

# 차량 로그 데이터 이용 데이터 엔지니어링

## 목차

- 구성 및 설정
- 수집
- 적재
- 탐색 및 처리
- 분석
  - 군집 분석
  - 이상 징후 판별
- 발생한 문제 및 해결
- 중점적으로 고려한 사항

## 구성 및 설정

### ▼ 주요 목표

- 차량의 다양한 장치로부터 발생하는 로그 파일을 수집하여 기능 별 상태 점검(배치)
- 운전자의 운행 정보가 담긴 로그를 실시간으로 수집하여 주행 패턴 분석

### ▼ 가상 머신 설정

- Oracle Virtual Box : 서버 2대 (Server01, Server02)
- OS : CentOS 6
- RAM : Server01 - 4GB, Server02 - 4GB

### • Cloudera Manager 사용

- 통일된 인터페이스를 통해 구성과 리소스를 간편하게 튜닝
- 다양한 환경, 다중 클러스터 관리를 용이하게 함
- Hadoop 모니터링 서비스

## 수집

### ▼ 수집 사용 기술

- **Flume** : `Source → Channel → Sink` 의 구조를 가지며, 데이터를 수집하기 위한 기능을 담당
- **Kafka** : 대규모 메시지 성 데이터 중계, `Producer` 와 `Consumer` 로 나뉘며 이를 중계하는 `Broker` 가 중간에 존재
- **Storm** : 데이터를 In-memory 상에서 병렬 처리하기 위한 소프트웨어, Kafka로부터 받은 데이터를 각각 HBase, Redis로 나누어서 전달
- **Esper** : 실시간 스트리밍 데이터로 복잡한 이벤트 처리가 필요할 때 사용하는 룰 엔진

