

Team 11 ALL_DREAM

Data Pipeline Construction for Data from Steam API

Steam API에서 받은 데이터를 처리하는 데이터 파이프라인 구축

TEAM & MEMBERS

팀 및 팀원 소개

변준호 dag master

leader 김민혁

데이터 파이프라인 설계 및 구축
개발환경 설정 및 전체적으로 서포팅



서용원

Elastic search, Kibana 구성
올라운더

Kafka 등 여러 플랫폼과 클러스터 구축에 참여
다른 팀원 서포팅
발표 자료 제작

이주홍

이시형 Elastic search, Kibana 구성
언더커버 서포터

CONTENTS

목차

- * 데이터 파이프라인 소개
- * 에어플로우 및 개발환경 세팅
- * 프로젝트의 subprojects
 - * 데이터 파이프라인 구축 built with Docker containers
 - * 구축하고자 했던 데이터 파이프라인 아키텍처 (스파크 클러스터와 앱 포함) 1슬
 - * 설치 플랫폼 & 플랫폼 선정 이유 1-2슬
 - * data flow
 - * 키비나 대시보드 (index patterns, stack management etc)
 - * overloading test
 - * 데이터 파이프라인 아키텍처 1슬
 - * 설치 플랫폼 & 플랫폼 선정 이유 1-2슬
 - * data flow
 - * 그라파나 대시보드 결과 (incl. 이미지, 짧은 영상)
 - * data pipeline architecture orchestrated by Kubernetes
 - * (1. 구축하고 싶었던, 2. 성공적으로 구축했던) 데이터 파이프라인 아키텍처 각 1슬라이드(쿠버 & 쿠버 대시보드) 1-2슬
 - * 설치 플랫폼 & 플랫폼 선정 이유 1-2슬
 - * data flow
 - * <삭제> 데이터베이스의 테이블 & 쿼리문 n슬라이드
 - * 한계점, 부족한 점, 개선할 점, 프로젝트가 진행된다면 나아갈 방향 1슬라이드
 - * 개인별 프로젝트 후기 1슬라이드



1

Constructing a Data Pipeline Built with Docker Containers

모든 애플리케이션을 **도커 컨테이너** 위에 띄운 형태의 데이터 파이프라인 구축하기

DATA PIPELINE ARCHITECTURE

데이터 파이프라인 내 애플리케이션과 아키텍처



Docker

- * 현재 국내, 국외 모두 가장 많이 쓰이는 개발 애플리케이션
- * 애플리케이션과 경우에 따라 필요한 데이터를 이미지에 담아 컨테이너화
- * 컨테이너에 장애가 발생할 경우 도커 이미지를 이용해 간단하게 재설치 및 세팅 가능
- * 도커 이미지 하나로 여러 개의 컨테이너를 생성할 수 있음



Kafka

- * 수집된 많은 양의 데이터가 카프카 브로커들의 queue에 배치되어 저장, 처리되기를 기다림
- * 한꺼번에 많은 양의 데이터가 몰려 병목 현상이 발생하는 등 서버가 비정상적으로 작동되는 것을 예방
- * 데이터 특성에 따라 지정된 컨슈머 애플리케이션에 데이터를 전송할 수 있음
- * 이번 프로젝트에서는 크라프트(KRaft)라는 메타데이터 관리 시스템을 구축해서 주키퍼의 역할을 대신함



Prometheus

- * 가장 많이 쓰이는 오픈 소스 카프카 모니터링 애플리케이션
- * 이번 프로젝트에서는 카프카 모니터링 역할
- * 시각화 서비스가 빈약해서 직관적인 전체 파악이 어려움
- * 카프카 익스포터를 따로 설치, 세팅 후 이것을 통해 카프카 클러스터의 실시간 메트릭 데이터를 얻을 수 있음



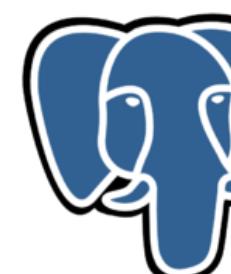
Grafana

- * 가장 많이 쓰이는 오픈 소스 카프카 모니터링 시각화 애플리케이션
- * 프로메테우스를 통해 카프카 모니터링으로 얻은 데이터를 시각화하여 직관적인 모니터링 가능



Airflow

- * 워크플로우 스케줄링에 특화된 오픈 소스 애플리케이션
- * 파이프라인에 이슈가 없다면 DAG을 실행시키는 것만으로 데이터 수집부터 저장까지 자동화 가능
- * 이번 프로젝트에서는 Steam API에 데이터를 요청해서 받고 카프카 프로듀서를 통해 컨슈머(엘라스틱서치와 포스트그레SQL)에 보내는 task를 지정된 주기에 따라 실행하는 역할



