

## ■ 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	예 시	활 용	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

## 데이터 웨어하우스와 트랜잭션 데이터베이스 비교

	데이터 웨어하우스	트랜잭션 데이터베이스
적합한 워크로드	분석, 빅 데이터	트랜잭션 처리
작업 유형	I/O를 최소화하고 데이터 처리량을 극대화할 수 있도록 배치 처리된 쓰기 작업 및 대용량 데이터 읽기에 최적화됨	트랜잭션 처리량을 극대화할 수 있도록 지속적인 쓰기 작업과 다수의 소규모 읽기 작업에 최적화됨
데이터 정규화	스타 스키마와 눈송이 스키마와 같이 비정규화된 스키마 사용	높은 트랜잭션 처리량 요구 사항에 좀 더 적합한 고도로 정규화된 스키마 사용
스토리지	열 형식 또는 다른 특별한 스토리지 필요	물리적 블록에 전체 행을 저장하는 행 지향 데이터베이스

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

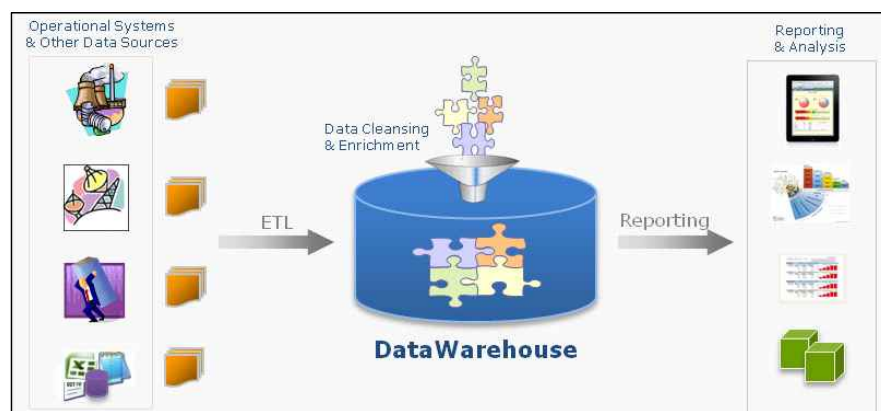
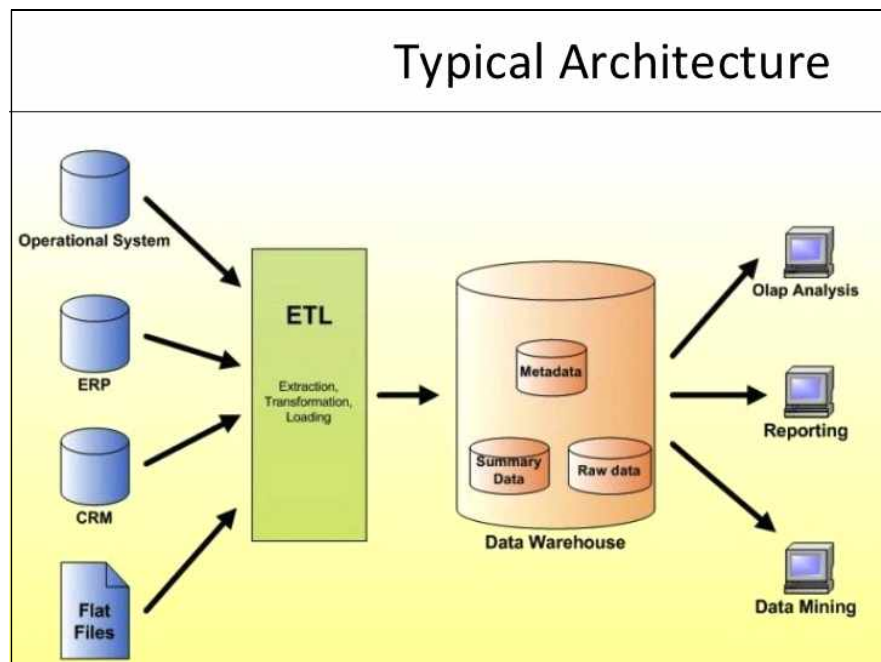
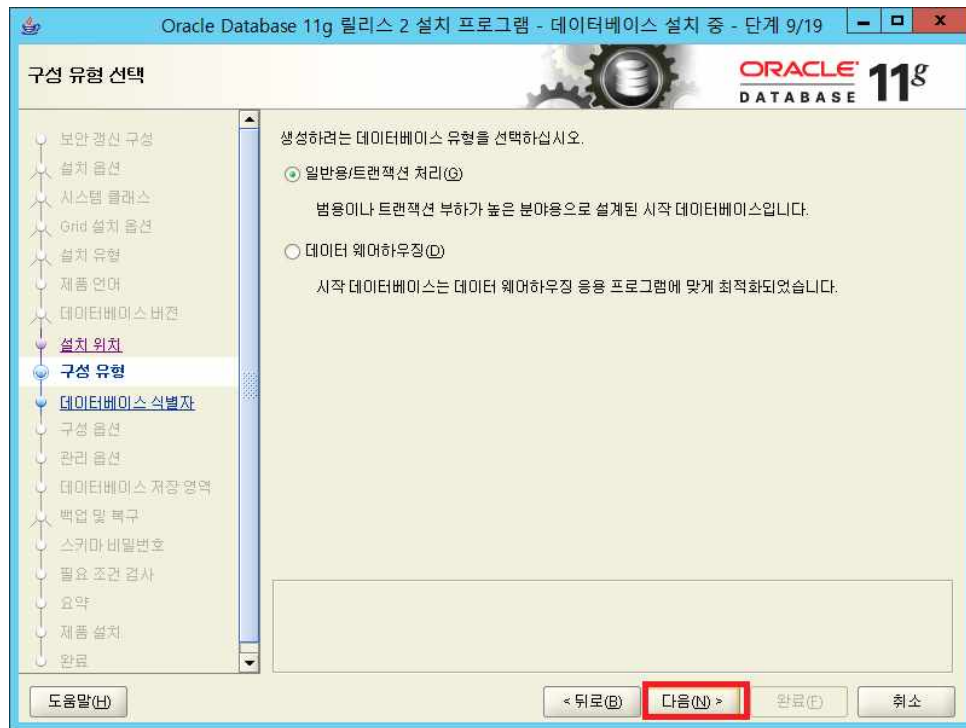
### crawling이란?

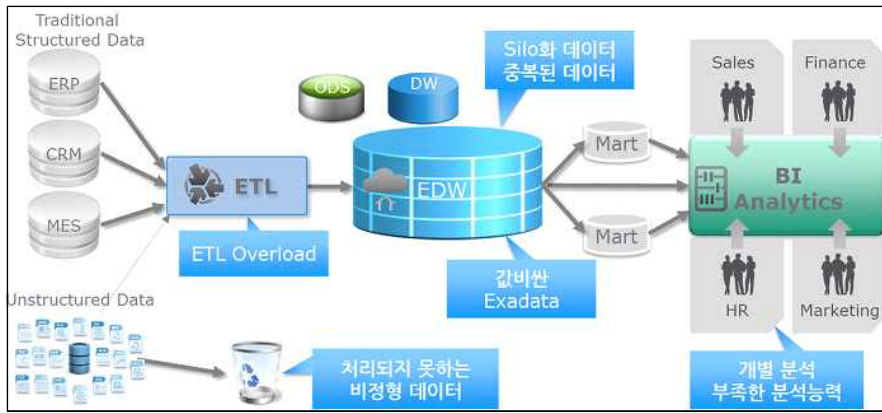
웹 페이지를 그대로 가져와서 거기서 데이터를 추출해 내는 행위

- 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





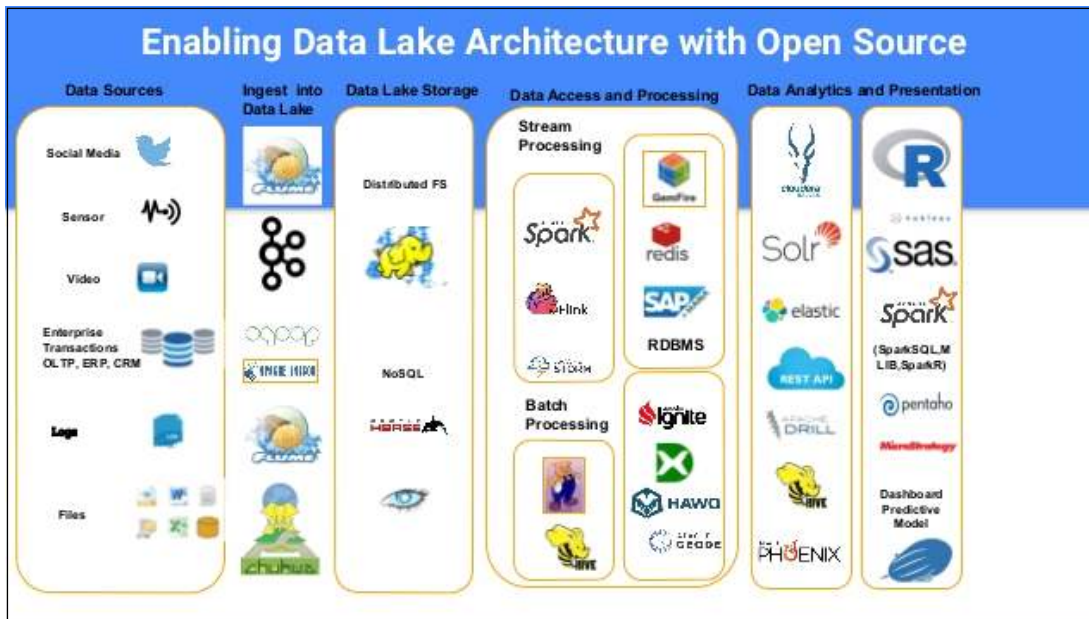
- 기존의 데이터웨어하우스는 RDBMS처럼 사전에 정의된 스키마(테이블)로 데이터를 변환하는 과정이 반드시 필요
- 이 과정을 ETL(Extract : 추출 / Transform : 변환 / Load : 적재)라고 함
- 이 ETL 개발단계를 거쳐 최종 애플리케이션에서 데이터를 활용하기까지 많은 시간과 노력, 비용이 소모되었으며, 특히 데이터 사용방법을 미리 결정해야 했음
- 만약 개발 초기단계에서 불필요하다고 판단했던 데이터는 ETL 과정에서 처리되지 못하고 버려지는데, 나중에 그 데이터가 필요한 경우가 생기면 큰일이 나는 경우가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

