데이터 엔지니어링

데이터 적재 개요

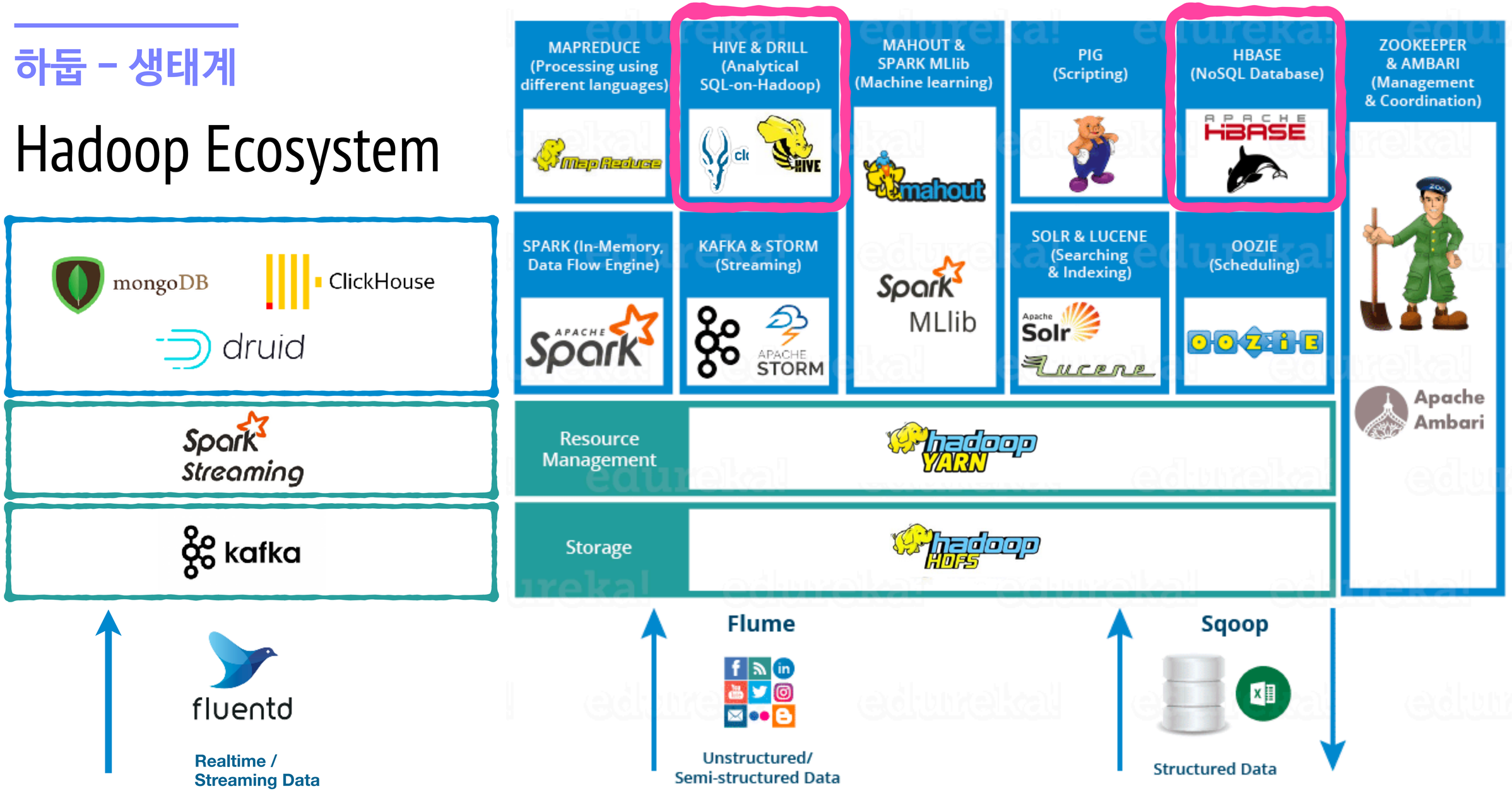

psyoblade


psyoblade

NCSOFT®

하둡 – 생태계

Hadoop Ecosystem



목차

Introduction

데이터 적재 서비스

데이터 적재 서비스 – 개요

Data Serving Layer

2006년 하둡의 등장 이후에 대용량 데이터 분산 저장 및 처리는 일반화되었고 누구나 사용할 수 있는 도구로 자리매김 하였습니다. 뿐만 아니라 하둡의 저장과 처리 성능의 제약을 극복하기 위해 다양한 오픈소스들과 엔진이 출현하게 되면서 하둡 생태계는 더욱 크게 성장할 수 있었습니다. 하둡의 저장 시스템은 블록 단위의 파일 저장소이기 때문에 배치 처리에는 적합하지만 레코드 단위의 처리나 변경 및 빠른 입출력에는 취약할 수 밖에 없습니다. 가장 널리 알려져 있는 분산처리 기술 중 하나가 MapReduce 이며 이 기술 기반의 데이터 변환 및 적재 도구가 Hive 이며 HiveQL 이라는 기존의 SQL 과 아주 흡사한 질의언어를 통해 분산 환경에서의 데이터 분석 및 적재를 가능하게 하였습니다. 그리고 MapReduce 의 가장 큰 단점인 레코드 수준에서의 트랜잭션 및 조회 성능이 느리다는 점인데, 이러한 부분을 개선하기 위해 고안된 엔진이 HBase 이며, 기존 MapReduce 기반의 조회의 성능을 극대화 시키기 위해 고안된 엔진이 실시간 SQL 엔진인 Impala 입니다. 무엇보다도 기존의 BI 개발자와 엔지니어에게 친숙한 SQL 을 통해 보다 빠른 질의와 분석을 제공할 수 있다는 점이 큰 장점이라고 말할 수 있습니다. 데이터 처리량 면에서는 Parquet 기반의 MapReduce 작업이 뛰어나고 랜덤 액세스 면에서는 HBase 가 월등하지만 그 사이의 랜덤 액세스와 처리량을 모두 담당할 수 있는 엔진이 Apache Kudu 입니다.

