Hadoop Ecosystem

19기 엔지니어링 박선홍

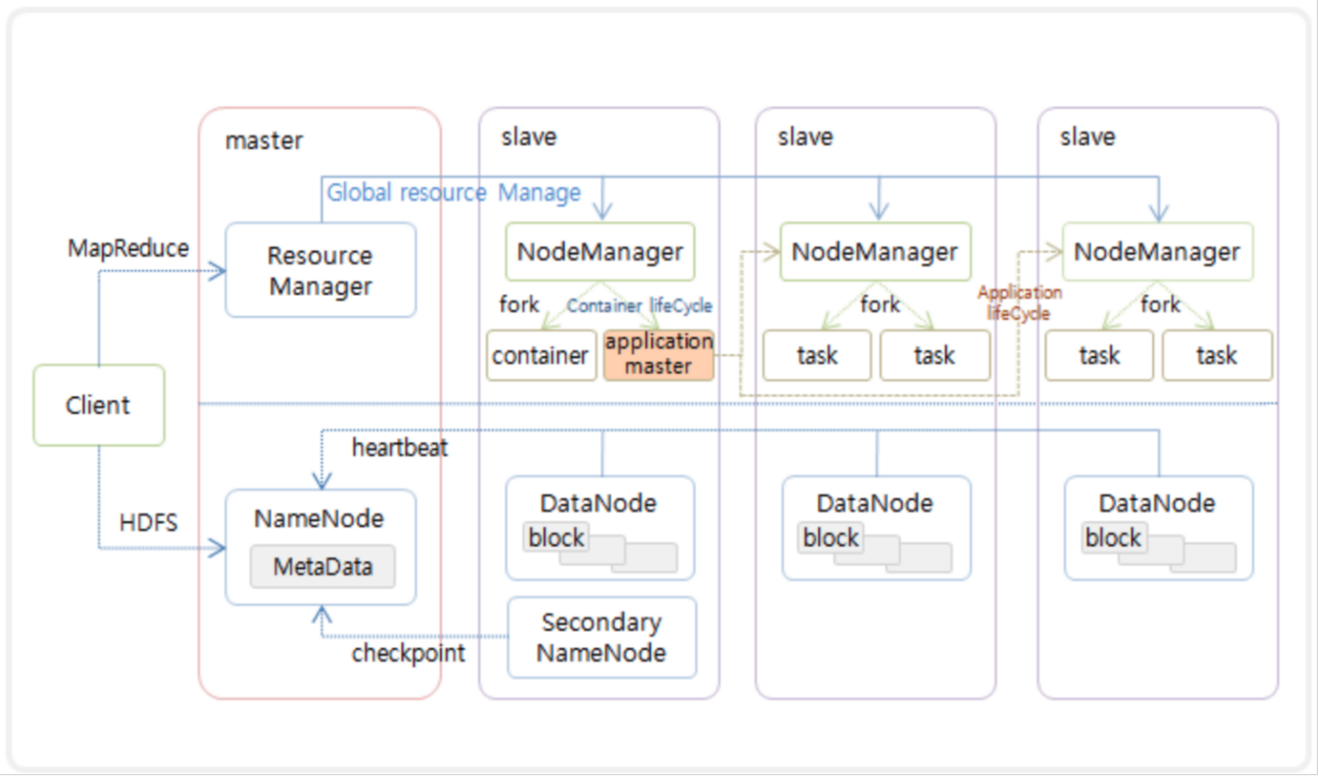
Hadoop

- 대용량의 데이터를 **분산처리** 할 수 있는 Java 기반 오픈 소스 프레임워크
- 하나의 대형 컴퓨터를 활용하여 데이터를 수집, 처리하는 대신, 하둡 서버가 설치된 다른 상용 하드웨어와 함께 클러스터링 하여 **대규모의 데이터 세트를 병렬적으로 분석**할 수 있음
- 정형 데이터 뿐만 아니라 **비정형 데이터**도 다룰 수 있음

Hadoop 특징

1. 분산저장 : HDFS (하둡 분산 파일 시스템)
2. 분산처리 : 맵리듀스 (어플리케이션 처리할 수 있는 메커니즘)
3. 선형적 확장(Scale out) : 기존 시스템의 용량이 부족하면 용량 늘리는 것이 아닌 노드를 더 추가
4. 오픈 소스 기반
5. 장애 허용 시스템(Fault-tolerant): 하나 이상 컴퓨터가 고장나는 경우에도 시스템이 정상 동작

