

■ 빅데이터 수집



- 빅데이터의 수집 기술은 조직의 내외부에 있는 다양한 시스템으로부터 로우데이터(raw data)를 효과적으로 수집하는 기술
- 빅데이터 수집에는 기존의 수집 시스템보다 더 크고 다양한 형식의 데이터를 빠르게 처리해야 하는 기능이 필요한데, 그래서 확장이 가능하고 분산 처리가 가능한 형태로 구성
- 빅데이터 수집기는 로우(raw) 시스템의 다양한 인터페이스 유형(DB, 파일, API, 메시지 등)과 연결되어 정형, 반정형, 비정형 데이터를 대용량으로 수집

구조적 형태	저장 형태	예시	활용	처리 난이도
정형 데이터	DBMS File	RDBMS Excel	관계형 데이터베이스 시스템의 테이블과 같이 고정된 컬럼에 저장되는 데이터로써 Excel에 잘 정리된 테이블 형식의 데이터도 포함	하
반정형 데이터	File DBMS NoSQL	JSON XML HTML	JSON, XML, HTML 파일과 같이 형식을 가지고 있으며 구조화되어 있으나 메타데이터를 같이 포함하며 일반적으로 파일로 저장됨	중
비정형 데이터	File NoSQL	동영상 이미지 스크립트(기사, SNS 텍스트 등)	언어 분석이 가능한 기사, SNS 등의 텍스트 데이터 또는 이미지, 동영상과 같은 바이너리 데이터가 대표적인 비정형 데이터로 구분	상

- 특히 외부 데이터(SNS, 블로그, 포털 등)를 수집할 때는 크롤링 등 비정형 처리를 위한 기술이 선택적으로 적용됨

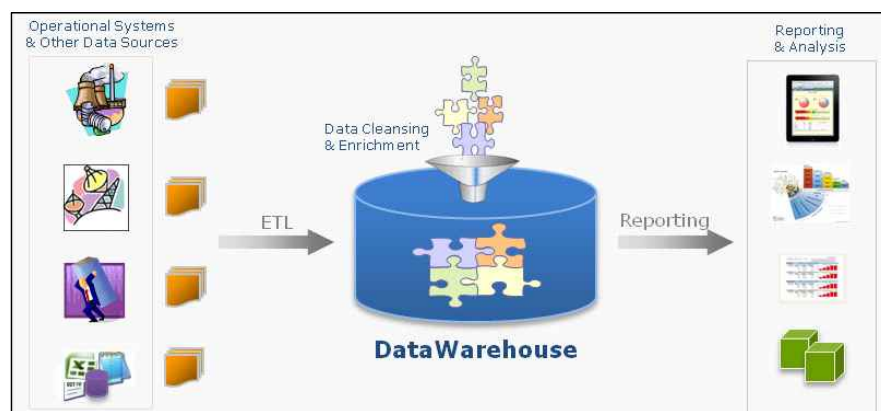
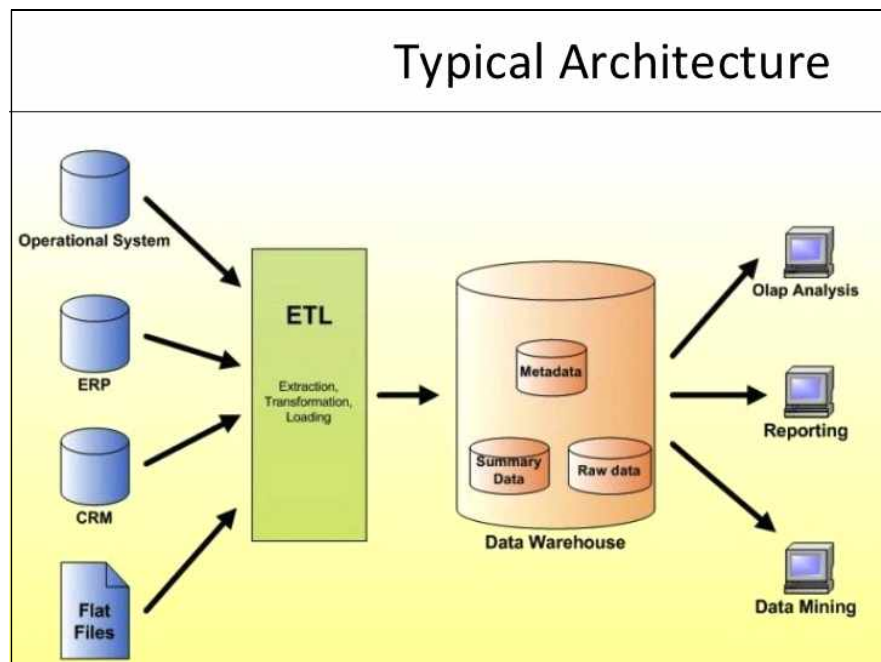
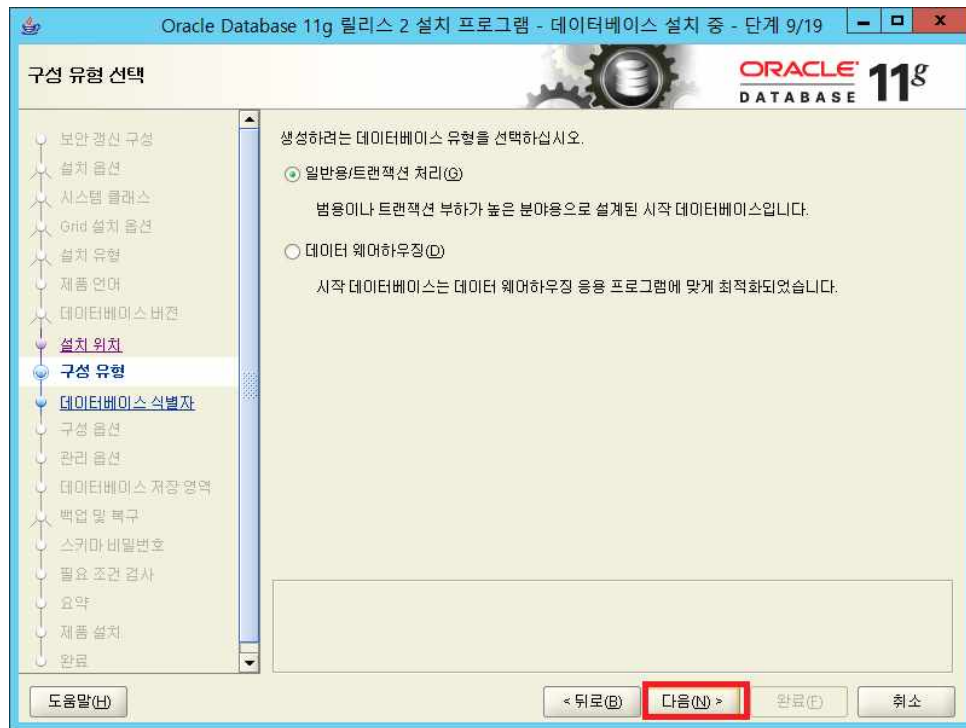
crawling이란?

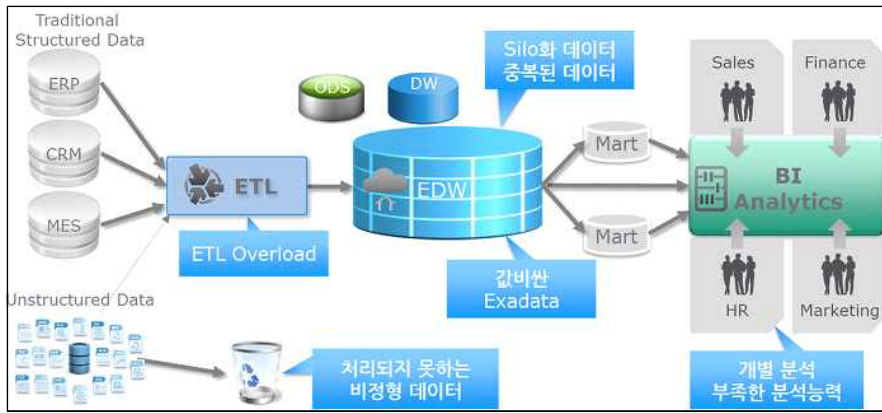
웹 페이지를 그대로 가져와서 거기서 데이터를 추출해 내는 행위

- 수집 처리에는 대용량 파일 수집과 실시간 스트림 수집으로 나뉨
- 실시간 수집의 경우 CEP(Complex Event Processing: 시간에 따라 변화하는 이벤트를 빠르게 분석할 수 있는 기술), ESP(Event Stream Processing: 다양하고 복잡한 이벤트를 분석할 수 있는 기술) 기술이 적용되어 수집 중인 데이터로 부터 이벤트를 감지해 빠른 후속 처리를 수행
- 수집된 데이터는 필요시 정제, 변환, 필터링 등의 작업을 추가로 진행해 데이터 품질을 향상시킨 후 빅데이터 저장소에 적재
- 아파치 플럼(flume)과 스쿱(scoop)이 대표적임

■ 데이터 레이크(Data Lake : 정보의 강)