Data Service Engines

데이터 서비스 엔진 (Hive, Impala, HBase, Kudu, MongoDB, Kafka)

데이터 서비스 엔진 – Hive

Apache Hive

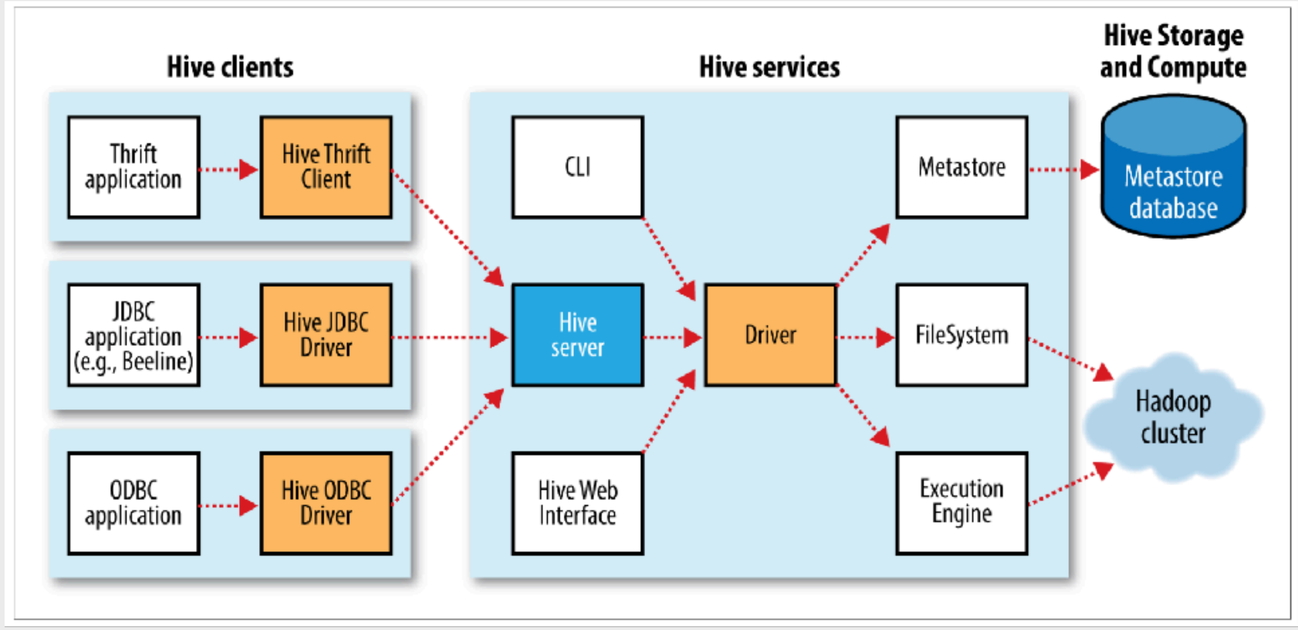
복잡한 MapReduce 코드 대신 SQL 만으로 대용량 데이터 분산처리가 가능하여, 기존의 DW 기반의 BI 개발자들도 손쉽게 사용할 수 있으며, SQL 에서 제공하는 다양한 함수 및 Optimizer 등이 거의 대부분 적용되어 확장성과 유지보수성이 뛰어나지만, 반복적인 처리와 업데이트가 자주 발생해야 하는 기계학습 혹은 그래프 분석 알고리즘에는 SQL 을 통한 데이터 파이프라인으로는 부족할 수 있습니다

Pros

1. SQL on Hadoop 기능을 구현한 오픈소스 데이터 웨어하우스
2. SQL 엔진의 특징인 쿼리 옵티마이저에 따른 최적화
3. 유저, 그룹 별 접근 권한 및 스키마에 에볼루션 지원

Cons

1. 실시간 쿼리나 레코드 단위 업데이트 등의 OLTP 기능에 부적합
2. MapReduce 기반의 처리가 기본이므로 쿼리 레이턴시는 느림
3. SQL 기반의 데이터 처리로 파이프라인 구성 혹은 모듈화가 어려움



데이터 서비스 엔진 – Impala

Apache Impala - "Real-Time Queries in Apache Hadoop, For Real"