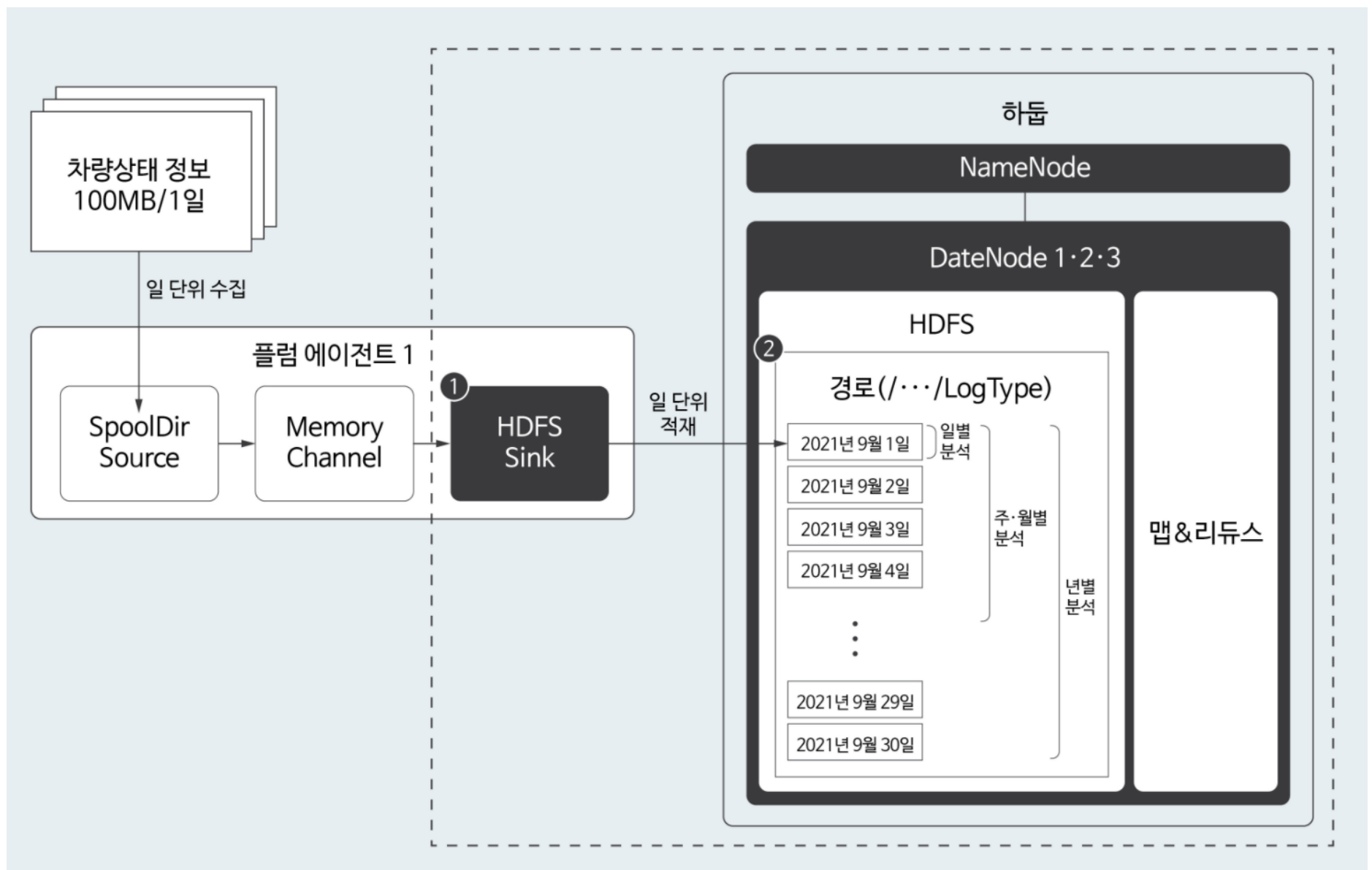
## 적재

### ▼ 적재 사용 기술

- **HDFS** : 파일을 블록 단위로 나누어서 각 클러스터에 분산 저장
- **Zookeeper** : 분산 코디네이터, 분산 환경에서 작동되는 작업들을 감시/감독(Supervisor)

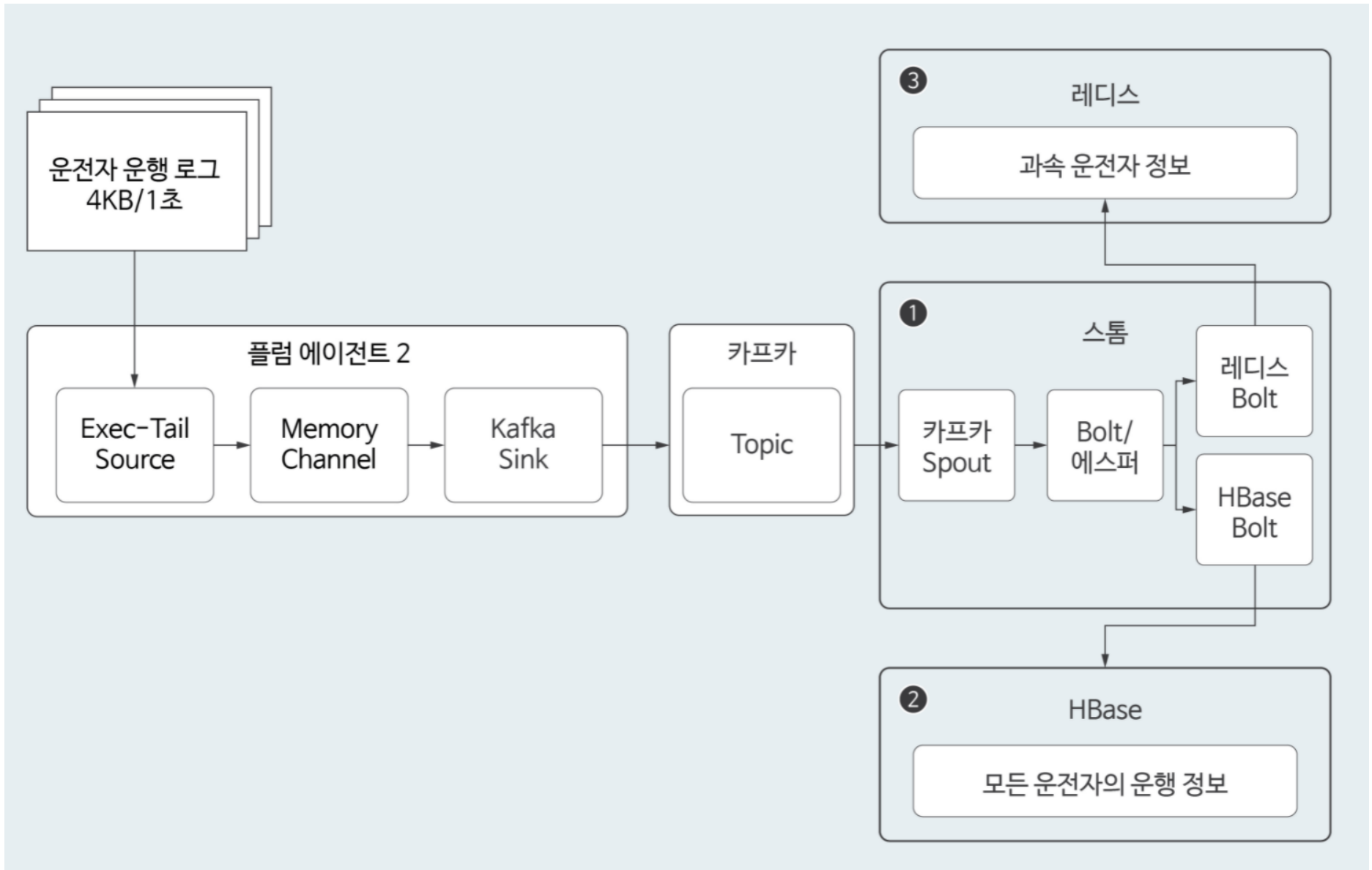
- **HBase**
  - Hadoop 기반 **Column 지향** NoSQL
  - Schema 변경이 자유롭다.
  - Region 분산 서버로 **샤딩**과 **복제** 지원 → 성능/안정성 향상
- **Redis**
  - 분산 Cache
  - **In-memory** Data Grid Software
  - 실시간 성 데이터 중 일부만 HBase에 저장하기 전에 Redis에 저장할 필요가 있어서 사용(가속 데이터를 Redis에 저장)