1) 기존의 (빅)데이터 처리 개념 : 데이터 웨어하우스





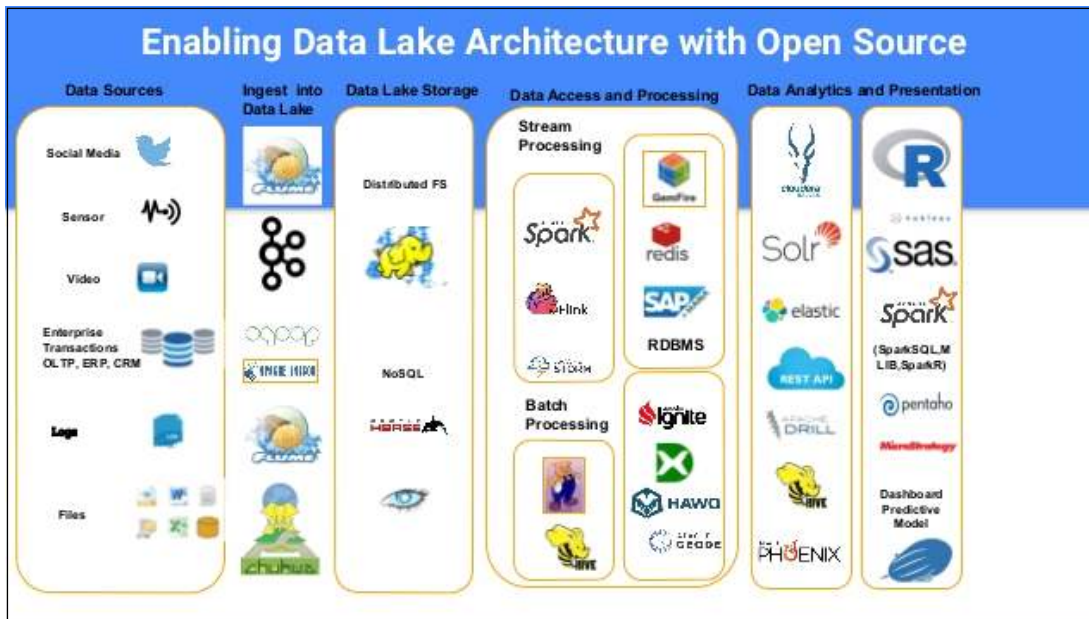
- 기존의 데이터웨어하우스는 RDBMS처럼 사전에 정의된 스키마(테이블)로 데이터를 변환하는 과정이 반드시 필요
- 이 과정을 ETL(Extract : 추출 / Transform : 변환 / Load : 적재)라고 함
- 이 ETL 개발단계를 거쳐 최종 애플리케이션에서 데이터를 활용하기까지 많은 시간과 노력, 비용이 소모되었으며, 특히 데이터 사용방법을 미리 결정해야 했음
- 만약 개발 초기단계에서 불필요하다고 판단했던 데이터는 ETL 과정에서 처리되지 못하고 버려지는데, 나중에 그 데이터가 필요한 경우가 생기면 큰일이 나는 경우가 빈번함

2) Data Lake의 개념

Data Lake	데이터 웨어하우스
데이터 웨어하우스를 보완(대체제는 아님)	Data Lake는 데이터웨어하우스의 소스가 될 수 있음
읽을 때 스키마를 처리(Schema on read)	쓸 때 미리 정의된 스키마로(Schema on write)
정형, 반정형, 비정형 데이터 전부	정형 데이터만
새로운 데이터/컨텐츠의 빠른 수집	ETL 과정이 필요하기 때문에 시간이 필요
데이터 과학+예측 고급 분석	BI에만 사용(고급 분석 불가능)
저수준(low level) 데이터 사용 가능	요약에서 집계된 세부적인 데이터만 사용가능
서비스 수준 협의가 느슨함	서비스 수준 협의가 매우 디테일해야 함
툴 선택에 유연함(고급 분석을 위한 오픈 소스툴)	툴의 선택이 유연하지 못함(SQL만 사용가능)

BI(Business Intelligence)란?

기업에서 데이터를 수집, 정리, 분석하고 활용하여 효율적인 의사결정을 할 수 있는 방법에 대해 연구하는 학문



1) 모든 데이터를 사용할 수 있음

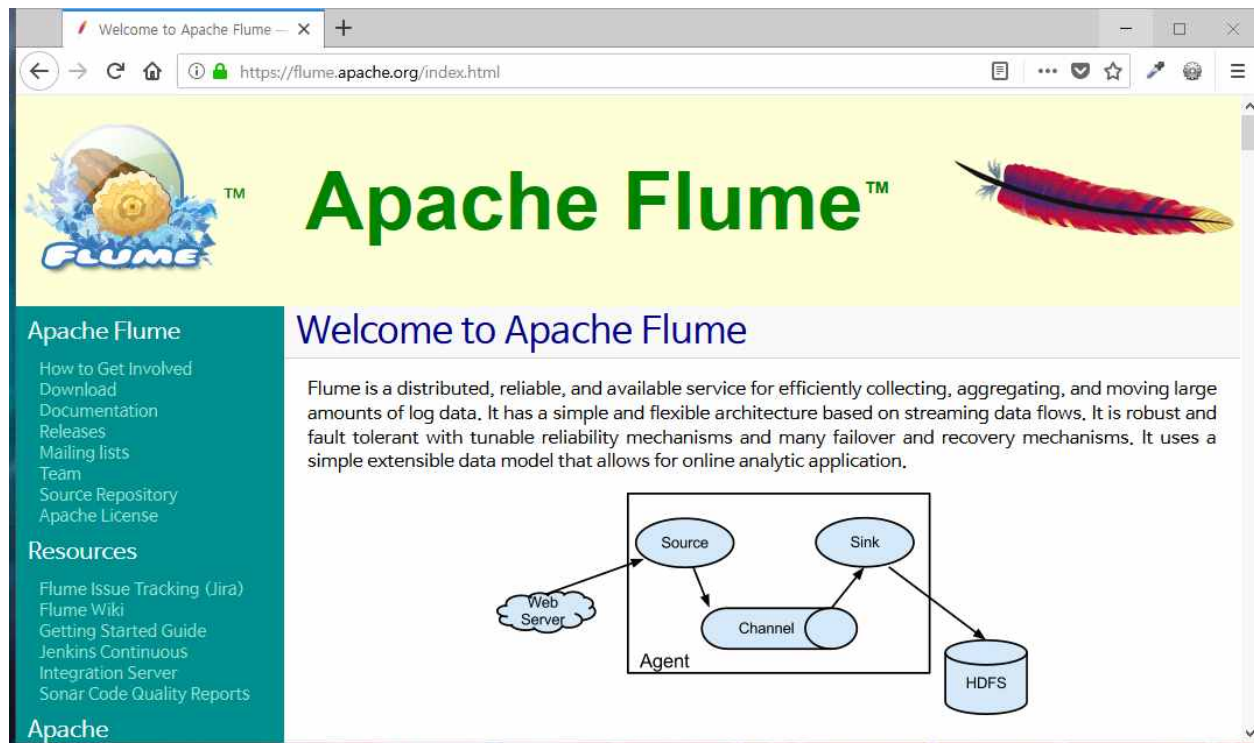
- 향후 데이터를 어떻게 사용할지 사전에 예측할 필요가 없음

2) 모든 데이터를 공유할 수 있음

3) 모든 데이터 접근 방식이 가능함

- 처리엔진(맵리듀스, 엘라스틱 서치, 스파크)과 애플리케이션(하이프, 스파크, SQL, 피그 등등)을 이용해 데이터를 탐색하고 처리할 수 있음

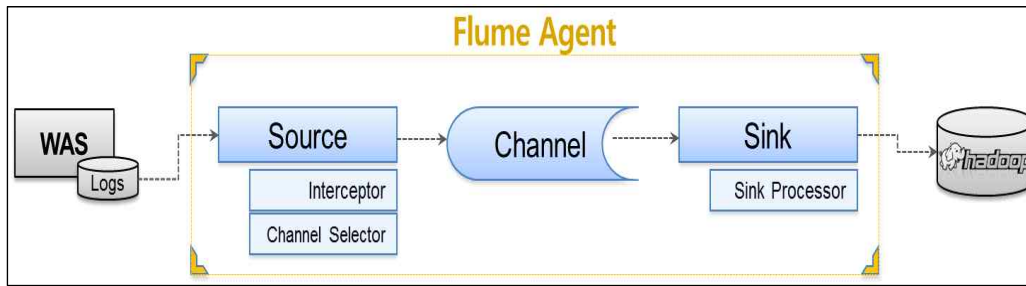
■ Apache Flume(아파치 플럼)



- <https://flume.apache.org/index.html>

- 1) 앞서 실행했던 것처럼, HDFS에 데이터를 입력할 때 아래와 같은 명령으로 간단히 처리 가능
-> `$ hdfs dfs -put [디렉토리]`
- 2) 이런 경우는 미리 잘 준비된 데이터를 업로드할 때 유용함
- 3) 하지만 현실에서는 서비스의 로그가 계속 유입이 되고, 즉시/대량으로 HDFS같은 DataStore에 저장되어 분석해야 함
- 4) 또한, 데이터를 유실 없이 안정적으로 전송하기 위해 다양한 옵션이 필요함
- 5) Hadoop으로 데이터를 입력하기 위해 간단하고 유연하며 확장이 가능한 솔루션으로서 Apache Flume이 적합함
- 6) 플럼은 데이터 스트림을 수집 / 전송하고 HDFS에 저장할 수 있는 도구
- 7) Apache Flume은 2011년에 Cloudera CDH3에 처음으로 소개되었으며, 현재는 Apache의 Top-Level Project로 이전 CDH3에서의 Version인 0.9.x버전은 Flume-0G라 명명하고 1.0.0 이후의 버전부터 Flume-NG(Next Generation)이라 명명함

■ Flume의 개념



1) 소스(Source)

- 이벤트를 수집하여 채널로 전달
- Interceptor : Source와 Channel 사이에서 데이터 필터링 및 가공
- Timestamp(이벤트 헤더에 현재 시간 값 추가),
- Static Interceptor(이벤트 헤더에 지정한 값 추가),
- Regex Filtering Interceptor (정규표현식에 일치하는지에 따라 이벤트를 버릴지 결정)
- Channel Selector : Source가 받은 이벤트를 전달할 채널을 선택
-

2) 소스의 종류

1. avro : Avro 클라이언트에서 전송하는 이벤트를 입력으로 사용, Agent와 Agent를 연결해줄 때 유용
2. netcat : TCP로 라인 단위 수집
3. seq : 0부터 1씩 증가하는 EVENT 생성
4. exec : 명령행을 실행하고 콘솔 출력 내용을 데이터 입력으로 사용
5. syslogtcp : System 로그를 입력으로 사용
6. spooldir : 디렉토리에 새롭게 추가되는 파일을 데이터로 사용
7. thrift : Thrift 클라이언트에서 전송하는 이벤트를 입력으로 사용
8. jms : JMS 메시지 수집

3) 채널(Channel)

- 이벤트를 Source와 Sink로 전달하는 통로
- 이벤트를 임시로 보관
- 종류: 메모리 채널, 파일 채널, JDBC채널
- 메모리 채널: Source에서 받은 이벤트를 Memory에 잠시 저장, 간편하고 빠른 고성능을 제공하지만 프로세스가 비정상적으로 종료시 이벤트 유실 가능성 있음
- 파일 채널: 속도는 메모리 채널에 비해 느리지만, 프로세스가 비정상적으로 죽더라도 프로세