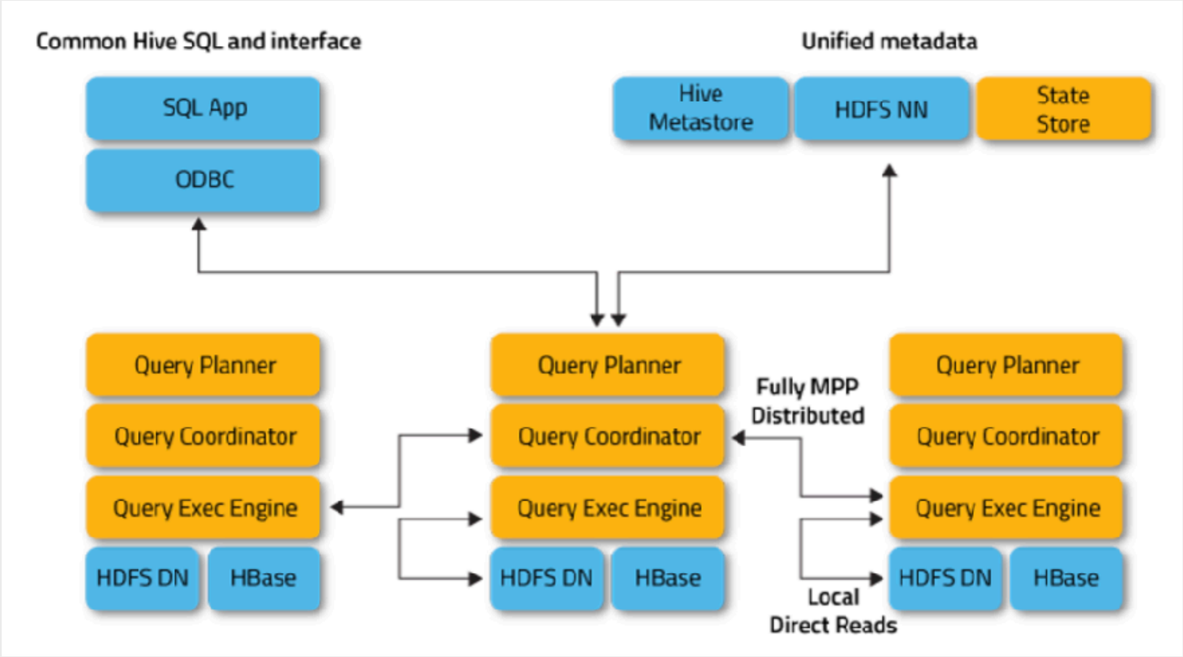
Hive SQL 을 인터페이스로 채택 하였으나 모든 Hive SQL 을 지원하는 것은 아니며, Hive 가 MapReduce 프레임워크를 통해 배치 방식의 데이터 조회와 변환을 수행한다고 하면 Imapla 는 응답시간을 최소화 하기 위해 고유의 분산 질의 엔진을 이용하는데 이 엔진은 클러스터 내의 모든 데이터 노드에 설치되어 있으며 로컬 스토리지의 Locality 와 메모리에 캐시된 정보를 이용하여 성능을 높이는 데 집중합니다.

Pros

1. 순수한 I/O 질의는 3~4배에서 캐시된 경우라면 20배 이상 좋은 성능
2. 메타데이터 자동 갱신 등을 통한 수평확장이 용이
3. 모든 노드가 브로커 역할을 수행하며 Failover 및 서비스 강건함

Cons

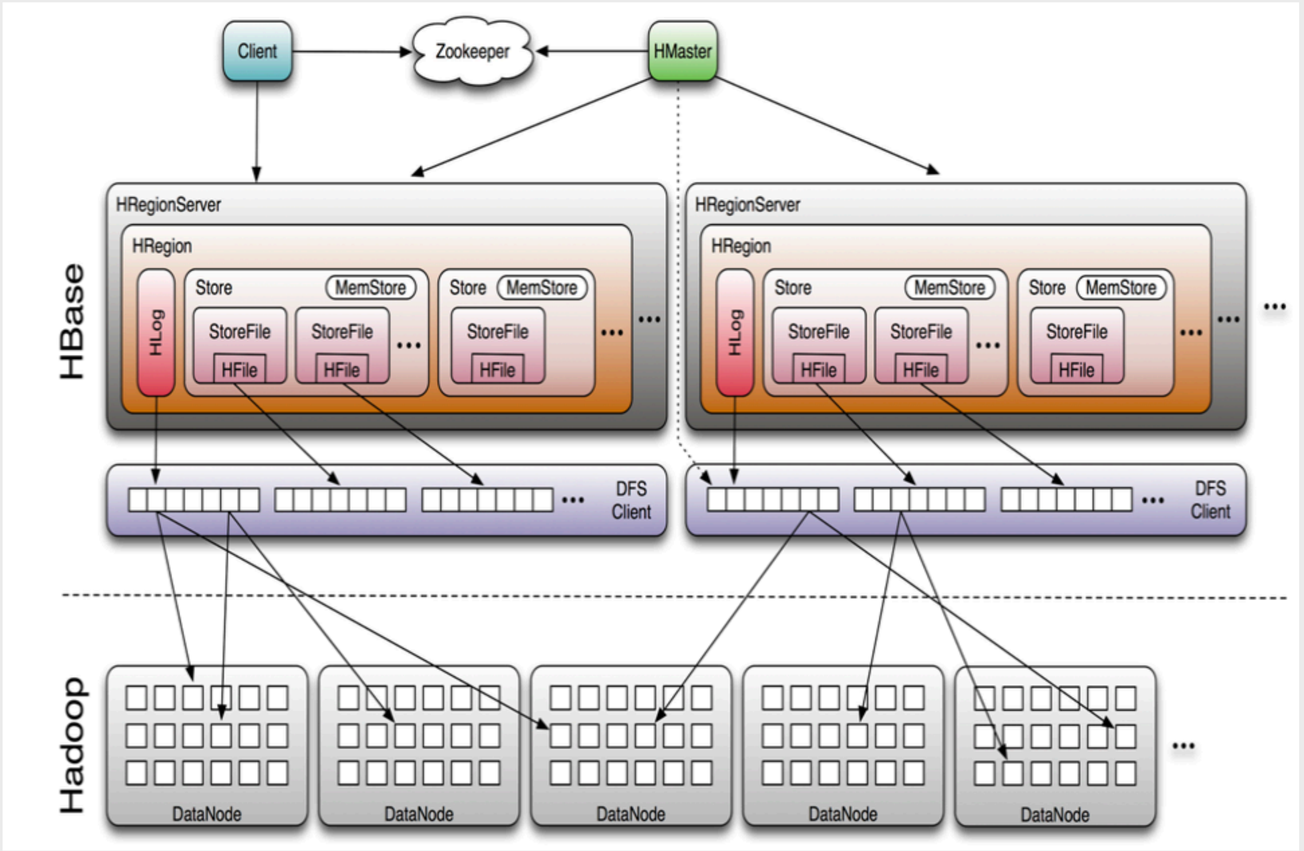
1. 캐시를 제대로 활용하려면 모든 노드에 설치 및 운영이 필요함
2. 메타 동기화 및 테이블 리프래시 등의 운영이 의외로 까다로움
3. 모든 하이브 SQL 을 지원하지 않으며, UPDATE, DELETE 지원 불가