[ 배치성 수집 → 적재 ]



- **일** 단위로 Flume에서 수집한 데이터를 날마다 설정한 시간에 **HDFS**에 적재

[ 실시간 수집 → 적재 ]



- 실시간 수집의 경우 **Flume**에서 수집 이후 바로 **HBase**와 같은 곳에 적재를 진행하게 되면, **Fault Tolerance**를 보장하지 못한다. 적재 시 **HBase**에 오류가 발생한다면, 실시간으로 수집되고 있는 데이터들이 손실되거나 예상하지 못한 문제가 발생할 수 있다.
- 이런 점을 해결하기 위해서 중간에 **Kafka**를 거쳐 **Kafka**의 높은 처리량, 신뢰성, 즉각적인 피드백을 통해 수집되고 있는 데이터가 손실되지 않고, 즉각적으로 장애에 대응할 수 있도록 구성하였다.

## 탐색 및 처리

### ▼ 탐색 및 처리 사용 기술

- **Hive**
  - 기존 방식: 적재된 데이터를 탐색/분석하기 위해 MapReduce를 주로 사용(복잡도 커짐, Java 이용 필요)
  - **Hive(SQL on Hadoop)**를 이용하여 MapReduce로 변환 및 실행 가능
    - Query Engine : 사용자가 입력한 Hive Query를 분석하여 실행 계획을 수립 → Hive Query를 MapReduce 코드로 변환 및 실행
  - **Interactive**(대화형) 방식에는 적합하지 않다.
- **Spark** : In-memory 방식을 통해 MapReduce보다 데이터를 더욱 효율적으로 처리(적은 데이터의 경우 Spark나 Impala가 Hive보다 유용)
- **Impala** : 대화형 쿼리를 위한 쿼리 엔진.
- **Oozie** : 예약 및 실행을 이용한 **Workflow** 구성
- **Hue** : Web UI를 이용하여 **HDFS** 및 Query를 간편하게 이용 가능

### 기존의 로그를 활용하여 추가 주제 영역 테이블 생성

- 스마트카 상태 모니터링 정보(**managed\_smartcar\_status\_info**)
  - **smartcar\_master\_over18**, **smartcar\_status\_info** 이용

- 스마트카 운전자 운행기록 정보( `managed_smartcar_drive_info` )
  - `smartcar_master_over18` , `smartcar_drive_info_2` 이용
- 이상 운전 패턴 스마트카 정보( `managed_smartcar_symptom_info` )
  - `managed_smartcar_drive_info` 이용
- 긴급 점검이 필요한 스마트카 정보( `managed_smartcar_emergency_check_info` )
  - `managed_smartcar_status_info` 이용
- 운전자의 차량 용품 구매 이력 정보( `managed_smartcar_item_buylist_info` )
  - `smartcar_master_over18` , `smartcar_item_buylist` 이용

