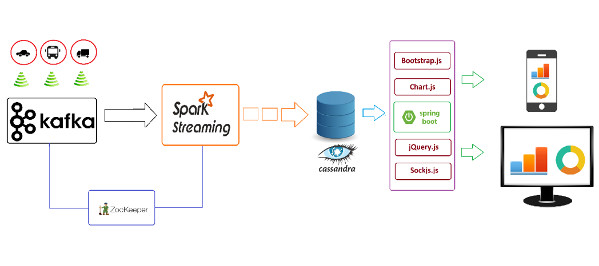
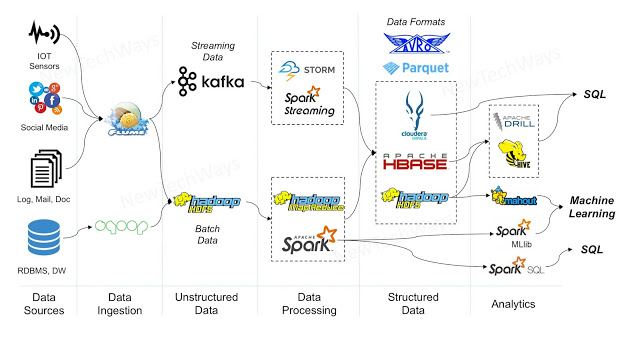
**빅데이터 처리를 위한 윈도우 버전 hadoop, spark 설치 및 간단한 사용법**

빅데이터 처리를 할 수 있으면 SNS, 사물인터넷, 공간정보 등에서 발생되는 수많은 데이터에서 인사이트 있는 정보를 쉽게 얻을 수 있다. 이 글은 윈도우 환경에서 빅데이터 분산 처리를 위한 [hadoop](https://en.wikipedia.org/wiki/Apache_Hadoop), spark 설치 및 사용 방법을 간단히 다룬다.   
  
보통 빅데이터 처리할 때 다음과 같은 아키텍처를 가지고 있는 데, hadoop과 spark는 각각 분산 저장 및 관리, hadoop 기반 분석 및 통계 처리 역할을 한다.   
  
빅데이터 활용 초기에는 하둡이 많이 사용되었으나, 빈번한 디스크 입출력으로 인한 속도저하와 빅데이터 처리 프로그램 개발 및 배포의 불편함(java 기반) 등으로 인해, 스파크가 점차 인기를 끌고 있다. 스파크는 In-memory 처리 방식으로 분산 처리 성능을 크게 향상시켰다. RDD(resilient distributed datasets)을 이용해 메모리 내에서 분산 처리 연산 및 분산된 데이터 유실에 대한 문제를 해결해 가용성을 높였다. 이 방법은 Hadoop의 HDFS(Hadoop Distributed File System) 읽기쓰기 시 디스크 작업 속도보다 10~100배 빠르다.   
  
다음은 일반적인 빅데이터처리 아키텍처를 보여준다. 카프카로 IoT, SNS 등 massive data를 스트리밍해, No SQL로 저장한다. 만약, 빅데이터 분산 저장 및 분석 등이 필요한 경우 스파크를 통해 처리한다. 이를 다양한 가시화 도구를 통해 결과를 보여준다.

[](https://res.infoq.com/articles/traffic-data-monitoring-iot-kafka-and-spark-streaming/en/resources/1fig1.jpg)

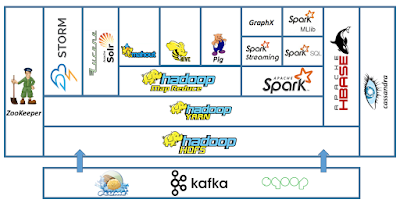
빅데이터 처리 시스템 구조 개념도

다음 그림은 다양한 소스를 통해 얻어지는 빅데이터를 카프카나 하둡으로 얻어 스파크를 통해 No SQL DB에 분산 저장하거나 처리한 후 저장한다. 로그 형식으로 발생되는 데이터는 좀 더 단순한 [플럼(Flume](http://flume.apache.org/download.html). [참고](http://taewan.kim/post/flume_images/))으로 스트리밍 저장할 수 도 있다. 이를 스카파크 SQL 등을 통해 질의한다. 정재된 데이터는 유사도, 군집, 데이터 마이닝을 위해 [머하웃(Mohout](https://mahout.apache.org/general/downloads). [참고)](https://www.3pillarglobal.com/insights/how-to-tame-the-machine-learning-beast-with-apache-mahout)을 사용할 수도 있다. 머하웃은 하둡 맵리듀스(map reduce) 기반 분산 계산이 가능하다. 이 과정은 BI(business intelligent)에서 사용하는 ETL(Extract, Transform and Load)와 매우 유사하다.