http://impala.apache.org/overview.html

Apache HBase - hadoop based distributed scalable big data store

대량의 데이터를 분산 환경에서 관리 하면서도 데이터의 일관성을 보장해주는 분산 저장소입니다. 컬럼 기반 NoSQL 특성에 따라 효율적인 압축 저장 구조를 가지면, 저장 및 조회 성능이 뛰어납니다. 버전 관리 및 레코드 단위의 트랜잭션이 지원됩니다. 모든 데이터는 로우 키를 기준으로 리전(region)이라는 파티션으로 구분되어 중복없이 나뉘어 저장 및 조회



Pros

1. 분산 환경에서 일관성을 보장하는 컬럼 기반 저장소
2. 상대적으로 작은 범위의 분산된 읽기/쓰기 일관성 및 성능 보장

Cons

1. 대용량 분석작업을 위한 스캔작업에는 성능이 좋지 않음
2. 특정 리전/서버에 집중되는 경우 성능이 떨어질 수 있음
3. Minor/Major Compaction I/O Storm

https://blog.cloudera.com/introducing-apache-hbase-medium-object-storage-mob-compaction-partition-policies/

데이터 서비스 엔진 – Kudu

Apache Kudu - Hadoop's storage layer to enable **fast analytics on fast data.**

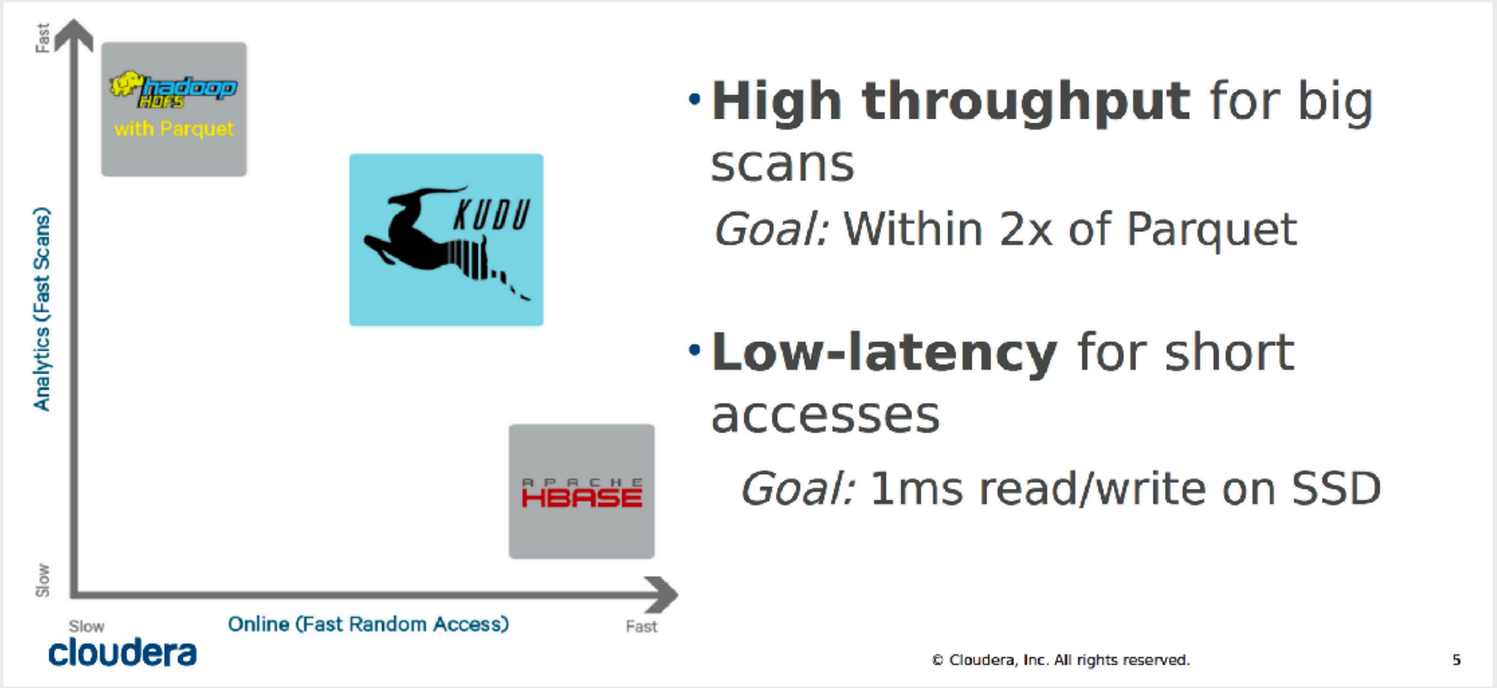
Parquet 와 같은 파일을 경우 순차 읽기에는 강하나 랜덤 액세스에는 약하며, HBase 의 경우 랜덤 액세스는 강하지만 반대로 순차 읽기에는 적합하지 않습니다. 이러한 간극을 매울 수 있는 High-throughput Low-latency 스토리지 엔진이 Kudu 엔진 이며, 다른 엔진과 다른 제약 조건이 존재하지만 CRUD 지원, 파케이 수준의 처리량과 밀리초 수준의 레이턴시 를 보장하는 컬럼 기반 데이터 스토리지입니다.

