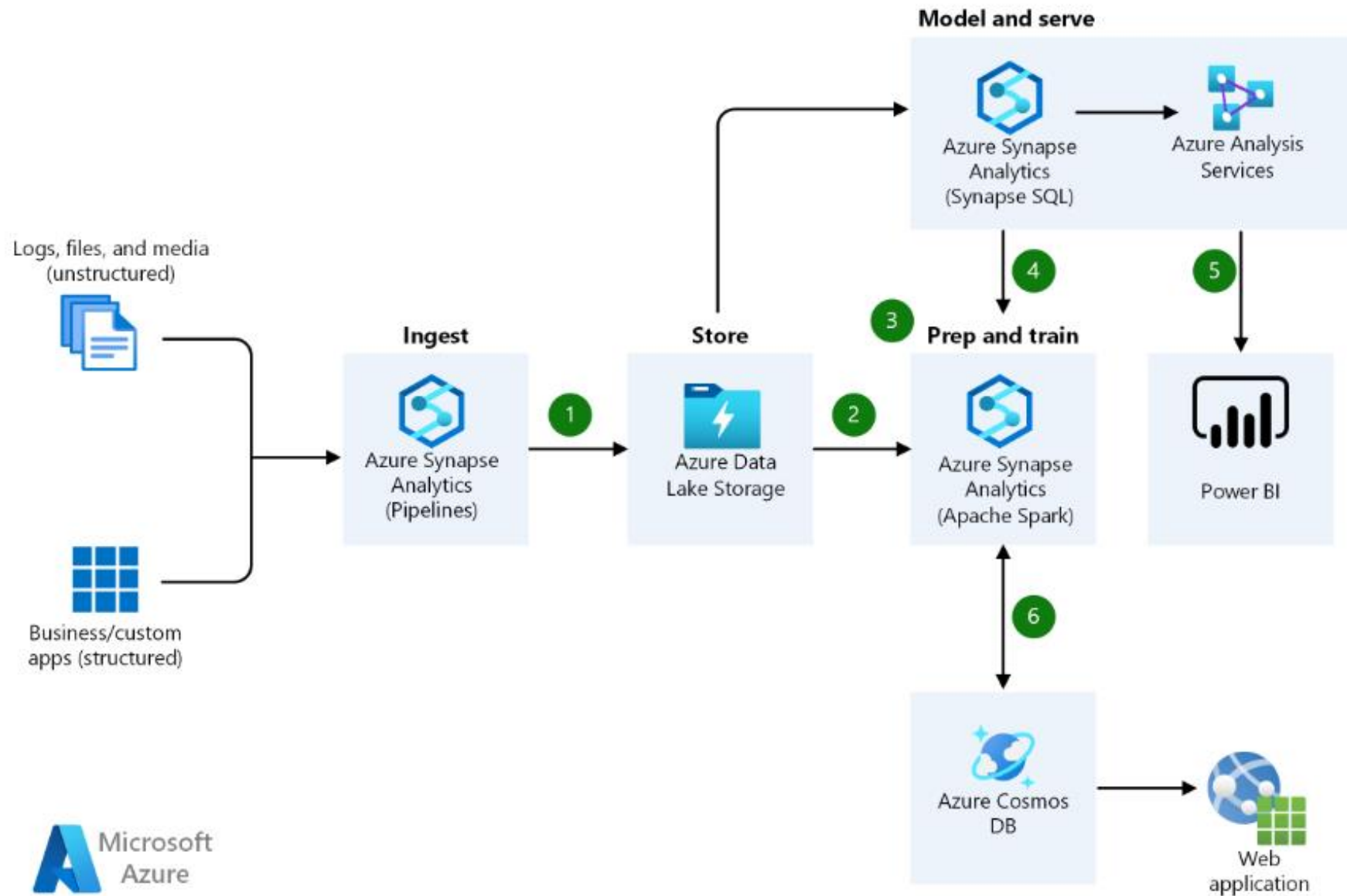
A new generation of cloud-only processing frameworks

