데이터 엔지니어링 파일 수집

파일 데이터 수집 개요

Park Suhyuk

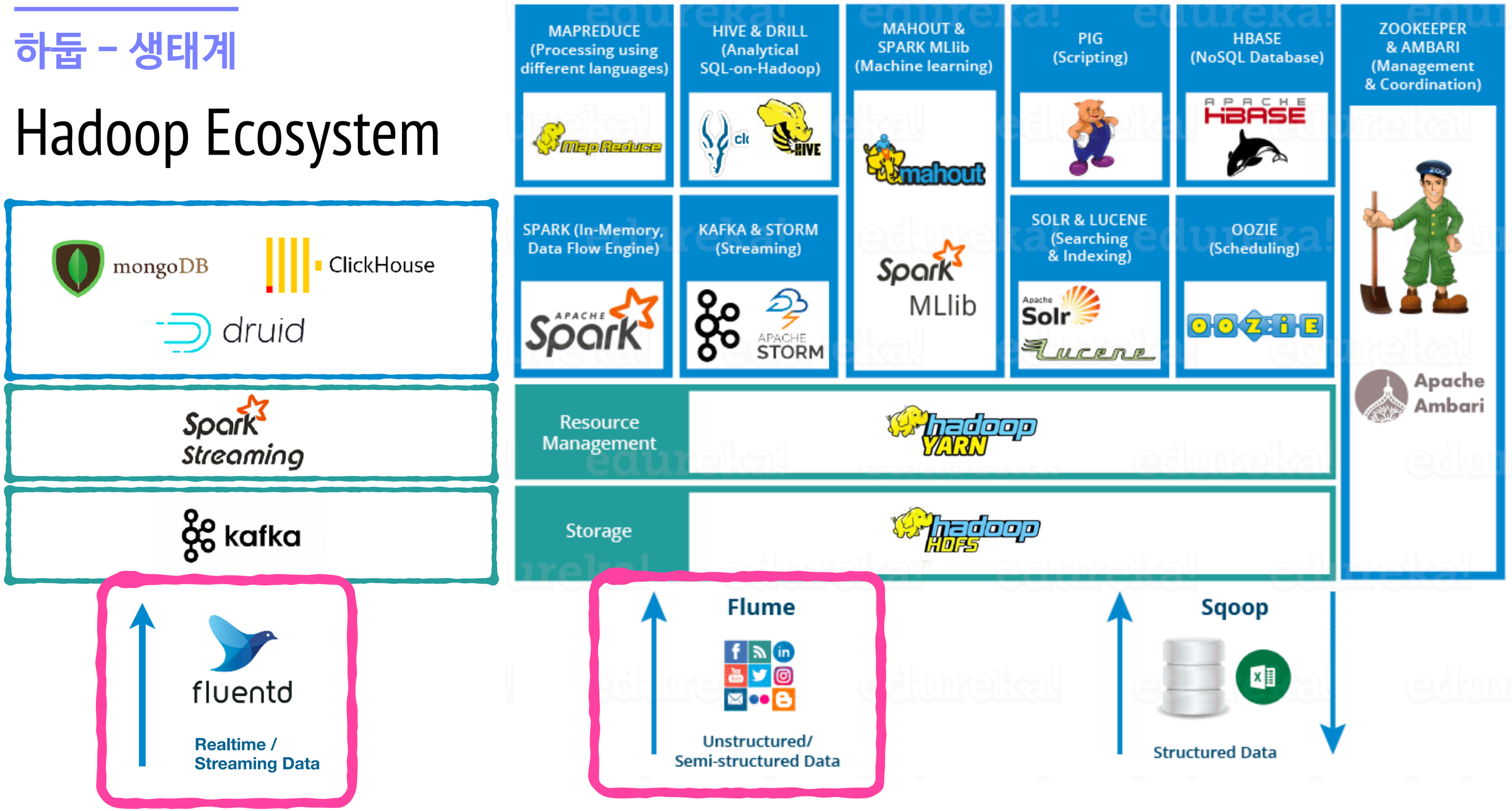
 psyoblade

 psyoblade

ubuntu 18.04 LTS

하둡 – 생태계

Hadoop Ecosystem



목차

파일 데이터 수집

파일 데이터 수집 개요

What is 'File Collection'?

관계형 데이터베이스 테이블의 수집이 정형 데이터인 반면, 파일의 경우 반 정형 혹은 비정형 데이터에 가깝습니다. 데이터베이스에 비해 프로토콜 (FTP, SSH, SAMBA, HTTP 등) 및 포맷 또한 다양합니다. 특히 테이블 스냅샷 수집의 경우 배치 처리에도 편하고, 직관적이며 관련 오픈소스가 제한적이라 선택의 여지가 없어 오히려 용이하지만, 파일 수집의 경우 파일 스트림 데이터를 끊임없이 동기화 하거나 적재해야 하므로 운영 및 유지 관리에 더 많은 어려움이 있으며, 다양한 접근 방법이 있으며, 일반적으로 운영 및 장애 처리에 더 까다롭고 많은 시간이 소요됩니다.

1. 데이터의 성격에 따라 해당 **이벤트가 발생한 시간** 혹은 **데이터를 수신한 시간** 기준으로 처리합니다
2. 데이터의 특성에 따라 특정 시간 이상으로 **지연된 데이터**는 지표의 **멱등성**을 해칠 수 있기 때문에 별도로 처리하거나 버리기도 합니다
3. **Message Delivery Semantics** 의 수준에 따라 아키텍처는 **성능과 리소스의 트레이드 오프**를 고려한 설계가 필요합니다
   - 확장성 및 유연성도 크게 차이가 날 수 있습니다. 한 건의 이벤트 메시지도 유실되지 않도록 처리하기 위해서는 운영, 유지보수에 더 많은 리소스가 소요
4. 과거 데이터 아키텍처는 **Redundant** (이중화, 고가용성, 클러스터링 및 트랜잭션 유지) 한 방향으로 충분히 적은 데이터를 **안정적인 운영**인 반면
5. 최근의 경우는 **Resilience** (빠른 장애 인지 및 복구, 지연된 트랜잭션 등) 한 방향으로 분산 환경에서의 **고 가용성과 데이터 처리 수준**에 초점을 둡니다
   - 분산 환경에서의 대용량 데이터 처리를 효과적으로 운영해야 하므로, 장애가 나는 상황을 예외적인 상황이 아니라 일반적인 상황으로 보아야 했고, 특히 고 가용성에 따른 비용을 낮추는 전략이라고 볼 수 있습니다

Throttling 조절 - 네트워크 혹은 데이터 상황에 맞추어 전송 등을 조정하는 것

파일 데이터 수집 개요 – 시간

Event time vs. Processing time

데이터 처리 시스템 관점에서 시간과 관련된 용어 가운데 가장 중요한 2가지 개념이 있습니다. "이벤트 타임"은 실제 이벤트가 발생한 바로 그 시간을 말하며, "처리 시간"은 이벤트 데이터가 시스템에 수신 혹은 관찰된 시점을 말합니다.

가장 이상적인 상황은 이 두 시간이 동일한 경우지만, 대부분의 경우 그렇지 못 합니다. 특히 모바일 장치의 경우 여러가지 이유로 잘못된 시스템 시간을 가지고 있을 수 있으며, 실제 모바일 장치가 아닌 에뮬레이터일 수도 있습니다. 또한 정상적인 시간을 가지고 있더라도 지구 반대편에 있을 수도 있는 시스템에 도착하기 까지 지연될 수 있으며, 장치, 소프트웨어 혹은 네트워크의 영향으로 전송하지 못 한 이벤트를 재시도 하는 경우, 심지어 다음날 장치가 기동 되는 시점에 전송되는 경우까지 고려한다면 일치하는 경우가 오히려 드물다고 할 수 있습니다.