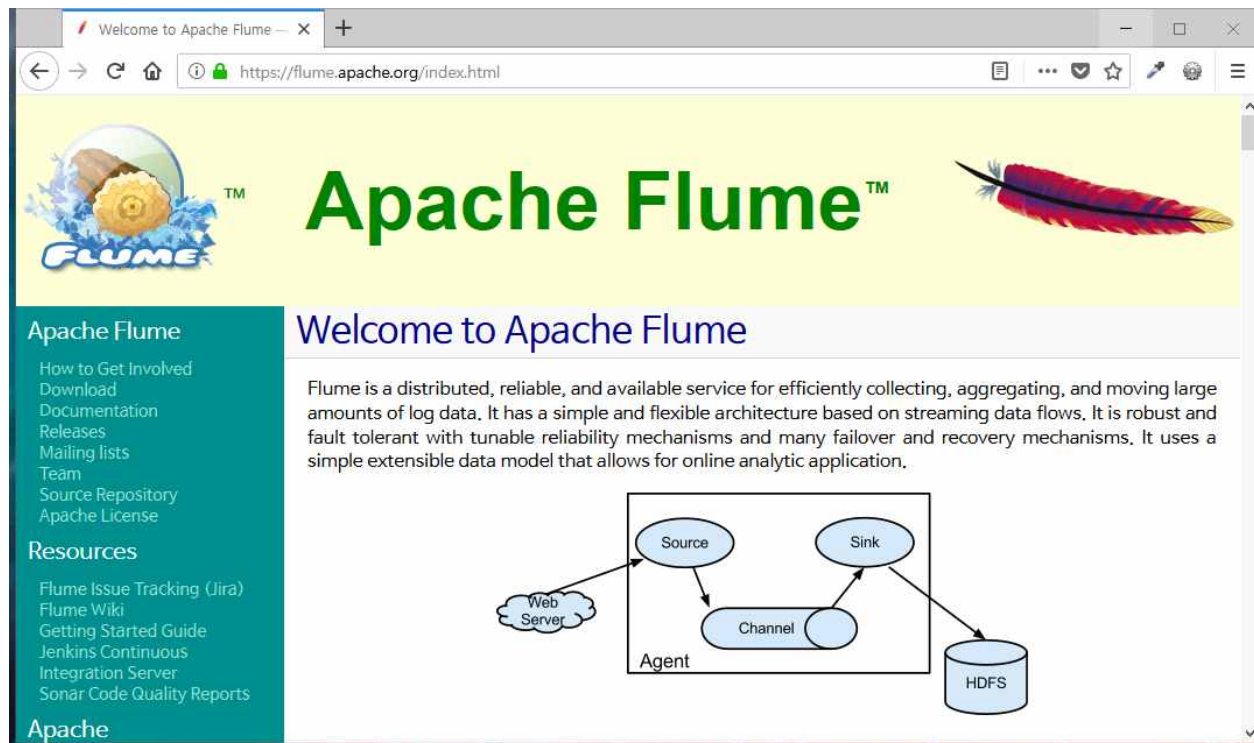
2) 모든 데이터를 공유할 수 있음

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



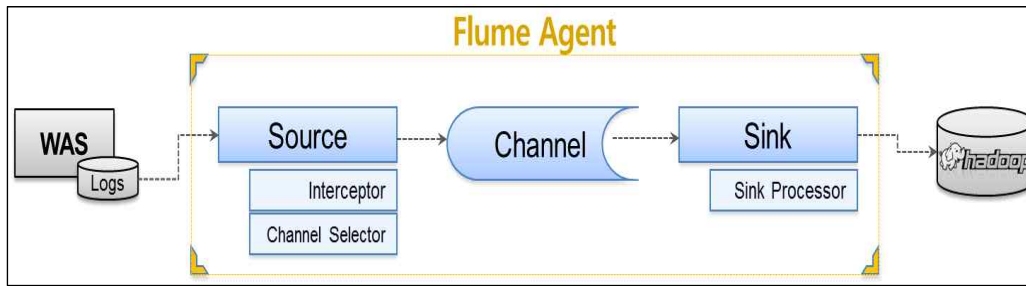
## ■ Apache Flume(아파치 플럼)



- <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
- Interceptor 종류
  - ① Timestamp: 이벤트 헤더에 현재 시간 값 추가
  - ② Static Interceptor: 이벤트 헤더에 지정한 값 추가
  - ③ Regex Filtering Interceptor: 정규표현식에 일치하는지에 따라 이벤트를 버릴지 결정
  - ④ Channel Selector: Source가 받은 이벤트를 전달할 채널을 선택
  - ⑤ Host: 에이전트가 운영중인 호스트이름이나 IP 주소를 헤더에 추가
  - ⑥ UUID: 고유의 식별자 제공
  - ⑦ Search and Replace: 자바 정규식 통해 문자열 기반의 검색 및 교체 기능 제공

### 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 프로세스를 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
  1. 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

- server01에 bigdata 계정으로 접속 후 bigdata 홈디렉터리에 flume 파일 다운

```
$ wget http://mirror.navercorp.com/apache/flume/1.8.0/apache-flume-1.8.0-bin.tar.gz
```

- 다운 받은 'apache-flume-1.8.0-bin.tar.gz' 파일 압축 풀기

```
$ tar xvfz apache-flume-1.8.0-bin.tar.gz
```

- server01 세션을 하나 더 열어서 root로 로그인 후 /etc/profile 열기

```
# vi /etc/profile
```

- /etc/profile에 아래와 같이 플룸 바이너리 경로를 입력후 저장

```
export FLUME_HOME=/home/bigdata/apache-flume-1.8.0-bin  
export PATH=$PATH:$FLUME_HOME/bin:$JAVA_HOME/bin
```

- bigdata 계정에서 hadoop-env.sh에 추가

```
export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar:${HADOOP_CLASSPATH}
```

- source로 /etc/profile 적용

```
# source /etc/profile
```

- bigdata 계정으로 접속한 server01 세션을 열고 플룸의 conf 디렉터리로 이동

```
$ cd /home/bigdata/apache-flume-1.8.0-bin/conf
```

- flume-env.sh.template 파일을 flume-env.sh로 복사

```
$ cp flume-env.sh.template flume-env.sh
```

- flume-env.sh 파일을 열어 아래와 같이 자바 힙 메모리 설정

```
export JAVA_OPTS= Xms100m Xmx2000m
```

```
# Give Flume more memory and pre-allocate, enable GC logging
export JAVA_OPTS= Xms100m Xmx2000m
```

- source로 flume-env.sh 파일 적용

```
$ source flume-env.sh
```

## ■ 플럼 수집 기능 구현

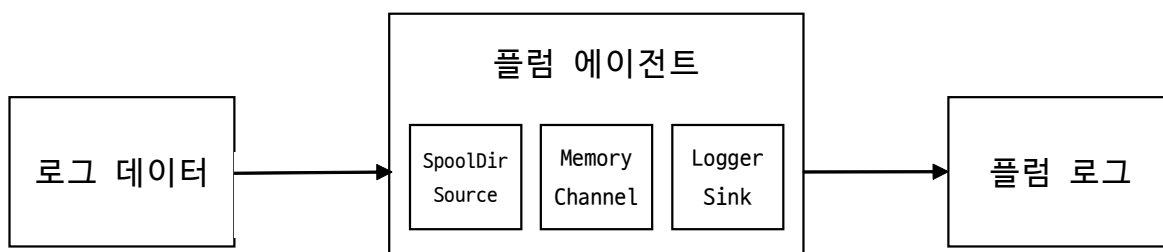
- HDFS로 적재하기 전, 플럼의 수집 기능을 테스트
- flume의 conf 디렉터리에서 **test.conf** 파일 생성

```
[bigdata@server01 conf]$ vi test.conf
```

- test.conf에 아래의 Agent 구성 입력

### ○ 플럼 Agent 구성 파일 설명

- Agent 이름: Test\_Agent
- Test\_Agent 구성: Source-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/bigdata/working/jbm-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/bigdata/working/jbm-batch-log” 경로에서 생성되는 로그 파일 수집
- deletePolicy: 수집이 시작되면 수집하는 파일이 디렉터리(jbm-batch-log)에서 즉각 제거됨

#### - 경로 생성

```
[bigdata@server01 ~]$ mkdir working
[bigdata@server01 ~]$ mkdir working/jbm-batch-log
```

- batchSize 설정값 만큼 읽어서 Channel에 전송

```
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 Sink:** 테스트용으로 수집한 데이터를 플럼의 로그에 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

- 작성후 플럼의 홈디렉터리(apache-flume-1.8.0-bin)로 이동

```
[bigdata@server01 conf]$ cd ..
```

- Test\_Agent 실행

```
flume-ng agent -c conf -f conf파일 -n Agent이름
```

```
[bigdata@server01 apache-flume-1.8.0-bin]$ ./bin/flume-ng agent -c conf -f
conf/test.conf -n Test Agent
```