스를 재처리하여 이벤트 유실이 없음

- JDBC 채널: JDBC로 저장

4) 싱크(Sink)

- Channel로 부터 받은 이벤트를 저장 또는 전달
- Sink Processor: 한 채널에 여러 Sink를 그룹으로 묶을 때 사용
- Sink Processor를 통해 Sink할 대상을 다중 선택하거나 여러개의 Sink를 하나의 그룹으로 관리하여 Failover(장애 극복)에 대응

- 종류

2. null : 이벤트를 버림

2. logger : 테스트 또는 디버깅을 위한 로깅

3. avro : 다른 Avro 서버(Avro Source)로 이벤트 전달

4. hdfs : HDFS에 저장

5. hbase : HBase에 저장

6. elasticsearch : 이벤트를 변환해서 Elasticsearch에 저장

7. file_roll : 로컬 파일에 저장

8. thrift : 다른 Thrift 서버(Thrift Source)로 이벤트 전달

5) Event : Flume을 통해 전달되는 데이터 단위



6) Event는 Header와 body로 나뉨

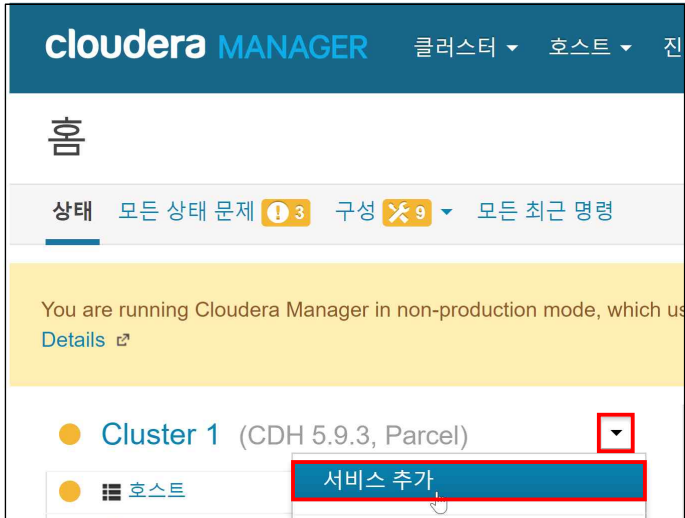
- Header는 key/value 형태이고 라우팅(데이터를 보낼 경로 선택)을 결정하거나 구조화된 정보 (예: 이벤트가 발생한 서버의 호스트명, timestamp 등 추가 가능)를 운반할 때 활용
- Body는 전달되는 데이터를 확인 가능

7) **Agent** : 이벤트를 전달하는 컨테이너, Source, Channel, Sink로 구성해 흐름을 제어함

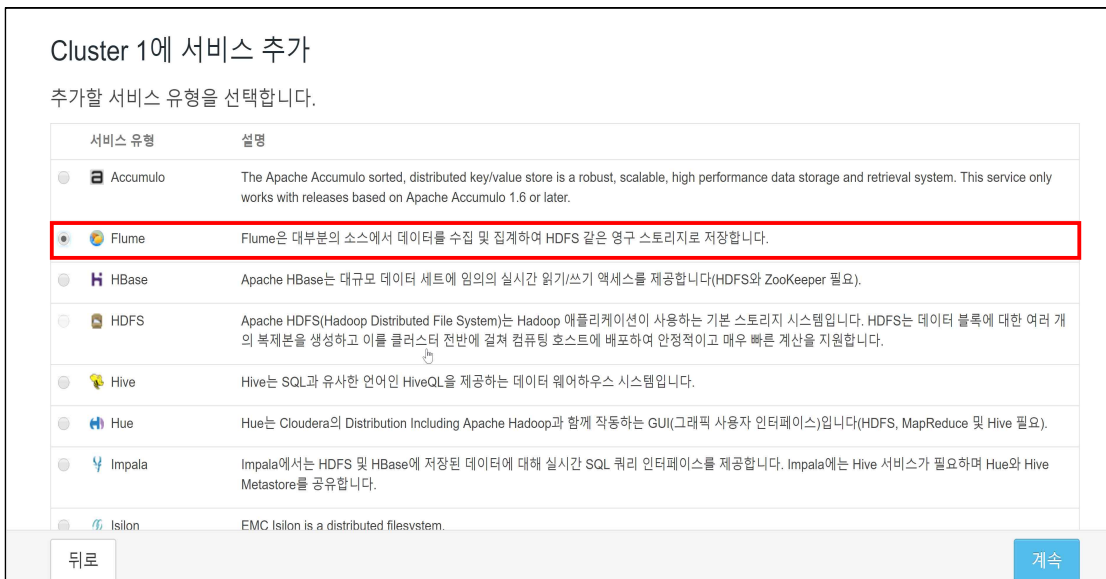
8) 에이전트끼리의 데이터 이동이 가능해서, (예: 1개의 에이전트가 다수의 에이전트와 연결 가능) 플룸은 목적에 따라 에이전트 간의 연결 및 확장이 가능

■ Flume 설치

1. [CM 홈]에서 [Cluster 1] 우측 콤보박스 클릭 -> [서비스 추가] 클릭



2. Cluster 1에 서비스 추가 [Flume] 선택 -> [계속]



3. [Agent 호스트 선택] 클릭 -> “server02.hadoop.com” 선택 -> [확인] 버튼 클릭



1개의 호스트가 선택됨

새 또는 기존 역할에 대한 호스트를 선택하십시오. 호스트 목록은 유효하지 않은 호스트를 제거하도록 필터링되었습니다. 이러한 호스트에는 상태가 불량하거나, 다른 클러스터의 구성원이거나, 호환되지 않는 CDH 버전이 설치되어 있는 호스트가 포함됩니다.

호스트 이름(host01, host[01-10]), IP 주소 또는 랙을 입력하십시오.

<input type="checkbox"/>	호스트 이름	IP 주소	랙	코어	물리적 메모리	기존 역할	추가된 역할
<input type="checkbox"/>	server01.hadoop.com	192.168.56.101	/default	1	4.7 GiB	B ON NN SNN JHS NM RM	
<input checked="" type="checkbox"/>	server02.hadoop.com	192.168.56.102	/default	1	4.7 GiB	A ON S	A
<input type="checkbox"/>	server03.hadoop.com	192.168.56.103	/default	1	2.7 GiB	DN AP ES HM RM SM	

팁: 첫 번째 확인란을 클릭하고 Shift 키를 누른 채로 마지막 확인란을 클릭하여 범위를 선택합니다.

Cluster 1에 Flume 서비스 추가

축하합니다!

새 서비스가 클러스터에 설치 및 구성되었습니다.

참고: 여전히 새 서비스를 시작해야 할 수 있습니다. 시작하기 전에 오래된 구성이 포함된 모든 종속 서비스를 재시작하는 것이 좋습니다. 이러한 작업은 아래에서 완료를 클릭하면 주 페이지에서 수행할 수 있습니다.

4. Heap Memory를 크게 늘림

4-1. CM 홈 -> [Flume] 클릭 -> [구성] 검색란에 “java heap” 검색 -> [Agent의 Java 힙 크기]에서 “100”MiB 로 늘린다. -> [변경 내용 저장] 버튼을 클릭

Flume (Cluster 1) 작업

상태 인스턴스 **구성** 명령 메트릭 세부 정보 차트 라이브러리 감사 쿼리 링크

클래스 레이아웃으로 전환 역할 그룹 기록 및 롤백

필터

범위

- Flume (서비스 차원) 0
- Agent 3

범주

- Flume-NG Solr 링크 0
- 고급 2
- 기본 0
- 로그 0
- 리소스 관리 1
- 모니터링 0
- 보안 0

java heap

메모리가 부족하면 힙 덤프 Agent Default Group

힙 덤프 디렉토리 Agent Default Group

oom_heap_dump_dir /tmp

Agent의 Java 힙 크기(단위: 바이트) Agent Default Group C

100 MiB

표시 25 페이지 기준

편집된 값 1개 변경 이유

5. CM 홈 -> [Flume] 우측 콤보박스 클릭 -> [시작] 선택

■ 플럼 수집 기능 구현

- HDFS로 적재하기 전, 플럼의 수집 기능을 테스트

1) 플럼 Agent 구성 파일 설명

Agent 이름: Test_Agent

Test_Agent 구성: Source-Interceptor-Channel-Sink

```
Test_Agent.sources = TestSource_SpoolSource
Test_Agent.channels = TestChannel_Channel
Test_Agent.sinks = TestSink_LoggerSink
```

- 플럼의 에이전트에 사용할 Source, Channel, Sink의 각 리소스 변수 정의
- Test_Agent Source의 변수는 TestSource_SpoolSource
- Test_Agent Channel의 변수는 TestChannel_Channel
- Test_Agent Sink의 변수는 TestSink_LoggerSink

```
Test_Agent.sources.TestSource_SpoolSource.type = spooldir
Test_Agent.sources.TestSource_SpoolSource.spoolDir = /home/smartCar-project/
working/smartcar-batch-log
Test_Agent.sources.TestSource_SpoolSource.deletePolicy = immediate
Test_Agent.sources.TestSource_SpoolSource.batchSize = 1000
```

- Test_Agent의 Source 설정
- spooldir: 파일이 생성되면 그 파일의 내용을 수집
- “/home/smartCar-project/working/smartcar-batch-log” 이 경로에서 생성되는 로그 파일 수집

```
Test_Agent.sources.TestSource_SpoolSource.interceptors = filterInterceptor
Test_Agent.sources.TestSource_SpoolSource.interceptors.filterInterceptor.type
= regex_filter
Test_Agent.sources.TestSource_SpoolSource.interceptors.filterInterceptor.regex
= ^\\d{14}
Test_Agent.sources.TestSource_SpoolSource.interceptors.filterInterceptor.exclud
eEvents = false
```