Faster feature introduction

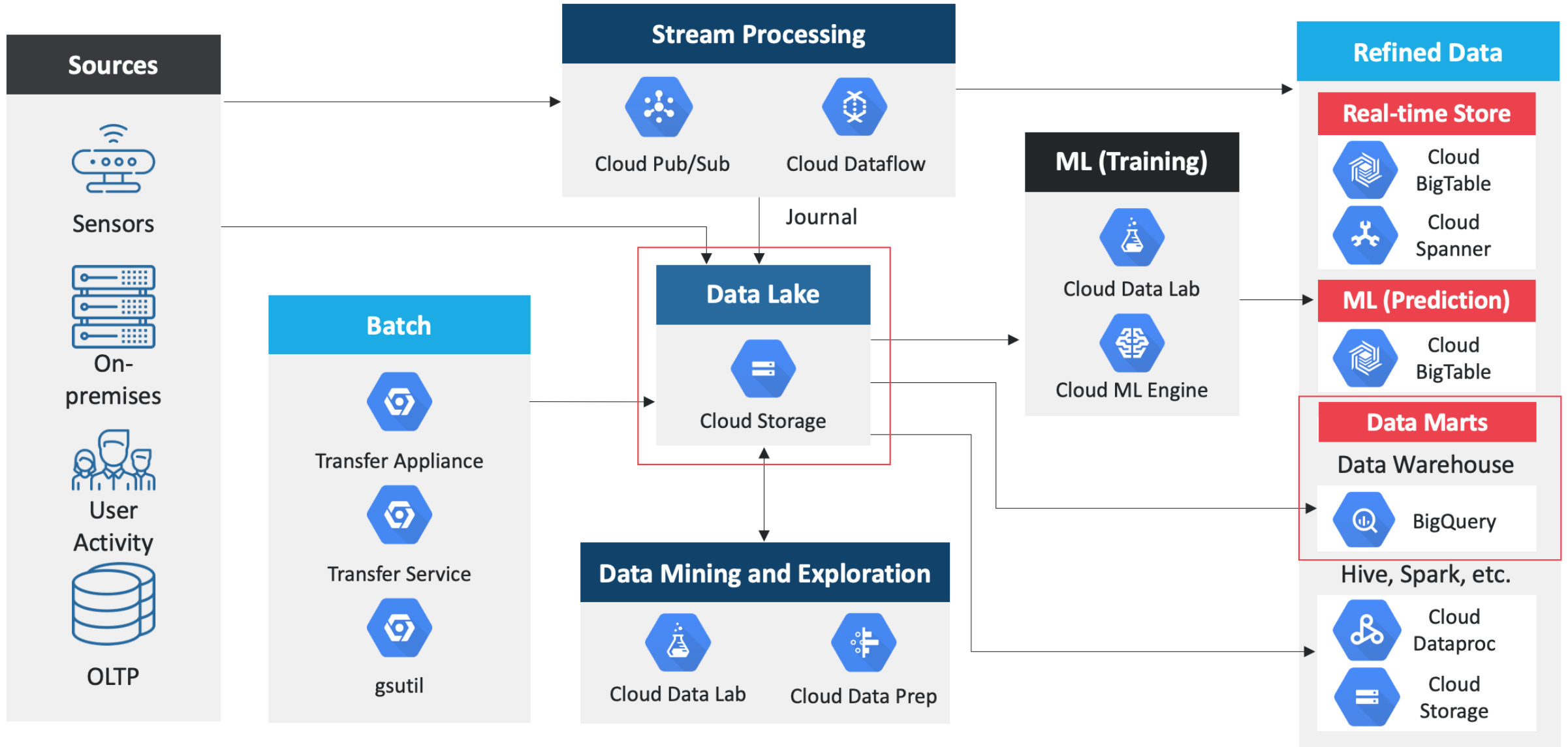
퍼블릭 클라우드 활용 - AWS (Amazon Web Services)



퍼블릭 클라우드 활용 - Microsoft Azure



퍼블릭 클라우드 활용 - GCP (Google Cloud Platform)



THANK YOU

kgpark88@gmail.com