1. 공유 자원의 한계 : 네트워크 토폴로지 구성, 네트워크 혼잡 혹은 공유 자원 (CPU 등)의 한계
2. 소프트웨어 문제점 : 분신 시스템 처리 혹은 경쟁 상황
3. 데이터 자체의 문제점 : 비행기 모드, 의도적인 변경, 에뮬레이터 등의 복제된 시스템 이미지에 의한 데이터 가공

https://www.oreilly.com/radar/the-world-beyond-batch-streaming-101/

파일 데이터 수집 개요 – 시간

Data Processing & Idempotent

'멱등성' 이란 용어는 전산학 이나 수학에서 사용하는 용어인데, 동일한 연산을 여러 번 적용하더라도 결과가 달라 지지 않는 성질을 의미합니다. 즉, f(f(x)) = f(x) 과 같이 동일한 함수를 여러번 적용하더라도 같은 등식이 성립하는 경우를 말합니다. 데이터 처리 관점에서 보았을 때에 동일한 입력에 대해서는 임의의 애플리케이션을 여러번 수행 하더라도 동일한 지표와 리포트를 만들어내는 특성을 말합니다.

데이터 엔지니어링 관점에서 보았을 때에 데이터 처리의 멱등성은 아무리 강조해도 지나치지 않습니다. 이는 분산환경에서 발생할 수 있는 다양한 문제점들 (장비, 네트워크 혹은 소프트웨어 장애 등)에 의해 언제든지 작업은 실패할 수 있으며, 다시 처리해야 할 수도 있습니다. 하지만 매번 처리 시마다 지표가 변화하거나 달라진다면 지표 자체를 신뢰할 수 없기 때문에 이러한 멱등성을 감안한 데이터 애플리케이션의 설계는 아주 중요합니다.

이러한 멱등성을 보장하기 위해서는 데이터 소스의 특성을 잘 이해해야 하고, 이러한 특성(지연, 유실, 중복 등)을 고려한 정책(SLA, Service Level Agreement)을 마련해야 하며, 서비스의 성격에 따라 메시지 시맨틱(Message Semantics)을 선택합니다

파일 데이터 수집 개요 – 시간

Message Delivery Semantics

메시지 레벨의 보증 전략은 크게 3가지로 구분 됩니다.

at-most-once

: 흔히 fire-and-forget 이라고 부르기도 하는데, 전송에 실패한 경우는 재시도 혹은 재전송을 하지 않고 유실될 수도 있음을 말하는데요, 짧은 간격으로 주기적으로 발생하는 센서 데이터와 같이 일부 유실되더라도 대세에 큰 영향이 없거나, 너무 많은 데이터가 발생하여 복잡한 처리가 어려운 경우에 선택할 수 있습니다.

exactly-once

: 정확하게 한 번의 메시지를 전송을 보장합니다. 유실, 중복을 허용하지 않기 때문에 데이터 소스, 처리 및 저장소 모두 이러한 기능을 지원해야 하고 당연히 구현이 어려우며, 성능에도 영향을 줍니다. 분산환경에서의 exactly-once 를 지원하기는 상당히 어려운데, 실패의 경우는 주기적으로 재시도를 하면 되겠지만 지연이 발생하거나 타임아웃이 발생하는 경우 상당히 처리하기 까다롭습니다

at-least-once

: 중복은 허용하되 유실은 허용하지 않습니다. 즉, 전송은 되었지만 Ack 를 받지 못해 재전송하는 것을 허용하는 것을 말하며, 분산환경에서는 이후 처리에서 중복은 제거할 수 있기 때문에 그나마 유실 없는 가장 적절한 타협점이 될 수 있습니다

https://www.joinc.co.kr/w/man/12/Kafka/exactlyonce

파일 데이터 수집 개요 - 시간

CAP & PACELC Theorem

"CAP 이론"은 분산 데이터베이스 시스템의 세 가지 속성인 일관선(Consistency), 가용성(Availability), 파티션 허용성(Partition tolerance)를 말합니다. 분산 데이터베이스 시스템은 네트워크 파티션이 발생했을 때에 세 가지 속성을 동시에 모두 만족시키는 시스템은 존재하기 어렵다는 것이며, 그래서 분산 데이터베이스 시스템은 세 가지 가운데 두 가지의 특성에 최적화되어 있다고 얘기할 수 있습니다. 이후에 CAP 이론은 "일관성과 가용성은 상충관계에 있으나 반드시 둘 중에 하나만 선택할 필요는 없다"고 볼 수 있으며, 파티션이 없는 상황에 대한 설명이 어려운데, 이후 "PACELC 이론"을 통해 설명할 수 있습니다.

속성	설명	부연 설명
일관성	모든 요청은 최신 데이터 혹은 오류를 응답합니다	시점의 차이로 다른 값을 반환하지 않는다는 의미 입니다
가용성	모든 요청은 정상적인 응답을 받습니다	일부 노드의 장애 시에도 시스템은 정상적인 처리가 가능하다는 의미입니다
파티션 허용성	노드 간의 통신이 실패하더라도 정상동작합니다	일부 노드간의 통신 상의 문제가 발생하더라도 시스템은 정상이라는 의미입니다

http://happinessoncode.com/2017/07/29/cap-theorem-and-pacelc-theorem/

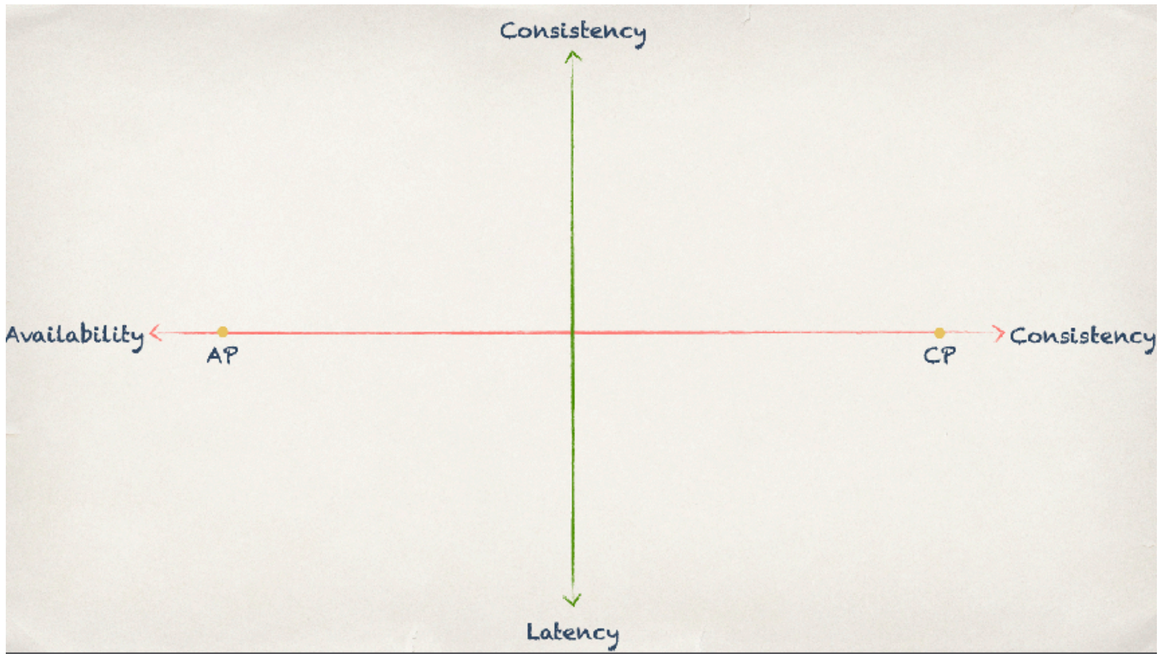
파일 데이터 수집 개요 - 시간

CAP & PACELC Theorem

CAP 의 단점을 보완하기 위해 나온 이론이며, CAP 이론이 네트워크 파티션 상황에서 일관성-가용성 측면을 이용하여 설명하려고 했다면, PACELC 의 경우는 정상적인 상황이라는 축을 하나 더 넣어 1~4 사분면에 해당 시스템 혹은 서비스를 설명할 수 있습니다.

