빅데이터 프로젝트 실습하기

부산 더조은 직업훈련학교

훈련강사 김철학

차례

빅데이터 파일럿 프로젝트

빅데이터 파일럿 프로젝트

1.1 빅데이터 파일럿 프로젝트 도메인의 이해

1) 프로젝트 도메인 이해

본 파일럿 프로젝트에서 다루고자 하는 빅데이터 도메인은 자동차의 최첨단 전자장치와 무선통신을 결합한 스마트카 서비스이다.

향후 10년 안에 우리가 일상에서 타고 다니는 자동차 안에는 컴퓨터, 무선 인터넷, 전자장치들이 설치되어 자동차 안에서 이메일을 주고받고, 인터넷을 통해 각종 정보 검색과 무선 네트워크를 통해 차량의 원격 진단하고 운전습관, 날씨, 교통정보 등을 분석해서 운전자의 안전과 편의를 도모한다.

이렇게 똑똑한 스마트카 안에는 수백 개의 IoT 센서가 장착돼 있으며, 자동차의 상태를 모니터링하면서 수많은 차량 상태 정보를 실시간을 만들어 낸다.

스마트카의 센싱 정보는 무선 네트워크를 타고 중앙의 빅데이터 시스템으로 전송되며, 이러한 데이터들은 수집 → 적재 → 처리 → 분석 및 응용 단계를 거쳐 운전자의 안전과 편의성을 지원하는 스마트카 서비스로 제공된다.

따라서 본 파일럿 프로젝트는 스마트카에서 발생하는 수많은 데이터로부터 가치와 통찰력을 찾기 위한 빅데이터 시스템과 분석 실습을 진행하고자 한다.

2) 요구사항 파악

· **요구사항1 :** 차량의 다양한 장치로부터 발생하는 로그파일을 수집해서 기능별 상태를 점검한다.

분류	설명		
데이터 발생위치	100대의 시범 운행 차량		
데이터 발생 및 수집주기	3초 / 24시간		
데이터 수집규모	1MB/1대 (1일 수집규모: 약 100MB/100대)		
데이터 타입 및 처리유형	CSV 반정형 파일 / 배치		
데이터 스키마	발생일시	로그 발생일시	20200822104023(년월일시분초)
	차량번호	차량 고유번호	A12345
	타이어 FL	차량 앞왼쪽 타이어 상태	정상:80~100, 비정상:80이하
	타이어 FR	차량 앞오른쪽 타이어 상태	정상:80~100, 비정상:80이하
	타이어 BL	차량 뒤왼쪽 타이어 상태	정상:80~100, 비정상:80이하
	타이어 BR	차량 뒤오른쪽 타이어 상태	정상:80~100, 비정상:80이하
	라이트 FL	차량 앞왼쪽 라이트 상태	1:정상, 2:비정상
	라이트 FR	차량 앞오른쪽 라이트 상태	1:정상, 2:비정상
	라이트 BL	차량 뒤왼쪽 라이트 상태	1:정상, 2:비정상
	라이트 BR	차량 뒤오른쪽 라이트 상태	1:정상, 2:비정상
	엔진	차량 엔진 상태	A:정상, B:점검 필요, C:고장
	브레이크	차량 브레이크 상태	A:정상, B:점검 필요, C:고장
	배터리	차량 배터리 충전 상태	1 ~ 100%
	작업요청일	수집 작업 요청일	20200823(년월일)

· **요구사항2 :** 운전자의 운행 정보가 담긴 로그를 실시간으로 수집해서 주행 패턴을 분석한다.

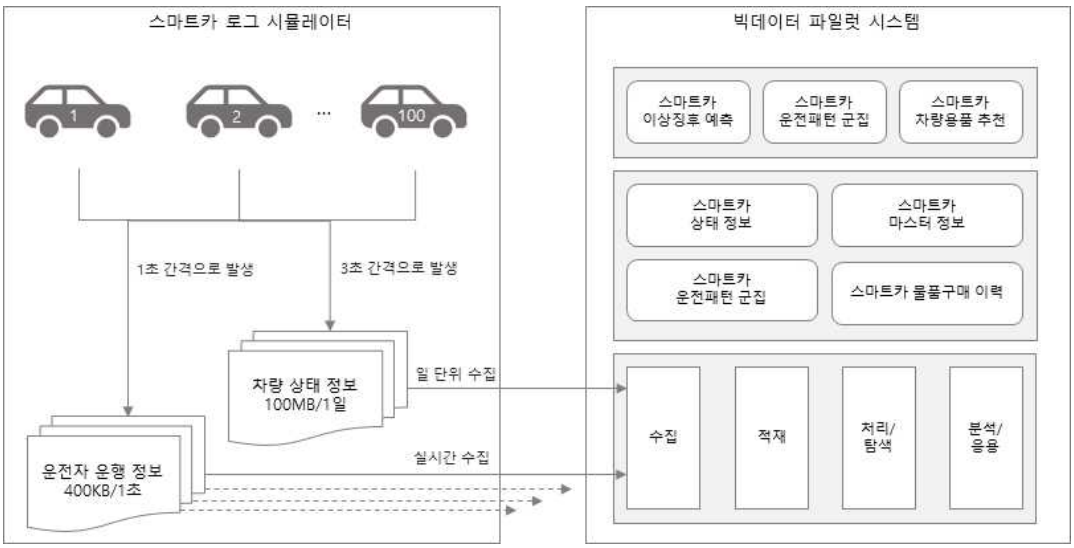
분류	설명		
데이터 발생위치	100대의 시범 운행 차량		
데이터 발생 및 수집주기	1초 / 실시간		
데이터 수집규모	4KB/1대 (초당 수집규모 : 약 400KB/100대)		
데이터 타입 및 처리유형	CSV 반정형 파일 / 실시간		
데이터 스키마	발생일시	로그 발생일시	20200822104023(년월일시분초)
	차량번호	차량 고유번호	A12345
	가속 페달	가속 페달 이벤트	0 ~ 5 단계
	브레이크 페달	브레이크 페달 이벤트	0 ~ 3 단계
	운전대 회전각	운전대 회전 이벤트	F:직진
			L1:좌회전각 1~10
			L2:좌회전각 11~20
			L3:좌회전각 21~30
			R1:좌회전각 1~10
			R2:좌회전각 11~20
			R3:좌회전각 21~30
	방향지시등	방향지시등 이벤트	L:왼쪽, R:오른쪽, N:없음
	주행속도	차량 주행속도	0 ~ 250km/h
	주행지역	운전 중인 구역번호	A, B, C, D, E, F 구역 1~10
	작업요청일	수집 작업 요청일	20200823(년월일)

3) 데이터셋 파악

스마트카에서 발생하는 기본 데이터셋은 총 4가지 유형을 사용한다.

데이터셋	설명
스마트카 상태 정보 데이터	요구사항1의 차량 상태 정보 데이터셋, 로그 시뮬레이터로 생성
스마트카 운전자 운행 데이터	요구사항2의 운전자 운행 정보 데이터셋, 로그 시뮬레이터로 생성
스마트카 마스터 데이터	스마트카 운전자의 프로파일 정보가 담긴 데이터셋, 샘플파일 제공
스마트카 물품 구매 이력 데이터	스마트카 운전자가 구매한 물품 구매 데이터셋, 샘플파일 제공

· 파일럿 프로젝트 데이터셋

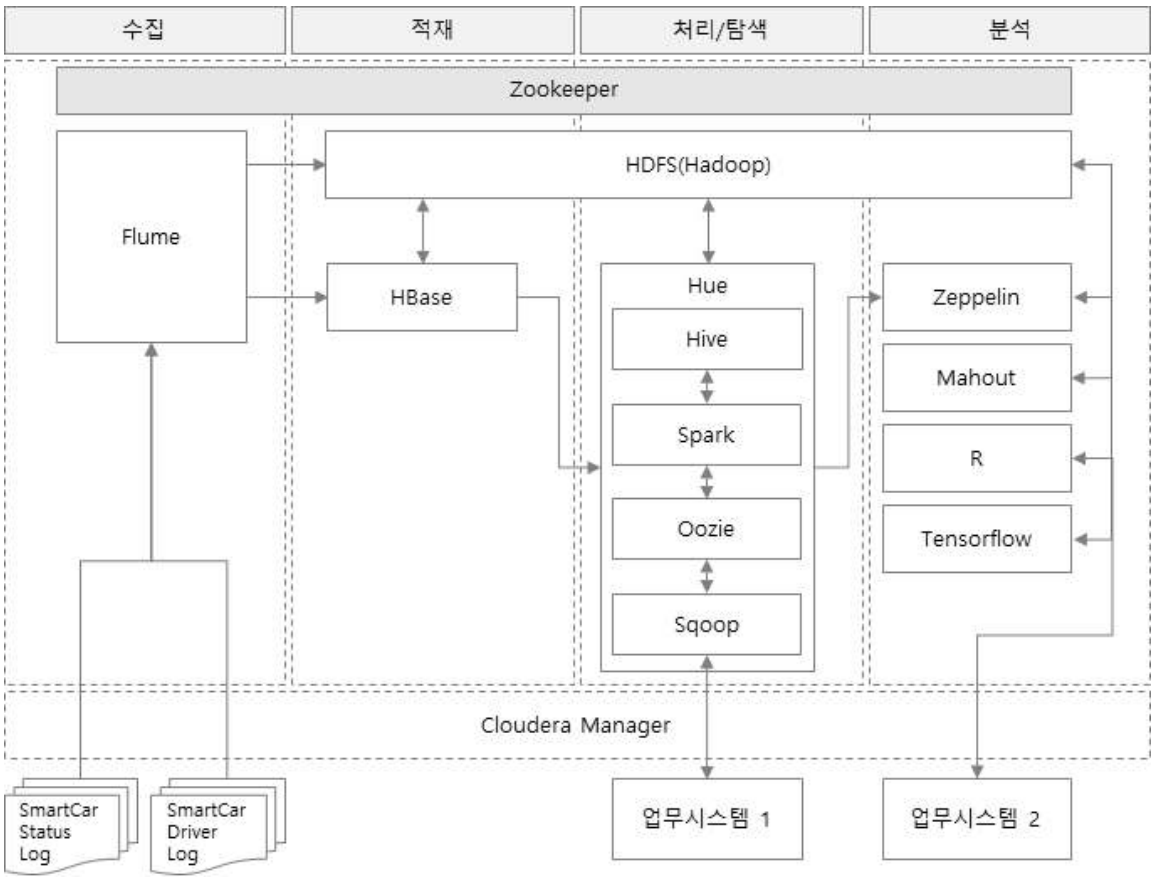


1.2 빅데이터 파일럿 프로젝트 아키텍처 이해

1) 소프트웨어 아키텍처

파일럿 프로젝트의 소프트웨어 아키텍처는 메인 오픈소스 소프트웨어만 17종이고 분석 소프트웨어인 R, 텐서플로, 케라스 까지 포함하면 총 20종이 된다. 각 오픈소스 프로젝트는 크게 수집, 적재, 처리/탐색, 분석 및 응용으로 분류되며 해당 프로젝트는 4개의 레이어로 분리된다.

· 파일럿 프로젝트 소프트웨어 구성도



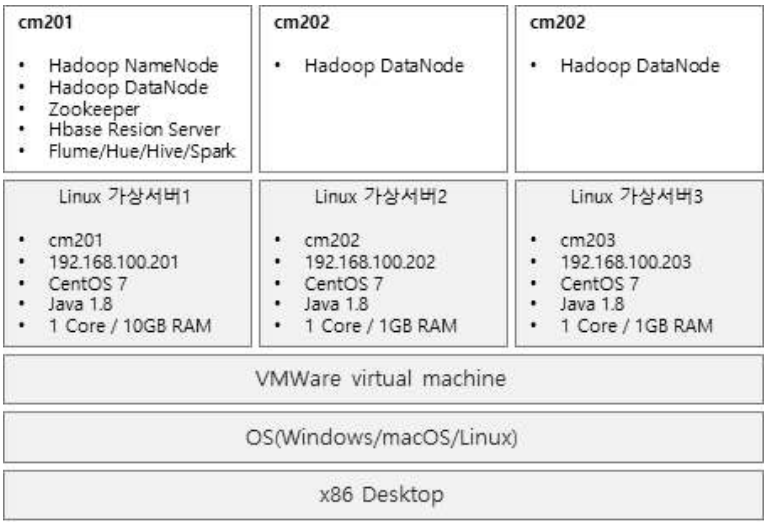
2) 하드웨어 아키텍처

빅데이터의 하드웨어 아키텍처는 3V 관점에서 구성한다. 서버, 네트워크, 디스크, 랙 등을 3V 관점에서 면밀히 검토해서 설계하고 규모를 산정한다.

· 파일럿 프로젝트의 하드웨어 환경

환경	CPU	RAM	HDD	비고
저사양 PC	듀얼코어 이상	8GB 이상	100GB 이상	- SSD 권장 - RAM 최소 32GB 권장
고사양 PC	i5 이상	16GB 이상	120GB 이상	

· 파일럿 프로젝트 하드웨어 구성

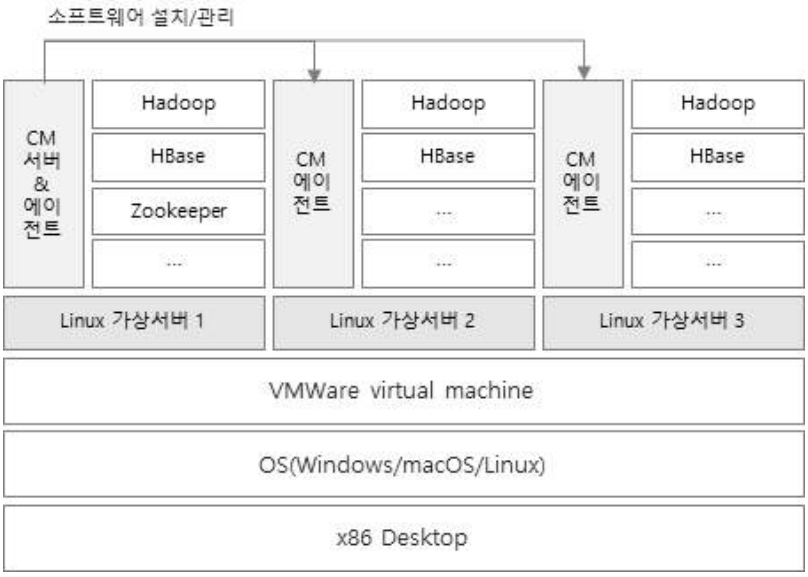


3) 구축 환경의 이해

앞서 설명한 소프트웨어/하드웨어 아키텍처를 만들기 위해 3대의 가상머신을 만들고, 가상머신에 총 17개의 소프트웨어를 설치해야한다.

빅데이터의 모든 기술 요소를 다 갖춘 환경으로서 17종의 소프트웨어를 수작업으로 설치 및 구성하기에는 웬만한 전문가들도 하기 어려운 작업이다. 그래서 파일럿 프로젝트에서는 빅데이터 자동화 솔루션인 클라우데라의 CM(Cloudera Manager)을 이용한다. 그리고 이 CM을 통해서 하둡을 포함한 에코시스템 17개를 편리하게 설치 및 관리한다.

· 파일럿 프로젝트 구축 환경



1.3 빅데이터 파일럿 프로젝트 개발 환경구성

1) 가성 머신 환경구성

빅데이터 파일럿 서버를 구성하기 위해 VMWare를 설치하고, 이를 통해 3대의 가성머신을 생성한다. 이 3대의 가상머신에 CentOS 리눅스를 설치한다. 가상머신의 CPU와 메모리 리소스 설정을 파일럿 프로젝트에 맞게 설정한다. 메모리의 경우 아래 표를 참고해서 본인의 PC 환경에 맞춰 재설정한다.

실습1-1 가상 머신 생성하기

ㄴ Step01. VMWare 프로그램으로 'server01', 'server02', 'server03' 이름으로 총 3대의 가상 머신 생성

ㄴ Step02. 3대의 가상머신에 HDD 용량 모두 40GB 설정 아래 표를 참고하여 메모리를 할당

· 가상머신 메모리 할당

설치 PC환경	server01	server02	server03
저사양 PC CPU:듀얼코어, RAM:8GB	4096MB(4GB)	2048MB(2GB)	N/A
고사양 PC CPU:i5 이상, RAM:16GB	10240MB(10GB)	1024MB(1GB)	1024MB(1GB)

2) CentOS 설치 및 설정

각 가상머신에 CentOS 리눅스를 설치한다. 설치 후 기본 설정 및 필수 패키지를 설치한다.

