프로젝트 계획 보고서

작 품 명	“역세권에 따른 아파트 거래가격에 미치는 영향”
팀 명	구해줘 홈 조 (1조)
제 작 기 간	2021년 05월 31일 ~ 2021년 06월 04일
프로젝트 매니저	안 명 진 (인)

조 원			
No.	성 명	연 락 처	E-Mail
1	김 도 균	010 - 8819 - 9407	didiekdzm16@gmail.com
2	민 지 호	010 - 4710 - 7923	minjihonim2@gmail.com
3	안 명 진	010 - 3011 - 4176	ahnmyoungjin@gmail.com
4	이 채 진	010 - 4104 - 3024	celesta3024@gmail.com
5	임 홍 록	010 - 8733 -7565	krlim1202@gmail.com

1. 팀 소개

- **팀명 : 구해줘 홈**

서울에 수많은 집이 있지만 아직 발 뻗고 잘 내 집이 없습니다. 언제쯤 어디를 살면 좋을지 알고 싶었고 꿈과 희망을 가지고 혹시 서울에 집을 산다고 가정했을 때 좀 더 적당하고 괜찮은 집을 살 수 있을까에 대한 고민으로 미니프로젝트를 진행하게 되었고 구해줘 홈으로 팀 이름을 정하게 되었습니다.

- **역할 :**

- 전체적인 과정에서 문제 발생 시 팀원 모두가 서로 도와가면서 빠른 문제 해결 목표
  - 김 도 균 : 파이프라인 구축 / 데이터 전처리
  - 민 지 호 : 데이터 전처리 (수집 , 정제) / ML

- 안 명 진 : PM / 파이프라인 구축
- 이 채 진 : 파이프라인 구축(클러스터 세부 설정) / 시각화
- 임 홍 록 : 데이터 전처리 (수집 , 정제) / 파이프라인 구축(클러스터 세부 설정)

2. Project 주제 : “역세권이 아파트에 미치는 영향”

3. Project 개요

- 소개 :

최근 3년 7개월 간 서울 아파트 평균값은 4억2059만원 늘어났습니다. 비율로 따지면 약 69.3% 급상승하였고 정부의 규제 속에서도 지하철역과 가까운 역세권에 대한 인기는 꾸준히 증가하고 있습니다.

또한 같은 지역이라도 역과의 거리에 따라 수천만 원에서 수억 원까지 가격 상승 차이가 나타나고 있습니다. 한 예로 6월 입주를 앞둔 성복역 역세권 아파트인 성복역 롯데캐슬골드타운의 전용면적 84.91㎡ 분양권은 지난 3월, 분양가보다 3억원 가량 오른 8억3,809만원(27층)에 거래될 정도로 높은 프리미엄을 형성하고 있지만 역과 멀리 떨어진 비역세권 아파트는 프리미엄 없이 분양가 수준에 머무르고 있는 것으로 나타났다는 서울경제신문의 기사를 통해 알 수 있었습니다.

부동산 데이터를 분석하여 역과의 거리에 따른 집값들의 적절한 가격을 모델링하여 추천하는 모델을 만들어 제공할 것입니다.

(집값의 상승률에 관해 영향을 주는 요인은 역세권 이외에 여러 가지 요인이 있지만 서울시에서는 역과의 거리가 가장 큰 요인으로 작용하기에 역에서의 반경을 기준으로 한정하였습니다.)

■ 4. 활용 범위

▶ 역과의 거리에 따라 차이나는 집값의 데이터를 제공하게 된다면 같은 지역 내에 집값의 차이를 줄이는 기대효과와 지금보다 효율적으로 집 선택을 가능하게 해줄 것입니다. 그리고 이러한 분석을 통해 부동산에 투자하고자 하는 사람들에게 참고할 수 있는 지표를 제공해줄 수 있다.

5. DB 구축 및 데이터 분석 내용

● 데이터 구성

국토교통부 실거래가 공개시스템

- - 국토 교통부
- 2016.01 ~ 2021.05.31
- 서울시 부동산 아파트 실거래가 정보

DATA 공공데이터포털 .GO.KR

- - 공공데이터포털
- 지하철 역 위치데이터('위도', '경도')

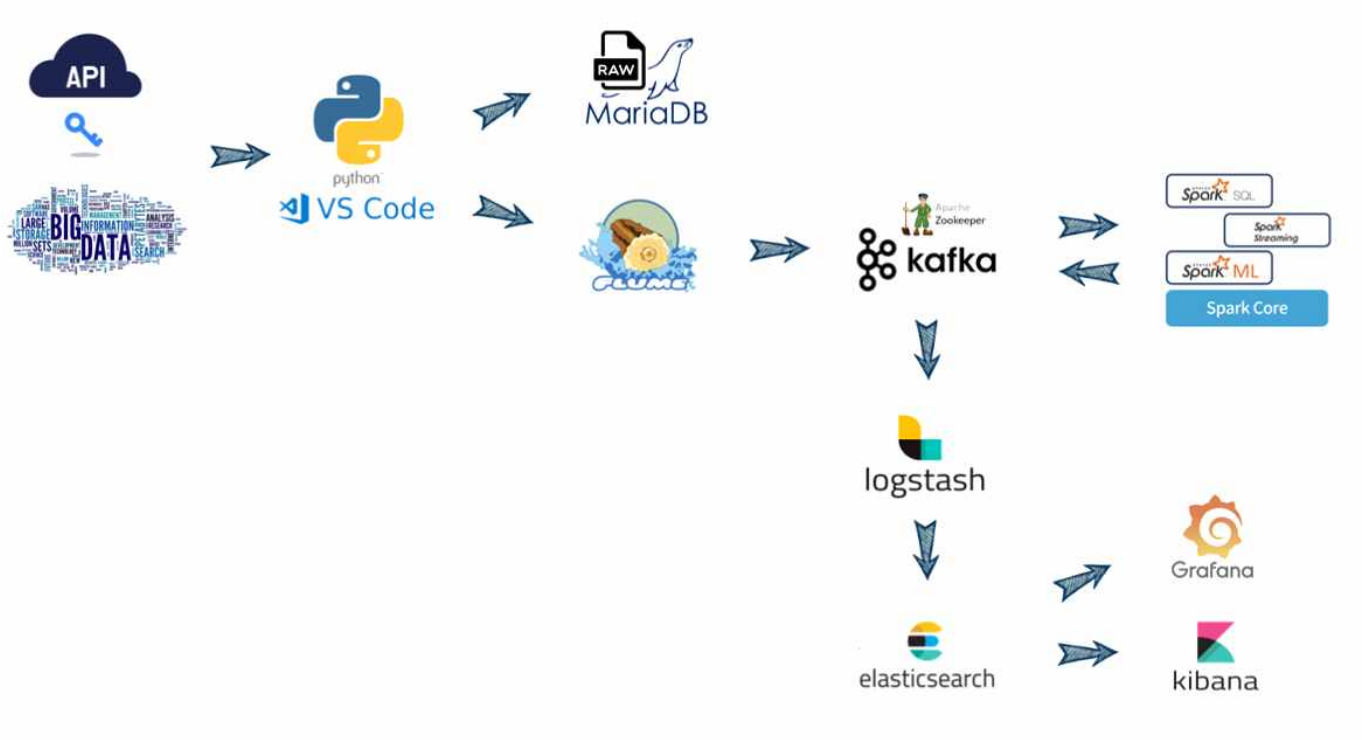
서울 열린데이터 광장

서울시 동별 아파트 매매거래 현황

- - 서울 열린 데이터 광장
- 2021.05.31~
- 서울시 부동산 아파트 실거래가 정보

● 논리 구성

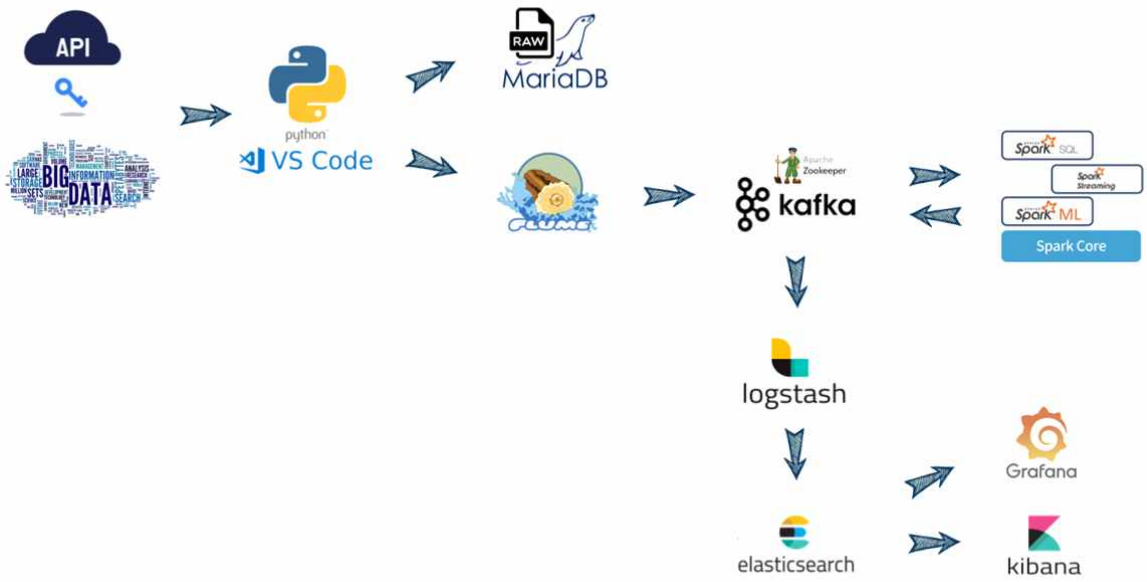
- 프로젝트 목표 구성도



- 데이터 파이프라인 구축 (데이터 흐름 확인)

- 프로젝트 최종 구성도

•



- VirtualBox VM에 Linux Centos7 2009ver 64bit 기반으로 설치.

- 국토교통부에서 5년간의 부동산 Data와 공공데이터포탈에서 지하철역 위치 Data 수집 하고 서울 열린 데이터 광장에서 Open API로 실시간 거래정보를 가져와 python으로 전 처리하여 CSV파일로 저장

- 전처리된 CSV파일은 MariaDB에 원본(RawData)를 적재하고 실시간 분석을 위한 변환Data는 Flume을 통해 Kafka로 전송

- Kafka에서 Spark로 Data를 전송하고 ML model을 만들어 실시간 분석에 대한 데이터를 생성하여 다시 Kafka에 전송

- Logstash를 통해 Kafka의 ML 데이터를 받아 ElasticSearch에 전달해 Mapping 확인한 후 Grafana와 Kibana로 데이터 시각화
(ML-model로 집값 상승률 예측과 서울지역의 평당 집값에 대한 위치 시각화)

● 사용 기술

구분	이름	초기버전			현재버전
개발 환경	Oracle VM virtualBox	6.1.18 r142142 (Qt5.6.2)			동 일
	CentOS7	2009_64bit			
	CPU	2 core			
	RAM	8gb			
	disk	swap			
	memory	boot	500mb	ext4	
	전체	/	나머지	xfs	동 일
사용 Tool	OracleJDK	1.8.0_281			동 일
	MariaDB	10.4.18			동 일
	Elasticsearch	7.12.1			7.13.0
	Logstash	7.12.1			7.13.0
	Kibana	7.12.1			7.13.0
	Kafka	2.12 - 2.8.0			동 일
	Spark	3.0.2			동 일
	Flume	1.8.0			동 일
	Zookeeper	3.5.6			동 일

- **산출물**

- 통계적 분석을 통해, 집값에 결과에 영향을 가장 많이 주는 요인들을 선정하여 활용해 예측 모델을 만들 예정이였으나 실시간 자동화 부분을 구현하지는 못하였다.

6. 결과물

6-1. 구축 (프로그램 설치)

6-1-1 리눅스 시스템 구축 (CentOS 7 설치 및 구축)

1) VM 만들기
   - 리눅스, Redhat으로 설정
   - '새로 만들기' 누르면 설정 창이 확인
   - 설정 창에서 체크부분대로 입력 후 확인 선택 그리고 실행버튼
   - 시동 디스트 "CentOs-7-...2009.iso" 선택 후 시작
   - 인스톨 후 언어는 한국어 선택한다.

