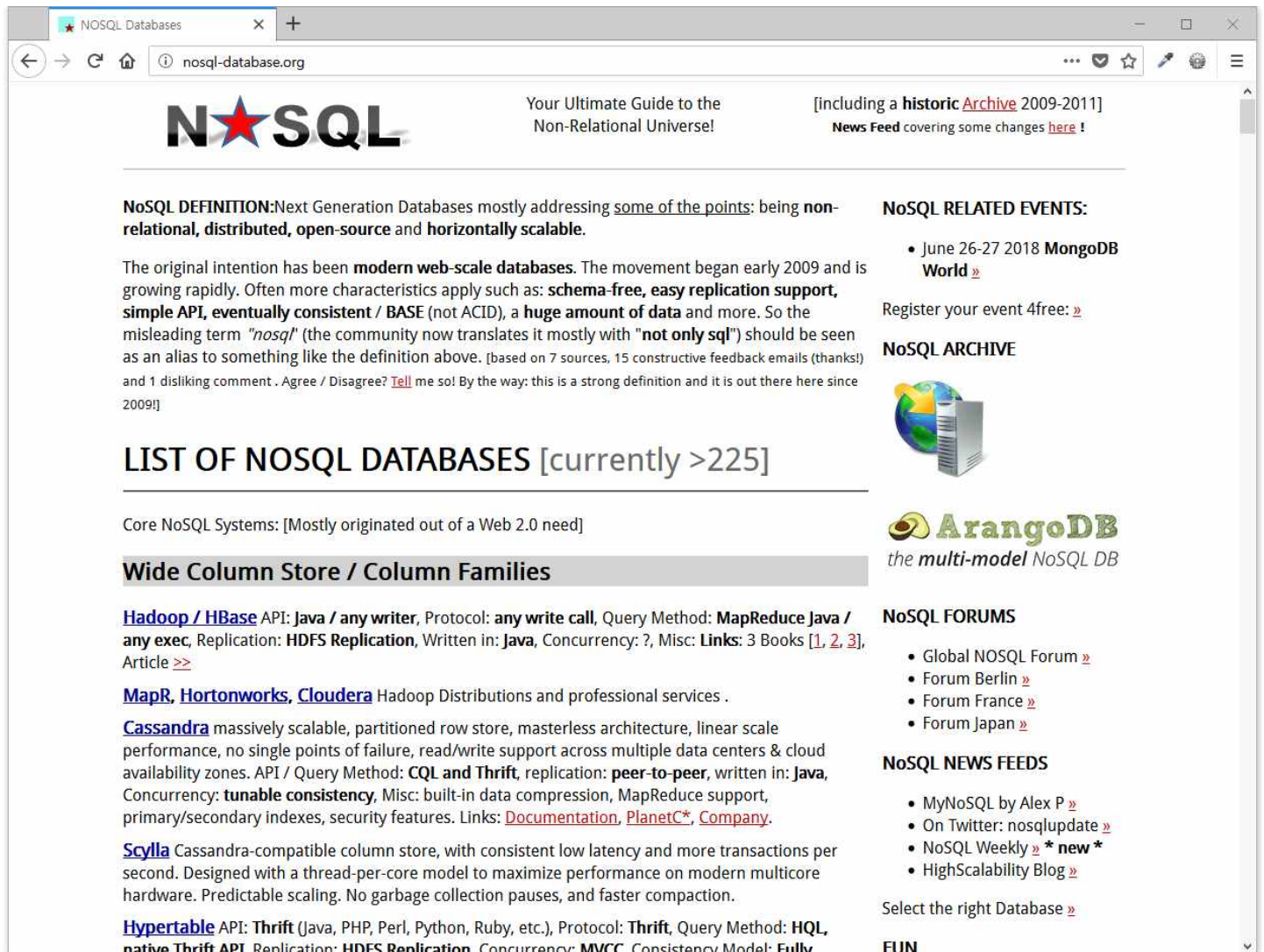


■ NoSQL

1) <http://nosql-database.org/>



- 현재 no sql의 갯수는 225개



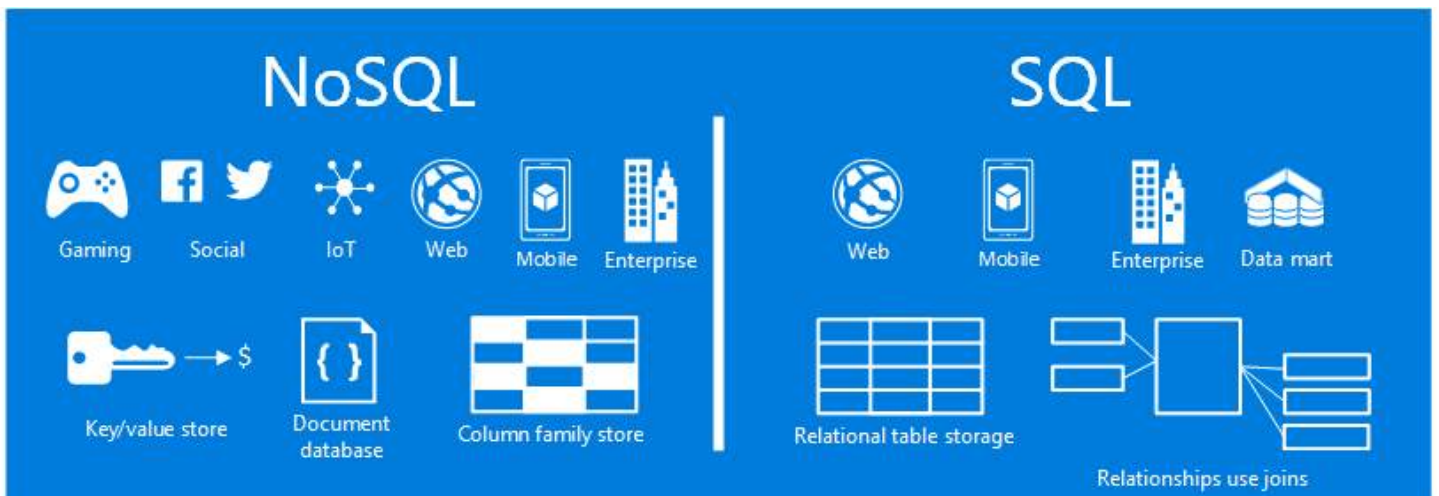
- Not Only SQL : 기존의 RDMS가 갖고 있는 특성 뿐 아니라 다른 특성을 부가적으로 지원