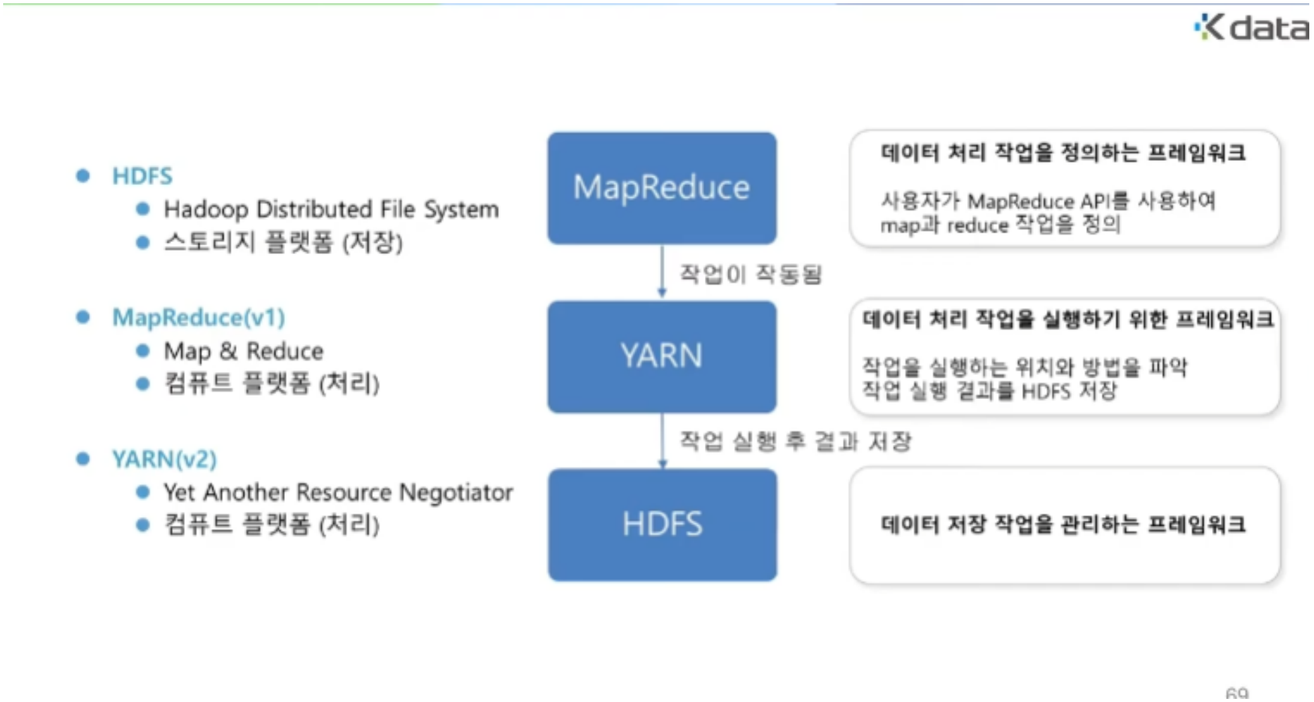
Hadoop Architecture



마스터 - 슬레이브 구조

Master : 전체적인 행위 조절, 제어
Slave : 데이터를 저장 및 실행

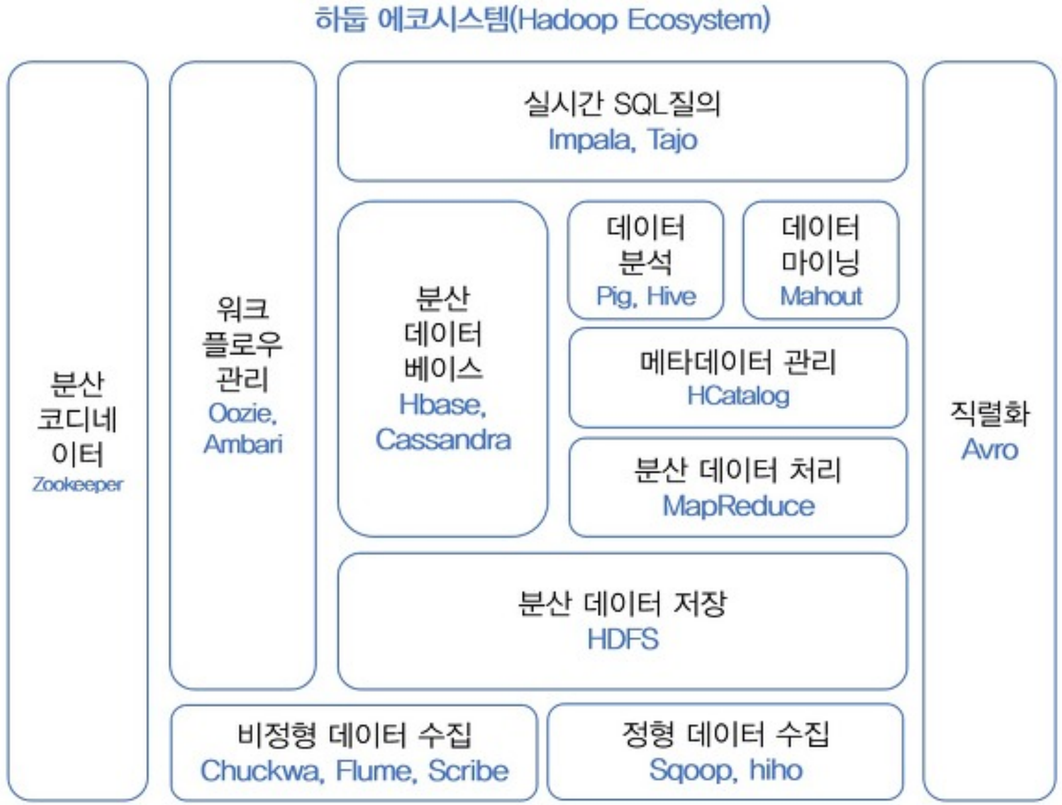
Hadoop Architecture

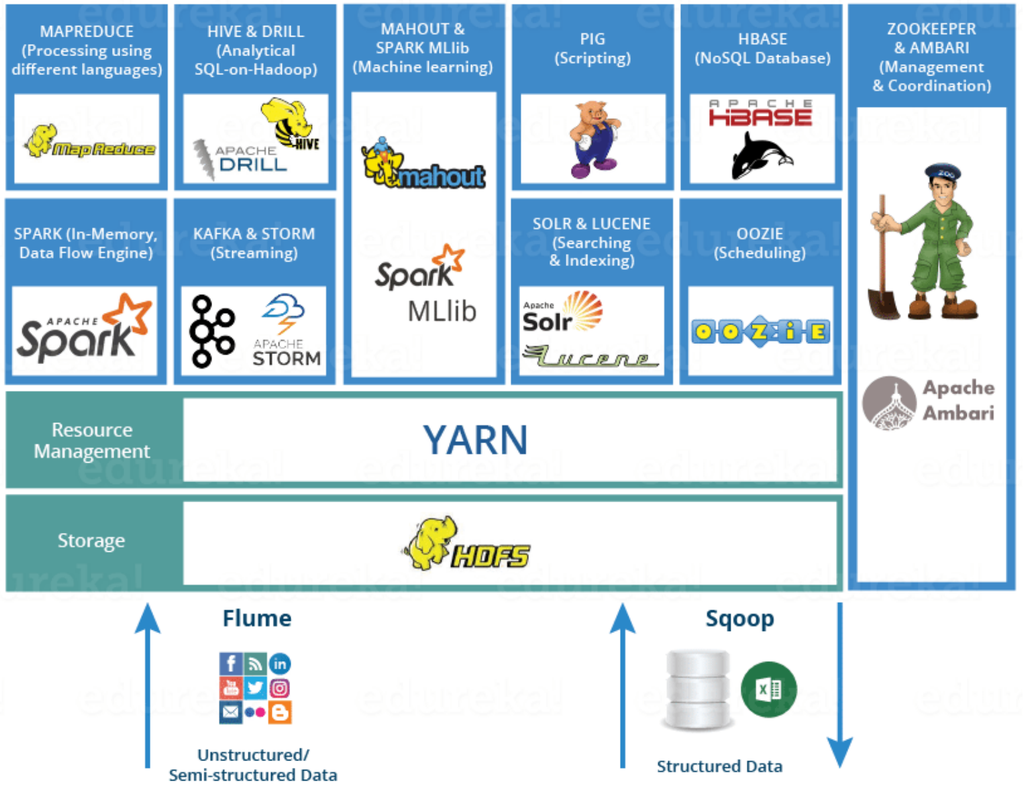


- 맵리듀스 : 데이터 처리 작업
- YARN : 클러스터한테 데이터 던졌을때, 어떻게 실행할건지 등 자원관리 담당
- HDFS : 데이터 저장, 관리

