

Hadoop Eco System과

Funda 카드거래액 데이터를 활용한

카드거래액 예측모델링 및 실시간 대시보드 구축 프로젝트



PL / GyuBeom Park

Front-End / Back-End / Data Analysis



PL / JunSoo Ahn

Data Architecture / Data Science

- 1. Introduction
- 2. Architecture
- 3. Collection
- 4. Storage
- 5. Analysis
- 6. Visualization
- 7. Conclusion

Contents

1. Introduction
2. Architecture
3. Collection
4. Storage
5. Analysis
6. Visualization
7. Conclusion

Contents

1. Introduction
2. Architecture
3. Collection
4. Storage
5. Analysis
6. Visualization
7. Conclusion

1.1 Description



- 핀테크 기업인 'FUNDA(펀다)'는 신용 점수가 낮거나 담보를 가지지 못하는 우수 상점들에게 금융 기회를 제공하려 함
- 이를 위해서는 실시간으로 대출 현황을 모니터링할 수 있는 시스템과 합리적으로 대출을 집행할 수 있는 예측모델이 필요함
- 이에 우리는 하둡 에코시스템을 활용하여 실시간 대출현황 대시보드를 구축함과 동시에 AR Model(자기회귀 모델)을 활용하여 저신용 소상공인들의 각 상점별 상환기간의 카드매출을 예측하는 모델을 구현함
- 2016년 06월부터 2019년 2월까지의 카드 거래 데이터를 활용하여, 상점의 고유 아이디(store_id) 및 연도와 월이 주어졌을 때, 각 상점별로 향후 3개월간 발생할 총 매출(amount의 합계)을 예측하는 방식으로 모델 구현

1.2 Data Field

FUNDA

Funda_train.csv

- 상점별 카드 매출 내역 데이터

키움증권 

키움증권 API

- 코스피, 코스닥 시세 데이터

❖ funda_train.csv (500 MB)

2016년 06월 ~ 2019년 02월까지의 상점별 카드 매출 내역

- 1) store_id : 상점의 고유 아이디
- 2) card_id : 사용한 카드의 고유 아이디
- 3) card_company : 비식별화된 카드 회사
- 4) transacted_date : 거래 날짜
- 5) transacted_time : 거래 시간(시:분)
- 6) installment_term : 할부 개월 수(포인트 사용 시 (60개월 + 실제할부개월)을 할부개월수에 기재한다.)
- 7) region : 상점의 지역
- 8) type_of_business : 상점의 업종
- 9) amount : 거래액 (단위 : 십원)

1. Introduction

2. Architecture

3. Collection

4. Storage

5. Analysis

6. Visualization

7. Conclusion

Contents

1. Introduction
- 2. Architecture**
3. Collection
4. Storage
5. Analysis
6. Visualization
7. Conclusion

1. Introduction

2. Architecture

3. Collection

4. Storage

5. Analysis

6. Visualization

7. Conclusion

2.1 Hadoop Eco System

수집



저장



분석



시각화



2.2 Data Architecture

1. Introduction

2. Architecture

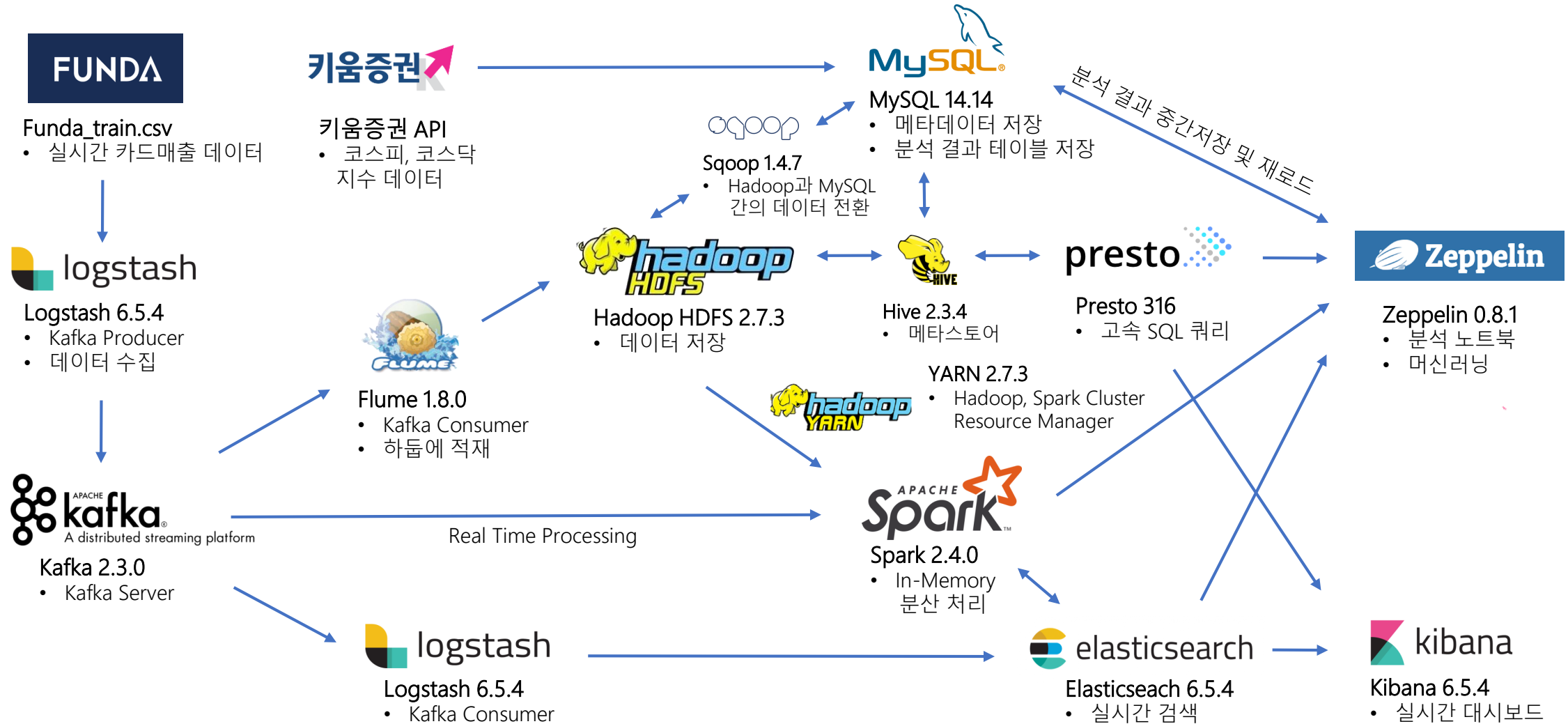
3. Collection

4. Storage

5. Analysis

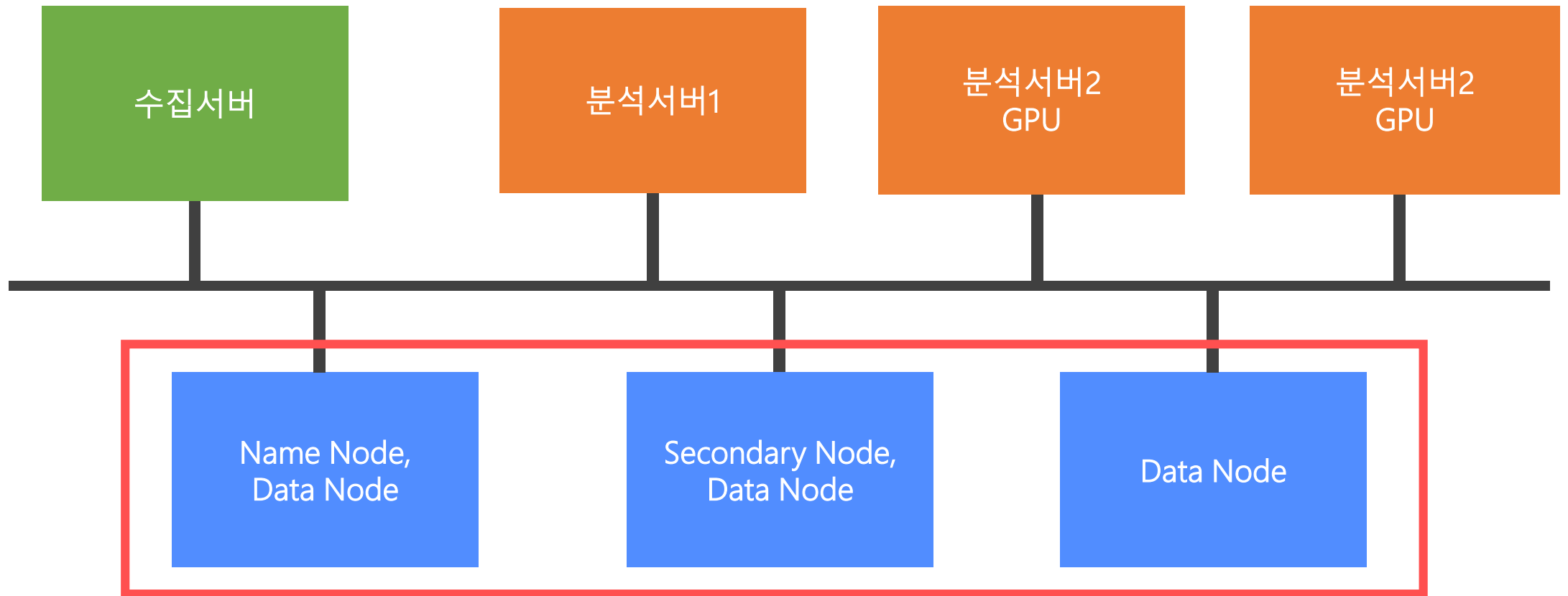
6. Visualization

7. Conclusion



- 1. Introduction
- 2. Architecture
- 3. Collection
- 4. Storage
- 5. Analysis
- 6. Visualization
- 7. Conclusion