Pros

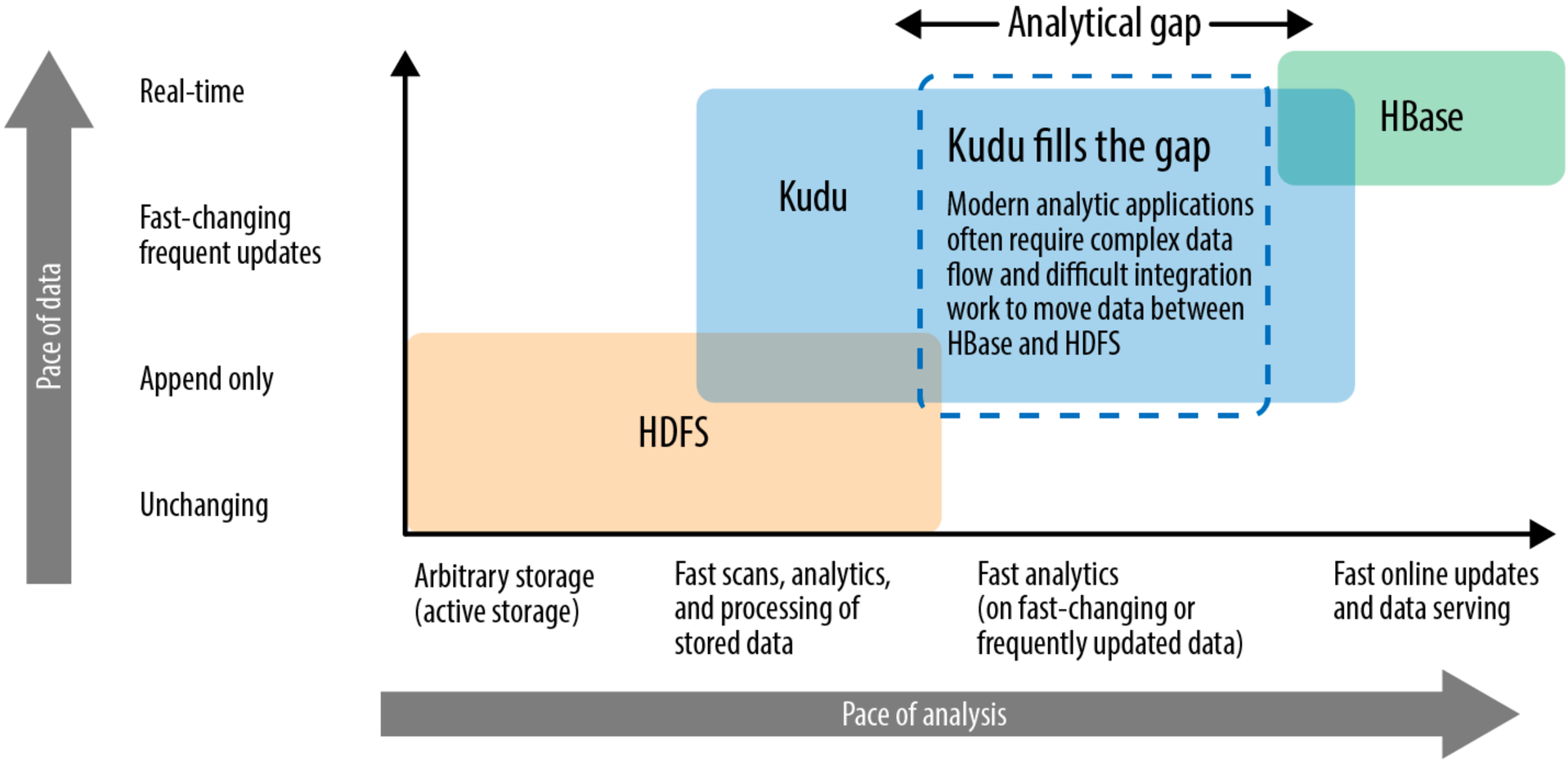
1. 밀리초 수준의 빠른 조회 속도와 CRUD 지원
2. 대규모 순차 읽기에도 최적화
3. 컬럼 기반 저장소 장점 필드 별 인코딩 및 압축지원