- Source로 들어온 이벤트는 Interceptor를 거친 뒤에 Channel로 전달
- Regex Filter Interceptor: 정규 표현식에 일치하는지에 따라 이벤트를 버릴지 결정
- filterInterceptor라는 변수를 선언해 TestSource_SpoolSource에 할당
- ^\\d{14}: 로그 데이터의 앞자리가 14자리 날짜형식이 아니면 버림

- “excludeEvents = false”이면 조건에 맞지 않는 데이터는 버리고 “excludeEvents = true”이면 제외 대상만 수집

```
Test_Agent.channels.TestChannel_Channel.type = memory
Test_Agent.channels.TestChannel_Channel.capacity = 100000
Test_Agent.channels.TestChannel_Channel.transactionCapacity = 10000
```

- Test_Agent의 Channel 설정
- Channel의 종류를 “memory”로 설정
- Memory Channel: Source로부터 받은 데이터를 메모리상에 중간 적재. 성능은 높지만 메모리에 저장하는 거라 데이터가 유실될 위험이 있음

```
Test_Agent.sinks.TestSink_LoggerSink.type = logger
```

- Test_Agent의 Sink 설정
- Logger Sink는 테스트용으로 수집한 데이터를 플럼의 표준 출력 로그 파일인 /var/log/flume-ng/flume-cmf-flume-AGENT-server02.hadoop.com.log 에 출력

```
Test_Agent.sources.TestSource_SpoolSource.channels = TestChannel_Channel
Test_Agent.sinks.TestSink_LoggerSink.channel = TestChannel_Channel
```

- Spooldir Source와 Memory Channel을 연결
- Logger Sink와 Memory Channel을 연결

2) Test_Agent 생성

[CM 홈] -> [Flume] -> [구성] -> [검색창]에 “구성” 검색 ->

Agent 이름: Test_Agent

구성 파일 : 나눠준 텍스트 복사/붙여넣기

-> [변경 내용 저장] 클릭 -> [Flume] 재시작



2. putty를 통해 Server02에 root 계정으로 로그인

3. “/home/smartCar-project/working/smartcarFile/” 경로로 이동해, 미리 저장해 놓은 로그파일 “SmartCarStatusInfo_20171227.txt”를 플럼에서 수집하는

“/home/smartCar-project/working/smartcar-batch-log/”경로에 복사

```
# cd /home/smartCar-project/working/smartcarFile/
```

```
# cp SmartCarStatusInfo_20171227.txt /home/smartCar-project/working/  
smartcar-batch-log/
```

4. Test_Agent의 수집이 정상적으로 작동되는지 확인

```
# tail -f /var/log/flume-ng/flume-cmf-flume-AGENT-server02.hadoop.com.log
```

```
[root@server02 working]# tail -f /var/log/flume-agg/flume-cmf-flume-AGENT-server02.hadoop.com.log
2017-12-26 15:33:06,233 INFO org.apache.flume.sink.LoggerSink: Event: { headers:{} body: 32 30 31 37 31 32 32 37 32 33 35 39 32 34 2C 45 20171227235924,E }
2017-12-26 15:33:06,233 INFO org.apache.flume.sink.LoggerSink: Event: { headers:{} body: 32 30 31 37 31 32 32 37 32 33 35 39 32 38 2C 45 20171227235928,E }
2017-12-26 15:33:06,233 INFO org.apache.flume.sink.LoggerSink: Event: { headers:{} body: 32 30 31 37 31 32 32 37 32 33 35 39 33 32 2C 45 20171227235932,E }
2017-12-26 15:33:06,233 INFO org.apache.flume.sink.LoggerSink: Event: { headers:{} body: 32 30 31 37 31 32 32 37 32 33 35 39 33 36 2C 45 20171227235936,E }
2017-12-26 15:33:06,233 INFO org.apache.flume.sink.LoggerSink: Event: { headers:{} body: 32 30 31 37 31 32 32 37 32 33 35 39 34 30 2C 45 20171227235940,E }
2017-12-26 15:33:06,233 INFO org.apache.flume.sink.LoggerSink: Event: { headers:{} body: 32 30 31 37 31 32 32 37 32 33 35 39 34 34 2C 45 20171227235944,E }
2017-12-26 15:33:06,233 INFO org.apache.flume.sink.LoggerSink: Event: { headers:{} body: 32 30 31 37 31 32 32 37 32 33 35 39 34 38 2C 45 20171227235948,E }
2017-12-26 15:33:06,233 INFO org.apache.flume.sink.LoggerSink: Event: { headers:{} body: 32 30 31 37 31 32 32 37 32 33 35 39 35 32 2C 45 20171227235952,E }
2017-12-26 15:33:06,233 INFO org.apache.flume.sink.LoggerSink: Event: { headers:{} body: 32 30 31 37 31 32 32 37 32 33 35 39 35 36 2C 45 20171227235956,E }
2017-12-26 15:33:06,233 INFO org.apache.flume.sink.LoggerSink: Event: { headers:{} body: 32 30 31 37 31 32 32 37 32 34 30 30 30 2C 45 20171227240000,E }
```

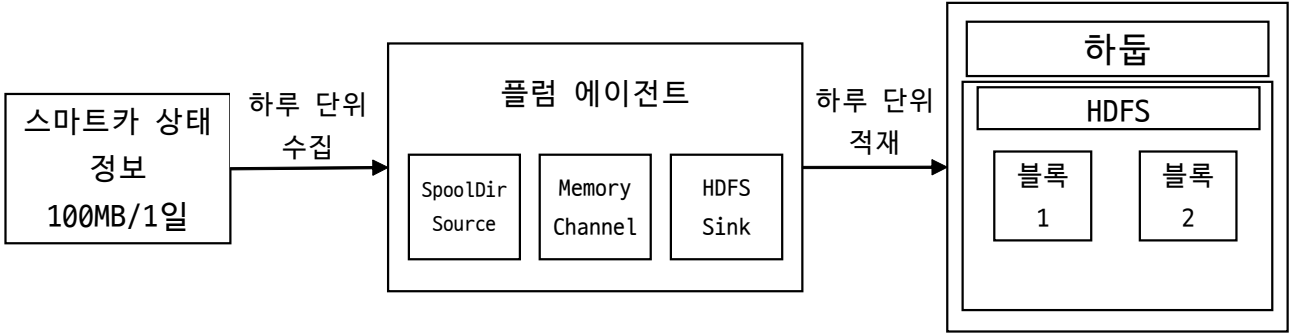
- 플럼의 이벤트가 header와 body로 이뤄진 것을 확인할 수 있음

■ 대용량 로그 파일 적재

- 플럼에서 수집한 로그 파일을 HDFS에 적재
- 요구사항(시나리오): 차량의 다양한 장치로부터 발생하는 로그 파일을 수집해서 기능 별 상태를 점검

적재 요구 사항 구체화	분석 및 해결방안
1. 100대의 스마트카 상태정보가 일 단위로 취합	플럼에서 수집발생 시점의 날짜를 Hdfs Sink에 전달해 해당 날짜 단위로 적재
2. 매일 100대의 스마트카 상태정보는 약 100MB정도 이며, 220만 건으로 상태 정보가 발생	1년 적재 시 8억 건 이상의 데이터 적재되며, 연 단위 분석에 하둡 분산 병렬 처리 사용
3. 스마트카의 상태 정보 데이터의 발생일과 수집/적재되는 날짜가 다를 수 있다.	수집/적재 도는 모든 데이터마다 발생일 외에 수집/적재 처리돼야 하는 처리일 추가
4. 적재된 스마트카들의 상태 정보를 일, 월, 년 단위로 분석해야함	HDFS에 수집 일자별로 디렉터리 경로를 만들어서 적재
5. 적재 및 생성되는 파일은 HDFS의 특성을 고려	플럼의 HdfsSink의 옵션을 스마트카 프로젝트의 HDFS에 최적화해서 설정
6. 적재가 완료된 후 원본 파일은 삭제	플럼의 Source 컴포넌트 중 SpoolDir의 DeletePolicy 옵션 사용

- 프로젝트 아키텍처



1) 플럼 Agent 구성 파일 설명

Agent 이름: SmartCar_Agent

SmartCar_Agent 구성: Source-Interceptor-Channel-Sink

Agent

```
SmartCar_Agent.sources = SmartCarInfo_SpoolSource
SmartCar_Agent.channels = SmartCarInfo_Channel
SmartCar_Agent.sinks = SmartCarInfo_HdfsSink
```

- HdfsSink 로 연결

```
SmartCar_Agent.sources.SmartCarInfo_SpoolSource.interceptors =
timeInterceptor typeInterceptor collectDayInterceptor filterInterceptor
```

- 타임스탬프를 사용하기 위한 “timeInterceptor”와 수집 일자를 추가하기 위한 만들어진 “collectDayInterceptor” 추가

```
SmartCar_Agent.sources.SmartCarInfo_SpoolSource.interceptors.timeInterceptor.type = timestamp
SmartCar_Agent.sources.SmartCarInfo_SpoolSource.interceptors.timeInterceptor.preserveExisting = true
```

- “timeInterceptor” 설정. 플럼의 헤더에 현재 타임 스탬프가 설정되어 필요하면 헤더로부터 타임스탬프값 가져와 활용

```
SmartCar_Agent.sources.SmartCarInfo_SpoolSource.interceptors.typeInterceptor.type = static
SmartCar_Agent.sources.SmartCarInfo_SpoolSource.interceptors.typeInterceptor.key = logType
SmartCar_Agent.sources.SmartCarInfo_SpoolSource.interceptors.typeInterceptor.value = smartcar-batch-log
```

- 플럼의 해당 이벤트 내에 사용할 상수 값 설정. “logType” 이라는 상수를 선언했고 값은 “smartcar-batch-log”. “smartcar-batch-log” 값은 HDFS에 적재된 데이터들을 구분 할 수 있게 사용

```
SmartCar_Agent.sources.SmartCarInfo_SpoolSource.interceptors.collectDayIntercep
tor.type = com.jbm.bigdata.smartcar.flume.CollectDayInterceptor$Builder
```

- 사용자 정의 인터셉터인 “collectDayInterceptor” 설정. 플럼 이벤트 바디에 수집된 당일의 작업날짜(YYYYMMDD)를 추가하는 인터셉터

```
public String getToDate() {  
  
    long todaytime;  
    SimpleDateFormat day;  
    String toDay;  
  
    todaytime = System.currentTimeMillis();  
    day = new SimpleDateFormat("yyyyMMdd");  
  
    toDay = day.format(new Date(todaytime));  
  
    return toDay;  
}
```

```
@Override  
public Event intercept(Event event) {  
  
    String eventBody = new String(event.getBody()) + "," + getToDate();  
    event.setBody(eventBody.getBytes());  
    return event;  
}
```