## [ 결과 확인 ]

```

managed_smartcar_drive_info
managed_smartcar_emergency_check_info
managed_smartcar_item_buylist_info
managed_smartcar_status_info
managed_smartcar_symptom_info
smartcar_drive_info
smartcar_drive_info_2
smartcar_item_buylist
smartcar_master
smartcar_master2income
smartcar_master_over18
smartcar_status_info

```

## 분석 및 분석 환경

- 군집 분석
  - Mahout을 사용한 Canopy 분석(이후 K-means의 적절한 K값 찾기)
  - Canopy 분석의 결과 중 유효한 변수만 선정하여 Feature → **실루엣 분석**
- 차량 이상 징후 판별

### ▼ 군집 분석 시 사용 기술

- **Mahout** : 분산 처리, 확장성을 가진 라이브러리
- **Spark K-means**

## 군집 분석

### Mahout을 사용한 Canopy 분석

- 정보 생성

```

insert overwrite local directory '/home/pilot-pjt/mahout-data/clustering/input'
row format delimited
fields terminated by ' '
select
  car_number,
  case
    when (car_capacity < 2000) then '소형'
    when (car_capacity < 3000) then '중형'
    when (car_capacity < 4000) then '대형'
  end as car_capacity
  case
    when ((2016 - car_year) <= 2) then 'NEW'
    when ((2016 - car_year) <= 8) then 'NORMAL'
    else 'OLD'
  end as car_year

```

```
end as car_year,
car_model,
sex as owner_sex,
floor (cast(age as int) * 0.1) * 10 as owner_age,
marriage as owner_marriage,
job as owner_job,
region as owner_region
from smartcar_master
```

- Canopy 분석을 위한 입력 데이터

```
$ hdfs dfs -mkdir -p /pilot-pjt/mahout/clustering/input
$ cd /home/pilot-pjt/mahout-data/clustering/input
$ mv 000000_0 smartcar_master.txt

$ hdfs dfs -put smartcar_master.txt /pilot-pjt/mahout/clustering/input
```

```
$ mahout seq2sparse -i /pilot-pjt/mahout/clustering/output/seq -o /pilot-pjt/mahout/clustering/output/vec
-wt tf -s 5 -md 3 -ng 2 -x 85 --namedVector
```

- Canopy 군집 분석 (Centroid로부터 거리를 나타내는 t1, t2 옵션을 바꿔가면서 수행
  - Centroid ~ t2까지 : 해당 군집
  - t2 ~ t1까지 : 다른 군집의 데이터로 취합 가능

```
$ mahout canopy -i /pilot-pjt/mahout/clustering/output/vec/tf-vectors/ -o /
pilot-pjt/mahout/clustering/canopy/out -dm org.apache.mahout.common.distance.
SquaredEuclideanDistanceMeasure -t1 50 -t2 45 -ow
```

## Canopy 분석을 통해 K-means Clustering에 적절한 K 값 변수 발견

### Canopy 분석의 결과 중 유효한 변수만 선정하여 Feature → 실루엣 분석

- 차량 용량
- 차량 연식
- 차량 모델
- 성별
- 결혼 여부

```
val dsSmartCar_Master_12 = dsSmartCar_Master_11.drop("car_capacity").drop("car_year").drop("car_model").drop("sex").drop("age").drop("marriage").drop("job").drop("region").drop("features").withColumnRenamed("scaledfeatures", "features")
dsSmartCar_Master_12.show(5)
val Array(trainingData, testData) = dsSmartCar_Master_12.randomSplit(Array(0.7, 0.3))
```

| car_number | car_capacity_n | car_year_n | car_model_n | sex_n | age_n | marriage_n | job_n | region_n | features              |
|------------|----------------|------------|-------------|-------|-------|------------|-------|----------|-----------------------|
| A0001      | 0.0            | 2.0        | 0.0         | 0.0   | 4.0   | 0.0        | 3.0   | 14.0     | [0.0,1.0,0.0,0.0,...] |
| A0002      | 1.0            | 1.0        | 1.0         | 1.0   | 2.0   | 0.0        | 2.0   | 12.0     | [0.5,0.5,0.142857...  |
| A0003      | 1.0            | 1.0        | 5.0         | 0.0   | 3.0   | 1.0        | 5.0   | 2.0      | [0.5,0.5,0.714285...  |
| A0004      | 1.0            | 2.0        | 6.0         | 1.0   | 4.0   | 0.0        | 6.0   | 1.0      | [0.5,1.0,0.857142...  |
| A0005      | 0.0            | 0.0        | 7.0         | 1.0   | 3.0   | 0.0        | 6.0   | 0.0      | [0.0,0.0,1.0,1.0,...] |