2.3 Development System



Hadoop & Spark Yarn Cluster

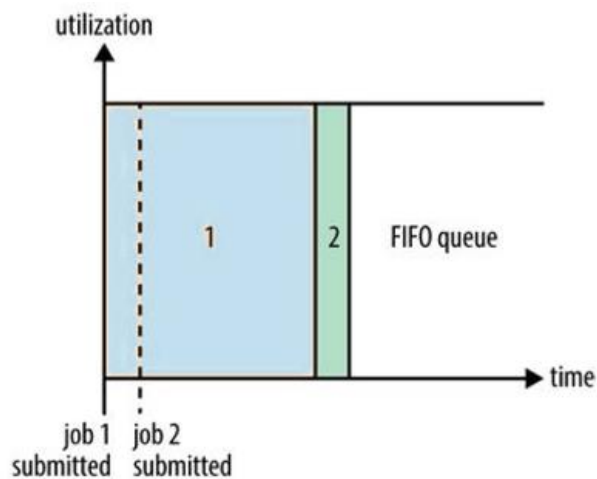
2.4 Resource Management: Yarn

- 하둡 에코시스템은 운영과 분석 Application이 Chaotic하게 거미줄처럼 연결되어 있을 뿐 아니라, 동시작업이 빈번하게 발생함, 특히 Spark는 In-Memory 시스템으로서 Memory, CPU를 매우 많이 점유하고 있기 때문에 자원관리가 중요
- 시간과 자원을 많이 점유하는 분석관련 Job들로 인해 Cluster내에 병목현상 발생
- Fair Scheduler를 통해 Job들 간의 Task를 Round robin 방식으로 할당하여 모든 Job들이 동시에 처리될 수 있도록 함

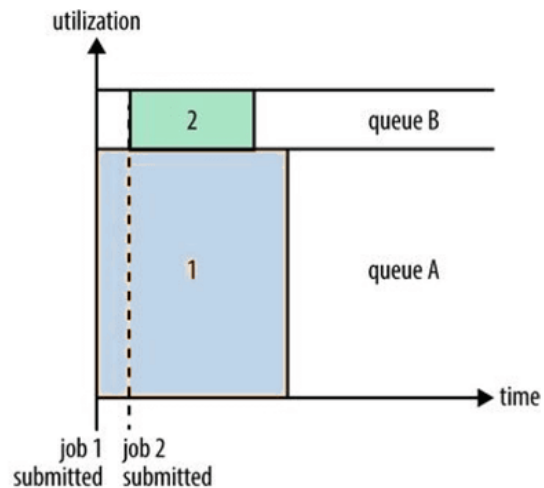
[yarn-site.xml](#)

```
<property>
  <name>yarn.resourcemanager.scheduler.class</name>
  <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value>
</property>
```

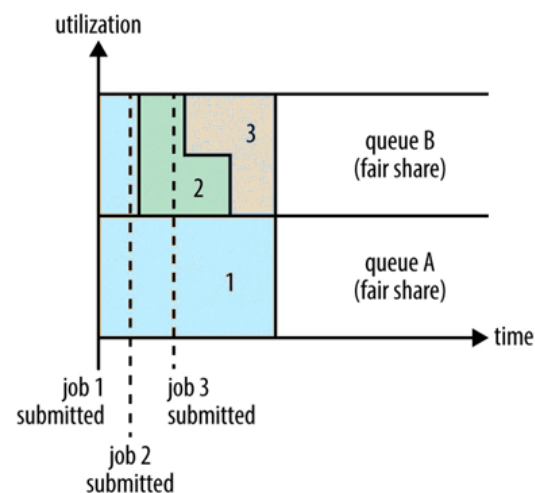
❖ FIFO Scheduler



❖ Capacity Scheduler



❖ Fair Scheduler



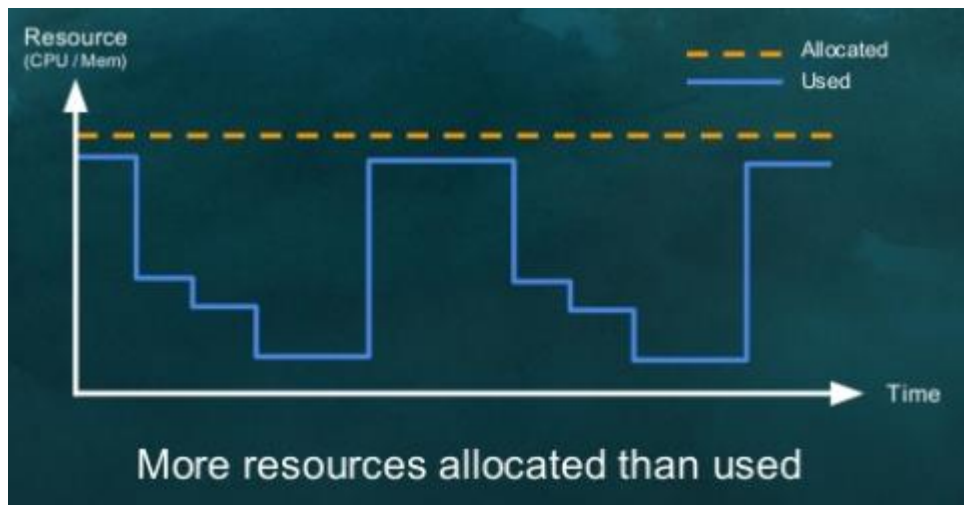
2.4 Resource Management: Yarn

- Cluster 내에 여러 Application이 존재할 때 자원이 효율적으로 사용되지 않는 문제가 발생
- Dynamic Resource Allocation 방식으로 문제 해결

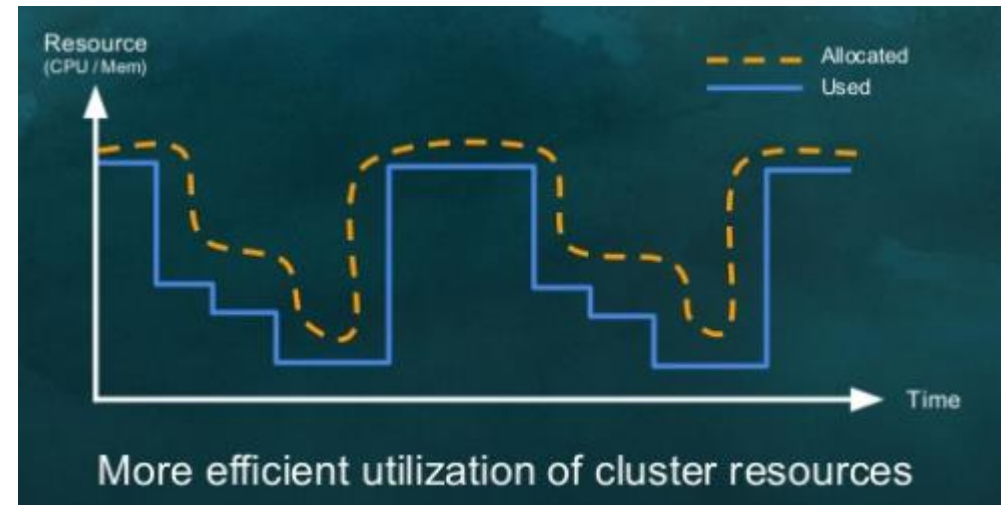
spark-default.conf

```
spark.shuffle.service.enabled true  
spark.dynamicAllocation.enabled true  
spark.dynamicAllocation.schedulerBacklogTimeout 3s  
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout 3s
```