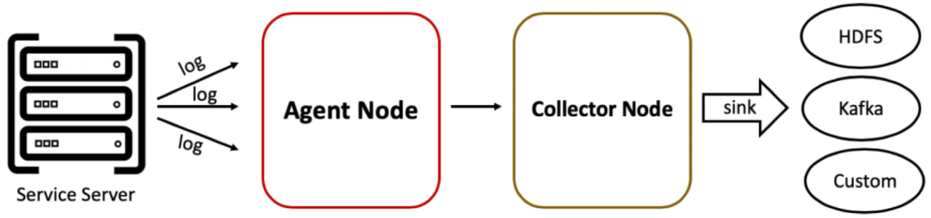
Map : *key* - *value*의 형태로 흩어져 있는 정보를 연관성 있는 데이터 분류로 묶는 작업
Reduce : Map화 한 작업 중 중복 데이터를 제거하고 원하는 데이터를 추출

Hadoop Ecosystem



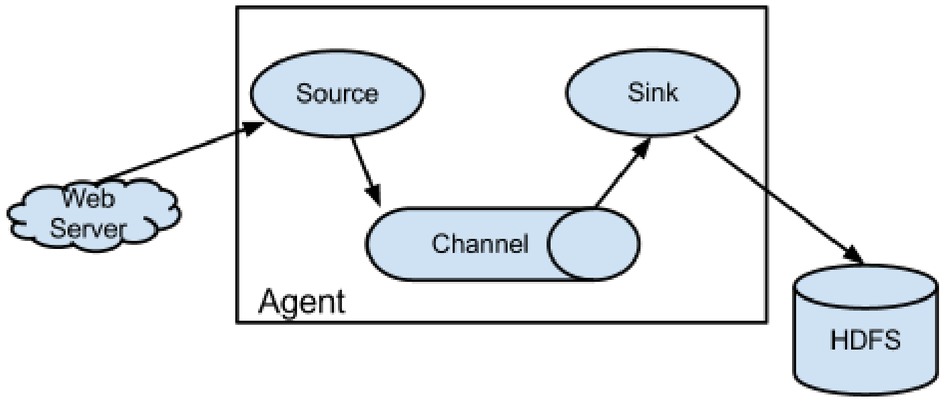


1. 데이터 수집 - Flume



- 클라우데라에서 처음 개발 돼, 아파치 소프트웨어 재단으로 이관
- 연속적으로 생성되는 **데이터 스트림**(로그 파일, 소셜 미디어 데이터, 이메일 메시지 등)을 수집 및 전송하고 데이터베이스에 저장할 수 있는 도구
- Multiplexing 구조로 Flume을 구동시키면 동시에 여러개의 싱크로 로그를 전달 할 수도 있음
- 많은 기업들에서 실제 서비스 로그데이터 관리를 위해 사용

1. 데이터 수집 - Flume

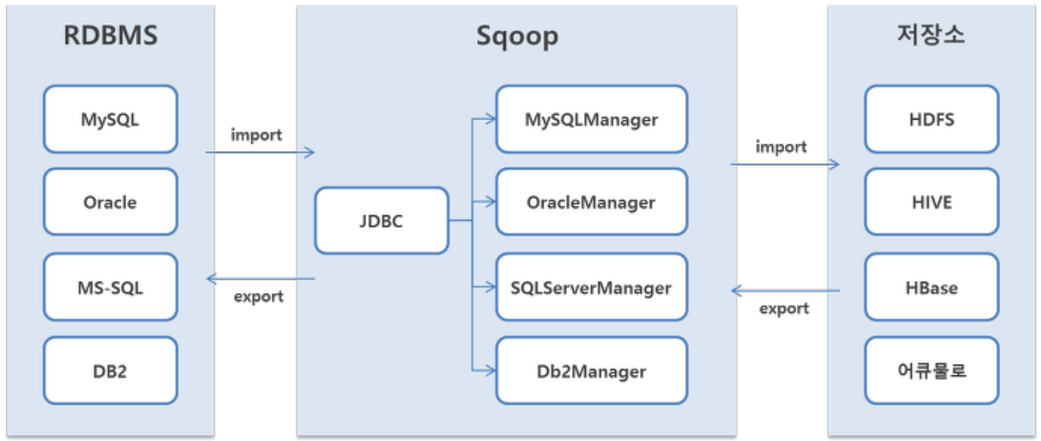


1. 소스(Source) : 외부 데이터 소스에 설치되는 에이전트
   - 데이터가 발생하는 소스로부터 (로그)데이터 수집 담당
   - Channel에 이벤트 입력
   - Avro, Thrift, Exec, HTTP 등

2. 채널(Channel) : 소스와 싱크 간에 데이터를 받는 통로
   - 소스 데이터를 싱크로 전달하는 데이터 큐(Queue)
   - 소스의 속도와 싱크의 속도를 조절하는 일종의 버퍼(Buffer)
   - 이벤트의 transaction 관리
   - Memory, JDBC, Kafka, File 등

3. 싱크(Sink) : 데이터 목적지에 설치되는 에이전트
   - 수집한 데이터를 외부로 보내는 역할
   - 데이터를 HDFS, 로컬 파일, 혹은 다른 Flume 에이전트 등에 전달하는 모듈
   - 하나의 싱크는 오직 한 채널에서만 데이터를 전달받을 수 있음
   - HDFS, Hive, Logger, Avro, HTTP 등