PostgreSQL

- * 관계형 데이터베이스 오픈 소스 애플리케이션
- * 이번 프로젝트에서는 엘라스틱서치 외 데이터 백업용 데이터베이스 역할



Elasticsearch

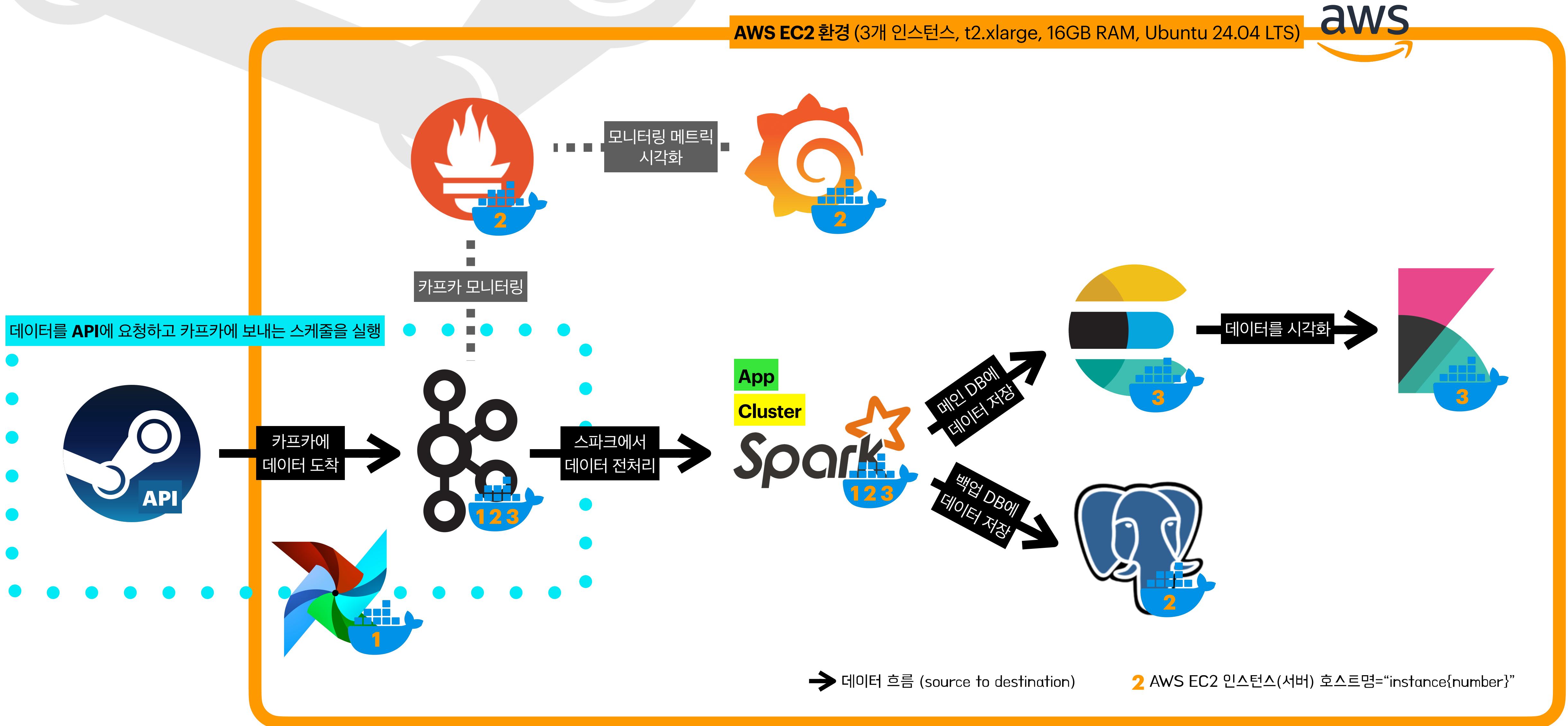
- * 검색 엔진, 데이터베이스 등이 가능한 오픈 소스 애플리케이션
- * 이번 프로젝트에서는 데이터베이스의 역할



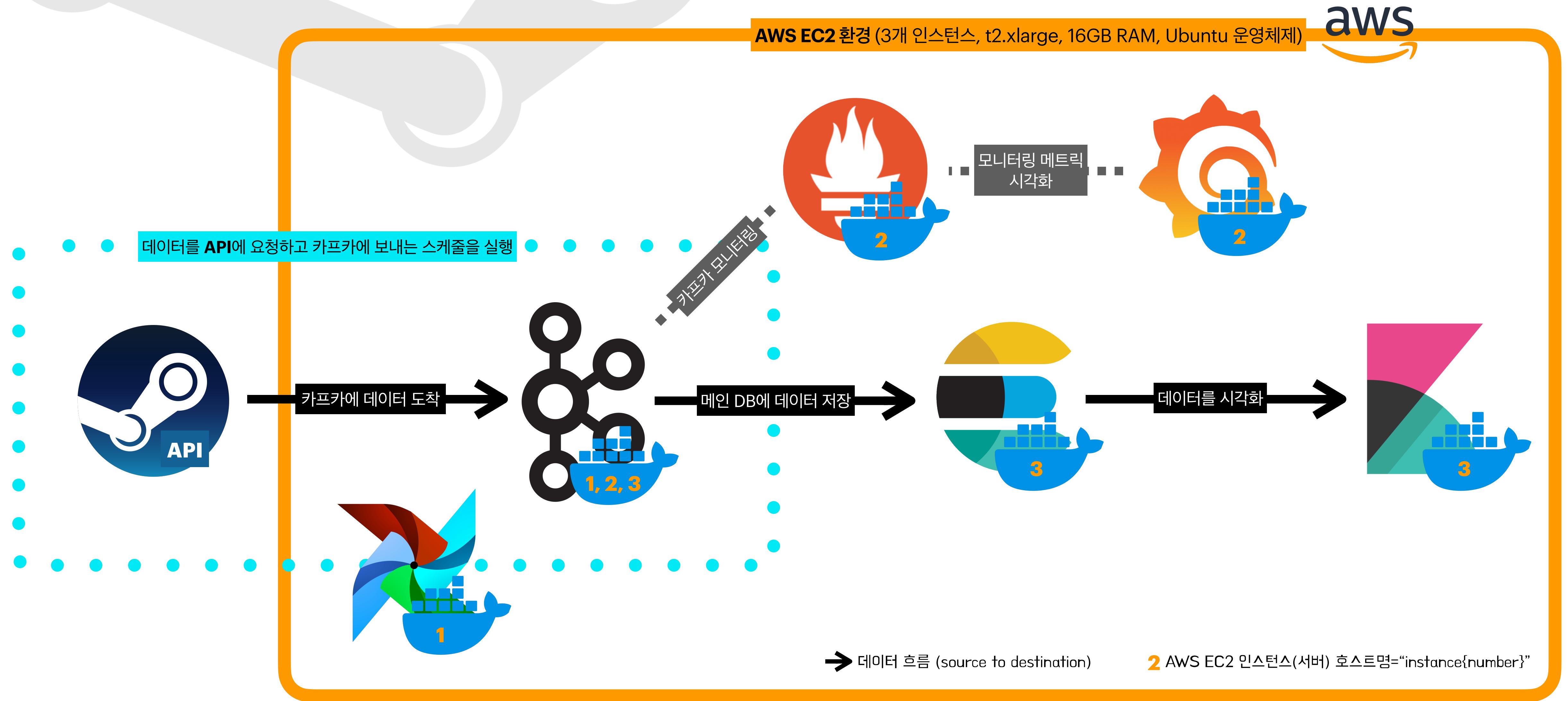
Kibana

- * 저장된 데이터를 시각화하여 특성, 조건에 따라 데이터를 직관적으로 파악하고 분석하는데 적합한 오픈 소스 애플리케이션
- * 이번 프로젝트에서는 엘라스틱서치에 저장된 데이터를 웹 브라우저에서 파악하고 시각화하는 역할