❖ Static Allocation



❖ Dynamic Allocation



1. Introduction

2. Architecture

3. Collection

4. Storage

5. Analysis

6. Visualization

7. Conclusion

Contents

1. Introduction
2. Architecture
- 3. Collection**
4. Storage
5. Analysis
6. Visualization
7. Conclusion

3.1 Collection Flow

1. Introduction

2. Architecture

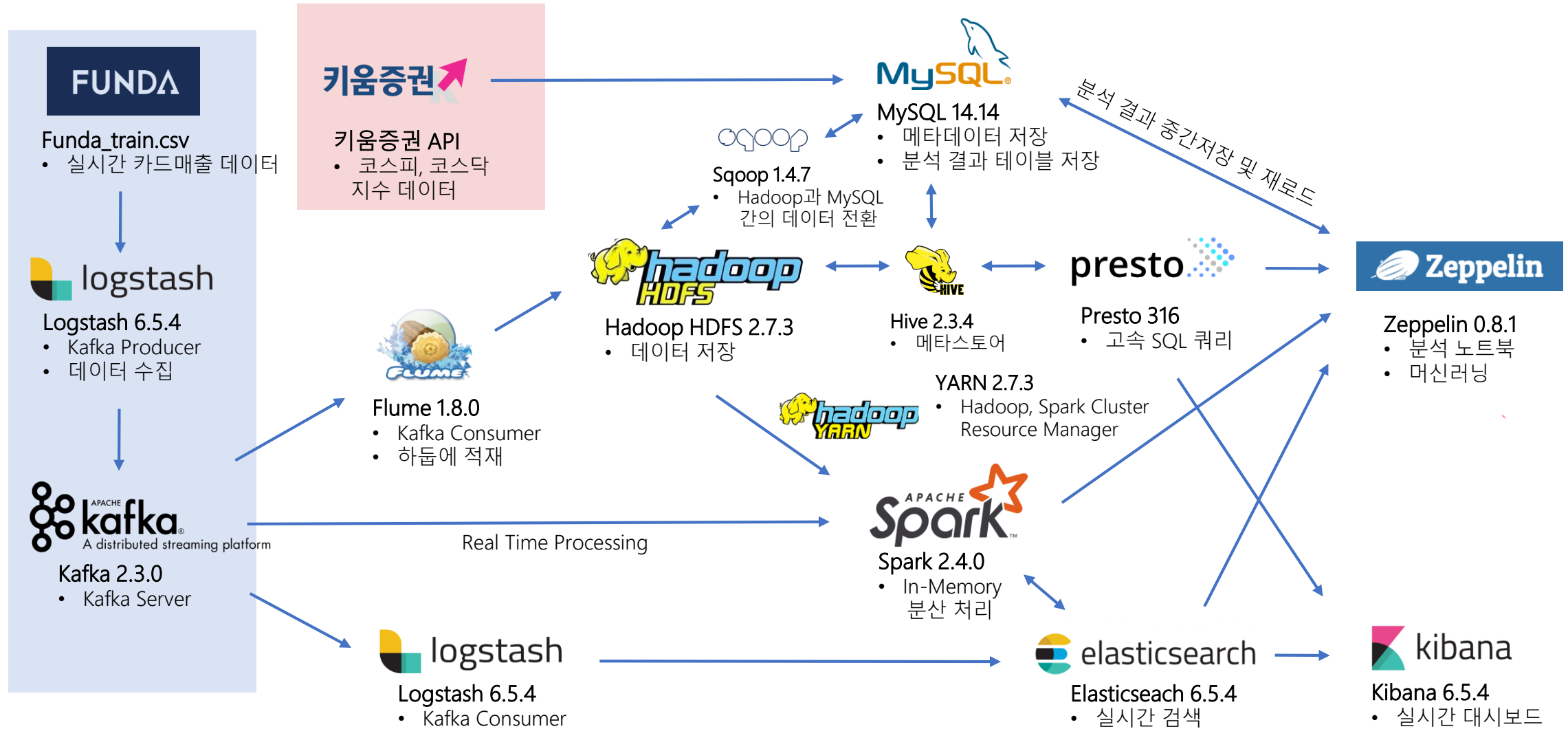
3. Collection

4. Storage

5. Analysis

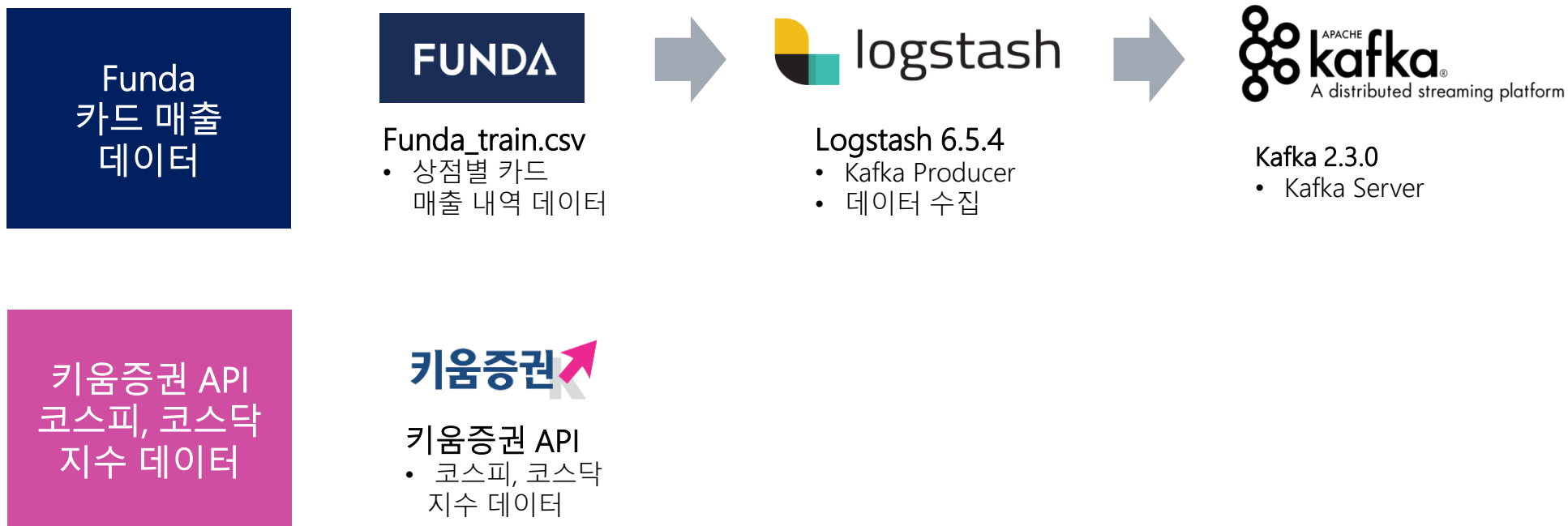
6. Visualization

7. Conclusion



3.1 Collection by Logstash & Kafka

- Funda 데이터는 비주기적이며 대량으로 발생하기 때문에 데이터 유실의 우려가 있음
- 이를 방지하기 위해 Kafka를 Buffer로 활용, Kafka Producer 역할을 하는 Logstash를 통해 Kafka Server에 중간 저장
- 키움증권 API는 데이터가 주기적으로 발생하고 데이터가 크지 않기 때문에 유실의 우려가 없음



3.1 Collection by Logstash & Kafka

❖ Logstash Code (to kafka)

```
input {
  file {
    path => "/home/a/datamart/funda_train.csv"
    start_position => "beginning"
    sincedb_path => "nul"
  }
}
filter {
  csv {
    separator => ","
    skip_header => true
    Columns => ["store_id", "card_id", "card_company", "transacted_date", "transacted_time", "installment_term", "region", "type_of_business", "amount"]
  }
}
mutate {
  add_field => {
    "timestamp" => "%{transacted_date} %{transacted_time}"
  }
}
```

```
date {
  match => ["timestamp", "yyyy-MM-dd HH:mm:ss", "yyyy-MM-dd HH:mm", "MMM dd yyyy HH:mm:ss", "MMM d yyyy HH:mm:ss", "ISO8601"]
  timezone => "UTC"
  locale => "ko"
  target => "@timestamp"
}
mutate{convert=>["store_id","string"]}
mutate{convert=>["card_id","string"]}
mutate{convert=>["card_company","string"]}
mutate{convert=>["installment_term","float"]}
mutate{convert=>["region","string"]}
mutate{convert=>["type_of_business","string"]}
mutate{convert=>["amount","float"]}
}

output {
  kafka {
    bootstrap_servers => "hd2.cluster.kr:9092"
    acks => "1"
    topic_id => "funda"
    codec => json
  }
}
```