Cons

1. 세컨더리 인덱스 부재로 통계분석에 부적합
2. 저장소 엔진으로 조인, 집계 등의 SQL 엔진 부재
3. Aggregate Pushdown 등의 최적화 부재



Apache HDSF, Kudu and HBase



데이터 서비스 엔진 – MongoDB

MongoDB - general purpose, document-based, distributed database

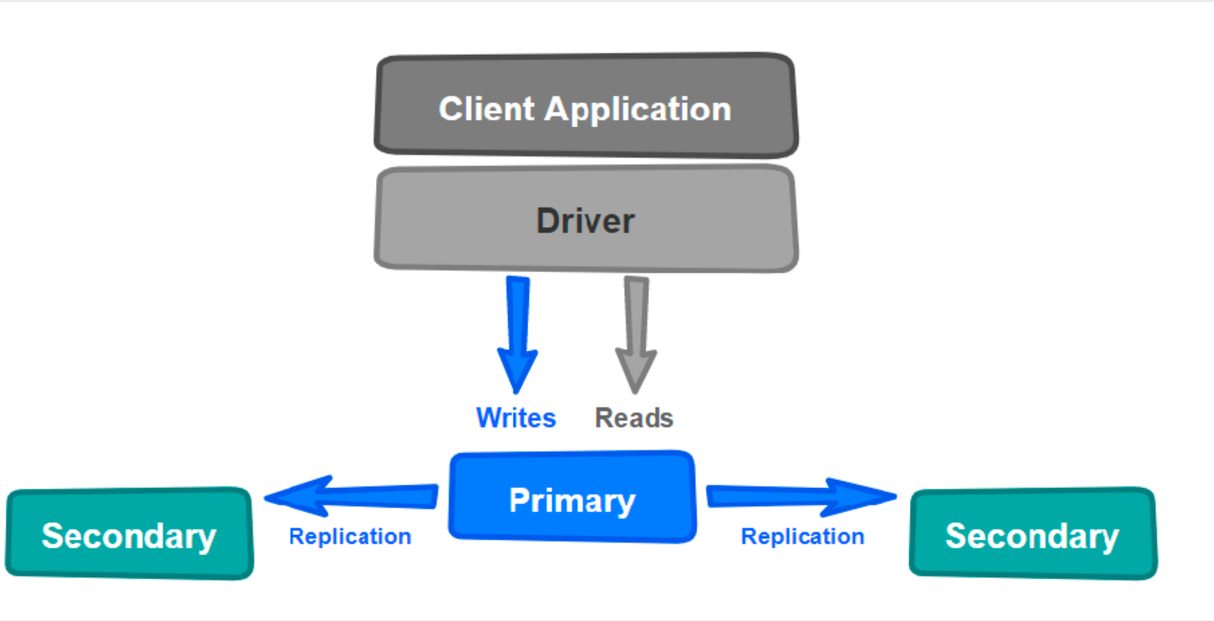
대규모 고가용성 메모리 기반 문서 저장소로써, 동적 스키마, 손쉬운 확장 및 관리 편의성을 가진 엔진입니다. 데이터를 BSON(Binary JSON) 형태의 문서로 저장하며 데이터베이스와 콜렉션(테이블) 단위로 관리됩니다. 레코드 단위 트랜잭션 및 필드 업데이트, 세컨더리 인덱스 지원과, 정규식 및 텍스트 검색까지 지원합니다. 다만, 관계형 데이터베이스와 다르게 조인을 지원하지 않으므로 정규화를 통한 콜렉션 관리가 어려워 기존의 관계형 데이터베이스와 모델링과 다릅니다

Pros

1. 필드, 복합키, 위치기반, 텍스트 등 다양한 인덱스를 제공
2. 메모리 기반의 저장 및 조회로 처리량 및 레이턴시가 아주 뛰어남
3. 리플리케이션, 고가용성 및 스케일링이 유연하며 쉽게 가능하다
4. 다양한 질의가 가능하다 (필터, 집계, 정규식 및 맵리듀스)

Cons

1. 메모리에 의존적이라 데이터가 커지면 성능 저하가 커짐
2. 조인을 지원하지 않아 슈퍼셋 콜렉션 혹은 별도 구현이 필요하다
3. SQL 과 유사하지만 다르기 때문에 최적화에 어려움이 있다