[](https://1.bp.blogspot.com/-dEq7FsM5Uyk/Xl2ypauUcuI/AAAAAAAAuUo/3gJcTBv1ZcUKXIM3EiQEjf6z4mpycB9bACLcBGAsYHQ/s1600/pipeline.jpg)

빅데이터 처피 파이프 라인 예시([Big Data, Streaming Data - ETL Analytics Pipeline](http://blog.newtechways.com/2017/11/big-data-streaming-data-etl-analytics.html))

이렇게 구축된 플랫폼위에 트위터에서 사용되는 [스톰(Storm)](https://storm.apache.org/downloads.html)과 같은 실시간 분석 분산 시스템, 하둡파일을 SQL 질의할 수 있는 [Hive](https://hive.apache.org/downloads.html), 대용량 데이터셋을 좀 더 쉽게 다룰수 있는 스크립트를 지원하는 [피그(pig)](https://pig.apache.org/), 솔라([Solr](https://lucene.apache.org/solr/downloads.html))와 같은 웹 기반 검색 엔진을 사용할 수 있다.

[](https://1.bp.blogspot.com/-FviJisvMvj4/XlpMkczqUHI/AAAAAAAAtvI/bJ_SxjJYt986h1Zqmb2sR2bzWc3ZPcYPgCLcBGAsYHQ/s1600/1_34LbLI6jblyMh5LIkZTZEQ.png)

빅데이터 처리 시스템

(Reference: https://intellipaat.com/blog/tutorial/hadoop-tutorial/introduction-hadoop/)

윈도우 환경은 우분투에서 설치하는 것보다 방법이 좀 까다롭다. 이 글은 윈도우 환경에서 하둡, 스파크 설치 테크트리를 정리해 놓는다. 참고로, 이 글은 분산 메시지 스트리밍 플랫폼으로 많이 사용되는 카파(kafka)는 다루지 않는다. 카파 설치 및 사용법은 아래 링크를 참고바란다.

* [분산 메시지 스트리밍 플랫폼 아파치 Kafka 설치 후 사용하기](https://daddynkidsmakers.blogspot.com/2020/03/kafka.html)

기타, 이 예제에 대한 상세 내용은 레퍼런스를 참고 바란다.

소프트웨어 다운로드 및 설치

다음 링크에서 Java, Hadoop, Spark 를 다운로드 한다.

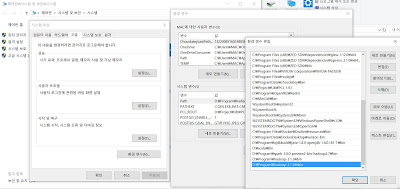
* [Java (8u230+ version)](https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)
* [Skala (2.13.1 windows version)](https://downloads.lightbend.com/scala/2.13.1/scala-2.13.1.msi) in [Skala download page](https://www.scala-lang.org/download/)
* [Hadoop (3.1.3 version)](https://hadoop.apache.org/releases.html)
* [Spark (3.0.0 preview)](https://spark.apache.org/downloads.html)

만약, 오라클 Java Licence 문제를 피하고 싶다면 [OpenJDK](https://adoptopenjdk.net/)를 대신 다운받아 설치한다. Java, Skala는 설치하고, 다른 프로그램은 압축을 푼 후 hadoop, spark 폴더를 만들어 복사해 넣는다(주의 - 폴더에 공백이나 한글이 포함되면 안됨).

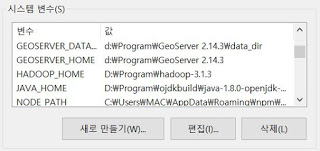
환경 변수 설정

각 설치 프로그램에 대한 실행 경로나 환경 변수를 설정해야 한다. 다음과 같이 시스템 환경 변수를 설정한다.

* JAVA\_HOME: Java 설치 폴더
* HADOOP\_HOME: hadoop 설치 폴더
* SPARK\_HOME: spark 설치폴더

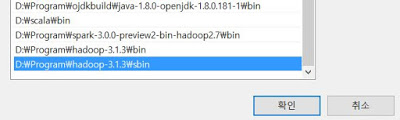
[](https://1.bp.blogspot.com/-zh_-tnAE534/XlpBUPppsKI/AAAAAAAAtt8/5qwF_gCXNQEEUwKPEEFpePqi-HhH2FWWgCLcBGAsYHQ/s1600/60.JPG)

시스템 환경변수 설정 화면

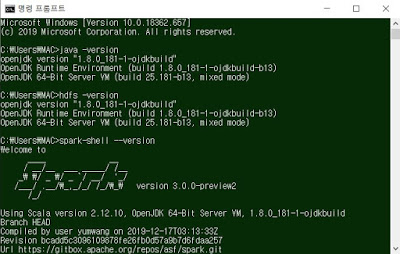
[](https://1.bp.blogspot.com/-LQVj3FwhlU0/XlpCysaW6UI/AAAAAAAAtuc/ExoE3_jWieQcq3MOaiVCdNH1a-sy1fFdgCLcBGAsYHQ/s1600/62.JPG)

설정된 HADOO\_HOME 환경변수

그리고 hadoop, spark, scala 각 프로그램의 bin, sbin폴더를 시스템  Path에 추가해 놓는다. 본인의 경우 다음과 같이 설정하였다.