- NoSQL 데이터베이스는 확장 가능한 성능 및 스키마 없는 데이터 모델에 최적화된 비관계형 데이터베이스
- NoSQL 데이터베이스 시스템에서는 인 메모리 키-값 스토어, 그래프 데이터 모델, 문서 스토어 등 데이터 관리를 위해 다양한 모델을 사용
- 큰 데이터 볼륨, 짧은 지연 시간, 유연한 데이터 모델이 필요한 애플리케이션에 최적화
- 기존 관계형 데이터베이스의 데이터 일관성 제약을 일부 완화함

1. 등장배경

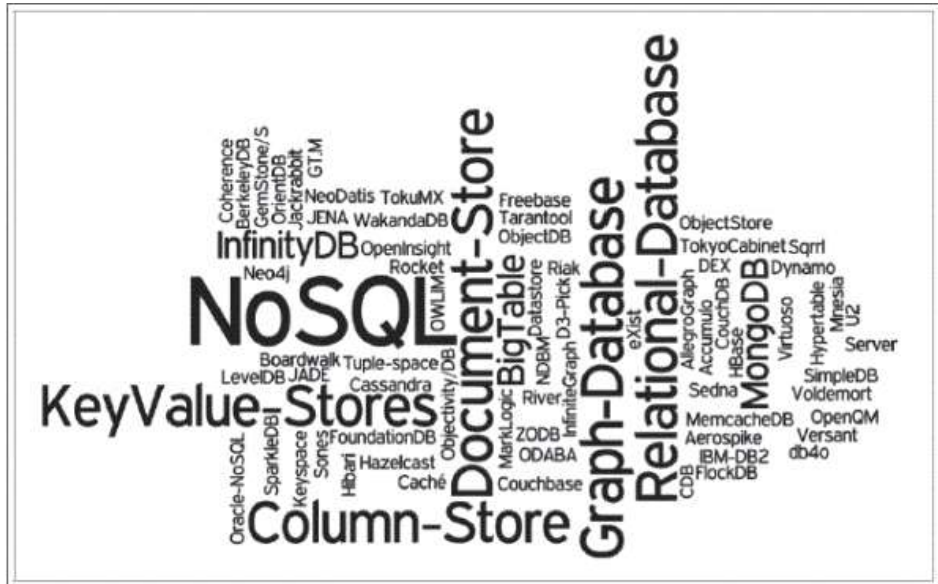
- 1988년 카를로 스트로찌가 공개한 표준 SQL인터페이스를 이용하지 않은 자신의 경량 오픈소스 관계형 데이터베이스를 NoSQL이라고 명명한 데서 유래
- 2000년대 후반에 넘어오면서 인터넷이 활성화 되고 소셜 네트워크 서비스등이 등장하면서 관형형 데이터 또는 비정형 데이터를 보다 쉽게 담아서 저장하고 처리할 수 있는 NoSQL 데이터베이스가 각광을 받음
- 기존 관계형 데이터베이스의 특성을 제공하지는 않지만 뛰어난 확장성과 성능 등의 특성을 갖는 비관계형 분산 데이터베이스들이 등장
- 이때 구글과 아마존에 의해 빅테이블(Bigtable)과 Dynamo라는 논문이 발표되었고, 이 두 논문은 새로운 데이터 저장 기술을 만들어내는 시발점이 됨

2. 특징



- NoSQL은 RDBMS와는 달리 데이터 간의 관계를 정의하지 않음
- RDBMS가 데이터의 관계를 Foreign Key 등으로 정의하고 이를 이용해 Join 등의 관계형 연산을 함
- NoSQL은 데이터 간의 관계를 정의하지 않음
- 데이터 테이블은 그냥 하나의 테이블이며 각 테이블 간의 관계를 정의하지도 않고 일반적으로 테이블 간의 Join도 불가능
- RDBMS에 비해 훨씬 더 대용량의 데이터를 저장할 수 있음
- 분산형 구조
- 스키마가 고정되어 있지 않음

3. 종류



① Key/Value Model

- Key와 Value의 쌍으로 데이터가 저장
- Oracle Coherence, Redis

② Ordered Key/Value Model

- Key/Value Store의 확장된 형태로 Key/Value Store와 데이터 저장 방식은 동일
- 데이터가 내부적으로 Key 순서로 Sorting되어 저장
- Cassandra