```
2018-06-11 01:27:59,891 (lifecycleSupervisor-1-4) [INFO - org.apache.flume.source.SpoolDirectorySource.start(SpoolDirectorySource.java:83)] SpoolD
irectorySource source starting with directory: /home/bigdata/working/jbm-batch-log
2018-06-11 01:27:59,989 (lifecycleSupervisor-1-4) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.register(MonitoredCounterGroup.ja
va:119)] Monitored counter group for type: SOURCE, name: TestSource_SpoolSource: Successfully registered new MBean.
2018-06-11 01:27:59,989 (lifecycleSupervisor-1-4) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.start(MonitoredCounterGroup.java:
95)] Component type: SOURCE, name: TestSource_SpoolSource started
```

- FileZilla를 통해 /home/bigdata/working/jbm-batch-log에 파일 업로드

- 파일 업로드와 동시에 Flume 수집기가 실행됨

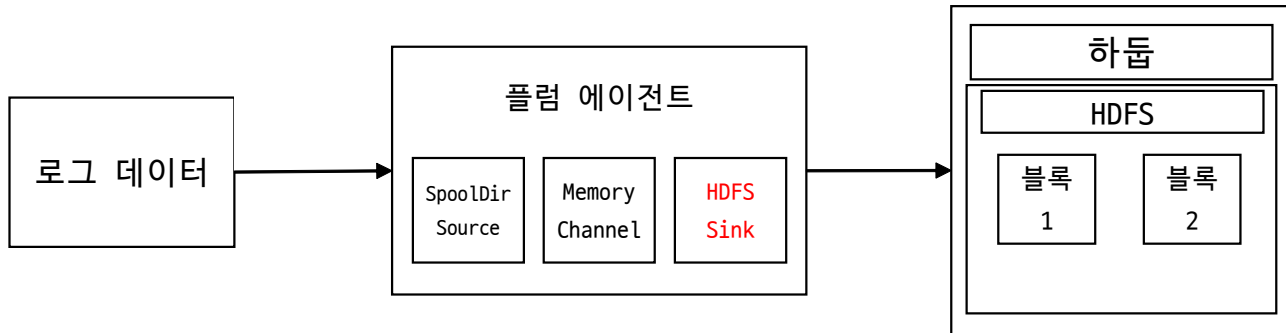
```
2018-06-11 01:42:30,866 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:95)]
Event: { headers:{} body: 22 32 30 31 38 30 32 32 38 22 2C 22 EC 88 98 22 "20180228","..." }
2018-06-11 01:42:30,866 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:95)]
Event: { headers:{} body: 22 32 30 31 38 30 32 32 38 22 2C 22 EC 88 98 22 "20180228","..." }
2018-06-11 01:42:30,866 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:95)]
Event: { headers:{} body: 22 32 30 31 38 30 32 32 38 22 2C 22 EC 88 98 22 "20180228","..." }
2018-06-11 01:42:30,866 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:95)]
Event: { headers:{} body: 22 32 30 31 38 30 32 32 38 22 2C 22 EC 88 98 22 "20180228","..." }
```

- 플럼의 이벤트가 header와 body로 구성됨을 확인
- 데이터의 이동을 바로 확인 가능

## ■ Agent 이름: JBM\_Agent

JBM\_Agent 구성: Source-Interceptor-Channel-Sink

### 1) HDFS Sink로 HDFS에 데이터를 적재



- flume의 conf 디렉터리에서 hdfs.conf 파일 생성

```
[bigdata@server01 conf]$ vi hdfs.conf
```

- hdfs.conf에 아래와 같이 입력

```
JBM_Agent.sources = JBM_SpoolSource
JBM_Agent.channels = JBM_MemChannel
JBM_Agent.sinks = JBM_HdfsSink
```

- HdfsSink로 연결

```
JBM_Agent.sources.JBM_SpoolSource.interceptors = timeInterceptor
JBM_Agent.sources.JBM_SpoolSource.interceptors.timeInterceptor.type = timestamp
```

- **Timestamp Interceptor**
- 플럼의 헤더에 현재 타임 스탬프가 설정되어 필요하면 헤더로부터 타임스탬프값 가져와 활용
- timeInterceptor라는 이름을 붙임

```
JBM_Agent.sources.JBM_SpoolSource.type = spooldir
JBM_Agent.sources.JBM_SpoolSource.spoolDir =
/home/bigdata/working/jbm-batch-log
JBM_Agent.sources.JBM_SpoolSource.deletePolicy = immediate
JBM_Agent.sources.JBM_SpoolSource.batchSize = 1000
```



- JBM\_Agent의 Source 설정
- spooldir: 파일이 생성되면 그 파일의 내용을 수집
- “/home/bigdata/working/jbm-batch-log” 경로에서 생성되는 로그 파일 수집
- batchSize 설정값 만큼 읽어서 Channel에 전송

```
JBM_Agent.channels.JBM_MemChannel.type = memory
JBM_Agent.channels.JBM_MemChannel.capacity = 100000
JBM_Agent.channels.JBM_MemChannel.transactionCapacity = 10000
```

- JBM\_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

```
JBM_Agent.sinks.JBM_HdfsSink.type = hdfs
JBM_Agent.sinks.JBM_HdfsSink.hdfs.path=/user/flume/%y/%m/%d
JBM_Agent.sinks.JBM_HdfsSink.hdfs.filePrefix=event
JBM_Agent.sinks.JBM_HdfsSink.hdfs.fileSuffix=.log
JBM_Agent.sinks.JBM_HdfsSink.hdfs.inUsePrefix=_
JBM_Agent.sinks.JBM_HdfsSink.hdfs.fileType=DataStream
JBM_Agent.sinks.JBM_HdfsSink.hdfs.writeFormat = Text
JBM_Agent.sinks.JBM_HdfsSink.hdfs.batchSize = 10000
JBM_Agent.sinks.JBM_HdfsSink.hdfs.rollInterval = 0
JBM_Agent.sinks.JBM_HdfsSink.hdfs.rollCount = 0
JBM_Agent.sinks.JBM_HdfsSink.hdfs.idleTimeout = 100
JBM_Agent.sinks.JBM_HdfsSink.hdfs.callTimeout = 600000
JBM_Agent.sinks.JBM_HdfsSink.hdfs.rollSize = 67108864
JBM_Agent.sinks.JBM_HdfsSink.hdfs.threadPoolSize = 10
```

- **HDFS sink** 상세 설정값
- hdfs.path: 앞에 정의한 Interceptor의 값을 이용해 HDFS의 경로를 동적으로 나누는 “path” 설정
- hdfs.filePrefix : HDFS에 생성된 파일의 접두사에 붙을 이름
- hdfs.fileSuffix : 파일에 추가되는 접미사
- inUsePrefix : flume이 현재 쓰고 있는 임시 파일에 사용되는 접두사
- hdfs.rollSize : 파일크기(64MB) 정의

```
JBM_Agent.sources.JBM_SpoolSource.channels = JBM_MemChannel
JBM_Agent.sinks.JBM_HdfsSink.channel = JBM_MemChannel
```

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

- 작성후 플럼의 홈디렉터리(apache-flume-1.8.0-bin)로 이동

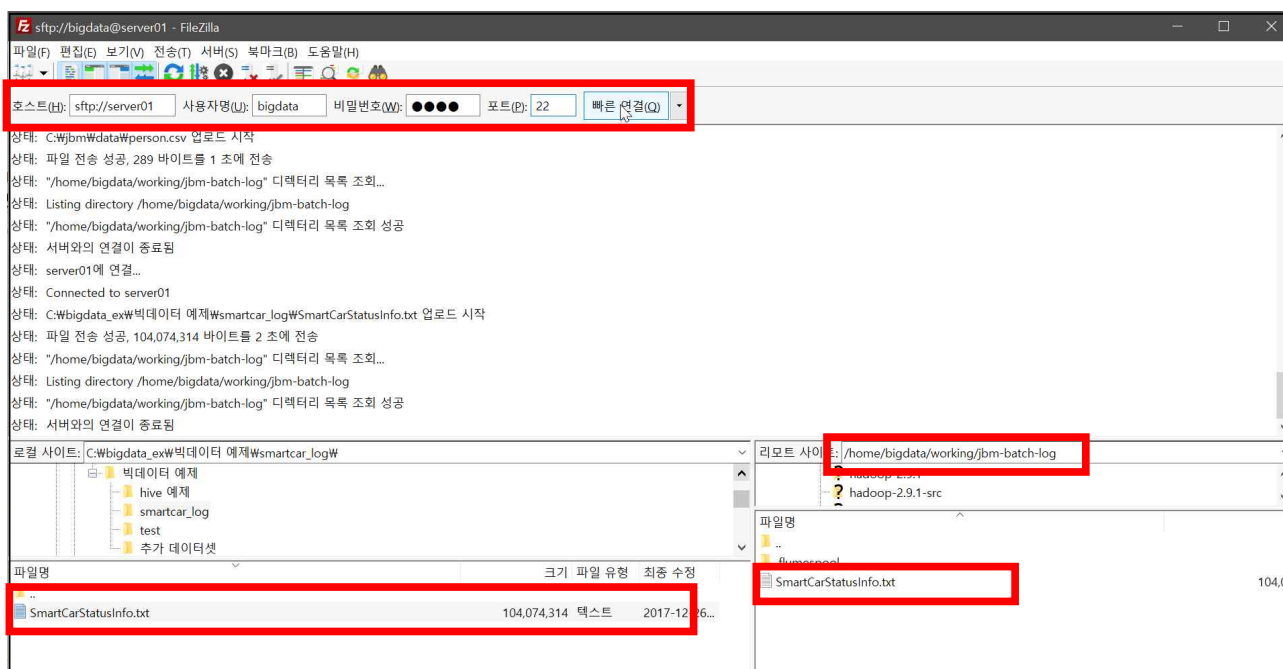
```
[bigdata@server01 conf]$ cd ..
```

- JBM\_Agent 실행