2) 설치요약 순서대로 진행
   - 소프트웨어 선택 - 기본환경에서 "GNOME 데스크탑"만 선택 후 완료버튼
   - 파티션 설정 선택 - 기타저장옵션에서 "파티션을 설정합니다." 선택 후 완료버튼
   - "+"버튼 눌러서 추가해주기
   - /boot 입력 500mb 입력 후 추가버튼-> 파일시스템 "ext4" 설정
   - Swap 입력 8GB 입력 후 추가버튼 -> 파일시스템 "swap" 설정
   - / 입력 후 추가버튼 ->파일시스템 "xfs" 설정

3) 네트워크 및 호스트명 들어가서 이더넷 "켬"으로 변경

4) 위에 설정 완료 후 설치시작 버튼

5) 설치가 시작되면 루트 비밀번호 설정

6) 설치 완료 후 루트계정으로 마우스, 디스플레이, 클립보드, 파일 드래그 앤 드롭 설정
   - $yum groupinstall "Development Tools"
   - $yum -y install kernel source
   - $yum -y install kernel-devel
   - $yum update
   - $reboot
   - 상단 바 장치 탭 > 클립보드 공유 / 드래그앤드롭 > 양방향 체크
     (없으면 마우스 우측 클릭하여 설정)
   - 재부팅 후 상단 바 장치 탭 > 게스트 확장 CD 이미지 삽입.
   - 설치 진행
   - 재부팅 후 마우스 연동, 화면 조절, 드래그앤드롭, 클립보드
     (잘라내기/복사 > 붙여넣기) 작업 확인

7) selinux, 방화벽 off
   - $vi /etc/selinux/config

SELINUX=disabled
SELINUX=permissive

   - $systemctl disable firewalld

6-1-2. oracle java 1.8 설치하기

참고 : https://copycoding.tistory.com/290

6-1-3. ELK beat 설치

▶ Elasticsearch 설치

1) java 버전확인

① $java -version

② oracle JDK 1.8.x 확인

2) rpm으로 elasticsearch 설치

① $rpm —import https://artifacts.elastic.co/GPG-KEY-elasticsearch ② $vi /etc/yum.repos.d/elasticsearch.repo (아래 내용 작성 후 저장) ③ $yum install —enablerepo=elasticsearch elasticsearch
파일 : /etc/yum.repo.d/elassticsearch.repo
[elasticsearch] name=Elasticsearch repository for 7.x packages baseurl=https://artifacts.elastic.co/packages/7.x/yum gpgcheck=1 gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch enabled=0 autorefresh=1 type=rpm-md

3) elasticsearch 환경설정

① $vi /etc/elasticsearch/elasticsearch.yml

(아래 그림과 같이 Network 부분에서 network.host의 주석 제거 후 127.0.0.1로 설정 / http.port의 주석제거)

```
51 # ---------------------------------- Network -----------------------------------
52 #
53 # Set the bind address to a specific IP (IPv4 or IPv6):
54 #
55 network.host: localhost
56 #
57 # Set a custom port for HTTP:
58 #
59 http.port: 9200
60 #
```

4) elasticsearch 재시작 후 정상설치 되었는지 확인

① $systemctl restart elasticsearch ② $curl http://127.0.0.1:9200

```
[root@localhost ~]# curl http://127.0.0.1:9200
{
  "name" : "localhost.localdomain",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "e9tj-SElQ_2cSyOqxHCToQ",
  "version" : {
    "number" : "7.10.1",
    "build_flavor" : "default",
    "build_type" : "rpm",
    "build_hash" : "1c34507e66d7db1211f66f3513706fdf548736aa",
    "build_date" : "2020-12-05T01:00:33.671820Z",
    "build_snapshot" : false,
    "lucene_version" : "8.7.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

▶ kibana 설치

1) rpm으로 kibana 설치

① (elasticsearch에서 설정 변경 시 ①단계 생략 후 ②부터) $rpm —import https://artifacts.elastic.co/GPG-KEY-elasticsearch ② $vi /etc/yum.repos.d/kibana.repo (아래 내용 작성 후 저장) ③ $yum install kibana
[파일: /etc/yum.repos.d/kibana.repo]
[kibana-7.x] name=Kibana repository for 7.x packages baseurl=https://artifacts.elastic.co/packages/7.x/yum gpgcheck=1 gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch enabled=1 autorefresh=1 type=rpm-md

2) kibana 환경설정

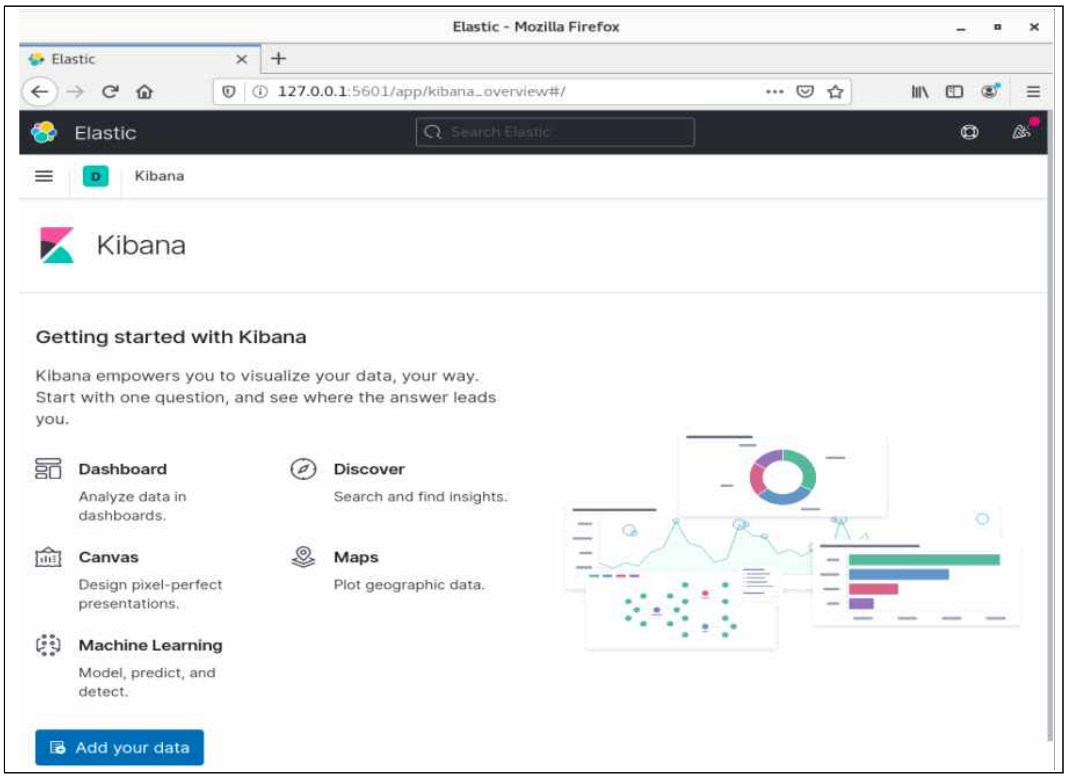
① $vi /etc/kibana/kibana.yml (2, 7, 28번 줄 주석 제거)

```
2 server.port: 5601

4 # Specifies the address to which the Kibana server will bind. IP addresses and h
  alid values.
5 # The default is 'localhost', which usually means remote machines will not be ab
7 server.host: "localhost"

9 # Enables you to specify a path to mount Kibana at if you are running behind a p
10 # Use the `server.rewriteBasePath` setting to tell Kibana if it should remove th
11 # from requests it receives, and to prevent a deprecation warning at startup.
12 # This setting cannot end in a slash.
13 #server.basePath: ""
14
15 # Specifies whether Kibana should rewrite requests that are prefixed with
16 # `server.basePath` or require that they are rewritten by your reverse proxy.
17 # This setting was effectively always `false` before Kibana 6.3 and will
18 # default to `true` starting in Kibana 7.0.
19 #server.rewriteBasePath: false
20
21 # The maximum payload size in bytes for incoming server requests.
22 #server.maxPayloadBytes: 1048576
23
24 # The Kibana server's name.  This is used for display purposes.
25 #server.name: "your-hostname"
26
28 elasticsearch.hosts: ["http://localhost:9200"]
```

3) kibana 재시작 후 정상설치 확인

① $systemctl restart kibana

② (firefox 실행) http://127.0.0.1: 5601



▶ Logstash 설치

$rpm —import https://artifacts.elastic.co/GPG-KEY-elasticsearch

$vi /etc/yum.repos.d/logstash.repo (아래 내용 작성 후 저장)

```
[logstash-7.x]
name=Elastic repository for 7.x packages
baseurl=https://artifacts.elastic.co/packages/7.x/yum
gpgcheck=1
gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch
enabled=1
autorefresh=1
type=rpm-md
```

$yum install logstash

6-1-4. mariadb 설치

참고 : https://bamdule.tistory.com/59

6-1-5. zookeeper 설치

- 주키퍼 설치

$wget http://archive.apache.org/dist/zookeeper/zookeeper-3.5.6/apache-zookeeper-3.5.6-bin.tar.gz

$tar -zxvf apache-zookeeper-3.5.6.tar.gz

$mv apache-zookeeper-3.5.6 /home/tdata/

$cd /home/tdata/

$ln -s apache-zookeeper-3.5.6 zookeeper

- 주키퍼 데이터 폴더 생성

$mkdir -p /home/tdata/data/zookeeper

- 주키퍼 설정

$vi /home/tdata/zookeeper/conf/zoo.cfg

$cp zoo_sample.cfg zoo.cfg

$vi zoo.cfg

+ 여기에서 12번째 줄 dataDir 수정 후 맨 아래에 서버 추가

12번줄) dataDir=/home/tdata/data/zookeeper

서버추가) server.1=hadoop01:2888:3888

admin.serverPort=18080

- 데이터폴더에 myid 파일 생성

$cd /home/tdata/data/zookeeper

$vi myid

1

- 부팅시 자동으로 실행하도록 서비스 등록

$cd /etc/init.d

$vi zookeeper-server

```
#!/bin/sh
#
# chkconfig: - 80 05
# description: zookeeper server
#

BIN_PATH=/home/tdata/zookeeper/bin
case $1 in
          start)  ${BIN_PATH}/zkServer.sh start   ;;
          stop)  ${BIN_PATH}/zkServer.sh stop;;
          status)  ${BIN_PATH}/zkServer.sh status;;
          restart)  ${BIN_PATH}/zkServer.sh restart;;
              *)  echo "require start|stop|status|restart"  ;;
esac
```

$chmod +x zookeeper-server

$chkconfig —add zookeeper-server

$chkconfig zookeeper-server on

- 주키퍼 실행 (.bash_profile에 ZOOKEEPER_HOME 변수 설정 시 폴더 위치 상관없이 실행 가능)

$service zookeeper-server start

(안될 시 /home/tdata/zookeeper/bin/에 가서 ./zkServer.sh start 명령어 실행)

$zkServer.sh status

(에러 발생 시 zoo.cfg의 port 설정 후 문제가 해결 될 수 있음.)

$vi zoo.cfg

---

admin.serverPort=18080

6-1-6. kafka 설치

1) kafka 다운로드 후 압축해제, 심볼릭 링크 생성(kafka_2.13-2.4.1)

```
$wget https://downloads.apache.org/kafka/2.8.0/kafka_2.12-2.8.0.tgz
$tar -xvzf kafka_2.12-2.8.0.tgz
$ln -s kafka_2.12-2.8.0 kafka
```

2) kafka 시작(Cluster가 없다면 환경설정 하지 않는다)

```
$./bin/kafka-server-start.sh -daemon ./config/server.properties
 (kafka 실행 중 잠깐 나오려면 ctrl+z, 들어갈때는 "fg 1")
```

6-1-7. spark 설치

1) spark-3.0.2-bin-hadoop2.7.tgz 파일 root에 압축풀기

```
$wget http://apache.mirror.cdnetworks.com/spark/spark-3.0.2/spark-3.0.2-bin-hadoop2.7.tgz
$-tar -zxvf spark-3.0.2-bin-hadoop2.7.tgz (압축해제)
$ln -s spark-3.0.2-bin-hadoop2.7 spark
```

(사용 편의를 위해 심볼릭링크 설정)

2) .bash_profile 에서 환경변수 설정

