[SK BigData 5Group Tutorial] 3rd day ----------------------------------------------------------------------------

Life is exam !

[SK BigData 5Group Tutorial] 2nd day ----------------------------------------------------------------------------

1. Business Table

Business table 의 categories column을 explode해서 Restaurants가 포함된 레코드만 추출한 쿼리 작성

|  |
| --- |
| with restaurants as (  select \* from business lateral view explode(categories) exploded as cat\_exploded where cat\_exploded="Restaurants"  )  select name, review\_count, stars, cat\_exploded from restaurants limit 5 |

Attributes -> ambience -> romantic이 true인 레코드 검색 (새로 매핑한 테이블 이용)

|  |
| --- |
| with restaurants as (  select \* from business3 lateral view explode(categories) exploded as cat\_exploded where cat\_exploded="Restaurants"  )  ,  ambience\_romantic\_tbl as (  select \*, attributes.ambience.romantic ambi\_romantic from restaurants  )  select \* from ambience\_romantic\_tbl where ambi\_romantic=true  limit 5; |

Restaurants 개수

|  |
| --- |
| with restaurants as (  select \* from business3 lateral view explode(categories) exploded as cat\_exploded where cat\_exploded="Restaurants"  )  select count(\*) from restaurants |

Json을 그대로 Table로 생성할경우(가변 스키마일 경우 사용함)

|  |
| --- |
| CREATE EXTERNAL TABLE yelp2.`business2`  (json\_string STRING)  LOCATION 's3://yelp-sk-test/yelp/business'; |

JsonString으로 된 Table에 필드를 조회할 경우

|  |
| --- |
| drop table yelp2.`business3` ;  CREATE TABLE yelp2.business3  STORED AS parquet  AS select get\_json\_object(json\_string, '$.business\_id') business\_id  ,get\_json\_object(json\_string, '$.name') name  ,get\_json\_object(json\_string, '$.neighborhood') neighborhood  ,get\_json\_object(json\_string, '$.address') address  ,get\_json\_object(json\_string, '$.city') city  ,get\_json\_object(json\_string, '$.state') state  ,get\_json\_object(json\_string, '$.postal\_code') postal\_code  ,cast(get\_json\_object(json\_string, '$.latitude') as double) latitude  ,cast(get\_json\_object(json\_string, '$.longitude') as double) longitude  ,cast(get\_json\_object(json\_string, '$.stars') as int) stars  ,cast(get\_json\_object(json\_string, '$.review\_count') as int) review\_count  ,cast(get\_json\_object(json\_string, '$.is\_open') as int) is\_open  ,get\_json\_object(json\_string, '$.attributes') attributes  ,get\_json\_object(json\_string, '$.categories') categories  ,get\_json\_object(json\_string, '$.hours') hours  from yelp2.business2 |

Review를 date, bussiness\_id로 Group by를 수행한 결과를 신규 테이블로 생성

|  |
| --- |
| create table yelp2.review\_year\_count  as select business\_id, substr(`date`,1,4) year, count(\*) from review group by business\_id , substr(`date`, 1, 4) |

Year별 restaurant의 review count

|  |
| --- |
| select \* from yelp2.review\_year\_count inner join restaurant on restaurant.business\_id = review\_year\_count.business\_id  limit 10; |

Review\_filtered table 생성

|  |
| --- |
| create table review\_filtered\_in STORED AS parquet as  select r.\*, v.review\_id, v.user\_id, v.stars review\_stars, v.`date`, v.text, v.useful, v.funny, v.cool from restaurants\_in r join review\_in v on r.business\_id = v.business\_id; |

* Join 성능을 고려할때 레코드 수가 적은 restaurants 테이블을 왼쪽에 쓰면 좋다. 하지만 query analyser가 어떻게 쓰던 restaurants 테이블을 왼쪽에 위치하게 변경해서 실제 쿼리를 수행한다. (실행계획 참고, 테이블 통계가 정확하지 않은 경우 다르게 동작할 수 있음)

Restaurants 의 review count

|  |
| --- |
| select count(\*) from review\_filtered\_in  -> 3221419 |

User distinct count

|  |
| --- |
| select count(distinct user\_id) from `user`;  -> 1326101 |

user테이블의 상위 4개 레코드의 elite필드를 조회(array<string> 필드의 갯수가 1 이상인경우)

|  |
| --- |
| select elite from yelp2.`user` limit 4;  Select elite from yelp2.`user` where size(elite) > 0 limit 4; |

user테이블에서 array<string> 필드인 elite를 explode해서 신규 테이블 생성

|  |
| --- |
| create table yelp2.elite\_users  as select `user`.\*, adTable.elite\_year from yelp2.`user` lateral view explode(elite) adTable as elite\_year; |

state 별 restaurants 갯수 시각화

|  |
| --- |
| select state, count(business\_id) number\_restaurants from restaurants\_in group by state; |
|  |

[참고]경위도 값을 이용한 MarkerMap 작성

|  |
| --- |
| MarkerMap 선택, 경위도값 설정 |

도시별 레스토랑 개수

|  |
| --- |
| select city, count(business\_id) number\_restaurants from restaurants\_in group by city order by number\_restaurants desc; |
|  |

Restaurants, Food를 제외한 category 카운트

|  |
| --- |
| select cat\_exp, count(\*) cnt from restaurants\_in lateral view explode(categories) exploded as cat\_exp  where cat\_exp not in ('Restaurants','Food') group by cat\_exp order by cnt desc; |
|  |

Star rating 과 category

|  |
| --- |
| select cat\_exp, stars, count(\*) cnt from restaurants\_in lateral view explode(categories) exploded as cat\_exp  where cat\_exp not in ('Restaurants','Food') group by cat\_exp, stars order by cnt desc limit 10; |
|  |

Illinois 의 star rating이 가장 높은 100개 restaurants

|  |
| --- |
| select name, stars, review\_count, latitude, longitude  from restaurants\_in where state='IL'  order by stars desc, review\_count desc limit 100; |
|  |

Arizona state에서 Star rating 이 낮은 restaurants heap map

|  |
| --- |
| select name, stars, review\_count, latitude, longitude  from restaurants\_in where state='AZ'  order by stars asc limit 100; |
|  |

What ratings do the majority of restaurants have?

|  |
| --- |
| select stars, count(\*) from restaurants\_in group by stars; |
|  |