1. Introduction

2. Architecture

3. Collection

4. Storage

5. Analysis

6. Visualization

7. Conclusion

Contents

1. Introduction
2. Architecture
3. Collection
- 4. Storage**
5. Analysis
6. Visualization
7. Conclusion

4.1 Storage Flow

1. Introduction

2. Architecture

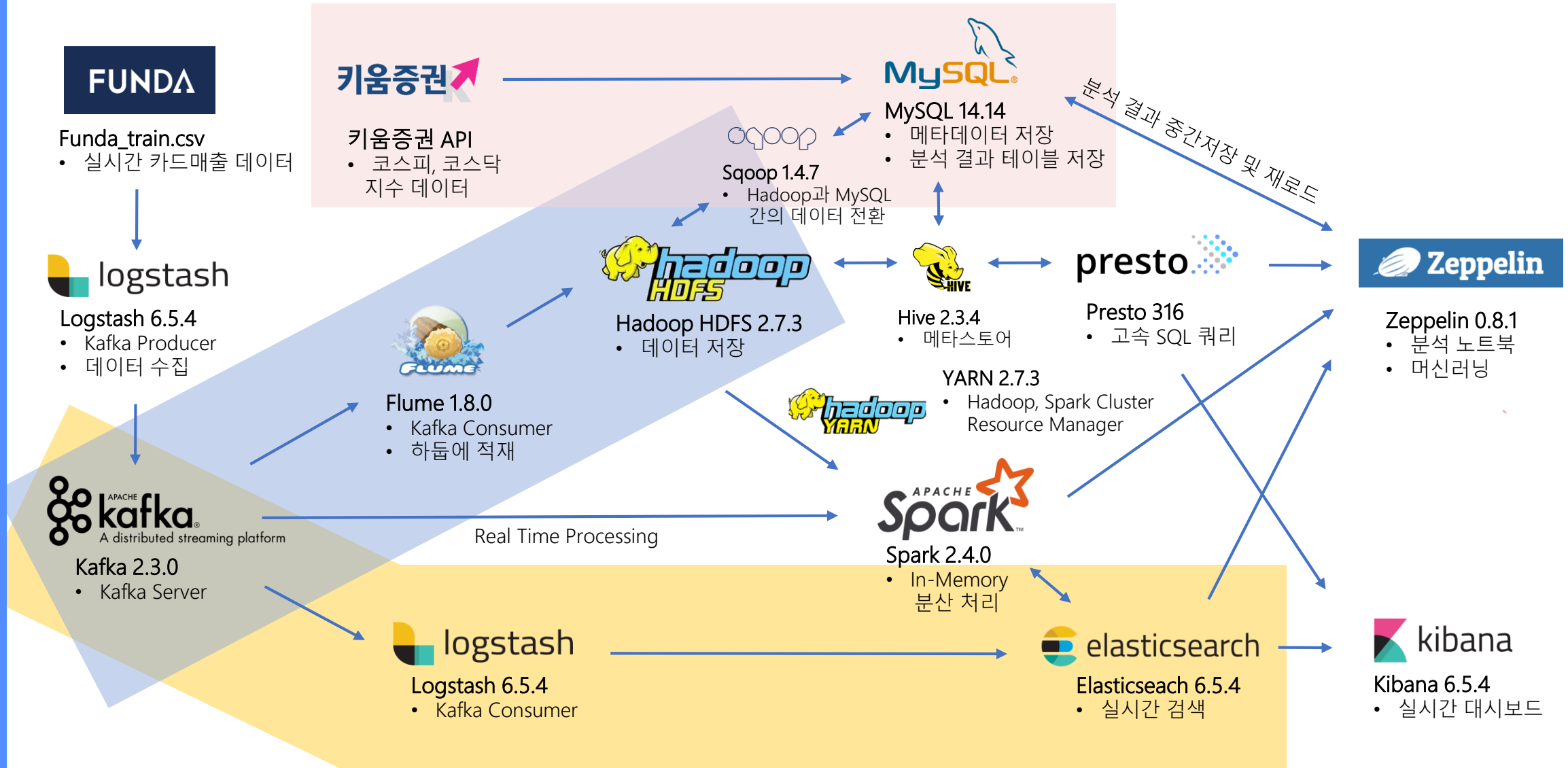
3. Collection

4. Storage

5. Analysis

6. Visualization

7. Conclusion



4.1 Storage for Real-time Search

- Real-time Search와 대시보드 구성을 위해 1~3달 치의 Short-term 데이터를 elasticsearch에 저장



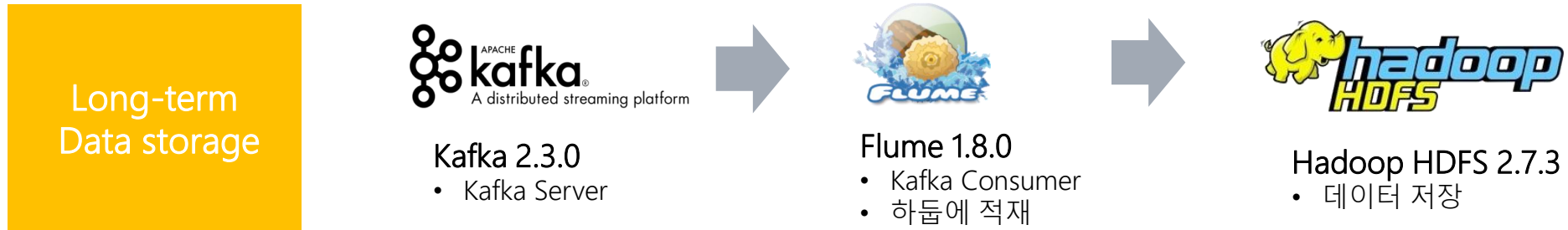
❖ Logstash Code (to elastic)

```
input {
  kafka {
    bootstrap_servers => "hd2.cluster.kr:9092"
    topics => ["funda"]
    consumer_threads => 1
    decorate_events => true
    codec => json
  }
}
```

```
output {
  elasticsearch {
    hosts => "hd4.cluster.kr:9200"
    index => "funda-%{+YYYYMM}"
  }
  stdout {
    codec => rubydebug
  }
}
```

4.2 Long-term Data Storage

- Long-term 데이터는 하둡에 일간 단위로 파티션을 나누어 json format으로 저장



❖ Flume Code

```

agent1.sources.kafka-source.type
= org.apache.flume.source.kafka.KafkaSource
agent1.sources.kafka-source.zookeeperConnect
= hd2.cluster.kr:2181
agent1.sources.kafka-source.topic = funda
agent1.sources.kafka-source.groupId = flume
agent1.sources.kafka-source.channels = memory-channel
agent1.sources.kafka-source.interceptors = i1
agent1.sources.kafka-source.interceptors.i1.type = timestamp
agent1.sources.kafka-source.kafka.consumer.timeout.ms = 100
  
```

```

agent1.channels.memory-channel.type = memory
agent1.channels.memory-channel.capacity = 10000
agent1.channels.memory-channel.transactionCapacity = 1000
  
```

```

agent1.sinks.hdfs-sink.type = hdfs
agent1.sinks.hdfs-sink.hdfs.path
= hdfs://hd4.cluster.kr:8020/tmp/kafka/{topic}/{y-m-d}
agent1.sinks.hdfs-sink.hdfs.rollInterval = 5
agent1.sinks.hdfs-sink.hdfs.rollSize = 0
agent1.sinks.hdfs-sink.hdfs.rollCount = 0
agent1.sinks.hdfs-sink.hdfs.fileType = DataStream
agent1.sinks.hdfs-sink.channel = memory-channel
  
```

```

agent1.sources = kafka-source
agent1.channels = memory-channel
agent1.sinks = hdfs-sink
  
```

4.3 Meta Data Storage

- Hive의 Metastore를 통해 하둡 내 데이터의 스키마 관리
- Metastore의 원격저장소로 MySQL를 사용함



Hadoop HDFS 2.7.3
• 데이터 저장



Hive 2.3.4
• 메타스토어



MySQL 14.14
• 메타데이터 저장

❖ Hive Schema Table Code

```
CREATE EXTERNAL TABLE IF NOT EXISTS funda (
  `store_id` STRING,
  `card_id` STRING,
  `card_company` STRING,
  `transacted_date` STRING,
  `transacted_time` STRING,
  `installment_term` STRING,
  `region` STRING,
  `type_of_business` STRING,
  `amount` STRING
)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
with serdeproperties ( 'paths'='store_id, card_id, card_company, transacted_date, transacted_time, installment_term, region, type_of_business, amount' )
LOCATION 'hdfs://hd4.cluster.kr:8020/tmp/kafka/funda'
```