Apache Kafka - distributed event streaming platform

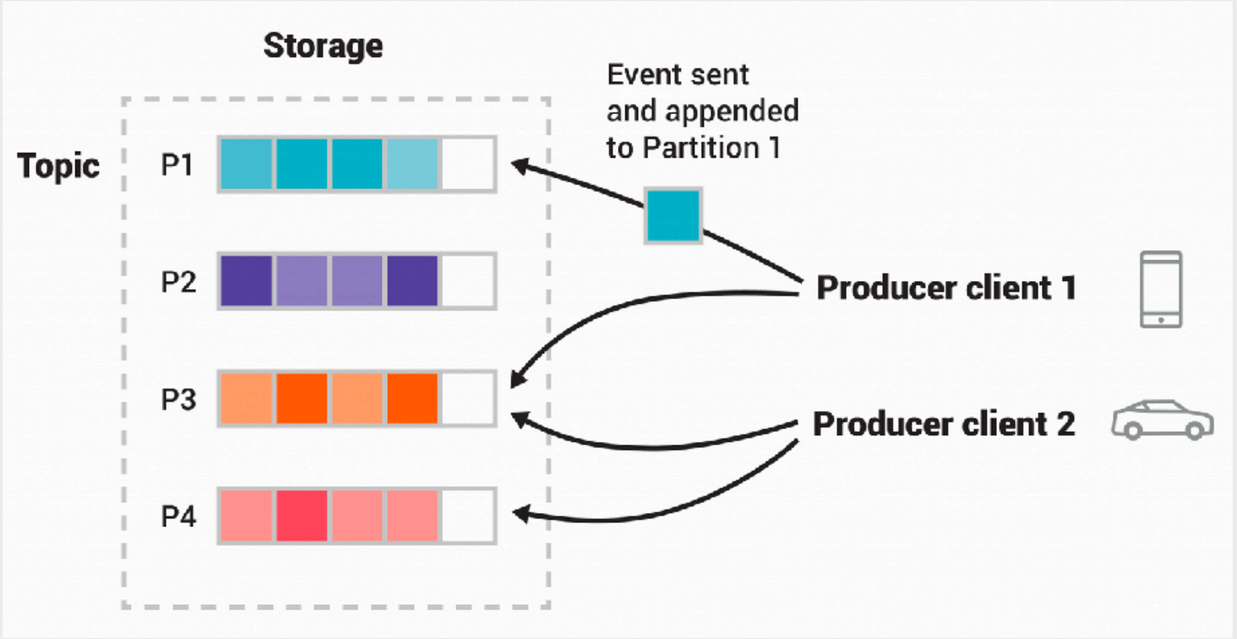
분산 환경에서, 고성능 / 고가용성 / 확장성을 보장하는 메시지 큐 서비스이며, 디스크 기반의 비동기 Publish-Subscribe 모델의 메시지 전송 방식을 가지며, 메시지를 생성하는 프로듀서가 브로커의 토픽으로 메시지를 전달하고, 이렇게 저장된 메시지를 컨슈머가 전달받는 방식으로 동작합니다. 여기서 토픽은 메시지의 논리적으로 묶은 단위이며 데이터베이스의 테이블과 유사합니다. 파티션은 실제 데이터가 저장되는 위치라고 볼 수 있고, 병렬성을 보장할 수 있는 장치이며, 프로듀서 및 컨슈머의 수 보다 파티션의 수가 많거나 같아야 리소스 유실이 없습니다.

Pros

1. 디스크에 저장되어 리텐션 기간 동안 영속성 보장
2. 멀티 프로듀서, 컨슈머를 통한 고성능, 확장성 보장
3. 일부 브로커가 죽더라도 다른 노드 지속하여 고가용성 보장