실습2-1 CentOS 설치하기

ㄴ Step01. CentOS 7버전, 최소설치, 기본 파티션, Kdump 해제

ㄴ Step02. 최고관리자 root 비밀번호 설정, 'bigdata' 관리자 계정추가

ㄴ Step03. 전원 종료 후 각 가상머신 Settings - Network Adapter - Custom(VMnet8) 로 변경

실습2-2 CentOS 설정하기

ㄴ Step01. 가상머신 고정 IP 할당(모든 시스템 동일), vmnetcfg 설정은 매뉴얼 참고

```
#vi /etc/sysconfig/network-scripts/ifcfg-ens33
 수정 : BOOTPROTO="static"
 추가 : IPADDR="192.168.56.101"
 추가 : NETMASK="255.255.255.0“
 추가 : GATEWAY="192.168.100.2"
```

ㄴ Step02. 네트워크 재시작, dhclient 실행, 핑 테스트

```
#systemctl restart network
#dhclient
#ping google.com
(핑테스트 종료 Ctrl+c)
```

ㄴ Step03. 기본 업데이트 진행, 필수 패키지 설치(모든 시스템 동일)

```
#yum -y update
#yum install -y vim wget net-tools java-1.8*
```

ㄴ ftp 설치 및 설정(CentOS 6 이상에서는 sftp 기본설치가 되어있음)

ㄴ ssh 설치 및 설정(CentOS 6 이상에서는 openSSH 기본설치가 되어있음)

ㄴ Step04. SELINUX 해제(모든 시스템 동일)

```
#vi /etc/selinux/config
7라인 : SELINUX=disabled
```

ㄴ Step05. NetworkManager 끄기(모든 시스템 동일)

```
#systemctl status NetworkManager
#systemctl stop NetworkManager
#systemctl disable NetworkManager
```

ㄴ Step06. 방화벽 해제(모든 시스템 동일)

```
#systemctl status firewalld
#systemctl stop firewalld
#systemctl disable firewalld
```

ㄴ Step07. host 설정(모든 시스템 동일)

```
#vi /etc/hostname
// 아래 3개 이름 중 하나만 입력
server01   // 만약 192.168.xxx.201
server02   // 만약 192.168.xxx.202
server03   // 만약 192.168.xxx.203
```

ㄴ Step08. hosts 설정(모든 시스템 동일)

```
#vi /etc/hosts
// 기존 내용 모두 삭제후 아래 내용 그대로 추가
192.168.xxx.201 server01
192.168.xxx.202 server02
192.168.xxx.203 server03
```

ㄴ Step09. 재부팅 후 putty 접속

```
#reboot
```

3) Cloudera Manager 설치

CM(Cloudera Manager)은 빅데이터 에코시스템을 쉽게 설치하고 관리해주는 클라우데라의 강력한 빅데이터 시스템 솔루션이다. 파일럿 프로젝트를 진행하면서 필요한 소프트웨어들을 CM을 통해 설치 및 설정 한다.

실습3-1 CM 다운로드 하기(server01 서버만 적용)

ㄴ Step01. https://www.cloudera.com/ 접속 메뉴 - Products - Downloads 클릭

ㄴ Step02. 'GET IT NOW' 클릭, 폼 작성(이름, 국가, 폰 번호 임의 입력 가능) 후 Continue 클릭

ㄴ Step03. 약관체크 및 Submit 클릭

ㄴ Step04. Production Installation 항목에서 installation instructions 링크 클릭

ㄴ Step05. Step1: Configure a Repository 링크 클릭

ㄴ Step06. Cloudera Manager 6 Version and Download Information 링크 클릭

ㄴ Step07. Cloudera Manager 6.3.1표에서 cloudera-manager.repo 마우스 오른쪽 버튼 링크주소 복사

· 2020.08월 기준 Cloudera Manager 6.3.3버전은 정식 계정이 있어야 다운로드 됨

ㄴ Step08. 링크 복사한 Cloudera-manager 저장소파일 server01서버에 다운로드 및 yum 저장소로 이동

```
#wget https://archive.cloudera.com/cm6/6.3.1/redhat7/yum/cloudera-manager.repo
#mv ./cloudera-manager.repo /etc/yum.repos.d/
```

실습3-2 CM 설치하기(server01 서버만 적용)

ㄴ Step01. CM(Cloudera Manager) 설치, 파일크기 1.1GB

```
#yum install cloudera-manager-daemons cloudera-manager-server
```

ㄴ Step02. CM PostgreSQL DB 설치 및 실행

```
#yum install cloudera-manager-server-db-2
#systemctl start cloudera-scm-server-db
```

ㄴ Step03. CM PostgreSQL DB 설정

```
#vi /var/lib/cloudera-scm-server-db/data/pg_hba.conf
마지막 밑에서 3번째 줄 'reject' → 'md5' 수정
맨 마지막 줄에 새로 추가 → host all all 0.0.0.0/0 trust
```

ㄴ Step04. CM PostgreSQL DB 변경사항 적용을 위해 재기동

```
#systemctl restart cloudera-scm-server-db
```

ㄴ Step05. CM 실행

```
#systemctl start cloudera-scm-server
#systemctl status cloudera-scm-server
```

ㄴ Step06. CM 브라우저 접속확인(CM서버가 기동하는데 2~3분 걸림)

· 주소 : http://192.168.xxx.201:7180

· 계정 : admin

· 암호 : admin

실습3-3 CM 설정하기

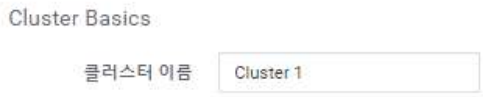
ㄴ Step01. CM 로그인하기
- 주소 : http://192.168.xxx.201:7180
- 계정 : admin
- 암호 : admin

ㄴ Step02. CM 동의체크 후 [계속] 클릭

ㄴ Step03. Cloudera Express 선택 후 [계속] 클릭

ㄴ Step04. CM 클러스터 설치 시작 [계속] 클릭

ㄴ Step05. Cluster Basics
- 클러스터 이름 'Cluster 1' 기본값 사용, [계속] 클릭



ㄴ Step06. Specify Host
- 호스트 이름 입력 후 검색

ㄴ Step07. 리포지토리 선택
- 변경사항 없이 기본값으로 [계속] 클릭

ㄴ Step08. JDK 설치옵션
- openJDK를 사용하기 때문에 체크 안함 [계속] 클릭

ㄴ Step09. SSH 로그인 정보 제공 설정
- 리눅스 root 비밀번호 입력 후 [계속] 클릭

ㄴ Step10. Install Agents
- Cloudera Manager 관련 패키지 설치 진행, 설치 완료 후 [계속] 클릭
- 최소 20분 소요, 네트워크 상태에 따라 최대 40분 소요

ㄴ Step11. Install Parcels

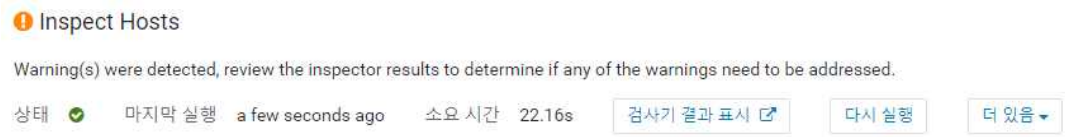
· Cloudera Manager 추가 패키지 설치 진행, 설치 완료 후 [계속] 클릭

· 최소 20분 소요, 네트워크 상태에 따라 최대 40분 소요

ㄴ Step12. Install Cluster

· [Inspect Network Performance] 버튼 클릭 점검

· [Inspect Hosts] 버튼 클릭 점검



· Inspect Hosts Warning 발생, 검사기 결과표시 클릭해서 내용 확인

· 시스템 Swap 메모리 설정(모든 시스템 적용)

```
#sysctl -w vm.swappiness=10

#vi /etc/sysctl.conf
마지막 줄에
vm.swappiness=10 ← 추가
```

· 시스템 kernal hugePage 설정(모든 시스템 적용)

```
#echo never > /sys/kernel/mm/transparent_hugepage/defrag
#echo never > /sys/kernel/mm/transparent_hugepage/enabled

#vi /etc/rc.local
마지막 줄에 아래 2줄 추가
echo never > /sys/kernel/mm/transparent_hugepage/defrag
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

· Inspect Hosts 점검 다시 실행, 정상 확인 후 [계속] 클릭

ㄴ Step13. 클러스터 서비스 선택

· 사용자 지정 서비스 선택

· HDFS, YARN(MR2 Included), Zookeeper 3개 선택

· 3개 서비스 선택 후 [계속] 클릭

ㄴ Step14. 역할 할당 사용자 지정

[HDFS]

· NameNode : server01 선택
· SecondaryNameNode : server01 선택
· Balancer : server01 선택
· HttpFS : 미설치
· NFS Gateway : 미설치
· DataNode : 모든 호스트

[Cloudera Management Service]

· Service Monitor : server01 선택
· Activity Monitor : 미설치
· Host Monitor : server01 선택
· Event Monitor : server01 선택
· Alert Publisher : server01 선택

[YARN (MR2 Included)]

· ResourceManager : server01 선택
· JobHistory Server : server01 선택
· NodeManager : 'DataNode(으)로 저장' 선택

[ZooKeeper]

· Server : server01 선택

ㄴ Step15. 데이터베이스 설정

· 변경사항 없음, [테스트 연결] 클릭 후 [계속] 클릭

ㄴ Step16. 변경 내용 검토

· 변경사항 없음, [계속] 클릭

ㄴ Step17. 첫 번째 실행 명령

· 설치가 완료된 후 [계속] 클릭

ㄴ Step18. 요약 [완료] 클릭

4) 스마트카 로그 시뮬레이터

실제 100대의 차량으로 실습하는 것은 불가능하므로 스마트카를 시뮬레이션하는 스마트카 로그 시뮬레이터 프로그램을 사용한다.

로그 시뮬레이터로 생성된 스마트카 데이터는 수집 → 적재 → 처리 → 분석 및 응용 단계를 거치고, 각 단계마다 파일럿에서 활용하기 쉬운 데이터셋으로 재구성한다.

실습4-1 스마트카 운전자 운행 정보 로그 시뮬레이터 실행하기

ㄴ Step01. server01서버 파일럿 프로젝트 작업 디렉터리 생성

```
#mkdir /home/pilot
#mkdir /home/pilot/working
#mkdir /home/pilot/working/car-batch-log
#mkdir /home/pilot/working/driver-realtime-log
#chmod 777 -R /home/pilot
```

ㄴ Step02. 로그 시뮬레이터 프로그램 server01서버로 업로드

· 파일질라 FTP클라이언트 실행

· server01에 SFTP 접속

· 제공된 bigdata.smartcar.loggen-1.0.jar 파일을 /home/pilot/working에 업로드

ㄴ Step03. 스마트카 운행 정보 로그 시뮬레이터 실행 및 데이터 확인

```
#cd /home/pilot/working
#java -cp bigdata.smartcar.log.jar com.bigdata.smartcar.log.DriverLogMain 20200822 10 &
#cd /home/pilot/working/driver-realtime-log
#tail -f SmartCarDriverInfo.log
```

ㄴ Step04. 로그 시뮬레이터 종료

```
#ps -ef | grep smartcar.log
#kill -9 [pid]
```

실습4-2 스마트카 상태 정보 로그 시뮬레이터 실행하기

ㄴ Step01. 스마트카 상태 정보 로그 시뮬레이터 실행 및 데이터 확인

```
#cd /home/pilot/working
#java -cp bigdata.smartcar.log.jar com.bigdata.smartcar.log.CarLogMain 20200822 10 &
#cd /home/pilot/working/SmartCar
#tail -f SmartCarStatusInfo_20200822.txt
```

ㄴ Step02. 로그 시뮬레이터 종료

```
#ps -ef | grep smartcar.log
#kill -9 [pid]
```

5) CM서버 개발 환경관리

파일럿 프로젝트 환경은 3대의 리눅스 가상머신에서 다수의 빅데이터 소프트웨어를 구성하기 때문에 개발 환경이 무겁고 복잡하다. 파일럿 프로젝트와 관련된 작업을 시작하거나 종료할 때는 다음과 같은 절차로 안정적인 빅데이터 파일럿 환경을 유지할 수 있다.

실습5-1 CM 개발 환경 시작하기

ㄴ Step01. 가상머신을 실행해서 server01, server02, server03 순서대로 시작

ㄴ Step02. Putty로 server01에 SSH 접속 후 CM 구동 여부 확인

```
#service cloudera-scm-server status
```

ㄴ Step03. 크롬 브라우저를 실행 CM 관리자 접속

· 주소 : http://192.168.xxx.201:7180

· 계정 : admin

· 암호 : admin

ㄴ Step04. CM 홈 왼쪽 메뉴에서 [Cluster1] 콤보박스를 선택하고 [재시작] 선택

ㄴ Step05. CM 홈 왼쪽 메뉴에서 [Cloudera Management Service] 콤보박스를 선택하고 [재시작] 선택

ㄴ Step06. Cluster1과 Cloudera Management Service의 모든 메뉴들의 상태 활성화 확인

실습5-2 CM 개발 환경 종료하기

ㄴ Step01. CM 홈 왼쪽 메뉴에서 [Cluster1] 콤보박스를 선택하고 [중지] 선택

ㄴ Step02. CM 홈 왼쪽 메뉴에서 [Cloudera Management Service] 콤보박스를 선택하고 [중지] 선택

ㄴ Step03. 모든 가상머신 종료