DATA PIPELINE ARCHITECTURE



DATA PIPELINE ARCHITECTURE



1

Fetching Data from Steam API with Airflow

에어플로우 환경 구축 및 STEAM API 활용하여 데이터 가져오는 DAG 만들기

AIRFLOW 구축

AIRFLOW 구축 및 Jupyter lab을 설치해서 DAG 개발환경 구축

목표는 Airflow Dag를 생성해서 Steam API를 호출해 원하는 데이터 가져오는 것

이에 맞게 instance 1번에 Airflow with Docker 설치

airflow 메뉴 Admin > Variables 메뉴에서 API KEY 환경변수로 추가

	Key	Val	Description
<input type="checkbox"/>	STEAM_API_KEY	*****	steam web api key

tip: docker compose파일에서 예제코드 비활성화

```

environment:
  &airflow-common-env
  AIRFLOW__CORE__EXECUTOR: CeleryExecutor
  AIRFLOW__DATABASE__SQL_ALCHEMY_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow
  AIRFLOW__CELERY__RESULT_BACKEND: db+postgresql://airflow:airflow@postgres/celery_results
  AIRFLOW__CELERY__BROKER_URL: redis://:@redis:6379/0
  AIRFLOW__CORE__FERNET_KEY: ''
  AIRFLOW__CORE__DAGS_ARE_PAUSED_AT_CREATION: 'true'
  AIRFLOW__CORE__LOAD_EXAMPLES: 'false'
  AIRFLOW__API__AUTH_BACKENDS: airflow.api.auth.backend.default
# yamllint disable rule:line-length
# Use simple http server on scheduler for health checks
# See https://airflow.apache.org/docs/apache-airflow/sta

```

AIRFLOW 구축

Jupyter lab 설치 및 환경설정

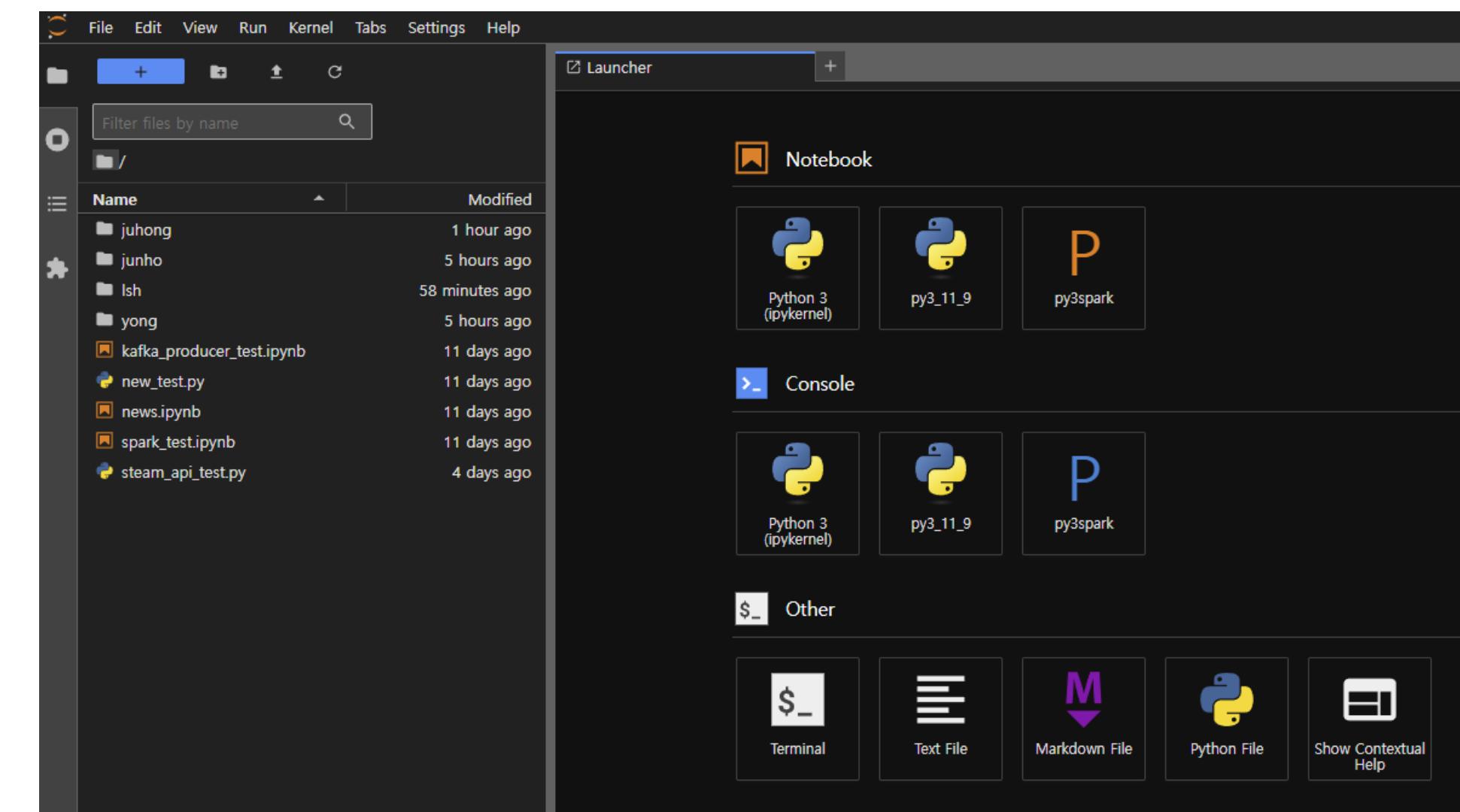
Jupyter lab을 Airflow가 있는 곳에 설치해서 Dag 개발하기 편한 환경 구성