- 소스를 살펴보면, 현재 날짜를 “yyyyMMdd”형식으로 가져와서 바디 뒤에 “,”를 붙이고 추가됨

```
SmartCar_Agent.sinks.SmartCarInfo_HdfsSink.type = hdfs  
SmartCar_Agent.sinks.SmartCarInfo_HdfsSink.hdfs.path =  
/jbm/collect/{logType}/wrk_date=%Y%m%d  
SmartCar_Agent.sinks.SmartCarInfo_HdfsSink.hdfs.filePrefix = {logType}  
SmartCar_Agent.sinks.SmartCarInfo_HdfsSink.hdfs.fileSuffix = .log  
SmartCar_Agent.sinks.SmartCarInfo_HdfsSink.hdfs.fileType = DataStream  
SmartCar_Agent.sinks.SmartCarInfo_HdfsSink.hdfs.writeFormat = Text  
SmartCar_Agent.sinks.SmartCarInfo_HdfsSink.hdfs.batchSize = 10000  
SmartCar_Agent.sinks.SmartCarInfo_HdfsSink.hdfs.rollInterval = 0  
SmartCar_Agent.sinks.SmartCarInfo_HdfsSink.hdfs.rollCount = 0  
SmartCar_Agent.sinks.SmartCarInfo_HdfsSink.hdfs.idleTimeout = 100  
SmartCar_Agent.sinks.SmartCarInfo_HdfsSink.hdfs.callTimeout = 600000  
SmartCar_Agent.sinks.SmartCarInfo_HdfsSink.hdfs.rollSize = 67108864  
SmartCar_Agent.sinks.SmartCarInfo_HdfsSink.hdfs.threadPoolSize = 10
```

- HDFS sink 상세 설정값

- 앞에 정의한 Interceptor의 값을 이용해 HDFS의 경로를 동적으로 나누는 “path” 설정
- 적재 시 HDFS에 생성되는 파일명의 규칙
- 파일크기(64MB)등 정의

```
SmartCar_Agent.sources.SmartCarInfo_SpoolSource.channels = SmartCarInfo_Channel
SmartCar_Agent.sinks.SmartCarInfo_HdfsSink.channel = SmartCarInfo_Channel
```

- Spooldir Source와 Memory Channel을 연결
- HDFS Sink와 Memory Channel을 연결

■ 실행

1) “bigdata.smartcar.flume-1.0.jar”를 파일질라를 통해 플럼 라이브러리 디렉터리에 추가

호스트:

sftp://server02.hadoop.com/opt/cloudera/parcels/CDH-5.9.3-1.cd5.9.3.p0.4/lib/flume-ng/lib

사용자명: root

비밀번호: 1111

포트: 22

2) SmartCar_Agent 생성

[CM 홈] -> [Flume] -> [구성] -> [검색창]에 “구성” 검색 ->

Agent 이름: SmartCar_Agent

구성 파일 : 나눠준 텍스트 복사/붙여넣기

-> [변경 내용 저장] 클릭 -> [Flume] 재시작

Flume (Cluster 1) 작업

Today, 6:27 PM KST

상태 인스턴스 구성 명령 메트릭 세부 정보 차트 라이브러리 감사 쿼리 링크

클래식 레이아웃으로 전환 역할 그룹 기록 및 블록

필터

범위

- Flume (서비스 차원) 13
- Agent 31

범주

- Flume-NG Solr 링크 0
- 고급 6
- 기본 2
- 로그 0
- 리소스 관리 0
- 모니터링 9
- 보안 0

구성

Agent 이름 Agent Default Group C SmartCar_Agent

구성 파일 Agent Default Group C

```
SmartCar_Agent.sources = SmartCarInfo_SpoolSource
SmartCar_Agent.channels = SmartCarInfo_Channel
SmartCar_Agent.sinks = SmartCarInfo_HdfsSink
```

Flume 서비스 환경 고급 구성 스니펫(안전 밸브) Flume(서비스 전체)

편집된 값 2개 변경 이유 변경 내용 저장

3. putty를 통해 Server02에 접속

4. 로그 파일 “SmartCarStatusInfo_20171227.txt”을 /home/smartCar-project/working/
smartcar-batch-log/ 로 복사

```
# cp SmartCarStatusInfo_20171227.txt /home/smartCar-project/working/  
smartcar-batch-log/
```

5. 플럼의 실행 로그를 통해 SmartCar_Agent가 정상적으로 작동하는지 확인

```
# tail -f /var/log/flume-ng/flume-cmf-flume-AGENT-server02.hadoop.com.log
```

```
2017-12-26 19:03:52,663 INFO org.apache.flume.source.SpooledDirectorySource: SpooledDirectorySource source starting with directory: /home/smartCar-project/working/smartcar-batch-log
2017-12-26 19:03:52,727 INFO org.apache.flume.instrumentation.MonitoredCounterGroup: Monitored counter group for type: SINK, name: SmartCarInfo_HdfsSink: Successfully registered ne
w MBean.
2017-12-26 19:03:52,727 INFO org.apache.flume.instrumentation.MonitoredCounterGroup: Component type: SINK, name: SmartCarInfo_HdfsSink started
2017-12-26 19:03:52,765 INFO org.apache.flume.instrumentation.MonitoredCounterGroup: Monitored counter group for type: SOURCE, name: SmartCarInfo_SpoolSource: Successfully register
ed new MBean.
2017-12-26 19:03:52,765 INFO org.apache.flume.instrumentation.MonitoredCounterGroup: Component type: SOURCE, name: SmartCarInfo_SpoolSource started
2017-12-26 19:03:52,807 INFO org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter(org.mortbay.log) via org.mortbay.log.Slf4jLog
2017-12-26 19:03:52,992 INFO org.mortbay.log: jetty-6.1.26.cloudera.4
2017-12-26 19:03:53,059 INFO org.mortbay.log: Started SelectChannelConnector@0.0.0.0:41414
2017-12-26 19:03:53,550 INFO org.apache.flume.sink.hdfs.HDFSDataStream: Serializer = TEXT, UseRawLocalFileSystem = false
2017-12-26 19:03:54,423 INFO org.apache.flume.sink.hdfs.BucketWriter: Creating /jbm/collect/smartcar-batch-log/wrk_date=20171226/smartcar-batch-log.1514282633551.log.tmp
2017-12-26 19:04:27,721 INFO org.apache.flume.sink.hdfs.BucketWriter: Closing /jbm/collect/smartcar-batch-log/wrk_date=20171226/smartcar-batch-log.1514282633551.log.tmp
2017-12-26 19:04:27,748 INFO org.apache.flume.sink.hdfs.BucketWriter: Renaming /jbm/collect/smartcar-batch-log/wrk_date=20171226/smartcar-batch-log.1514282633551.log.tmp to /jbm/co
llect/smartcar-batch-log/wrk_date=20171226/smartcar-batch-log.1514282633551.log
2017-12-26 19:04:27,786 INFO org.apache.flume.sink.hdfs.BucketWriter: Creating /jbm/collect/smartcar-batch-log/wrk_date=20171226/smartcar-batch-log.1514282633552.log.tmp
2017-12-26 19:04:50,317 INFO org.apache.flume.client.avro.ReliableSpoolingFileEventReader: Last read took us just up to a file boundary. Rolling to the next file, if there is one.
2017-12-26 19:04:50,317 INFO org.apache.flume.client.avro.ReliableSpoolingFileEventReader: Preparing to delete file /home/smartCar-project/working/smartcar-batch-log/SmartCarStatus
Info_20171227.txt
2017-12-26 19:06:34,954 INFO org.apache.flume.sink.hdfs.BucketWriter: Closing idle bucketWriter /jbm/collect/smartcar-batch-log/wrk_date=20171226/smartcar-batch-log.1514282633552.1
og.tmp at 1514282794954
2017-12-26 19:06:34,954 INFO org.apache.flume.sink.hdfs.BucketWriter: Closing /jbm/collect/smartcar-batch-log/wrk_date=20171226/smartcar-batch-log.1514282633552.log.tmp
2017-12-26 19:06:34,977 INFO org.apache.flume.sink.hdfs.BucketWriter: Renaming /jbm/collect/smartcar-batch-log/wrk_date=20171226/smartcar-batch-log.1514282633552.log.tmp to /jbm/co
llect/smartcar-batch-log/wrk_date=20171226/smartcar-batch-log.1514282633552.log
2017-12-26 19:06:34,982 INFO org.apache.flume.sink.hdfs.HDFSEventSink: Writer callback called.
```

- “Creating /jbm/collect/smartcar-batch-log/...” : HDFS에 플럼에서 수집한 파일이 저장되는
경로가 생성됨

- “Writer callback called.” : 적재 완료

6. HDFS에 접속해 저장된 블록 단위로 나눠져 저장된 파일을 확인

```
# hdfs dfs -ls -R /jbm
```

```
[root@server02 smartcar-batch-log]# hdfs dfs -ls -R /jbm
drwxr-xr-x - flume supergroup 0 2017-12-26 19:03 /jbm/collect
drwxr-xr-x - flume supergroup 0 2017-12-26 19:03 /jbm/collect/smartcar-batch-log
drwxr-xr-x - flume supergroup 0 2017-12-26 19:06 /jbm/collect/smartcar-batch-log/wrk_date=20171226
-rw-r--r-- 2 flume supergroup 68303748 017-12-26 19:04 /jbm/collect/smartcar-batch-log/wrk_date=20171226/smartcar-batch-log.1514282633551.log
-rw-r--r-- 2 flume supergroup 55198666 017-12-26 19:06 /jbm/collect/smartcar-batch-log/wrk_date=20171226/smartcar-batch-log.1514282633552.log
```

- 스마트카 상태 정보 파일이 두 개의 블록으로 나눠져서 들어간 것을 확인할 수 있다.