- 군집 번호 별 차량 번호

```
val transKmeansModel = kmeansModel.transform(dsSmartCar_Master_12)
transKmeansModel.groupBy("prediction").agg(collect_set("car_number").as("car_number")).orderBy("prediction").show(200, false)
```

3, B0024, Q0035, W0042, P0085, D0058]

|24 | [K0087, E0024, U0099, V0031, M0074, C0006, N0090, L0006, B0085, M0094]

|25 | [M0047, W0011, X0024, F0080, O0073, Y0005, Y0082, O0082, Z0014, C0040, U0060, W0070]

|26 | [O0099, E0052, V0002, O0070, O0084, O0094, K0082, U0062, C0020, N0035, U0015, D0066, O0017, K0036, Z0001, L0016, M0030, M0052, T0055, N0094, L0039]

|27 | [T0028, P0036, P0032, D0087, R0038, A0008, R0068, H0016, G0045, O0095, T0017, W0093, F0006, Q0070, B0069, R0091, P0071, I0040, R0034, J0084, J0011, T0046, C0078, I0041, J0048, Z0094, P0063, S0071, B0046, V0038, Z0075, A0013, C0014]

|28 | [Z0098, X0050, S0025, O0087, N0096, C0088, L0060, B0033, N0052, R0065, Y0092, C0093, W0060, H0059, E0088]

|29 | [H0092, S0036, D0031, H0046, Z0034, X0048, B0025, E0065, P0006, K0027, L0093, L0081, G0015, S0062, P0018, Q0023, W0027]

|30 | [Q0048, S0004, J0026, D0019, R0035, K0040, Y0062]

|31 | [L0002, J0081, Z0004, O0009, P0005, T0095, F0039, K0018, L0052, R0099, P0093, W0002, Z0076, M0041]

|32 | [L0002, J0081, Z0004, O0009, P0005, T0095, F0039, K0018, L0052, R0099, P0093, W0002, Z0076, M0041]

- 실루엣 분석을 통해 군집이 잘 이루어 졌는지 판단

```
val evaluator = new ClusteringEvaluator()
val silhouette = evaluator.evaluate(transKmeansModel)

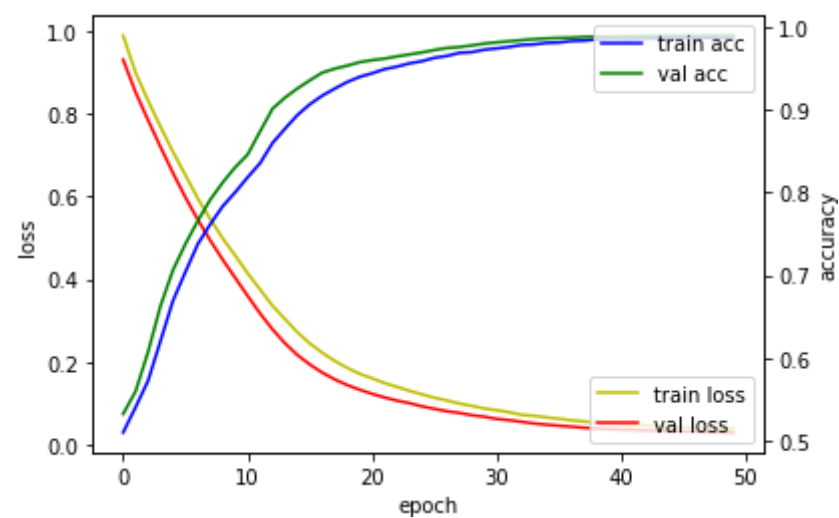
println(s"Silhouette Score = $silhouette")
```