③ BigTable-style Model

- 행(Row) 기반의 관계형 데이터베이스와는 다르게 value에 있어 컬럼(Column)을 기본 구성으로 함
- value 자체를 지속적으로 다차원적인 Map(Map의 Map을 지원)으로 구성
- HBase

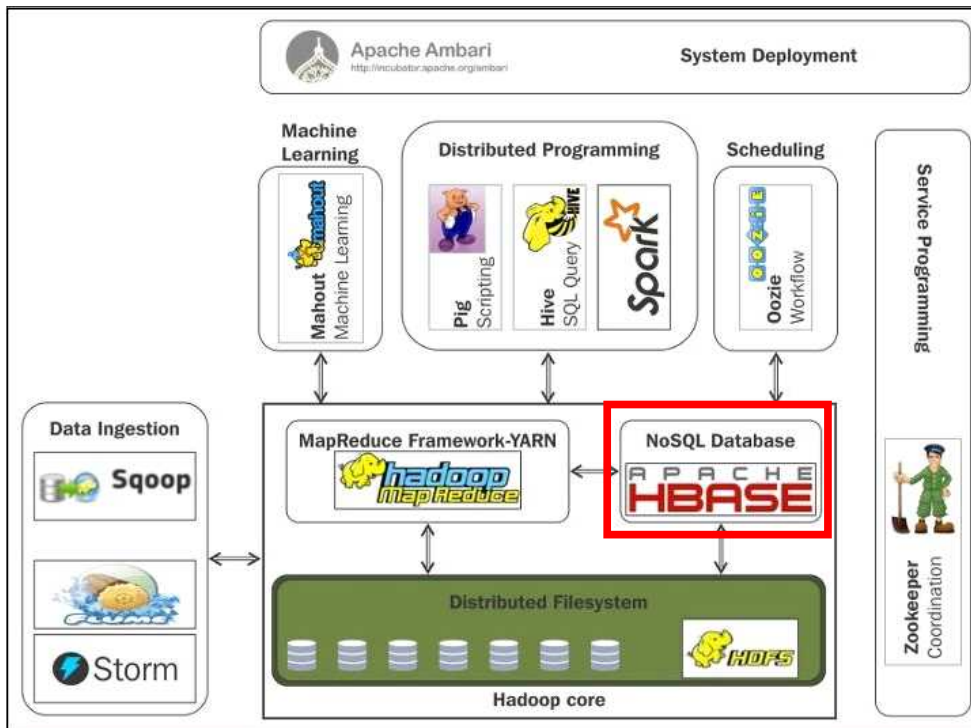
④ Document Key/Value Store

- Key/Value Store의 확장된 형태로, 기본적으로는 Key/Value Store
- Key에 해당하는 Value 필드에 데이터를 저장하는 구조는 같으나, 저장되는 Value의 데이터 타입이 Document라는 타입을 사용
- Document 타입은 XML, JSON, YAML과 같이 구조화된 데이터 타입으로, 복잡한 계층 구조를 표현
- MongoDB, CouchDB, Riak

■ HBase: 데이터 적재



- 공식사이트: hbase.apache.org
- 구글 Bigtable을 모델로 하여 초기 모델 개발(2007년)
- Hadoop 기반의 NoSQL



- 키/값(key/value) 형식으로 단순하게 구조화
- BigData 처리가 가능하고 MapReduce도 가능
- 분산 데이터 저장소
- 데이터를 다수 서버에 분산 보관
- 자주 접근되는 데이터를 메모리에 캐시
- 물리적인 저장소로 HDFS를 사용해 데이터를 영속적으로 보관
- 실시간으로 데이터를 랜덤(random)하게 읽기와 쓰기 가능
- 작은 데이터를 인터랙티브하게 읽고 쓸 수 있음

■ 데이터 모델



Hbase(2개의 컬럼 패밀리, 4개의 컬럼, 8개의 셀)

```

1010101 column=location:city, timestamp=1529469708315, value=서울
1010101 column=location:country, timestamp=1529469706772, value=한국
1010101 column=message:message, timestamp=1529469698762, value=안녕하세요
1010101 column=message:name, timestamp=1529469691283, value=김필구

1010102 column=location:city, timestamp=1529469708315, value=서울
1010102 column=location:country, timestamp=1529469706772, value=한국
1010102 column=message:message, timestamp=1529469698762, value=반갑습니다
1010102 column=message:name, timestamp=1529469691283, value=박구현
  
```

RDBMS(row key를 포함해 컬럼이 5개)

rowkey	name	message	country	city
1010101	김필구	안녕하세요	한국	서울
1010102	박구현	반갑습니다	한국	서울

① Hbase의 테이블은 1개 이상의 컬럼 패밀리를 가짐

② **행(Row)**

-rowKey와 하나 이상의 컬럼으로 구성됨

-rowKey를 기준으로 알파벳 순으로 정렬(유일한 인덱스)

③ **컬럼(Column)**

-Hbase는 Column Family와 Column Qualifier로 구성되어있음

-컬럼은 HBase의 최소 단위

-콜론(:)으로 구분됨

④ **컬럼 패밀리(Column Family)**

- 컬럼+값의 집합을 같은 장소에 모아 놓은 것

⑤ 컬럼 수식어(Column Qualifier)

- 데이터에 인덱스를 제공
- 예를 들어 message 컬럼패밀리의 senderName가 수식어-> 'message:name'

⑥ 셀(Cell)