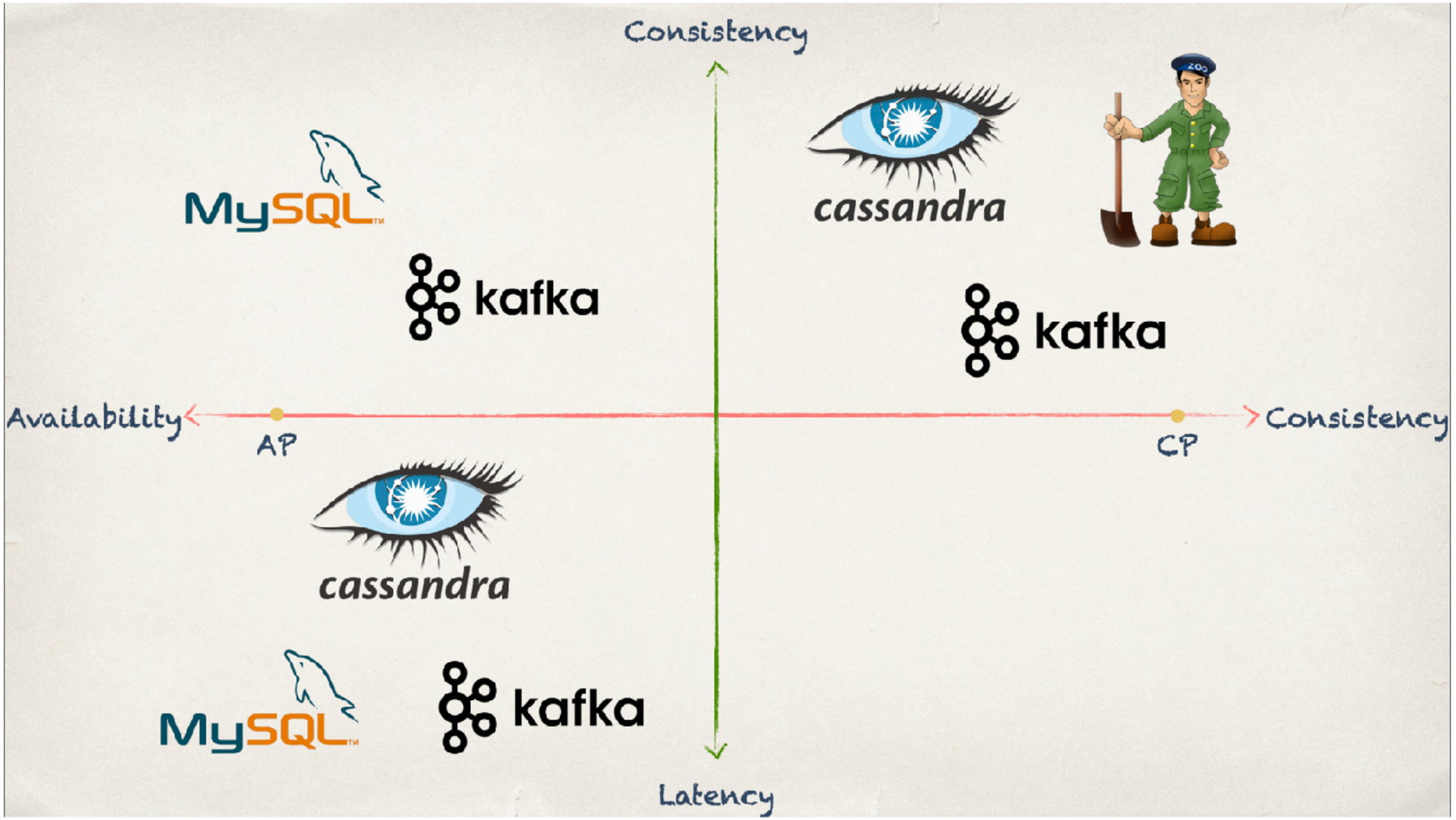
X 축은 네트워크 파티션 상황에서 일관성과 가용성 측면을 설명할 수 있습니다. 예를 들어 Master-Slave 로 구성된 MySQL 서버는 기본적으로 PA/EL 이지만 설정에 따라서 PA/EC 가 될 수도 있습니다. (여기서 지연시간의 Y축은 위로 갈수록 지연시간이 짧다는 의미입니다, slave 혹은 cluster 구성에서의 latency)

Y 축은 레이턴시와 일관성 즉, 정상적인 조회가 가능한지를 설명합니다.



http://happinessoncode.com/2017/07/29/cap-theorem-and-pacelc-theorem/

파일 데이터 수집 개요 – 시간



오픈소스 데이터 수집 도구

오픈소스 데이터 수집 도구

Scribe, Suro, Chukwa, Flume, Fluentd

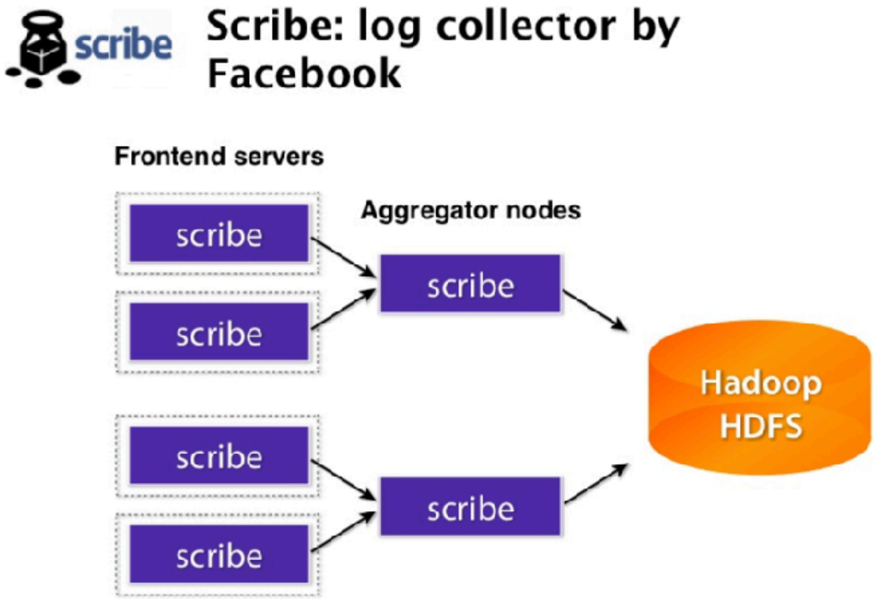
다양한 오픈소스 데이터 수집 및 처리 도구의 특성을 살펴보고 운영 환경이나 개발언어 및 운영 중인 인프라를 고려하여 선택하는 것이 바람직하나, 과거 폴링 기반의 중앙집중식 수집환경이 주를 이루었으나, 최근에는 컨테이너 기반의 가상 환경 아키텍처가 주를 이루고 있기 때문에 푸시 기반의 에이전트를 통한 수집이 더 각광받고 있습니다.