```
#poweroff
```

1.4 빅데이터 파일럿 프로젝트 수집

1) 수집 아키텍처

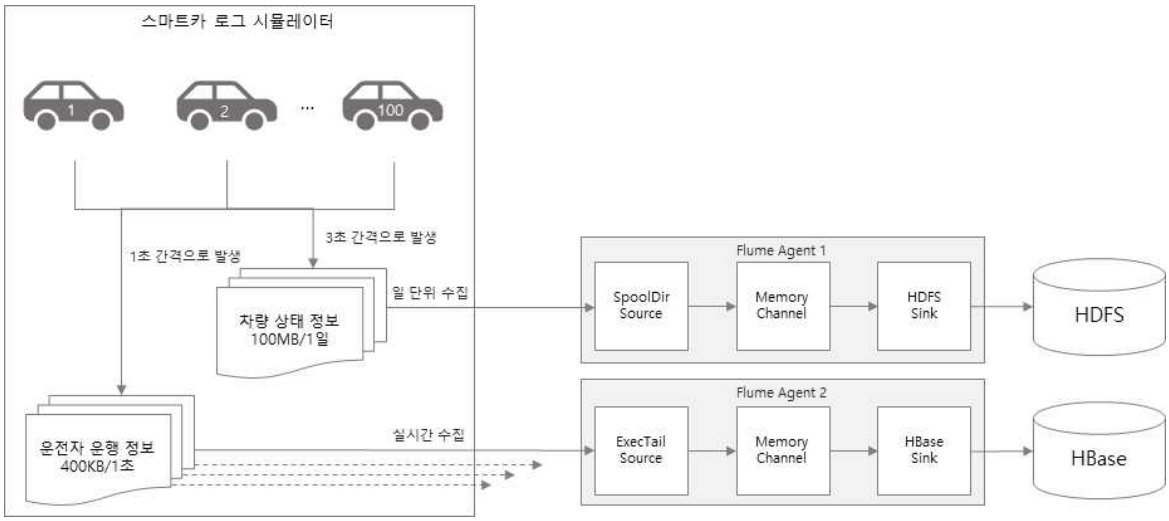
1파일럿 프로젝트 수집에서는 수집 관점에서 요구사항들을 더 구체화하고, 이를 해결하기 위한 솔루션과 기술 요소들을 도출한다.

- **요구사항1 :** 차량의 다양한 장치로부터 발생하는 로그파일을 수집해서 기능별 상태를 점검한다.
- **요구사항2 :** 운전자의 운행 정보가 담긴 로그를 실시간으로 수집해서 주행 패턴을 분석한다.

· 요구사항 구체화 및 분석

수집 요구사항 구체화	분석 및 해결방안
스마트카로부터 로그파일들이 주기적으로 발생한다.	Flume을 이용해 로그 파일 수집
스마트카의 배치 로그파일 이벤트를 감지해야 한다.	Flume의 SpoolDir을 이용해 주기적인 로그 발생 감지
스마트카의 실시간 로그 발생 이벤트를 감지해야 한다.	Flume의 Exec-Tail을 이용해 주기적인 로그 발생 감지

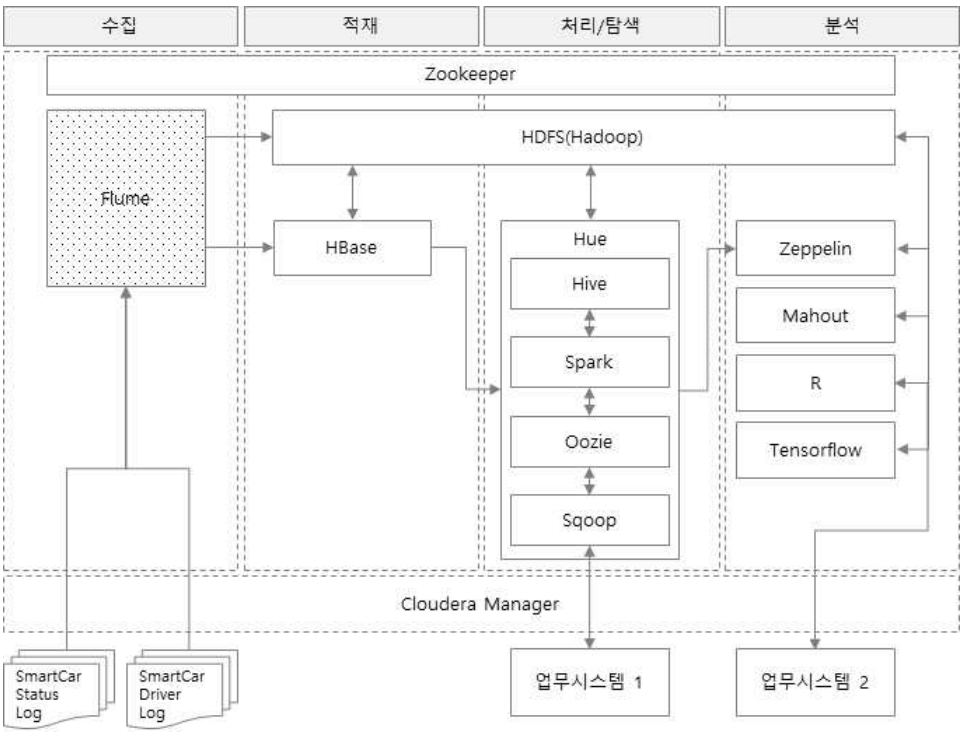
· 파일럿 프로젝트 로그 수집 아키텍처



2) 수집 환경구성

CM을 이용해 플럼을 server01 서버에 설치 후 설정한다.

· 파일럿 프로젝트 수집 환경



실습2-1 Flume 설치/설정하기

ㄴ Step01. CM 홈 왼쪽 메뉴에서 [Cluster1] 콤보박스를 선택하고 [서비스 추가] 선택

ㄴ Step02. 추가할 서비스 유형 중 [Flume]을 선택, 우측 하단 [계속] 클릭

ㄴ Step03. Flume을 설치할 서버 호스트 server01 선택, [확인] → [계속] 클릭

ㄴ Step04. 설치 완료 메시지 확인, [완료] 클릭

ㄴ Step05. CM 홈 → [Flume] → [구성]

· "java heap" 검색 후 '100'으로 힙 메모리 설정

ㄴ Step06. CM 홈 → [Flume] → [구성], 여러 항목 중 Agent 이름, 구성 파일 수정

· Agent 이름 : SmartCar_Agent

· 구성 파일 : 기존 내용 모두 제거 후 제공되는 파일 내용을 복사 입력

ㄴ Step07. CM 홈 → [Flume] → [재시작]

3) 수집 테스트

지금까지 구성한 빅데이터 수집 기능이 정상적으로 작동하는지 간단한 테스트를 통해 점검한다.
스마트카 로그 시뮬레이터를 작동시켜 2020/08/22에 10대의 스마트카 로그를 발생시킨다.

실습3-1 수집 기능 점검하기

ㄴ Step01. 스마트카 로그 시뮬레이터 작동

```
#cd /home/pilot/working
#java -cp bigdata.smartcar.log.jar com.bigdata.smartcar.log.CarLogMain 20200822 10 &
#java -cp bigdata.smartcar.log.jar com.bigdata.smartcar.log.DriverLogMain 20200822 10 &
```

ㄴ Step02. 테스트 로그파일 확인

```
#cd /home/pilot/working/SmartCar
#tail -f SmartCarStatusInfo_20200822.txt

#cd /home/pilot/working/driver-realtime-log
#tail -f SmartCarDriverInfo.log
```

ㄴ Step03. Flume Agent 실행

· CM 홈 → [Flume] → [재시작]

ㄴ Step04. Flume의 표준 출력 로그로 전송여부 확인

```
#tail -f /var/log/flume-ng/flume-cmf-flume-AGENT-server01.log
```

ㄴ Step05. 로그 시뮬레이터 종료

```
#ps -ef | grep smartcar.log
#kill -9 [pid]
```

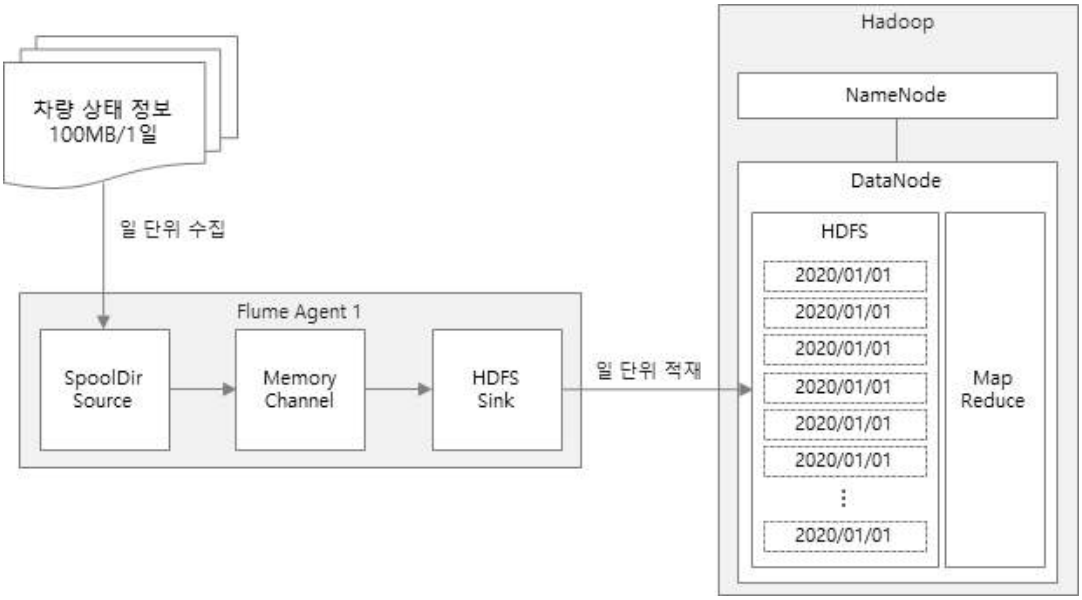
1.5 빅데이터 파일럿 프로젝트 적재

1) 적재 아키텍처

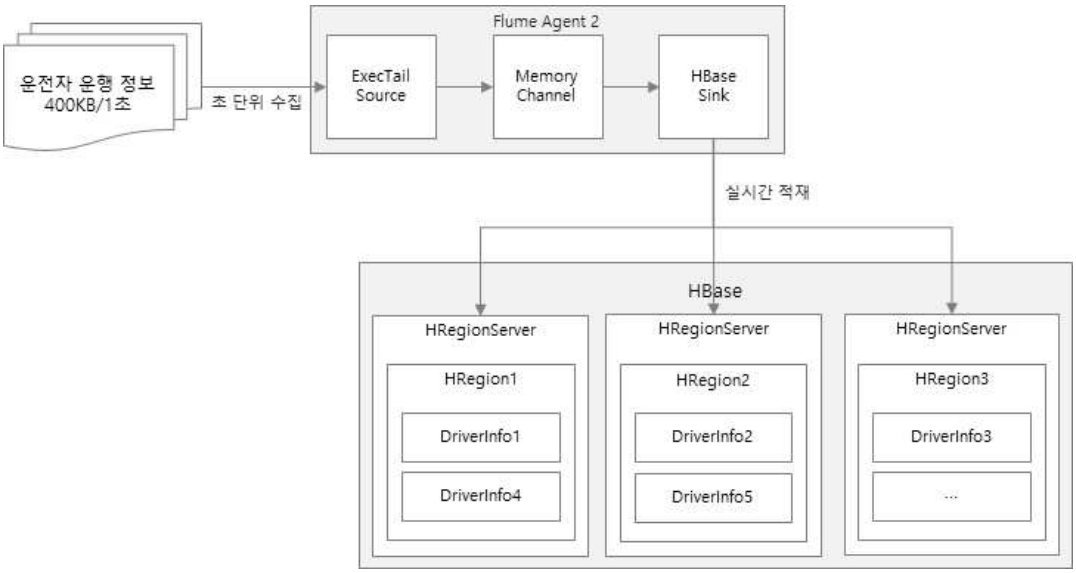
파일럿 프로젝트의 대용량 로그 파일을 적재하기 위한 요건으로 아키텍처는 아래 그림과 같이 구성한다. 플럼의 Source 컴포넌트로 대용량 파일을 읽어 들이고 Sink를 이용해 HDFS의 특정 경로에 적재하는 구성이다. HDFS에 적재할 때는 데이터의 포맷, 경로, 파티션 값을 신중하게 설정한다.

또한 HDFS에 적재된 데이터는 부분 수정/삭제가 어렵기 때문에 유형에 따라 특별한 관리 정책이 필요하다. 이러한 데이터 관리 정책에 통해 초기 적재 레이어에는 원천을 그대로 유지하며 데이터레이크라 불리는 영역을 만들게 되고, 이후 데이터 가공 작업으로 데이터의 품질을 높이며 빅데이터 웨어하우스와 데이터 마트를 구성한다.

· 파일럿 프로젝트 차량 상태 정보 적재 아키텍처



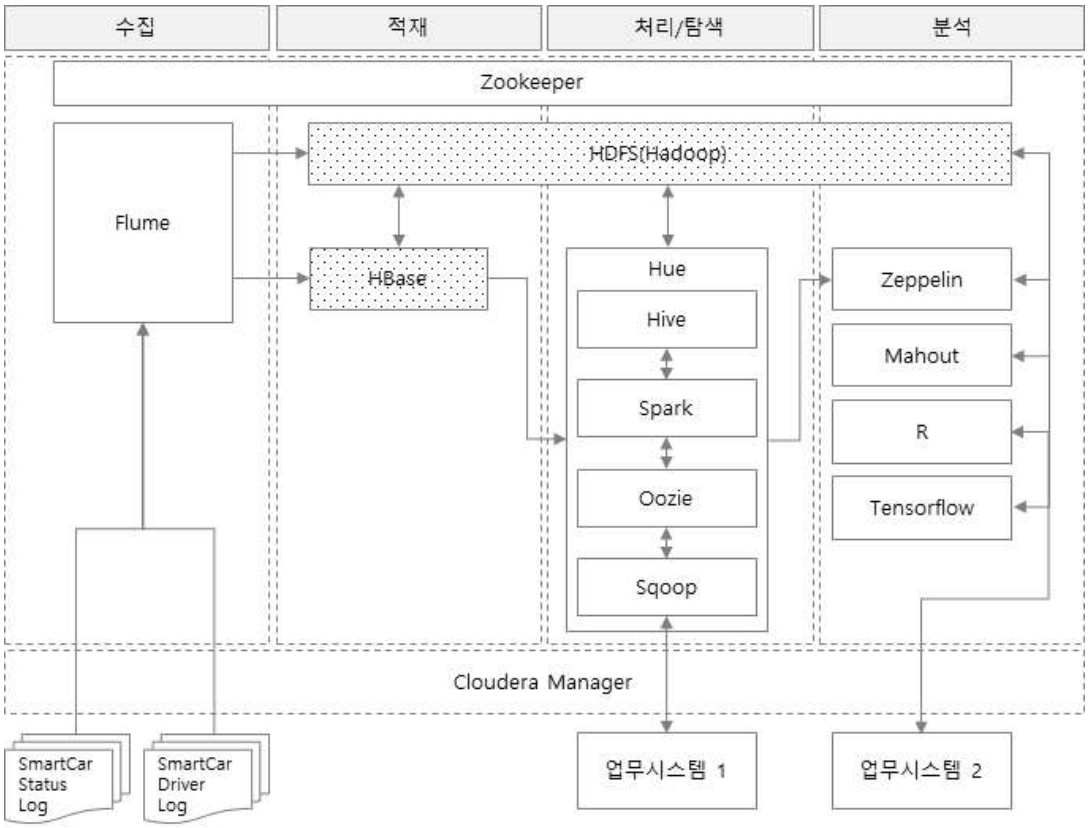
· 파일럿 프로젝트 운전자 운행 정보 적재 아키텍처



2) 적재 환경구성

파일럿 프로젝트에서 스마트카의 대용량 로그파일(스마트카 상태 정보)을 적재할 때와 실시간 로그파일(스마트카 운전자 정보)을 적재할 때의 환경이 다르다. 대용량 로그파일은 HDFS에 장기간 저장해 놓고 대규모 배치 작업과 분석에 이용되기 때문에 Hadoop을 설치한다. 실시간 로그파일은 데이터 크기는 작고 빠른 속도로 대량으로 생성되기 때문에 데이터를 영구 저장하기 위해서 Hadoop의 HDFS를 직접 이용하지 않는다. 그 이유는 유입된 작은 메시지 한 건을 곧바로 하둡에 저장할 경우 한 개의 HDFS 파일이 생성되는데, 초당 수천 건의 트랜잭션이 발생하는 메시지의 경우 파일 개수가 기하급수적으로 늘어나고 이로 인해 하둡 클러스터에 지나친 오버헤드가 발생한다. 이러한 문제를 해결하기 위해 대규모 트랜잭션 데이터를 처리하는 데 최적화된 NoSQL 데이터베이스인 Hbase를 이용한다. 마지막으로 하둡의 다양한 에코시스템의 기능들을 웹 UI로 통합 제공하여 기술의 복잡도는 숨기고 접근성과 편의성을 높이는 Hue를 설치한다.

· 파일럿 프로젝트 적재 환경



실습2-1 Hadoop 설치/설정하기

ㄴ Step01. Hadoop(HDFS) 설치는 CM 설치할 때 기본 설치 됨

ㄴ Step02. CM 홈 → [HDFS] → [구성], "복제 계수" 검색 후

· 복제 계수 변경 : 3 → 1

ㄴ Step03. CM 홈 → [HDFS] → [구성], "HDFS 권한 검사" 검색 후

· HDFS 권한 검사 : 해제

ㄴ Step04. CM 홈 → [HDFS] → [구성], "HDFS 블록 크기" 검색 후

· HDFS 블록 크기 변경 : 128 → 64

ㄴ Step05. CM 홈 → [YARN (MR2 Included)] → [구성], "Scheduler 클래스" 검색 후
· Scheduler 클래스 변경 : FairScheduler → FifoScheduler

ㄴ Step06. CM 홈 → [Cluster 1] → [재시작]
ㄴ Step07. CM 홈 → [HDFS] → [재배포 아이콘] → [클라이언트 구성 배포] 클릭
ㄴ Step08. CM 홈 → [YARN (MR2 Included)] → [재배포 아이콘] → [클라이언트 구성 배포] 클릭
ㄴ Step10. Hadoop 기본 명령어 실습

```
#cd /home/pilot/working/SmartCar
#hdfs dfs -ls /
#hdfs dfs -mkdir /sample
#hdfs dfs -put SmartCarStatusInfo_xxxxxxxx.txt /sample
```

실습2-2 HBase 설치/설정하기
ㄴ Step01. CM 홈 왼쪽 메뉴에서 [Cluster1] 콤보박스를 선택하고 [서비스 추가] 선택
ㄴ Step02. 추가할 서비스 유형 중 [Hbase]을 선택, 우측 하단 [계속] 클릭
ㄴ Step03. 역할 할당 사용자 지정을 아래와 같이 설정 후 [계속] 클릭
· Master : server01
· HBase REST Server : 사용하지 않음
· HBase Thrift Server : server01
· RegionServer : 모든 호스트

ㄴ Step04. 변경 내용 검토 기본값을 유지하고 [계속] 클릭
ㄴ Step05. 주요 환경 구성 실행 명령 완료 후 [계속] 클릭
ㄴ Step06. HBase 설치 [완료] 클릭
ㄴ Step07. CM 홈 → [HBase] → [구성], 검색란에 "HBase Thrift Http 서버 설정" 입력
· HBase(서비스 전체) 체크 후 [변경 내용 저장] 클릭

ㄴ Step08. CM 홈 → [HBase] → [재시작] 클릭
ㄴ Step09. HBase 설치로 다른 소프트웨어도 설정값 변동이 발생하므로 재배포를 통한 변경사항을 반영
· HBase [재배포 아이콘] 클릭 → [이전 서비스 재시작] 클릭 → [지금 재시작] 클릭

ㄴ Step10. HBase Shell 접속 및 기본 테이블 생성