1. 데이터 수집 - Sqoop

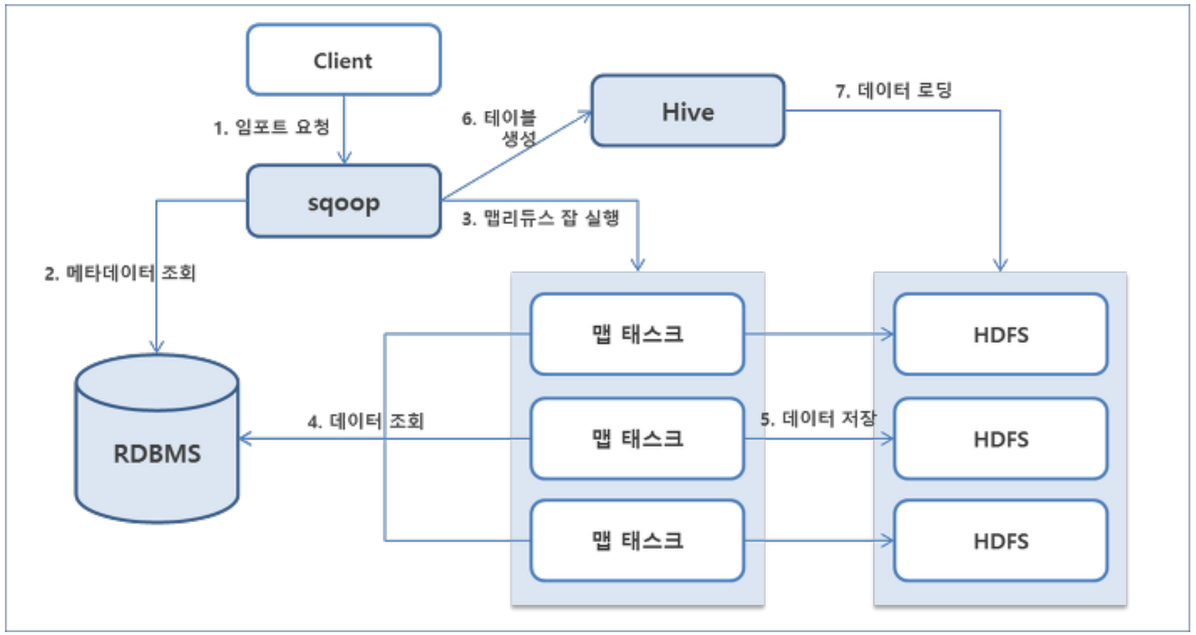


- 관계형 데이터베이스(RDB)와 HDFS사이의 양방향 데이터 전송을 위해 설계된 툴

- 배경 : 새롭게 생성되는 데이터(ex. 로그데이터 등)는 Flume, Kafka를 통해 HDFS로 전송될 수 있지만, 기존에 존재하는 RDB의 데이터 또한 HDFS로 전송할 필요성 느낌

- YARN, MapReduce, HDFS위에서 동작
- CLI만으로도 sqoop 실행, 제어 가능
- import(RDB -> HDFS), export(HDFS -> RDB)

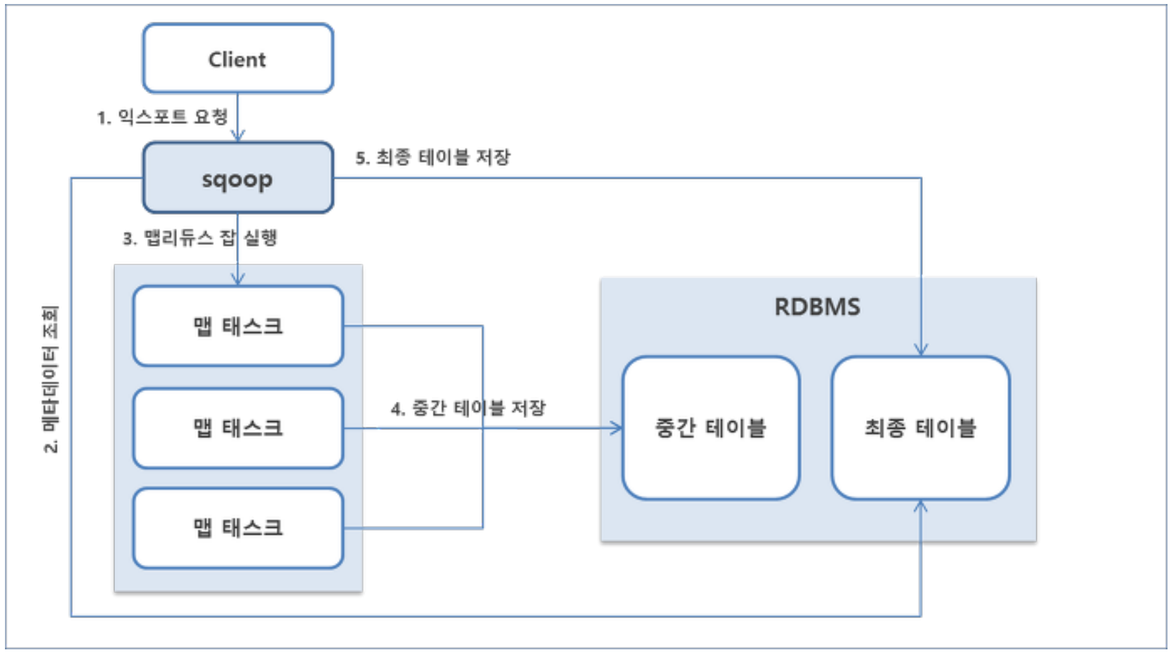
1. 데이터 수집 - Sqoop



Import 과정

1. Client가 Sqoop에 Import 요청.
2. Sqoop은 데이터베이스에서 해당 테이블의 메타데이터를 조회해 ORM(Object Relational Mapping) 클래스 생성.
3. Sqoop은 ORM클래스가 정상적으로 생성되면 맵리듀스 잡 실행을 요청. Sqoop은 맵 태스크의 출력 결과를 Import에 사용하기 때문에 리듀스 태스크는 수행하지 않음
4. 맵 태스크는 데이터베이스에 JDBC로 접속한 후 SELECT 쿼리를 실행.
5. 맵 태스크는 쿼리문을 실행한 결과를 HDFS에 저장. 전체 맵 태스크가 종료되면 Sqoop은 Client에게 작업이 종료되었다고 알림
6. 사용자가 설정한 하이브 테이블 생성
7. 맵 태스크에 저장된 결과를 하이브 테이블의 데이터 경로로 로딩

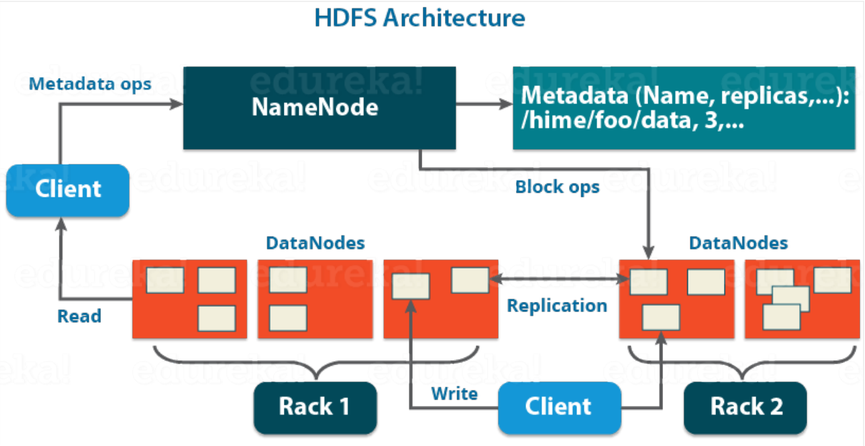
1. 데이터 수집 - Sqoop



Export 과정

1. Client는 Sqoop에 Export 요청
2. Sqoop은 데이터베이스에서 메타데이터를 조회한 후 맵리듀스 잡에서 사용한 ORM클래스 생성
3. Sqoop은 데이터베이스의 중간 테이블의 데이터를 모두 삭제한 후 맵리듀스 잡 실행
4. 맵 태스크는 HDFS에서 데이터를 조회한 후 INSERT 쿼리문을 만들어 중간 테이블에 데이터를 입력. 이때 쿼리문은 레코드당 한번씩 실행하는 것이 나이라 천개 단위로 배치 실행.
5. Sqoop은 맵리듀스 잡이 정상적으로 종료되면 중간 테이블의 결과를 최종 테이블에 입력.