오픈소스 데이터 수집 도구 - Scribe

Facebook Scribe

Facebook 에서 개발 운영하던 내부 엔진이며, 실시간 로그 수집 서버와 다수의 서버로부터 실시간 스트리밍 로그 데이터를 수집하기 위한 애플리케이션 입니다. Thrift 라이브러리를 통해 전송되며, C++ 로 구현되어 성능은 좋으나, 설치하기 상대적으로 어려워 추가 개발 및 확장에 어려움이 있습니다. 2014년 이후 추가 개선 사항은 없습니다.

현재 Fluentd 나 Logstash 등의 아키텍처 수준에서는 크게 다르지 않으며 다만 에이전트의 개발 혹은 플러그인의 접근성의 차이가 가장 큰 진입장벽으로 보입니다.



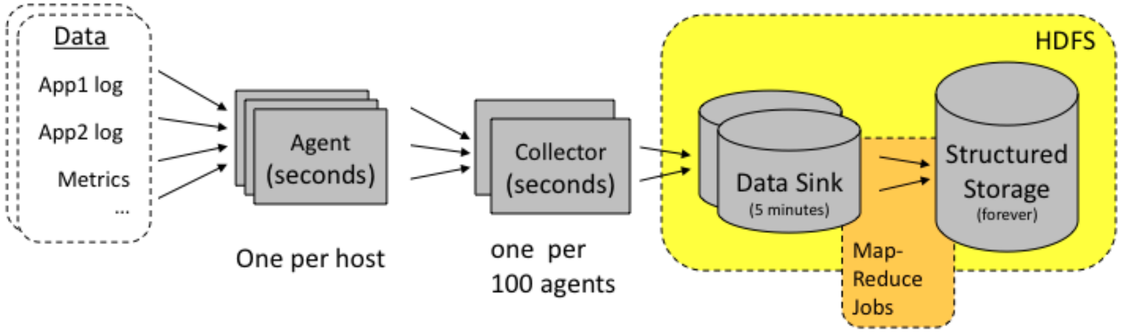
https://pt.slideshare.net/treasure-data/fluentd-loves-mongodb-at-mongosv-july172012/32-Scribe_log_collector_by_Facebook

오픈소스 데이터 수집 도구 - Chukwa

Apache Chukwa

분산 환경 서버 수집 에이전트, 시스템 로그, 응용 프로그램 및 하둡 로그를 수집 및 모니터링을 위한 도구로 고안되었습니다. 에이전트는 각 노드에 설치 되고, 컬렉터를 통해 수집된 로그를 하둡 분산 저장소에 시퀀스 파일 포맷으로 저장되어 MapReduce 작업을 통해서 후처리를 통해 K-V 으로 저장되어 Chukwa 레코드 형식으로 저장되어 웹 인터페이스를 통해 실시간 모니터링 기능을 제공합니다. 그리고 2016년 이후 커밋 없으며 관리되지 않습니다.

초기에 하둡 기반의 데이터를 전송하고 저장하는 구조가 다양한 데이터 수집에는 적합하지 않아 적용 대상에 제한이 있다는 점이 단점입니다



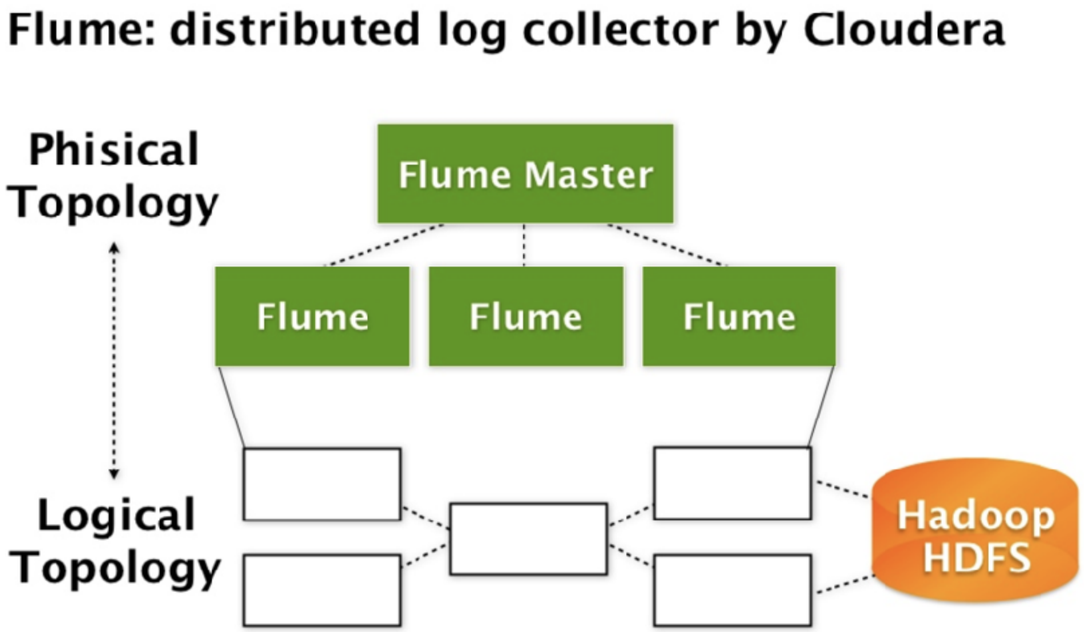
http://chukwa.apache.org/docs/r0.4.0/design.html

오픈소스 데이터 수집 도구 - Flume

Apache Flume

Apache Flume-NG 가 기존의 Flume 의 개선된 버전이며, 대용량 로그의 비동기 분산 수집이 가능한 채널 기반 파일 수집 도구. N개의 데이터 소스에서 N개의 채널 그리고 N개의 싱크로 유연한 데이터 처리 구조를 가지고 있으나, 신경써야 할 부분이 많고, 상세한 설정에 대한 이해가 수반되어야 하므로 개발, 에이전트 확장 및 운영 비용이 비교적 큰 편입니다.

플럼 마스터가 노드들을 관리하는 중앙집중식 구조이며, 에이전트 → 컬렉터 → 스토리지 티어로 전송하는 구조이며 각 에이전트간에 연동이나 확장성 면에서 유연하고, 다양한 Sink 에이전트를 통해 원 소스 멀티 유즈 데이터 파이프라인을 구성하는 데에 용이한 아키텍처를 가지고 있습니다. 다만 토폴로지나 물리서버와 설정 정보를 관리해야 하므로 다소 구성 및 유지가 복잡한 점이 단점입니다