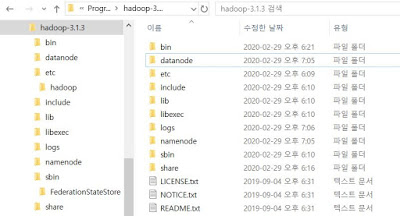
[](https://1.bp.blogspot.com/-CjSxdYfxjFU/XlpCLRjEUCI/AAAAAAAAtuU/Fiu9vRUzu2oS0XngUayqlW2MhGl-coI5gCLcBGAsYHQ/s1600/61.JPG)

제대로 환경변수가 설정되었다면, cmd 창을 실행해 각 프로그램 버전을 다음과 같이 확인할 수 있다.

[](https://1.bp.blogspot.com/-an-pVsUT0_g/XlpDZqZ0TOI/AAAAAAAAtuk/8ep2FUy5_DcD6FINdRq7QxRChYrSkd7rQCLcBGAsYHQ/s1600/63.JPG)

설치 프로그램 버전 확인

설치 후 다음과 같이 hadoop 폴더에 datanode와 namenode 폴더를 만든다.

[](https://1.bp.blogspot.com/-Bp8nEvGJGnU/XlpFvDUwRHI/AAAAAAAAtuw/cO0YLUwuDJkPtrnej1rB_hWoqBGTLFbMwCLcBGAsYHQ/s1600/64.JPG)

HADOOP 설정

Hadoop 설치 폴더의 etc/hadoop 폴더에 설정파일을 수정해야 한다. 다음과 같이 수정한다.

다음과 같이 core-site.xml를 수정한다.

<configuration>

  <property>

    <name>fs.defaultFS</name>

    <value>hdfs://localhost:9000</value>

  </property>

</configuration>

다음과 같이 mapred-site.xml 을 수정한다.

<configuration>

  <property>

    <name>mapreduce.framework.name</name>

    <value>yarn</value>

  </property>

</configuration>

다음과 같이 hdfs-site.xml 을 수정한다. 그리고, C:/hadoop 부분을 경로에 맞게 수정한다.

<configuration>

  <property>

    <name>dfs.replication</name>

    <value>1</value>

  </property>

  <property>

    <name>dfs.namenode.name.dir</name>

    <value>file:///C:/Hadoop/hadoop-<version>/namenode</value>

  </property>

  <property>

    <name>dfs.datanode.data.dir</name>

    <value>file:///C:/Hadoop/hadoop-<version>/datanode</value>

  </property>

</configuration>

다음과 같이 yarn-site.xml 을 수정한다.

<configuration>

  <property>

    <name>yarn.nodemanager.aux-services</name>

    <value>mapreduce\_shuffle</value>

  </property>

  <property>

    <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>

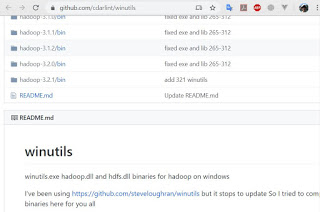
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>

  </property>

</configuration>

Winutils hadoop  윈도우버전 패치

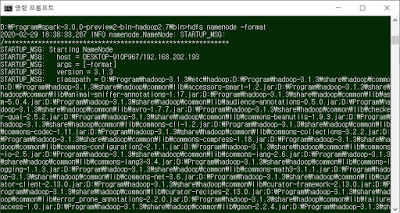
winutils는 hadoop을 윈도우에서 동작하도록 해준다. [winutils 링크](https://github.com/cdarlint/winutils)에서 폴더를 통채로 다운로드 하고, 압축을 푼 후, hadoop-3.1.2/bin의 파일들을 설치된 hadoop의 bin에 폴더에 덮어 쓴다.

[](https://1.bp.blogspot.com/-Es-AJYrmYQo/XlpGpwzx4kI/AAAAAAAAtu8/SPH3l6yaYowBrvVF2Kh5ElaUYxh7vjS-QCLcBGAsYHQ/s1600/65.JPG)

[winutils 링크](https://github.com/cdarlint/winutils)

이제 다음과 같이 hdfs namenode 을 포맷한다.

hdfs namenode -format

[](https://1.bp.blogspot.com/-Z_nL1V-SVHE/Xlo-OHfhvII/AAAAAAAAtss/MmqzwVSxSNcjbjUKNCRsP-tAZzDYdWzygCLcBGAsYHQ/s1600/50.JPG)

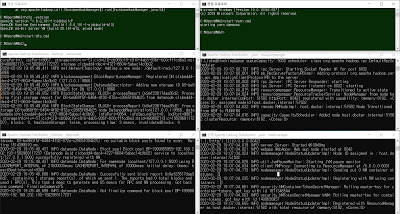
그리고, hadoop 설치폴더의 /share/hadoop/yarn/timelineservice 폴더 내 hadoop-yarn-server-timelineservice-<version> 파일을 hadoop 설치폴더 /share/hadoop/yarn 폴더로 복사해 넣는다.  
  