1. Introduction

2. Architecture

3. Collection

4. Storage

5. Anaysis

6. Visualization

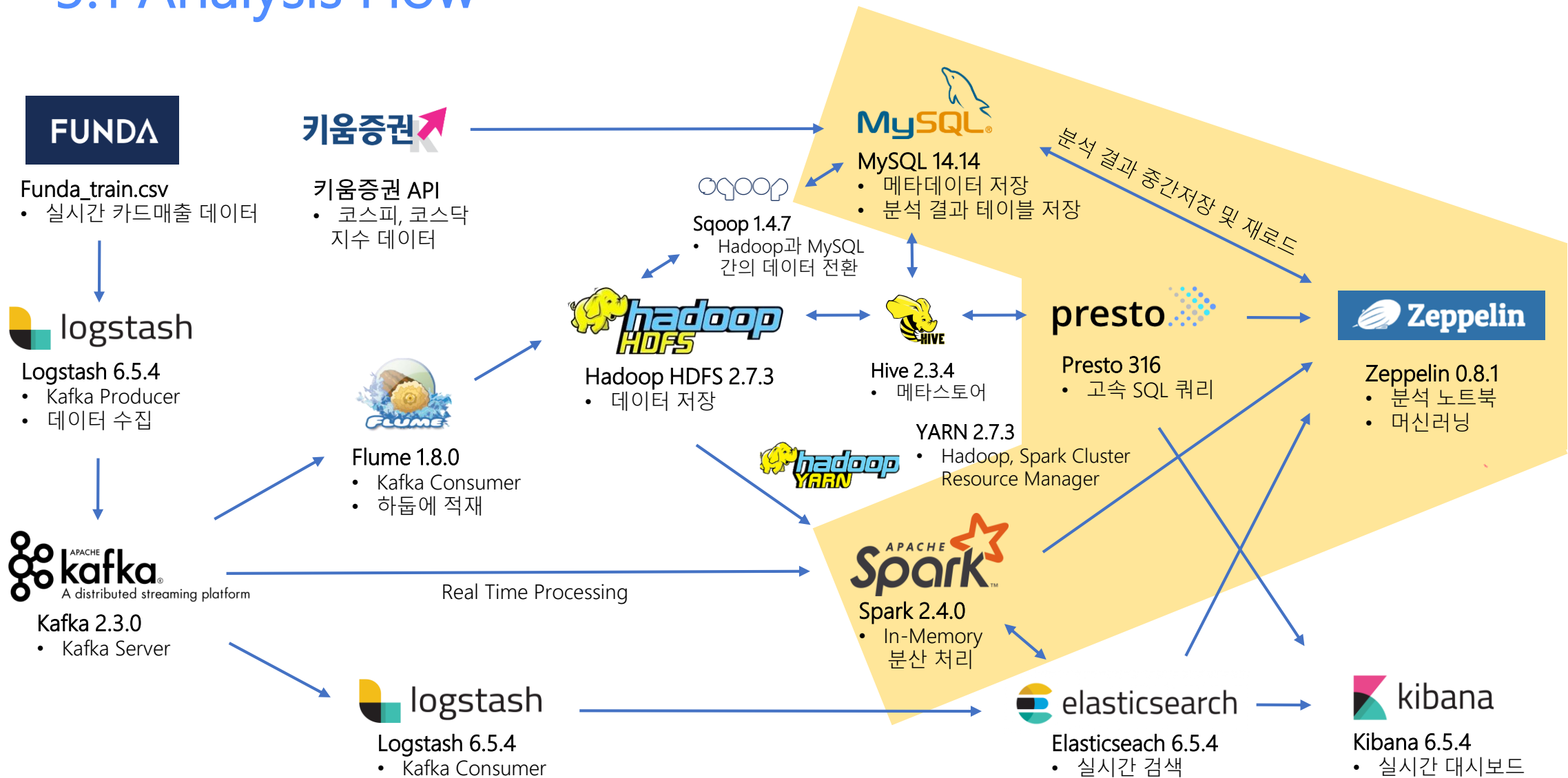
7. Conclusion

Contents

1. Introduction
2. Architecture
3. Collection
4. Storage
- 5. Analysis**
6. Visualization
7. Conclusion

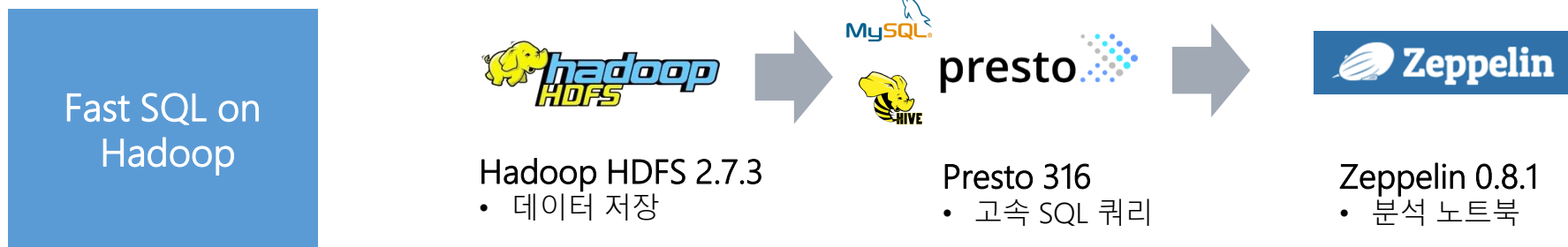
5.1 Analysis Flow

- 1. Introduction
- 2. Architecture
- 3. Collection
- 4. Storage
- 5. Analysis
- 6. Visualization
- 7. Conclusion

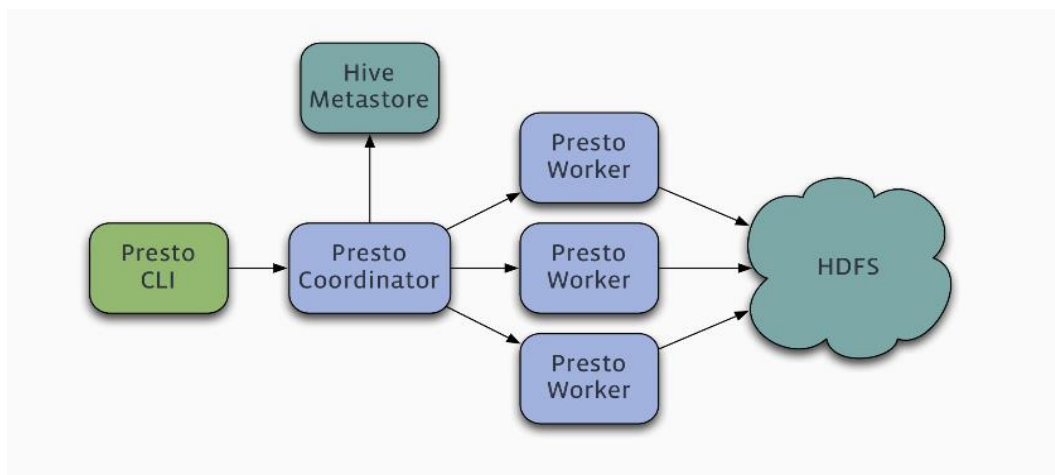


5.2 Analysis Tool – 1) Presto

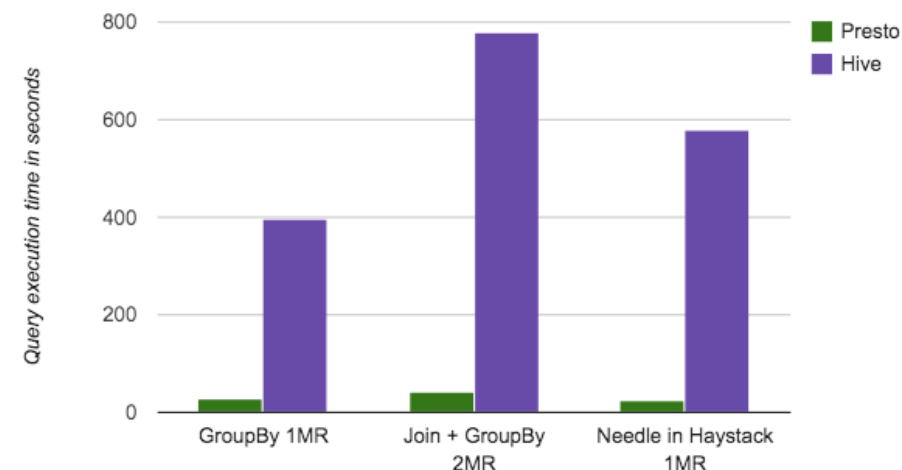
- Hive가 Mapreduce를 이용하는 반면에, Presto는 분산된 Query Engine을 통해 데이터에 access하고 이를 In-memory로 연산, Hive 대비 압도적으로 빠른 SQL On Hadoop



❖ Presto Architecture



❖ Presto vs Hive Performance



5.3 Data Exporation by Presto

- Hive metastore를 통해 Hadoop 데이터에 직접 접근 가능

Zeppelin Notebook Job

Funda/Presto

```
select * from hive.default.funda
```

store_id	card_id	card_company	transacted_date	transacted_time	installment_term	region	type_of_business	amount
127	301236	c	2018-03-31	15:08	0	null	null	1571
127	300506	c	2018-03-31	18:07	0	null	null	1714
127	301570	a	2018-04-01	10:25	0	null	null	8357
127	300818	c	2018-04-01	10:57	0	null	null	6857
127	301529	b	2018-04-01	14:13	0	null	null	10000
127	301570	a	2018-04-01	16:18	0	null	null	9285
127	301571	b	2018-04-01	16:32	0	null	null	6000
127	299584	b	2018-04-01	16:56	0	null	null	3857
127	301572	b	2018-04-01	17:31	0	null	null	8285
127	301076	b	2018-04-01	18:08	0	null	null	4285
127	301483	e	2018-04-01	18:12	0	null	null	1714
127	300909	g	2018-04-01	18:42	0	null	null	2000
127	300089	a	2018-04-01	18:55	0	null	null	2285