편의성 위해 token 비활성화 및 기본경로 airflow dag 폴더로 지정

```
# Configuration file for lab.

c = get_config() #noqa
c.ServerApp.ip = '0.0.0.0'
c.ServerApp.token = ''
c.ServerApp.password = ''
c.ServerApp.open_browser = False
c.ServerApp.root_dir = '/home/ubuntu/airflow/dags'
```

jupyter lab 세팅 및 pyenv로 파이썬 개발환경 세팅



ALL AIRFLOW DAGS

Airflow DAGs Cluster Activity Datasets Security ▾ Browse ▾ Admin ▾ Docs ▾ 15:55 JST (+09:00) ▾ AA ▾

DAGs

All 7 Active 4 Paused 3 Running 3 Failed 0 Filter DAGs by tag Search DAGs Auto-refresh C

i	DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
runDag	airflow	759	144	***** i	2024-08-05, 15:44:00 i	2024-08-05, 15:54:00 i	1	▶ ⚡	...
steam_current_player_dag	airflow	108	47	/1 ***** i	2024-08-05, 15:07:00 i	2024-08-05, 15:53:00 i	1	▶ ⚡	...
steam_get_recently_played_games	airflow	372	90	/1 ***** i	2024-08-05, 15:54:00 i	2024-08-05, 15:55:00 i	2	▶ ⚡	...
steam_news_to_kafka_requests	airflow	920	132	1 day, 0:00:00	2024-08-05, 11:14:00 i	2024-08-05, 11:15:00 i	3	▶ ⚡	...
steam_test_dags	airflow	18		/5 ***** i	2024-07-24, 16:32:02 i	2025-07-24, 09:00:00 i		▶ ⚡	...
steam_user	airflow	48	393	/1 ***** i	2024-08-05, 15:54:00 i	2024-08-05, 15:55:00 i	1	▶ ⚡	...
steam_usrinfo_dag	airflow	658	117	/1 ***** i	2024-08-05, 15:54:00 i	2024-08-05, 15:55:00 i	2	▶ ⚡	...

« ‹ 1 › ›» Showing 1-7 of 7 DAGs

DAG GET_RECENTLY_PLAYED_GAMES

레디스 DB에 데이터 읽고 쓰기

```
# Redis 설정
redis_client = redis.StrictRedis(host='172.19.0.2', port=6379, db=0) # airflow-redis-1 docker
KAFKA_TOPIC = 'rct_plyd_games'
KAFKA_BROKER = '172.31.2.88:9092'
API_KEY = '1D6B63E8C3375FDCE46BD38620595819'
START_STEAM_ID = 76561197960499900

def create_kafka_producer():
    return Producer({
        'bootstrap.servers': KAFKA_BROKER,
        'client.id': 'kafkaproducer-rct_plyd_games'
    })

# Redis에 존재하지 않을 때
def get_recently_played_games(**kwargs):
    try:
        last_steam_id = redis_client.get('last_steam_id')
        last_steam_id = int(last_steam_id.decode('utf-8')) if last_steam_id else START_STEAM_ID
    except Exception as e:
        print(f"Redis에서 가져오는 중 오류 발생: {str(e)}")
        last_steam_id = START_STEAM_ID

    redis_client.set('last_steam_id', steam_id)
```

레디스 DB와의 연결 설정

host=172.19.0.2 airflow-redis 도커 컨테이너의 호스트 IP

db=0 레디스의 기본 데이터베이스

레디스 DB에

마지막으로 불러온 Steam ID('last_steam_id')가 있다면 그것을 디코딩하고
없다면 START_STEAM_ID의 값을 할당

레디스 DB의 key 'last_steam_id'에 steam_id의 값을 저장

DAG GET_RECENTLY_PLAYED_GAMES

에어플로우의 TASK 간 데이터 교환 기능 XCOM

XCom은 에어플로우의 task 간 데이터를 교환하는 기능

Xcom 테이블은 메타 DB에 위치

"AIRFLOW_DATABASE_SQLALCHEMY_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow"

에어플로우 도커 컨테이너 구성 시 설정한 위의 환경변수에 따라 메타 DB는 포스트그레SQL DB

```
task1_get_recently_played_games = PythonOperator(
    task_id='task1_get_recently_played_games',
    python_callable=get_recently_played_games,
    provide_context=True,
    dag=dag
)
```

```
def get_recently_played_games(**kwargs):
    kwargs['task_instance'].xcom_push(key='recently_played_games', value=recently_played_games)
```

XCom 테이블에 key값을 'recently_played_games',
value값을 'recently_played_games'에 할당된 값으로 하여 데이터 저장

```
task2_send_data_to_kafka = PythonOperator(
    task_id='task2_send_data_to_kafka',
    python_callable=send_data_to_kafka,
    provide_context=True,
    dag=dag
)
```

```
def send_data_to_kafka(**kwargs):
    producer = create_kafka_producer()

    task_instance = kwargs['task_instance']
    recently_played_games = task_instance.xcom_pull(task_ids='task1_get_recently_played_games', key='recently_played_games')
```

task('task1_get_recently_played_games')에서 저장한 데이터를 XCom 테이블에서 key('recently_played_games')로 불러옴