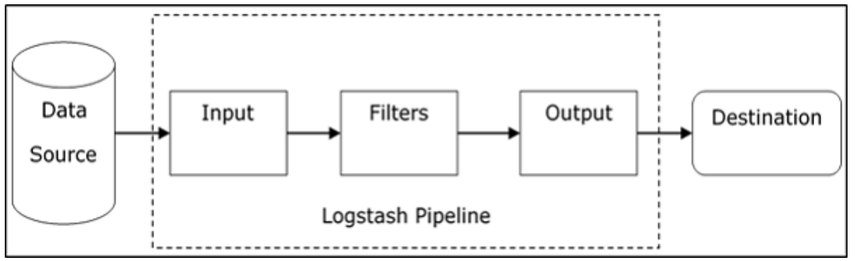


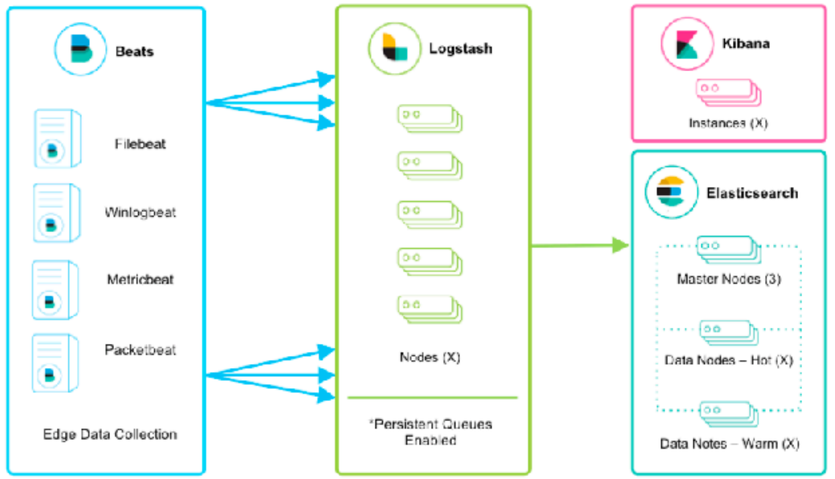
https://heodolf.tistory.com/35

Elastic Logstash

플러그인 개발및 배포가 Elastic 을 통해 관리되고 있어 기본적인 품질을 보증하고 있으며, 오랜 기간 개발된 많은 플러그인을 가지고 있습니다. 이벤트 라우팅이 프로그래밍 적으로 구성하기 때문에 다소 복잡하지만 세밀한 구성이 가능합니다.

전송 계층에 있어서 신뢰성을 높이기 위해 반드시 레디스와 같은 외부 큐 연동이 필수적이라는 점이 단점일 수 있습니다.





https://www.elastic.co/guide/en/logstash/current/deploying-and-scaling.html

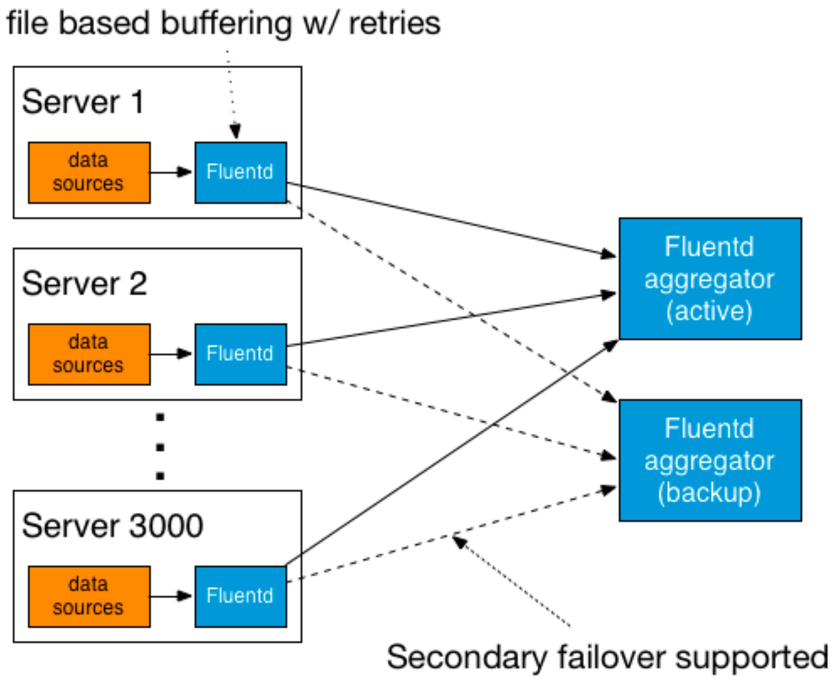
오픈소스 데이터 수집 도구 - Fluentd

Treasure Data Fluentd

fluentd 에이전트가 plugins 설치로 수집, 변환, 적재 및 라우팅이 가능하며, 별도의 fluent 로 전송하고 다시 저장소를 분산하게 할 수도 있으며, 좀 더 손쉬운 구성이나 운영이 가능합니다. 또한 다양한 플러그인이 제공되고 있어 최근 각광받고 있는 수집 도구로 알려져 있습니다. ruby 언어에 대한 부담이 없다면 상당히 많은 플러그인이 제공되고 있으며 쉽게 구현이 가능합니다

총 7개의 컴포넌트(Input, Parser, Engine, Filter, Buffer, Output, Formatter)로 구성되어 있으며 설정 파일을 통해 이러한 구성과 조합이 가능합니다

데이터의 흐름은 Input → Engine → Output 이고 필요에 따라 Parser, Buffer, Filter, Formatter 등을 활용할 수 있습니다



https://www.fluentd.org/blog/tag/architecture

오픈소스 데이터 수집 도구

Fluentd vs. Logstash vs. Flume vs. Scribe

최근에는 Fluentd 와 Logstash 가 가장 널리 사용되며 무엇보다도 ELK 혹은 EFK 스택을 통해 엔터프라이즈 환경에서 로그 수집에서 부터 시각화에 까지 손쉽게 적용할 수 있기 때문입니다. 아키텍처 구조나 플러그인 등 여러모로 유사한 구성을 가지고 있으나, Logstash 가 다소 무겁다는 성능 평가를 보이고 있으며 최근에는 양 제품 모두 Elasitc Beat, Fluent Bit 과 같은 경량화 에이전트가 등장하고 있어 서비스 환경이나 요구사항 그리고 데이터 엔지니어에 좀 더 친숙한 엔진을 선택하는 것도 좋을 듯 하며, Fluentd 의 경우 Docker 및 클라우드 환경에서 적용이 용이하여 최근 많이 활용되고 있으며 아키텍처 구성 시에 active-active 혹은 active-passive 모두 구성이 가능한 점도 장점으로 보입니다

속성	Fluentd	Logstash	Flume	Scribe
설치	gem/rpm/deb	tar/rpm/deb	jar/rpm/deb	make
코드	ruby 3000 lines	java	java - 50,000 lines	C++ 8,000 lines
플러그인	ruby	ruby	java	n/a
플러그인배포	rubygems.org	rubygems.org	n/a	n/a
마스터구조	n/a	n/a	yes	n/a