HDFS 서버 시작하기

이제 다음과 같이 HDFS 서버를 시작한다.

start-dfs.cmd

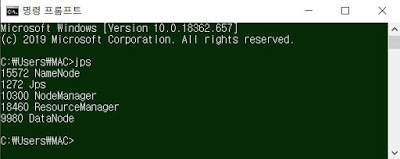
start-yarn.cmd

그럼 다음과 같이 서버가 실행된다.

[](https://1.bp.blogspot.com/-Wx2EcOgMPkw/Xlo-OBSMbpI/AAAAAAAAts0/HqnyOz_YorghBt4iqRarRtHRd6K9acv_gCLcBGAsYHQ/s1600/52.JPG)

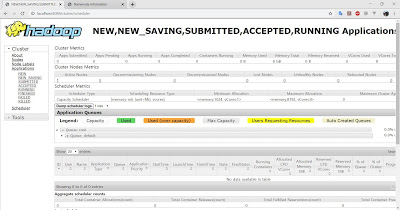
HDFS 서버 실행 모습

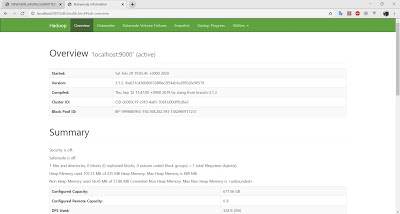
현재 HDFS 실행 상태를 확인하기 위해 다음과 같이 jps 명령을 입력한다.

[](https://1.bp.blogspot.com/-tpWVoq81kho/Xlo-O9aFDUI/AAAAAAAAts4/ts-DKa8wL9w146duZ1_FbGBmZvPdSGIdQCLcBGAsYHQ/s1600/53.JPG)

jps 실행 후 모습

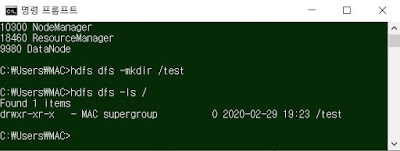
데쉬보드를 확인하기 위해 크롬에서 localhost:8088, localhost:9870 을 다음처럼 입력한다.

[](https://1.bp.blogspot.com/-Xat15uj_zr4/Xlo-PURBaJI/AAAAAAAAts8/mCUcRAiBRcQuAvm69hwFiToLkYg-Hz9fgCLcBGAsYHQ/s1600/54.JPG)

[](https://1.bp.blogspot.com/-I1XpwOuXmNk/Xlo-PSBxMMI/AAAAAAAAttA/OJlH63Y0dHsVuRQLrkWZfL5ylsWLUAn_QCLcBGAsYHQ/s1600/55.JPG)

Hadoop 서버 데쉬보드

HADOOP과 Spark 사용 테스트  
하둡과 스파크 간단한 테스트를 위해 다음과 같이 hadoop 파일 시스템 내 test 폴더를 만든다.  
hdfs dfs -mkdir /test  
hdfs dfs -ls /

[](https://1.bp.blogspot.com/-0mnSAeZ-BCc/Xlo-PqhmfEI/AAAAAAAAttE/8zwxQ6durfElVAgxaM-qfnIHzO_fs8IawCLcBGAsYHQ/s1600/56.JPG)

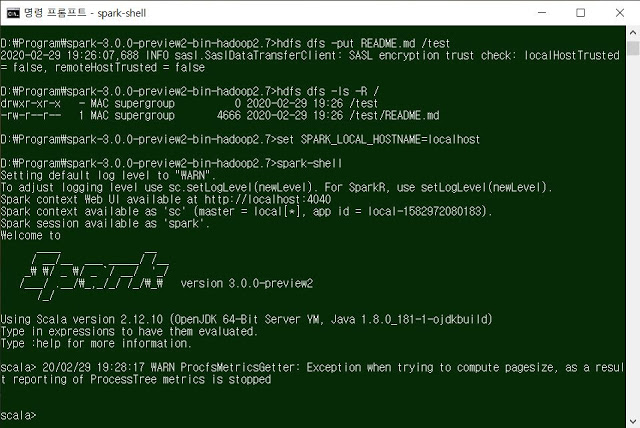
실행 결과

다음과 같이 spark 설치 폴더의 README 파일을 하둡 파일 시스템에 put 하고 이를 스파크로 테스트해본다.  
hdfs dfs -put README.md /test  
hdfs dfs -ls -R /

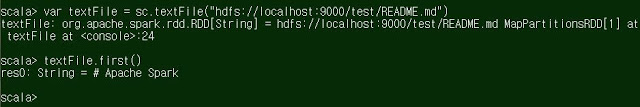
set SPARK\_LOCAL\_HOSTNAME=localhost

spark-shell

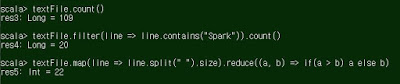
그럼 다음과 같이 spark shell 이 실행된다. 참고로, 실행 전에 SPARK\_LOCAL\_HOSTNAME=localhost 를 꼭 설정해야 한다(참고 - SparkContext 초기화 에러). 아울러, 스파크는 스칼라 이외에 파이썬, 자바 등 다양한 언어를 제공한다. 파이썬의 경우 pyspark를 실행해 입력할 수 있다.