```
[bigdata@server01 apache-flume-1.8.0-bin]$ ./bin/flume-ng agent -c conf -f
conf/hdfs.conf -n JBM_Agent -Dflume.root.logger=INFO,console
```

- FileZilla를 통해 /home/bigdata/working/jbm-batch-log에 파일 업로드

- 호스트: server01
- 사용자명: bigdata
- 비밀번호: 1111
- 포트: 22



- 파일 업로드와 동시에 플럼 수집기 실행

```

2018-06-11 06:06:37,897 (hdfs-JBMSink_HdfsSink-roll-timer-0) [INFO - org.apache.flume.sink.hdfs.BucketWriter$5.call(BucketWriter.java:456)] Closing idle bucketWriter /user/flume/18/06/11/_event.1528664638791.log.tmp at 1528664797897
2018-06-11 06:06:37,898 (hdfs-JBMSink_HdfsSink-roll-timer-0) [INFO - org.apache.flume.sink.hdfs.BucketWriter.close(BucketWriter.java:393)] Closing /user/flume/18/06/11/_event.1528664638791.log.tmp
2018-06-11 06:06:37,951 (hdfs-JBMSink_HdfsSink-call-runner-8) [INFO - org.apache.flume.sink.hdfs.BucketWriter$8.call(BucketWriter.java:655)] Renaming /user/flume/18/06/11/_event.1528664638791.log.tmp to /user/flume/18/06/11/event.1528664638791.log
2018-06-11 06:06:37,961 (hdfs-JBMSink_HdfsSink-roll-timer-0) [INFO - org.apache.flume.sink.hdfs.HDFSEventSink$1.run(HDFSEventSink.java:382)] Writer callback called.

```

## - HDFS의 /user/flume 에서 파일 확인 가능

```

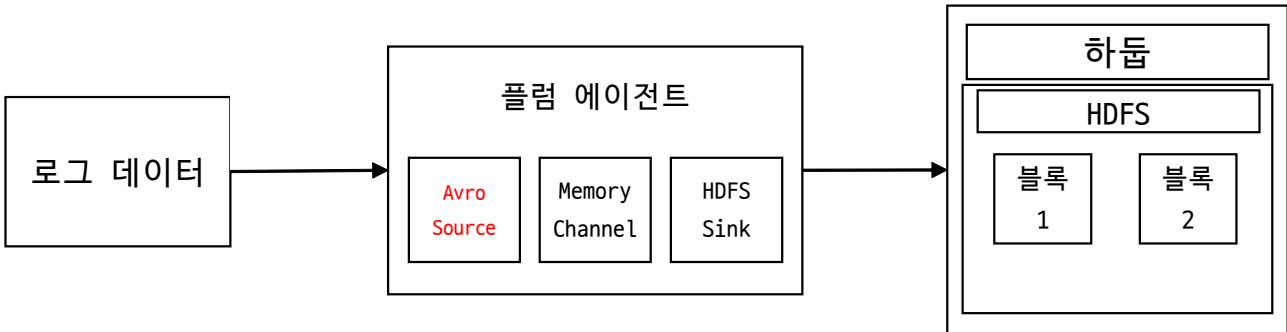
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls -R /user/flume
drwxr-xr-x - bigdata supergroup 0 2018-06-11 05:45 /user/flume/18
drwxr-xr-x - bigdata supergroup 0 2018-06-11 05:45 /user/flume/18/06
drwxr-xr-x - bigdata supergroup 0 2018-06-11 06:10 /user/flume/18/06/11
-rw-r--r-- 3 bigdata supergroup 68531514 2018-06-11 06:04 /user/flume/18/06/11/event.1528664638790.log
-rw-r--r-- 3 bigdata supergroup 35542800 2018-06-11 06:06 /user/flume/18/06/11/event.1528664638791.log

```

## ■ Agent 이름: CAFE\_Agent

CAFE\_Agent 구성: Source-Interceptor-Channel-Sink

1) Avro Source의 RPC 통신방식으로 데이터를 받아 하둡으로 적재



- flume의 conf 디렉터리에서 avro.conf 파일 생성

```

[bigdata@server01 conf]$ vi avro.conf

```

- avro.conf에 아래와 같이 입력

```

CAFE_Agent.sources.JBM_AvroSource.interceptors =
timeInterceptor typeInterceptor

```

```
CAFE_Agent.sources.JBM_AvroSource.interceptors.timeInterceptor.type = timestamp
CAFE_Agent.sources.JBM_AvroSource.interceptors.typeInterceptor.type = static
CAFE_Agent.sources.JBM_AvroSource.interceptors.typeInterceptor.key = logType
CAFE_Agent.sources.JBM_AvroSource.interceptors.typeInterceptor.value =
cafe-batch-log
```

- **Static Interceptor** 설정
- 플럼의 해당 이벤트 내에 사용할 상수값 설정
- “logType”이라는 상수이름을 선언함
- 상수값은 “cafe-batch-log”
- “cafe-batch-log” 값은 HDFS에 적재된 데이터들을 구분 할 수 있게 사용

```
CAFE_Agent.sources = JBM_AvroSource
CAFE_Agent.channels = JBM_MemChannel
CAFE_Agent.sinks = JBM_HdfsSink
```

- AvroSource로 연결

```
CAFE_Agent.sources.JBM_AvroSource.type = avro
CAFE_Agent.sources.JBM_AvroSource.bind = 0.0.0.0
CAFE_Agent.sources.JBM_AvroSource.port = 65111
```

- bind : 수신할 호스트 이름 또는 IP주소를 입력
- port : 바인딩할 포트번호

```
CAFE_Agent.channels.JBM_MemChannel.type = memory
CAFE_Agent.channels.JBM_MemChannel.capacity = 100000
CAFE_Agent.channels.JBM_MemChannel.transactionCapacity = 10000
```

- JBM\_Agent의 Channel 설정

```
CAFE_Agent.sinks.JBM_HdfsSink.type = hdfs
CAFE_Agent.sinks.JBM_HdfsSink.hdfs.path=/user/flume/%y/%m/%d
CAFE_Agent.sinks.JBM_HdfsSink.hdfs.filePrefix= %{logType}
CAFE_Agent.sinks.JBM_HdfsSink.hdfs.fileSuffix=.log
CAFE_Agent.sinks.JBM_HdfsSink.hdfs.inUsePrefix=_
CAFE_Agent.sinks.JBM_HdfsSink.hdfs.fileType=DataStream
CAFE_Agent.sinks.JBM_HdfsSink.hdfs.writeFormat = Text
CAFE_Agent.sinks.JBM_HdfsSink.hdfs.threadPoolSize = 10
```

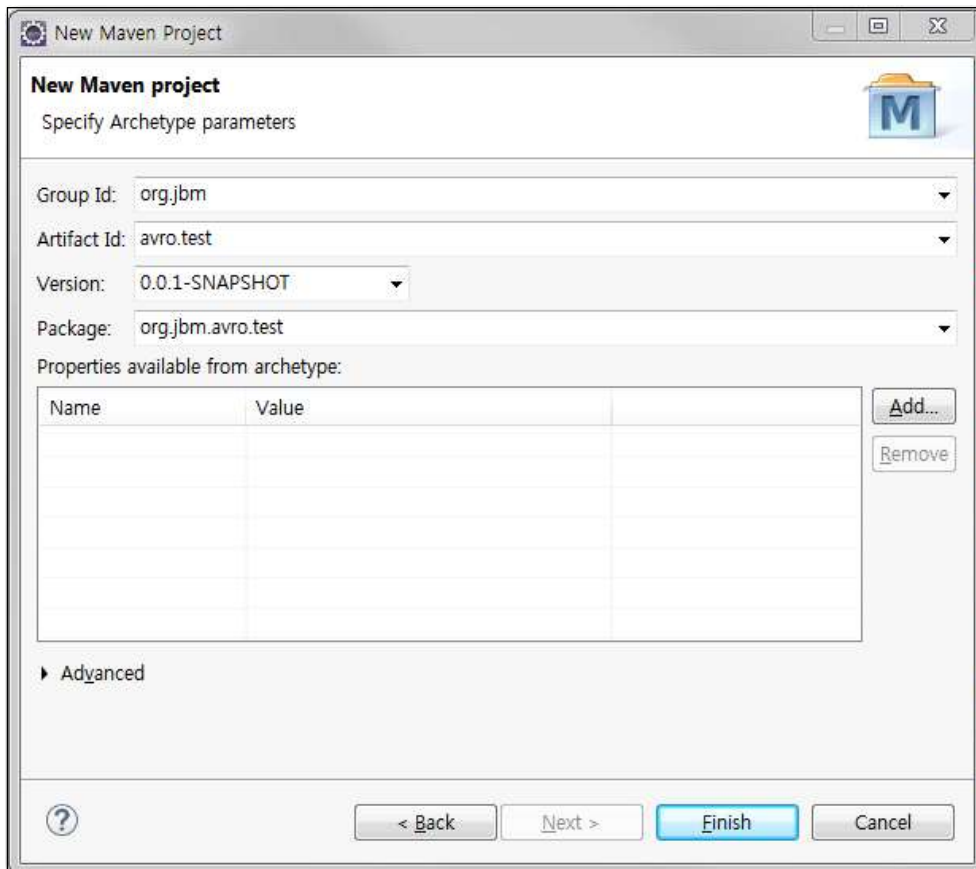
- HDFS sink 상세 설정값
- `%{logType}`: 파일 접두사에 interceptor에서 정의한 명칭이 붙음