Rating distribution in restaurant reviews

|  |
| --- |
| select review\_stars, round((count(review\_stars)\*100) / (sum(count(review\_stars)) over()), 2) ratio  from review\_filtered\_in group by review\_stars; |
|  |

good\_reviews

|  |
| --- |
| with restaurants\_cat as (  select cat\_exp, business\_id from restaurants\_in lateral view explode(categories) exploded as cat\_exp  where cat\_exp not in ('Restaurants','Food')  ),  rest\_rev as (  select r.cat\_exp, avg(v.stars) rat, count(\*) cnt  from restaurants\_cat r join restaurants\_in v on r.business\_id = v.business\_id  where v.stars >= 4  group by r.cat\_exp  )  select cat\_exp, rat, cnt from rest\_rev j  where cnt > 1000  order by cnt desc limit 100; |
|  |

|  |
| --- |
| with restaurants\_cat as (  select cat\_exp, business\_id from restaurants\_in lateral view explode(categories) exploded as cat\_exp  where cat\_exp not in ('Restaurants','Food')  ),  rest\_rev as (  select r.cat\_exp, avg(v.stars) rat, count(\*) cnt  from restaurants\_cat r join restaurants\_in v on r.business\_id = v.business\_id  where v.stars <= 2  group by r.cat\_exp  )  select cat\_exp, rat, cnt from rest\_rev j  order by cnt desc limit 100; |
|  |

8.Top 15 restaurants which have the most reviews

|  |
| --- |
| select r.name, r.city, r.review\_count, r.stars, r.attributes.restaurantspricerange2  from restaurants2 r  order by r.review\_count desc limit 15; |

9-a.Average rating by all users

|  |
| --- |
| select avg(average\_stars) from users;  Result > 3.7108414064992257 |

9-b.Average rating by all users

|  |
| --- |
| select avg(average\_stars) from elite\_users;  Result > 3.82042693386773 |

9-c. Count number of elite users by year

|  |
| --- |
| select elite\_exploded , count(1) as cnt from elite\_users  group by elite\_exploded; |

9-d. Count average reviews by elite users by year:

|  |
| --- |
| select elite\_exploded, avg(average\_stars) from elite\_users  group by elite\_exploded; |

[응용1] 1000개 이상 리뷰 중 100대 레스토랑 분포

|  |
| --- |
| CREATE TABLE restaurant\_repute  STORED AS parquet  AS select business\_id, count(1) as rv\_cnt, avg(review\_stars) as stars\_avg, max(review\_stars) as stars\_max, min(review\_stars) as stars\_min, stddev(review\_stars) as stars\_stddev from review\_filtered  group by business\_id;  select a.stars\_avg, b.longitude, b.latitude from restaurant\_repute a, restaurant b  where stars\_avg > 4.0 and rv\_cnt >1000  and a.business\_id = b.business\_id  order by stars\_avg desc; |

[응용2] 1000개 이상 리뷰 중 분산 정보

|  |
| --- |
| 4.5 이상의 레스토랑의 평점은 분산도 상대적으로 적어, 신뢰도를 높다고 판단할 수 있음.  select business\_id,stars\_avg, stars\_stddev, rv\_cnt from restaurant\_repute  where rv\_cnt>1000  order by stars\_avg ; |

[SK BigData 5Group Tutorial] 1st day -----------------------------------------------------------------------------

1. 개발환경 구축

1. Yelp data download (USB)

<https://www.yelp.com/dataset/download> (2.47G)

압축해제 : 6개의 json 파일 포함 (6.07G)



2) Gitbash download

<http://www.techoism.com/how-to-install-git-bash-on-windows/>

3) AWS CLI download

<https://s3.amazonaws.com/aws-cli/AWSCLI64.msi>

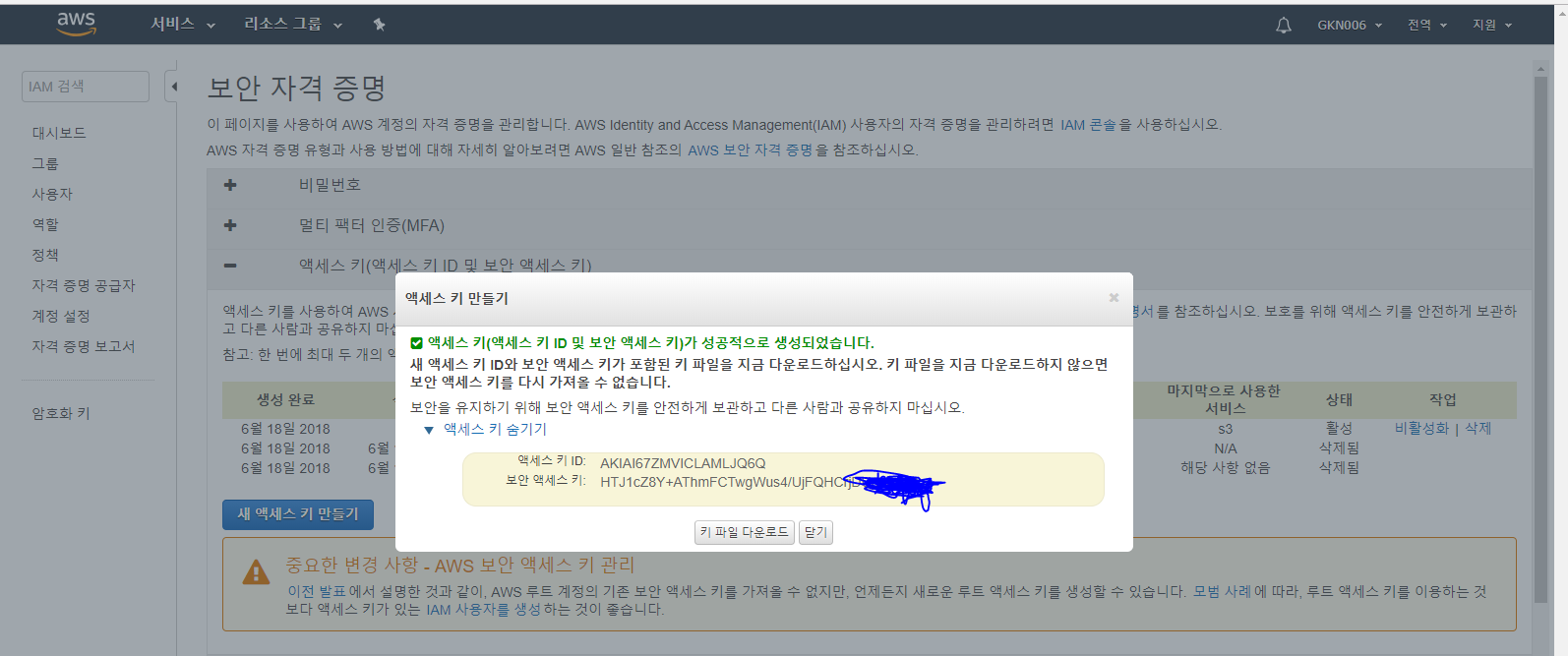
2. Key 생성

1. 화면상단 우측 계정명 > 내보안자격증명 > 액세스 키 > 새 액세스 키 만들기

ㄱ. 최대 2개의 키만 생성 가능

ㄴ. 생성시 자동으로 “액세스 키 ID”와 “액세스 키”가 생성됨

ㄷ. 최초 생성시 아래와 같이 값 확인 및 “키 파일 다운로드” 버튼 생성



ㄹ. 키파일 다운 로드 시 Down 폴더 저장: C:\Users\gknuser\Downloads\rootkey.csv (예)