[](https://1.bp.blogspot.com/-6XcBFn2e3ms/Xlo-QNHxUgI/AAAAAAAAttI/nCjEu4mZm7YxUFUFNLA5akv-ReiFN-EiQCLcBGAsYHQ/s1600/57.JPG)

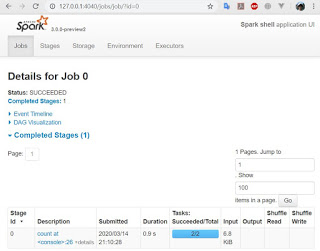
다음과 같이 코딩해 본다.  
scala> List(1, 2, 3).zip(List("a", "b", "c"))  
res2: List[(Int, String)] = List((1,a), (2,b), (3,c))  
  
다음과 같이 spark 에서 hadoop 파일을 열고, 읽어본다.  
var textFile = sc.textFile("hdfs://localhost:9000/test/README.md")  
textFile.first()

[](https://1.bp.blogspot.com/-TE9IoPxSGJI/Xlo-QQotC-I/AAAAAAAAttM/OAfrgnkXCr0Uouu5klnvnWmDMIMKLNkyQCLcBGAsYHQ/s1600/58.JPG)

다음과 같이 count, filter, map, reduce 함수를 사용해 코딩해 본다.  
textFile.count()  
textFile.filter(line => line.contains("Spark")).count()  
textFile.map(line => line.split(" ").size).reduce((a, b) => if(a > b) a else b)  
  
아래와 같이 결과가 출력되면 성공한 것이다.

[](https://1.bp.blogspot.com/-W7qc7rb8S2A/XlpPwoD7gCI/AAAAAAAAtvU/58FgEKFn-iwbA_D_zrJCThVLcRMFDY2twCLcBGAsYHQ/s1600/66.JPG)

스파크는 http://127.0.0.1:4040 에 웹UI를 제공한다.

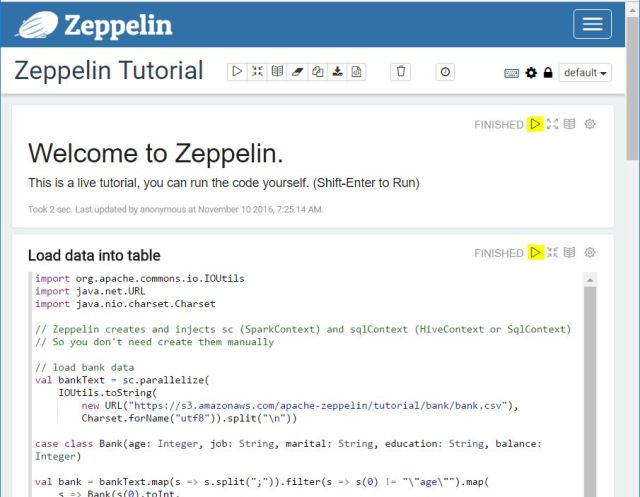
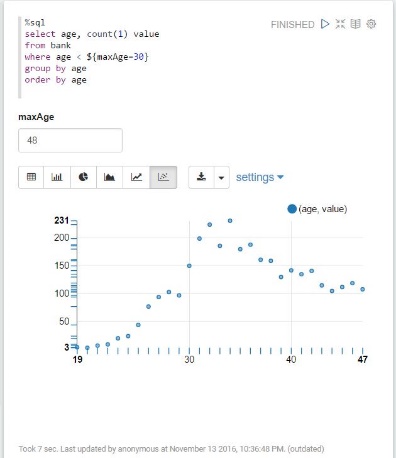
[](https://1.bp.blogspot.com/-47OgU_GxA3o/XmzKFbTisfI/AAAAAAAAubU/58gtolTpOS8wkdhiegYEbINErMtUz-TUQCLcBGAsYHQ/s1600/109.JPG)

스파크 웹 UI

스파크 어플리케이션을 스칼라, 파이선, 자바 등으로 개발해 jar 파일을 만들면, [spark-submit](https://spark.apache.org/docs/latest/submitting-applications.html)을 이용해 실행할 수 있다. 예를 들어, NoSQL에 저장된 IoT센서 빅데이터를  스파크, Hadoop으로 병렬처리해 일별, 월별, 년별 분석하는 등의 프로그램을 미리 만들어 놓고 필요할 때 실행할 수 있다. 이에 대한 예제는 다음 링크를 참고한다.