2. 데이터 저장 - HDFS



- 하둡 분산 파일 시스템(HDFS)은 하둡 프레임워크를 위해 자바 언어로 작성된 분산 확장 파일 시스템
- HDFS는 여러 기계에 대용량 파일을 나눠서 저장 => 데이터들을 여러 서버에 중복해서 저장함으로써 **데이터 안정성**을 얻음
- 많은 수의 작은 파일보다, 적은 수의 큰 파일을 다루는데 더 적합

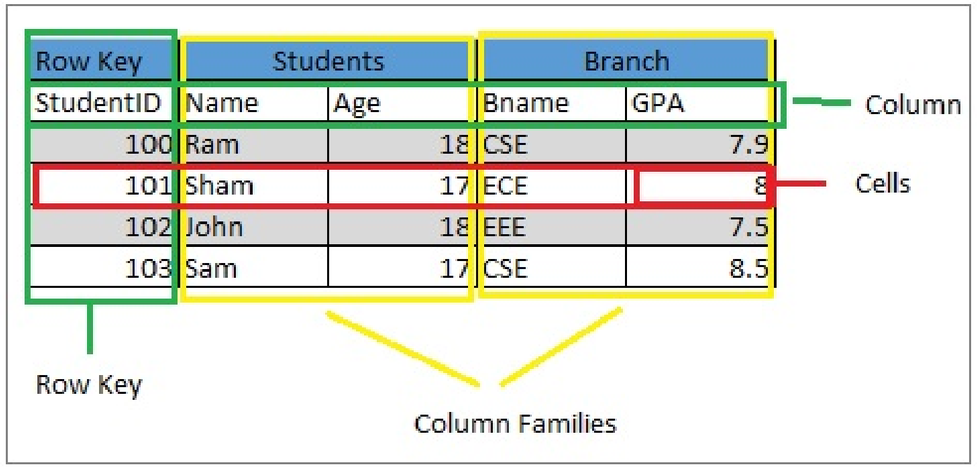
- 특징
  1. HDFS는 데이터를 저장하면 다수의 노드에 복제 데이터도 함께 저장해서 **데이터 유실을 방지**
  2. HDFS에 파일을 저장하거나, 저장된 파일을 조회하려면 스트리밍 방식으로 데이터에 접근해야 함
  3. 한번 저장한 데이터는 수정할 수 없고, 읽기만 가능해서 데이터 무결성을 유지
  4. 데이터 수정은 불가능하지만 파일 이동, 삭제, 복사할 수 있는 인터페이스를 제공

2. 데이터 저장 - HBase

- 배경
하둡은 순차적으로 만들어진 데이터에만 엑세스할 수 있었으며, 간단한 작업을 하려면 전체 데이터셋을 검색해야 함
어떤 작업을 통해 새로운 데이터셋을 만들어내면 이것들을 순차적으로 처리해야 함
=> **단일 시간 단위**로 데이터를 접근하기 위해 탄생

- 특징
1. HDFS 위에 만들어진 **분산 컬럼 기반**의 데이터베이스(Columnar DB)
2. HDFS의 데이터에 대한 **실시간 임의 읽기/쓰기** 기능을 제공
3. HBase를 이용해서 HDFS에 있는 데이터에 랜덤하게 액세스와 읽기를 할 수 있음
4. 고정된 schema를 요구하지 않으므로, 사전정의된 모델을 따르지 않고도 새로운 데이터를 쉽게 추가 가능

2. 데이터 저장 - HBase



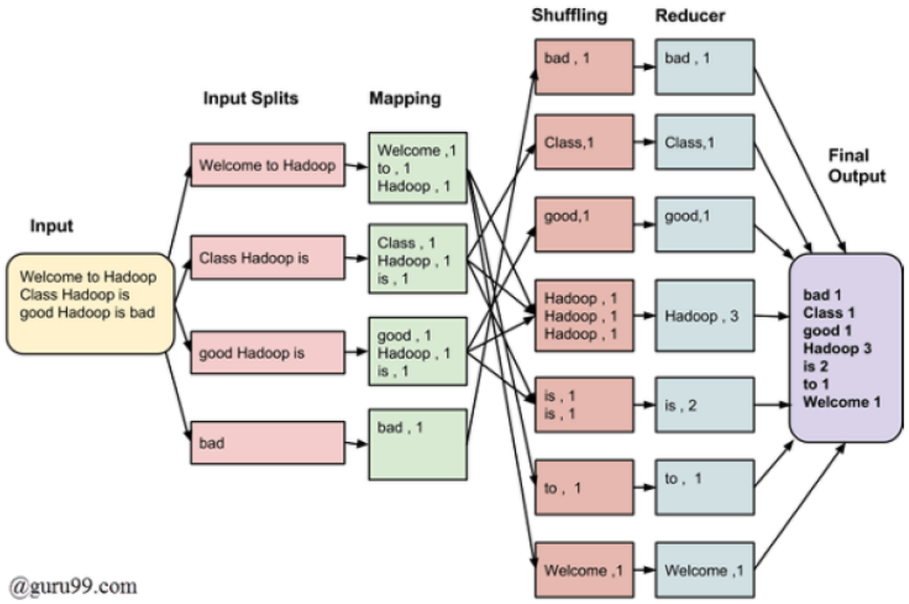
저장 원리

- 구성 : Row key와 Column family

1. HBase의 테이블들은 row로 정렬되어 있으며, 테이블 schema를 정의할 때는 Column Family만 정의하면 된다
2. Column Family : key - value가 한 쌍으로 이루어짐
3. 테이블인 이런 column family를 여러개 가지고 있고 각 column family는 여러개의 column을 가지고 있음
4. column의 값은 디스크에 이어서 연속적으로 저장되며, 각 cell의 값에는 **timestamp**가 포함