```
export SPARK_HOME=/root/spark
export PATH=$PATH:$SPARK_HOME/bin
```

6-1-8. flume 설치 (ver 1.8.0)

1) 설치 위치 설정

```
$cd /home/tdata의 경로에 설치 (자신이 원하는 위치에 설치해도 상관없음)
```

2) url로부터 Flume 설치.

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

3) 폴더명 변경

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

4) 압축 풀기

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

5) 경로 설정

```
$vi /etc/profile ( vi .bash_profile도 상관없음.)
```

```
export FLUME_HOME=/home/tdata/flume
export PATH=$PATH:$FLUME_HOME/bin
```

---

6) 변경사항 적용

```
$source /etc/profile
```

7) 기본 conf file을 복사하여 변경

```
$cd /home/tdata/flume/conf
$cp flume-env.sh.template flume-env.sh
```

8) 편집기로 복사한 flume-env.sh 수정 (주석 풀기)

```
$vi flume-env.sh
```

25~26번째 줄에 있는 export JAVA_OPTS 주석 풀기

```
export JAVA_OPTS="-Xms100m -Xmx2000m -Dcom.sun.management.jmxremote"
```

9) 변경사항 적용

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

[Flume 설정]

- flume의 conf 경로로 이동하여 test.conf 생성

```
$cd /home/tdata/flume/conf
$vi test.conf
```

- test.conf 작성

```
Test_Agent.sources = TestSource_SpoolSource
Test_Agent.channels = TestChannel_Channel
Test_Agent.sinks = TestSink_LoggerSink
Test_Agent.sources.TestSource_SpoolSource.type = spooldir
Test_Agent.sources.TestSource_SpoolSource.spoolDir = /home/tdata/flume/working/batch-log
Test_Agent.sources.TestSource_SpoolSource.deletePolicy = immediate
Test_Agent.sources.TestSource_SpoolSource.batchSize = 1000
Test_Agent.channels.TestChannel_Channel.type = memory
Test_Agent.channels.TestChannel_Channel.capacity = 100000
Test_Agent.channels.TestChannel_Channel.transactionCapacity = 10000
Test_Agent.sinks.TestSink_LoggerSink.type = logger
Test_Agent.sources.TestSource_SpoolSource.channels = TestChannel_Channel
Test_Agent.sinks.TestSink_LoggerSink.channel = TestChannel_Channel
```

- flume이 지켜볼 경로 생성