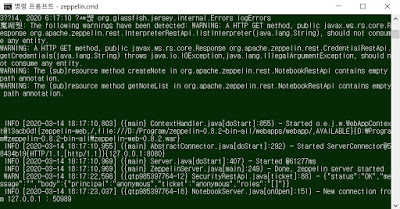
* [Spark Python Application](https://www.tutorialkart.com/apache-spark/submit-spark-application-python-example/)
* [How to Write Spark Applications in Python](http://blog.appliedinformaticsinc.com/how-to-write-spark-applications-in-python/)

빅데이터 분석 가시화와 NoSQL 데이터베이스  
명령행에서 스칼라 코딩하기 어렵거나 분석된 데이터를 손쉽게 가시화하고 싶다면, 데이터 분석 시각화 도구인 제플린([zeppelin](https://zeppelin.apache.org/download.html) [Binary package with all interpreters download](https://zeppelin.apache.org/download.html)) 노트북을 다운로드한다. 제플린에서는 UI에서 스파크 함수를 사용해 프로그램 실행할 수 있다.

[](https://hernandezpaul.files.wordpress.com/2016/11/running-notes-zeppelin-tutorial.jpg?w=640)[](https://hernandezpaul.files.wordpress.com/2016/11/note-reults.jpg)

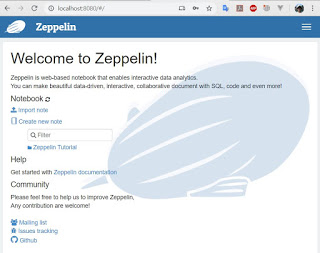
[제플린에서 스파크 코딩](https://hernandezpaul.wordpress.com/2016/11/14/apache-zeppelin-installation-on-windows-10/) 및 [빅데이터 분석 결과](https://hernandezpaul.wordpress.com/2016/11/14/apache-zeppelin-installation-on-windows-10/)

다운로드 받은 제플린 압축을 풀고, 다음 같이 제플린 서버를 실행한다.  
cd bin  
zeppelin.cmd  
  
앞서 언급한 환경이 제대로 설정되어 있다면, 다음과 같이 서버가 잘 실행될 것이다.

[](https://1.bp.blogspot.com/-1Z7TMMkiRPA/XmyiXQmAOFI/AAAAAAAAuac/Qhx9080d7owSSsQQIoVUrw5jcVw3HwivgCLcBGAsYHQ/s1600/106.JPG)

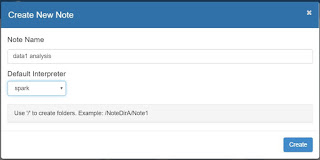
제플린 서버 실행 모습

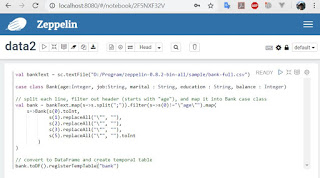
브라우저에서 <http://localhost:8080/#/> 에 접속한다.

[](https://1.bp.blogspot.com/-dHQNLLhqW7c/XmyiXC9o3kI/AAAAAAAAuaY/IJi4WQnW8SgBfPajZr2FvZTb7ozvYBtQACLcBGAsYHQ/s1600/105.JPG)