- 로우, 컬럼패밀리, 컬럼 수식어의 집합
- 내부적으로 값(value)과 타임스탬프(Timestamp)를 가지고 있음
- 값의 버전을 표현하기 위해 timestamp 값을 포함
- 셀의 값은 바이트 배열

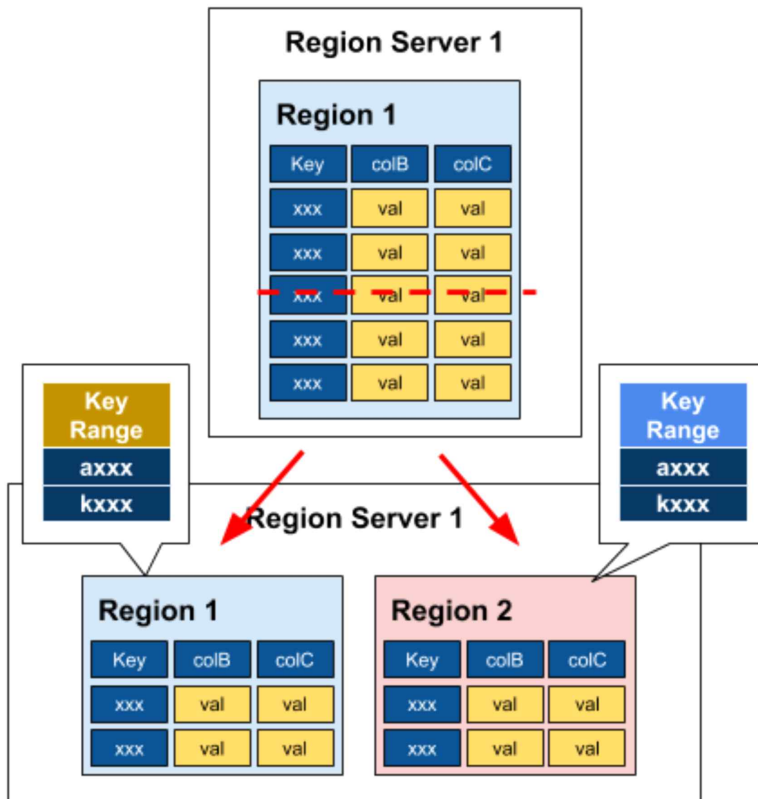
⑦ 타임스탬프(Timestamp)

- 주어진 값의 버전 식별자로 값과 나란히 기록
- 기본적으로는 RegionServer의 시간으로 셀에 데이터를 넣을 때 명시 가능
- 타임스탬프의 값을 직접적으로 조작하는 것은 드문일로, 보통 권장하지 않음
- 어플리케이션 레벨에서는 타임스탬프를 인코딩하는 것을 권장

■ 아키텍처

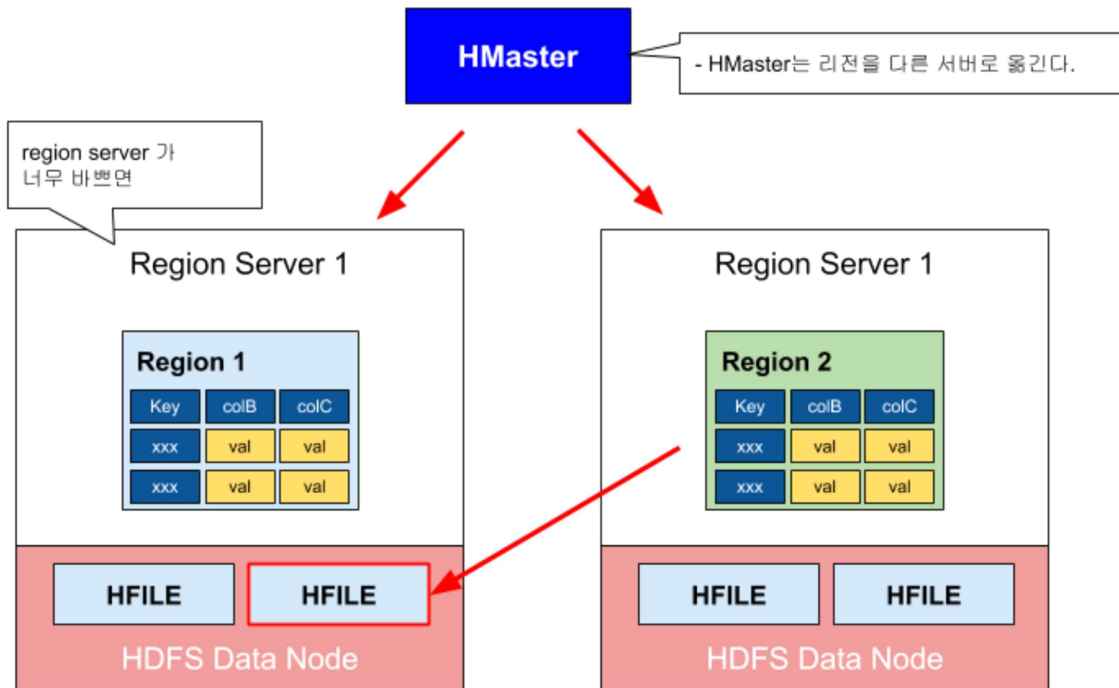
1. 리전(Region)

- HBase 테이블의 분할된 블록(기본 크기 64KB)
- 각 리전은 특정 범위의 rowkey 데이터를 다룸
- 예) 리전1: [~1000], 리전2: [1001~]
- 리전 서버마다 n개의 리전을 제공
- 일정 크기 이상 커지면 분리