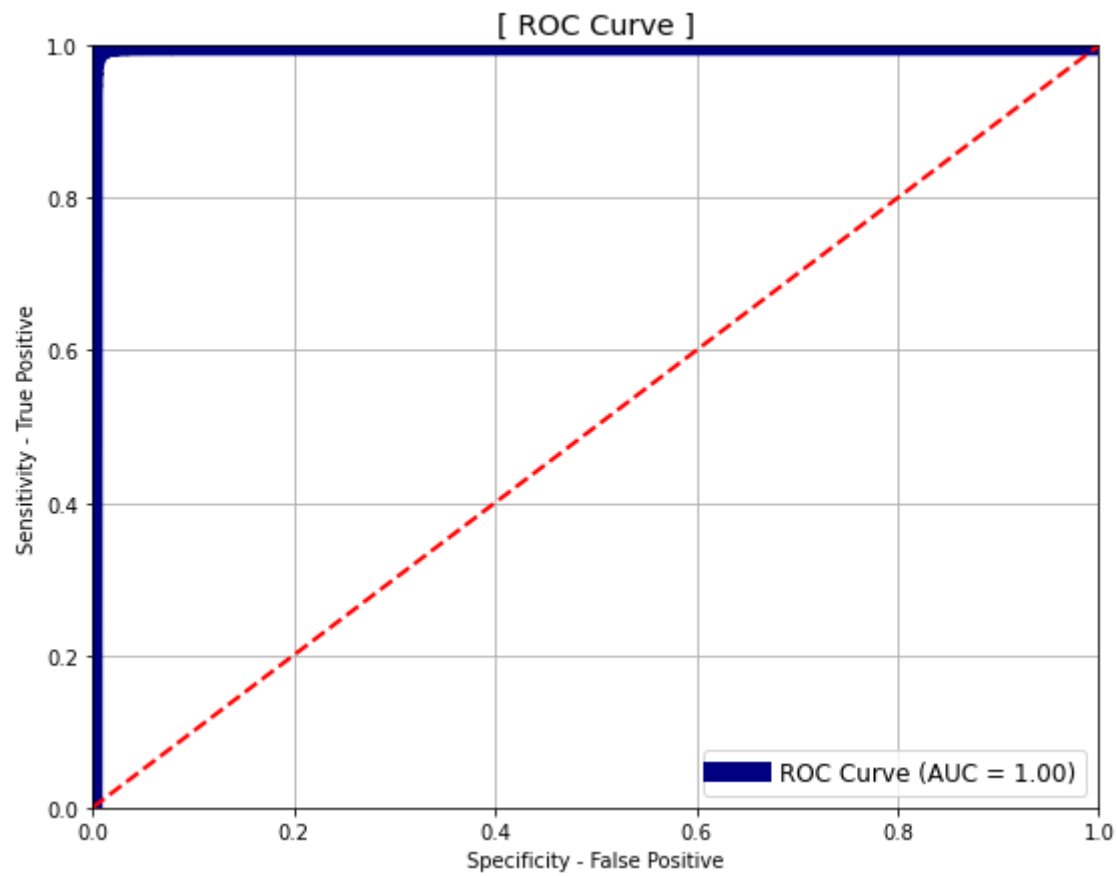
Silhouette Score = 0.8586774865237091  
evaluator: org.apache.spark.ml.evaluation.ClusteringEvaluator = cluEval\_03930801f4a4  
silhouette: Double = 0.8586774865237091