```
CAFE_Agent.sources.JBM_AvroSource.channels = JBM_MemChannel  
CAFE_Agent.sinks.JBM_HdfsSink.channel = JBM_MemChannel
```

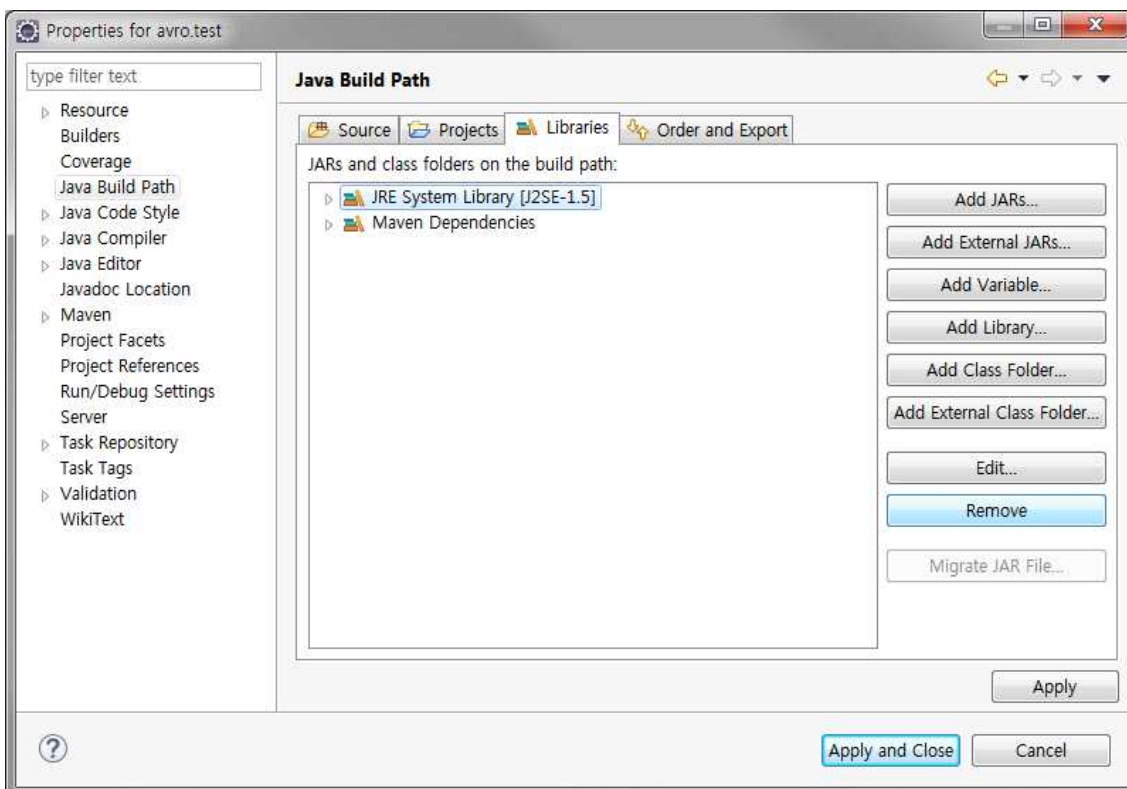
- Avro Source와 Memory Channel을 연결
- HDFS Sink와 Memory Channel을 연결

## ■ Avro Client 개발

### 1) eclipse에서 maven project 만들기

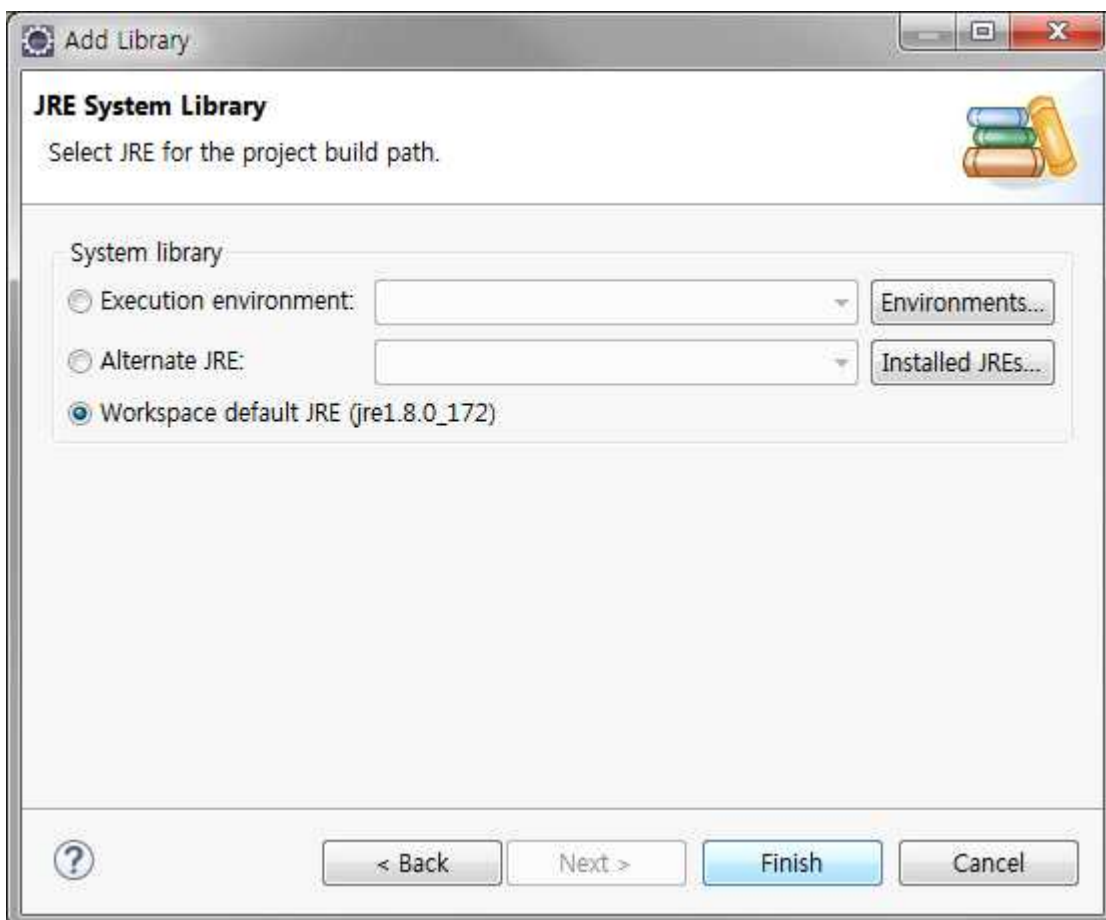
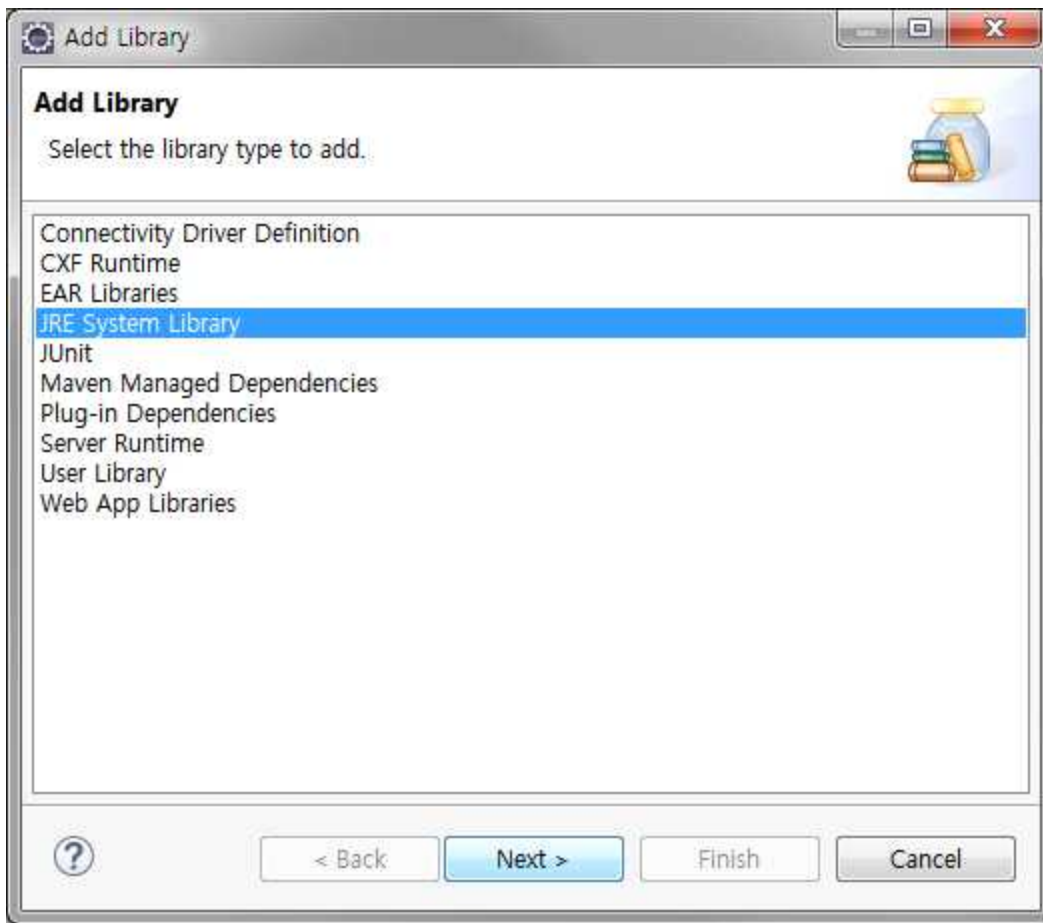


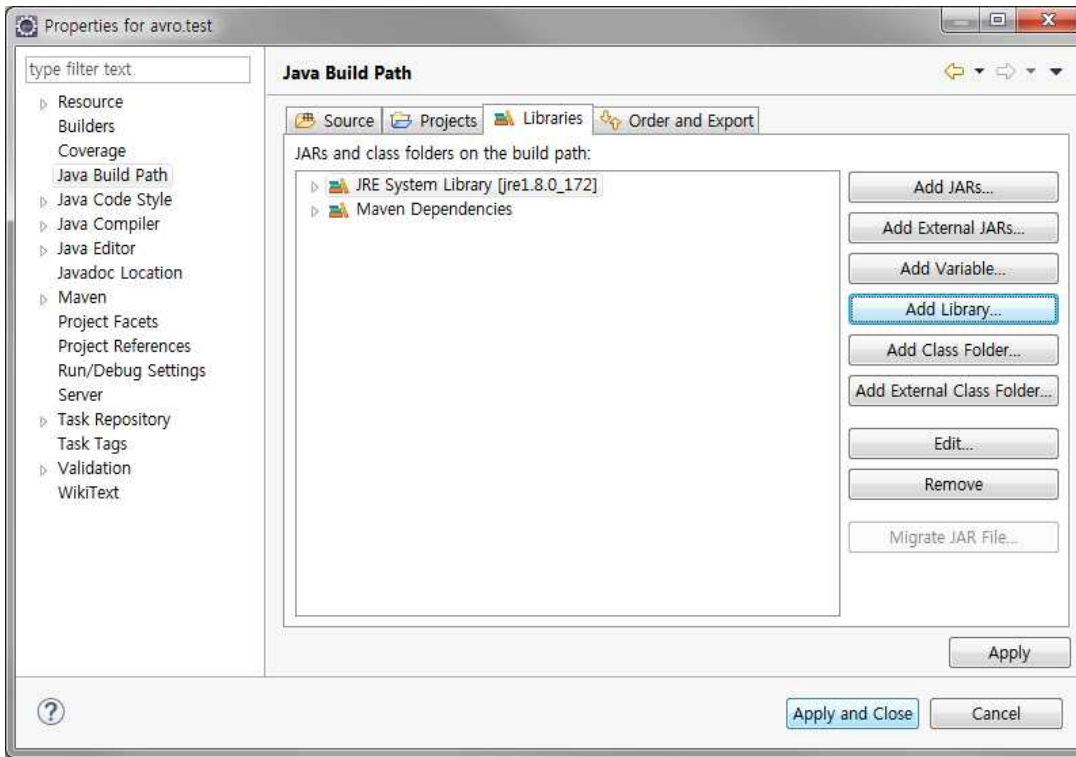
### 2) 자바 1.5버전 JRE 지우기





### 3) 시스템에 존재하는 JRE System Library 등록



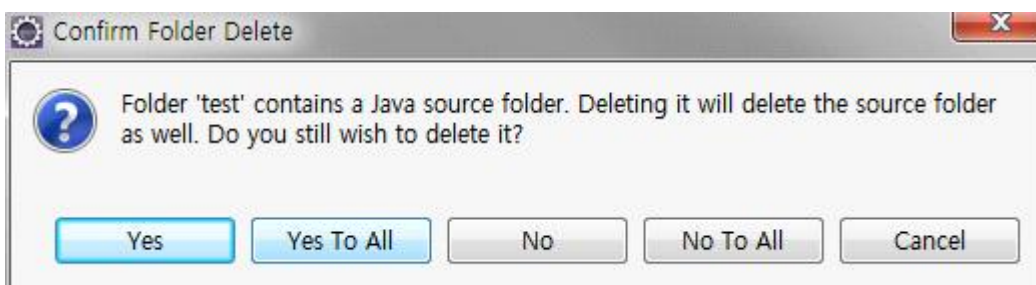
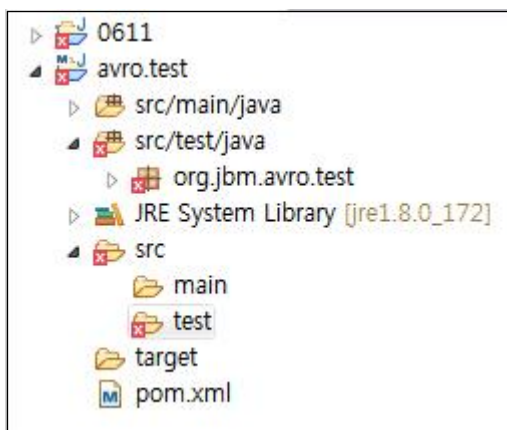


#### 4) JUnit dependency 지우기