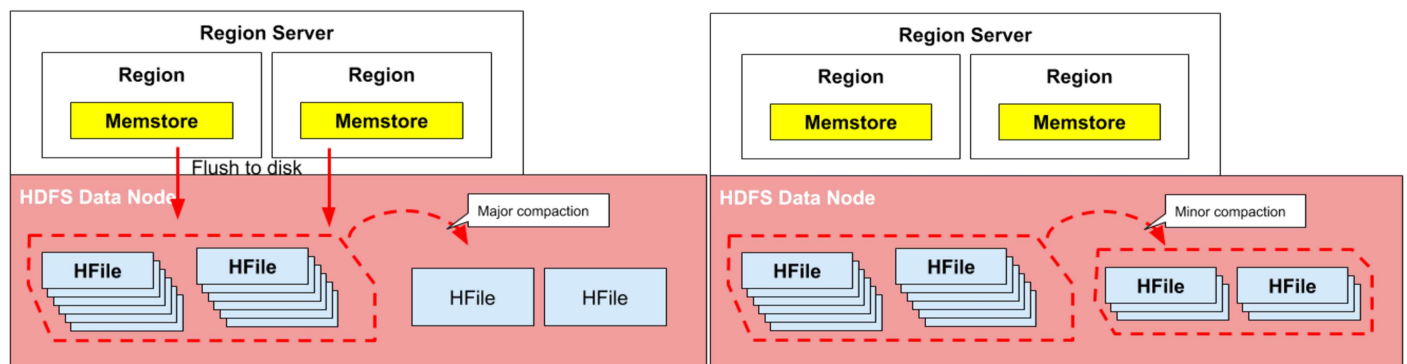
2. Master Server

- Region을 각 RegionServer에 할당
- Region들이 속한 각 Region서버의 메타 정보를 관리
- 업무 수행하기 위해 주키퍼와 소통
- 테이블의 생성이나 컬럼 패밀리 생성과 같은 스키마의 변화나 메타데이터 연산 담당



3. RegionServer

- 하나의 RegionServer에는 다수의 리전이 생성되어 리전을 관리
- 클라이언트와 통신하고 데이터 관련 연산을 관리
- 내부 리전의 읽기와 쓰기 요청 관리
- 내부에 메모리 저장소인 MemStore와 HFile을 가지고 있음
- 처음 데이터를 put하면 MemStore에 저장되고 MemStore의 설정된 임계치 값 이상이 되면 HFile로 플러시(flush) 하고 HFile도 임계치 이벤트 시점이 되면 하둡의 HDFS로 데이터를 플러시(flush)
- 이러한 플러시 과정을 Major/Minor Compaction 이라고 함
- 메모리 저장소는 캐시와 같은 동작 수행



■ 설치

- server01에 bigdata계정으로 접속 후 홈디렉터리에 hbase 다운

```
$ · wget http://mirror.navercorp.com/apache/hbase/stable/hbase-1.2.6.1-bin.tar.gz
```

- 다운 받은 tar파일 압축 풀기

```
$ · tar xvfz hbase-1.2.6.1-bin.tar.gz
```

- hbase 심볼릭 링크 설정

```
$ · ln -s hbase-1.2.6.1 hbase
```

- root 계정으로 **/etc/profile**을 열어 Hbase Path 설정

```
$ · vi /etc/profile
```

```
export HBASE_HOME=/home/bigdata/hbase  
export PATH=$PATH:$HBASE_HOME/bin
```

- **bigdata**계정으로 Hbase의 Pid 정보를 저장하는 디렉토리 생성

- Hbase는 시작되면 자신의 pid정보를 파일로 저장해 멈추게 되면 pid 파일에 있는 정보를 읽어 kill 시그널 전송
- Hbase의 Znode 파일이 생성

```
$ mkdir /home/bigdata/var/hbase
```

- hbase의 설정 파일은 conf 디렉터리 안에 있음

```
$ cd /home/bigdata/hbase/conf/
```

- hbase-env.sh 파일을 열어 아래와 같이 수정

```
$ vi hbase-env.sh
```

```
# Configure PermSize. Only needed in JDK7. You can safely remove it for JDK8+
export HBASE_MASTER_OPTS="$HBASE_MASTER_OPTS -XX:PermSize=128m -XX:MaxPermSize=128m"
export HBASE_REGIONSERVER_OPTS="$HBASE_REGIONSERVER_OPTS -XX:PermSize=128m -XX:MaxPermSize=128m"
```

- 이 부분은 JDK7에서만 필요함
- 삭제하거나 주석처리

```
export JAVA_HOME=/usr/local/java
export HBASE_PID_DIR=/home/bigdata/var/hbase
export HBASE_MANAGES_ZK=false
```

- 주석을 풀어서 우리에게 필요한 값으로 설정
- HBASE_MANAGES_ZK:true는 hbase가 직접 주키퍼를 키고 관리한다는 것이기 때문에 false로 설정

- hbase-site.xml 파일을 열어 아래와 같이 입력

```
$ vi hbase-site.xml
```

```
<property>
    <name>hbase.rootdir</name>
    <value>hdfs://server01:8020/hbase</value>
</property>
<property>
    <name>hbase.zookeeper.quorum</name>
    <value>server01,server02,server03</value>
</property>
<property>
```

```

        <name>hbase.zookeeper.property.dataDir</name>
        <value>/home/bigdata/var/zookeeper</value>
    </property>
    <property>
        <name>hbase.cluster.distributed</name>
        <value>true</value>
    </property>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>hbase.zookeeper.property.clientPort</name>
        <value>2181</value>
    </property>