```
$cd /home/tdata/flume
$mkdir working
$cd working
$mkdir batch-log
```

[Flume 실행]

- flume 경로로 이동

$cd /home/tdata/flume

- flume 실행

$./bin/flume-ng agent -c conf -f conf/test.conf -n Test_Agent -Dflume.root.logger=INFO,console

- 동작확인 (Test)

다른 터미널을 열고 /var/log 경로 중 아무 log file을 /home/tdata/flume/working/batch-log 경로에 복사.

$cp /var/log/logstash/logstash-deprecation.log /home/tdata/flume/working/batch-log/

- 기존 터미널 확인 시, event, header, body 등 기록이 남아있으면 동작 확인 완료

- log파일이 아닌 다른 파일 옮길 시, Error발생.

[참고 URL]

- https://bonahbruce.tistory.com/30
- https://dabingk.tistory.com/15

6-1-9. python 3.x 버전으로 변경

- 파이썬 버전과 경로 확인

$python -V

$which python

- 파이썬 실행 파일 확인

$ls -al /bin/python

$ls /bin | grep python

- 파이썬 3.6 버전 설치

$yum install python3

- 파이썬 버전 등록,변경

$update-alternatives --install /bin/python python /bin/python2.7 1

$update-alternatives --install /bin/python python /bin/python3.6 2

$update-alternatives --config python

2를 선택.

- 링크. 버전 변경 확인

$ls -al /bin/python

python -V

- 파이썬 버전 변경 시 yum 명령어가 실행이 안 되는 경우가 생기면 아래의 설정을 변경

$vi /usr/bin/yum

$vi /usr/libexec/urlgrabber-ext-down

실행 후

첫줄: #!/usr/bin/python -> #!/usr/bin/python2로 변경

- 파이썬 사용 시 ERROR: Python headers are missing in /usr/include/python3.6m 발생 시

$yum install python3-dev

- 파이썬 pip 다운로드

$yum install python3-pip (설치가 되어 있다면 설치가 되어있다고 나온다.)

6-1-10. pyspark3 - jupyter notebook 연동

- pip3 와 jupyter notebook 설치

$yum install pip3

$pip install jupyter

- 설치 후 실행 확인

$jupyter notebook --allow-root

- jupyter에서 python 실행

1) python실행

>> from notebook.auth import passwd

>> passwd()

2) 비밀번호 설정 후 설정 변경해주기

>> jupyter notebook --generate-config

- 터미널로 돌아가서 jupyter notebook config 수정

$vi ~/.jupyter/jupyter_notebook_config.py

---

c.NotebookApp.open_browser = True

$vi /root/.bash_profile

---

export SPARK_HOME=/스파크 받은 경로/spark

export PATH=$SPARK_HOME/bin:$PATH

export PYSPARK_DRIVER_PYTHON=jupyter

export PYSPARK_DRIVER_PYTHON_OPTS='notebook --allow-root'

- 설정 적용

$source .bash_profile

6-1-11. grafana 설치 및 연동

1) Grafana 설치

1. 설치 시 폴더 위치와는 상관없이 설치해도 괜찮다.

$wget https://dl.grafana.com/oss/release/grafana-7.1.0-1.x86_64.rpm
$yum install grafana-7.1.0-1.x86_64.rpm

2. Grafana 실행

$systemctl start grafana-server

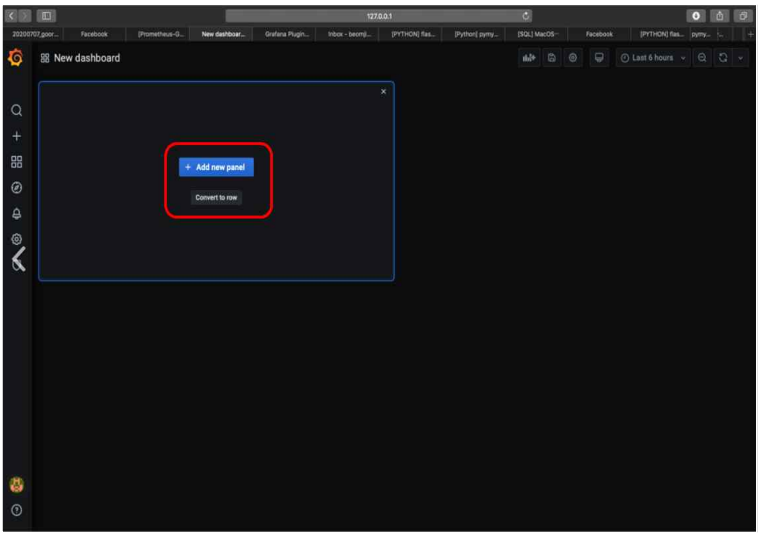
3. Grafana Port 확인 (Port 3000)

$netstat -tnl
(~:3000 유무 확인. 없으면 방화벽 끄고 2번 재실행.)

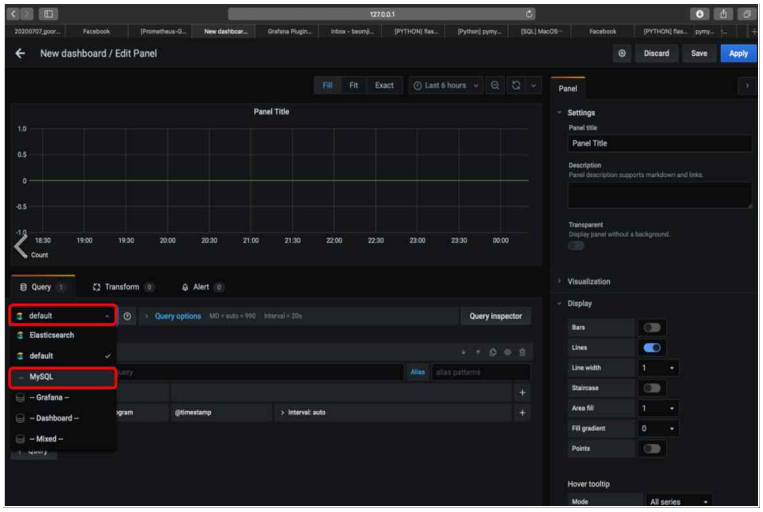
4. localhost:3000 접속

초기 계정 ID / Password : admin/admin

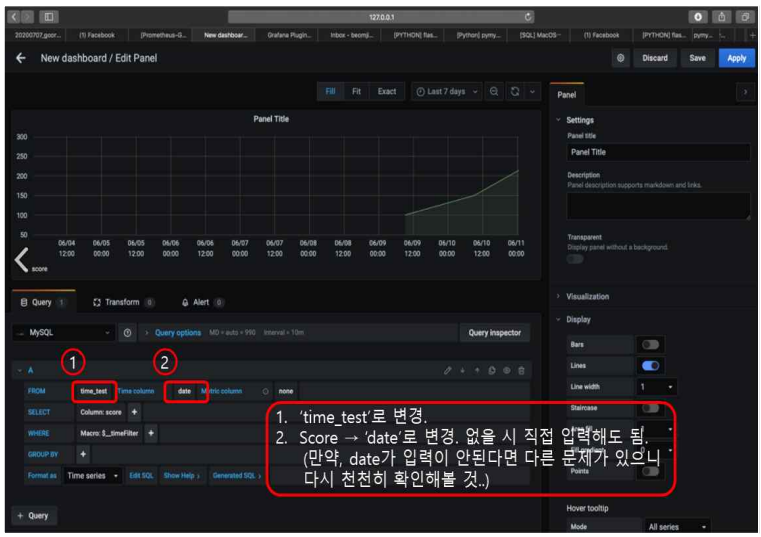
※접속이 되었다면 Grafana 설치 끝!



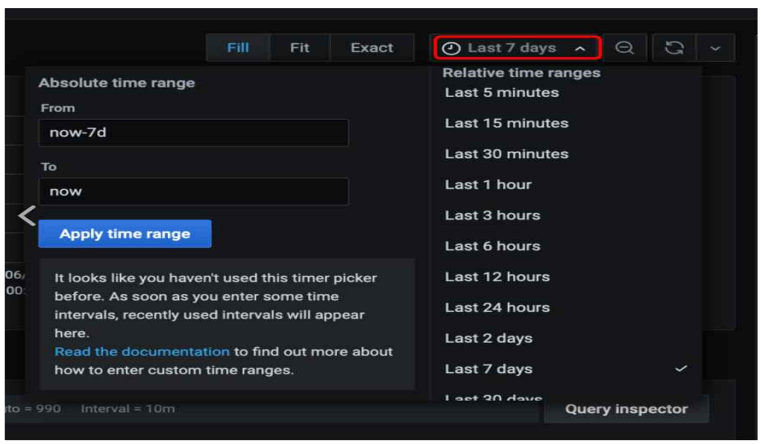
- 'Add New Pannel' 클릭



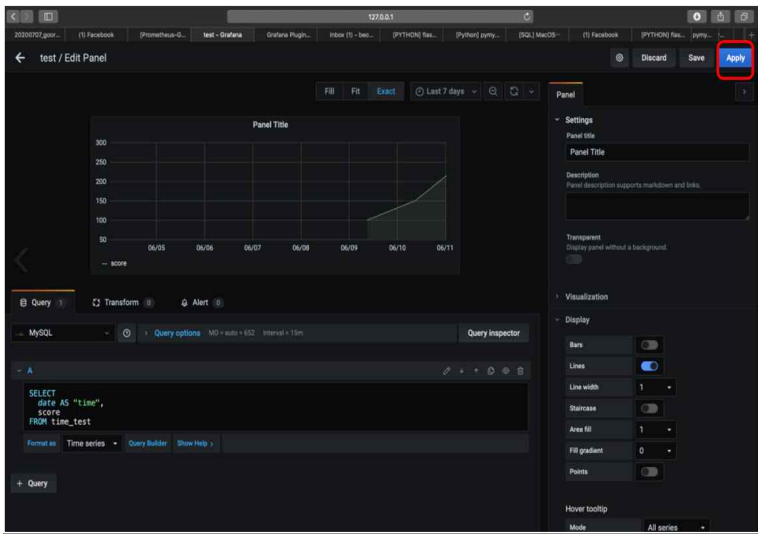
- 'Default'로 된 부분을 클릭 후 'MySQL'로 변경.



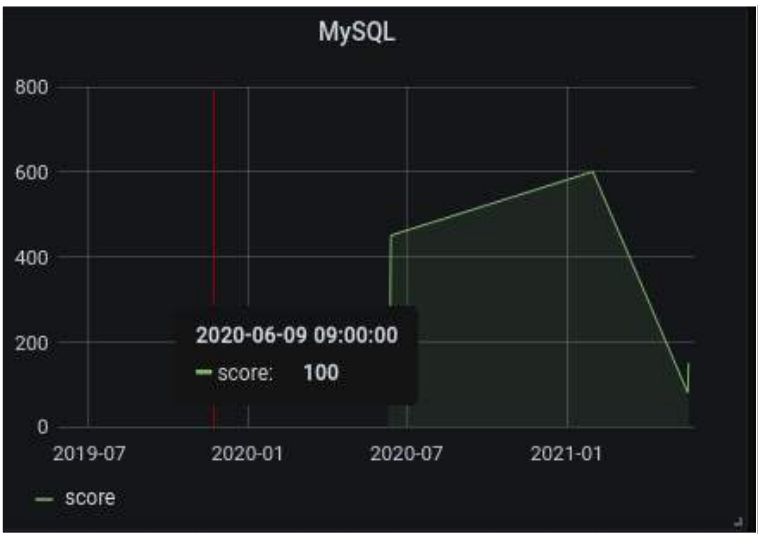
- 1, 2에 대한 정보 수정.



- 데이터 시각화는 'Table Date' 기준이기 때문에 '2020년 6월 9일 이후 데이터가 보이게끔 Time Scale 설정.



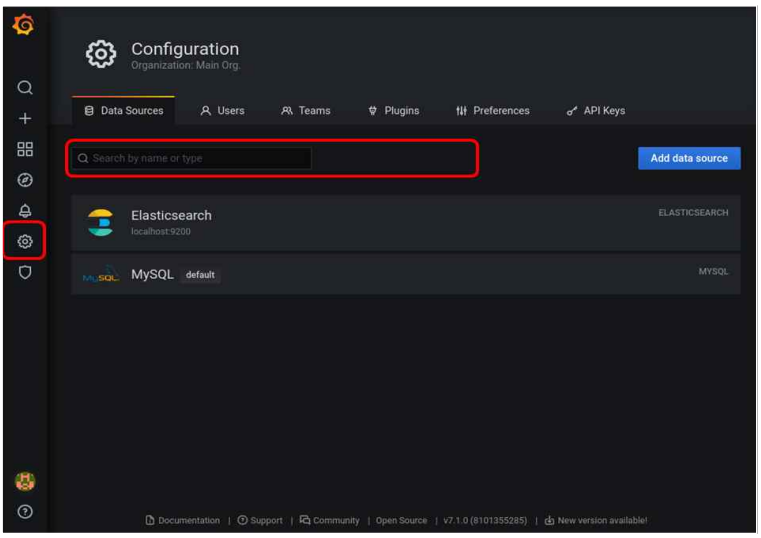
- Apply로 적용.



- 시각화된 데이터 확인

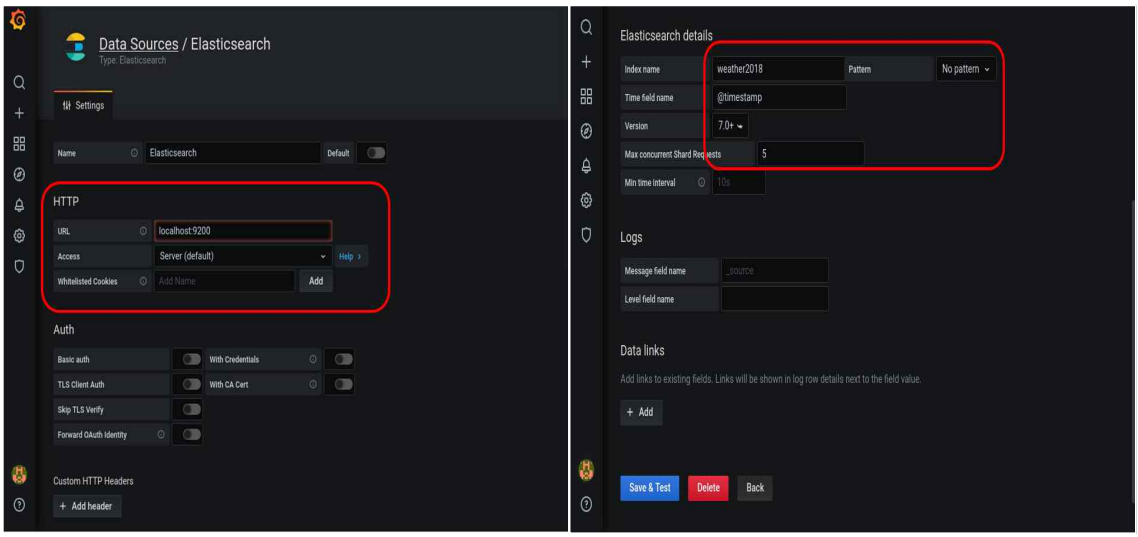
[Elasticsearch 연동]

1. Elasticsearch Dataset

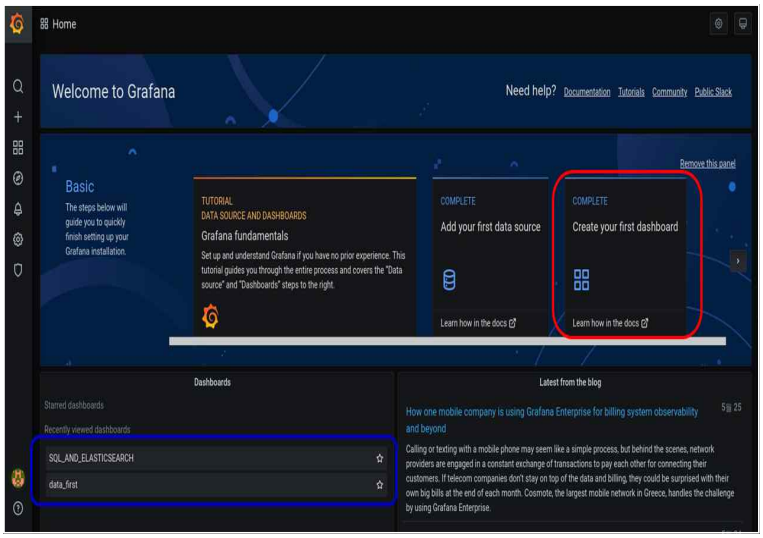


- 우측 '톱니바퀴'아이콘 클릭 후 Configuration 클릭.

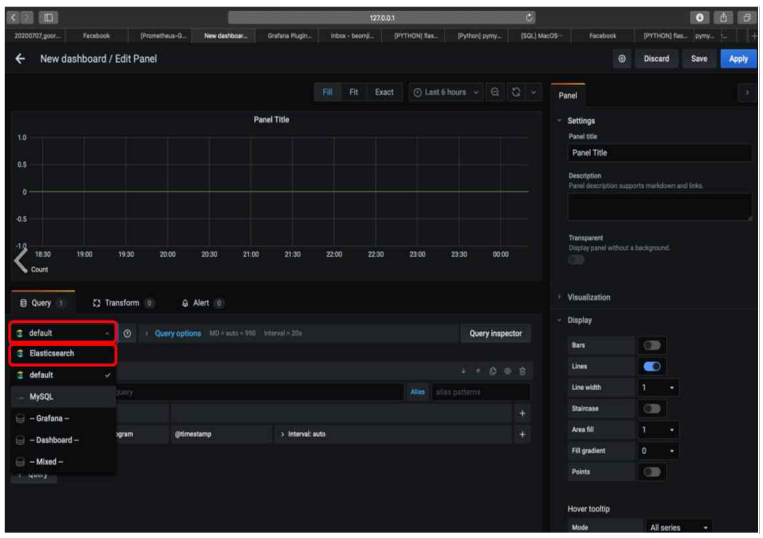
- 'elasticesarch' 검색



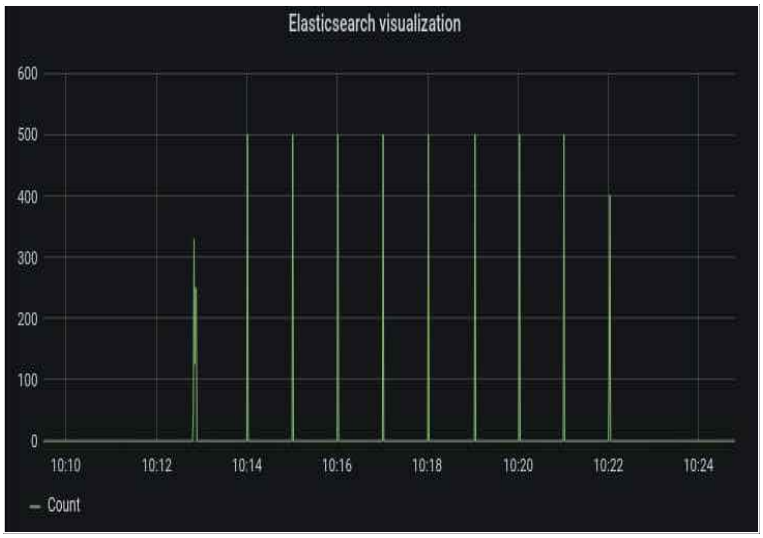
- 위 그림과 같이 'HTTP'와 'Elasticesarch details' 옵션 설정.



- 다시 Home 텝으로 이동하여 Dashboard 생성
(기존에 있던 dashboard에 생성하고 싶다면, 좌측 하단에 있는 이전 dashboard[파랑색] 선택)



- Elasticsearch로 변경하여 Index로 받아온 데이터의 TimeScale을 맞춰줌.
-
- 이후 우측 상단의 'Apply'로 변경된 설정 저장.



- 이후 Dashboard를 통해 Elasticsearch로 들어온 데이터를 확인할 수 있음.
- weather index의 경우, cron을 통해 1분당 500개의 날씨data를 불러오기 때문에 위와 같이 시각화된 것을 알 수 있음.

6-2. 데이터 수집 (전처리 과정)

- 전처리는 크게 두 가지 분류로 나뉜다. 과거 5년치와 올해 현 시점까지의 데이터(CSV File)를 하나로 통합한 뒤, MariaDB에 적재 및 Total 데이터를 CSV파일로 만들어줄 필요가 있었다. 이후 Web API로 데이터를 받아와 DB적재 및 CSV파일 생성까지 전처리를 진행하였다.

6-2-1 과거 데이터 수집

- 국토교통부의 실거래가 공개시스템에서 서울시 부동산 아파트 매매의 실거래가에 대한 데이터를 2016년부터 최근 2021년 5월 까지 데이터를 csv 파일을 이용하였다. 공공데이터 포탈에서 역세권의 거리 계산을 위한 서울지하철 역 및 역의 위치 정보(위도, 경도)를 얻기 위해 수집하였다. 실시간 아파트 정보는 서울 열린 데이터 광장에서 OpenAPI를 통해 동별 아파트 매매거래현황을 통해 전달 받았다.

6-2-2 과거 데이터 전처리

```
#### ==============================DB Connection==========================
connection_url = 'mysql+pymysql://root:sd0241@localhost/apart'
# MariaDB (MySQL) root계정에 비밀번호 root로 로그인하여 estate_db를 사용.
connection = create_engine(connection_url)
# DB에 쿼리를 보내기 위한 connection 생성
conn = connection.connect()
# connection에 대한 객체(conn) 를 생성
```

그림1. DB Connection

먼저, 위 그림1의 코드와 같이 MariaDB에 Connection을 해준다. 이후, 그림 2와 같이 2016년 ~ 2021년 현 시점까지의 모든 데이터를 하나의 Dataframe으로 만들어준 뒤 전처리가 시작된다.