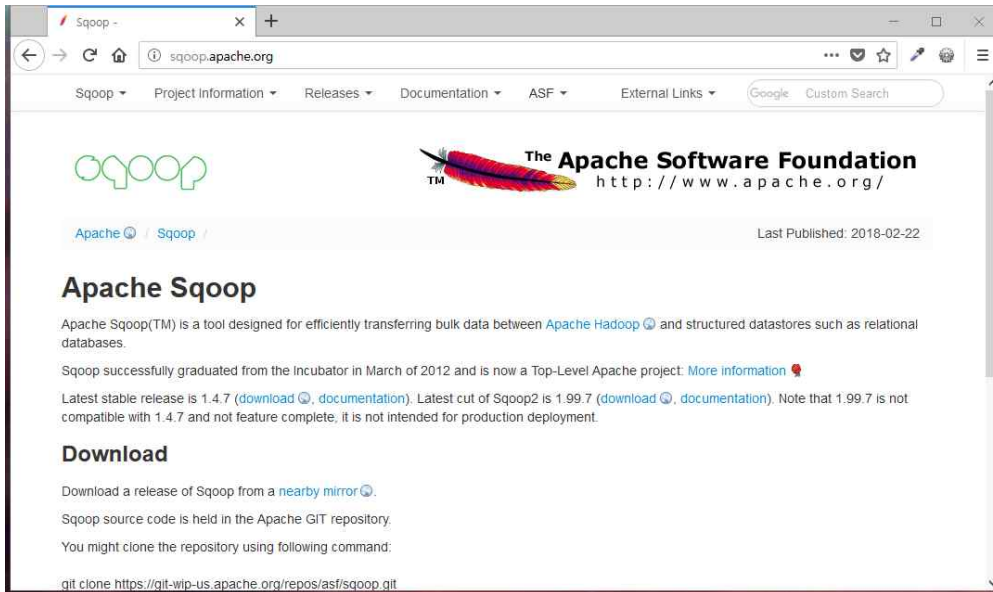
- wrk_date에서 적재한 날짜를 확인할 수 있다.

7. HDFS 저장 된 파일 내용 확인

```
# hdfs dfs -tail /jbm/collect/smartcar-batch-log/wrk_date=20171226  
/smartcar-batch-log.1514282633552.log
```

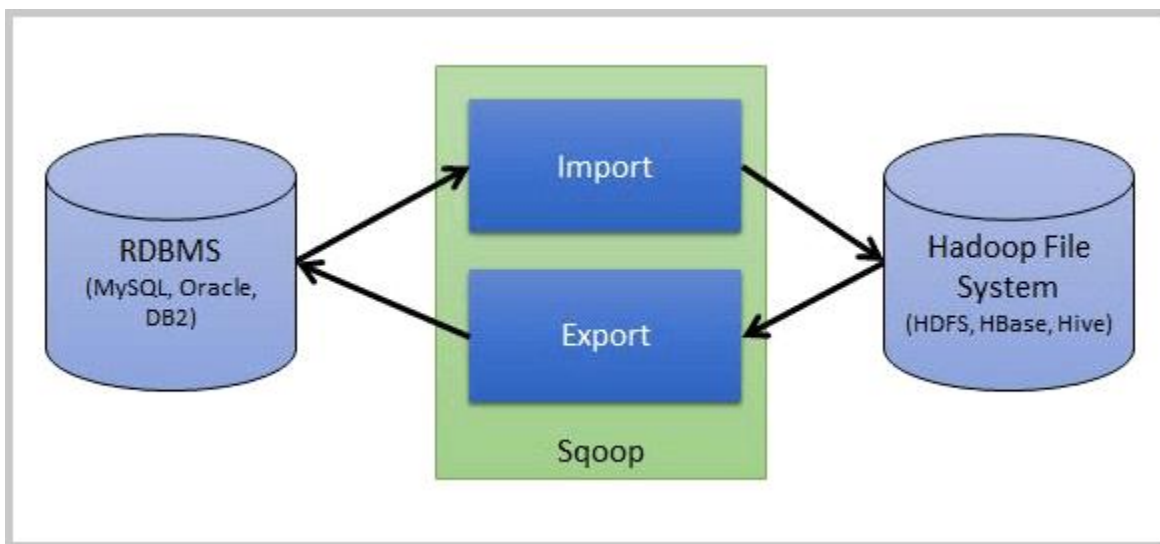
```
[root@server02 smartcarFile]# hdfs dfs -tail /jbm/collect/smartcar-batch-log/wrk_date=20171226/smartcar-batch-log.1514282633552.log
04,73,60,95,1,1,1,1,A,A,91,20171226
20171227235856,0090,79,90,60,89,1,1,1,1,A,A,0,20171226
20171227235900,0090,100,89,60,88,1,1,1,1,B,A,94,20171226
20171227235904,0090,99,86,60,97,1,1,1,1,A,A,4,20171226
20171227235908,0090,93,86,60,96,1,1,1,1,B,A,4,20171226
20171227235912,0090,82,82,60,99,1,1,1,1,A,A,0,20171226
20171227235916,0090,73,99,60,86,1,1,1,1,A,A,7,20171226
20171227235920,0090,72,99,60,95,1,1,1,1,A,B,9,20171226
20171227235924,0090,89,91,60,100,1,1,1,1,A,A,98,20171226
20171227235928,0090,95,89,60,88,1,1,1,1,A,A,1,20171226
20171227235932,0090,79,81,60,85,1,1,1,1,A,B,7,20171226
20171227235936,0090,97,99,60,71,1,1,1,1,A,B,2,20171226
20171227235940,0090,70,78,60,90,1,1,1,1,B,A,7,20171226
20171227235944,0090,99,80,60,77,1,1,1,1,A,A,00,20171226
20171227235948,0090,70,80,60,100,1,1,1,1,A,A,95,20171226
20171227235952,0090,73,94,60,85,1,1,1,1,B,A,6,20171226
20171227235956,0090,96,83,60,82,1,1,1,1,A,A,9,20171226
20171227240000,0090,90,89,60,99,1,1,1,1,A,A,0,20171226
```

■ 아파치 스쿱(Apache Sqoop)



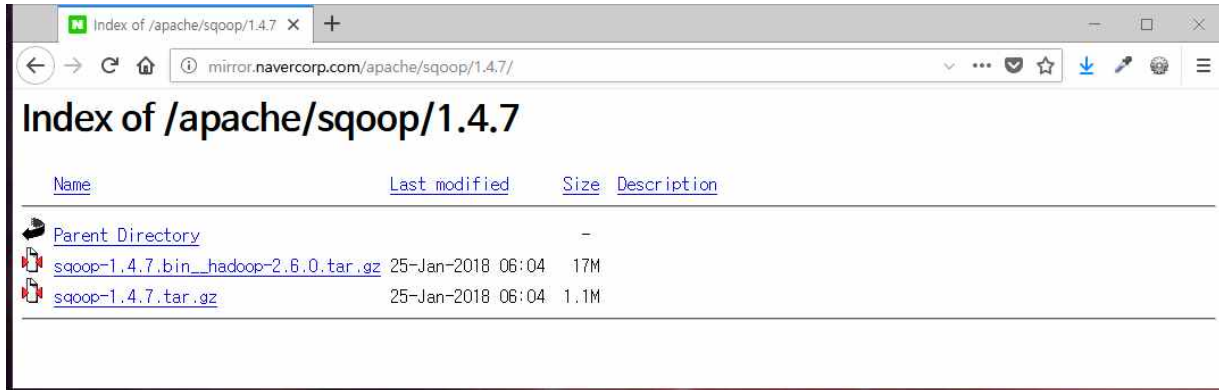
<http://sqoop.apache.org/>

- 1) **관계형 데이터베이스와 하둡 사이에서 데이터 이관을 지원하는 툴**
- 2) 기존에 운영 중인 관계형 데이터베이스(RDBMS)에 저장된 데이터를 처리
- 3) 하둡에서 수집한 비정형 데이터와 기존 데이터베이스에 저장된 정보를 조인해서 분석할 수도 있음



■ 스쿱의 설치

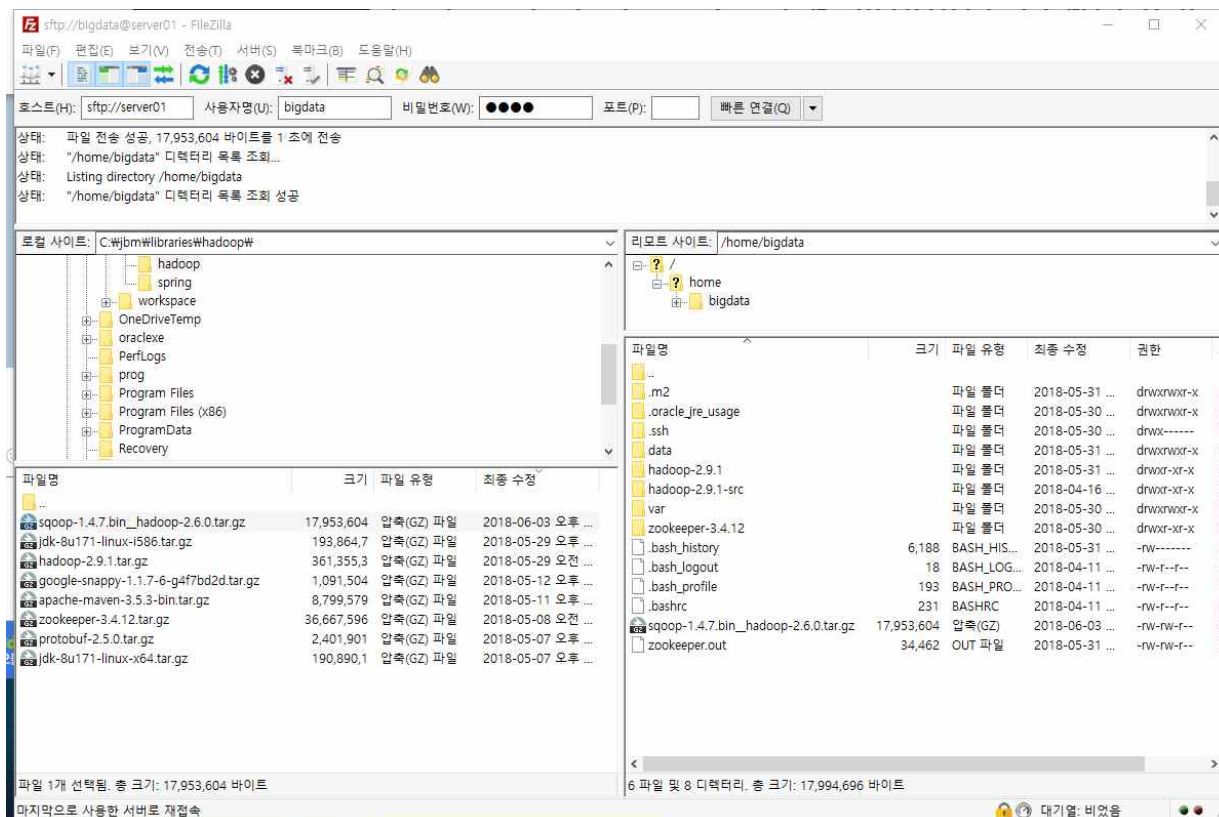
- 1) <http://mirror.navercorp.com/apache/sqoop/1.4.7/>에서
sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz를 다운로드