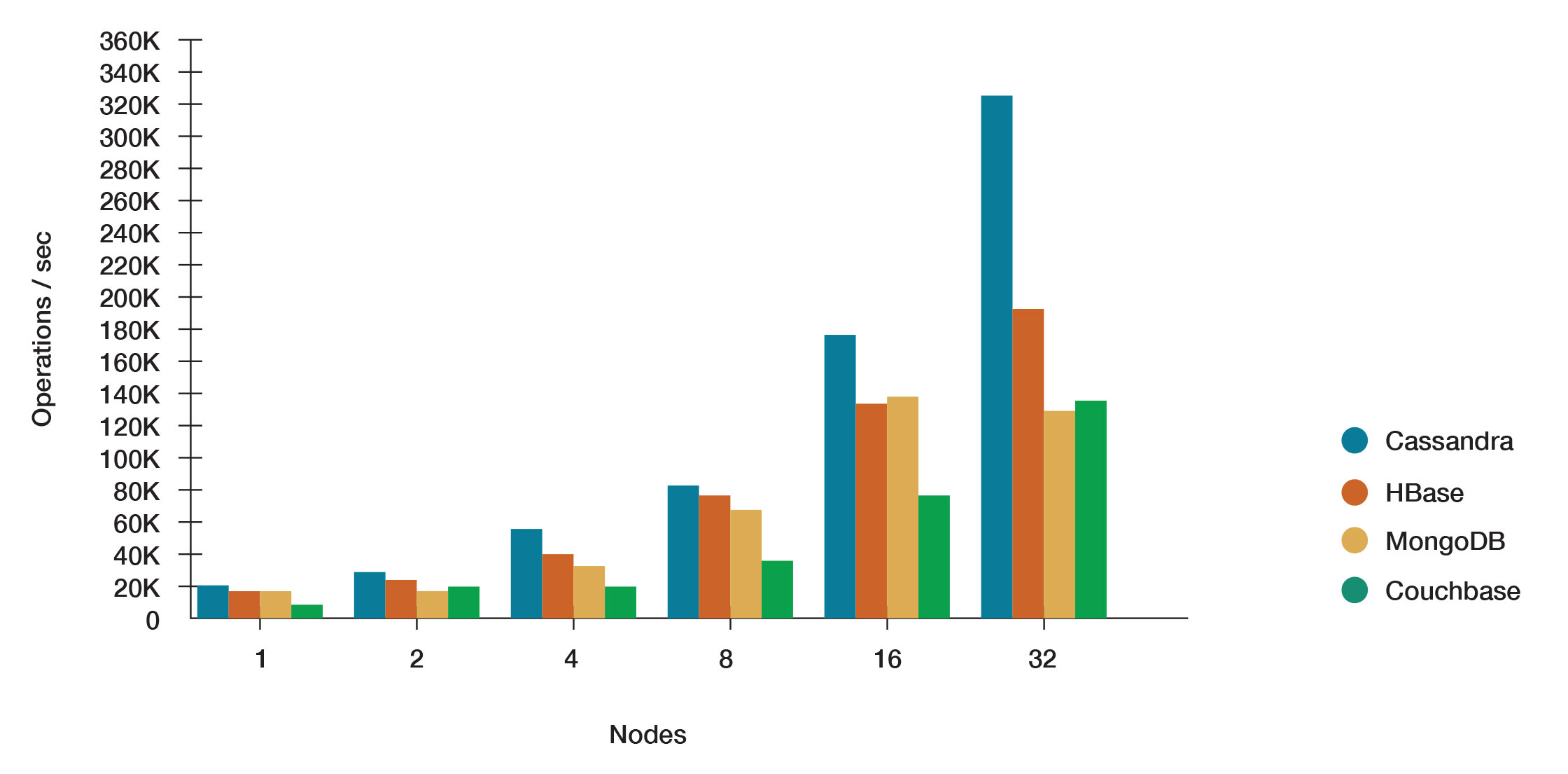
제플린 서버 접속 모습

제플린에서 Create new note 를 클릭한다. [Zeppelin Tutorial](https://zeppelin.apache.org/docs/0.6.2/quickstart/tutorial.html)에서 다음 예제와 [데이터](http://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank.zip)를 얻어 실행해 본다. 참고로, 제플린과 스파크 간에 호환성 문제가 있어 에러 발생할 수 있다. 이 경우, 아파치 스파크 2.3.2와 제플린 0.8 버전을 사용한다.  
val bankText = sc.textFile("D:/Program/zeppelin-0.8.2-bin-all/sample/bank-full.csv")  
  
case class Bank(age:Integer, job:String, marital : String, education : String, balance : Integer)  
  
// split each line, filter out header (starts with "age"), and map it into Bank case class  
val bank = bankText.map(s=>s.split(";")).filter(s=>s(0)!="\"age\"").map(  
    s=>Bank(s(0).toInt,   
            s(1).replaceAll("\"", ""),  
            s(2).replaceAll("\"", ""),  
            s(3).replaceAll("\"", ""),  
            s(5).replaceAll("\"", "").toInt  
        )  
)  
  
// convert to DataFrame and create temporal table  
bank.toDF().registerTempTable("bank")

[](https://1.bp.blogspot.com/-mht9hWqCZl4/Xmym33Pre5I/AAAAAAAAuas/GMLLCxl20KEhILvbSJiFASEiph27LQ3XwCLcBGAsYHQ/s1600/107.JPG)

[](https://1.bp.blogspot.com/-ECzkmY5i0w0/Xmym35ZFn0I/AAAAAAAAuaw/rKKNw_9xNG8aTprnPPj6D0LefO5_XbgdQCLcBGAsYHQ/s1600/108.JPG)

빅데이터 분석된 정보는 NoSQL 데이터베이스인 몽고 DB(MongoDB), 카산드라([Cassandra](https://www.baeldung.com/kafka-spark-data-pipeline)), [HBase](https://medium.com/@OpcitoTechnologies/building-a-real-time-data-pipeline-using-spark-streaming-and-kafka-a603495c4213) 등에 저장할 수 있다. NoSQL은 각 DB마다 장단점이 있고, 응용 목적에 따라 성능이 다르니 이에 맞게 사용해야 한다(참고 - [NoSQL DB 비교 #1](https://logz.io/blog/nosql-database-comparison/), [#2](https://db-engines.com/en/system/Cassandra%3BHBase%3BMongoDB)).

[](https://www.datastax.com/sites/default/files/inline-images/chart-load-process-v4-rev2.png)

[NoSQL 성능 비교(datastax.com)](https://www.datastax.com/products/compare/nosql-performance-benchmarks)

스칼라를 이용해 빅데이터를 시간, 날짜, 월, 년도별로 분석해 MongoDB로 저장하고 싶거나, 센서 데이터를 필터링한 후 필요한 데이터만 저장하고 싶다면, 다음 예시를 참고한다.