```
#hbase shell

hbase> version
hbase> create 'smartcar_test_table', 'cf'
hbase> put 'smartcar_test_table', 'a101', 'cf:model', 'Avante'
hbase> put 'smartcar_test_table', 'a101', 'cf:speed', '80'
hbase> put 'smartcar_test_table', 'a101', 'cf:date', '20xxxxxx'
hbase> get 'smartcar_test_table', 'a101'
hbase> exit
```

실습2-3 Hue 설치/설정하기

ㄴ Hue 설치 전 설정 & 반드시 Hive 먼저 설치(p29 실습2-1 Hive 설치 참고)

공유폴더 설정 매뉴얼 '[Bigdata] Hue 설치 전 시스템 설정 작업.txt' 참고

ㄴ Step01. CM 홈 왼쪽 메뉴에서 [Cluster1] 콤보박스를 선택하고 [서비스 추가] 선택
ㄴ Step02. 추가할 서비스 유형 중 [Hue]을 선택, 우측 하단 [계속] 클릭
ㄴ Step03. 역할 할당 사용자 지정을 아래와 같이 설정 후 [계속] 클릭
  · Hue Server : server01
  · Load Balancer : 사용하지 않음

ㄴ Step04. 데이터베이스 연결 테스트 후 [계속] 클릭
ㄴ Step05. 주요 환경 구성 실행 명령 완료 후 [계속] 클릭
ㄴ Step06. Hue 설치 [완료] 클릭
ㄴ Step07. CM 홈 → [Hue] → [구성], 검색란에 "시간대" 입력
  · 변경 전 : America/Los_Angeles
  · 변경 후 : Asia/Seoul

ㄴ Step08. CM 홈 → [Hue] → [구성], 검색란에 "HBase Thrift 서버" 입력
  · 선택 : HBase Thrift Server (server01)

ㄴ Step09. 설정을 모두 변경 후 CM 홈 → [Hue] → [재시작]
ㄴ Step10. Hue 웹UI 접속
  · Hue 웹 UI : http://192.168.56.101:8888
  · 계정 : admin
  · 암호 : admin

3) 적재 테스트

스마트카 상태 정보 로그파일(SmartCarStatusInfo_2020xxxx.txt))은 플럼의 Agent가 정상적으로 작동하고 있다면 SpoolDir이 참조하는 /home/pilot/working/car-batch-log 경로에 로그파일이 생성됨과 동시에 flume의 파일 수집 이벤트가 작동해서 HDFS 적재가 시작된다.

운전자 운행 정보파일(SmartCarDriverInfo.log)은 플럼 Hbase Sink를 이용해 HBase 데이터베이스에 실시간 적재한다.

실습3-1 스마트카 상태 정보 적재하기

ㄴ Step01. HDFS에 스마트카 상태 정보 로그파일을 적재하기 위한 Flume Agent 수정

· Agent 이름 : SmartCar_Agent

· 구성 파일 : 기존 내용 모두 제거 후 제공되는 파일 내용을 복사 입력

ㄴ Step02. Flume 설정 내용을 반영하기 위해 Flume 재시작

ㄴ Step03. SmartCar 로그 시뮬레이터 실행

```
#cd /home/pilot/working
#java -cp bigdata.smartcar.log.jar com.bigdata.smartcar.log.CarLogMain 20200822 100 &
```

ㄴ Step04. 차량 상태 데이터 로그 확인

```
#cd /home/pilot/working/SmartCar
#tail -f SmartCarStatusInfo_xxxxxxxx.txt
```

ㄴ Step05. Hadoop 적재를 위해 플럼 이벤트 디렉터리(SpoolDir)로 적재할 파일 이동

```
#cd /home/pilot/SmartCar
#mv ./SmartCarStatusInfo_xxxxxxxx.txt /home/pilot/working/car-batch-log/
```

ㄴ Step06. HDFS 적재 파일 확인

```
#hdfs dfs -ls /
#hdfs dfs -cat /pilot/collect/car-batch-log/wrk_date=20xxxxxx/car-batch-log.xxxxxxxxx.log
```

ㄴ Step07. SmartCar 로그 시뮬레이터 종료

```
#ps -ef | grep smartcar.log
#kill -9 [pid]
```

실습3-2 스마트카 운전자 운행 정보 적재하기

ㄴ Step01. HBase에 스마트카 상태 정보 로그파일을 적재하기 위한 Flume Agent 수정
- Agent 이름 : SmartCar_Agent
- 구성 파일 : 기존 내용 모두 제거 후 제공되는 파일 내용을 복사 입력

ㄴ Step02. Flume 설정 내용을 반영하기 위해 Flume 재시작

ㄴ Step03. HBase에 실시간 적재를 위한 테이블 생성

```
#hbase shell

hbase> create 'DriverCarInfo', 'cf1'
hbase> exit
```

ㄴ Step04. SmartCar 로그 시뮬레이터 실행

```
#cd /home/pilot/working
#java -cp bigdata.smartcar.log.jar com.bigdata.smartcar.log.DriverLogMain 20200822 100 &
```

ㄴ Step05. 실시간 운전자 운행 데이터 로그 확인

```
#cd /home/pilot/working/driver-realtime-log
#tail -f SmartCarDriverInfo.log
```

ㄴ Step06. HBase 적재 데이터 확인

```
#hbase shell

hbase> list
hbase> count 'DriverCarInfo'
hbase> scan 'DriverCarInfo', {LIMIT=>10}
```

ㄴ Step07. SmartCar 로그 시뮬레이터 종료

```
#ps -ef | grep smartcar.log
#kill -9 [pid]
```

실습3-3 Hue를 이용해 HDFS/HBase에 적재된 파일/데이터 확인

ㄴ Step01. Hue 웹UI 접속
- Hue 웹 UI : http://192.168.56.101:8888
- 계정 : admin
- 암호 : admin

ㄴ Step02. Hue → 메뉴 → [파일] 확인

ㄴ Step03. Hue → 메뉴 → [HBase] 확인

1.6 빅데이터 파일럿 프로젝트 처리탐색

1) 처리탐색 아키텍처

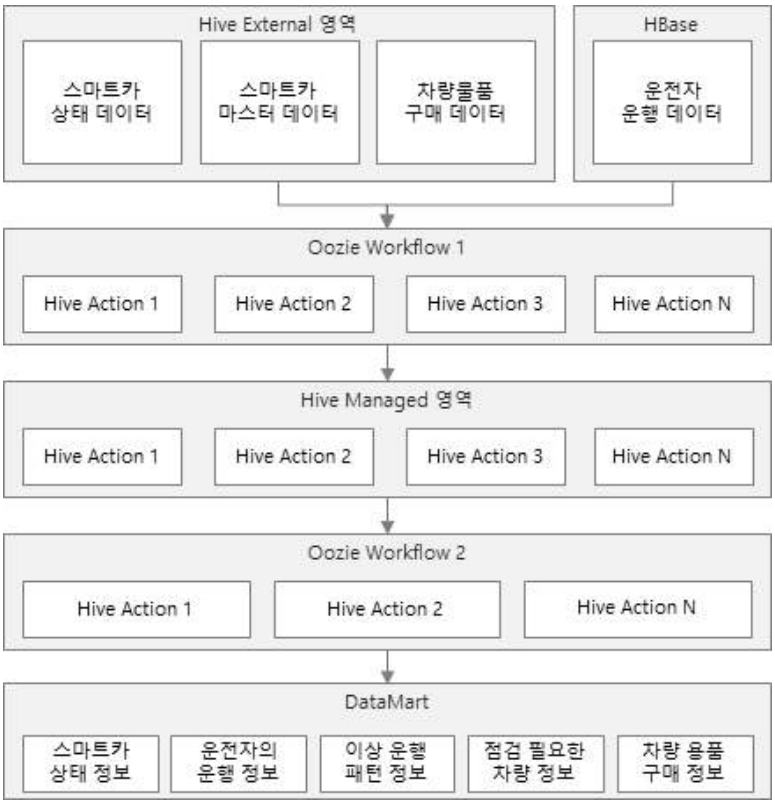
파일럿 프로젝트의 탐색/처리에서는 요구사항1에서 스마트카 차량의 기능별 상태를 점검하고, 요구사항2의 주행 패턴을 분석하기 위해서 적재된 데이터를 탐색 및 가공해서 분석하기 쉬운 데이터셋으로 재구성한다. 아래는 요구사항과 관련 요구사항을 구체화하고 이를 위한 기술 요건과 해결 방안으로 요구사항 구체화 열의 5번 항목은 4개의 데이터셋에서 1개가 추가되어 5개의 주제 영역으로 확장됐다.

· **요구사항1 :** 차량의 다양한 장치로부터 발생하는 로그파일을 수집해서 기능별 상태를 점검한다.
· **요구사항2 :** 운전자의 운행 정보가 담긴 로그를 실시간으로 수집해서 주행 패턴을 분석한다.

· **요구사항 구체화 및 분석**

탐색 요구사항 구체화	분석 및 해결방안
적재된 데이터는 하이브의 웨어하우스로 관리 되어야 한다.	초기 HDFS에 적재된 영역을 하이브 External영역으로 정의하고, 하이브의 데이터 웨어하우스 기능을 이용해 External → Managed → Mart 영역을 단계적으로 구성
데이터마트 구축에 필요한 데이터를 추가로 구성할 수 있어야 한다.	스마트카의 기본정보 데이터, 운전자의 차량용품 구매 이력 데이터셋을 HDFS External 영역에 적재
하이브의 데이터 웨어하우스의 이력성 데이터들을 일자별로 관리돼야 한다.	데이터 웨어하우스의 External 영역은 작업 처리일을 기준으로 파티션을 구성하며, Managed 영역은 데이터 생성일 기준으로 구성
분석 마트가 만들어 지는 일련의 과정들은 워크플로로 만들어져 관리돼야 한다.	데이터 웨어하우스를 만들기 위해 하이브-QL을 Job Designer에 등록해서 워크플로우로 만들고 완성된 워크플로우는 스케줄러에 등록 및 관리
최종 마트로 만들어질 데이터셋들은 주제영역별로 구성되어야 한다.	스마트카 분석 마트의 주제영역을 5개로 확장 ❶ 스마트카의 상태 모니터링 정보 ❷ 스마트카 운전자의 운행 기록 정보 ❸ 이상 운전 패턴 스마트카 정보 ❹ 운전자의 차량용품 구매 이력 정보 ❺ 긴급 점검이 필요한 스마트카 정보

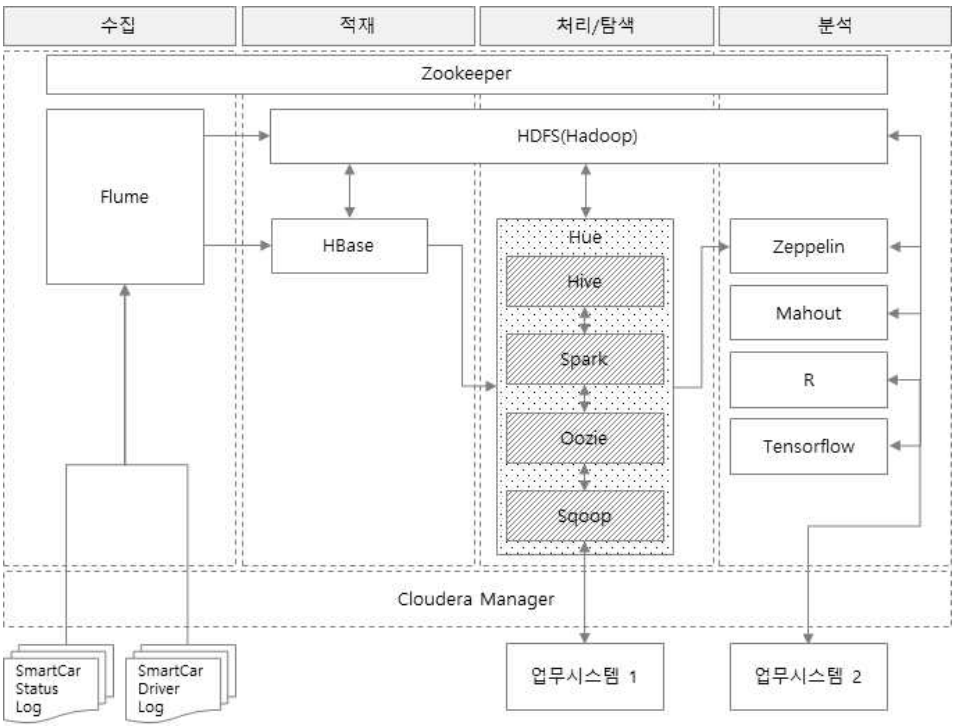
· 파일럿 프로젝트 처리탐색 아키텍처



2) 처리탐색 환경구성

파일럿 프로젝트의 탐색적 분석을 위해 Hive, Spark, Oozie를 설치한다.

· 파일럿 프로젝트 처리탐색 환경



실습2-1 Hive 설치/설정하기

ㄴ Step01. CM 홈 왼쪽 메뉴에서 [Cluster1] 콤보박스 → [서비스 추가] 선택
ㄴ Step02. 추가할 서비스 유형 중 [Hive]을 선택, 우측 하단 [계속] 클릭
ㄴ Step03. Hive 작동에 필요한 의존성 선택
  · HBase가 포함된 항목 선택 후 [계속] 클릭

ㄴ Step04. 역할 할당 사용자 지정을 아래와 같이 설정 후 [계속] 클릭
  · Gateway : server01 선택
  · Hive Metastore Server : server01 선택
  · WebHCat Server : 미설치
  · HiveServer2 : server01 선택

ㄴ Step05. Hive MetaStore 데이터베이스 설정
  · 기본값을 그대로 유지하고 [테스트 연결] → [계속] 클릭

ㄴ Step06. Hive 데이터웨어하우스 경로, 포트 설정
  · 기본값을 그대로 유지하고 [계속] 클릭

ㄴ Step07. 첫 번째 실행 명령 완료 후 [계속] 클릭
ㄴ Step08. Hive 설치 [완료] 클릭
ㄴ Step09. Hive 재시작
  · CM 홈 → [Hive] → [재시작] 클릭

실습2-2 Spark 설치/설정하기

ㄴ Step01. CM 홈 왼쪽 메뉴에서 [Cluster1] 콤보박스 → [서비스 추가] 선택
ㄴ Step02. 추가할 서비스 유형 중 [Spark]을 선택, 우측 하단 [계속] 클릭
ㄴ Step03. Spark 작동에 필요한 의존성 선택
  · HBase가 포함된 항목 선택 후 [계속] 클릭

ㄴ Step04. 역할 할당 사용자 지정을 아래와 같이 설정 후 [계속] 클릭
  · History Server : server01 선택
  · Gateway : server01 선택

ㄴ Step07. 첫 번째 실행 명령 완료 후 [계속] 클릭
ㄴ Step08. Spark 설치 [완료] 클릭
ㄴ Step09. Spark는 YARN 서비스와 의존관계로 YARN과 Spark 각 재시작
  · YARN 재시작 : CM 홈 → [YARN] → [재시작] 클릭
  · Spark 재배포 : CM 홈 → [Spark] → [클라이언트 구성 배포] 아이콘 클릭
  · Spark 재시작 : CM 홈 → [Spark] → [재시작] 클릭

실습2-3 Oozie 설치/설정하기

ㄴ Step01. CM 홈 왼쪽 메뉴에서 [Cluster1] 콤보박스 → [서비스 추가] 선택

ㄴ Step02. 추가할 서비스 유형 중 [Oozie]을 선택, 우측 하단 [계속] 클릭

ㄴ Step03. Oozie 작동에 필요한 의존성 선택

· HBase가 포함된 항목 선택 후 [계속] 클릭

ㄴ Step04. 역할 할당 사용자 지정을 아래와 같이 설정 후 [계속] 클릭

· Oozie Server : server01 선택

ㄴ Step05. Oozie 내장 데이터베이스 설정

· 기본값을 그대로 유지하고 [테스트 연결] → [계속] 클릭

ㄴ Step06. Oozie 루트경로, 데이터 디렉터리 설정

· 기본값을 그대로 유지하고 [계속] 클릭

ㄴ Step07. 첫 번째 실행 명령 완료 후 [계속] 클릭

ㄴ Step08. Oozie 설치 [완료] 클릭

ㄴ Step09. Oozie 기본 메모리값 수정

· CM 홈 → [Oozie] → [구성] 클릭, "Launcher Memory" 검색

· 변경 전 2GB → 변경 후 1GB

ㄴ Step10. Oozie TimeZone 변경

· CM 홈 → [Oozie] → [구성] 클릭 후 왼쪽 범주 → [고급] 클릭

· oozie-site.xml에 대한 Oozie Server 고급 구성에 아래와 같이 입력

ㄴ Step10. Oozie 재시작

· CM 홈 → [Oozie] → [재시작] 클릭

3) 처리탐색 테스트

Hue에 접속해 Hue의 편리한 기능들을 이용해 스마트카 데이터를 탐색한다. 그리고 Hive와 Spark를 이용해 데이터마트를 구성한다. 주로 External 영역에 적재된 데이터를 Managed 영역으로 통합은 Hive-QL를 이용하며 일련의 작업들을 Oozie의 워크플로우 정의해 batch job으로 실행한다.

실습3-1 Hive를 이용한 External 데이터 탐색하기

ㄴ Step01. [Hue 웹 UI] 접속

· http://192.168.56.101:8888 admin/admin

ㄴ Step02. Hue → [쿼리] → [편집기] → [Hive] 클릭

ㄴ Step03. 스마트카 상태 정보 External 테이블 생성 및 파티션 정보 추가

· Table 생성