- 혹은 **server01**의 **bigdata**계정으로 접속후 wget 다운로드

```
[bigdata@server01 ~]$  
wget mirror.navercorp.com/apache/sqoop/1.4.7/sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz
```

- 2) filezilla를 이용하여 tar.gz 파일을 복사



3) 압축 풀기

```
[bigdata@server01 ~]$ tar xvfz sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz
```

4) 테스트용 테이블 만들기(oracle에)

Column Name	Data Type	Nullable	Default	Primary Key
NO	NUMBER(9,0)	No	-	1
TITLE	VARCHAR2(100)	No	-	-
AUTHOR	VARCHAR2(100)	No	-	-
PUBLISH_DATE	DATE	Yes	-	-
				1 - 4

```
CREATE table "BOOKS" (  
    "NO"          NUMBER(9,0) NOT NULL,  
    "TITLE"       VARCHAR2(100) NOT NULL,  
    "AUTHOR"      VARCHAR2(100) NOT NULL,  
    "PUBLISH_DATE" DATE,  
    constraint "BOOKS_PK" primary key ("NO")  
)  
  
CREATE SEQUENCE "BOOKS_SEQ" MINVALUE 1 MAXVALUE 99999999999999999999 INCREMENT BY 1 START WITH 1  
NOCACHE NOORDER NOCYCLE ;
```

5) 테스트용 테이블에 입력하기

```
INSERT INTO books(no, title, author, publish_date)  
VALUES(books_seq.nextval, '인간관계론', '데일카네기', '2006-12-21');  
  
INSERT INTO books(no, title, author, publish_date)  
VALUES(books_seq.nextval, '부러진 용골', '요네자와 호노부', '2012-05-25');  
  
INSERT INTO books(no, title, author, publish_date)  
VALUES(books_seq.nextval, '정의는 무엇인가', '마이클 샌델', '2010-05-26');  
  
INSERT INTO books(no, title, author, publish_date)  
VALUES(books_seq.nextval, '오체불만족', '오토타케 히로타다', '2001-03-31');  
  
INSERT INTO books(no, title, author, publish_date)  
VALUES (books_seq.nextval, '반지의 제왕', 'J.R.R. 톨킨', '1954-07-29');  
  
INSERT INTO books(no, title, author, publish_date)  
VALUES(books_seq.nextval, '제3인류', '베르나르 베르베르', '2012-10-02');
```



```

INSERT INTO books(no, title, author, publish_date)
VALUES(books_seq.nextval, '상록수', '심훈', '2005-06-25');

INSERT INTO books(no, title, author, publish_date)
VALUES(books_seq.nextval, '철강왕 카네기', '카네기', '1993-07-11');

INSERT INTO books(no, title, author, publish_date)
VALUES(books_seq.nextval, '파리의 아파트', '기욤 뮈소', '2017-12-05');

INSERT INTO books(no, title, author, publish_date)
VALUES(books_seq.nextval, '나무', '베르나르 베르베르', '2013-05-30');

INSERT INTO books(no, title, author, publish_date)
VALUES(books_seq.nextval, '나의 라임 오렌지 나무', '조제 마우루 지 바스콘셀루스', '2001-04-03');

INSERT INTO books(no, title, author, publish_date)
values(books_seq.nextval, '원피스', '오다 에이이치로', '1999-01-31');

INSERT INTO books(no, title, author, publish_date)
values(books_seq.nextval, '신비한 동물사전', 'J. K. 롤링', '2018-01-25');

INSERT INTO books(no, title, author, publish_date)
VALUES(books_seq.nextval, '회랑정 살인사건', '히가시노 게이고', '2016-10-18');

INSERT INTO books(no, title, author, publish_date)
VALUES(books_seq.nextval, '모모', '미하엘 엔데', '1999-02-09');

```

6) 데이터 import / export 옵션들

- (import / export시) 공통인자

인자	설명
<code>--connect <jdbc-uri></code>	JDBC 연결 문자열을 명시한다.
<code>--connection-manager <클래스명></code>	사용할 연결 관리자 클래스를 명시한다.
<code>--driver <클래스명></code>	사용할 JDBC 드라이버 클래스를 별도로 명시한다.
<code>--hadoop-home <dir></code>	\$HADOOP_HOME을 덮어 쓴다.
<code>--help</code>	사용법을 출력한다.
<code>-P</code>	콘솔에서 비밀번호를 읽어들인다.
<code>--password <비밀번호></code>	인증 비밀번호를 설정한다.
<code>--username <사용자명></code>	인증 사용자명을 설정한다.
<code>--verbose</code>	동작 시 더 많은 정보를 출력한다.
<code>--connection-param-file <파일명></code>	연결 매개변수들을 제공하는 속성 파일을 선택적으로 명시한다.

- import 인자

인자	설명
--append	HDFS에 이미 존재하는 데이터셋에 데이터를 붙인다.
--as-avrodatafile	Avro 데이터 파일로 데이터를 가져온다.
--as-sequencefile	SequenceFile에 데이터를 가져온다.
--as-textfile	(기본값) 순수한 텍스트로서 데이터를 가져온다.
--boundary-query <문장>	스플릿을 생성하기 위하여 사용하는 경계 쿼리
--columns <컬럼,컬럼,컬럼...>	테이블로부터 가져올 컬럼들
--direct	직접 고속 가져오기 모드를 사용한다.
--direct-split-size <n>	직접 모드로 가져올 때 모든 n바이트 당 입력 스트림을 분할시킨다.
--inline-lob-limit <n>	인라인 LOB를 위한 최대 사이즈를 설정한다.
-m,--num-mappers <n>	병렬로 가져오기 위하여 n개의 맵 태스크를 사용한다.
-e,--query <쿼리문>	쿼리문의 결과를 가져온다.
--split-by <컬럼명>	분할 단위로 쓰이는 테이블의 컬럼
--table <테이블명>	읽을 테이블
--target-dir <디렉토리>	목적지 HDFS 디렉토리
--warehouse-dir <디렉토리>	테이블 목적지를 위한 부모 HDFS 디렉토리
--where <where 절>	가져오기 동안 사용할 WHERE절
-z,--compress	압축을 사용하게 한다.
--compression-codec <c>	(기본은 gzip) Hadoop 코덱을 사용한다.
--null-string <null-string>	문자열 컬럼에서 널 값 대신 쓰여질 문자열
--null-non-string <null-string>	문자열이 아닌 컬럼에서 널 값 대신 쓰여질 문자열

- export 인자

인자	설명
--direct	직접 고속 내보내기 모드를 사용한다.
--export-dir <디렉토리경로>	내보내기를 위한 HDFS 소스 경로
-m,--num-mappers <개수>	병렬로 내보내기하기 위한 n 개의 맵 태스크를 사용한다.
--table <테이블명>	데이터를 채울 테이블명
--update-key <컬럼명>	갱신을 위하여 사용하는 고정(anchor) 컬럼. 하나 이상의 컬럼이 있을 경우, 콤마로 분리되는 컬럼 목록을 사용한다.
--update-mode <mode>	데이터베이스에서 새 행들이 매칭되지 않는 키로 발견이 되었을 때 어떻게 갱신이 수행되는지를 명시한다. mode 에 해당하는 값은 updateonly (기본)과 allowinsert 이다.
--input-null-string <널문자열>	문자열 컬럼에 대하여 널로 해석될 문자열
--input-null-non-string <널문자열>	문자열이 아닌 컬럼에 대하여 널로 해석될 문자열
--staging-table <스테이징 테이블명>	목적 테이블에 삽입되기 전에 데이터가 스테이징될 테이블

<code>--clear-staging-table</code>	스테이징 테이블에 있는 어떤 데이터도 삭제될 수 있는지를 나타낸다.
<code>--batch</code>	내부적으로 문장을 실행할 때 배치 모드를 사용한다.