```
<dependencies>

</dependencies>
```

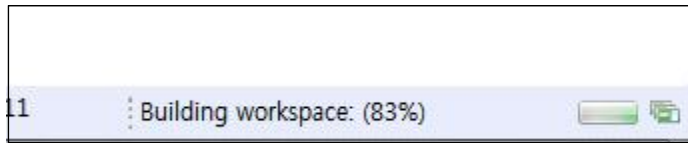
- src의 test폴더도 지우기(Junit용이었음)



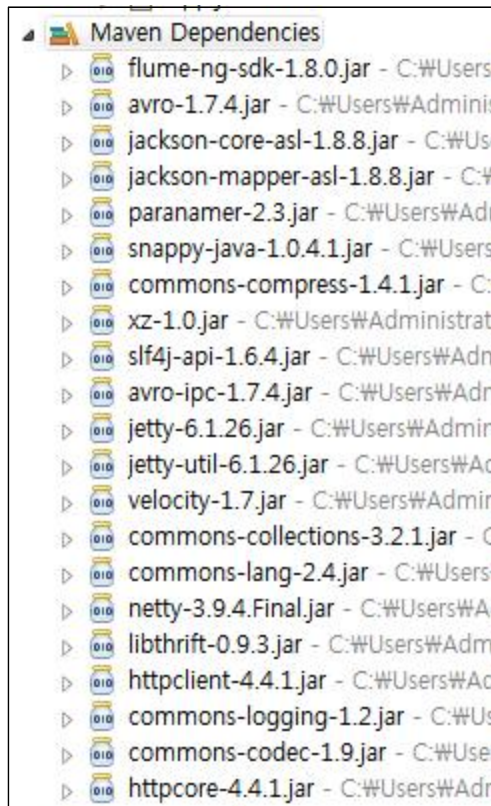
## 5) pom.xml에 의존성 추가

```
<dependencies>
    <dependency>
        <groupId>org.apache.flume</groupId>
        <artifactId>flume-ng-sdk</artifactId>
        <version>1.8.0</version>
    </dependency>
</dependencies>
```

- pom.xml을 저장하면 Maven이 알아서 다운받아서 jar 추가해줌



- 21개의 jar파일이 추가됨



## 6) RpcClientApp.java 만들기

```
package org.jbm.avro.test;

import java.nio.charset.Charset;
import org.apache.flume.Event;
import org.apache.flume.api.RpcClient;
import org.apache.flume.api.RpcClientFactory;
import org.apache.flume.event.EventBuilder;

public class RpcClientApp {

    public static void main(String[] args) {

        RpcClient client =
RpcClientFactory.getDefaultInstance("192.168.56.101", 65111);

        Event event =
EventBuilder.withBody("1,테스터,테스트중입니다.", Charset.forName("UTF-8"));

        try {
            client.append(event);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            client.close();
        } //try~catch~finally

    } //main end

} //App end
```

## 7) flume 에이전트 실행 후 작동

- CAFE\_Agent 실행

```
[bigdata@server01 apache-flume-1.8.0-bin]$ ./bin/flume-ng agent -c conf -f
conf/avro.conf -n CAFE_Agent
```

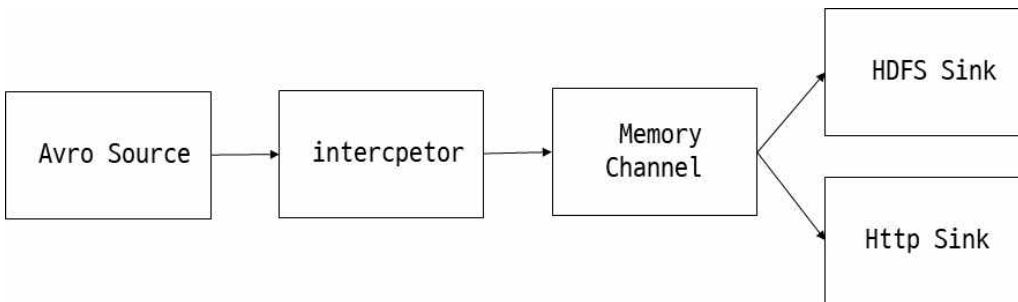
## - hadoop으로 이동 후 파일 확인

```
$ cd /home/bigdata/hadoop-2.9.1
```

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

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls -R /user/flume
drwxr-xr-x - bigdata supergroup 0 2018-06-11 05:45 /user/flume/18
drwxr-xr-x - bigdata supergroup 0 2018-06-11 05:45 /user/flume/18/06
drwxr-xr-x - bigdata supergroup 0 2018-06-11 09:34 /user/flume/18/06/11
-rw-r--r-- 3 bigdata supergroup 1001324 2018-06-11 09:34 /user/flume/18/06/11/cafe-batch-log.1528675569111.log
```

## ■ sink 2개 연결



- Avro Source로 데이터를 받아 HDFS와 Http 두개의 Sink로 데이터 전달

- Agent 명: JBM\_Agent

```
JBM_Agent.sources = JBM_AvroSource
```

```
JBM_Agent.channels = JBM_MemChannel
```

```
JBM_Agent.sinks = JBM_HttpSink JBM_HdfsSink
```

```
JBM_Agent.sources.JBM_AvroSource.type = avro
```

```
JBM_Agent.sources.JBM_AvroSource.bind = 0.0.0.0
```

```
JBM_Agent.sources.JBM_AvroSource.port = 65111
```

```
JBM_Agent.sources.JBM_AvroSource.interceptors = timeInterceptor typeInterceptor
```

```
JBM_Agent.sources.JBM_AvroSource.interceptors.timeInterceptor.type = timestamp
```

```
JBM_Agent.sources.JBM_AvroSource.interceptors.typeInterceptor.type = static
```

```
JBM_Agent.sources.JBM_AvroSource.interceptors.typeInterceptor.key = logType
```

```
JBM_Agent.sources.JBM_AvroSource.interceptors.typeInterceptor.value = cafe-batch-log
```

```
JBM_Agent.channels.JBM_MemChannel.type = memory
```

```
JBM_Agent.channels.JBM_MemChannel.capacity = 100000
```

```
JBM_Agent.channels.JBM_MemChannel.transactionCapacity = 10000
```

```
JBM_Agent.sinks.JBM_HttpSink.type = http
```

```
JBM_Agent.sinks.JBM_HttpSink.endpoint = http://192.168.0.103/test
```