데이터 아키텍처 예제

파일 데이터 수집 아키텍처

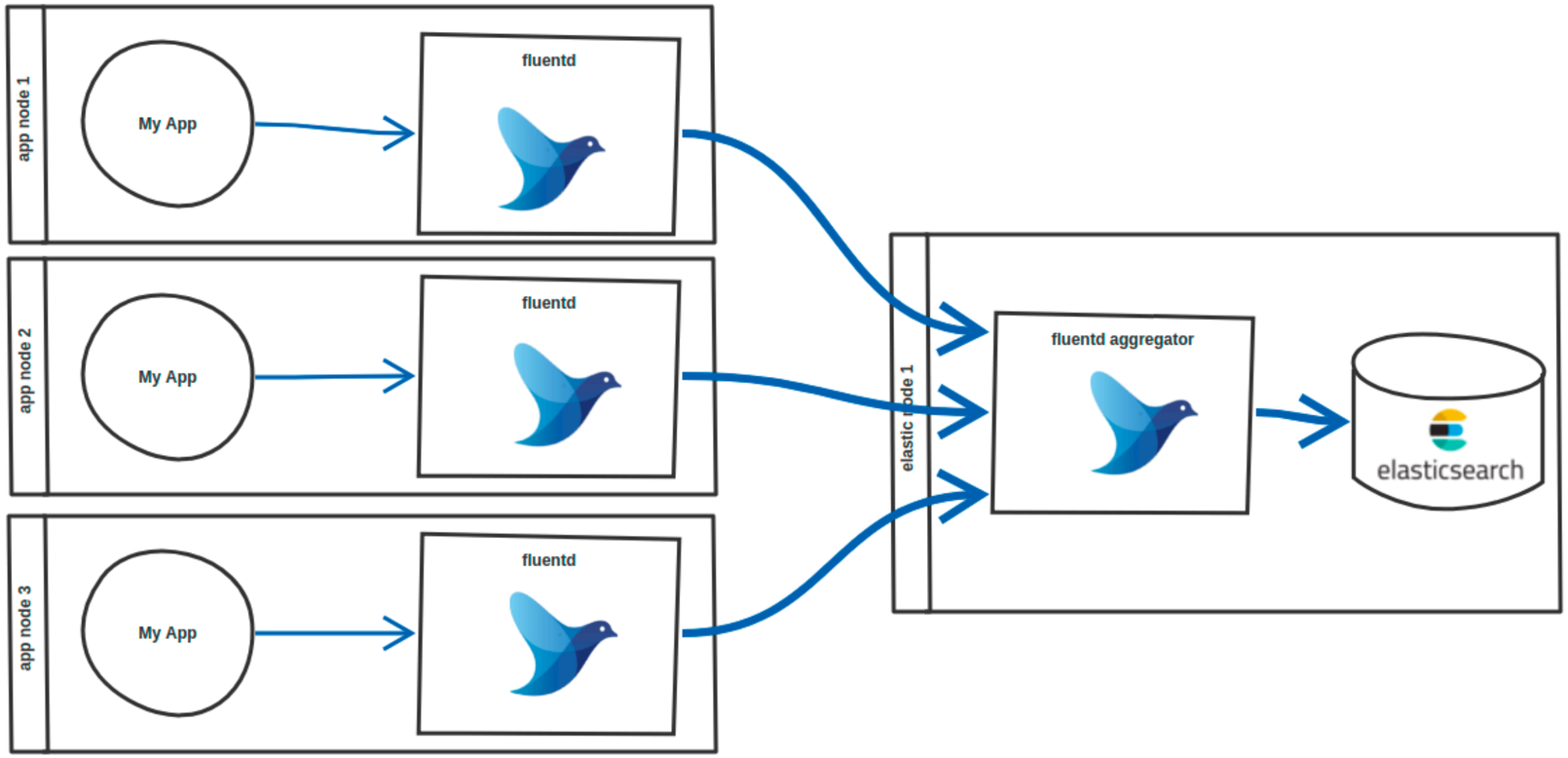
File Collector Architecture Design

최근에 가장 각광받고 있는 Fluentd 와 Logstash 가운데 선택하는 것이 적절하다 판단 되며, 본 강의에서도 대부분 Docker 기반의 데이터 엔지니어링에 초점을 맞추고 있으며 향후 고급 과정에서도 클라우드 환경에서 수행될 것을 감안하여 Fluentd 의 아키텍처를 기준으로 설명 드립니다.
추가적인 플러그인이 필요 없는 순수한 fluentd 에이전트만을 이용하여 적재하는 설계 부터 스트리밍 및 유실없는 데이터 전송 아키텍처 까지 간략히 소개해 드릴 예정입니다.

1. fluentd agents -> fluentd aggregator -> elastic-search
2. docker container -> fluentd aggregator -> elasitc-search -> kibana
3. fluentd log forwarders -> fluentd log aggregators
4. fluentd producer -> kafka -> fluetnd producer -> hdfs

데이터 아키텍처 예제 #1

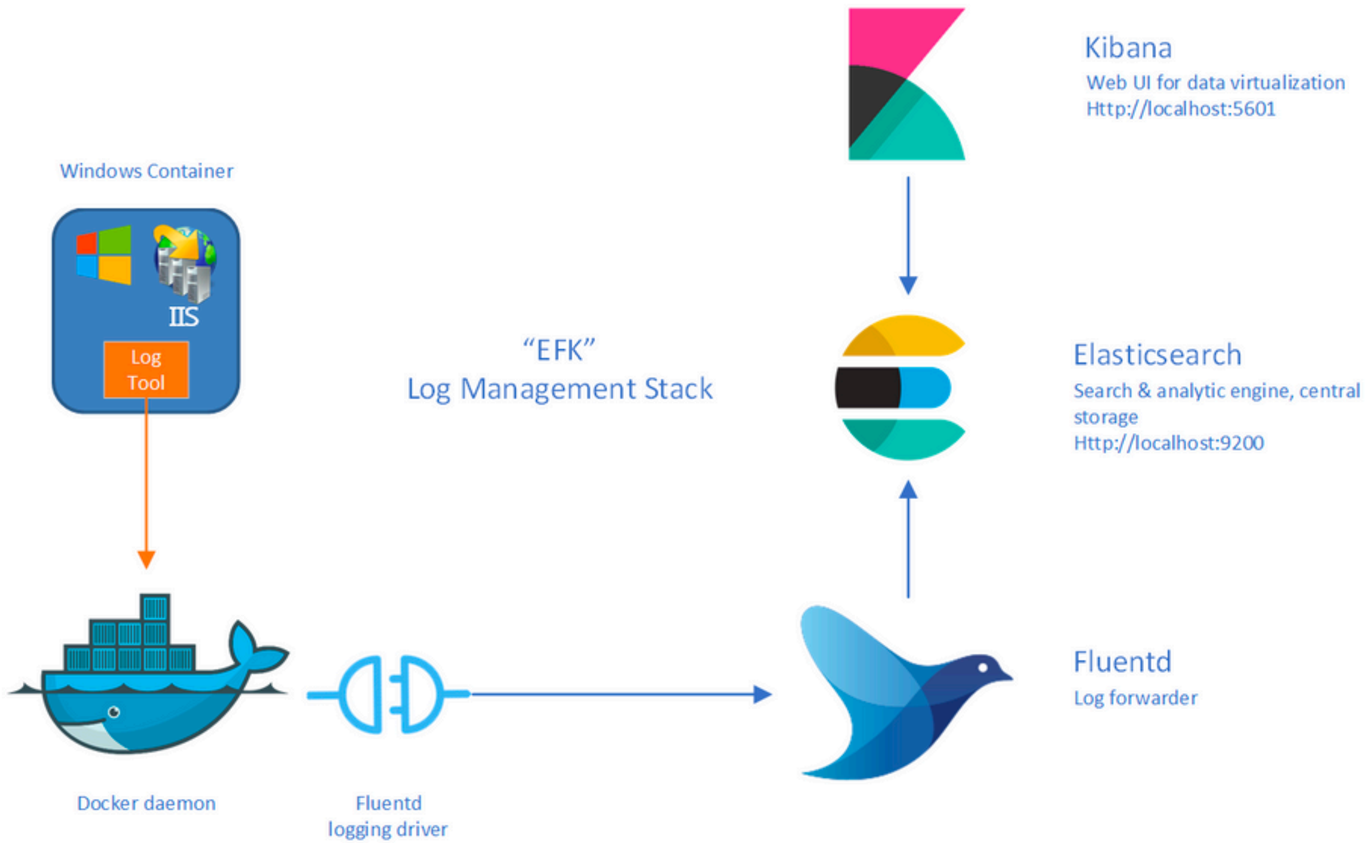
Forwarder -> Aggregator -> ElasticSearch



https://medium.com/@federicogaule/collecting-access-logs-into-elasticsearch-1a6f05288f8a