DAG STEAM_CURRENT_PLAYER_DAG

카프카 프로듀서를 만들고 토픽을 통해 컨슈머에게 데이터 보내기

```
def get_steam_players_data(**context):
    producer = create_kafka_producer()
    local_tz = pendulum.timezone("Asia/Seoul") # 한국 시간대 설정
    current_time = pendulum.now(local_tz).to_iso8601_string() # 현재 시간을 ISO 8601 형식으로 가져옴

    for game_name, game_id in GAMES.items():
        player_count = get_current_players(game_id)
        key = game_name # 게임 이름을 키로 사용
        value = json.dumps({
            'game': game_name,
            'player_count': player_count,
            'timestamp': current_time # 현재 시간을 추가
        }) # 게임 이름, 플레이어 수, 타임스탬프를 JSON으로 변환

        # Kafka로 메시지 전송
        try:
            producer.produce(KAFKA_TOPIC, key=key, value=value)
        except Exception as e:
            print(f"Kafka 전송 실패: {e}") # 에러 출력

    # 전송 완료 후 대기
    producer.flush()
```

create_kafka_producer() 카프카 프로듀서를 생성하는 함수

```
# Kafka Producer 설정
def create_kafka_producer():
    return Producer({
        'bootstrap.servers': KAFKA_BROKER
    })
```

key를 'game_name', value를 게임에 관한 데이터로 할당
value값은 json 데이터 형태로 변환된 게임별 시간에 따른
접속자 수 데이터

```
# 상수 정의
KAFKA_TOPIC = 'topic_currPlayer'
KAFKA_BROKER = '172.31.5.148:9092'
```

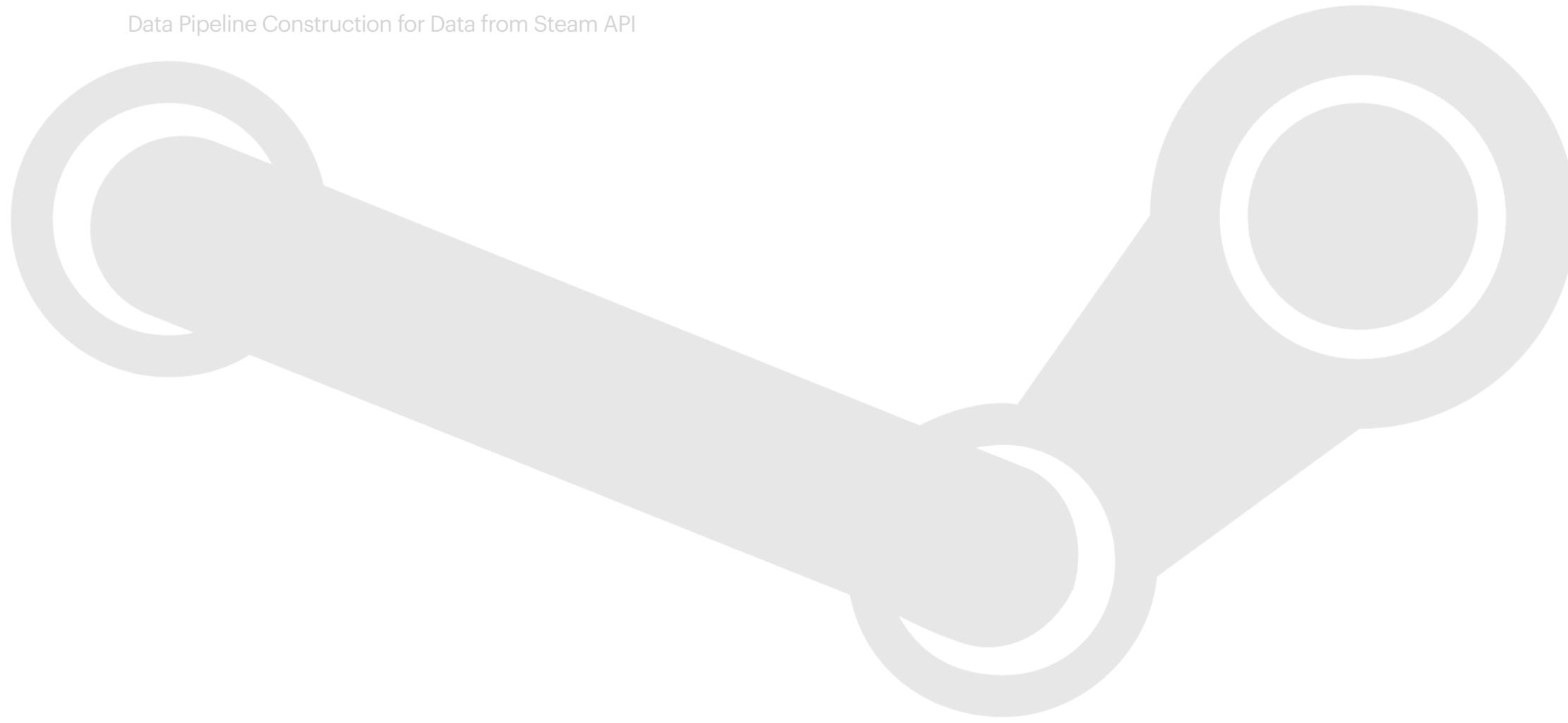
프로듀서가 사전에 지정된 토픽에
데이터를 전달
callback 함수의 출력값을 통해 데이터가 토픽에 도착했는지
여부를 확인할 수 있음