3. AWS Configure to connect to AWS:

|  |
| --- |
| $ aws configure  AWS Access Key ID [None]: \*\*\*\*\*\*\*\*\*\*\*\*  AWS Secret Access Key [None]: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Default region name [None]:  Default output format [None]: |

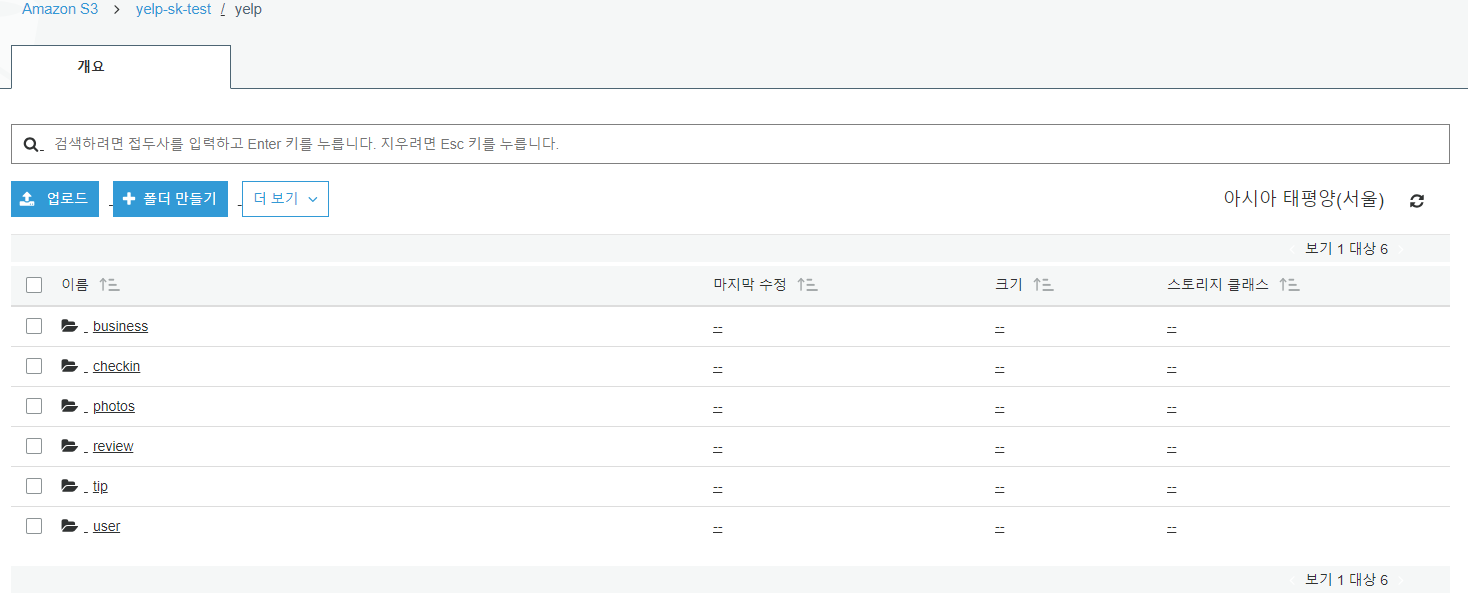
4. 데이터 적재 ( local - S3)

1) S3 서비스 > 버킷만들기

2) 6개의 json 파일 upload

|  |
| --- |
| $ aws s3 ls  $ aws s3 cp photos.json s3://yelp/photos  upload: .\photos.json to s3://yelp/photos/photos.json  $ aws s3 cp review.json s3://yelp/review  ++ |

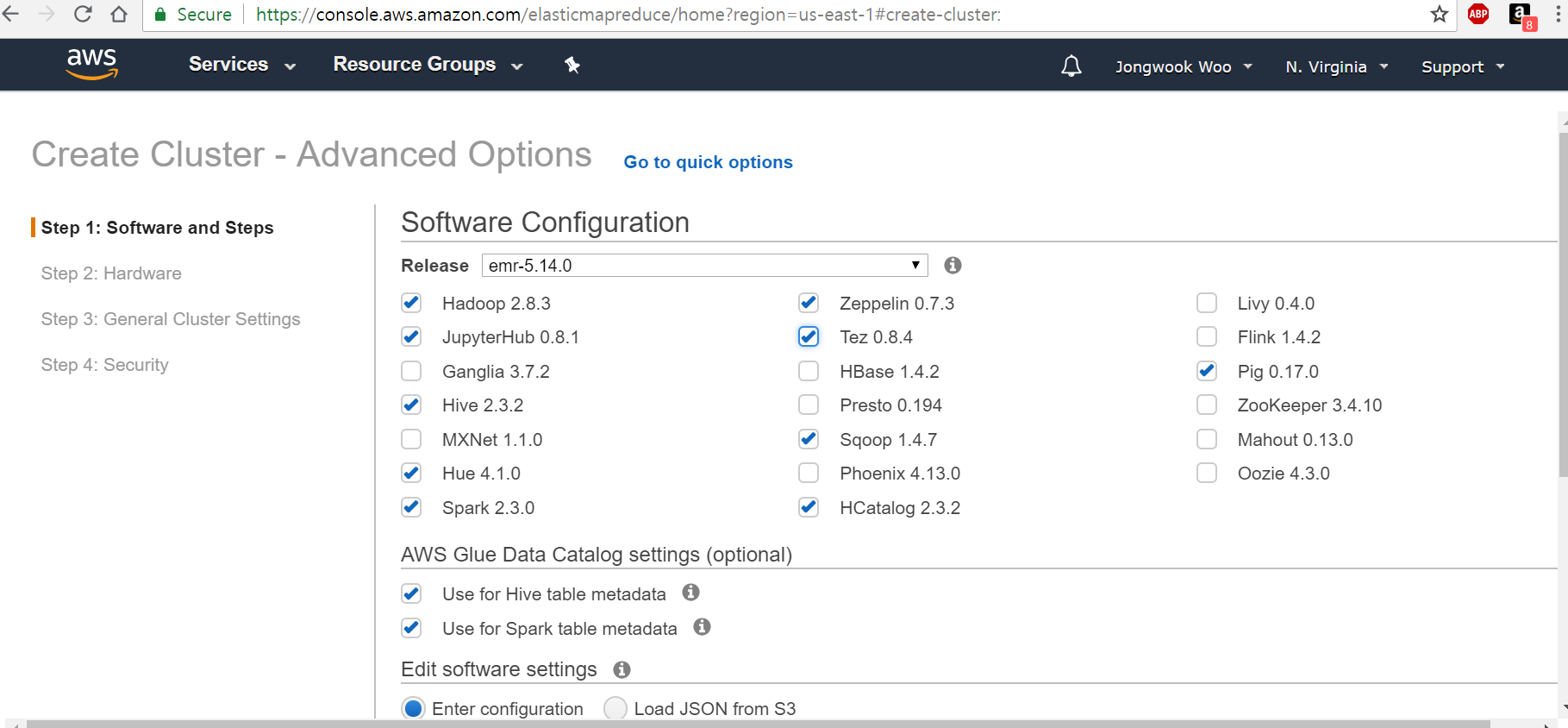
[결과]