```

- hbase.rootdir: HDFS의 /hbase 디렉터리를 생성해 데이터베이스 파일 관리
- hbase.zookeeper.quorum: 주키퍼 쿼럼 설정을 위해 주키퍼를 설정한 서버 입력
- hbase.zookeeper.property.dataDir: 주키퍼의 데이터가 있는 경로 지정
- hbase.cluster.distributed: HBase를 완전 분산으로 설치
- dfs.replication: 복제계수를 1로 설정
- hbase.zookeeper.property.clientPort: 주키퍼 클라이언트 포트는 2181

- regionservers을 열어 Region서버를 지정함

```
$ vi regionservers
```

```

server01
server02
server03
server04

```

- Hbase 파일을 server02, server03, server04에 전송 후 압축 풀기

```
$ tar cvfz hbase.tar.gz hbase-1.2.6.1
```

```
$ scp hbase.tar.gz bigdata@server02:/home/bigdata
$ scp hbase.tar.gz bigdata@server03:/home/bigdata
$ scp hbase.tar.gz bigdata@server04:/home/bigdata

$ ssh bigdata@server02 "cd /home/bigdata; tar xvfz hbase.tar.gz"
$ ssh bigdata@server03 "cd /home/bigdata; tar xvfz hbase.tar.gz"
$ ssh bigdata@server04 "cd /home/bigdata; tar xvfz hbase.tar.gz"
```

- server02,03,04 모두 hbase 심볼릭 링크 설정

```
$ ln -s hbase-1.2.6.1 hbase
```

- server02,03,04 모두 Hbase의 Pid 정보를 저장하는 디렉토리 생성

```
$ mkdir /home/bigdata/var/hbase
```

- server01,server02,server03에 bigdata 계정으로 접속후 주키퍼 실행

```
$ ./zookeeper-3.4.12/bin/zkServer.sh start
```

- server01에서 하둡 시작

```
$ ./hadoop-2.9.1/sbin/start-all.sh
```

- server01에 bigdata 계정으로 접속 후 hbase master 실행

```
$ start-hbase.sh
```

```
[bigdata@server01 ~]$ start-hbase.sh
starting master, logging to /home/bigdata/hbase/logs/hbase-bigdata-master-server01.out
server04: starting regionserver, logging to /home/bigdata/hbase/bin/../logs/hbase-bigdata-regionserver-server04.out
server03: starting regionserver, logging to /home/bigdata/hbase/bin/../logs/hbase-bigdata-regionserver-server03.out
server02: starting regionserver, logging to /home/bigdata/hbase/bin/../logs/hbase-bigdata-regionserver-server02.out
server01: starting regionserver, logging to /home/bigdata/hbase/bin/../logs/hbase-bigdata-regionserver-server01.out
```

- 만약 실행이 안되면 먼저 hbase_home/logs 에서 로그를 확인해 원인을 알아야함
 - 아래의 두 가지는 HBase 실행 실패에 빈번하게 발생할 수있는 상황
- ① Server01의 Namenode가 standby이고 Server02의 Namenode가 active이면 작동이 안될 수 있으므로 수동으로 바꿔줌

```
$ hdfs haadmin -failover nn2 nn1
```

- ② hadoop01 safe mode로 들어갔을 수 있기 때문에 풀어줘야함

```
$ hdfs dfsadmin -safemode leave
```

- 각 서버의 JVM을 확인
 - Server01: HMaster, HRegionServer
 - Server02: HRegionServer
 - Server03: HRegionServer
 - Server04: HRegionServer

```
$ jps
```

```
[bigdata@server01 ~]$ jps
4304 DFSZKFailoverController
7715 HRegionServer
3781 NameNode
3893 DataNode
7573 HMaster
8134 Jps
4423 ResourceManager
4121 JournalNode
4555 NodeManager
3565 QuorumPeerMain
```