```
create external table if not exists SmartCar_Status_Info (
    reg_date string,
    car_number string,
    tire_fl string,
    tire_fr string,
    tire_bl string,
    tire_br string,
    light_fl string,
    light_fr string,
    light_bl string,
    light_br string,
    engine string,
    break string,
    battery string
)
partitioned by(wrk_date string)
row format delimited
fields terminated by ','
stored as textfile
location '/pilot/collect/car-batch-log/'
```

· /pilot/collect/car-batch-log/wrk_date=20200311 데이터를 조회를 위해 파티션 정보 추가

```
ALTER TABLE SmartCar_Status_Info ADD PARTITION(wrk_date='20200311');
```

ㄴ Step03. 스마트카 상태 정보 테이블 조회

```
select * from SmartCar_Status_Info limit 5;
```

ㄴ Step04. 스마트카 상태 정보 테이블 조건 조회

```
select car_number, avg(battery) as battery_avg
from SmartCar_Status_Info
where battery < 60
group by car_number;
```

실습3-2 Hive를 이용한 HBase 데이터 탐색하기

ㄴ Step01. [Hue 웹 UI] 접속

· http://192.168.56.101:8888 admin/admin

ㄴ Step02. Hue → [쿼리] → [편집기] → [Hive] 클릭

ㄴ Step03. 스마트카 운행 정보 External 테이블 생성

```
CREATE EXTERNAL TABLE SmartCar_Drive_Info(
  r_key string,
  r_date string,
  car_number string,
  speed_pedal string,
  break_pedal string,
  steer_angle string,
  direct_light string,
  speed string,
  area_number string)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES (
 "hbase.columns.mapping" = "cf1:Date,
                            cf1:CarNum,
                            cf1:AccStep,
                            cf1:BrkStep,
                            cf1:WheelStep,
                            cf1:DirLightStep,
                            cf1:Speed,
                            cf1:AreaNum")
TBLPROPERTIES(
  "hbase.table.name" = "DriverCarInfo");
```

ㄴ Step03. 스마트카 상태 정보 테이블 조회

```
SELECT* FROM SmartCar_Drive_Info LIMIT 10;
```

실습3-3 데이터셋 추가/탐색하기

ㄴ Step01. [Hue 웹 UI] 접속

· http://192.168.56.101:8888 admin/admin

ㄴ Step02. Hue → [메뉴] → [파일] 클릭 후 디렉터리 생성

· /pilot/collect/car-list 디렉터리 생성
· /pilot/collect/item-list 디렉터리 생성
· /pilot/collect/car-master 디렉터리 생성
· /pilot/collect/buy-list 디렉터리 생성

ㄴ Step03. 추가로 제공되는 데이터 파일 업로드

· CarList.txt 파일을 /pilot/collect/car-list 업로드
· ItemList.txt 파일을 /pilot/collect/item-list 업로드
· CarItemBuyList_202003.txt 파일을 /pilot/collect/buy-list 업로드
· CarMaster.txt 파일을 /pilot/collect/car-master 업로드
· CarItemBuyList_202003.txt 파일을 /pilot/collect/buy-list 업로드

ㄴ Step04. 추가한 두 개의 데이터셋을 탐색하기 위한 Hive External 테이블 생성 및 조회

· SmartCar_Car_List 테이블 생성

```
create external table SmartCar_Car_List (
    car_seq int,
    car_num string
)
row format delimited
fields terminated by ','
stored as textfile
location '/pilot/collect/car-list'
```

· SmartCar_Item_List 테이블 생성

```
create external table SmartCar_Item_List (
    item_seq  int,
    item_name string
)
row format delimited
fields terminated by ','
stored as textfile
location '/pilot/collect/item-list'
```

· SmartCar_Master 테이블 생성

```
CREATE EXTERNAL TABLE SmartCar_Master (
car_number string,
sex string,
age string,
marriage string,
region string,
job string,
car_capacity string,
car_year string,
car_model string
)
row format delimited
fields terminated by '|'
stored as textfile
location '/pilot/collect/car-master';
```

· SmartCar_Item_BuyList 테이블 생성

```
CREATE EXTERNAL TABLE SmartCar_Item_BuyList (
car_number string,
Item string,
score string,
month string
)
row format delimited
fields terminated by ','
stored as textfile
location '/pilot/collect/buy-list';
```

· SmartCar_Car_List 테이블 조회

```
SELECT * from SmartCar_Car_List;
```

· SmartCar_Item_List 테이블 조회

```
SELECT * from SmartCar_Item_List;
```

· SmartCar_Master 테이블 조회

```
SELECT * FROM SmartCar_Master;
```

· SmartCar_Item_BuyList 테이블 조회

```
SELECT * FROM SmartCar_Item_BuyList LIMIT 10;
```

· SmartCar_Master 18세이상 정제 테이블 생성

```
CREATE TABLE SmartCar_Master_Over18
AS SELECT * FROM SmartCar_Master Where age >= 18;
```

4) Oozie 워크플로우

하이브, 피그, 스파크 등을 이용해 빅데이터의 처리, 탐색, 분석하는 과정은 복잡한 선후행 관계를 맺고 반복적으로 진행된다. 대규모 빅데이터 시스템에서는 수집 및 적재된 수백 개 이상의 데이터셋을 대상으로 다양한 후처리 Job이 데이터 간의 의존성을 지켜가며 복잡하게 실행된다. 이 같은 복잡한 데이터 파이프라인 작업을 방향성 있는 비순환 그래프(DAG)로 Job의 시작, 처리, 분기, 종료 등의 액션을 정의하는 워크플로우를 위해 Oozie를 활용한다. 총 5개 주제 영역에 대한 데이터마트를 구성하며, 5개의 Oozie 워크플로우를 Hue의 Job Designer에서 구성한다.

· 처리탐색 단계의 5가지 주제영역

	주제영역1	**주제영역2**	**주제영역3**	**주제영역4**	**주제영역5**
주제영역	스마트카 상태 모니터링 정보	스마트카 운전자 운행 기록 정보	이상 운전 패턴 스마트카 정보	긴급점검이 필요한 스마트카 정보	운전자 차량용품 구매 정보
이용할 테이블	SmartCar_ Master_Over18 SmartCar_ Status_Info	SmartCar_ Master_Over18 SmartCar_ Drive_Info_2	Managed_ SmartCar_ Drive_Info	Managed_ SmartCar_ Status_Info	SmartCar_ Master_Over18 SmartCar_ Item_BuyList
워크플로우	Subject 1 - Workflow	Subject 2 - Workflow	Subject 3 - Workflow	Subject 4 - Workflow	Subject 5 - Workflow
스케줄러	Subject 1 - Coordinator	Subject 2 - Coordinator	Subject 3 - Coordinator	Subject 4 - Coordinator	Subject 5 - Coordinator
수행주기	01:00 / 1 Day	02:00 / 1 Day	03:00 / 1 Day	04:00 / 1 Day	05:00 / 1 Day
생성되는 Mart 테이블	Managed_ SmartCar_ Status_Info	Managed_ SmartCar_ Drive_Info	Managed_ SmartCar_ Symptom_Info	Managed_ SmartCar_ Emergency_ Check_Info	Managed_ SmartCar_Item_ BuyList_Info

실습4-1 Oozie를 이용한 주제영역1 워크플로우 작성하기

ㄴ Step01. [Hue 웹 UI] 접속

· http://192.168.56.101:8888 admin/admin

ㄴ Step02. Hue → [메뉴] → [문서] 클릭

ㄴ Step03. Hive Script 파일을 저장하기 위한 작업 폴더 생성

· [새 문서] → [새 폴더] 클릭

· 디렉터리 이름 'workflow' 입력 후 [생성] 클릭

· workflow 디렉터리 하위에 hive_script 디렉터리 추가 생성

· hive_script 디렉터리 하위에 subject1, subject2, subject3, subject4, subject5 디렉터리 추가 생성

ㄴ Step04. 주제영역1 첫 번째 Hive Script 파일생성 및 쿼리 작성

· 내 문서 > workflow > hive_script > subject1 이동

· [새 문서] → [Hive 쿼리] 클릭 후 아래 쿼리 작성

```
create table if not exists Managed_SmartCar_Status_Info (
    car_number string,
    sex string,
    age string,
    marriage string,
    region string,
    job string,
    car_capacity string,
    car_year string,
    car_model string,
    tire_fl string,
    tire_fr string,
    tire_bl string,
    tire_br string,
    light_fl string,
    light_fr string,
    light_bl string,
    light_br string,
    engine string,
    break string,
    battery string,
    reg_date string
)
partitioned by( biz_date string )
row format delimited
fields terminated by ','
stored as textfile;
```

· [저장] 클릭 파일명 create_table_managed_smartcar_status_info.hql 입력 후 [Save] 클릭

ㄴ Step05. 주제영역1 두 번째 Hive Script 파일생성 및 쿼리 작성
· 내 문서 > workflow > hive_script > subject1 이동
· [새 문서] → [Hive 쿼리] 클릭 후 아래 쿼리 작성

```
alter table SmartCar_Status_Info
add if not exists partition(wrk_date='${working_day}');
```

· [저장] 클릭 파일명 alter_partition_smartcar_status_info.hql 입력 후 [Save] 클릭

ㄴ Step06. 주제영역1 세 번째 Hive Script 파일생성 및 쿼리 작성
· 내 문서 > workflow > hive_script > subject1 이동
· [새 문서] → [Hive 쿼리] 클릭 후 아래 쿼리 작성

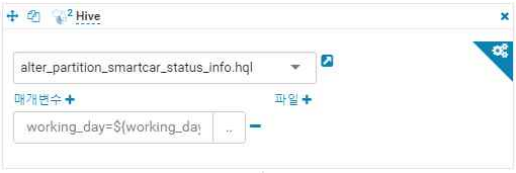
```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;

insert overwrite table Managed_SmartCar_Status_Info partition(biz_date)
select
    t1.car_number,
    t1.sex,
    t1.age,
    t1.marriage,
    t1.region,
    t1.job,
    t1.car_capacity,
    t1.car_year,
    t1.car_model,
    t2.tire_fl,
    t2.tire_fr,
    t2.tire_bl,
    t2.tire_br,
    t2.light_fl,
    t2.light_fr,
    t2.light_bl,
    t2.light_br,
    t2.engine,
    t2.break,
    t2.battery,
    t2.reg_date,
    substring(t2.reg_date, 0, 8) as biz_date
from SmartCar_Master_Over18 t1
join SmartCar_Status_Info t2
on t1.car_number = t2.car_number
and t2.wrk_date = '${working_day}';
```

· [저장] 클릭 파일명 insert_table_managed_smartcar_status_info.hql 입력 후 [Save] 클릭

ㄴ Step07. 상단 쿼리 콤보박스에서 [스케줄러] → [Workflow] 클릭

ㄴ Step08. Oozie 편집기로 첫 번째 워크플로우 작업 추가
· 작업 툴에서 Hive 아이콘을 워크플로우의 첫 번째 노드에 드래그 앤 드롭
· create_table_managed_smartcar_status_info.hql 선택 후 [추가] 클릭

ㄴ Step09. Oozie 편집기로 두 번째 워크플로우 작업 추가
· 작업 툴에서 Hive 아이콘을 워크플로우의 두 번째 노드에 드래그 앤 드롭
· alter_partition_smartcar_status_info.hql 선택 후 [추가] 클릭
· [매개변수+] 클릭 후 working_day=${working_day} 매개변수 입력



ㄴ Step10. Oozie 편집기로 세 번째 워크플로우 작업 추가
· 작업 툴에서 Hive 아이콘을 워크플로우의 세 번째 노드에 드래그 앤 드롭
· insert_table_managed_smartcar_status_info.hql 선택 후 [추가] 클릭
· [매개변수+] 클릭 후 working_day=${working_day} 매개변수 입력

ㄴ Step11. 워크플로우 이름 지정 및 작성 완료
· 작업 툴바 아래 'My Workflow'를 'Subject1 - Workflow'로 변경 → [체크] 아이콘 클릭
· 우측 상단 [저장] 아이콘 클릭

ㄴ Step12. 상단 쿼리 콤보박스에서 [스케줄러] → [예약] 클릭

ㄴ Step13. "My Schedule" 예약 작업 이름을 "Subject1-예약" 입력 후 [체크] 아이콘 클릭
ㄴ Step14. 예약 작업에 사용할 워크플로우 선택
· 앞서 만든 주제영역1의 워크플로우인 'Subject1-Workflow' 선택

ㄴ Step15. 예약 작업을 주기적으로 실행하기 위한 시간 설정
· 실행간격 : 매일, 01시
· 시작일자 : 기본 시작일 사용
· 종료일자 : 2020년 12월 31일, 23시 59분
· 시간대 : Asia/Seoul

ㄴ Step16. 워크플로우에서 사용할 매개변수 값 설정
· 매개변수 working_day 선택
· ${coord:formatTime(coord:dateTzOffset(coord:nominalTime(), "Asia/Seoul"), 'yyyyMMdd')} 입력

ㄴ Step17. 우측상단 [저장] 아이콘 클릭 후 [실행] 아이콘 → [제출] 클릭
ㄴ Step18. 우측상단 [Job] → [일정] 클릭 후 제출된 예약 작업 상태 확인
ㄴ Step19. 예약 작업 실행 후 결과 데이터셋 탐색

```
SELECT * FROM managed_smartcar_status_info
WHERE biz_date = '20200322' LIMIT 10
```

실습4-2 Oozie를 이용한 주제영역2 워크플로우 작성하기

ㄴ Step01. [Hue 웹 UI] 접속

· http://192.168.56.101:8888 admin/admin

ㄴ Step02. Hue → [메뉴] → [문서] 클릭

ㄴ Step03. 주제영역2 첫 번째 Hive Script 파일생성 및 쿼리 작성

· 내 문서 > workflow > hive_script > subject2 이동

· [새 문서] → [Hive 쿼리] 클릭 후 아래 쿼리 작성

```
create external table if not exists SmartCar_Drive_Info_2 (
  r_key string,
  r_date string,
  car_number string,
  speed_pedal string,
  break_pedal string,
  steer_angle string,
  direct_light string,
  speed string,
  area_number string
)
partitioned by( wrk_date string )
row format delimited
fields terminated by ','
stored as textfile
location '/pilot/collect/drive-log/'
```

· [저장] 클릭 파일명 create_table_smartcar_drive_info_2.hql 입력 후 [Save] 클릭

ㄴ Step04. 주제영역2 두 번째 Hive Script 파일생성 및 쿼리 작성

· 내 문서 > workflow > hive_script > subject2 이동

· [새 문서] → [Hive 쿼리] 클릭 후 아래 쿼리 작성

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;

insert overwrite table SmartCar_Drive_Info_2 partition(wrk_date)
select
  r_key ,
  r_date ,
  car_number ,
  speed_pedal ,
  break_pedal ,
  steer_angle ,
  direct_light ,
  speed ,
  area_number ,
  substring(r_date, 0, 8) as wrk_date
from SmartCar_Drive_Info
where substring(r_date, 0, 8) = '${working_day}';
```

· [저장] 클릭 파일명 insert_table_smartcar_drive_info_2.hql 입력 후 [Save] 클릭

ㄴ Step05. 주제영역2 세 번째 Hive Script 파일생성 및 쿼리 작성

· 내 문서 > workflow > hive_script > subject2 이동

· [새 문서] → [Hive 쿼리] 클릭 후 아래 쿼리 작성

```
create table if not exists Managed_SmartCar_Drive_Info (
    car_number string,
    sex string,
    age string,
    marriage string,
    region string,
    job string,
    car_capacity string,
    car_year string,
    car_model string,

    speed_pedal string,
    break_pedal string,
    steer_angle string,
    direct_light string,
    speed string,
    area_number string,
    reg_date string
)
partitioned by( biz_date string )
row format delimited
fields terminated by ','
stored as textfile;
```