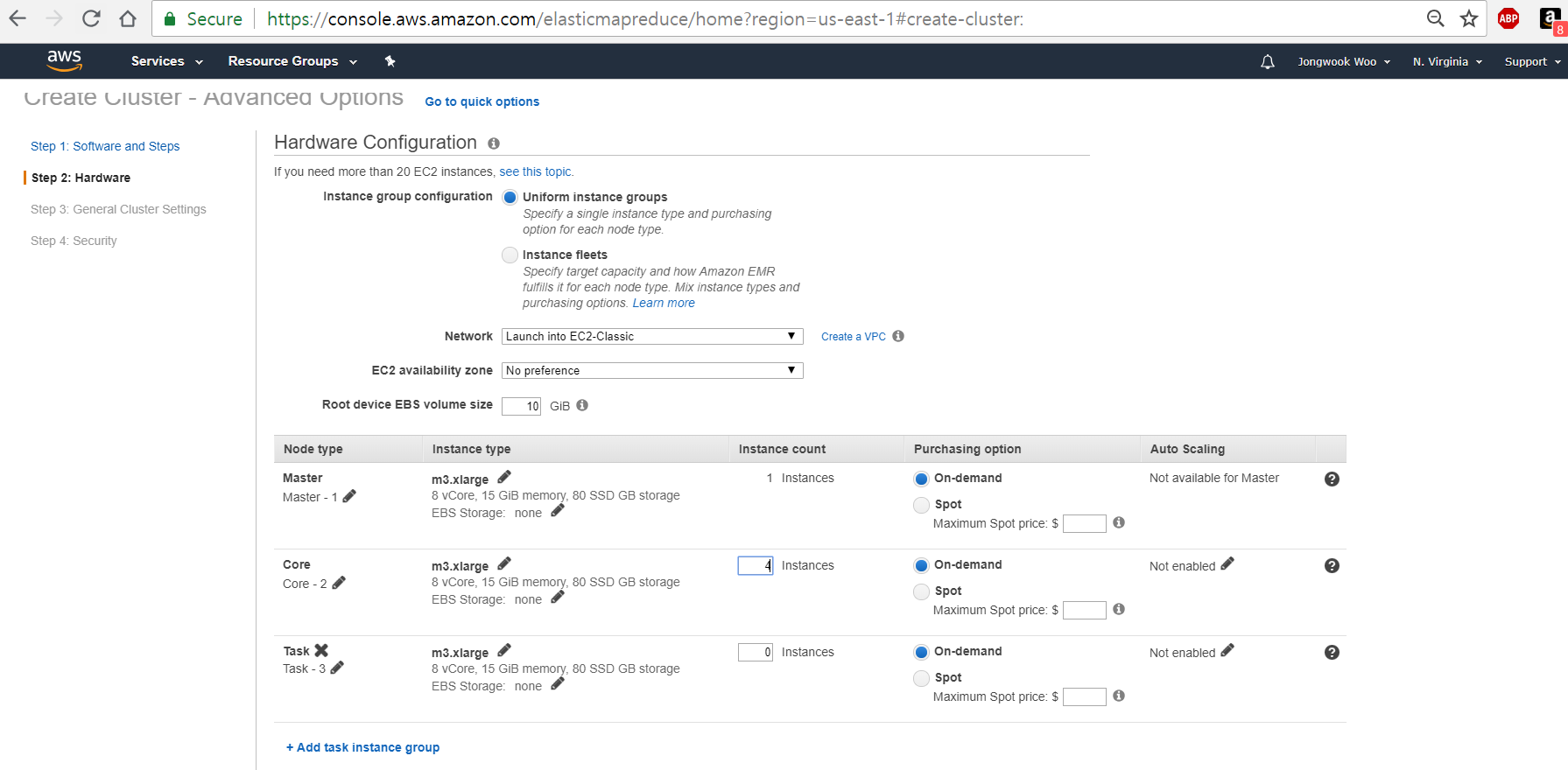


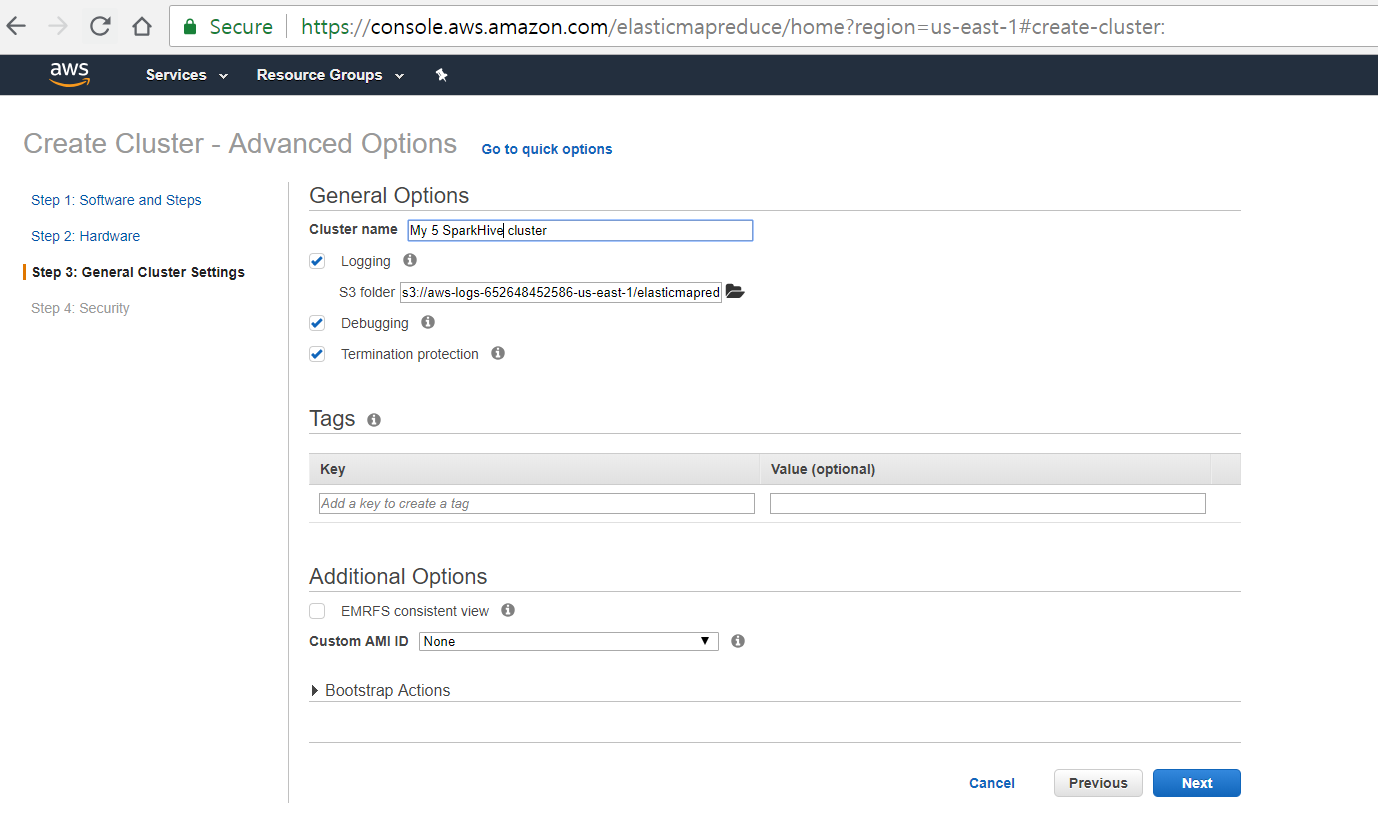
5. EMR 클러스터 생성