1. Introduction

2. Architecture

3. Collection

4. Storage

5. Anaysis

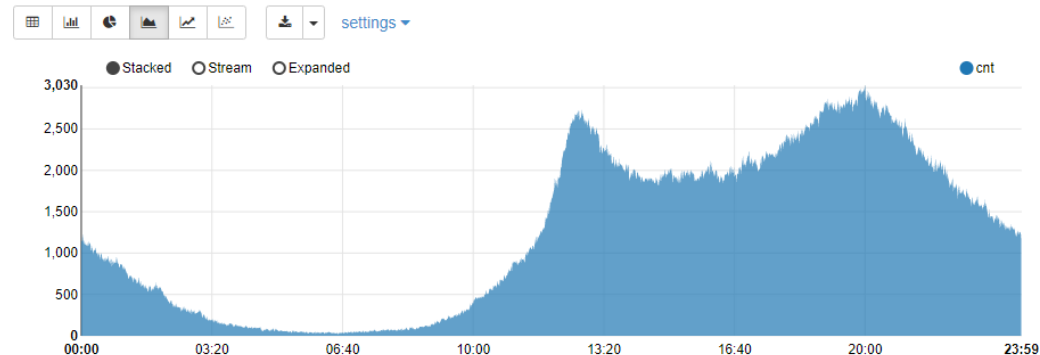
6. Visualization

7. Conclusion

5.3 Data Exporation by Presto

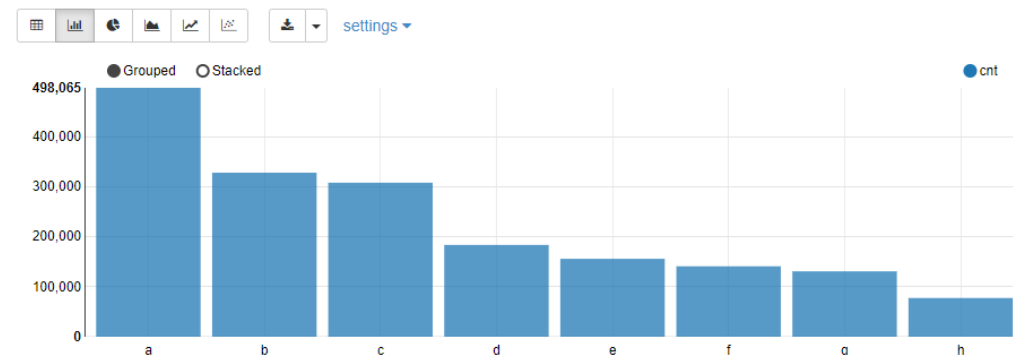
시간대별 카드결제 빈도수 추이

```
select transacted_time, count(*) as cnt from hive.default.funda
group by transacted_time
order by transacted_time
```



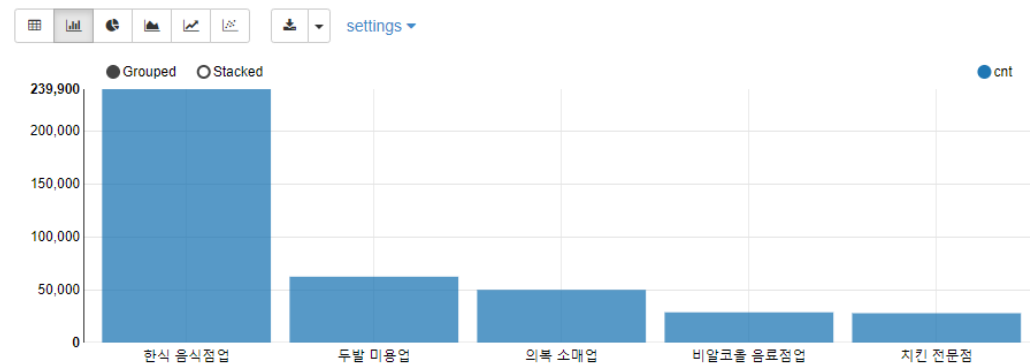
카드사별 카드결제 빈도수 순위

```
select card_company, count(*) as cnt from hive.default.funda
where card_company != ''
group by card_company
order by cnt desc
```



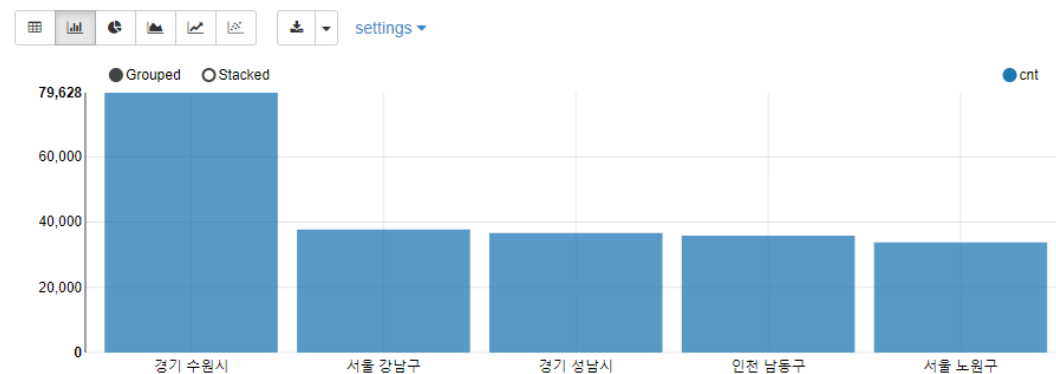
업종별 카드결제 빈도수 순위

```
select type_of_business, count(*) as cnt from hive.default.funda
where type_of_business != ''
group by type_of_business
order by cnt desc
limit 5
```



지역별 카드결제 빈도수 순위

```
select region, count(*) as cnt from hive.default.funda
where region != ''
group by region
order by cnt desc
limit 5
```



5.4 Analysis Tool – 2) Spark

- 하둡 MapReduce 보다 발전된 새로운 분산병렬처리 Framework
- 저장소는 로컬파일시스템, 하둡 HDFS, NoSQL(Hbase, Redis), RDBMS(오라클,MSSQL)
- Spark는 분산병렬처리 엔진으로서, 기존 Hive, Pig, Mahout, R, Storm 등을 모두 대체 가능

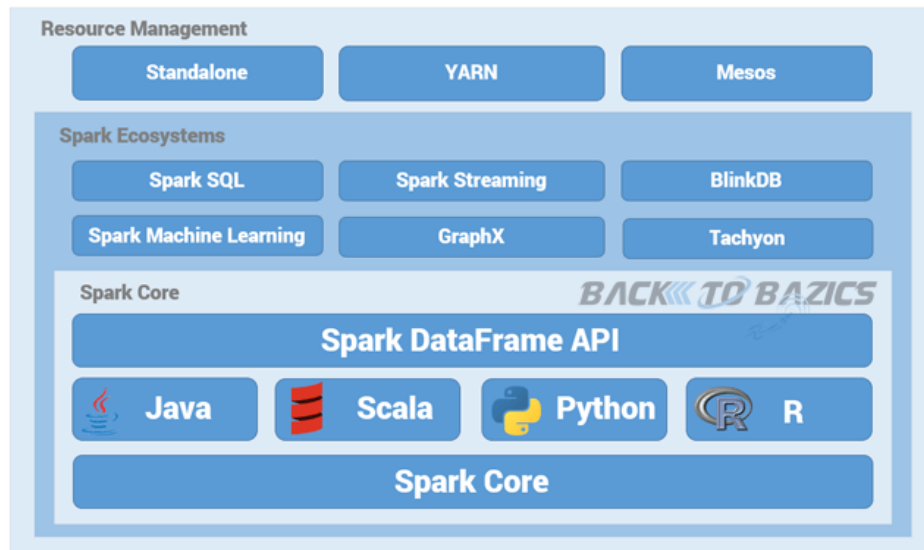
In-Memory
Distributed
Processing



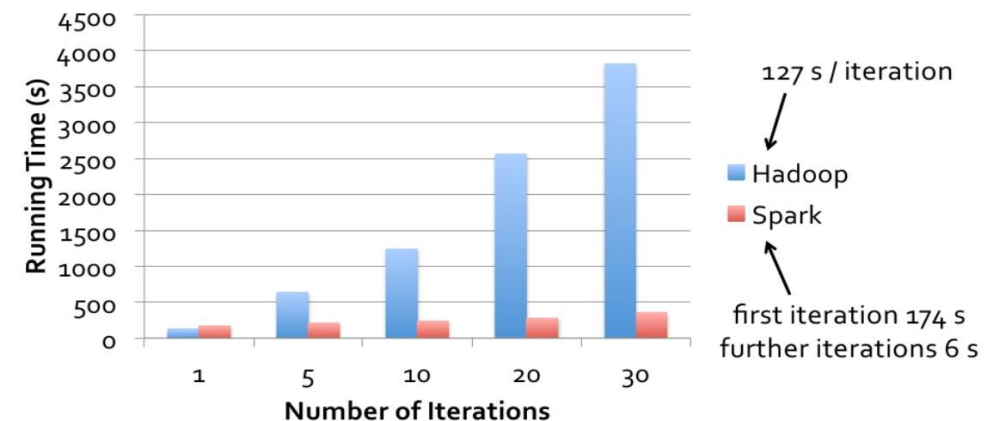
Spark 2.4.0
• In-Memory
분산 처리



Zeppelin 0.8.1
• 분석 노트북



❖ Spark vs Hadoop Performance



5.5 Pre-Processing by Pyspark

- 하둡으로 부터 Funda 테이블과 지역 위도경도 테이블을 로드
- 지역 위도경도 테이블은 elasticsearch의 geo_type에 맞게 전처리