데이터 아키텍처 예제 #2

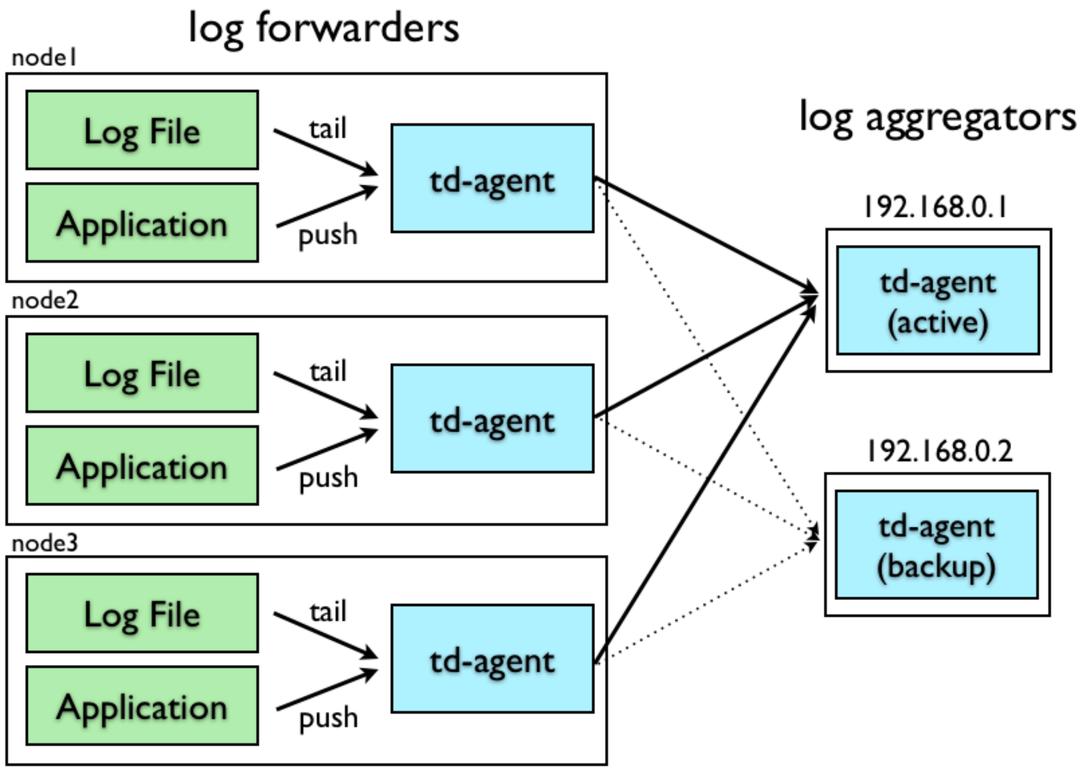
Docker -> Forwarder -> ElasticSearch -> Kibana

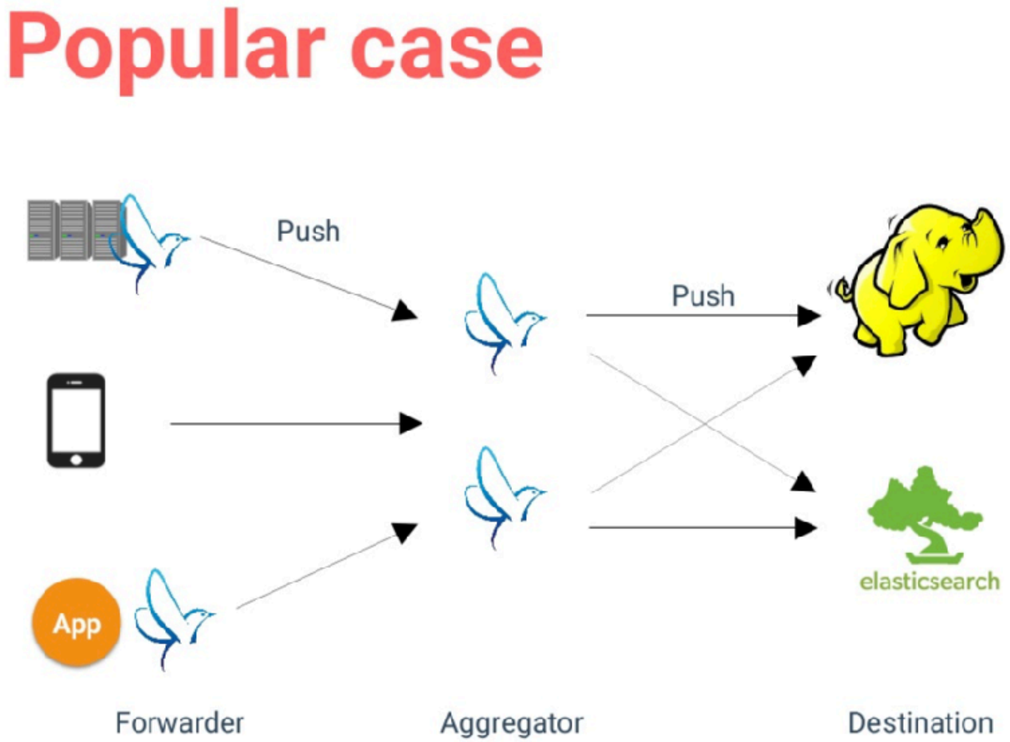


https://techcommunity.microsoft.com/t5/containers/announcing-container-log-tooling-and-survey/ba-p/536088

데이터 아키텍처 예제 #3

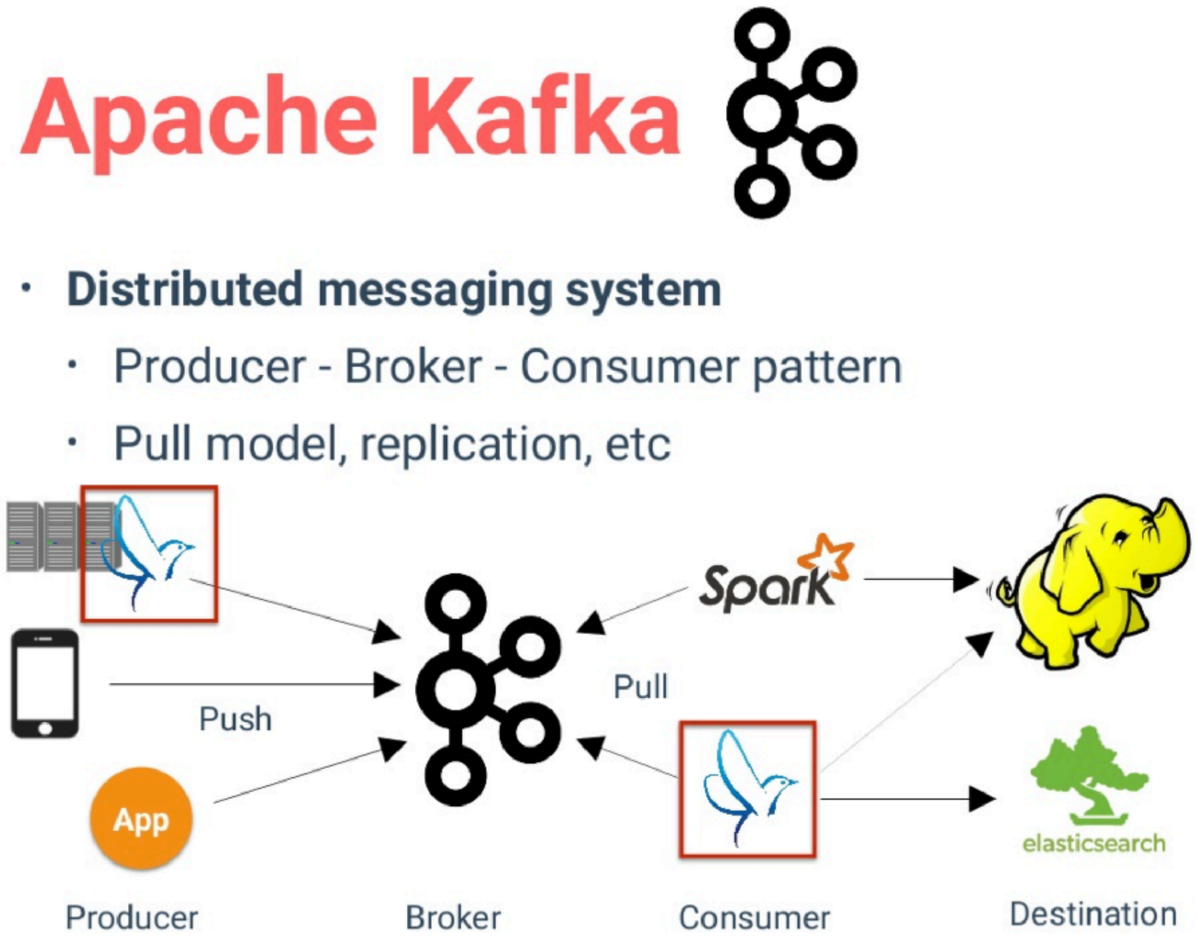
Forwards -> Aggregators -> Distributed Storage