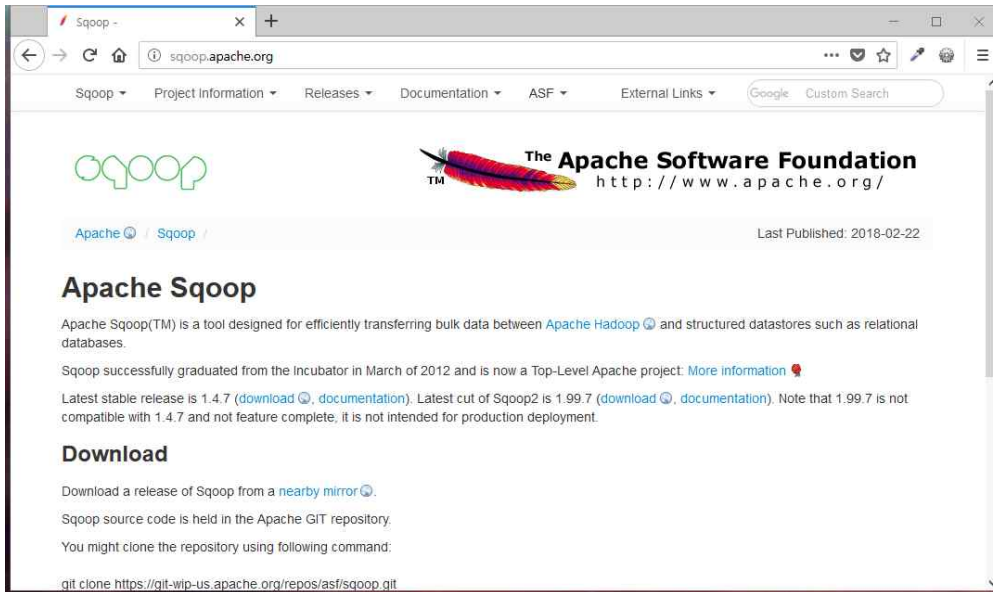
```
JBM_Agent.sinks.JBM_HttpSink.connectTimeout = 2000
JBM_Agent.sinks.JBM_HttpSink.requestTimeout = 2000
JBM_Agent.sinks.JBM_HttpSink.acceptHeader = application/json
JBM_Agent.sinks.JBM_HttpSink.contentTypeHeader = application/json
JBM_Agent.sinks.JBM_HttpSink.defaultBackoff = true
JBM_Agent.sinks.JBM_HttpSink.defaultRollback = true
JBM_Agent.sinks.JBM_HttpSink.defaultIncrementMetrics = false
JBM_Agent.sinks.JBM_HttpSink.backoff.4XX = false
JBM_Agent.sinks.JBM_HttpSink.rollback.4XX = false
JBM_Agent.sinks.JBM_HttpSink.incrementMetrics.4XX = true
JBM_Agent.sinks.JBM_HttpSink.backoff.200 = false
JBM_Agent.sinks.JBM_HttpSink.rollback.200 = false
JBM_Agent.sinks.JBM_HttpSink.incrementMetrics.200 = true
```

```
JBM_Agent.sinks.JBM_HdfsSink.type = hdfs
JBM_Agent.sinks.JBM_HdfsSink.hdfs.path=/user/flume/%y/%m/%d
JBM_Agent.sinks.JBM_HdfsSink.hdfs.filePrefix=event
JBM_Agent.sinks.JBM_HdfsSink.hdfs.fileSuffix=.log
JBM_Agent.sinks.JBM_HdfsSink.hdfs.inUsePrefix=_
JBM_Agent.sinks.JBM_HdfsSink.hdfs.fileType=DataStream
JBM_Agent.sinks.JBM_HdfsSink.hdfs.writeFormat = Text
JBM_Agent.sinks.JBM_HdfsSink.hdfs.batchSize = 10000
JBM_Agent.sinks.JBM_HdfsSink.hdfs.rollInterval = 0
JBM_Agent.sinks.JBM_HdfsSink.hdfs.rollCount = 0
JBM_Agent.sinks.JBM_HdfsSink.hdfs.idleTimeout = 100
JBM_Agent.sinks.JBM_HdfsSink.hdfs.callTimeout = 600000
JBM_Agent.sinks.JBM_HdfsSink.hdfs.rollSize = 67108864
JBM_Agent.sinks.JBM_HdfsSink.hdfs.threadsPoolSize = 10
```

```
JBM_Agent.sources.JBM_AvroSource.channels = JBM_MemChannel
JBM_Agent.sinks.JBM_HttpSink.channel = JBM_MemChannel
JBM_Agent.sinks.JBM_HdfsSink.channel = JBM_MemChannel
```