· [저장] 클릭 파일명 create_table_managed_smartcar_drive_info.hql 입력 후 [Save] 클릭

ㄴ Step06. 주제영역2 네 번째 Hive Script 파일생성 및 쿼리 작성

· 내 문서 > workflow > hive_script > subject2 이동

· [새 문서] → [Hive 쿼리] 클릭 후 아래 쿼리 작성

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;

insert overwrite table Managed_SmartCar_drive_Info partition(biz_date)
select
    t1.car_number,
    t1.sex,
    t1.age,
    t1.marriage,
    t1.region,
    t1.job,
    t1.car_capacity,
    t1.car_year,
    t1.car_model,
    t2.speed_pedal,
    t2.break_pedal,
    t2.steer_angle,
    t2.direct_light ,
    t2.speed ,
    t2.area_number ,
    t2.r_date,
    substring(t2.r_date, 0, 8) as biz_date
from  SmartCar_Master_Over18 t1 join SmartCar_Drive_Info_2 t2
on t1.car_number = t2.car_number and substring(t2.r_date,0,8) = '${working_day}';
```

· [저장] 클릭 파일명 insert_table_managed_smartcar_drive_info.hql 입력 후 [Save] 클릭

ㄴ Step07. 상단 쿼리 콤보박스에서 [스케줄러] → [Workflow] 클릭

ㄴ Step08. Oozie 편집기로 첫 번째 워크플로우 작업 추가
· 작업 툴에서 Hive 아이콘을 워크플로우의 첫 번째 노드에 드래그 앤 드롭
· create_table_smartcar_drive_info_2.hql 선택 후 [추가] 클릭

ㄴ Step09. Oozie 편집기로 두 번째 워크플로우 작업 추가
· 작업 툴에서 Hive 아이콘을 워크플로우의 두 번째 노드에 드래그 앤 드롭
· insert_table_smartcar_drive_info_2.hql 선택 후 [추가] 클릭
· [매개변수+] 클릭 후 working_day=${working_day} 매개변수 입력

ㄴ Step10. Oozie 편집기로 세 번째 워크플로우 작업 추가
· 작업 툴에서 Hive 아이콘을 워크플로우의 세 번째 노드에 드래그 앤 드롭
· create_table_managed_smartcar_drive_info.hql 선택 후 [추가] 클릭
· HiveQL에 선언한 매개변수 ${working_day}를 연결하기 위해 매개변수 추가

ㄴ Step11. Oozie 편집기로 네 번째 워크플로우 작업 추가
· 작업 툴에서 Hive 아이콘을 워크플로우의 두 번째 노드에 드래그 앤 드롭
· insert_table_managed_smartcar_drive_info.hql 선택 후 [추가] 클릭
· [매개변수+] 클릭 후 working_day=${working_day} 매개변수 입력

ㄴ Step11. 워크플로우 이름 지정 및 작성 완료
· 작업 툴바 아래 'My Workflow'를 'Subject2-Workflow'로 변경 → [체크] 아이콘 클릭
· 우측 상단 [저장] 아이콘 클릭

ㄴ Step12. 상단 쿼리 콤보박스에서 [스케줄러] → [예약] 클릭

ㄴ Step13. "My Schedule" 예약 작업 이름을 "Subject2-예약" 입력 후 [체크] 아이콘 클릭
ㄴ Step14. 예약 작업에 사용할 워크플로우 선택
· 앞서 만든 주제영역2의 워크플로우인 'Subject2-Workflow' 선택

ㄴ Step15. 예약 작업을 주기적으로 실행하기 위한 시간 설정
- 실행간격 : 매일, 02시
- 시작일자 : 기본 시작일 사용
- 종료일자 : 2020년 12월 31일, 23시 59분
- 시간대 : Asia/Seoul

ㄴ Step16. 워크플로우에서 사용할 매개변수 값 설정
- 매개변수 working_day 선택
- ${coord:formatTime(coord:dateTzOffset(coord:nominalTime(), "Asia/Seoul"), 'yyyyMMdd')} 입력

ㄴ Step17. 우측상단 [저장] 아이콘 클릭 후 [실행] 아이콘 → [제출] 클릭
ㄴ Step18. 우측상단 [Job] → [일정] 클릭 후 제출된 예약 작업 상태 확인
ㄴ Step19. 예약 작업 실행 후 결과 데이터셋 탐색

```
1 select * from managed_smartcar_drive_info where biz_date='20200322';
```

실습4-3 Oozie를 이용한 주제영역3 워크플로우 작성하기

ㄴ Step01. [Hue 웹 UI] 접속

· http://192.168.56.101:8888 admin/admin

ㄴ Step02. Hue → [메뉴] → [문서] 클릭

ㄴ Step03. 주제영역3 첫 번째 Hive Script 파일생성 및 쿼리 작성

· 내 문서 > workflow > hive_script > subject3 이동

· [새 문서] → [Hive 쿼리] 클릭 후 아래 쿼리 작성

```
create table if not exists Managed_SmartCar_Symptom_Info (
    car_number string,
    speed_p_avg string,
    speed_p_symptom string,
    break_p_avg string,
    break_p_symptom string,
    steer_a_cnt string,
    steer_p_symptom string,
    biz_date string
)
row format delimited
fields terminated by ','
stored as textfile;
```

· [저장] 클릭 파일명 create_table_managed_smartcar_symptom_info.hql 입력 후 [Save] 클릭

ㄴ Step04. 주제영역3 두 번째 Hive Script 파일생성 및 쿼리 작성

· 내 문서 > workflow > hive_script > subject3 이동

· [새 문서] → [Hive 쿼리] 클릭 후 아래 쿼리 작성

```
insert into table Managed_SmartCar_Symptom_Info
select
   t1.car_number,
   t1.speed_p_avg_by_carnum,
   case
    when (abs((t1.speed_p_avg_by_carnum - t3.speed_p_avg) / t4.speed_p_std)) > 2
      then 'abnormal'
    else   'normal'
   end
   as speed_p_symptom_score,
   t1.break_p_avg_by_carnum,
   case
    when (abs((t1.break_p_avg_by_carnum - t3.break_p_avg) / t4.break_p_std)) > 2
      then 'abnormal'
    else   'normal'
   end
   as break_p_symptom_score,
   t2.steer_a_count,
   case
    when (t2.steer_a_count) > 2000
      then 'abnormal'
    else   'normal'
   end
   as steer_p_symptom_score,
   t1.biz_date
from
    (select car_number, biz_date, avg(speed_pedal) as speed_p_avg_by_carnum, avg(break_pedal) as break_p_avg_by_carnum
    from managed_smartcar_drive_info where biz_date = '${working_day}'
    group by car_number, biz_date) t1
join
    (select car_number, count(*) as steer_a_count
    from managed_smartcar_drive_info where steer_angle in ('L2','L3','R2','R3') and biz_date = '${working_day}'
    group by car_number) t2
on
    t1.car_number = t2.car_number,
    (select avg(speed_pedal) as speed_p_avg, avg(break_pedal) as break_p_avg from managed_smartcar_drive_info ) t3,
    (select stddev_pop(s.speed_p_avg_by_carnum) as speed_p_std, stddev_pop(s.break_p_avg_by_carnum) as break_p_std
    from (select car_number, avg(speed_pedal) as speed_p_avg_by_carnum, avg(break_pedal) as break_p_avg_by_carnum
          from managed_smartcar_drive_info
          group by car_number) s) t4
```

· [저장] 클릭 파일명 insert_table_managed_smartcar_symptom_info.hql 입력 후 [Save] 클릭

ㄴ Step05. 상단 쿼리 콤보박스에서 [스케줄러] → [Workflow] 클릭

ㄴ Step06. Oozie 편집기로 첫 번째 워크플로우 작업 추가
· 작업 툴에서 Hive 아이콘을 워크플로우의 첫 번째 노드에 드래그 앤 드롭
· create_table_managed_smartcar_symptom_info.hql 선택 후 [추가] 클릭

ㄴ Step07. Oozie 편집기로 두 번째 워크플로우 작업 추가
· 작업 툴에서 Hive 아이콘을 워크플로우의 두 번째 노드에 드래그 앤 드롭
· insert_table_managed_smartcar_symptom_info.hql 선택 후 [추가] 클릭
· [매개변수+] 클릭 후 working_day=${working_day} 매개변수 입력

ㄴ Step08. 워크플로우 이름 지정 및 작성 완료
· 작업 툴바 아래 'My Workflow'를 'Subject3-Workflow'로 변경 → [체크] 아이콘 클릭
· 우측 상단 [저장] 아이콘 클릭

ㄴ Step12. 상단 쿼리 콤보박스에서 [스케줄러] → [예약] 클릭

ㄴ Step09. "My Schedule" 예약 작업 이름을 "Subject3-예약" 입력 후 [체크] 아이콘 클릭
ㄴ Step10. 예약 작업에 사용할 워크플로우 선택
· 앞서 만든 주제영역3의 워크플로우인 'Subject3-Workflow' 선택

ㄴ Step11. 예약 작업을 주기적으로 실행하기 위한 시간 설정
· 실행간격 : 매일, 03시
· 시작일자 : 기본 시작일 사용
· 종료일자 : 2020년 12월 31일, 23시 59분
· 시간대 : Asia/Seoul

ㄴ Step12. 워크플로우에서 사용할 매개변수 값 설정
· 매개변수 working_day 선택
· ${coord:formatTime(coord:dateTzOffset(coord:nominalTime(), "Asia/Seoul"), 'yyyyMMdd')} 입력

ㄴ Step13. 우측상단 [저장] 아이콘 클릭 후 [실행] 아이콘 → [제출] 클릭
ㄴ Step14. 우측상단 [Job] → [일정] 클릭 후 제출된 예약 작업 상태 확인
ㄴ Step15. 예약 작업 실행 후 결과 데이터셋 탐색

```
SELECT
    car_number,
    cast(speed_p_avg as int),
    speed_p_symptom,
    cast(break_p_avg as float),
    break_p_symptom,
    cast(steer_a_cnt as int),
    steer_p_symptom,
    biz_date
FROM managed_smartcar_symptom_info
where biz_date = '20200322'

```

실습4-4 Oozie를 이용한 주제영역4 워크플로우 작성하기

ㄴ Step01. [Hue 웹 UI] 접속

· http://192.168.56.101:8888 admin/admin

ㄴ Step02. Hue → [메뉴] → [문서] 클릭

ㄴ Step03. 주제영역4 첫 번째 Hive Script 파일생성 및 쿼리 작성

· 내 문서 > workflow > hive_script > subject4 이동

· [새 문서] → [Hive 쿼리] 클릭 후 아래 쿼리 작성

```
create table if not exists Managed_SmartCar_Emergency_Check_Info (
    car_number string,
    tire_check string,
    light_check string,
    engine_check string,
    break_check string,
    battery_check string,
    biz_date string
)
row format delimited
fields terminated by ','
stored as textfile;
```

· [저장] 클릭 파일명 create_table_managed_smartcar_emergency_check_info.hql 입력 후 [Save] 클릭

ㄴ Step04. 주제영역4 두 번째 Hive Script 파일생성 및 쿼리 작성

· 내 문서 > workflow > hive_script > subject4 이동

· [새 문서] → [Hive 쿼리] 클릭 후 아래 쿼리 작성

```
insert into table Managed_SmartCar_Emergency_Check_Info

select
        t1.car_number,
        t2.symptom as tire_symptom,
        t3.symptom as light_symptom,
        t4.symptom as engine_symptom,
        t5.symptom as break_symptom,
        t6.symptom as battery_symptom,
        t1.biz_date
from
       (select distinct car_number as car_number, biz_date from managed_smartcar_status_info  where biz_date = '${working_day}') t1

left outer join (  select
                    car_number,
                    avg(tire_fl) as tire_fl_avg ,
                    avg(tire_fr) as tire_fr_avg ,
                    avg(tire_bl) as tire_bl_avg ,
                    avg(tire_br) as tire_br_avg ,
                    'Tire Check' as symptom
                   from managed_smartcar_status_info where biz_date ='${working_day}'
                   group by car_number
                   having  tire_fl_avg < 80 or tire_fr_avg < 80 or  tire_bl_avg < 80 or tire_br_avg < 80 ) t2
on t1.car_number = t2.car_number

left outer join (  select
                    distinct car_number,
                    'Light Check' as symptom
                   from managed_smartcar_status_info
                   where biz_date = '${working_day}' and (light_fl = '2' or light_fr = '2' or light_bl = '2' or light_br = '2')) t3
on t1.car_number = t3.car_number

left outer join (   select
                    distinct car_number,
                   'Engine Check' as symptom
                   from managed_smartcar_status_info
                   where biz_date = '${working_day}' and engine = 'C' ) t4
on t1.car_number = t4.car_number

left outer join (   select
                    distinct car_number,
                    'Brake Check' as symptom
                   from managed_smartcar_status_info
                   where biz_date = '${working_day}' and break = 'C' ) t5

on t1.car_number = t5.car_number

left outer join (   select
                    car_number,
                    avg(battery) as battery_avg,
                    'Battery Check' as symptom
                   from managed_smartcar_status_info where biz_date = '${working_day}'
                   group by car_number having battery_avg < 30 ) t6
on t1.car_number = t6.car_number

where t2.symptom is not null or t3.symptom is not null or t4.symptom is not null  or t5.symptom is not null  or t6.symptom is not null
```

· [저장] 클릭 파일명 insert_table_managed_smartcar_emergency_check_info.hql 입력 후 [Save] 클릭

ㄴ Step05. 상단 쿼리 콤보박스에서 [스케줄러] → [Workflow] 클릭
ㄴ Step06. Oozie 편집기로 첫 번째 워크플로우 작업 추가
· 작업 툴에서 Hive 아이콘을 워크플로우의 첫 번째 노드에 드래그 앤 드롭
· create_table_managed_smartcar_emergency_check_info.hql 선택 후 [추가] 클릭

ㄴ Step07. Oozie 편집기로 두 번째 워크플로우 작업 추가
· 작업 툴에서 Hive 아이콘을 워크플로우의 두 번째 노드에 드래그 앤 드롭
· insert_table_managed_smartcar_emergency_check_info.hql 선택 후 [추가] 클릭
· [매개변수+] 클릭 후 working_day=${working_day} 매개변수 입력

ㄴ Step08. 워크플로우 이름 지정 및 작성 완료
· 작업 툴바 아래 'My Workflow'를 'Subject4-Workflow'로 변경 → [체크] 아이콘 클릭
· 우측 상단 [저장] 아이콘 클릭

ㄴ Step12. 상단 쿼리 콤보박스에서 [스케줄러] → [예약] 클릭

ㄴ Step09. "My Schedule" 예약 작업 이름을 "Subject4-예약" 입력 후 [체크] 아이콘 클릭
ㄴ Step10. 예약 작업에 사용할 워크플로우 선택
· 앞서 만든 주제영역4의 워크플로우인 'Subject4-Workflow' 선택

ㄴ Step11. 예약 작업을 주기적으로 실행하기 위한 시간 설정
· 실행간격 : 매일, 04시
· 시작일자 : 기본 시작일 사용
· 종료일자 : 2020년 12월 31일, 23시 59분
· 시간대 : Asia/Seoul

ㄴ Step12. 워크플로우에서 사용할 매개변수 값 설정
· 매개변수 working_day 선택
· ${coord:formatTime(coord:dateTzOffset(coord:nominalTime(), "Asia/Seoul"), 'yyyyMMdd')} 입력

ㄴ Step13. 우측상단 [저장] 아이콘 클릭 후 [실행] 아이콘 → [제출] 클릭
ㄴ Step14. 우측상단 [Job] → [일정] 클릭 후 제출된 예약 작업 상태 확인
ㄴ Step15. 예약 작업 실행 후 결과 데이터셋 탐색

```
1 select * from managed_smartcar_emergency_check_info where biz_date = '20200322';
```

실습4-5 Oozie를 이용한 주제영역5 워크플로우 작성하기

ㄴ Step01. [Hue 웹 UI] 접속

· http://192.168.56.101:8888 admin/admin

ㄴ Step02. Hue → [메뉴] → [문서] 클릭

ㄴ Step03. 주제영역5 첫 번째 Hive Script 파일생성 및 쿼리 작성

· 내 문서 > workflow > hive_script > subject5 이동

· [새 문서] → [Hive 쿼리] 클릭 후 아래 쿼리 작성