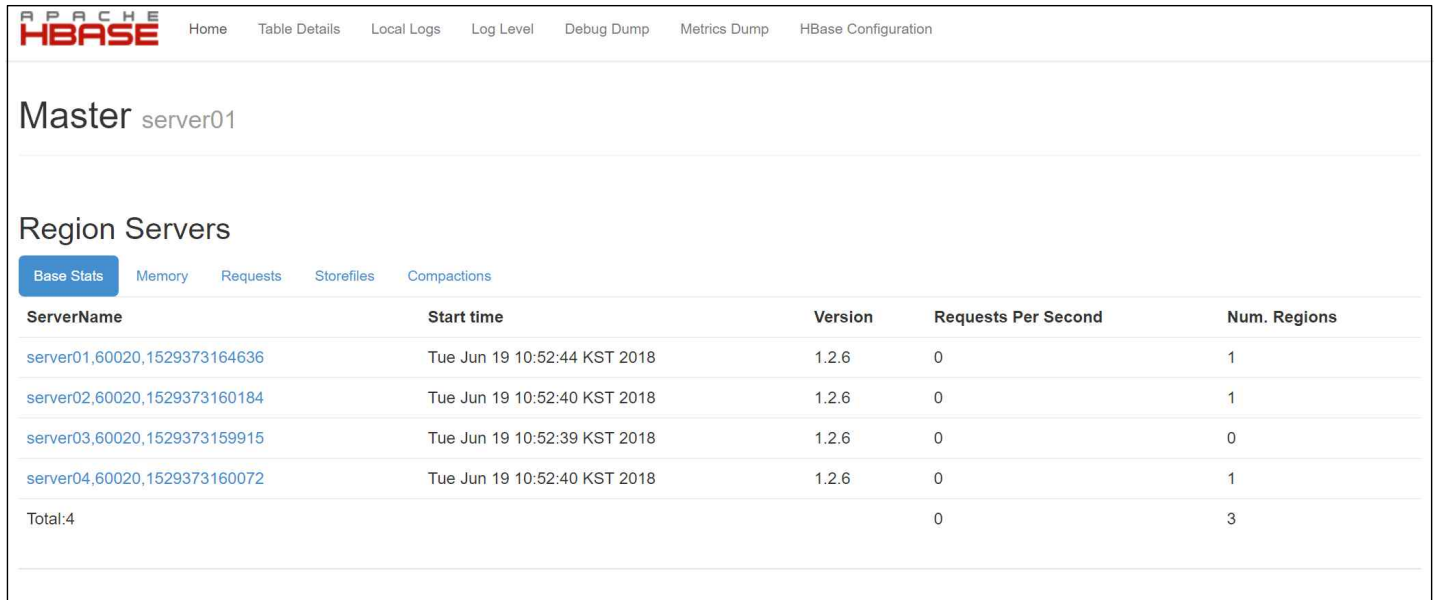
```
[bigdata@server02 ~]$ jps
3090 DFSZKFailoverController
4131 Jps
2788 NameNode
3191 NodeManager
2970 JournalNode
2716 QuorumPeerMain
2860 DataNode
3901 HRegionServer
```

```
[bigdata@server03 ~]$ jps
2802 DataNode
3570 HRegionServer
3012 NodeManager
2905 JournalNode
3738 Jps
2716 QuorumPeerMain
```

```
[bigdata@server04 ~]$ jps
3584 Jps
2740 DataNode
3412 HRegionServer
2863 NodeManager
```


- HBase 사용자 UI확인

http://server01:16010



ServerName	Start time	Version	Requests Per Second	Num. Regions
server01,60020,1529373164636	Tue Jun 19 10:52:44 KST 2018	1.2.6	0	1
server02,60020,1529373160184	Tue Jun 19 10:52:40 KST 2018	1.2.6	0	1
server03,60020,1529373159915	Tue Jun 19 10:52:39 KST 2018	1.2.6	0	0
server04,60020,1529373160072	Tue Jun 19 10:52:40 KST 2018	1.2.6	0	1
Total:4			0	3

- Server01에 bigdata 계정으로 접속
- hbase shell을 이용해 hbase 작동 테스트

```
$ hbase shell
```

```
[bigdata@server01 ~]$ hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/bigdata/hbase-1.2.6/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/bigdata/hadoop-2.9.1/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017
```

- 테이블 리스트 출력

```
hbase(main):001:0> list
```

```
hbase(main):001:0> list
TABLE
0 row(s) in 0.4060 seconds

=> []
```

- Hbase 상태 출력

```
hbase(main):002:0> status
```

```
hbase(main):002:0> status
1 active master, 0 backup masters, 4 servers, 0 dead, 0.7500 average load
```

- HTable을 생성

```
hbase(main):003:0> create 'chat_table', 'message', 'location'
```

- 테이블명: **chat_table**
- 컬럼 패밀리: **message, locaiton**

- 'chat_table'에 데이터 입력

```
hbase(main):004:0> put 'chat_table', '1010101', 'message:name', 'pilgu'
```

- **put**: 특정 rowkey에 해당하는 데이터를 추가
- put '테이블명', 'rowkey', '컬럼(컬럼패밀리:컬럼 수식어)', '셀 값'

- 'chat_table'에 확인

```
hbase(main):005:0> scan 'chat_table'
```

- **scan**: 특정 범위의 rowkey에 속하는 row 목록접근
- scan '테이블명'

```
hbase(main):023:0> scan 'chat_table'
ROW                                COLUMN+CELL
1010101                            column=message:name, timestamp=1529475318314, value=pilgu
1 row(s) in 0.0190 seconds
```

- 셀이 출력되는 것을 볼 수 있음

1010101 로우키 | message 컬럼패밀리 | name 컬럼수식어 | 타임스탬프 | pilgu 값

- 'chat_table'에 데이터를 더 넣어보고 확인

```
> put 'chat_table', '1010101', 'message:message', 'hello'
> put 'chat_table', '1010101', 'location:country', 'Korea'
> put 'chat_table', '1010101', 'location:city', 'Seoul'
> put 'chat_table', '1010102', 'message:name', 'wallE'
> put 'chat_table', '1010102', 'message:message', 'i am wall-E~'
> put 'chat_table', '1010102', 'location:country', 'USA'
> put 'chat_table', '1010102', 'location:city', 'California'
```

```
> scan 'chat_table'
```

```
hbase(main):027:0> scan 'chat_table'
ROW                                COLUMN+CELL
1010101                            column=location:city, timestamp=1529475770736, value=Seoul
1010101                            column=location:country, timestamp=1529475769486, value=Korea
1010101                            column=message:message, timestamp=1529475769440, value=hello
1010101                            column=message:name, timestamp=1529475318314, value=pilgu
1 row(s) in 0.0680 seconds
```

```
hbase(main):022:0> scan 'chat_table'
ROW                                COLUMN+CELL
1010101                            column=location:city, timestamp=1529479614356, value=Seoul
1010101                            column=location:country, timestamp=1529479614286, value=Korea
1010101                            column=message:message, timestamp=1529479614259, value=hello
1010101                            column=message:name, timestamp=1529479584524, value=pilgu
1010102                            column=location:city, timestamp=1529479615413, value=California
1010102                            column=location:country, timestamp=1529479614415, value=USA
1010102                            column=message:message, timestamp=1529479614396, value=i am wall-E~
1010102                            column=message:name, timestamp=1529479614374, value=wallE
2 row(s) in 0.0310 seconds
```