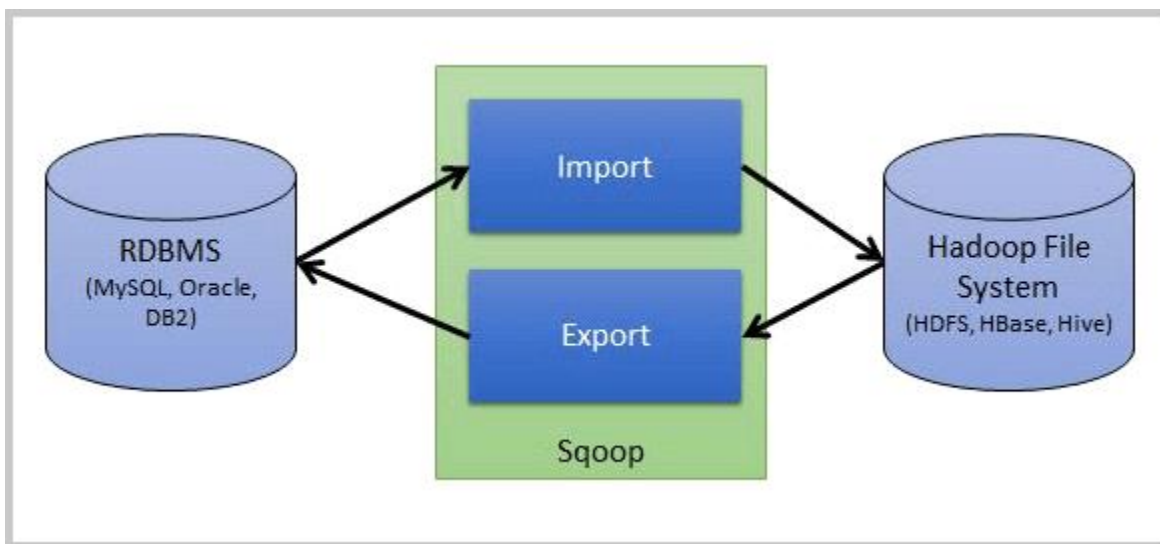


## ■ 아파치 스쿱(Apache Sqoop)



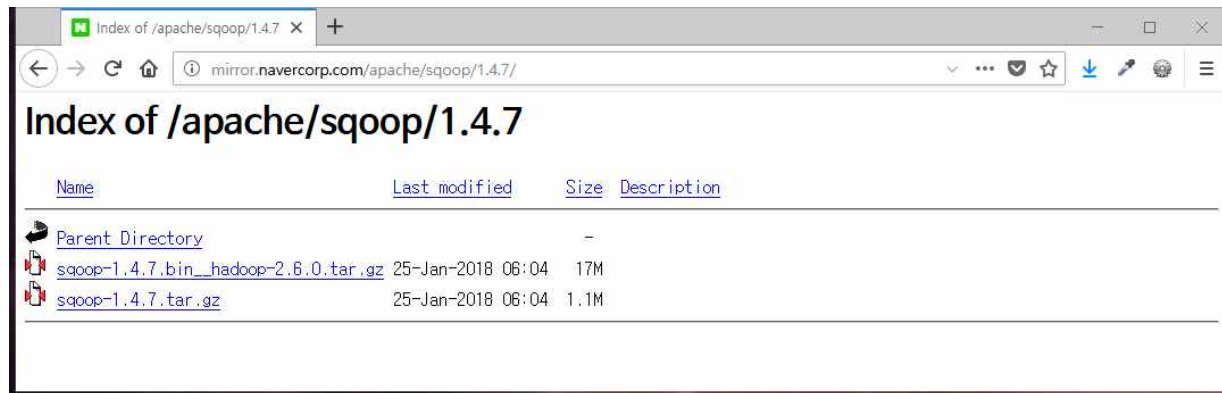
<http://sqoop.apache.org/>

- 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

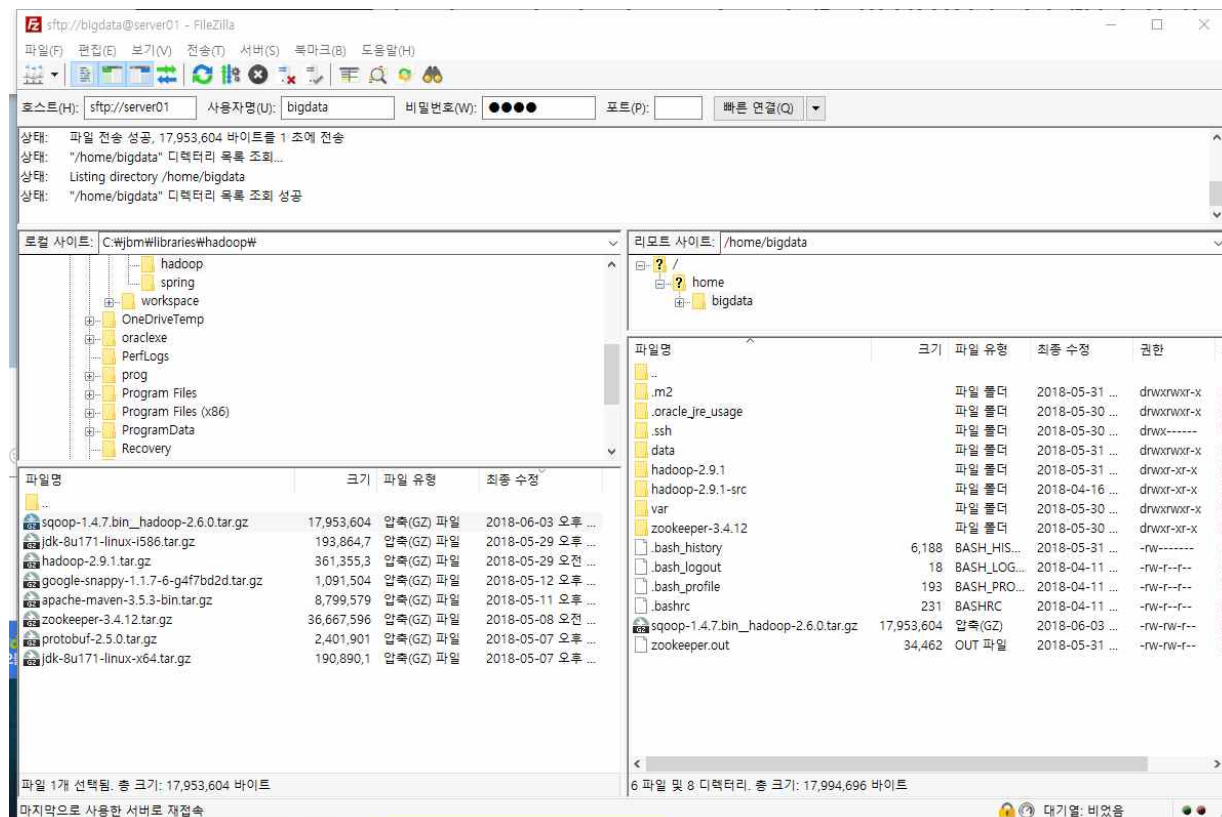
- 1) <http://mirror.navercorp.com/apache/sqoop/1.4.7/>에서  
sqoop-1.4.7.bin\_\_hadoop-2.6.0.tar.gz를 다운로드



- 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

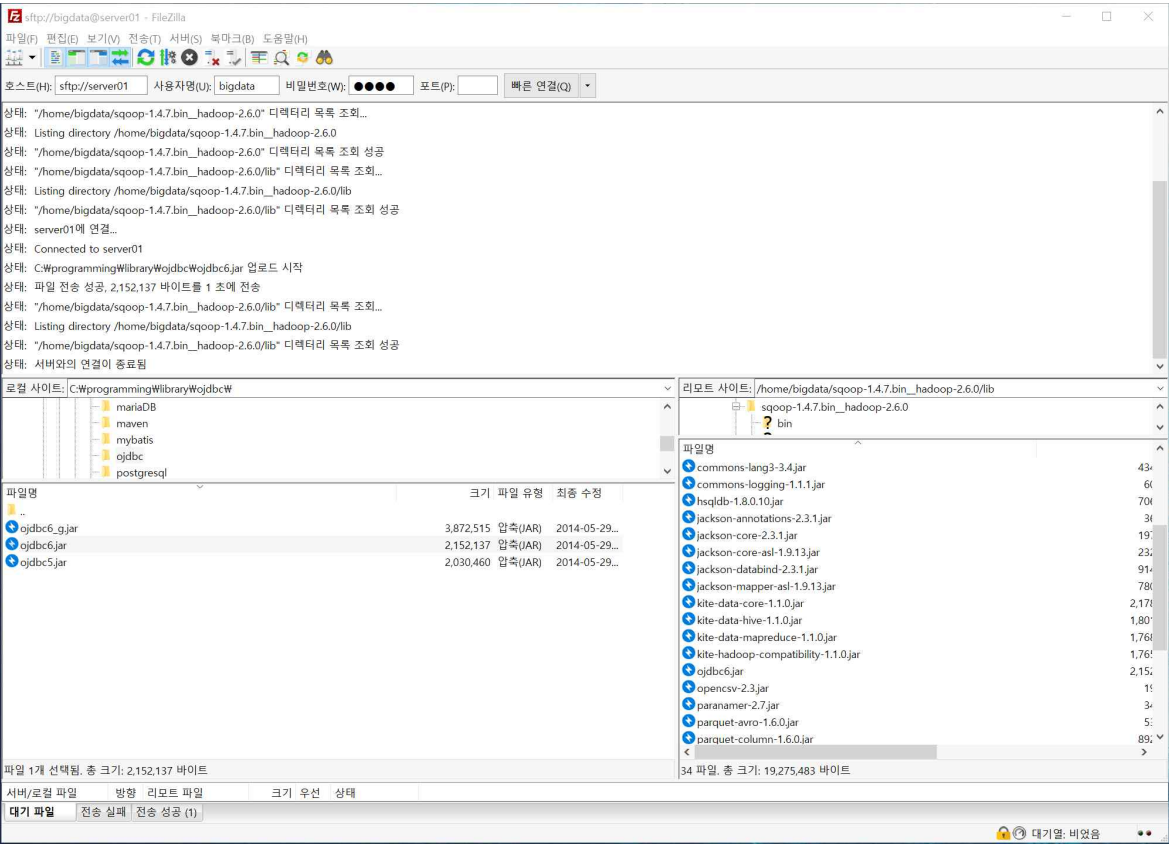


3) 압축 풀기

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

4) FileZilla를 통해 sqoop의 lib디렉터리에 ojdbc.jar 파일 갖다 놓기

```
/home/bigdata/sqoop-1.4.7.bin__hadoop-2.6.0/lib
```



5) 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 ;

```

## 6) 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 뭈소', '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');

```

## 7) 데이터 import / export 옵션들

### - (import / export시) 공통인자

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

## 8) 데이터 import 실행

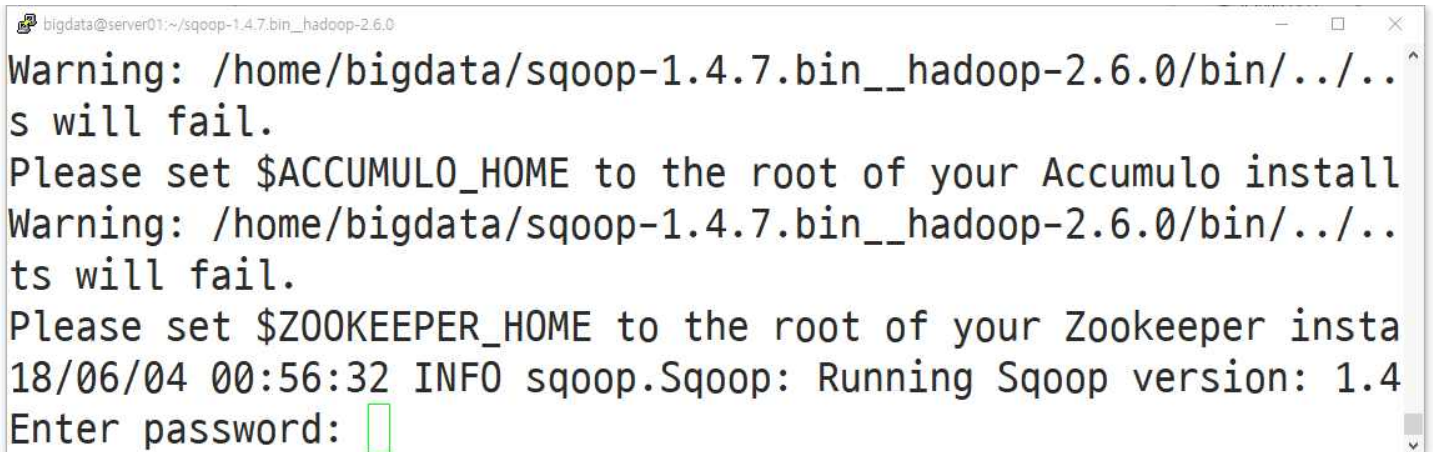
- 커멘드라인에서 임포트가 가능하지만 파일을 만들어서 옵션을 저장한 후 실행

```
[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 WHERE $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 뭈소,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
```

## 9) 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.
```

Tables ▼

ARTICLES  
**BOOKS**  
 BOOKS\_TEST  
 HTMLDB\_PLAN\_TABLE  
 LIKES  
 REPLIES  
 USERS

## BOOKS

Table **Data** Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

[Query](#) [Count Rows](#) [Insert Row](#)

EDIT	NO	TITLE	AUTHOR	PUBLISH_DATE
	4	인간관계론	데일카네기	12/21/2006
	5	부러진 용골	요네자와 호노부	05/25/2012
	6	정의는 무엇인가	마이클 샌델	05/26/2010
	7	오체불만족	오토타케 히로타다	03/31/2001
	8	반지의 제왕	J.R.R. 톨킨	07/29/1954
	9	제3인류	베르나르 베르베르	10/02/2012
	10	상록수	심훈	06/25/2005
	11	철강왕 카네기	카네기	07/11/1993
	12	파리의 아파트	기욤 뮈소	12/05/2017
	13	나무	베르나르 베르베르	05/30/2013
	14	나의 라임 오렌지 나무	조제 마우루 지 바스콘셀루스	04/03/2001
	15	원피스	오다 에이이치로	01/31/1999
	16	신비한 동물사전	J. K. 롤링	01/25/2018
	17	희랑정 살인사건	히가시노 게이고	10/18/2016
	18	모모	미하엘 엔데	02/09/1999

row(s) 1 - 15 of 15

[Download](#)