```
#### ==================================CSV Merge=====================================
input_file = r'./'                  # '' 안에 불러올 csv 파일 위치 확인 !!
# 읽고 싶은 파일의 경로 선택.
output_file = r'./apt_total.csv'                # 결합 후 내보낼 파일명 및 위치 확인 !!
# 출력 파일의 경로 및 파일명,확장자 선택.
allFile_list = glob.glob(os.path.join(input_file, '아파트*'))
# input_file 변수 경로에 있는 '아파트'로 시작하는 모든 파일을 호출하여 파일명을 list에 담음.
allData = []                    # Dataframe에 적용할 빈 List 생성.
for file in allFile_list:
    df = pd.read_csv(file, encoding='cp949')        # 각 파일을 cp949 encoding방식으로 불러옴.
    allData.append(df)              # 각 내용을 list에 추가.
df_apt = pd.concat(allData, axis=0, ignore_index=True)  # concat함수를 통해 리스트 내용을 지속적으로 이어서 작성.
df_apt = df_apt.drop_duplicates()           # 중복 데이터 제거.
df_apt = df_apt.sort_values('계약년월')       # '집계일자' 컬럼 기준 데이터 정렬.

df_apt.to_csv(output_file, encoding='utf-8-sig', index=False)   # 완성된 데이터는 utf-8-sig형식으로 인코딩 된 CSV 파일로 반환.
```

그림2. 과거 데이터 통합

```
# ==================================READ EXCEL=====================================
df_apt = pd.read_csv('./apt_total.csv', encoding='utf-8-sig')
# #위 코드 경로는 자신의 경로에 맞출것!
date = ''
nowDate = int(datetime.datetime.now().strftime('%Y'))   # 현재 날짜의 년도 호출
date_list = []
m2_price = []
df_apt['계약년월'] = df_apt['계약년월']*100+df_apt['계약일']   # 계약년월 컬럼과 계약일 컬럼을 하나의 날짜로 통합
df_apt = df_apt.drop(['계약일'], axis=1)   # 분석에 필요없는 컬럼 제거
df_apt = df_apt.drop(['본번'], axis=1)    # 분석에 필요없는 컬럼 제거
df_apt = df_apt.drop(['부번'], axis=1)    # 분석에 필요없는 컬럼 제거
df_apt = df_apt.drop(['해제사유발생일'], axis=1)   # 분석에 필요없는 컬럼 제거
df_apt.rename(columns={'계약년월': '거래날짜'}, inplace=True)  # 컬럼명 변경
df_apt.rename(columns={'거래금액(만원)': '거래금액'}, inplace=True)# 컬럼명 변경
```

그림3. 불필요한 컬럼 제거

이전 데이터에서 apt_total.csv로 저장된 파일을 다시 read.csv 메서드로 불러오는 코드가 있다. 위와 같이 CSV파일로 저장하지 않고 cp949 타입으로 이를 진행했을 때, dataframe의 index 문제로 지속적인 에러가 발생되는 것을 확인할 수 있었다. 이러한 이유로 utf-8-sig 형식 CSV 파일 추출 후 다시 호출하는 방식으로 진행하였다.

```
dong = []
city = []
for i in df_apt['시군구']:                # '시군구'컬럼에는 시/구/동의 정보가 담겨있음.
    temp = i.split(' ')                   # 해당 컬럼에서 '시/군'과 '동' 데이터를 따로 취득해야 하기 때문에
    city.append(temp[0] + ' ' + temp[1])  # for 구문을 이용하여 이를 나눠줌.
    dong.append(temp[2])
df_apt['시군구'] = city                   # 나눠진 '시/구' 데이터는 '시군구' 컬럼에 적용.
df_apt['법정동명'] = dong                 # 나머지 '동' 데이터는 '법정동명'에 적용.
df_apt = df_apt[['시군구', '법정동명', '번지', '단지명', '전용면적(㎡)', '거래날짜', '거래금액', '층', '건축년도', '도로명']]
# 컬럼명 순서로 데이터 재정렬
df_apt = df_apt.rename(columns={'전용면적(㎡)': '전용면적'})  # 컬럼명 변경
df_apt = df_apt.rename(columns={'층': '층정보'})             # 컬럼명 변경
df_apt = df_apt.rename(columns={'단지명': '건물명'})          # 컬럼명 변경
```

그림4. 컬럼 나누기, 재정렬 및 컬럼명 변경

그림4는 기존 '시군구' 데이터의 '시/구/동' 형식을 '시/구'와 '동' 데이터를 나눠주는 작업을 나타낸다. 이렇게 나눠주는 이유는 도로명 데이터를 사용할 때 '동'과 '건물명'을 탐색하여 일치하였을 때 해당 index에 대한 위도/경도 값을 추출해야 하기 때문에 나눠주었다.

```
date = []
df_apt = df_apt.astype({'거래날짜': 'str'})        # 데이터 타입을 string 형태로 형변환.
for i in range(0, len(df_apt['거래날짜'])):        # 거래 날짜의 년도/월/일 순으로 나누고, 날짜 형식 yyyy-mm-dd 형태로 저장.
    date.append(df_apt['거래날짜'][i][:4] + '-' + df_apt['거래날짜'][i][4:6] + '-' + df_apt['거래날짜'][i][6:8])
hms = ' 00:00:00'                  #데이터 베이스 및 logstash의 timestamp에 맞추어 시/분/초 데이터를 추가해주기 위한 hms변수 선언
df_apt['거래날짜'] = date           # 나눠진 데이터 데이터를 '거래날짜' 컬럼에 저장.
df_apt['거래날짜'] = df_apt['거래날짜']+hms # 기존 데이터에 시/분/초 문자열을 더함.
```

그림5. 날짜 형식 바꾸기

Logstash나 CentOS7에서의 MariaDB는 반드시 연/월/일/시/분/초의 형식에 맞추어 저장해야 Timestamp axis를 이용할 수 있다. 시/분/초에 대한 정의는 반드시 정확한 시간이 요구되는 것은 아니기 때문에 00시 00분 00초로 지정하였다. (그림5 참조)

```
for i, row in df_apt.iterrows():                  # df_apt의 영역을 탐색함.
    if nowDate - (df_apt.at[i, '건축년도']) > 10:    # 올해의 년도에서 '건축년도' 컬럼을 뺀 값이 10년 이상이면
        df_apt.at[i, '신축구분'] = '구축'           # 해당 건물은 '구축'으로 판단.
    else: df_apt.at[i,'신축구분'] = '신축'          #그렇지 않으면 '신축'으로 판단.
```

그림6. 신축/구축 정의

일반적인 건물은 완공 이래로 10년까지는 신축으로 인정해준다는 자료를 참고하여 10년 이하는 '신축'으로, 그 이상은 '구축'으로 판단하였다. 여기서 아쉬웠던 점은, Spark ML을 연계하기 위해서는 숫자로 표기했으면 하나의 지표를 더 사용할 수 있었을 텐데 그렇지 못했던 부분이 매우 아쉽다.

```
df_apt['거래금액'] = df_apt['거래금액'] * 10000
df_apt['평당금액'] = (df_apt['거래금액']) / (df_apt['전용면적']/3.3)
```

그림7. 거래금액 컬럼 수정

'거래금액'컬럼은 단위가 10,000원 단위로 적용되어 있었다. 이를 Web API 형식과 맞춰주기 위해 그림7과 같은 작업을 수행하였다. '평당금액'은 거래금액에서 전용면적과 3.3㎡당 1평이기 때문에 위와 같이 코드를 작성하였다.

```
####================================위도/경도=============================================
df_road = pd.read_csv('C:/Users/Admin/Desktop/PBL수업내용/__MINI/__Estimate자료/mini_dataset/seoul_axis.csv', encoding='cp949')
df_mindistance = pd.read_csv('C:/Users/Admin/Desktop/minDistance.csv')
new_road = []
for i in range(0, len(df_apt['도로명'])):
    rn = df_apt['도로명'][i].split(' ')[0]
    new_road.append(rn)

df_apt['수정된 도로명'] = new_road
roadInfo = list(df_road['도로명'])
lat = list(df_road['위도'])
lon = list(df_road['경도'])
```

그림8. 위도/경도 및 역과의 최소거리 데이터 추출

그림8은 이미 도로명, 건물명 기준으로 구해진 위/경도 값을 통해 아파트 기준 가장 가까운 역을 탐색하여 저장해놓은 CSV파일을 호출한다.

```
latitude_list = []
longitude_list = []
for i in range(0, len(df_apt['수정된 도로명'])):
    nr = new_road[i]
    try:
        indexN = roadInfo.index(nr)
        latitude_list.append(lat[indexN])
        longitude_list.append(lon[indexN])
    except ValueError as e:
        latitude_list.append(None)
        longitude_list.append(None)
```

그림9. 검색되지 않는 도로명.

그림9는 도로명 및 건물명으로 검색되는 데이터는 위/경도 데이터를 한 리스트에 저장하고, 검색되지 않은 도로명을 None값을 대입하여 처리하였다. 이후 None값을 포함한 Row는 모두 Drop시키기 편하도록 작업을 진행하였다.

```
df_apt['최소거리'] = df_mindistance['최소거리']
df_apt['역명'] = df_mindistance['역명']
df_apt['위도'] = latitude_list
df_apt['경도'] = longitude_list
df_apt = df_apt.drop(['수정된 도로명'], axis=1)
df_apt = df_apt.dropna(subset = ['위도'], how = 'any', axis=0)

for i, row in df_apt.iterrows() :
    if df_apt.at[i, '최소거리'] > 0.5 :
        df_apt.at[i, '역세권여부'] = 0
    else : df_apt.at[i,'역세권여부'] = 1


df_apt.to_csv('C:/Users/Admin/Desktop/total_apt.csv', encoding='utf-8-sig', index=False)
```

그림10. 역세권 여부 판별

그림 10은 마지막으로 CSV파일 생성에 앞서 필요한 컬럼을 추가하고, 불필요한 Row를 제거한다. 또한, '역세권여부'를 판별하기 위해 500m를 기준으로 아파트 기준 500m 이내에 지하철 역이 존재한다면 1, 없으면 0으로 표기하였다.

6-2-2 Web API에서 DATA 불러오기

```
#### ================================WEB API=======================================
max_size = 13
# 사용하고자 하는 실시간 API Page는 총 1000 Page지만,
# 이전 데이터는 csv파일로 받아 처리를 했기 때문에 앞에 5Page만 실시간으로 가져올 계획.

column_list = ['BJDONG_NM', 'DEAL_YMD', 'BLDG_AREA', 'FLR_INFO',
               'BLDG_MUSE_NM', 'OBJ_AMT', 'BLDG_NM', 'BUILD_YEAR']
# API가 포함하고있는 Coloum명
column_name = ['법정동명', '거래날짜', '전용면적', '층정보',
               '건물주용도', '거래금액', '건물명', '건축년도']
```

그림11. Web에서 필요한 컬럼 정의

Web에 연결하여 우리가 필요한 컬럼을 미리 조사하여 list로 작성하였다. 그림 11에서 보이는바와 같이 column_name변수에 변경할 컬럼명을 미리 대입하는 작업을 거쳤다.