Funda 데이터 로드

```
%pyspark
sc = spark.sparkContext
path = "hdfs://hd4.cluster.kr:8020/tmp/kafka/funda/19-08-19/**"
funda = spark.read.json(path)
funda.printSchema()
funda.createOrReplaceTempView("funda")

root
|-- @timestamp: string (nullable = true)
|-- @version: string (nullable = true)
|-- amount: long (nullable = true)
|-- card_company: string (nullable = true)
|-- card_id: string (nullable = true)
|-- host: string (nullable = true)
|-- installment_term: long (nullable = true)
|-- message: string (nullable = true)
|-- path: string (nullable = true)
|-- region: string (nullable = true)
|-- store_id: string (nullable = true)
|-- timestamp: string (nullable = true)
|-- transacted_date: string (nullable = true)
|-- transacted_time: string (nullable = true)
|-- type_of_business: string (nullable = true)
```

지역별 위도경도 테이블 전처리

```
%pyspark
from pyspark.sql.functions import split, explode, concat, col, column, lit, expr
from pyspark.sql.functions import *

location = spark.read.load("hdfs://hd4.cluster.kr:8020/datamart/detail_simple_coordinate.csv", format="csv", sep="\n", inferSchema="false", header="true")
location.toDF()
location = location.withColumnRenamed("region,detail_coordinate_array,simple_coordinate_array", "loc")
location = location.withColumn('loc', regexp_replace('loc', ',', ''))
location = location.withColumn('loc', regexp_replace('loc', 'lat', 'lat'))
location = location.withColumn('loc', regexp_replace('loc', 'lon', 'lon'))
location = location.withColumn("region", split(col("loc"), ",")[0])
location = location.withColumn("detail_loc", concat(split(col("loc"), ",")[1], lit(","), split(col("loc"), ",")[2]))
location = location.withColumn("simple_loc", concat(split(col("loc"), ",")[3], lit(","), split(col("loc"), ",")[4]))
location = location.drop(location.loc)
location.createOrReplaceTempView("location")
z.show(location)
```

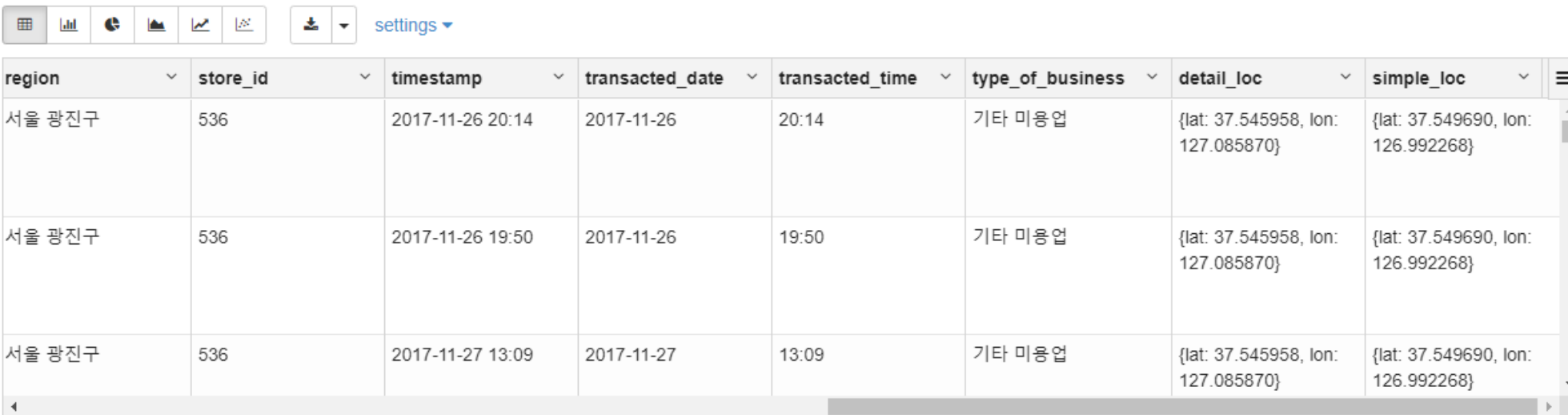
region	detail_loc	simple_loc
강원 횡성군	{lat: 37.510960, lon: 128.075948}	{lat: 37.817690, lon: 128.153659}
경기 가평군	{lat: 37.819852, lon: 127.450592}	{lat: 37.406997, lon: 127.500183}
경기 고양시	{lat: 37.664849, lon: 126.837306}	{lat: 37.406997, lon: 127.500183}
경기 과천시	{lat: 37.433927, lon: 127.002290}	{lat: 37.406997, lon: 127.500183}
경기 광명시	{lat: 37.445184, lon: 126.864709}	{lat: 37.406997, lon: 127.500183}
경기 광주시	{lat: 37.403717, lon: 127.300948}	{lat: 37.406997, lon: 127.500183}
경기 구리시	{lat: 37.599468, lon: 127.131356}	{lat: 37.406997, lon: 127.500183}
경기 구포시	{lat: 37.343615, lon: 126.920728}	{lat: 37.406997, lon: 127.500183}

5.5 Pre-Processing by Pyspark

- funda 테이블과 지역 위도경도 테이블을 조인하여 elasticsearch에 저장
- Zeppelin의 Cron 기능을 활용하여 하루에 한번 배치 처리

Funda 테이블과 지역 위도경도 테이블 조인

```
%pyspark
joinExpression = funda["region"] == location["region"]
joinType = "left_outer"
funda_join = funda.join(location, joinExpression, joinType).drop(location.region)
z.show(funda_join)
```



region	store_id	timestamp	transacted_date	transacted_time	type_of_business	detail_loc	simple_loc
서울 광진구	536	2017-11-26 20:14	2017-11-26	20:14	기타 미용업	{lat: 37.545958, lon: 127.085870}	{lat: 37.549690, lon: 126.992268}
서울 광진구	536	2017-11-26 19:50	2017-11-26	19:50	기타 미용업	{lat: 37.545958, lon: 127.085870}	{lat: 37.549690, lon: 126.992268}
서울 광진구	536	2017-11-27 13:09	2017-11-27	13:09	기타 미용업	{lat: 37.545958, lon: 127.085870}	{lat: 37.549690, lon: 126.992268}

elasticsearch에 저장

```
%pyspark
from pyspark.sql import SQLContext
funda_join.write.format("org.elasticsearch.spark.sql")\
.option("es.resource", "funda_data3/apache").option("es.nodes", "hd4.cluster.kr:9200").save()
```

5.6 Modeling

- AR Model(자기회귀모델)을 통한 상점별 카드 거래액 예측

❖ AR Model Code

The notation $AR(p)$ refers to the autoregressive model of order p . The $AR(p)$ model is written

$$X_t = c + \sum_{i=1}^p \varphi_i X_{t-i} + \varepsilon_t.$$

```
def predict_amount(df, pred_start, pred_end, y='amount', lamd=0.5):
```

```
    df["boxcox"] = sp.stats.boxcox(df[y].apply(lambda x: x if x > 0 else 0.1), 0.5)
    m = sm.tsa.ARMA(df.boxcox, (3, 0))
    r = m.fit()
    s_pred = r.predict(start=pred_start, end=pred_end)
    prediction = (((s_pred * lamd) + 1)**(1 / lamd))
    return prediction
```

2018년 12월 상점별 거래액 예측치 vs 실제치

```
%pyspark
z.show(amount_18_12)
```


[settings](#)

index	pred_18_12	actual_18_12	error_18_12
0.0	703518.2837445489	874571.4285714291	171053.1448268802
1	89864.10938900827	85285.71428571429	-4578.39510329398
2	340525.44258618273	340714.28571428574	188.84312810300617
4	850779.4822379742	923857.1428571433	73077.66061916912
5	370605.90775655024	399571.42857142875	28965.520814878517
6	2303292.1655223714	2230285.714285716	-73006.45123665547
7	345526.2109349267	302414.2857142857	-43111.92522064102
8	1175784.6031245864	1157257.142857144	-18527.46026744251