* [Submit spark command](https://docs.qubole.com/en/latest/rest-api/command_api/submit-a-spark-command.html), [AWS example](https://aws.amazon.com/ko/premiumsupport/knowledge-center/emr-submit-spark-job-remote-cluster/), [spark application in cloudera](https://docs.cloudera.com/runtime/7.0.3/running-spark-applications/topics/spark-run-sample-apps.html)
* [spark Mongo IoT](https://github.com/giorbernal/spark-mongo-iot)
* [write to MongoDB](https://docs.mongodb.com/spark-connector/master/java/write-to-mongodb/)

마무리  
빅데이터 처리를 위해 하둡은 데이터를 분산 저장, 처리 및 병합하는 Map reduce 방식을 사용한다. 스파크는 이를 이용해 대용량 데이터 분석 및 통계 처리를 할 수 있다. 환경 설정이 쉽지는 않지만, 앞의 내용을 잘 따라 설치하면 큰 문제 없이 빅데이터 처리를 지원하는 하둡 개발 환경을 만들 수 있을 것이다.  
  
레퍼런스  
스파크 실행 및 관련 환경 예시는 다음을 참고한다.

* [Installing and Running Hadoop and Spark on Windows](https://dev.to/awwsmm/installing-and-running-hadoop-and-spark-on-windows-33kc)
* [How to install Apache Spark on Windows 10](https://guendouz.wordpress.com/2017/07/18/how-to-install-apache-spark-on-windows-10/)
* [Spark Install in windows](https://mas-dse.github.io/DSE230/installation/windows/)
* [Getting started with apache spark](https://www.bigdata-toronto.com/2016/assets/getting_started_with_apache_spark.pdf)
* [Traffic Data Monitoring Using IoT, Kafka and Spark Streaming](https://www.infoq.com/articles/traffic-data-monitoring-iot-kafka-and-spark-streaming/)
* [Interactive Analysis with the Spark Shell](https://ourcstory.tistory.com/128)
* [First Steps with PySpark and Big data processing](https://realpython.com/pyspark-intro/)
* [PySpark Tutorial for Beginners: Machine Learning Example](https://www.guru99.com/pyspark-tutorial.html)
* [Learn how to use PySpark in under 5 minutes (Installation + Tutorial)](https://www.kdnuggets.com/2019/08/learn-pyspark-installation-tutorial.html)
* [Building a Data Pipeline with Kafka, Spark and Cassandra](https://www.baeldung.com/kafka-spark-data-pipeline)
* [How to Build data pipeline using kafka, spark, and Hive](https://dzone.com/articles/data-pipeline-using-kafka-spark-amp-hive)
* [Setting up and running apache kafka on windows](https://dzone.com/articles/running-apache-kafka-on-windows-os)
* [Kfaka and in Cloud&BigData](https://blog.acronym.co.kr/category/Cloud%26BigData?page=6)
* [Smoke detection system, build with Arduino, MQTT, Hadoop, Spark, MongoDB and Flask](https://github.com/riyadh11/Fumaca)
* [IoT Spotlight: Why did we bring an Arduino to Hadoop Summit?](https://mapr.com/blog/iot-spotlight-why-did-we-bring-arduino-hadoop-summit/)
* [하둡과 맵리듀스 스파크의 관계](https://3months.tistory.com/511)

스칼라 설명 및 스파크 어플리케이션 개발은 다음을 참고한다.

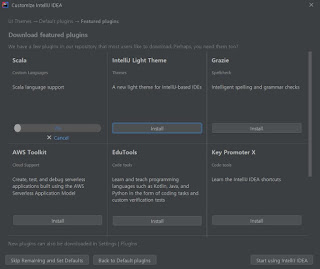
* [Spark Cluster, Yarn, PySpark](http://incredible.ai/spark/2016/02/11/Spark-YARN-Cluster/)
* [스칼라, 스파크 시작하기](https://wikidocs.net/book/2350)

제플린 설치 방법은 아래 링크를 참고한다.

* [Zeppelin, Spark, PySpark Setup on Windows (10)](https://gist.github.com/codspire/7b0955b9e67fe73f6118dad9539cbaa2)
* [Apache Zeppelin installation on Windows 10](https://hernandezpaul.wordpress.com/2016/11/14/apache-zeppelin-installation-on-windows-10/)
* [How To Locally Install & Configure Apache Spark & Zeppelin](https://dziganto.github.io/anaconda/shiro/spark/zeppelin/zeppelinhub/How-To-Locally-Install-Apache-Spark-And-Zeppelin/)

IntelliJ을 설치한 후 바로 Scala 플러그인을 설치해 IDE환경에서 편하게 코딩할 수 있다.

* [IntelliJ IDEA Community Edition Download](https://www.jetbrains.com/idea/download/#section=windows)

[](https://1.bp.blogspot.com/-6ni_S-8R_r8/XlpSR810BtI/AAAAAAAAtvg/UQNPtS7HgQMMhYl9gTPySGiiI-WBfQzbQCLcBGAsYHQ/s1600/67.JPG)

IntelliJ 에서 Scala 플러그인 설치 모습