```python
result = []
data = []
default_date = 0
old_date = 0
new_date = 0
for page_index in range(1, max_size) :
    if max_size - page_index <3 :
        break
    try :
        print('Page : ', page_index, '/', max_size-3)
        df = pdx.read_xml("http://openapi.seoul.go.kr:8088/634472564a68726c313232547272786f/xml/landActualPriceInfo/"
                          +str(page_index) + '/' + str(max_size), ['landActualPriceInfo', 'row'])
        for i in range(0, df.size) :
            for j in column_list :
                data.append(df[i][0][j])        # RangeIndex
            result.append(data)
            data = []
    except KeyError :
        print('ERR!')
        pass
```

그림12. Web 연결 및 데이터 받아오기

그림 12는 Web에서 제공하는 xml파일을 read하여 반복문을 통해 각 페이지별 모든 item을 list에 저장하는 작업을 나타낸다. 이러한 작업을 통해 거래별 정보를 취득할 수 있다.

이후, 모든 작업은 DB에 넣는 작업과 동일하게 진행하면 DB에 저장된 데이터와 동일한 컬럼 형식으로 데이터를 저장할 수 있다.

6-3. 데이터 적재

- mariaDB

```sql
# csv 컬럼에 맞는 컬럼명 구성 데이터 크기 지정

create database apart;
create table apartzz (시군구 char(50), 법정동명 char(50), 번지 char(50), 건물명 char(50), 전용면적 double, 거래날짜 datetime, 거래금액 in
load data local infile '/home/minidata/total_apt.csv' into table apartzz fields terminated by ',' enclosed by '"' lines terminated b

# csv import

load data local infile '/home/tdata/total_apt.csv'
into table apartzz
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 lines ;
```

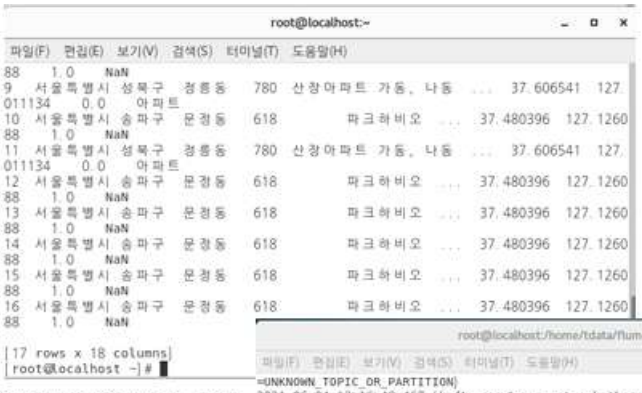
MariaDB [apart]> select * from apartzz limit 5;

시군구	법정동명	번지	건물명	전용면적	거래날짜	거래금액	층정보	건축년도	도로명	신축구분	평당금액	최소거리	역명	위도	경도	역세권여부
서울특별시 중랑구	상봉동	63	건영2차아파트	84.96	2016-01-15 00:00:00	358000000	19	1996	신내로7나길 24	구축	13905367.23	1.06863	상봉	37.6017409	127.095084	0
서울특별시 은평구	불광동	643	북한산 힐스테이트7차	84.891	2016-01-26 00:00:00	552000000	15	2011	통일로 796	신축	21458105.1	0.68921	녹번	37.5953578	126.9391898	0
서울특별시 성동구	행당동	345	두산	84.87	2016-01-18 00:00:00	450000000	17	1997	고산자로16길 8-1	구축	17497348.89	0.44414	왕십리	37.5654856	127.0370066	1
서울특별시 은평구	불광동	643	북한산 힐스테이트7차	59.981	2016-01-19 00:00:00	467000000	11	2011	통일로 796	신축	25693136.16	0.68921	녹번	37.5953578	126.9391898	0
서울특별시 은평구	불광동	643	북한산 힐스테이트7차	84.947	2016-01-09 00:00:00	480000000	1	2011	통일로 796	신축	18646921.02	0.68921	녹번	37.5953578	126.9391898	0

들어온 데이터 확인 (select * from apt_price limit 5;)

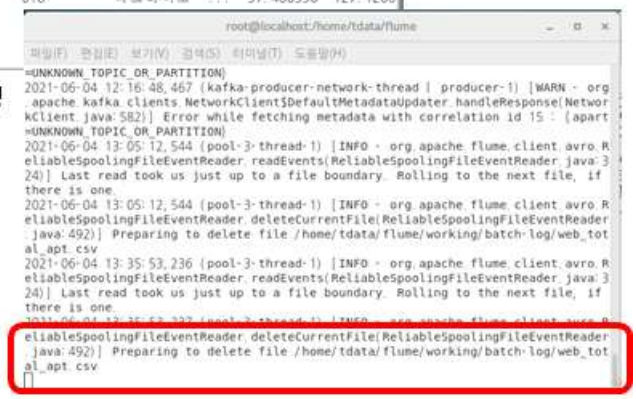
- Flume (OpenAPI data)

새롭게 들어오는 OpenAPI의 부동산 data를 플럼으로 감시하고 새롭게 수집된 데이터가 Kafka의 Topic으로 저장되도록 Python Code로 자동화하여 수행하도록 하였다.

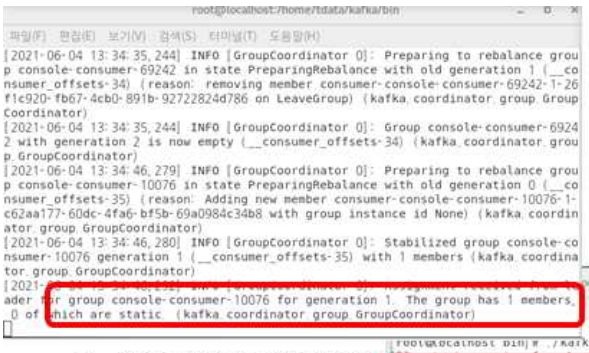


1 . 파이썬 파일 플럼 실행

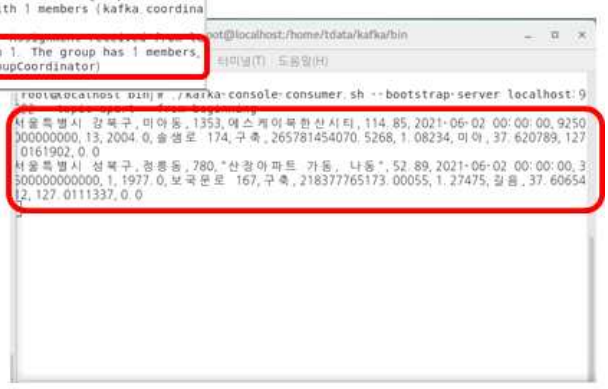
중복 message 제거



2 . 플럼 (데이터 수집 확인)



3 . Kafka (Topic 생성 확인)



4 . Kafka consumer (Topic 내용 확인)

- Logstash (Elasticsearch와 kafka로 DB데이터 전송)

logstash폴더의 conf파일 생성하여 최초 1회만 raw data 전송시킨다.

filter에 위도와 경도를 맵핑하여 kibana 시각화 시 map으로 볼 때 location으로 시각화가 가능하다.

단. conf파일 실행 전에 폴더에 맵핑할 apart.json파일을 위치(/root/apart.json)지정해주고 elasticsearch의

$curl -i -X PUT 'http://localhost:9200/_template/db?pretty' -H 'Content-Type: application/json'

--data-binary "@/root/apart.json"

위의 명령어를 수행하고 logstash_es_kafka.conf를 실행 시켜야 한다.

```
input {
    jdbc {
      jdbc_driver_library => "/usr/java/mysql-connector-java-8.0.18.jar"
      jdbc_connection_string => "jdbc:mysql://localhost:3306/apart?serverTimezone=Asia/Seoul"
      jdbc_driver_class => "com.mysql.cj.jdbc.Driver"
      jdbc_user => "root"
      jdbc_password => "sd0241"
      statement => "SELECT * FROM apartzz"
    }
}
filter {
    if [위도] and [경도] {
          mutate {
                  rename => {
                              "위도" => "[location][lat]"
                              "경도" => "[location][lon]"
                  }
          }
          mutate {
                  convert => {"[location]" => "float"}
          }
    }
}
output {
        kafka  {
                bootstrap_servers => "http://localhost:9092"
                topic_id => "apart"
                codec => json
        }
        elasticsearch {
                action => "index"
                hosts => "localhost:9200"
                index => "db"
                template => "/etc/logstash/esmapping.json"
                template_name => "db"
                template_overwrite => "true"
        }
        stdout{
                codec => rubydebug
        }
```

Kibana (Map) 시 각 화

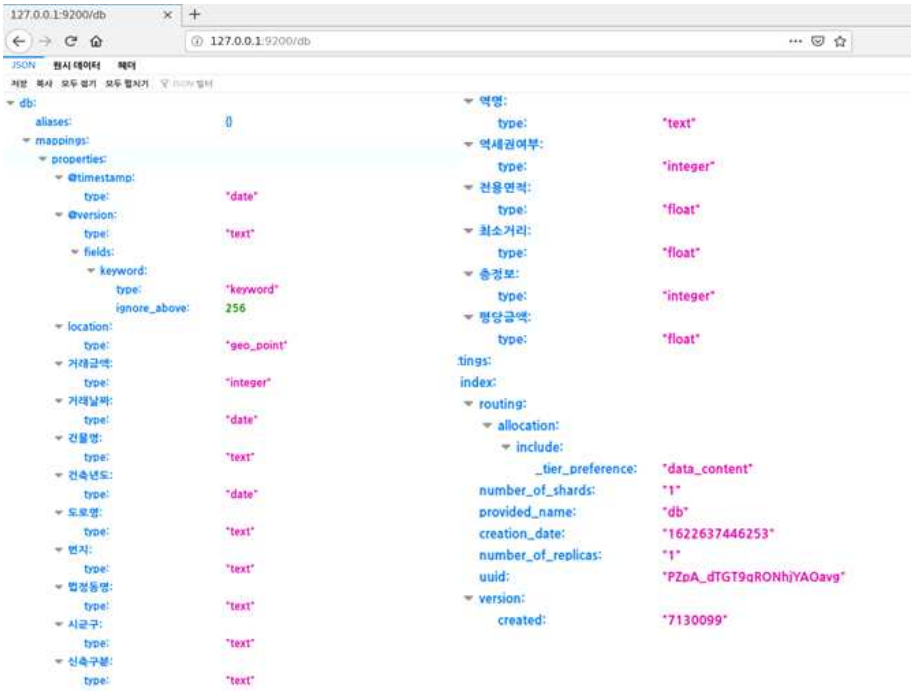
Location : Geo point 생성에 필요

[그림] logstash_es_kafka.conf

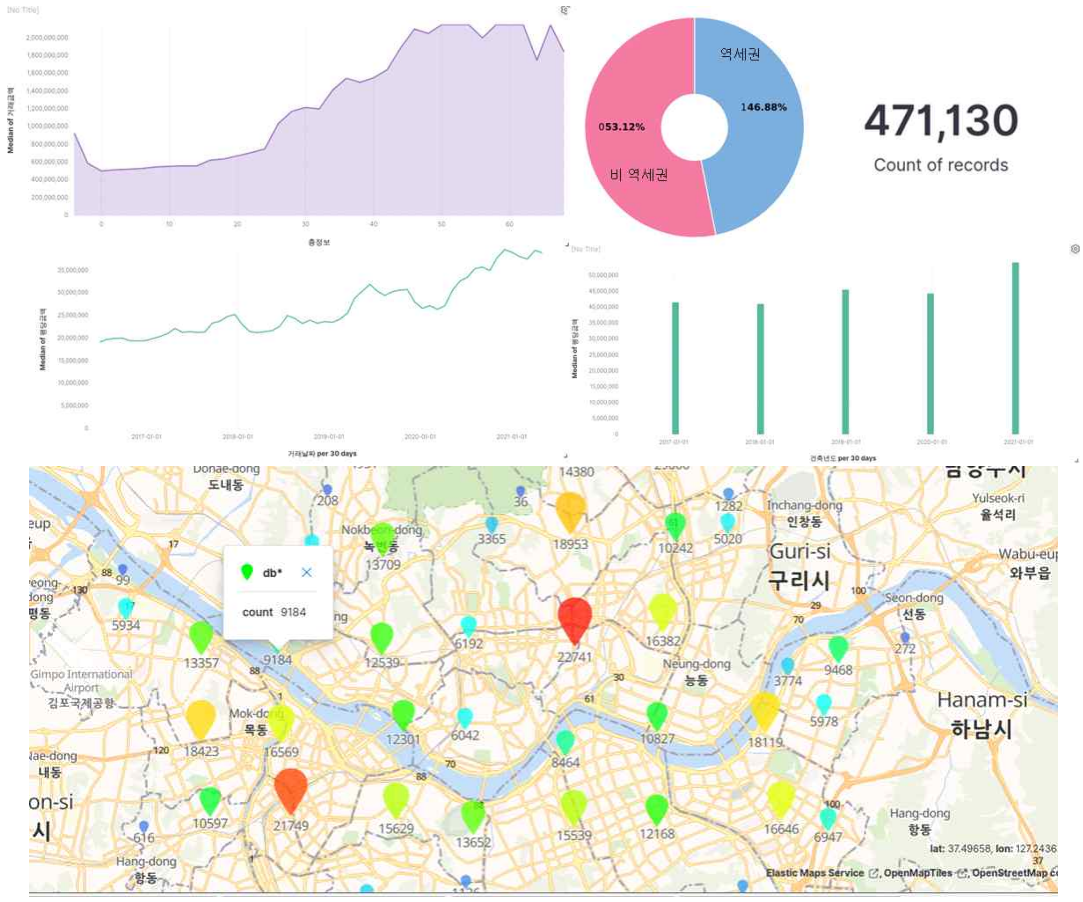
```
{
   "index_patterns":["db*"],
   "mappings":{
         "properties" : {
                        "시군구" : { "type" : "text" },
                        "법정동명" : { "type" : "text" },
                        "번지" : { "type" : "text" },
                        "건물명" : { "type" : "text" },
                        "전용면적" : { "type" : "float" },
                        "거래날짜" : { "type" : "date" },
                        "거래금액" : { "type" : "integer" },
                        "층정보" : { "type" : "integer" },
                        "건축년도" : { "type" : "date" },
                        "도로명" : { "type" : "text" },
                        "신축구분" : { "type" : "text" },
                        "평당금액" : { "type" : "float" },
                        "최소거리" : { "type" : "float" },
                        "역명" : { "type" : "text" },
                        "location" : { "type" : "geo_point"},
                        "역세권여부" : {"type" : "integer" },
                        "@timestamp" : { "type" : "date" }
                }
        }
}
```