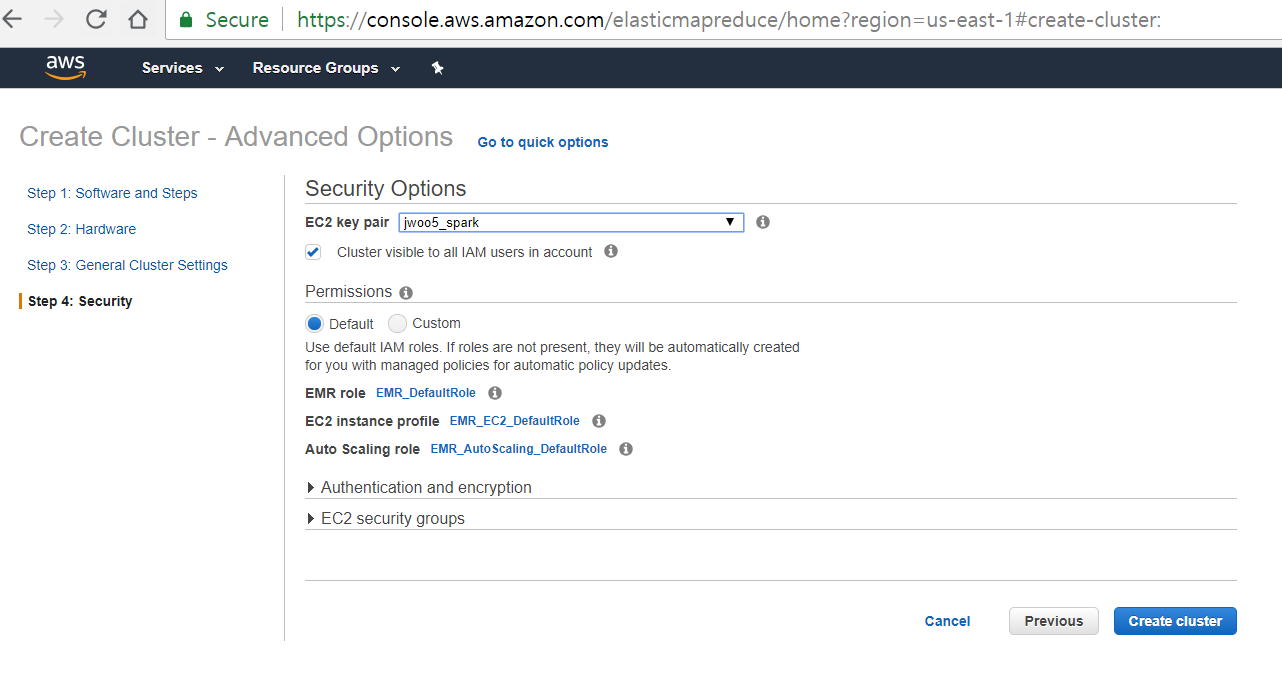
1. EMR 서비스
2. 클러스터 생성 > 고급 옵션

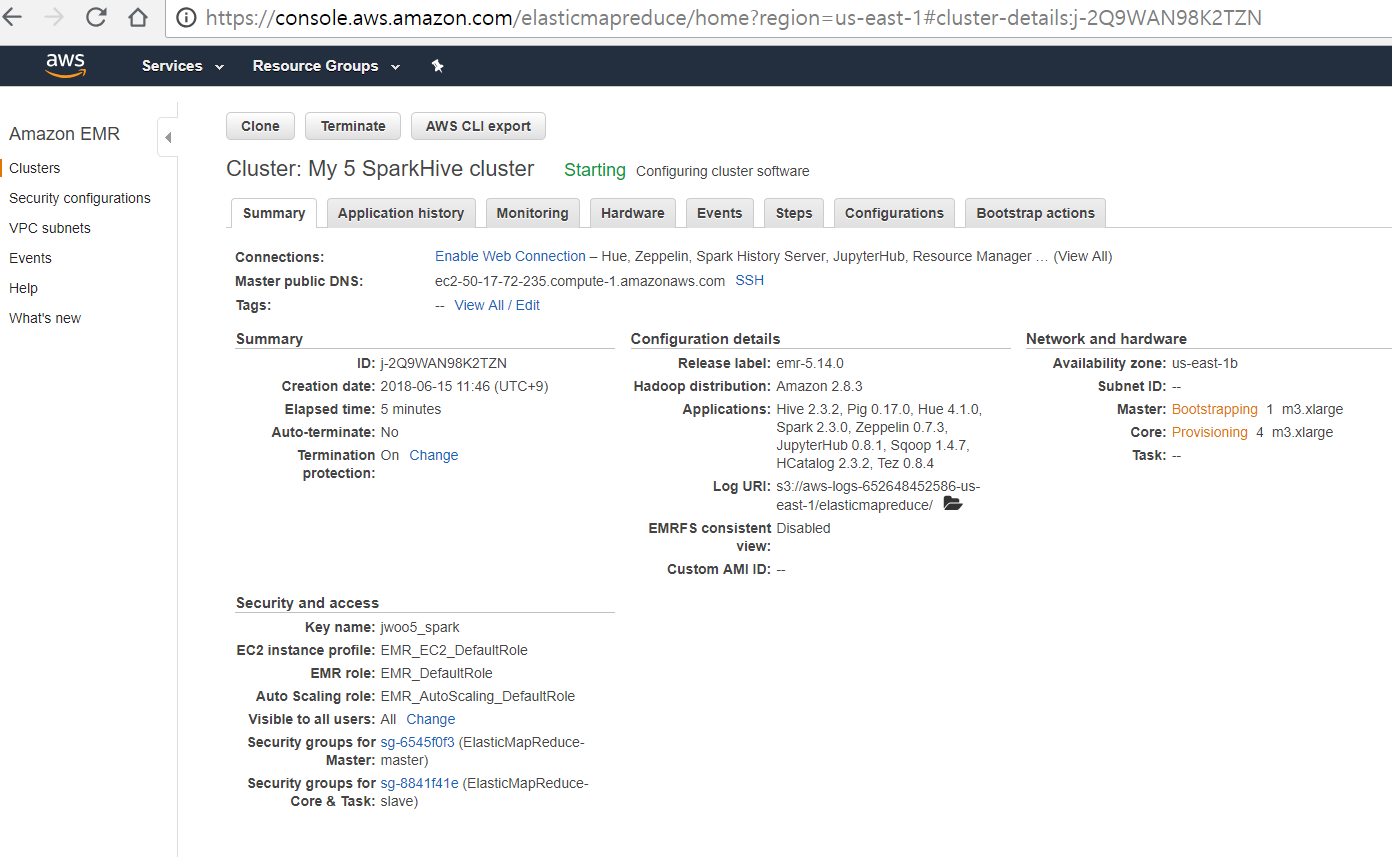
* Hue 옵션 선택