```
create table if not exists Managed_SmartCar_Item_BuyList_Info (
 car_seq int,
 car_number string,
 sex string,
 age string,
 marriage string,
 region string,
 job string,
 car_capacity string,
 car_year string,
 car_model string,
 item_seq int,
 item string,
 score string
)
partitioned by( biz_month string )
row format delimited
fields terminated by ','
stored as textfile;
```

· [저장] 클릭 파일명 create_table_managed_smartcar_item_buylist_info.hql 입력 후 [Save] 클릭

ㄴ Step04. 주제영역5 두 번째 Hive Script 파일생성 및 쿼리 작성

· 내 문서 > workflow > hive_script > subject5 이동

· [새 문서] → [Hive 쿼리] 클릭 후 아래 쿼리 작성

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;

insert overwrite table Managed_SmartCar_Item_BuyList_Info partition(biz_month)
select
   t3.car_seq,
   t1.car_number,
   t1.sex,
   t1.age,
   t1.marriage,
   t1.region,
   t1.job,
   t1.car_capacity,
   t1.car_year,
   t1.car_model,
   t4.item_seq,
   t2.item,
   t2.score,
   t2.month as biz_month
 from
  SmartCar_Master_Over18 t1
 join SmartCar_Item_Buylist t2 on t1.car_number = t2.car_number
 join smartcar_car_list t3     on t2.car_number = t3.car_num
 join smartcar_item_list t4    on t2.item = t4.item_name
where
  t2.month = '202003'
```

· [저장] 클릭 파일명 insert_table_managed_smartcar_item_buylist__info.hql 입력 후 [Save] 클릭

ㄴ Step05. 주제영역5 세 번째 Hive Script 파일생성 및 쿼리 작성
· 내 문서 > workflow > hive_script > subject5 이동
· [새 문서] → [Hive 쿼리] 클릭 후 아래 쿼리 작성

```
insert overwrite local directory '/home/pilot/item-buy-list'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
select car_number, concat_ws("," , collect_set(item))
from managed_smartcar_item_buylist_info
group by car_number
```

· [저장] 클릭 파일명 local_save_managed_smartcar_item_buylist__info.hql 입력 후 [Save] 클릭

ㄴ Step06. 상단 쿼리 콤보박스에서 [스케줄러] → [Workflow] 클릭
ㄴ Step07. Oozie 편집기로 첫 번째 워크플로우 작업 추가
· 작업 툴에서 Hive 아이콘을 워크플로우의 첫 번째 노드에 드래그 앤 드롭
· create_table_managed_smartcar_item_buylist_info.hql 선택 후 [추가] 클릭

ㄴ Step08. Oozie 편집기로 두 번째 워크플로우 작업 추가
· 작업 툴에서 Hive 아이콘을 워크플로우의 두 번째 노드에 드래그 앤 드롭
· insert_table_managed_smartcar_item_buylist__info.hql 선택 후 [추가] 클릭

ㄴ Step09. Oozie 편집기로 세 번째 워크플로우 작업 추가
· 작업 툴에서 Hive 아이콘을 워크플로우의 두 번째 노드에 드래그 앤 드롭
· local_save_managed_smartcar_item_buylist__info.hql 선택 후 [추가] 클릭

ㄴ Step10. 워크플로우 이름 지정 및 작성 완료
· 작업 툴바 아래 'My Workflow'를 'Subject5-Workflow'로 변경 → [체크] 아이콘 클릭
· 우측 상단 [저장] 아이콘 클릭

ㄴ Step11. 상단 쿼리 콤보박스에서 [스케줄러] → [예약] 클릭
ㄴ Step12. "My Schedule" 예약 작업 이름을 "Subject5-예약" 입력 후 [체크] 아이콘 클릭
ㄴ Step13. 예약 작업에 사용할 워크플로우 선택
· 앞서 만든 주제영역5의 워크플로우인 'Subject5-Workflow' 선택

ㄴ Step14. 예약 작업을 주기적으로 실행하기 위한 시간 설정
· 실행간격 : 매일, 05시
· 시작일자 : 기본 시작일 사용
· 종료일자 : 2020년 12월 31일, 23시 59분
· 시간대 : Asia/Seoul

ㄴ Step15. 우측상단 [저장] 아이콘 클릭 후 [실행] 아이콘 → [제출] 클릭
ㄴ Step16. 우측상단 [Job] → [일정] 클릭 후 제출된 예약 작업 상태 확인
ㄴ Step17. 예약 작업 실행 후 결과 데이터셋 탐색

```
SELECT * FROM managed_smartcar_item_buylist_info WHERE biz_month = '202003';
```

4) 마치며

데이터 처리탐색은 초기 적재된 데이터셋들을 탐색 및 가공해서 좀 더 활용도가 높은 새로운 데이터셋들로 만드는 단계이다. 스마트카의 마트를 구성하기 위해 5가지 주제영역을 도출했고, 각 영역별 워크플로우를 작성해 실행 스케줄링을 수립했다.

처리탐색 과정에서 구성한 데이터마트를 이용해 다음 분석단계에서 머신러닝 기반의 고급 분석을 진행한다.

1.7 빅데이터 파일럿 프로젝트 분석

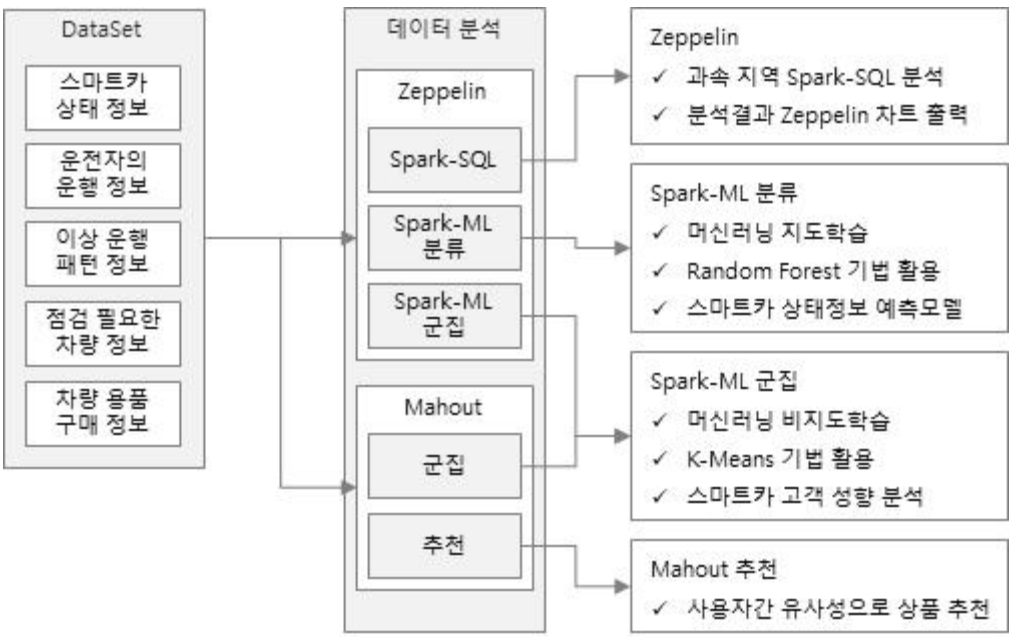
1) 분석 아키텍처

파일럿 프로젝트 탐색을 통해서 요구사항 1, 2에 대한 기본적인 분석 요건은 해결했다. 이번 분석 단계에서는 기존 요구사항을 좀 더 확장해서 빅데이터의 실시간 탐색 및 시각화와 머신러닝을 이용한 데이터 마이닝까지 진행한다.

· 요구사항 구체화 및 분석

분석 요구사항 구체화	분석 및 해결방안
스마트카 데이터셋의 탐색결과를 이해하기 쉽도록 시각화한다.	Zeppelin을 이용해 Spark-SQL로 탐색한 데이터셋을 다양한 차트로 표현
차량용품 구매 이력을 분석해 최적의 상품 추천 목록을 만든다.	Mahout의 추천을 이용해 차량용품 구매 이력을 분석해 성향에 따른 상품 추천 목록을 생성
스마트카의 상태 정보를 분석해 이상 징후를 예측한다.	Spark-ML의 머신러닝 기법중 분류 지도학습을 통해 이상 징후에 대한 예측 모델을 구성
스마트카 운전자의 마스터 정보를 분석해 고객 군집을 도출한다.	Mahout과 Spark-ML로 비지도 학습인 군집분석 수행
스마트카 배기량에 따른 운전자 연소득을 예측한다.	R을 활용하여 회귀분석을 수행
스마트카 주행 중 위험 징후를 판별한다.	Tensorflow를 이용해 딥러닝 수행

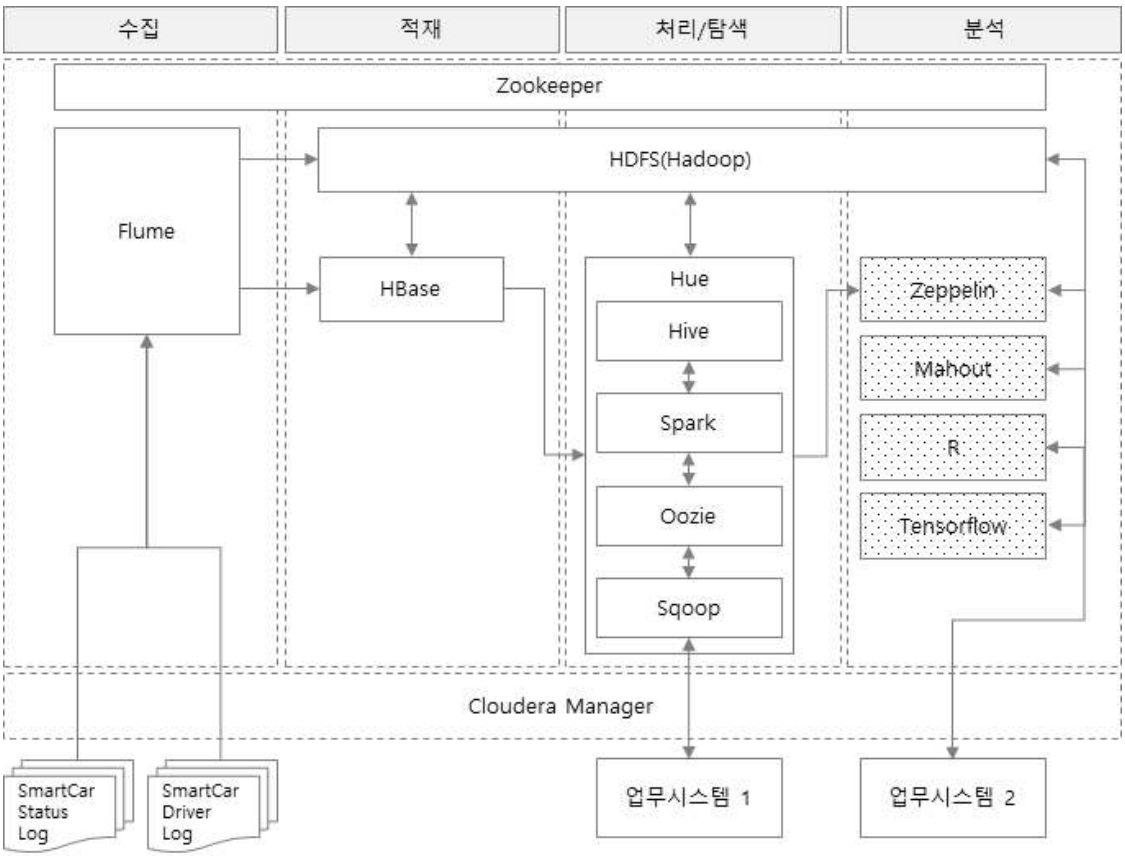
· 파일럿 프로젝트 분석 아키텍처



2) 분석 환경구성

Zeppelin과 CM의 서비스 구성요소가 아니기 때문에 직접 설치해야 한다. R과 Tensorflow는 외부 분석 솔루션으로 파일럿 프로젝트 시스템 외부에 설치한다.

· 파일럿 프로젝트 분석 환경



실습2-1 Zeppelin 설치/설정하기

ㄴ Step01. Zeppelin 0.9.0 버전 다운로드/압축해제/이동/링크 생성

```
#wget http://apache.tt.co.kr/zeppelin/zeppelin-0.9.0/zeppelin-0.9.0-preview2-bin-all.tgz
#tar xvfz zeppelin-0.9.0-preview2-bin-all.tgz
#mv zeppelin-0.9.0-preview2-bin-all /home/bigdata/
#cd /home/bigdata/
#ln -s zeppelin-0.9.0-preview2-bin-all/ zeppelin
```

ㄴ Step02. Zeppelin 환경변수 등록

```
#vi ~/.bashrc
// 맨 아래에 선언 추가
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
export ZEPPELIN_HOME=/home/bigdata/zeppelin
export PATH=$PATH:$ZEPPELIN_HOME/bin
```

ㄴ Step03. 현재 쉘 반영

```
#source ~/.bashrc
```

ㄴ Step04. Zeppelin 환경정보 설정

```
#cd /home/bigdata/zeppelin/conf
#cp zeppelin-env.sh.template zeppelin-env.sh
#vi zeppelin-env.sh

// 18, 74, 83라인 근처 주석 해제 후 입력
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
export SPARK_HOME=/opt/cloudera/parcels/CDH/lib/spark
export HADOOP_CONF_DIR=/etc/hadoop/conf
```

ㄴ Step05. Hive 임시 디렉터리 권한 변경

```
#chmod 777 /tmp/hive
```

ㄴ Step06. Hive 설정파일 복사

```
#cp /etc/hive/conf/hive-site.xml /home/bigdata/zeppelin/conf
```

ㄴ Step08. Zeppelin 서버정보 설정

```
#cd /home/bigdata/zeppelin/conf
#cp zeppelin-site.xml.template zeppelin-site.xml
#vi zeppelin-site.xml

// 29, 35라인 근처 수정
zeppelin.server.addr
변경 전: 127.0.0.1
변경 후: 0.0.0.0

zeppelin.server.port
변경 전: 8080
변경 후: 8081
```

ㄴ Step10. Zeppelin 실행

```
#zeppelin-daemon.sh start
```

ㄴ Step11. Zeppelin 브라우저 확인

· http://192.168.56.101:8081

실습2-2 Mahout 설치/설정하기

ㄴ Step01. Mahout 0.13.0 버전 다운로드/압축해제/이동/링크 생성

```
#wget https://downloads.apache.org/mahout/0.13.0/apache-mahout-dist-0.13.0.tar.gz
#tar xvfz apache-mahout-distribution-0.13.0.tar.gz
#mv apache-mahout-distribution-0.13.0 /home/bigdata/
#cd /home/bigdata/
#ln -s apache-mahout-distribution-0.13.0/ mahout
```

ㄴ Step02. Mahout 환경변수 등록

```
#vi ~/.bashrc
// 맨 아래에 선언 추가
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
export MAHOUT_HOME=/home/bigdata/mahout
export PATH=$PATH:$MAHOUT_HOME/bin
```

ㄴ Step03. 현재 쉘 반영

```
#source ~/.bashrc
```

ㄴ Step04. Mahout 실행

```
#mahout
```

3) 분석 테스트

데이터분석은 단순한 기술적 분석에서부터 머신러닝 같은 고급 분석까지 빅데이터 이전부터 오랜 기간 많은 연구가 이뤄졌던 분야이다. 이 가운데 고급분석 수행에 있어서 분야별 다양한 지식이 필요한데, 파일럿 프로젝트에서는 이러한 개론과 이론 등을 최소화하고 빅데이터 분석을 경험하고 관련 기술 활용에 집중한다.