```
curl -i -X PUT 'http://localhost:9200/_template/db?pretty' -H 'Content-Type: application/json' --data-binary
"@root/apart.json" <--- json 생성후 매핑정보 필수로 넣어줘야됨!!
```

[그림] apart.json



[그림] elasticsearch의 mapping 확인



[그림] kibana 시각화

- Kafka의 consumer로 받아진 Topic 메시지 확인

```
>> kafka-console-consumer.sh  --bootstrap-server localhost:9092 --topic apart --from-beginning
```


{"번지":"145","층정보":13,"건물명":"내이처해밀","시군구":"서울특별시 영등포구","거래금액":140000000,"건축년도":"2011-12-31T15:00:00.000Z","평당금액":3.064473335E7,"location":{"lon":126.9153158,"lat":37.517936},"거래날짜":"2021-05-16T15:00:00.000Z","전용면적":15.076,"신축구분":"신축","법정동명":"영등포동1가","최소거리":0.18683,"도로명":"경인로114길 62","역세권여부":1,"@version":"1","@timestamp":"2021-06-02T12:20:35.738Z","역명":"신길"}
{"번지":"587","층정보":14,"건물명":"동대문더퍼스트데시앙","시군구":"서울특별시 동대문구","거래금액":919000000,"건축년도":"2018-12-31T15:00:00.000Z","평당금액":5.054988649E7,"location":{"lon":127.071342,"lat":37.5758212},"거래날짜":"2021-05-14T15:00:00.000Z","전용면적":59.9942,"신축구분":"신축","법정동명":"장안동","최소거리":1.37724,"도로명":"장한로27가길 37","역세권여부":0,"@version":"1","@timestamp":"2021-06-02T12:20:35.738Z","역명":"용마산"}
{"번지":"1257","층정보":15,"건물명":"럭키","시군구":"서울특별시 구로구","거래금액":680000000,"건축년도":"1991-12-31T15:00:00.000Z","평당금액":3.142857143E7,"location":{"lon":126.8894543,"lat":37.4877458},"거래날짜":"2021-04-30T15:00:00.000Z","전용면적":71.4,"신축구분":"구축","법정동명":"구로동","최소거리":0.27049,"도로명":"도림로 107","역세권여부":1,"@version":"1","@timestamp":"2021-06-02T12:20:35.738Z","역명":"남구로"}
^CProcessed a total of 471130 messages
```

- Spark session과 sql로 kafka topic 메시지 불러와 데이터 프레임 생성

**spark - kafka 연동 확인**

```
In [1]: import pyspark
        import pyspark.sql
        from pyspark.sql import *
        import time
        from pyspark.sql.functions import *
        from pyspark.sql.types import *
        from pyspark.sql import SparkSession

In [2]: spark = SparkSession.builder.master("local").appName("api").getOrCreate()

In [12]: df = spark.read.format("kafka").option("kafka.bootstrap.servers", "localhost:9092")\
             .option("subscribe", "apart").load()

In [13]: df.show()
```

key	value	topic	partition	offset	timestamp	timestampType
null	[7B 22 EB B2 88 E...]	apart	0	0	2021-06-02 21:15:...	0
null	[7B 22 EB B2 88 E...]	apart	0	1	2021-06-02 21:15:...	0
null	[7B 22 EB B2 88 E...]	apart	0	2	2021-06-02 21:15:...	0
null	[7B 22 EB B2 88 E...]	apart	0	3	2021-06-02 21:15:...	0
null	[7B 22 EB B2 88 E...]	apart	0	4	2021-06-02 21:15:...	0
null	[7B 22 EB B2 88 E...]	apart	0	5	2021-06-02 21:15:...	0
null	[7B 22 EB B2 88 E...]	apart	0	6	2021-06-02 21:15:...	0
null	[7B 22 EB B2 88 E...]	apart	0	7	2021-06-02 21:15:...	0
null	[7B 22 EB B2 88 E...]	apart	0	8	2021-06-02 21:15:...	0
null	[7B 22 EB B2 88 E...]	apart	0	9	2021-06-02 21:15:...	0
null	[7B 22 EB B2 88 E...]	apart	0	10	2021-06-02 21:15:...	0
null	[7B 22 EB B2 88 E...]	apart	0	11	2021-06-02 21:15:...	0
null	[7B 22 EB B2 88 E...]	apart	0	12	2021-06-02 21:15:...	0

Data Type > Binary 상태
```

- Binary 상태의 Data Type 변경

```
df_struct = StructType()₩
            .add("시군구", StringType())₩
            .add("법정동명", StringType())₩
            .add("번지", StringType())₩
            .add("건물명", StringType())₩
            .add("전용면적", DoubleType())₩
            .add("거래날짜", DateType())₩
            .add("거래금액", IntegerType())₩
            .add("층정보", DoubleType())₩
            .add("건축년도", DateType())₩
            .add("도로명", StringType())₩
            .add("평당금액", DoubleType())₩
            .add("최소거리", DoubleType())₩
            .add("역명", StringType())₩
            .add("역세권여부", IntegerType())
```

```
dfstr1 = df.selectExpr("CAST(value AS STRING)")
```

```
df_struct = dfstr1.select(from_json(col("value"), df_struct).alias("df"))
dfnew1 = df_struct.select("df.*")
```

```
test = dfnew1.toPandas()
test
```

	시군구	법정동명	번지	건물명	전용면적	거래날짜	거래금액	층정보	건축년도	도로명	평당금액	최소거리	역명	역세권여부
0	서울특별시 은평구	불광동	643	북한산 힐스테이트7차	84.8910	2016-01-25	552000000	15.0	2010-12-31	통일로 796	21458105.10	0.68921	녹번	0
1	서울특별시 중랑구	상봉동	63	건영2차아파트	84.9600	2016-01-14	358000000	19.0	1995-12-31	신내로7나길 24	13905367.23	1.06863	상봉	0
2	서울특별시 은평구	불광동	643	북한산 힐스테이트7차	59.9810	2016-01-18	467000000	11.0	2010-12-31	통일로 796	25693136.16	0.68921	녹번	0

- SparkML에 필요한 데이터만 남기고 컬럼 정리

```
dfnew1 = dfnew1.drop("거래날짜")
dfnew1 = dfnew1.drop("법정동명")
dfnew1 = dfnew1.drop("번지")
dfnew1 = dfnew1.drop("건물명")
dfnew1 = dfnew1.drop("전용면적")
dfnew1 = dfnew1.drop("거래금액")
dfnew1 = dfnew1.drop("도로명")
dfnew1 = dfnew1.drop("역명")
dfnew1 = dfnew1.drop("역세권여부")
```

```
split_col = pyspark.sql.functions.split(dfnew1['건축년도'], '-')
dfnew1 = dfnew1.withColumn('built_year', split_col.getItem(0))
dfnew1 = dfnew1.withColumn('built_year', dfnew1['built_year'].cast(DoubleType()))
dfnew1 = dfnew1.drop('건축년도')
split_col = pyspark.sql.functions.split(dfnew1['시군구'], ' ')
dfnew1 = dfnew1.withColumn('gu', split_col.getItem(1))
dfnew1 = dfnew1.drop('시군구')
```

6-4. Machine Learning (SparkML)

1) 데이터 벡터화 및 MinMaxScaler 적용

```
from pyspark.ml.feature import VectorAssembler

vectorAssembler = VectorAssembler(inputCols = ["층정보", "최소거리", "built_year"], outputCol = "Features")
vpp_df = vectorAssembler.transform(dfnew1)
vpp_df.show(2, False)
```

```
+------+------------+--------+----------+------+-------------------+
|층정보|평당금액    |최소거리|built_year|gu    |Features           |
+------+------------+--------+----------+------+-------------------+
|15.0  |2.14581051E7 |0.68921 |2010.0    |은평구|[15.0,0.68921,2010.0]|
|19.0  |1.390536723E7|1.06863 |1995.0    |중랑구|[19.0,1.06863,1995.0]|
+------+------------+--------+----------+------+-------------------+
only showing top 2 rows
```

```
from pyspark.ml.feature import MinMaxScaler

minMax = MinMaxScaler().setMin(-1).setMax(10).setInputCol("Features").setOutputCol("MinMaxScaler")
fittedminMax = minMax.fit(vpp_df)
minMaxdf=fittedminMax.transform(vpp_df)
minMaxdf.show(5, False)
```

```
+------+------------+--------+----------+------+-------------------+-----------------------------------------------------------+
|층정보|평당금액    |최소거리|built_year|gu    |Features           |MinMaxScaler                                               |
+------+------------+--------+----------+------+-------------------+-----------------------------------------------------------+
|15.0  |2.14581051E7 |0.68921 |2010.0    |은평구|[15.0,0.68921,2010.0]|[1.75,1.1751138438231403,8.166666666666666]                |
|19.0  |1.390536723E7|1.06863 |1995.0    |중랑구|[19.0,1.06863,1995.0]|[2.3611111111111116,2.410047845474209,5.416666666666666]  |
|11.0  |2.569313616E7|0.68921 |2010.0    |은평구|[11.0,0.68921,2010.0]|[1.1388888888888893,1.1751138438231403,8.166666666666666] |
|1.0   |1.864692102E7|0.68921 |2010.0    |은평구|[1.0,0.68921,2010.0] |[-0.38888888888888884,1.1751138438231403,8.166666666666666]|
|9.0   |2.126373821E7|0.68921 |2010.0    |은평구|[9.0,0.68921,2010.0] |[0.8333333333333335,1.1751138438231403,8.166666666666666] |
+------+------------+--------+----------+------+-------------------+-----------------------------------------------------------+
```

머신러닝을 수행하기 위해 컬럼을 벡터화 하고 MinMax scaler를 적용하였다.

1) Decision Tree Regression Model

```
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.evaluation import RegressionEvaluator
```

```
dt = DecisionTreeRegressor(featuresCol="MinMaxScaler", labelCol="평당금액")
dt_model = dt.fit(trainingData)
dt_predictions = dt_model.transform(testData)
dt_predictions.show(5, False)
```

```
+------+------------+--------+----------+------+-------------------+-----------------------------------------------------------+-----------------------+
|층정보|평당금액    |최소거리|built_year|gu    |Features           |MinMaxScaler                                               |prediction             |
+------+------------+--------+----------+------+-------------------+-----------------------------------------------------------+-----------------------+
|-3.0  |2.357375111E7|0.18069 |2009.0    |은평구|[-3.0,0.18069,2009.0]|[-1.0,-0.4800140843820182,7.9833333333333325]            |3.359447423
9072904E7|
|-2.0  |1.439619182E7|0.18069 |2009.0    |은평구|[-2.0,0.18069,2009.0]|[-0.8472222222222222,-0.4800140843820182,7.9833333333333325]|3.3
59447423072904E7|
|-2.0  |1.836796128E7|1.06651 |2013.0    |구로구|[-2.0,1.06651,2013.0]|[-0.8472222222222222,2.40314768184683,8.716666666666667]  |3.183
419445083042E7 |
|-2.0  |4.104710608E7|0.94594 |2018.0    |성북구|[-2.0,0.94594,2018.0]|[-0.8472222222222222,2.0107171495104449,9.633333333333333]|3.11
2064391548149E7 |
|-2.0  |6.500130446E7|0.17712 |2015.0    |성동구|[-2.0,0.17712,2015.0]|[-0.8472222222222222,-0.4916336995469919,9.083333333333332]|5.3
138133622665 41E7 |
+------+------------+--------+----------+------+-------------------+-----------------------------------------------------------+-----------------------+
only showing top 5 rows
```