## 차량 이상 징후 판별

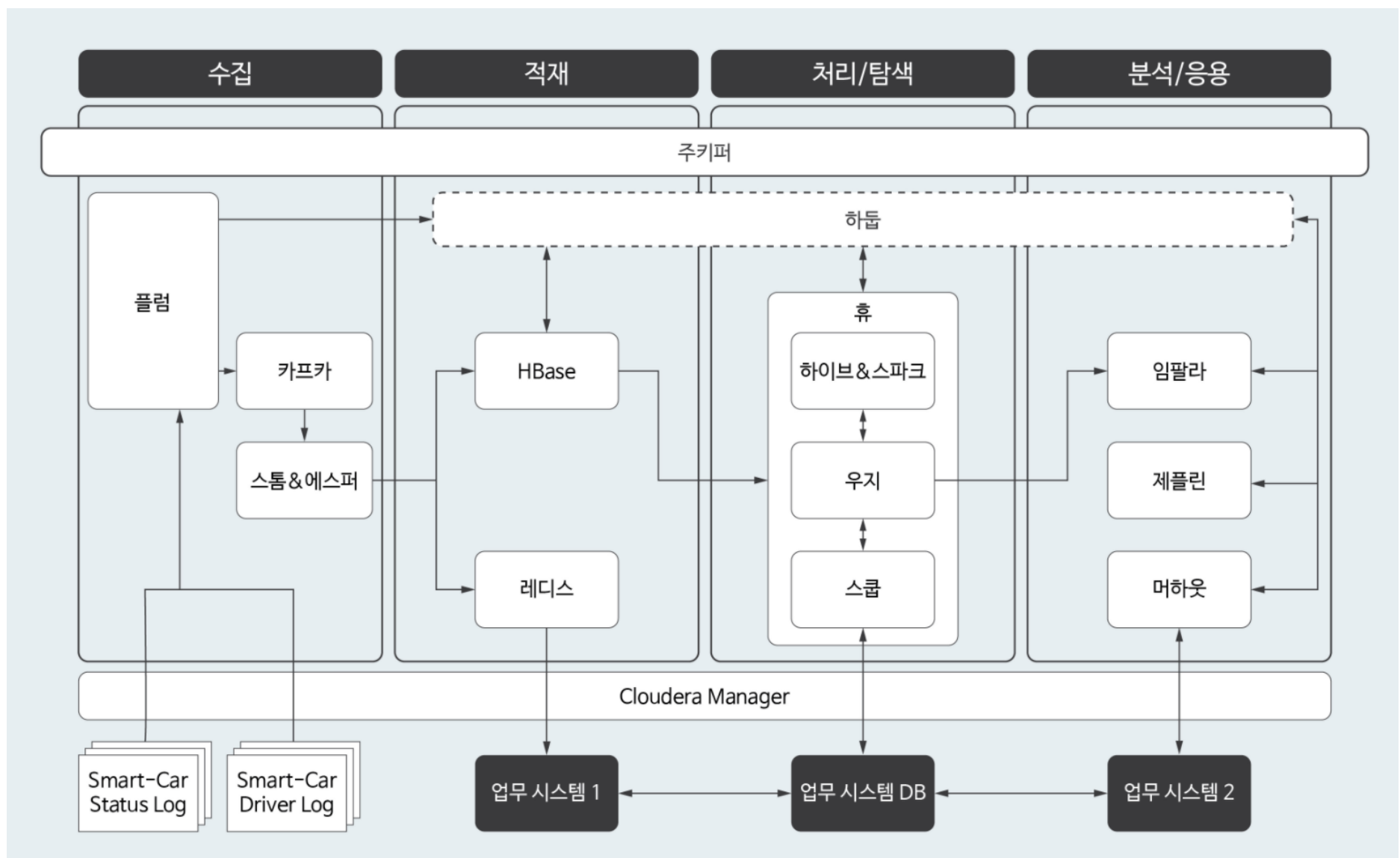
### ▼ 차량 이상 징후 판별 시 사용 기술

- Python, Impyla 이용 Hive 데이터 웨어하우스 연결
- Tensorflow, Keras, Scikit-learn, Pandas, Matplotlib 활용





## [ 최종 구성 ]



### 발생한 문제 및 해결

1. 앞선 수집/적재 프로세스에서 실시간 로그를 수집/적재 진행 중에 과속 데이터 발생 시 Redis에 적재되도록 했다.
  - 과속 시 실시간으로 적재가 되지 않는 오류가 발생했다. 시간이 흐른 후에 적재가 되는데 이속도가 매우 늦다.

### 원인



- Kafka의 Topic에서 실시간으로 쌓이는 데이터의 속도보다, Storm에서 데이터를 빼와 처리하고 Redis로 전송하는 속도가 느려서 발생한 문제


## 해결

- 아직 Kafka의 Topic에 이전에 처리 중인 데이터가 남아 있었다면, Storm에서 과속 여부를 판단해 Redis로 전송할 수 있기 때문에 **이전에 처리 중인 데이터가 해결되지 않아서 현재 실시간으로 데이터가 전송되지 않는다.**
- Kafka의 Topic에 해당 데이터가 남아 있는지 확인

```
$ kafka-console-consumer --bootstrap-server server02.hadoop.com:9092 --topic SmartCar-Topic --partition 0 --from-beginning
```

이전의 데이터가 너무 늦게 전송되어 Kafka Retention Time을 줄여서 이전에 전송 실패한 데이터들을 제거하고 다시 진행함으로써 해결하였다.