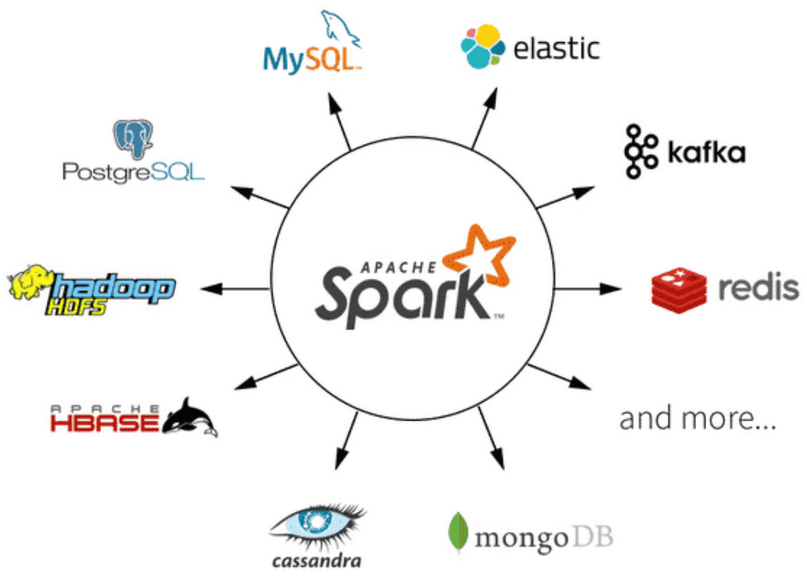
- Timestamp : Value와 함께 쓰여지며, 값의 version을 위해 식별자로 활용함. 기본적으로 데이터가 쓰여질 때 리전서버(RegionServer) 상의 시간을 기록함.

3. 데이터 처리 - Mapreduce



- 구글에서 대용량 데이터 처리를 **분산 병렬 컴퓨팅** 에서 처리하기 위한 목적으로 제작하여 2004년 발표한 소프트웨어 프레임워크
- HDFS상에서 동작하는 가장 기본적인 분석 기술로 간단한 단위작업을 반복할 때 효율적임
- 하둡에서 분산처리를 담당하는 맵리듀스 작업은 크게 **맵(Map)**과 **리듀스(Reduce)**로 나누어져 처리됨
- 맵, 리듀스 작업은 병렬로 처리가 가능한 작업으로, 여러 컴퓨터에서 동시에 작업을 처리하여 속도를 높일 수 있음

3. 데이터 처리 - Spark

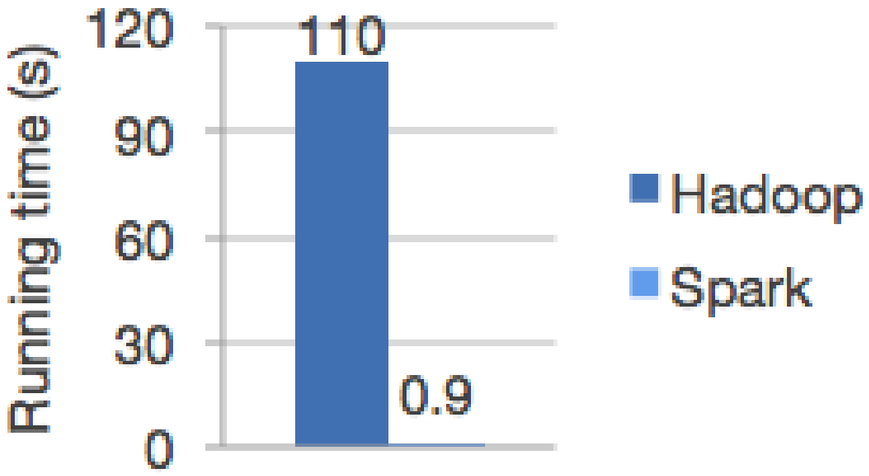


- 2009년 버클리 대학의 AMPLab에서 시작됐으며, 오픈소스이며 범용적인 목적을 지닌 인메모리 기반의 대용량 데이터 고속 처리 엔진
- 하둡 MapReduce 보다 발전된 새로운 분산병렬처리 Framework
- 배치 처리, SQL 질의 처리, 스트리밍 데이터, 머신러닝, 그래프 라이브러리 처리와 같은 다양한 작업을 수용할 수 있도록 설계되어 있음

- Spark는 클러스터들을 관리하는 **Cluster Manager**와 데이터를 분산 저장하는 **Distributed Storage System**이 필요

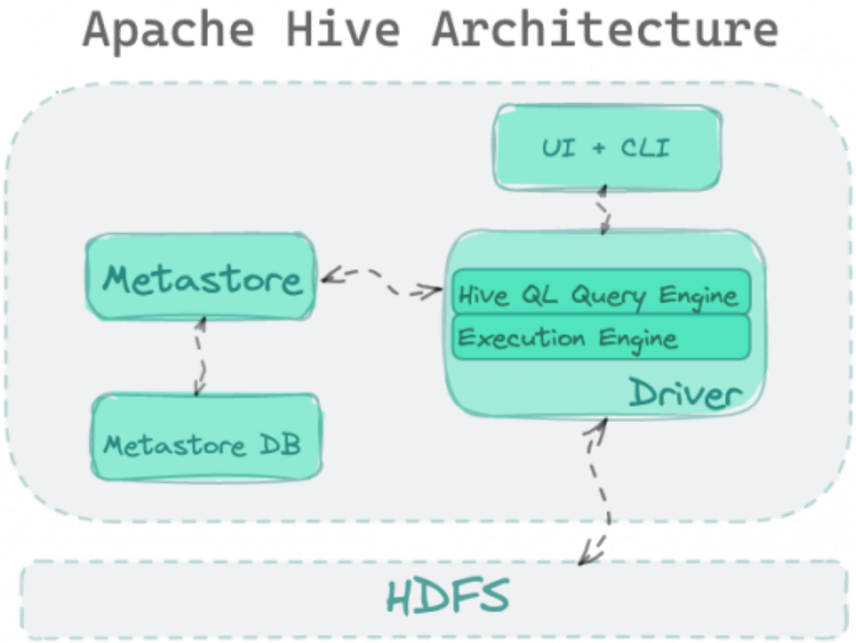
자주 사용되는 Cluster Manager : *Hadoop의 YARN이나 Apache Mesos*
적용할 수 있는 Distributed Storage System : *HDFS, MapR-FS(MapR File System), Cassandra* 등이 있음
=> 압축 알고리즘(zlib 등) 지원, spark와 같은 머신에서 구동가능하다는 점에서 HDFS 가장 많이 사용

3. 데이터 처리 - Spark



1. Speed
인메모리(In-Memory) 기반의 빠른 처리
Disk I/O를 기반으로 한 hadoop보다 약 100배정도 빠름(메모리작업)
2. Ease of Use
다양한 언어 지원(Java, Scala, Python, R, SQL)을 통한 사용의 편이성
3. Generality
SQL, Streaming, 머신러닝, 그래프 연산 등 다양한 컴포넌트 제공
4. Run Everywhere
YARN, Mesos, Kubernetes 등 다양한 클러스터에서 동작 가능
HDFS, Casandra, HBase 등 다양한 파일 포맷 지원