데이터 아키텍처 예제 #4

Producer -> Kafka -> Consumer



데이터 모델링 예제

데이터 모델링 예제

File Modeling Design

파일 데이터의 경우는 기본적으로 Unstructured Data 이기 때문에 사전 로그 정의가 더욱 중요하며 SLA, 수집 및 적재 전략이 사전에 협의 되어야만 합니다.

1. 데이터 소스 설계
2. 데이터 성격에 따른 SLA 설계 및 협의
3. 글로벌 데이터 정책
4. 수집 및 적재 전략

데이터 소스 설계

Data Source Design

파일 데이터 로그의 설계 시에 EventTime, ProcessingTime, Tag 와 Record 등의 필수 필드에 대한 설계가 필수이며, 데이터의 특성에 따라 어떤 시간을 기준 시간으로 지정할 것인지 결정할 필요가 있습니다. 특히 모바일 혹은 IoT 등의 이벤트 로그의 경우 수신 타임을 기준으로 데이터 처리하는 것을 추천하고, 서버, 시스템 및 애플리케이션 로그는 신뢰할 만한 시간을 보장하기 때문에 이벤트 타임을 기준으로 데이터 처리하는 것이 더 유용할 수 있습니다

컬럼 영문 명	컬럼 한글 명	유형	컬럼 설명
Event Time	이벤트 타임	String	해당 이벤트가 발생한 실제 시간 (yyyy-MM-dd HH:mm:ss:SSS)
Processing Time	프로세싱 타임	String	해당 이벤트를 시스템이 수신한 시간
Tag:Country	국가 코드	String	해당 이벤트의 국가 코드 (KR, JP 등)
Tag:Category	대분류 코드	Enum	해당 이벤트의 대분류 (계정, 접속, 매출 혹은 부서, 센터 등)
Tag:SubCateogry	중분류 코드	Enum	해당 이벤트의 중분류 (데이터의 상세 분류)
Tag:TableName	테이블 이름	String	테이블 이름
Records	추가 필드		

데이터 성격에 따른 SLA 설계 및 협의

SLA (Service License Agreement)

"데이터 엔지니어링" 에서의 "서비스 수준 협약" 이란 수집, 변환 및 적재하는 데이터의 지연, 중복, 유실 및 복구에 대한 정책을 말하며, 이에 따라 시스템 아키텍처, 운영 및 장애 처리에 대한 전략이 달라질 수 있습니다

1. 지연 정책 : 어느 정도 지연되는 데이터를 스테이징 데이터에 포함할 것인지를 결정
2. 중복 정책 : 중복을 허용할 것인지 혹은 스테이징 단계에서는 허용하고 후처리를 통해 제거할 것인지 결정
3. 유실 정책 : 어느 정도 유실을 허용할 것인지, 스테이징은 되지 않아도 백필을 통해서 제공할 것인지 결정
4. 복구 정책 : 장애 발생 이후 몇 시간 이내에 인지해야 하는지, 장애 인지 후 몇 시간 이내에 복구가 완료되어야 하는지 결정

글로벌 데이터 정책

Globalization

데이터 플랫폼이 글로벌 서비스를 위한 것인지 혹은 로컬 서비스만 제공하거나 글로벌 서비스를 하더라도 실제 사용자는 국내에만 있는 경우, 데이터 엔지니어링 설계 시에 검토해야 할 사항들이 있습니다. 데이터의 저장 및 변환에 따른 전략과 데이터 플랫폼 및 애플리케이션에 대한 전략입니다.

1. 데이터 수집 및 저장 시에 타임존 설정
   - Timestamp 값을 UTC 저장 여부에 따라 저장 및 조회 시에 변화가 크기 때문에 초반에 의사결정이 중요합니다
   - 국내 서비스만 하는 경우는 UTC 대신 Asia/Seoul 기준의 데이터로 저장하는 것이 좋습니다
   - UTC 의 경우 조회 시에 이용자 쿼리에 Timezone 을 고려한 조회 및 분석이 필요하고 관련 도구가 모두 TZ 를 고려해서 개발해야 합니다

2. 데이터 플랫폼의 L10N 적용 여부
   - 데이터 플랫폼의 경우 리소스 파일 및 각종 로깅 메시지 등에 대해서도 L10N 적용을 감안한 설계 및 프로그래밍 개발이 필요합니다
   - 미국과 같이 같은 국가내에서도 Timezone 에 따른 조회가 필요한 경우 반드시 UTC 로 저장되어야 하고 조회 시에도 TZ 를 고려한 설계 구현이 필요합니다

수집 및 적재 전략

Collecting and Loading Strategies

모든 데이터를 수집, 변환 및 적재 시에 기본이 되는 것은 Persistent 한 저장소이며, 클라우드 환경에서는 S3 와 같은 오브젝트 스토리지이고 On-Premise 환경에서는 Hadoop FileSystem 과 같은 분산 저장소가 될 것입니다. 이와 같이 한 번 결정되면 향후 사용하는 대부분의 지표에서 참조하기 때문에, 스테이징, 가공 및 서비스 제공을 위한 요약 지표 등을 저장하기 위한 계층을 사전에 정의할 필요가 있습니다. (정말 중요합니다)

1. Datasource : 모든 원본 데이터를 가공하지 않은 상태의 원본 데이터를 그대로 저장하는 공간입니다
2. Datastaging : 1차 가공된 데이터를 저장하거나 후처리를 통해 만들어진 가공 데이터를 저장하는 공간입니다
3. Datastore : 가공된 지표 혹은 요약 지표를 저장하는 영역이며, 이 데이터를 2차 저장소 혹은 레이턴시가 빠른 저장소로 내보내기도 합니다

저장공간	한글 명	경로 예제	컬럼 설명
Datasource Area	데이터소스 영역	s3://bucket/datasource	데이터 원본을 그대로 저장하는 공간
Datastaging Area	스테이징 영역	s3://bucket/datastaging	1차 가공 및 수집한 데이터를 저장하는 공간
Datastore Area	가공데이터 영역	s3://bucket/datastore	2차 가공 및 요약 지표 등의 데이터를 저장하는 공간

Q&A