실습3-1 Zeppelin을 이용한 SmartCar 탐색 분석하기

ㄴ Step01. Zeppelin 브라우저 접속

· http://192.168.56.101:8081

ㄴ Step02. 메인 → [Notebook] → [Create new note] 클릭

· Note Name : SmartCar-Project 입력

· Default Interpreter : spark 선택

ㄴ Step03. 스마트카 운행 지역 분석

· 데이터셋 로드

```
val df = spark.read.csv("/pilot/collect/drive-log/wrk_date=20200831/000000_0")
df.show
```

· 컬럼명 지정

```
val drive_info_df = df.select('_c0.as("r_key"),
                              '_c1.as("r_date"),
                              '_c2.as("car_number"),
                              '_c3.as("speed_pedal"),
                              '_c4.as("break_pedal"),
                              '_c5.as("steer_angle"),
                              '_c6.as("direct_light"),
                              '_c7.as("speed"),
                              '_c8.as("area_number"))

drive_info_df.show
```

· 평균 그룹핑

```
val drive_info_speed_avg_df = drive_info_df
                                .groupBy("area_number", "car_number")
                                .agg("speed"->"mean")
                                .sort(desc("avg(speed)"))
drive_info_speed_avg_df.show()
```

· 컬럼명 변경

```
val drive_info_speed_avg_df2 = drive_info_speed_avg_df
                                .select($"area_number",
                                        $"car_number",
                                        $"avg(speed)".as("speed_avg"))
drive_info_speed_avg_df2.show
```

· 시각화

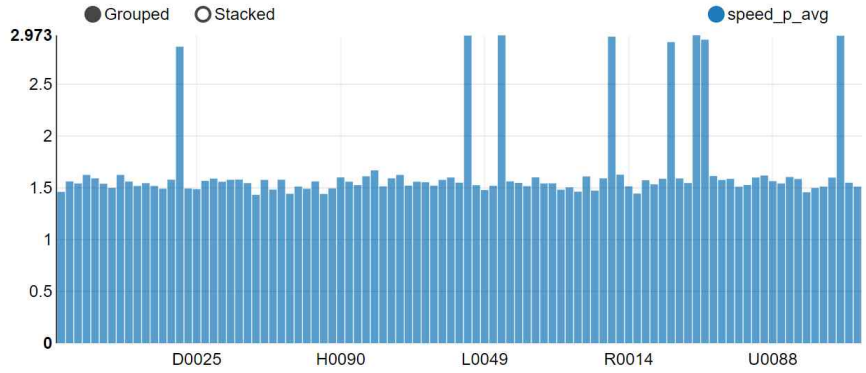
```
z.show(drive_info_speed_avg_df2)
```

ㄴ Step03. 스마트카 이상 운전 패턴 분석

```
%spark.sql
select * from managed_smartcar_symptom_info where biz_date='20200829';
```

SPARK JOB FINISHED

settings

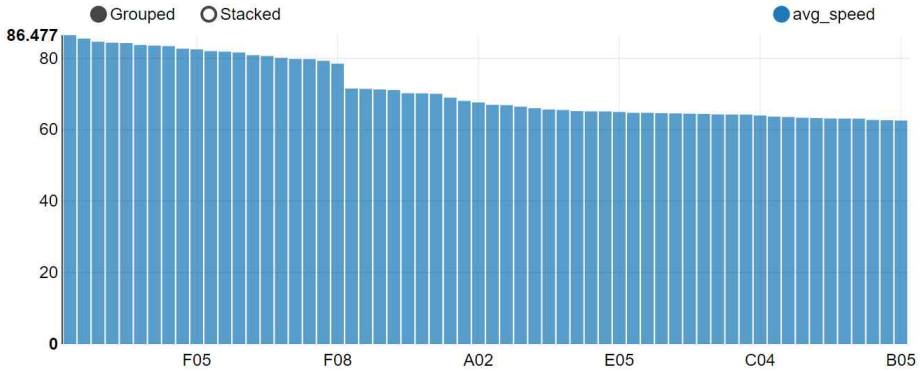


ㄴ Step04. 스마트카 운행 지역 분석

```
%spark.sql
select T1.area_number, T1.avg_speed
from  (select area_number, avg(speed) as avg_speed
       from managed_smartcar_drive_info
       group by area_number) T1
order by T1.avg_speed desc
```

SPARK JOB FINISHED

settings



실습3-2 Spark-ML을 이용한 상태 정보 예측/분류하기

ㄴ Step01. Hive를 이용한 Train Dataset 생성
- 제공되는 HiveQL 내용을 [Hue] → Hive Editor에 복사 입력
- HiveQL 실행 후 /home/pilot/spark-data/classification/input 로컬 디렉터리 확인

ㄴ Step02. 데이터셋 파일을 하나의 파일로 합친 후 HDFS에 적재

```
#cd /home/pilot/spark-data/classification/input
#cat 000000_0 000001_0 > train_dataset.txt
#hdfs dfs -mkdir -p /pilot/spark-data/classification/input
#hdfs dfs -put train_data.txt /pilot/spark-data/classification/input
```

ㄴ Step03. Zeppelin 브라우저 접속
- http://192.168.56.101:8081

ㄴ Step04. 메인 → [Notebook] → [Create new note] 클릭
- Note Name : SmartCar-Classification 입력
- Default Interpreter : spark 선택

ㄴ Step05. Spark 머신러닝 프로그래밍 코드 입력 후 분석 실행
- 제공되는 Spark-ML 프로그래밍 코드를 zeppelin Paragraph 복사 입력
- 분석 및 실행

실습3-3 Mahout을 이용한 차량용품 추천하기

ㄴ Step01. Mahout 추천기에 사용할 분석 파일 생성

```
insert overwrite local directory '/home/pilot/mahout-data/recommend/input'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
select car_seq, item_seq, score from managed_smartcar_item_buylist_info
```

ㄴ Step02. Mahout 분석 파일 확인 및 HDFS에 적재

```
#cd /home/pilot/mahout-data/recommend/input
#hdfs dfs -mkdir -p /pilot/mahout/recommend/input
#hdfs dfs -put * /pilot/mahout/recommend/input/item_buylist.txt
```

ㄴ Step03. Mahout 추천 분석 실행

```
#mahout recommenditembased -i /pilot/mahout/recommend/input/item_buylist.txt
 -o /pilot/mahout/recommend/output/ -s SIMILARITY_COOCCURRENCE -n 3
```

· i : 추천 분석에 사용할 입력 데이터
· o : 추천 분석 결과가 출력될 경로
· s : 추천을 위한 유사도 알고리즘
· n : 추천할 아이템 개수

· 추천 분석을 재실행할 때는 기존 결과 파일을 삭제한 후 재실행해야 됨

```
#hdfs dfs -rm -R -skipTrash /pilot/mahout/recommend/output
#hdfs dfs -rm -R -skipTrash /user/root/temp
```

ㄴ Step04. Mahout 분석결과 확인

홈 / pilot / mahout / recommendation / output / **part-r-00000**

```
2	[12:4.5175357,25:4.5009804,8:4.500924]
4	[25:3.703727,10:3.6943502,8:3.6928885]
6	[18:4.519502,30:4.5108085,28:4.50641]
8	[28:3.862285,23:3.8341968,29:3.8341668]
10	[25:3.4069877,29:3.3204365,10:3.3165407]
12	[5:4.217128,14:4.211461,1:4.2077923]
14	[1:4.556062,13:4.5520363,12:4.54902]
18	[1:3.7597537,3:3.757736,7:3.7547634]
20	[29:4.4387584,30:4.420402,20:4.4167914]
```

실습3-4 R을 이용한 운전자의 연소득 예측하기

ㄴ Step01. R 파일럿 분석 데이터셋 생성

· 제공되는 CarMaster2Income.txt 파일을 /pilot/collect/car-master2-income 업로드

· 추가한 파일 탐색하기 위한 SmartCar_Master2Income 테이블 생성

```
CREATE EXTERNAL TABLE SmartCar_Master2Income (
    car_number string,
    sex string,
    age string,
    marriage string,
    region string,
    job string,
    car_capacity int,
    car_year string,
    car_model string,
    income int
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION '/pilot/collect/car-master2-income';
```

· SmartCar_Master2Income 테이블 조회

```
select
    car_number,
    car_capacity,
    income
from SmartCar_Master2Income;
```

ㄴ Step02. 하이브 클라이언트 라이브러리 구성

· RHive 관련 라이브러리 저장하기 위해 R Workspace의 lib디렉터리 생성

· FTP 클라이언트 실행 후 아래의 jar 라이브러리 파일을 R Workspace의 lib디렉터리로 복사

/opt/cloudera/parcels/CDH/jars/hadoop-client-3.0.0-cdh6.3.2.jar
/opt/cloudera/parcels/CDH/jars/hadoop-common-3.0.0-cdh6.3.2.jar
/opt/cloudera/parcels/CDH/jars/hive-jdbc-2.1.1-cdh6.3.2.jar
/opt/cloudera/parcels/CDH/jars/hive-jdbc-2.1.1-cdh6.3.2-standalone.jar
/opt/cloudera/parcels/CDH/jars/hive-service-2.1.1-cdh6.3.2.jar
/opt/cloudera/parcels/CDH/jars/httpclient-4.5.3.jar
/opt/cloudera/parcels/CDH/jars/httpcore-4.4.6.jar
/opt/cloudera/parcels/CDH/jars/libthrift-0.9.3.jar

ㄴ Step03. R-Studio 실행 후 단계별 코드 실행

```
#필요한 패키지 설치
install.packages("DBI")
install.packages("rJava")
install.packages("RJDBC")
install.packages("log4r")

#설치한 패키지 로드
library("DBI")
library("rJava")
library("RJDBC")

#R과 Hive 연동을 위한 R 클래스 path 설정
hive.class.path = list.files(path=c("../lib"), pattern="jar", full.names=T);
hadoop.lib.path = list.files(path=c("../lib"), pattern="jar", full.names=T);
hadoop.class.path = list.files(path=c("../lib"), pattern="jar", full.names=T);
class.path = c(hive.class.path, hadoop.lib.path, hadoop.class.path);
.jinit(classpath=class.path)

#하이브 JDBC 로드
drv <- JDBC("org.apache.hive.jdbc.HiveDriver",
            "../lib/hive-jdbc-2.1.1-cdh6.3.2.jar",
            identifier.quote="`")

#하이브 접속
conn <- dbConnect(drv,
                  "jdbc:hive2://192.168.100.201:10000/default",
                  "hive",
                  "1234")

#하이브 테이블 목록 조회
dbListTables(conn);


#SmartCar_Master2Income 테이블 조회
data <- dbGetQuery(conn, "select * from smartcar_master2income")
View(data)


#조회한 테이블 데이터 시각화 파악
hist(data$smartcar_master2income.income)


#스마트카의 배기량에 따른 운전자의 연소득 예측 회귀모델 생성및 요약
model <- lm(data$smartcar_master2income.income ~
            smartcar_master2income.car_capacity,
            data=data)

summary(model)


#회귀모델 테스트 확인
test_data <- read.csv("../data/CarMaster2Income_Test.txt",
                      sep="|",
                      header=T,
                      encoding = "UTF-8")

predict(model, test_data, interval = "prediction")

```

실습3-5 Tensorflow를 이용한 위험 징후 판별하기

ㄴ Step01. Tensorflow 파일럿 분석 데이터셋 생성

· 제공되는 CarDrivingIncidentInfo.csv 파일을 구글 드라이브 /tensorflow/data 업로드

· 제공되는 CarDrivingIncidentInfo_Test.csv 파일을 구글 드라이브 /tensorflow/data 업로드

ㄴ Step02. Google Colab 실행 후 단계별 코드 실행

· **SmartCar_Incident_model.ipynb**

```
#필요한 라이브러리 임포트
import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt

from time import time
from tensorflow.python.keras.callbacks import TensorBoard
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, roc_auc_score, auc
from keras.utils import to_categorical
from keras.models import load_model

tf.__version__
```

```
#구글 드라이브 연동
from google.colab import drive
drive.mount('/content/drive')
```

```
#데이터셋 로드
df = pd.read_csv('/content/drive/My Drive/Tensorflow_works/data/CarDrivingIncidentInfo.csv')
df
```

```
#데이터 전처리
df_train = df.iloc[:, :-1].values
df_label = df.iloc[:, -1].values

X_train, X_test, Y_train, Y_test = train_test_split(df_train,
                                                    df_label,
                                                    test_size=0.2,
                                                    random_state=1)

Y_train = to_categorical(Y_train)
Y_test = to_categorical(Y_test)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train
```

```
#DNN 모델 구성 및 요약 정보 출력
model = Sequential([
    Dense(10, input_dim=10, activation='relu'),
    Dense(20, activation='relu'),
    Dropout(0.25),
    Dense(10, activation='relu'),
    Dense(3, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])

model.summary()
```

```
#DNN 모델 학습
hist = model.fit(X_train,
                 Y_train,
                 batch_size=2000,
                 epochs=50,
                 validation_data=(X_test, Y_test))
```

```
#모델을 학습 결과
score = model.evaluate(X_test, Y_test, verbose=0)
print(model.metrics_names)
print(score)
```

```
#DNN 모델의 학습 결과 시각화
fig, loss_ax = plt.subplots()
acc_ax = loss_ax.twinx()

loss_ax.plot(hist.history['loss'], 'y', label='train loss')
loss_ax.plot(hist.history['val_loss'], 'r', label='val loss')
acc_ax.plot(hist.history['acc'], 'b', label='train acc')
acc_ax.plot(hist.history['val_acc'], 'g', label='val acc')

loss_ax.set_xlabel('epoch')

loss_ax.set_ylabel('loss')
loss_ax.legend(loc='lower right')

acc_ax.set_ylabel('accuracy')
acc_ax.legend(loc='upper right')

plt.show()
```

```
#모델 저장
model.save('/content/drive/My Drive/Tensorflow_works/smartcar_model')
```

· **SmartCar_Load_Model.ipynb**

```
#필요한 라이브러리 임포트
import tensorflow as tf
import pandas as pd

from keras.models import load_model
from keras.utils import to_categorical
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

tf.__version__
```

```
#구글 드라이브 연동
from google.colab import drive
drive.mount('/content/drive')
```

```
#데이터셋 로드
df = pd.read_csv('/content/drive/My Drive/Tensorflow_works/data/CarDrivingIncidentInfo_Test.csv')
df
```

```
#데이터 전처리
df_train = df.iloc[:, :-1].values
df_label = df.iloc[:, -1].values

X_train, X_test, Y_train, Y_test = train_test_split(df_train,
                                                    df_label,
                                                    test_size=0.2,
                                                    random_state=1)

Y_train = to_categorical(Y_train)
Y_test = to_categorical(Y_test)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train
```

```
#학습된 모델 로드
model = load_model('/content/drive/My Drive/Tensorflow_works/smartcar_model')
model
```

```
#학습된 모델 평가
result = model.evaluate(X_train, Y_train)
result
```

```
#모델 결과 예측
pre_result = model.predict(X_train)

from sklearn.metrics import accuracy_score
score = accuracy_score(pre_result.round(), Y_train)
print('점수 :', score)
```

4) 마치며

오픈소스 기반의 빅데이터 생태계 기술은 다양하면서 발전 속도가 매우 빨라서 개인이 모든 것을 쫓아가기는 불가능에 가깝다. 파일럿 프로젝트의 전체 구축 과정을 통해 빅데이터의 전반적인 큰 이해를 바탕으로 각자의 전문 분야를 확장해야 한다.

부록 참고문헌

참고 자료

[NCS] LM2001020903 빅데이터 수집시스템 개발
[NCS] LM2001020904 빅데이터 저장시스템 개발
[NCS] LM2001020905 빅데이터 처리시스템 개발
[NCS] LM2001020906 빅데이터 분석시스템 개발
[NCS] LM2001020907 머신러닝 기반 데이터 분석
[NCS] LM2001010506 통계기반 데이터 분석
개정2판 실무로 배우는 빅데이터 기술(김강원 지음, 위키북스) - 2020.06
개정2판 하둡 프로그래밍(정재화 지음, 위키북스) - 2017.03
빅데이터 컴퓨팅 기술(박두순 외, 한빛아카데미) - 2014.06
빅데이터 기획 및 분석(주해종/김혜선/김형로 공저, 크라운출판사) - 2017.06
제2판 빅데이터 시대의 하둡 완벽 입문(오오타 카스기 외/김완섭 옮김, 제이펍) - 2014.06
빅데이터 분석을 위한 스파크2 프로그래밍(백성민 지음, 위키북스) - 2018.04
빅데이터 분석 기사(정혜정/장희선 편저, 시대고시기획) - 2020.06
파이썬으로 배우는 데이터 과학 입문과 실습(데이비 실린 외/최용 옮김, 위키북스) - 2018.02
하둡과 스파크를 활용한 실용 데이터과학(오퍼 멘델리비치 외/이춘오 옮김) - 2017.08
제대로 알고 쓰는 R 통계분석(이윤환 지음, 한빛아카데미) - 2016.08
쉽게 배우는 R 데이터 분석(김영우 지음, 이지스퍼블리싱) - 2017.09
머신러닝 워크북(제이슨 벨 지음/곽승주 옮김) - 2016.04
파이썬을 이용한 머신러닝, 딥러닝 실전 개발 입문(쿠지라 히코우즈쿠에 지음/윤인성 옮김) - 2017.10

참고 사이트

https://www.apache.org
https://www.python.org/
https://www.cloudera.com/
https://flume.apache.org/
https://kafka.apache.org/
https://sqoop.apache.org/
https://www.mongodb.com/
https://robomongo.org/
http://hadoop.apache.org/
https://hbase.apache.org/
https://hive.apache.org/
https://spark.apache.org/
https://www.r-project.org/
https://rstudio.com/
https://www.tensorflow.org/
https://keras.io/