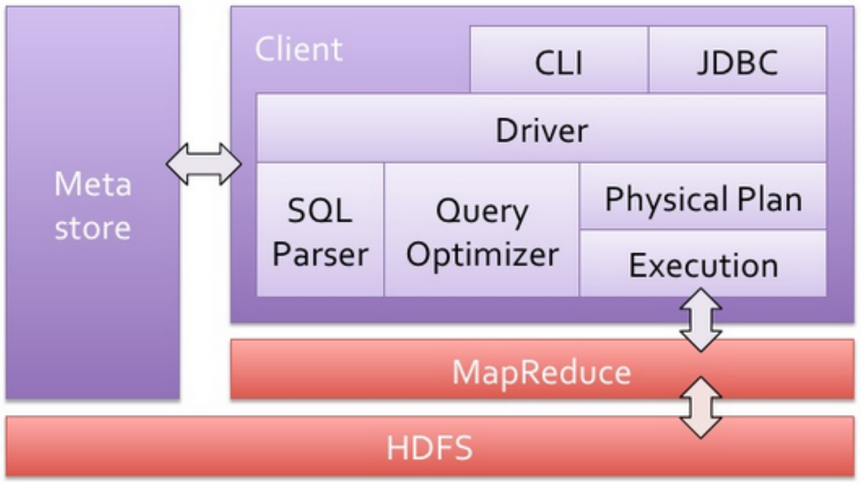
3. 데이터 처리 - Hive



1. 배경
- 하둡은 **복잡한 MapReduce 코드**를 작성해야 되고, 테이블과 같은 논리적인 개념이 없어서 **파일 단위 또는 디렉토리 단위로 데이터를 관리**해야 함
- 이를 해결하기 위해 페이스북에서 **SQL과 매우 유사한 방식**으로 하둡 데이터에 접근성을 높인 Hive를 개발하게 됨
- 페이스북에서 개발이 시작되어, 넷플릭스와 같은 다른 기업도 기여

2. 특징
- 데이터 웨어하우징과 같은 SQL 기반의 접근을 통한 데이터 처리, 데이터베이스, 테이블, 파티션과 같은 논리적 레벨의 카탈로그 및 메타스토어를 제공
- Java를 잘 모르는 데이터 분석가들도 쉽게 하둡 데이터를 분석할 수 있도록 도와줌
- HiveQL 이라고 불리는 SQL 같은 언어를 제공하며 맵리듀스의 모든 기능 지원

3. 데이터 처리 - Hive



- 구성요소

1. CLI : 사용자가 하이브 쿼리를 입력하고 실행할 수 있는 인터페이스
2. JDBC / ODBC Driver : 하이브의 쿼리를 다양한 데이터베이스와 연결하기 위한 드라이버를 제공
3. Query Engine : 사용자가 입력한 하이브 쿼리를 분석해 실행 계획을 수립하고 Hive QL을 맵리듀스 코드로 변환 및 실행
4. MetaStore : 하이브에서 사용하는 테이블의 스키마 정보를 저장 및 관리하며, 기본적으로 Derby DB가 사용되나 다른 DBMS(MySQL, PostgreSQL 등)로 변경이 가능

4. 분산 코디네이터 : Zookeeper

1. 코디네이션 시스템의 필요성

과거 : 한 대의 컴퓨터에서 동작하는 단일 프로그램이 대다수
현재 : 빅데이터와 클라우드 환경에서 대규모의 시스템들이 동작, 많은 서버와 인프라, 수많은 어플리케이션으로 구성
=> 개별적인 시스템들을 각각 조율해야 하는 코디네이션 시스템의 수요 증가

BUT, 코디네이션 시스템을 작성하는 것은 아주 까다롭고 복잡한 과정
=> 코디네이션 시스템을 대충 작성하거나, 메인 로직을 제대로 못 짜거나

2. 분산 코디네이션 시스템을 제대로 작성하지 못하게 될 경우

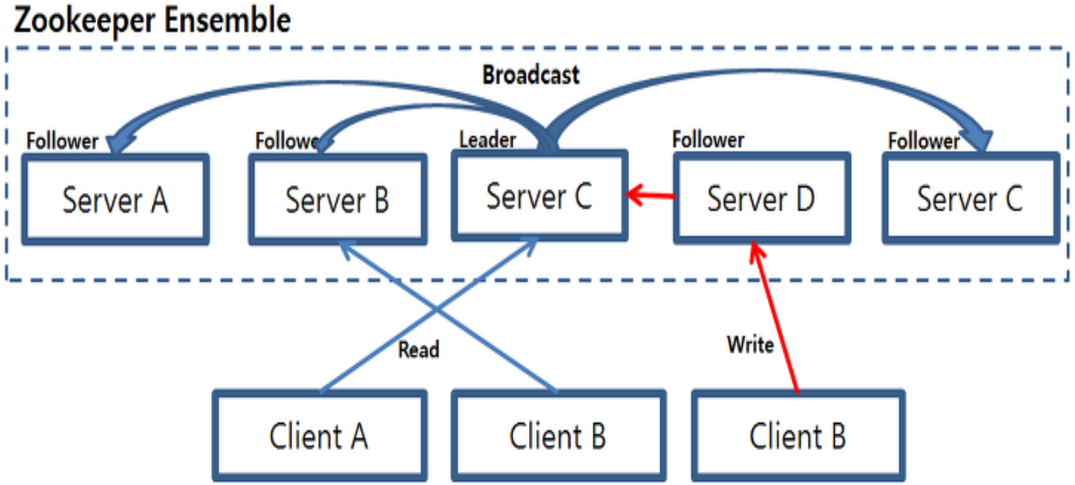
- **단일 장애점**(Single Point of Failure)가 발생하게 된다(시스템의 신뢰성을 떨어뜨리는 요소)
- 어플리케이션이 공유하고 있는 클러스터 자원에 무분별한 쓰기 동작(write operation)으로 인해 **경쟁상태**(Race condition)가 일어나게 될 가능성이 커짐

4. 분산 코디네이터 : Zookeeper

주키퍼는(Zookeeper) : 분산 코디네이션 서비스를 제공하는 오픈소스 프로젝트
- 개발자가 코디네이션 로직보다는 비즈니스 핵심 로직에 집중하게끔 지원하는 역할
- 하둡 에코시스템의 프로그램들을 전체적으로 컨트롤하는 통제 프로그램
- 분산 시스템 간의 정보 공유 및 상태 체크, 동기화를 처리