2

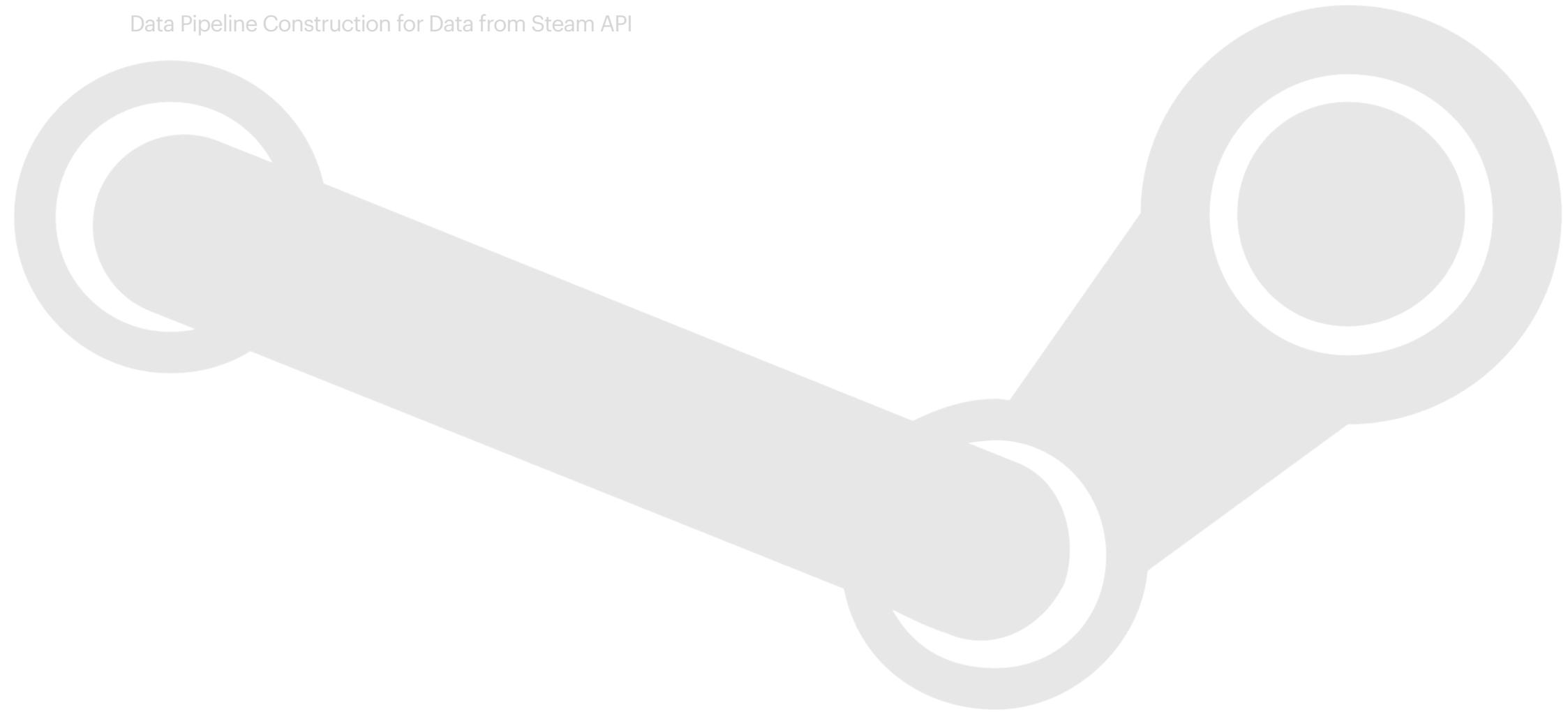
Kafka Cluster

카프카 클러스터 구축하기





What is Kraft Mode?



No Zookeeper!

확장성

간단한 설치 운영 난이도

즉각적 장애 조치

- kafka docker compose 설정코드 전문

```
docker-compose.yml
1  version: '3.8'
2
3  volumes:
4    kafka_volume:
5      driver: local
6
7  services:
8    kafka:
9      image: apache/kafka:3.7.1
10     container_name: kafka1
11     ports:
12       - "9092:9092"
13       - "9093:9093"
14     environment:
15       KAFKA_PROCESS_ROLES: broker,controller
16       KAFKA_NODE_ID: 1
17       KAFKA_CONTROLLER_QUORUM_VOTERS: 1@172.31.2.88:9093,2@172.31.5.148:9093,3@172.31.9.18:9093
18       KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9092,CONTROLLER://0.0.0.0:9093
19       KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://172.31.2.88:9092
20       KAFKA_LOG_DIRS: /var/lib/kafka/data
21       KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,CONTROLLER:PLAINTEXT
22       KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
23       KAFKA_AUTO_CREATE_TOPICS_ENABLE: "false"
24       KAFKA_CONTROLLER_LISTENER_NAMES: CONTROLLER
25       KAFKA_LOG_RETENTION_MS: 300000
26     volumes:
27       - kafka_volume:/var/lib/kafka/data
28
```

```
services:  
  kafka:  
    image: apache/kafka:3.7.1  
    container_name: kafka1  
    ports:  
      - "9092:9092"  
      - "9093:9093"
```

9092: 클라이언트와 통신하기 위한 포트

9093: 브로커 간에 내부 통신을 위한 포트. 열지 않으면 클러스터 구성 불가능

```
KAFKA_CONTROLLER_QUORUM_VOTERS: 1@172.31.2.88:9093,2@172.31.5.148:9093,3@172.31.9.18:9093  
KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9092,CONTROLLER://0.0.0.0:9093  
KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://172.31.2.88:9092
```

Controller가 되기 위한 **voters**는 모든 브로커 IP가 입후보자.