```
dt_evaluator = RegressionEvaluator(labelCol="평당금액", predictionCol="prediction", metricName="rmse")
dt_rmse = dt_evaluator.evaluate(dt_predictions)
print("The RMSE of Decision Tree regression Model is {}".format(dt_rmse))
#The RMSE of Decision Tree regression Model is 13788735.979753083
```

```
The RMSE of Decision Tree regression Model is 13773355.316648483
```

RMSE(오차)가 대략 1377만원정도 나온 것을 알 수 있다.

2)RandomForest Regression Model

```
from pyspark.ml.regression import RandomForestRegressor
```

```
rf = RandomForestRegressor(featuresCol="MinMaxScaler", labelCol="평당금액")
model = rf.fit(trainingData)
rf_predictions = model.transform(testData)
rf_predictions.show(5)
```

```
+------+-----------+--------+----------+------+--------------------+--------------------+--------------------+
|층정보|  평당금액|최소거리|built_year|    gu|            Features|       MinMaxScaler|          prediction|
+------+-----------+--------+----------+------+--------------------+--------------------+--------------------+
|  -3.0|2.357375111E7| 0.18069|    2009.0|은평구|[-3.0,0.18069,200...|[-1.0,-0.48001408...| 3.134068945102297E7|
|  -2.0|1.439619182E7| 0.18069|    2009.0|은평구|[-2.0,0.18069,200...|[-0.8472222222222...| 3.134068945102297E7|
|  -2.0|1.836796128E7| 1.06651|    2013.0|구로구|[-2.0,1.06651,201...|[-0.8472222222222...|2.926067227083273 6E7|
|  -2.0|4.104710608E7| 0.94594|    2018.0|성북구|[-2.0,0.94594,201...|[-0.8472222222222...| 2.935246058937944E7|
|  -2.0|6.500130446E7| 0.17712|    2015.0|성동구|[-2.0,0.17712,201...|[-0.8472222222222...|3.335919055807618 4E7|
+------+-----------+--------+----------+------+--------------------+--------------------+--------------------+
only showing top 5 rows
```

```
rf_evaluator = RegressionEvaluator(labelCol="평당금액", predictionCol="prediction", metricName="rmse")
rf_rmse = rf_evaluator.evaluate(rf_predictions)
print("The RMSE of Random Forest regression Model is {}".format(rf_rmse))
#The RMSE of Random Forest regression Model is 14155522.941261763
```

```
The RMSE of Random Forest regression Model is 14226718.955583578
```

RMSE(오차)가 대략 1422만원정도 나온 것을 알 수 있다.

3) Gradient-Boosted Tree Regression Model

Gradient-Boosted Tree Regression model

In [28]:
```
from pyspark.ml.regression import GBTRegressor
```

In [29]:
```
gbt = GBTRegressor(featuresCol="MinMaxScaler", labelCol="평당금액")
gbt_model = gbt.fit(trainingData)
gbt_predictions = gbt_model.transform(testData)
gbt_predictions.show(5, False)
```

```
+------+-----------+--------+----------+------+--------------------+----------------------------------------------------------------------------------------------------+--------------------+
|층정보|평당금액   |최소거리|built_year|gu    |Features            |MinMaxScaler                                                                                        |prediction          |
+------+-----------+--------+----------+------+--------------------+----------------------------------------------------------------------------------------------------+--------------------+
|-3.0  |2.357375111E7|0.18069 |2009.0    |은평구|[-3.0,0.18069,2009.0]|[-1.0,-0.4800140843820182,7.9833333333333325]                                                       |2.860609836553856E7 |
|-2.0  |1.439619182E7|0.18069 |2009.0    |은평구|[-2.0,0.18069,2009.0]|[-0.8472222222222222,-0.4800140843820182,7.9833333333333325]                                        |2.860609836553856E7 |
|-2.0  |1.836796128E7|1.06651 |2013.0    |구로구|[-2.0,1.06651,2013.0]|[-0.8472222222222222,2.40314768184683,8.716666666666667]                                            |3.251059398631246E7 |
|-2.0  |4.104710608E7|0.94594 |2018.0    |성북구|[-2.0,0.94594,2018.0]|[-0.8472222222222222,2.010717149510449,9.633333333333333]                                           |3.395562080349816E7 |
|-2.0  |6.500130446E7|0.17712 |2015.0    |성동구|[-2.0,0.17712,2015.0]|[-0.8472222222222222,-0.4916336995469919,9.083333333333332]                                         |5.23176097785527 56E7|
+------+-----------+--------+----------+------+--------------------+----------------------------------------------------------------------------------------------------+--------------------+
only showing top 5 rows
```

In [30]:
```
gbt_evaluator = RegressionEvaluator(labelCol="평당금액", predictionCol="prediction", metricName="rmse")
gbt_rmse = gbt_evaluator.evaluate(gbt_predictions)
print("The RMSE of Random Forest regression Model is {}".format(gbt_rmse))
#The RMSE of Random Forest regression Model is 12996387.694069464
```

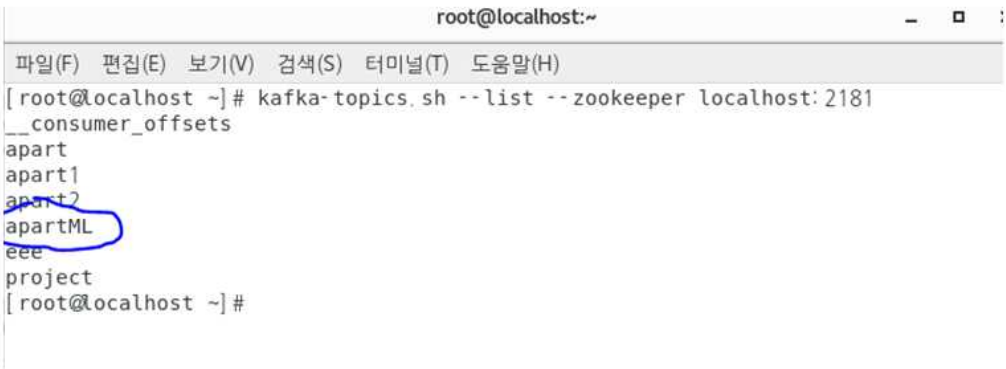
```
The RMSE of Random Forest regression Model is 12869558.405286547
```

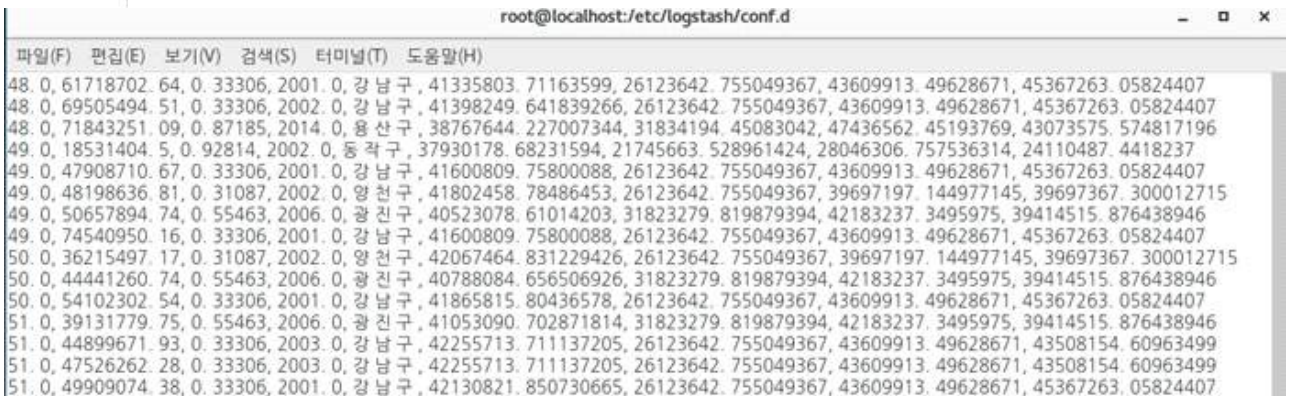
RMSE(오차)가 여러 Model중 가장 작은 대략 1287만원정도 나온 것을 알 수 있다.

6-5. SparkML 결과 Kafka로 전송 및 확인

```
========== kafka로 topic 저장하기 ==============
import pandas as pd

# 예측모델을 pandas 데이터 셋으로 변경
lr_dfpd = lr_predictions.toPandas()
dt_dfpd = dt_predictions.toPandas()
rf_dfpd = rf_predictions.toPandas()
gbt_dfpd = gbt_predictions.toPandas()

#prediction 컬럼 이름 변경
lr_dfpd.columns = ['층정보', '평당금액', '최소거리', 'built_year', 'gu', ' Features', 'MinMaxScaler', 'lr_prediction']
dt_dfpd.columns = ['층정보', '평당금액', '최소거리', 'built_year', 'gu', ' Features', 'MinMaxScaler', 'dt_prediction']
rf_dfpd.columns = ['층정보', '평당금액', '최소거리', 'built_year', 'gu', ' Features', 'MinMaxScaler', 'rf_prediction']
gbt_dfpd.columns = ['층정보', '평당금액', '최소거리', 'built_year', 'gu', ' Features', 'MinMaxScaler', 'gbt_prediction']

# pd.concat으로 병합
df_result = pd.concat([lr_dfpd["층정보"], lr_dfpd["평당금액"], lr_dfpd["최소거리"], lr_dfpd["built_year"], lr_dfpd["gu"], lr_dfpd
["lr_prediction"], dt_dfpd["dt_prediction"], rf_dfpd["rf_prediction"], gbt_dfpd["gbt_prediction"]], axis=1)
# df_result.head()

# Spark dataframe으로 적용 후, 새로운 value 라는 컬럼을 생성 -> 모든 컬럼의 내용을 csv 형식으로 담는 lambda 함수 사용하여 적용
df_final = spark.createDataFrame(df_result)
mergeCols = udf(lambda 층정보, 평당금액, 최소거리, built_year, gu, lr_prediction, dt_prediction, rf_prediction, gbt_prediction :str
(층정보) +','+  str(평당금액) +','+ str(최소거리) +','+ str(built_year) +','+ gu +','+ str(lr_prediction) +','+ str(dt_prediction) +',
'+ str(rf_prediction) +','+ str(gbt_prediction))
df_final = df_final.withColumn("value", mergeCols(col("층정보"), col("평당금액"), col("최소거리"), col("built_year"), col("gu"), col
("lr_prediction"), col("dt_prediction"), col("rf_prediction"), col("gbt_prediction")))
#print(df_final.tail(5))


# kafka에 topic 생성 후 전송
df_final.selectExpr("CAST(value AS STRING)").write.format("kafka") \
              .option("kafka.bootstrap.servers", "localhost:9092") \
              .option("topic", "apartML").save()
```

[그림] SparkML predict값 kafka로 전송





[그림] kafka에서 생성된 topic 확인

6-6. Grafana 시각화

7. 개발 중 문제점

- 전체 시스템 실시간 자동화를 해내지 못했다. (CronTab 기능 활성화 미적용)
- SparkML 실시간 분석 자동화를 시도했으나 구현을 완성하지 못하였다.
- ML feature 및 좀 더 분석하고 고민 해볼 시간이 없어서 Model에 대한 완성도가 아쉬웠다.

8. 소감 및 아쉬운 점

- 각각의 프로그램을 완벽하게 이해하지 못한 상태로 프로젝트를 수행하게 되어 여러 가지 미흡했던 것 같습니다.
- 처음으로 프로젝트를 수행하다보니 시간 및 역할 분배가 알맞게 이뤄지기 어려웠고 짧게 느껴지고 프로젝트를 완벽하게 끝내지 못해 너무 아쉬웠습니다.
- 여러 가지 프로그램을 사용해볼 수 있었고 생각했던 대로 잘 되지 않았지만 여러 가지 Error들을 보면서 짧지만 좋은 경험과 프로젝트를 통해 조금 더 성장 할 수 있었던 시간이 되었던 것 같습니다.