## 2. MapReduce가 진행되지 않는 문제(Accepted 단계에서 Running 단계로 넘어가지 않음)



### ACCEPTED Applications

Cluster

About  
Nodes  
Node Labels  
Applications  
NEW  
NEW SAVING  
SUBMITTED  
ACCEPTED  
RUNNING  
FINISHED  
FAILED  
KILLED  
Scheduler  
Tools

Cluster Metrics

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Memory Used | Memory Total | Memory Reserved | VCores Used | VCores Total |
|----------------|--------------|--------------|----------------|--------------------|-------------|--------------|-----------------|-------------|--------------|
| 2              | 1            | 0            | 1              | 0                  | 0 B         | 0 B          | 0 B             | 0           | 0            |

Cluster Nodes Metrics

| Active Nodes | Decommissioning Nodes | Decommissioned Nodes | Lost Nodes | Unhealthy Nodes | Rebooted Nodes |
|--------------|-----------------------|----------------------|------------|-----------------|----------------|
| 0            | 0                     | 0                    | 0          | 1               | 0              |

Scheduler Metrics

| Scheduler Type | Scheduling Resource Type      | Minimum Allocation      | Maximum Allocation      | Maximum Cluster Application Pr |
|----------------|-------------------------------|-------------------------|-------------------------|--------------------------------|
| Fifo Scheduler | [memory-mb (unit=Mi), vcores] | <memory:1024, vCores:1> | <memory:1536, vCores:1> | 0                              |

Show 20 entries

| ID                             | User  | Name   | Application Type | Queue   | Application Priority | StartTime                     | LaunchTime | FinishTime | State    | FinalStatus | Running Containers | Allocated CPU VCo | Allocated Memory MB | Reserved CPU VCo | Reserved Memory MB | % of Queue | % of Cluster | Progress | Track   |
|--------------------------------|-------|--|------------------|---------|----------------------|-------------------------------|------------|------------|----------|-------------|--------------------|-------------------|---------------------|------------------|--------------------|------------|--------------|----------|---------|
| application_1633530373141_0002 | admin | insert overwrite local directory '/home...T1 (Stage-1) | MAPREDUCE        | default | 0                    | Thu Oct 7 00:08:46 +0900 2021 | N/A        | N/A        | ACCEPTED | UNDEFINED   | 0                  | 0                 | 0                   | 0                | 0                  | 0.0        | 0.0          |          | Applica |

Showing 1 to 1 of 1 entries

## 원인

1. MapReduce가 많이 몰려있고 끝나지 않았을 때(Reducer 제한이 5개인데 5개 모두 동작 중인 경우)
2. 디스크 제한 임계치가 90%인데 이 임계값을 넘어서 Unhealthy Node로 설정되어 MapReduce가 실행되지 않았다.

## 해결

### 1번의 경우

- 실행 중인 것들을 종료해주고, 필요한 것부터 우선적으로 실행
- Reducer 개수 늘이기 ( 무작정 늘인다고 속도가 빨라지는 것이 아니므로 실행 쿼리/워크플로우 및 작업을 고려해서 설정할 필요가 있다

### 2번의 경우

- 디스크 제한 임계치를 변경하거나 ex) 80% → 90%( 90%이상은 추천되지 않는 방법 예상치 못한 문제 발생할 수 있다)
- 디스크 용량을 늘이는 방법을 선택할 수 있다.



### 중점적으로 고려한 사항

- 실제로 현업에서 처리되는 만큼의 많은 데이터나 고성능의 환경이 아니기 때문에 제약이 있으나, **DB나 Pandas**의 데이터 처리 방법을 쓰는 방식이 아닌 **데이터 웨어하우스**를 구성하는 이유가 잘 반영되도록 구성했다.
  - 실시간 수집/적재 시 Flume에서 Kafka를 거쳐 적재하는 구조를 이용(Fault Tolerance)
  - Oozie를 통한 예약 및 실행을 통한 Workflow 구성(실시간 수집/적재)
  - 분산 처리 및 In-memory 기술들을 잘 활용하여 효율적으로 구성