1. Introduction

2. Architecture

3. Collection

4. Storage

5. Analysis

6. Visualization

7. Conclusion

Contents

1. Introduction
2. Architecture
3. Collection
4. Storage
5. Analysis
- 6. Visualization**
7. Conclusion

6.1 Visualization Flow

1. Introduction

2. Architecture

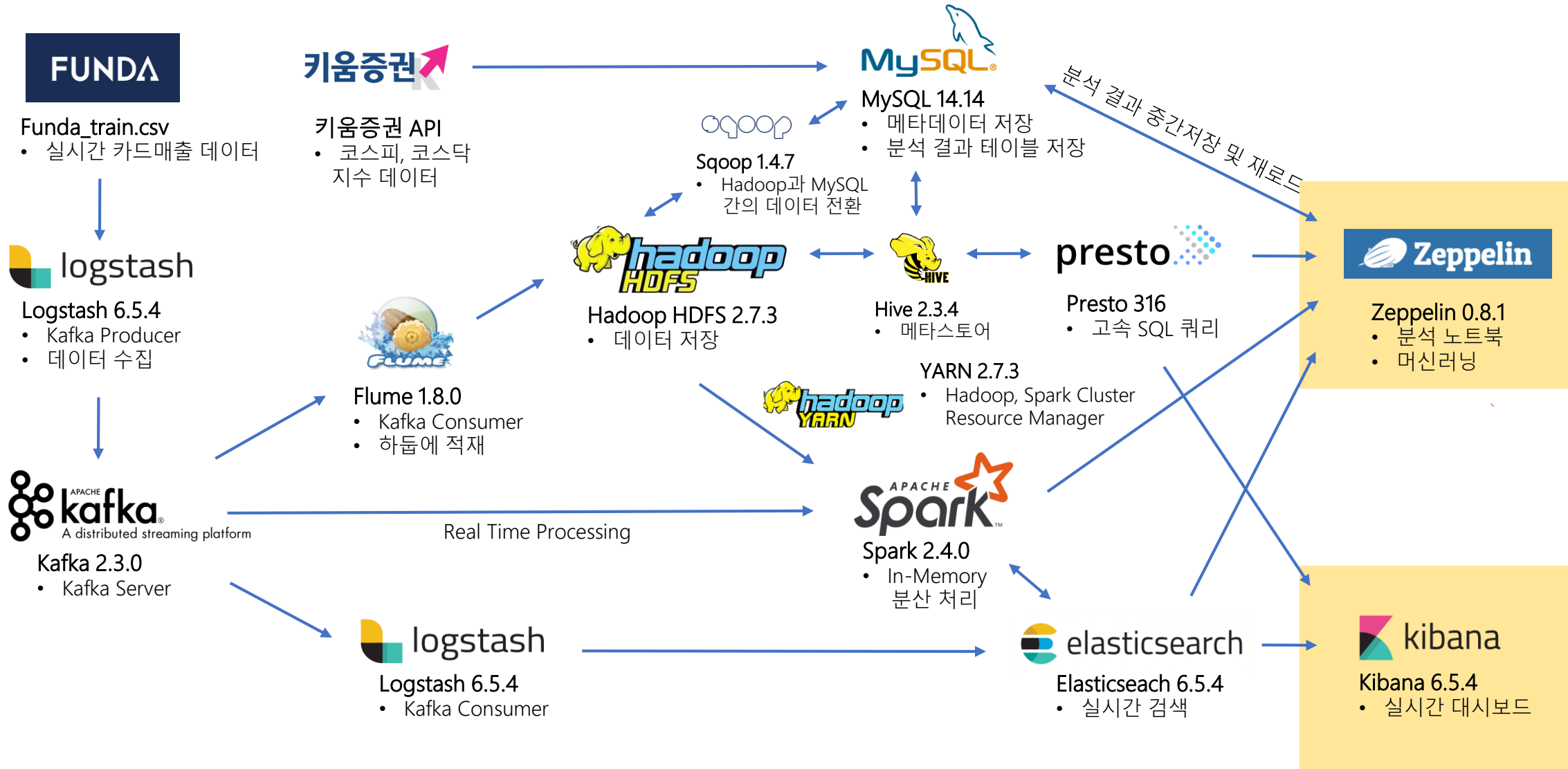
3. Collection

4. Storage

5. Analysis

6. Visualization

7. Conclusion



1. Introduction

2. Architecture

3. Collection

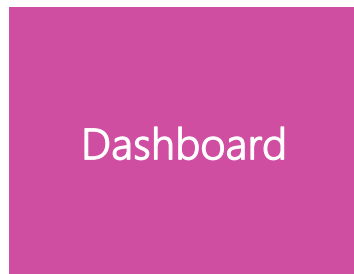
4. Storage

5. Analysis

6. Visualization

7. Conclusion

6.2 Kibana Dashboard



Elasticsearch 6.5.4

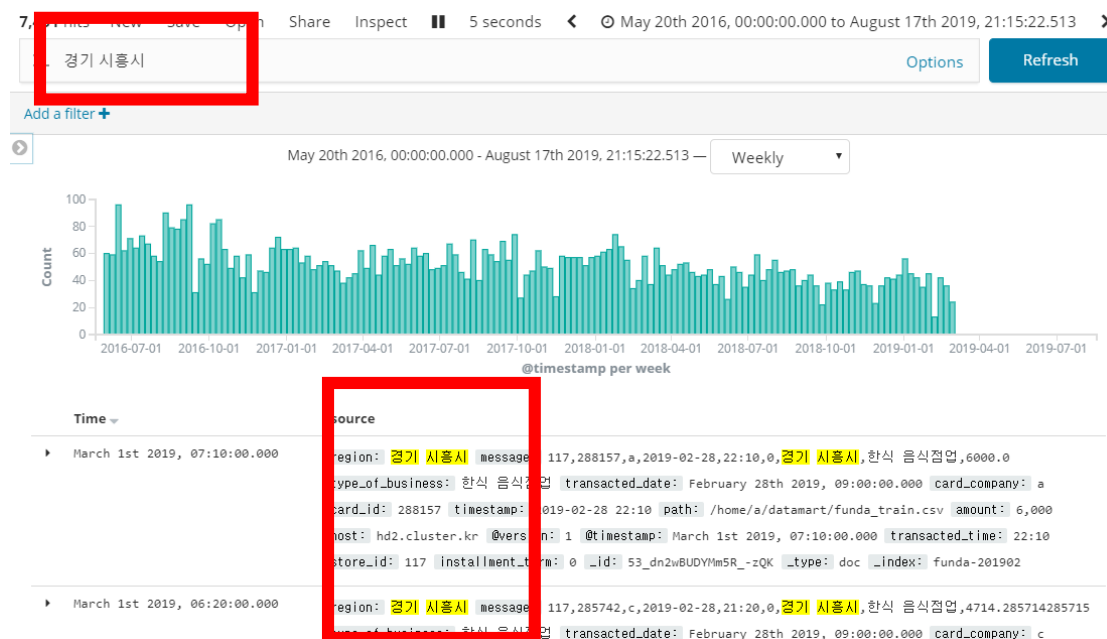
- 실시간 검색



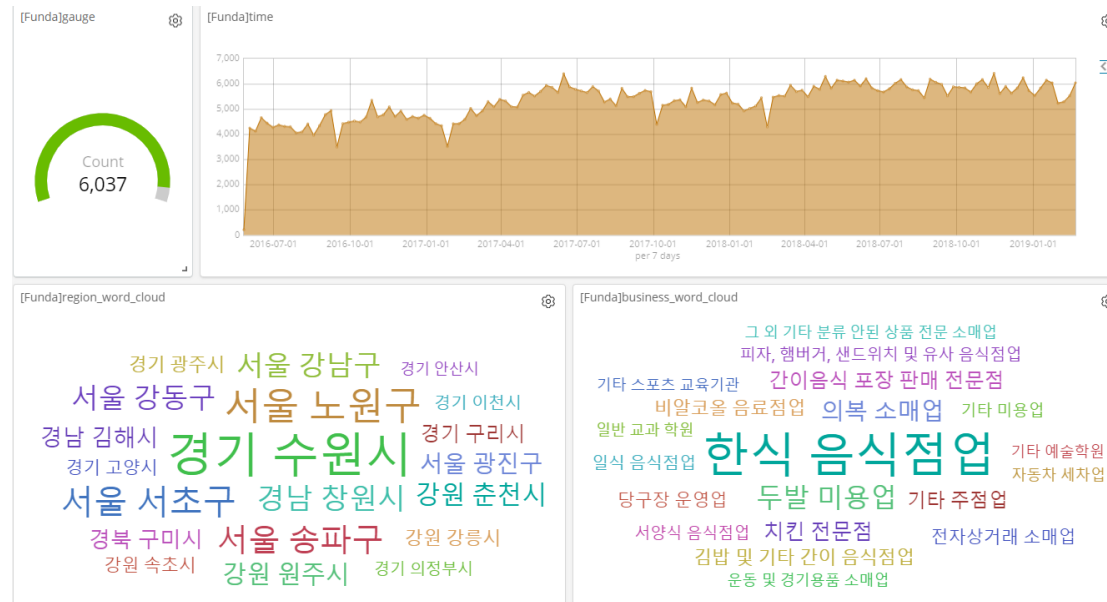
Kibana 6.5.4

- 실시간 대시보드

❖ Elasticsearch '경기 시흥시' 검색



❖ Kibana Dashboard



1. Introduction

2. Architecture

3. Collection

4. Storage

5. Analysis

6. Visualization

7. Conclusion

Contents

1. Introduction
2. Architecture
3. Collection
4. Storage
5. Analysis
6. Visualization
- 7. Conclusion**

7. Conclusion

- AR(Auto Regressive) Model을 활용한 시계열 분석을 통해 카드거래액 예측모델을 구축하고, 예측치와 실제치를 비교 분석을 통해 대출기간 내 카드결제액이 높게 예측되는 상점에게는 대출규모를 확대하고, 카드거래액이 감소할 것으로 예측되는 상점은 대출규모를 축소하여 효율적인 대출시스템이 이뤄질 수 있도록 함
- 또한 실시간 대시보드를 구축하여 월 예측치와 실시간 매출 달성 진도율을 비교하여, 예측치와 실제치가 큰 차이를 보이는 상점들을 중점적으로 관리할 수 있도록 함

2018년 12월 상점별 거래액 예측치 vs 실제치

```
%pyspark  
z.show(amount_18_12.sort_values(["error_18_12"], ascending=False))
```

settings