내부 브로커끼리의 통신을 위한 **LISTENERS**

외부 클라이언트를 위한 **ADVERTISED_LISTENERS**

```
KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,CONTROLLER:PLAINTEXT  
KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT  
KAFKA_AUTO_CREATE_TOPICS_ENABLE: "false"
```

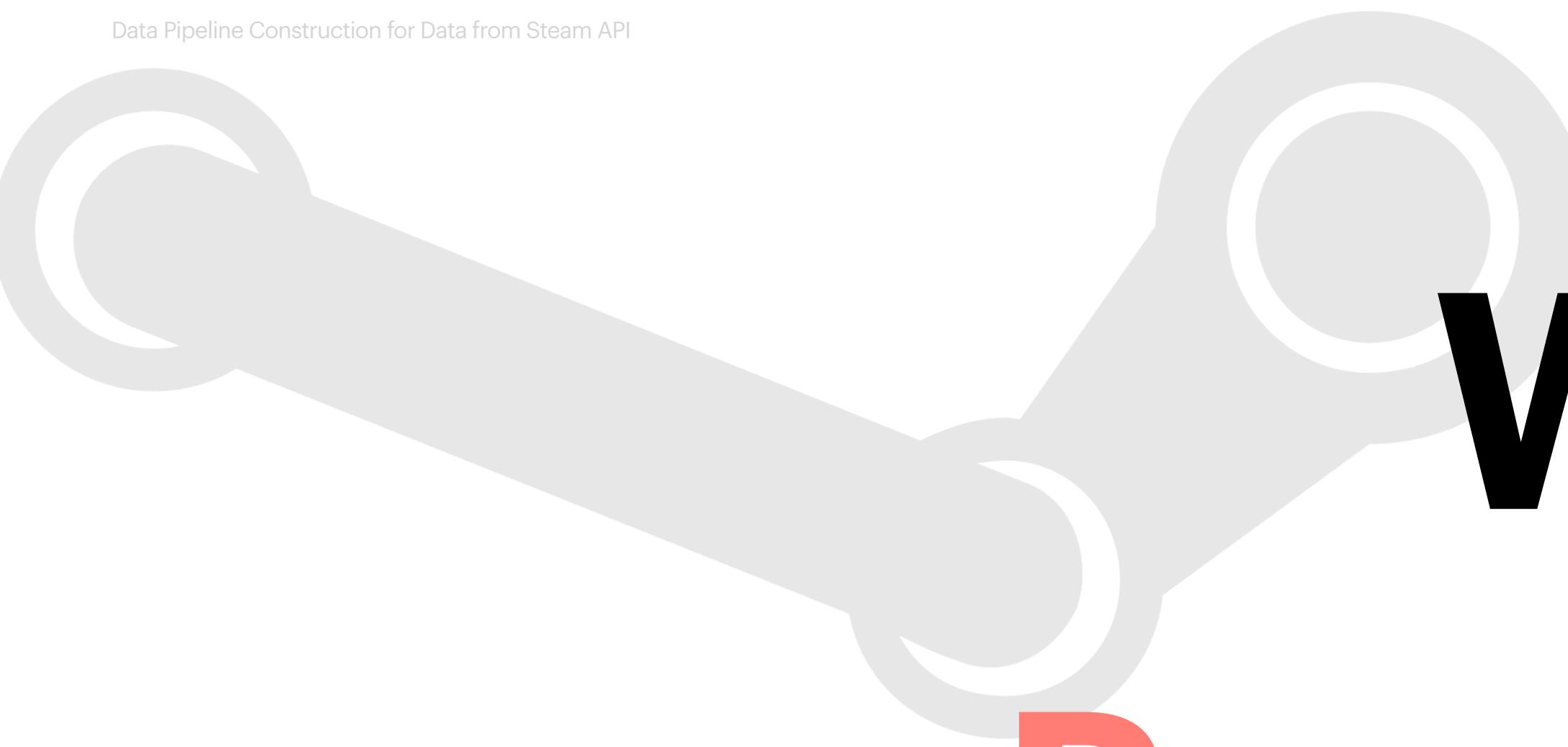
반드시 설정해야하는 보안 프로토콜. 코드 상으론 비보안 상태

토픽 자동생성은 꺼주기!

3

Monitoring from Kafka Cluster with Prometheus & grafana

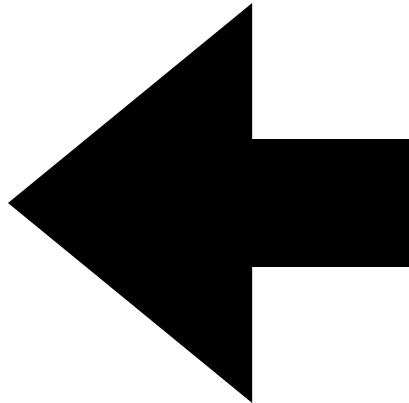
시스템 성능 모니터링 구축하기



why Prometheus & Grafana

- Prometheus 기본설정 코드

```
global:  
  scrape_interval: 15s  
  
scrape_configs:  
  - job_name: 'node_exporter'  
    static_configs:  
      - targets: ['localhost:9090']  
  
  - job_name: 'kafka_exporter'  
    static_configs:  
      - targets: ['172.31.5.148:9308']
```



Kafka exporter

설치한 **kafka exporter**를 추가

- Prometheus 실행코드

```
docker run -d ¶
--name=prometheus ¶
-p 9090:9090 ¶
-v $(pwd)/prometheus.yml:/etc/prometheus/prometheus.yml ¶
prom/prometheus|
```

이후 실행 명령어

docker start prometheus

- Kafka exporter 실행 코드

```
docker run -d --name kafka-exporter \
--network work_default \
-e KAFKA_SERVER=172.31.2.88:9092,172.31.5.148:9092,172.31.9.18:9092 \
-p 9308:9308 \
danielqsj/kafka-exporter|
```

kafka가 사용중인 네트워크와 동일해야하고
브로커들의 정보를 빼내기 위해 **advertised listener IP**와 포트에 연결합니다.

이후 실행 명령어

docker start kafka-exporter

- Prometheus UI

The screenshot shows the Prometheus UI Targets page. At the top, there is a navigation bar with links for Prometheus, Alerts, Graph, Status, and Help. Below the navigation bar, the title "Targets" is displayed. Under the "Targets" title, there are buttons for "All scrape pools", "All" (which is selected), "Unhealthy", and "Collapse All". A search bar with the placeholder "Filter by endpoint or labels" is also present. Two target sections are listed: "kafka_exporter (1/1 up)" and "node_exporter (1/1 up)". Each section contains a table with columns: Endpoint, State, Labels, Last Scrape, and Scrape Duration.