Cons

1. pull 방식으로 정기적인 polling 에 의한 자원 사용
2. 기본 제공 운영도구가 없어 별도로 구성이 필요

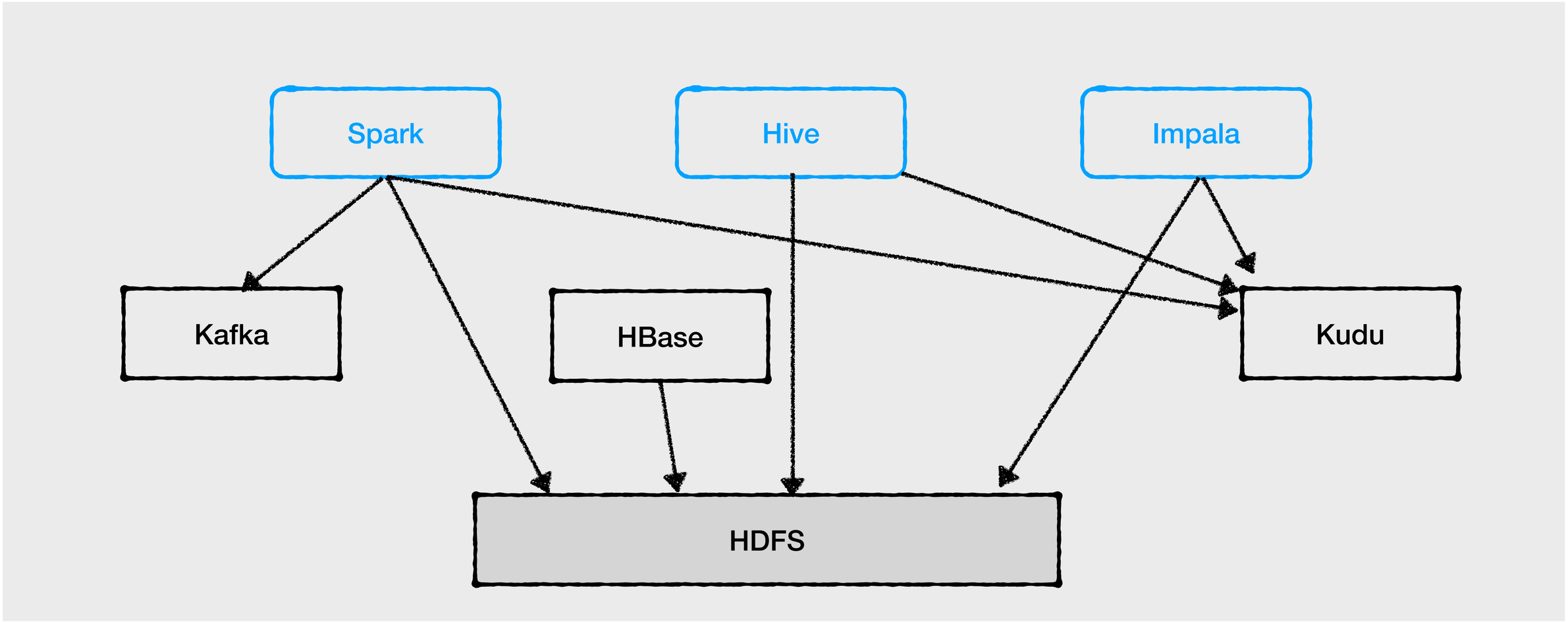


https://www.confluent.io/blog/kafka-streams-tables-part-3-event-processing-fundamentals/

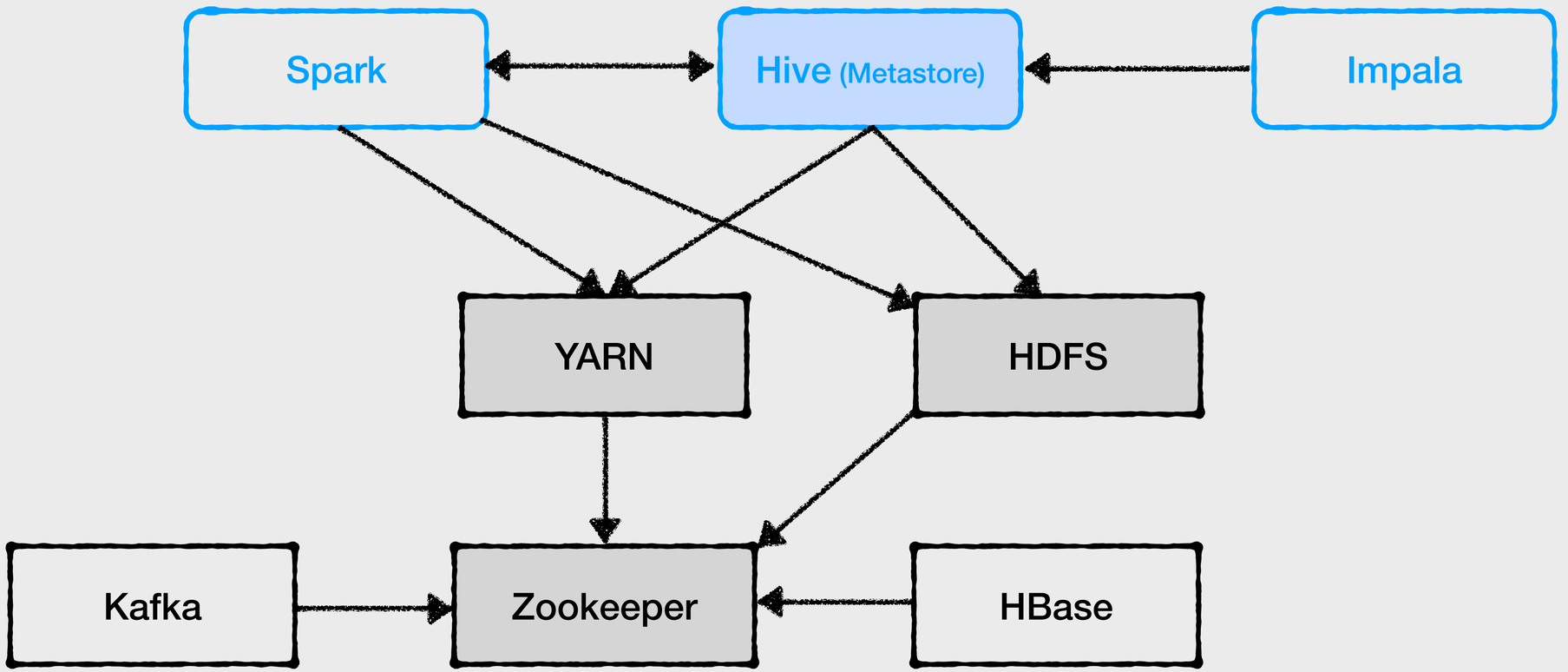
Relationship

데이터의 흐름 및 의존성에 의한 관계

Data flow between data services



Dependencies between Infra softwares



수고 하셨습니다