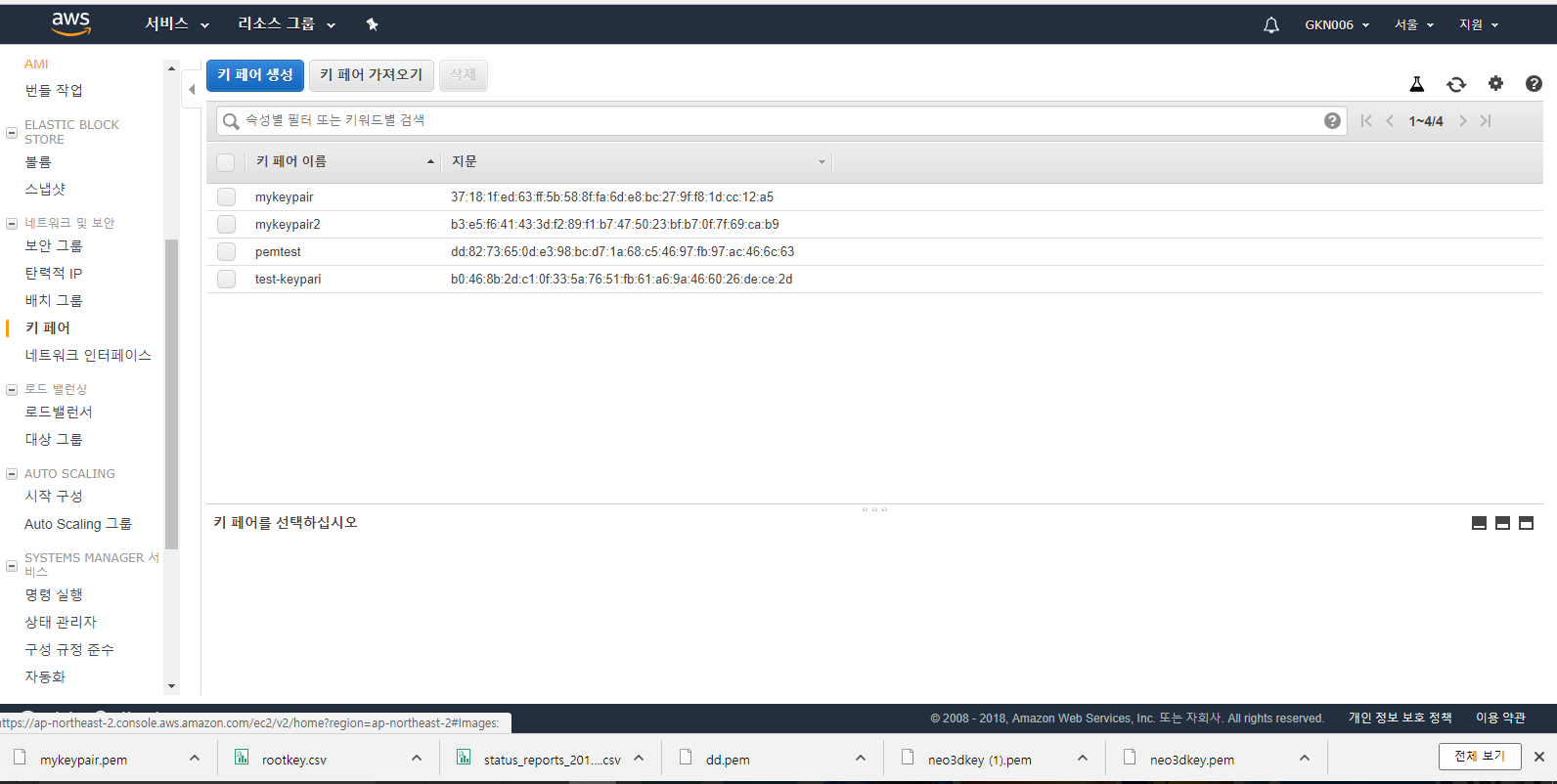


6. SSH 세팅

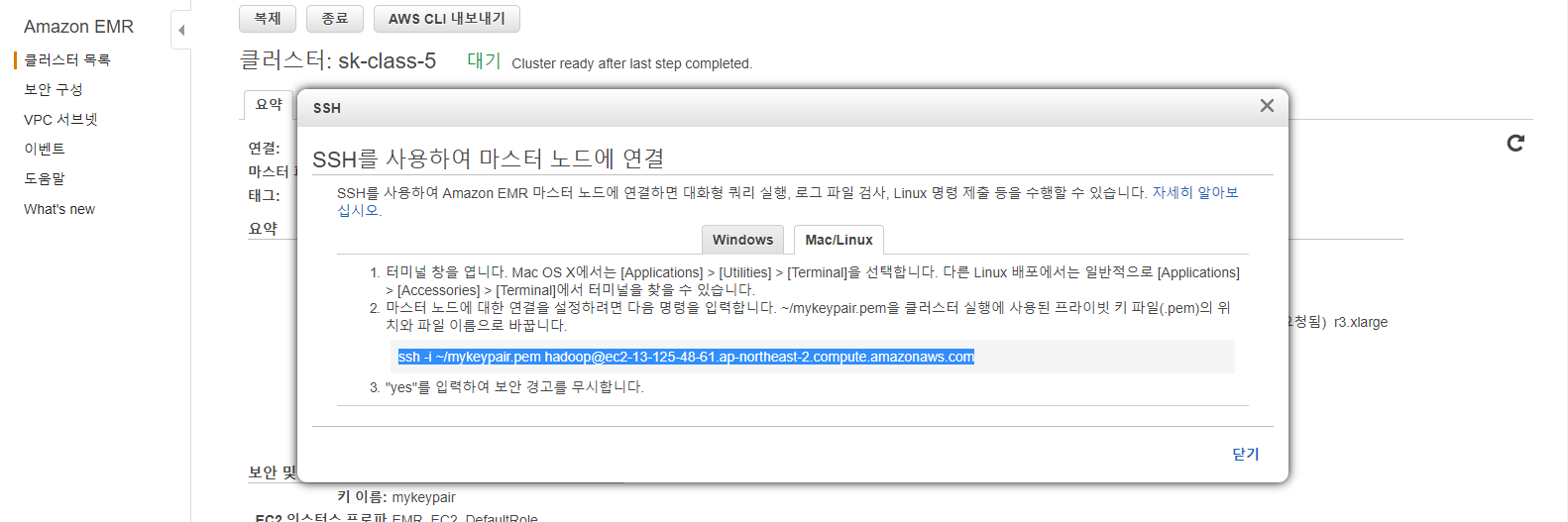
1. 키페어 생성 ( pem 파일 생성)