- 'chat_table'의 대화내용 모두 출력

```
> scan 'chat_table', {COLUMNS=>['message:message']}
```

```
hbase(main):031:0> scan 'chat_table', {COLUMNS=>['message:message']}  
ROW                                COLUMN+CELL  
1010101                            column=message:message, timestamp=1529477368870, value=hello  
1010102                            column=message:message, timestamp=1529477405525, value=i am wall-E~  
2 row(s) in 0.0620 seconds
```

■ 하이브와 Hbase 연동

- server01을 bigdata 계정으로 접속해 hive 켜기

```
$ hive
```

- HBase의 테이블과 연동하기 위해 Hive 외부테이블 생성

```
CREATE EXTERNAL TABLE chat_table(rowkey string, name string, message string, country  
string, city string)  
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'  
WITH SERDEPROPERTIES("hbase.columns.mapping" = ":key, message:name, message:message,  
location:country, location:city")  
TBLPROPERTIES("hbase.table.name"="chat_table");
```

- EXTERNAL TABLE: Hbase의 값을 참조하기 때문에 외부테이블로 선언
- SERDEPROPERTIES: 하이브의 컬럼과 hbase의 컬럼과 매핑 설정
- TBLPROPERTIES: 접근할 hbase 테이블 설정

- 데이터 확인

```
select * from chat_table;
```

```
1010101 pilgu    hello    Korea    Seoul
1010102 wallE   i am wall-E~    USA      California
Time taken: 4.544 seconds, Fetched: 2 row(s)
```

- Hive에서 HBase로 데이터를 로딩

- Hbase로 데이터를 로딩하기 위한 테이블 만들기

```
CREATE TABLE chat_table2(rowkey string, name string, message string, country string,
city string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

- /home/bigdata/working 경로에 chat_table2에 넣을 test.csv를 생성

```
$ vi test.csv
```

```
1010103,guhyun,안녕하세요~,Korea,Seoul
1010104,pilgu,식사는 하셨나요?,Korea,Seoul
1010105,wallE,nice to meet you too,USA,California
1010106,guhyun,네 점심 맛있게 먹었어요,Korea,Seoul
```

- hive에서 chat_table2에 데이터 로드

```
load data local inpath '/home/bigdata/working/test.csv'
into table chat_table2;
```

- 데이터 확인

```
select * from chat_table2;
```

1010103	guhyun	안녕하세요~	Korea	Seoul
1010104	pilgu	식사는 하셨나요?	Korea	Seoul
1010105	wallE	nice to meet you too	USA	California
1010106	guhyun	네 점심 맛있게 먹었어요	Korea	Seoul

- 만들어둔 HBase 테이블인 chat_table에 chat_table2를 로드

```
INSERT OVERWRITE TABLE chat_table
SELECT * FROM chat_table2;
```

- HBase shell을 열어 chat_table을 확인

```
$ hbase shell
```

```
> scan 'chat_table'
```

1010103	column=location:city, timestamp=1529556349596, value=Seoul
1010103	column=location:country, timestamp=1529556349596, value=Korea
1010103	column=message:message, timestamp=1529556349596, value=\xEC\x95\x88\xEB\x85\x95\xED\x95\x98\xEC\x84\xB8\xEC\x9A\x94~
1010103	column=message:name, timestamp=1529556349596, value=guhyun
1010104	column=location:city, timestamp=1529556349596, value=Seoul
1010104	column=location:country, timestamp=1529556349596, value=Korea
1010104	column=message:message, timestamp=1529556349596, value=\xEC\x8B\x9D\xEC\x82\xAC\xEB\x8A\x94 \xED\x95\x98\xEC\x85\xA8\xEB\x82
1010104	column=message:name, timestamp=1529556349596, value=pilgu
1010105	column=location:city, timestamp=1529556349596, value=California
1010105	column=location:country, timestamp=1529556349596, value=USA
1010105	column=message:message, timestamp=1529556349596, value=nice to meet you too
1010105	column=message:name, timestamp=1529556349596, value=wallE
1010106	column=location:city, timestamp=1529556349596, value=Seoul
1010106	column=location:country, timestamp=1529556349596, value=Korea
1010106	column=message:message, timestamp=1529556349596, value=\xEB\x84\xA4 \xEC\xA0\x90\xEC\x8B\xAC \xEB\xA7\x9B\xEC\x9E\x88\xEA\xE
1010106	column=message:name, timestamp=1529556349596, value=guhyun

- 값이 들어간 것을 확인 할 수 있음
- hbase는 바이트 배열로 값(Value)이 들어감
- 한글이 값(Value)으로 들어갈 때는 16진수로 출력됨

- chat_table을 삭제

```
> disable 'chat_table'
```

- 삭제를 하려면 테이블을 비활성화 시킴

```
> drop 'chat_table'
```

- 테이블 삭제