- 주키퍼의 사용용도
1. 설정 관리 (Configuration Management) : 클러스터의 설정 정보를 최신으로 유지하기 위한 조율 시스템으로 사용
2. 클러스터 관리 (Cluster Management) : 클러스터의 서버가 추가 or 제외될 때 그 정보를 클러스터 안 서버들이 공유하는데 사용
3. 리더 채택 (Learder Selection) : 다중 어플리케이션 중에서 어떤 노드를 리더로 선출할지 정하는 로직을 만드는데 사용
4. 락, 동기화 서비스(Locking and synchronization service) : 빈번하게 클러스터 쓰기 연산이 발생할 경우 경쟁상태에 들어가 데이터 불일치를 발생시킬 수 있음. 이 때, 클러스터 전체를 동기화해 락을 거는 것. 경쟁상태에 들어가는 것을 사전에 방지

4. 분산 코디네이터 : Zookeeper

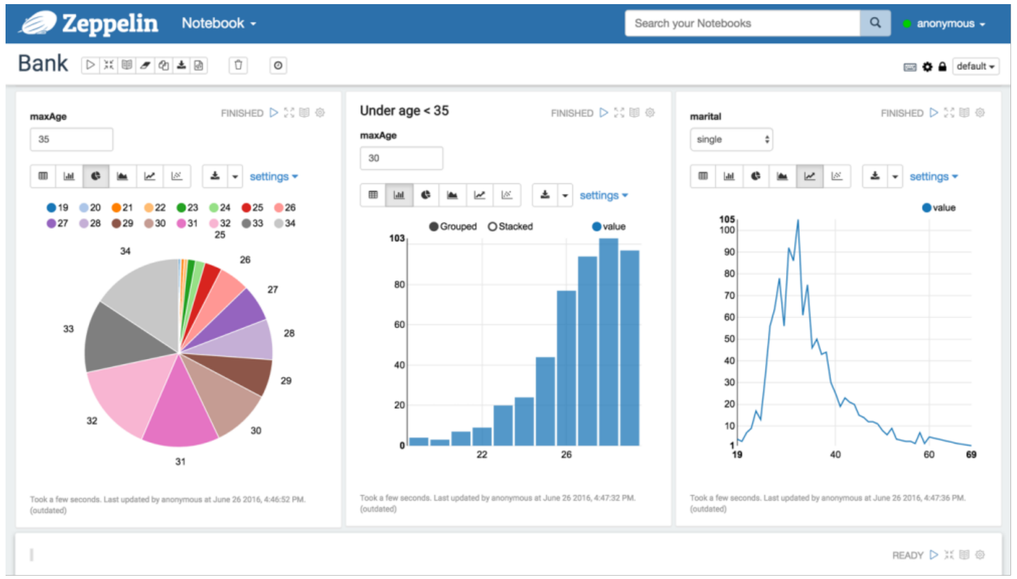


주키퍼 Architecture

- 클라이언트들은 주키퍼 서버들로 이루어진 앙상블(Ensemble)에 접근하여 **znode**의 데이터를 읽거나 데이터를 업데이트
- 앙상블안의 주키퍼 서버들은 조율된 상태이며 항상 동일한 데이터를 가지고 있음(어느 서버에서 데이터를 읽어도 동일)

- 주키퍼 서버에 쓰기 동작을 할 경우에, 클라이언트는 특정 서버에 접속하여 그 서버의 데이터를 업데이트, 업데이트 된 서버는 leader의 역할을 맡은 주키퍼 서버에 그 데이터를 알리고 업데이트하죠.
- leader 서버가 업데이트 감지하면 그 정보를 다른 곳에 **브로드캐스트(Broadcast)** 형식으로 알림
- 나머지 Follower 주키퍼 서버들은 그 내용을 갱신하여 전체 서버들의 데이터들이 일관된 상태로 유지된 상태로 있게 됩니다.

5. 시각화 : Zeppelin



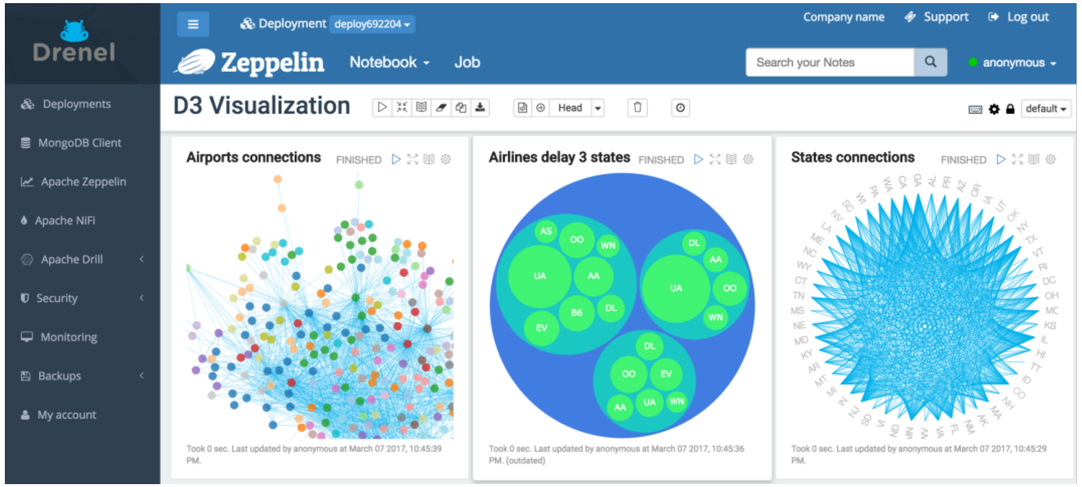
- 한국 태생 apache top-level 프로젝트
- Spark를 통한 데이터 분석의 불편함을 Web기반의 Notebook을 통해서 해결해보고자 만들어진 시각화 어플리케이션
- Zeppelin을 이용하여 Web에서 Python, Scala등의 다양한 언어를 섞어가며 분석 코드를 짤수 있고 이 결과를 바로 Graph로 시각화하여 볼수 있음
- spark 뿐만이 아닌 Livy, Cassandra, Lens, SQL 등등의 다른 데이터 분석도구나 데이터 베이스에 접근하여 쿼리하는 것을 쉽게 할 수 있는 **확장 기능**

5. 시각화 : Zeppelin


```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.2
      /_/

Using Python version 2.7.11 (default, Jun 15 2016 15:21:11)
SparkContext available as sc, HiveContext available as sqlContext.
>>> _
```

1) Spark 환경



2) Zeppelin

Sqoop 실습

https://brook-cod-143.notion.site/1-Sqoop-77a48a324387467c9558a61b83952bcc

참고자료

https://gritmind.blog/2020/09/29/sqoop_start/
https://loustler.io/data_eng/what-is-hbase/
https://mangkyu.tistory.com/128
https://engkimbs.tistory.com/660
https://j3sung.tistory.com/590
https://medium.com/apache-zeppelin-stories/
https://veil-prince-978.notion.site/Hadoop-Eco-System-58c20fb4d3c549f1959886bfa880102d(이전 hadoop ecosystem 세션 자료)

Thank You