ㄱ. EC2 > 네트워크 및 보안> 키 페어 > 키 페어 생성

ㄴ. 최초 생성시에는 pem 파일이 자동으로 노컬 PC로 다운됨.



1. \*.pem 파일을 ~/.ssh 디렉토리로 복사
2. SSH 연동



|  |
| --- |
| $ ssh -i ~**/.ssh/**mykeypair.pem hadoop@ec2-13-125-48-61.ap-northeast-2.compute.amazonaws.com |

7. 터널링 세팅

1. Hue : Web Interface 환경에서 접근할 수 있도록 해주는 하둡 에코 시스템

|  |
| --- |
| $ ssh -i ~/.ssh/mykeypair.pem -L 8234:[ec2-13-125-48-61.ap-northeast-2.compute.amazonaws.com:8888](http://ec2-13-125-48-61.ap-northeast-2.compute.amazonaws.com:8888/) hadoop@ec2-13-125-48-61.ap-northeast-2.compute.amazonaws.com |

2) Zeppeline : 빅데이터 분석 및 시각화 도구

|  |
| --- |
| $ ssh -i ~/.ssh/mykeypair.pem -L 8235:[ec2-13-125-48-61.ap-northeast-2.compute.amazonaws.com:8890](http://ec2-13-125-48-61.ap-northeast-2.compute.amazonaws.com:8890/) hadoop@ec2-13-125-48-61.ap-northeast-2.compute.amazonaws.com |

8. Data Insert

|  |
| --- |
| Hive EXTERNAL Table : 외부테이블은 hive.metastore.warehouse.dir 속성이 가리키는 디렉터리에 데이터를 저장하지 않고, 테이블 생성 시 설정한 경로(source data 경로)로 데이터를 저장, 사용자가 실수로 테이블을 DROP 했더라도 데이터가 보존되는 장점이 있다.  serde .... delimited와 같이 다른 설정으로 행 포맷을 설정할 수 있습니다  Internal Table : 내부 테이블은 Warehouse 폴더에 저장, 내부 테이블을 삭제하면 Schema와 Warehouse 폴더 아래의 데이터가 삭제됨. 일시적인 데이터 분석이나 Hive가 테이블과 데이터의 생명주기를 완벽하게 관리하려 할 때 사용  (HDFS 파일로 변경 확인 가능) |

|  |
| --- |
| CREATE EXTERNAL TABLE `yelp2.review`(  `review\_id` string,  `user\_id` string ,  `business\_id` string ,  `stars` int ,  `date` string ,  `text` string ,  `useful` int ,  `funny` int ,  `cool` int )  ROW FORMAT SERDE  'org.apache.hive.hcatalog.data.JsonSerDe'  LOCATION  's3://yelp-sk-test/yelp/review' |

|  |
| --- |
| create external table yelp2.checkin (  business\_id string, `time` map<string, map <string, int>>  )  ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'  LOCATION 's3://yelp-sk-test/yelp/checkin'; |

|  |
| --- |
| CREATE EXTERNAL TABLE IF NOT EXISTS business3 (  address string,  business\_id string,  categories array<string>,  city string,  hours struct<friday:string, monday:string, saturday:string, sunday:string, thursday:string, tuesday:string, wednesday:string>,  is\_open int,  latitude double,  longitude double,  name string,  neighborhood string,  postal\_code string,  review\_count int,  stars double,  state string,  Attributes struct<  Accepts\_Insurance:boolean,  Ages\_Allowed:string,  Alcohol:string,  Bike\_Parking:boolean,  Business\_Accepts\_Bitcoin:boolean,  Business\_Accepts\_Credit\_Cards:boolean,  By\_Appointment\_Only:boolean,  Byob:boolean,  BYOB\_Corkage:string,  Caters :boolean,  Coat\_Check:boolean,  Corkage:boolean,  Dogs\_Allowed:boolean,  Drive\_Thru:boolean,  Good\_For\_Dancing:boolean,  Good\_For\_Kids:boolean,  Happy\_Hour:boolean,  Has\_TV:boolean,  Noise\_Level:string,  Open24Hours:boolean,  Outdoor\_Seating:boolean,  Restaurants\_Attire:string,  Restaurants\_Counter\_Service:boolean,  Restaurants\_Delivery:boolean,  Restaurants\_Good\_For\_Groups:boolean,  Restaurants\_Reservations:boolean,  Restaurants\_Table\_Service:boolean,  Restaurants\_Take\_Out:boolean,  Smoking:string,  WheelchairAccessible:boolean,  WiFi:string,  Ambience:struct<  Casual:boolean,  Classy:boolean,  Divey:boolean,  Hipster:boolean,  Intimate:boolean,  Romantic:boolean,  Touristy:boolean,  Trendy:boolean,  Upscale:boolean>,  BestNights:struct<  Friday1:boolean,  Monday1:boolean,  Saturday1:boolean,  Sunday1:boolean,  Thursday1:boolean,  Tuesday1:boolean,  Wednesday1:boolean>,  BusinessParking:struct<  Garage:boolean,  Lot:boolean,  Street:boolean,  Valet:boolean,  Validated:boolean>,  DietaryRestrictions:struct<  Dairy\_Free:boolean,  Gluten\_Free:boolean,  Halal:boolean,  Kosher:boolean,  Soy\_Free:boolean,  Vegan:boolean,  Vegetarian:boolean>,  GoodForMeal:struct<  Breakfast:boolean,  Brunch:boolean,  Dessert:boolean,  Dinner:boolean,  Latenight:boolean,  Lunch:boolean>,  HairSpecializesIn:struct<  Africanamerican:boolean,  Asian:boolean,  Coloring:boolean,  Curly:boolean,  Extensions:boolean,  Kids:boolean,  Perms:boolean,  Straightperms:boolean>,  Music:struct<  BackgroundMusic:boolean,  Dj:boolean,  Jukebox:boolean,  Karaoke:boolean,  Live:boolean,  NoMusic:boolean,  Video:boolean>,  restaurantspricerange2:int>)  ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'  LOCATION 's3://yelp-sk-test/yelp/business'; |