7) 데이터 import 실행

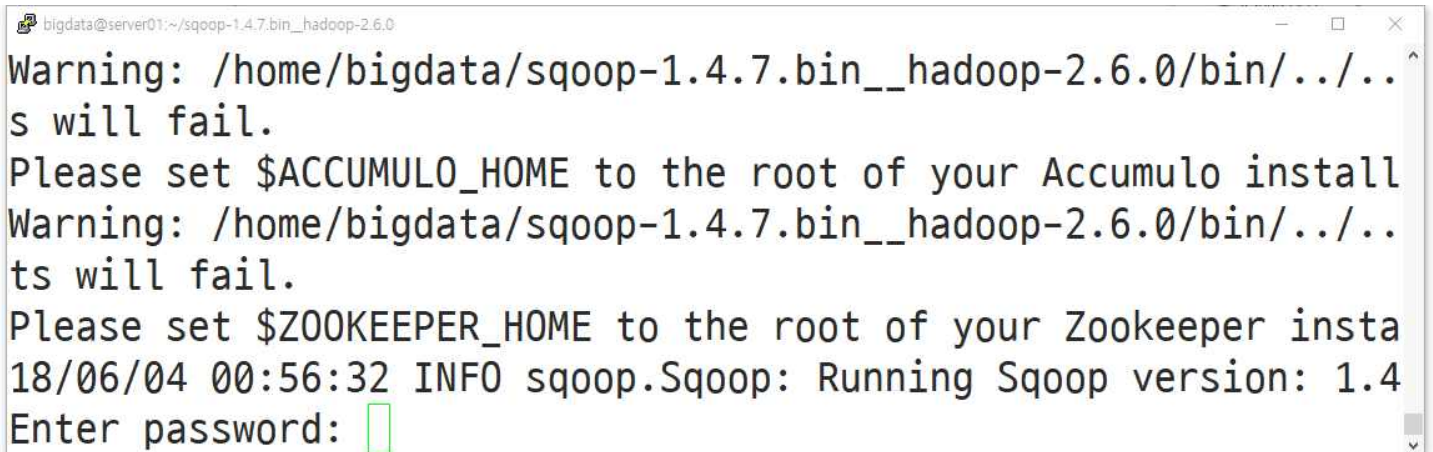
- 커멘드라인에서 임포트가 가능하지만 파일을 만들어서 옵션을 저장한 후 실행

```
[bigdata@server01 sqoop-1.4.7.bin__hadoop-2.6.0]$ vi books_import.sh
```

```
--username
test
--P
--connect
jdbc:oracle:thin:@192.168.0.103:1521:xe
--query
"SELECT no, title, author, publish_date FROM books $CONDITIONS"
--target-dir
books
--split-by
NO
--m
1
```

- books_import.sh를 실행

```
[bigdata@server01 sqoop-1.4.7.bin__hadoop-2.6.0]$ ./bin/sqoop import --options-file books_import.sh
```



```
bigdata@server01:~/sqoop-1.4.7.bin__hadoop-2.6.0
Warning: /home/bigdata/sqoop-1.4.7.bin__hadoop-2.6.0/bin/../../
s will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo install
Warning: /home/bigdata/sqoop-1.4.7.bin__hadoop-2.6.0/bin/../../
ts will fail.
Please set $ZOOKEEPER_HOME to the root of your Zookeeper insta
18/06/04 00:56:32 INFO sqoop.Sqoop: Running Sqoop version: 1.4
Enter password: 
```

- 비밀번호 입력

- 수행 완료

```
bigdata@server01:~/sqoop-1.4.7/bin_hadoop-2.6.0
Total time spent by all maps in occupied slots (ms)=6061
Total time spent by all reduces in occupied slots (ms)=0
Total time spent by all map tasks (ms)=6061
Total vcore-milliseconds taken by all map tasks=6061
Total megabyte-milliseconds taken by all map tasks=6206464
Map-Reduce Framework
  Map input records=15
  Map output records=15
  Input split bytes=87
  Spilled Records=0
  Failed Shuffles=0
  Merged Map outputs=0
  GC time elapsed (ms)=126
  CPU time spent (ms)=1560
  Physical memory (bytes) snapshot=145969152
  Virtual memory (bytes) snapshot=2090889216
  Total committed heap usage (bytes)=18759680
File Input Format Counters
  Bytes Read=0
File Output Format Counters
  Bytes Written=582
18/06/04 00:57:57 INFO mapreduce.ImportJobBase: Transferred 582 bytes in 25.2755 seconds (23.0262 bytes/sec)
18/06/04 00:57:57 INFO mapreduce.ImportJobBase: Retrieved 15 records.
[bigdata@server01 sqoop-1.4.7/bin_hadoop-2.6.0]$
```

- 경로 확인

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls
```

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls
Found 2 items
drwxr-xr-x - bigdata supergroup          0 2018-06-04 00:57 books
drwxr-xr-x - bigdata supergroup          0 2018-05-31 03:17 conf
```

- 실제 내용 확인

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -cat books/part-m-00000
```

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -cat books/part-m-00000
2,인간관계론,데일카네기,2006-12-21 00:00:00.0
3,부러진 용골,요네자와 호노부,2012-05-25 00:00:00.0
4,정의는 무엇인가,마이클 샌델,2010-05-26 00:00:00.0
5,오체불만족,오토타케 히로타다,2001-03-31 00:00:00.0
6,반지의 제왕,J.R.R. 톨킨,1954-07-29 00:00:00.0
7,제3인류,베르나르 베르베르,2012-10-02 00:00:00.0
8,상록수,심훈,2005-06-25 00:00:00.0
9,철강왕카네기,카네기,1993-07-11 00:00:00.0
10,파리의 아파트,기욤 뮈소,2017-12-05 00:00:00.0
11,나무,베르나르 베르베르,2013-05-30 00:00:00.0
12,나의 라임 오렌지 나무,조제 마우루 지 바스콘셀루스,2001-04-03 00:00:00.0
13,원피스,오다 에이이치로,1999-01-31 00:00:00.0
14,신비한 동물사전,J. K. 롤링,2018-01-25 00:00:00.0
15,회랑정 살인사건,히가시노 게이고,2016-10-18 00:00:00.0
16,모모,미하엘 엔데,1999-02-09 00:00:00.0
```

8) 데이터 export 실행

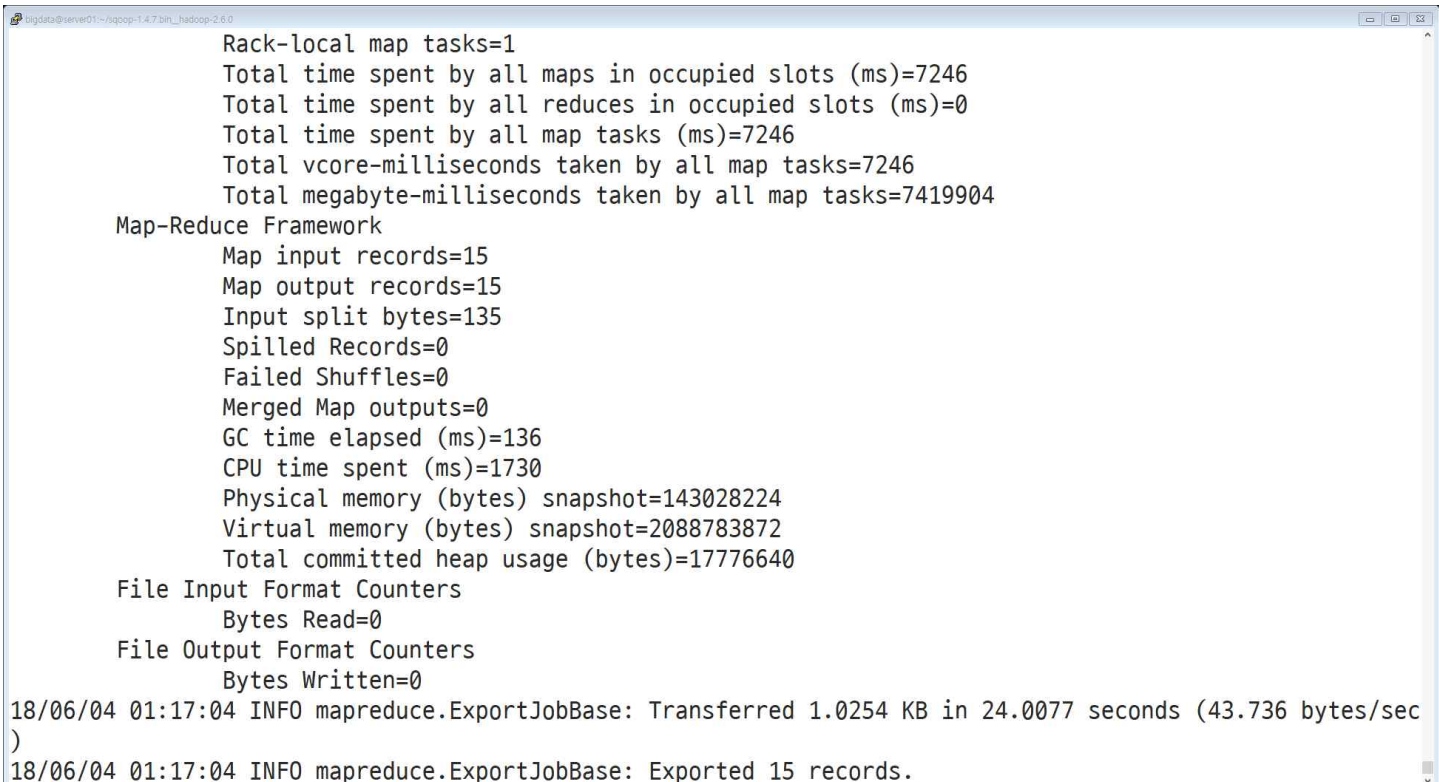
- 커멘드라인에서 익스포트가 가능하지만 파일을 만들어서 옵션을 저장한 후 실행

```
[bigdata@server01 sqoop-1.4.7.bin__hadoop-2.6.0]$ vi books_export.sh
```

```
--username  
test  
-P  
--connect  
jdbc:oracle:thin:@192.168.0.103:1521:xe  
--table  
"BOOKS"  
--export-dir  
books  
-m  
1  
--direct
```

- books_export.sh 파일을 수행

```
[bigdata@server01 sqoop-1.4.7.bin__hadoop-2.6.0]$ ./bin/sqoop export --options-file books_export.sh
```



```
bigdata@server01:~/sqoop-1.4.7/bin_hadoop-2.6.0
Rack-local map tasks=1
Total time spent by all maps in occupied slots (ms)=7246
Total time spent by all reduces in occupied slots (ms)=0
Total time spent by all map tasks (ms)=7246
Total vcore-milliseconds taken by all map tasks=7246
Total megabyte-milliseconds taken by all map tasks=7419904
Map-Reduce Framework
  Map input records=15
  Map output records=15
  Input split bytes=135
  Spilled Records=0
  Failed Shuffles=0
  Merged Map outputs=0
  GC time elapsed (ms)=136
  CPU time spent (ms)=1730
  Physical memory (bytes) snapshot=143028224
  Virtual memory (bytes) snapshot=2088783872
  Total committed heap usage (bytes)=17776640
File Input Format Counters
  Bytes Read=0
File Output Format Counters
  Bytes Written=0
18/06/04 01:17:04 INFO mapreduce.ExportJobBase: Transferred 1.0254 KB in 24.0077 seconds (43.736 bytes/sec)
18/06/04 01:17:04 INFO mapreduce.ExportJobBase: Exported 15 records.
```