|  |
| --- |
| CREATE EXTERNAL TABLE yelp\_yujin.tip (  text string,  `date` string,  likes int,  business\_id string,  user\_id string  )  ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'  LOCATION 's3://yelp-yujin/yelp/tip'; |

|  |
| --- |
| CREATE EXTERNAL TABLE yelp1.`user`(  `user\_id` string ,  `name` string ,  `review\_count` int ,  `yelping\_since` string ,  `friends` array<string> ,  `useful` int ,  `funny` int ,  `cool` int ,  `fans` int ,  `elite` array<int> ,  `average\_stars` double ,  `compliment\_hot` int ,  `compliment\_more` int ,  `compliment\_profile` int ,  `compliment\_cute` int ,  `compliment\_list` int ,  `compliment\_note` int ,  `compliment\_plain` int ,  `compliment\_cool` int ,  `compliment\_funny` int ,  `compliment\_writer` int ,  `compliment\_photos` int COMMENT 'from deserializer')  ROW FORMAT SERDE  'org.apache.hive.hcatalog.data.JsonSerDe'  STORED AS INPUTFORMAT  'org.apache.hadoop.mapred.TextInputFormat'  OUTPUTFORMAT  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'  LOCATION  's3://yelp-sk-test/yelp/user' |

|  |
| --- |
| CREATE EXTERNAL TABLE `yelp1.photos`(  `caption` string ,  `photo\_id` string ,  `business\_id` string ,  `label` string COMMENT 'from deserializer')  ROW FORMAT SERDE  'org.apache.hive.hcatalog.data.JsonSerDe'  STORED AS INPUTFORMAT  'org.apache.hadoop.mapred.TextInputFormat'  OUTPUTFORMAT  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'  LOCATION  's3://yelp-sk-test/yelp/photos' |

|  |
| --- |
| External table을 internal hive테이블로 생성  CREATE TABLE yelp2.review2  STORED AS parquet  AS select \* from yelp2.review  Internal hive 테이블로 생성하고 Load data로 s3 데이터를 입력하는 경우  CREATE TABLE `yelp2.review3`(  `review\_id` string,  `user\_id` string ,  `business\_id` string ,  `stars` int ,  `date` string ,  `text` string ,  `useful` int ,  `funny` int ,  `cool` int )  ROW FORMAT serde 'org.apache.hive.hcatalog.data.JsonSerDe'  STORED AS parquet  load data inpath 's3://yelp-sk-test/yelp/review/review.json'  overwrite into table yelp2.review3; |

[참고] AWS 서비스

|  |  |
| --- | --- |
| EC2 | Amazon Elastic Compute Cloud(Amazon EC2)는 크기를 조정할 수 있는 컴퓨팅 용량을 제공하는 웹 서비스 |
| S3 | 간편 스토리지 |
| EMC | 관리형 하둡 프레임워크로서 동적으로 확장 가능한 Amazon EC2 인스턴스 전체에서 대량의 데이터를 쉽고 빠르며 비용 효율적으로 처리.  [Apache Spark](https://aws.amazon.com/ko/emr/details/spark/), [HBase](https://aws.amazon.com/ko/emr/details/hbase/), [Presto](https://aws.amazon.com/ko/emr/details/presto/) 및 [Flink](https://aws.amazon.com/blogs/big-data/use-apache-flink-on-amazon-emr/)와 같이 널리 사용되는 분산 프레임워크를 실행하고, Amazon S3 및 Amazon DynamoDB와 같은 다른 AWS 데이터 스토어의 데이터와 상호 작업 가능 |

|  |  |
| --- | --- |
| 아파치 하이브  (Apache Hive) | 아파치 하이브(Apache Hive)는 하둡에서 동작하는 데이터 웨어하우스(Data Warehouse) 인프라 구조로서 데이터 요약, 질의 및 분석 기능을 제공한다. 초기에는 페이스북에서 개발되었지만 넷플릭스등과 같은 회사에서 사용되고 있으며 개발되고 있다.  아파치 하이브는 아파치 HDFS이나 아파치 HBase와 같은 데이터 저장 시스템에 저장되어 있는 대용량 데이터 집합들을 분석한다. HiveQL 이라고 불리는 SQL같은 언어를 제공하며 맵리듀스의 모든 기능을 지원한다. 쿼리를 빠르게 하기위해 비트맵 인덱스를 포함하여 인덱스 기능을 제공한다.  기본적으로 하이브는 메타데이터를 내장된 아파치 더비(Derby) 데이터 베이스안에 저장한다. 그렇지만 MySQL과 같은 다른 서버/클라이언트 데이터 베이스를 사용할 수 있는 선택권을 제공한다. 현재 TEXTFILE, SEQUENCEFILE, ORC 그리고 RCFILE등 4개의 파일 포맷을 지원한다. |
| 아파치 스파크  (Apache spark) | 스파크는 빅데이터 워크로드에 주로 사용되는 분산처리 시스템이며 하둡과 마찬가지로 오픈소스 입니다. 특징은 빠른 성능을 위해 인 메모리 캐싱과 최적화 된 실행을 사용하고 일반 배치처리, 스트리밍 분석, 머신러닝, 그래프 데이터 베이스 및 임시 쿼리를 지원합니다.  하둡없이도 클라우드 기반 데이터 플랫폼과 융합하며 원래는 하둡을 이용한 정보활용을 위한 데이터 프로세싱 툴입니다. 기존의 하둡을 통해 끌어오는 데이터들은 시간 소요가 크기 때문에 실시간으로 분석해야 하는 업무에서는 어려운 부분이 있어 새로이 개발 되었구요. 함수형 프로그래밍이 가능한 언어 스칼라(Scala)를 사용해 간단한 코드로 인터렉티브 쉘(Interactive shell)를 사용할 수 있답니다.  스파크는 전체의 데이터셋을 한꺼번에 처리합니다.  클러스터에서 데이터 읽기 ▶ 애널리틱스 운영 수행 및 결과값 클러스터 입력 동작과 같은 전 과정이 동시 진행  앞선 하둡과 달리 2단계로 처리가 되므로, 일반적 상황에서 스파크의 데이터 처리속도는 하둡에 비해 월등히 빠릅니다. 배치 프로세싱의 경우에는 하둡의 맵리듀스에 비하여 스파크가 10배 더 빠르고, 인 메모리 애널리틱스의 경우에는 100배 더 빠른 수행 속도를 낸다고 합니다. |
|  |  |
|  |  |
|  |  |