Endpoint	State	Labels	Last Scrape	Scrape Duration
http://43.203.211.151:9308/metrics	UP	instance="43.203.211.151:9308" job="kafka_exporter" ▾	4.762s ago	167.360ms

Endpoint	State	Labels	Last Scrape	Scrape Duration
http://localhost:9090/metrics	UP	instance="localhost:9090" job="node_exporter" ▾	9.942s ago	4.365ms

kafka exporter를 통해 매트릭 정보연결 성공

GRAFANA 구축 & 활용 (1)

Prometheus에서 Grafana로 Metric 데이터 가져오기

Data source 항목

데이터 주체인 prometheus

&

prometheus의 IP 입력

(설치된 인스턴스)

The screenshot shows the Grafana interface with the following details:

- Path:** Home > Connections > Data sources > prometheus
- Title:** prometheus
- Type:** Prometheus
- Settings Tab:** The "Settings" tab is selected.
- Name:** prometheus
- Default:** A toggle switch is shown, currently off.
- Connection Configuration:** The "Prometheus server URL *" field contains "http://43.203.211.151:9090/".
- Note:** A message at the bottom states: "Before you can use the Prometheus data source, you must configure it below or in the config file. For detailed information, see the documentation." It also notes that fields marked with * are required.

GRAFANA 구축 & 활용 (2)

다양한 Metric 정보를 시각화해서 Kafka 시스템 모니터링

- Metric 종류 -

go / kafka / process

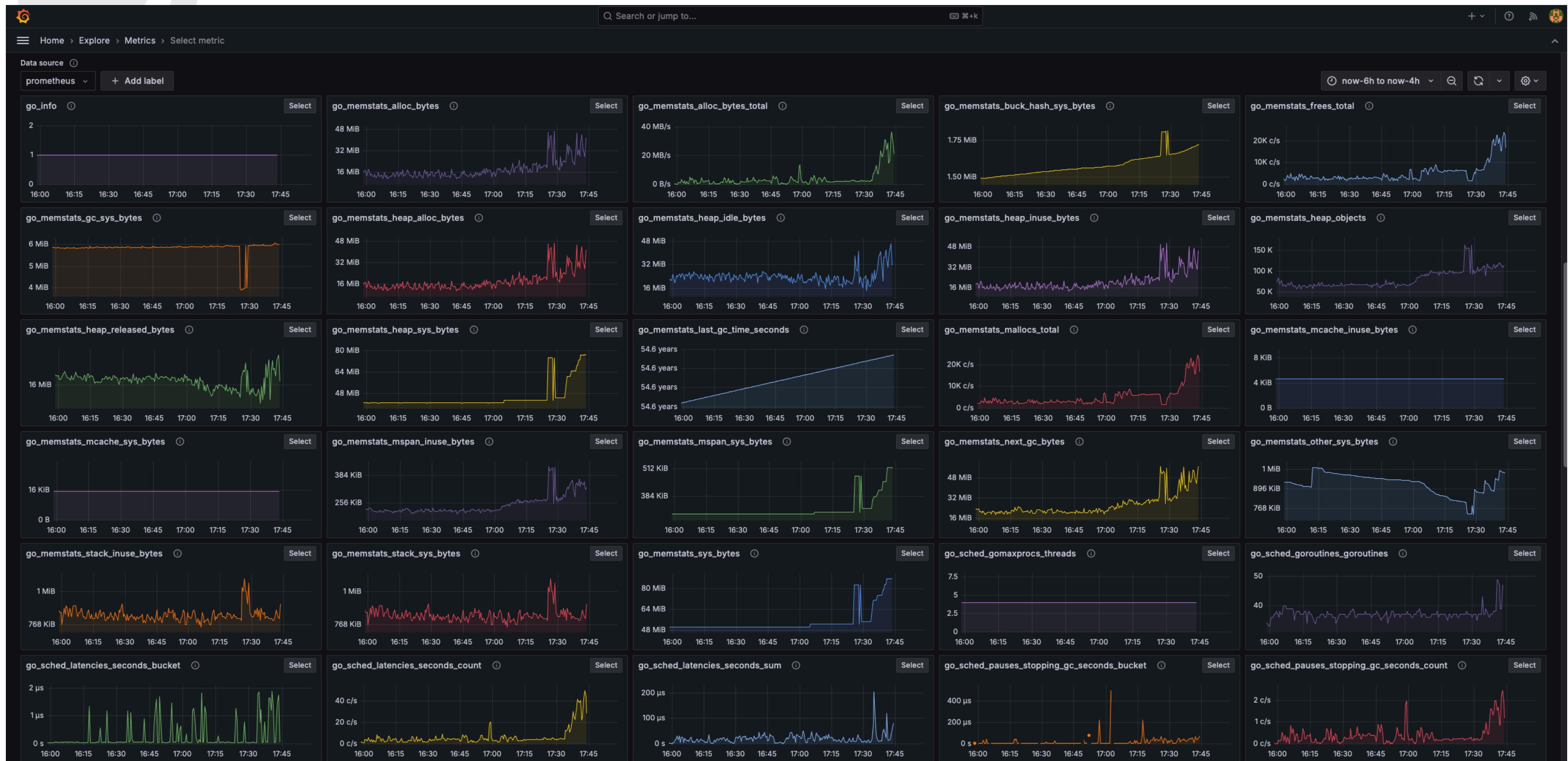
Prometheus / 스크랩 / 상태

카테고리별 수십가지 다양한

세부 지표들 중에서 필요한 내용을

선택해 대시보드에서 예시와 같이

활용해보자!





Kafka to Elastic search and Kibana Built with Docker Containers

도커 컨테이너 환경에서 카프카 데이터 엘라스틱서치에 넣고 키바나에서 확인하기

KAFKA TO ELASTIC SEARCH

kafka connect 설치 및 환경설정

Dockerfile

```
FROM confluentinc/cp-kafka-connect:6.2.0
# Install the Elasticsearch connector
RUN confluent-hub install --no-prompt confluentinc/kafka-connect-elasticsearch:11.1.0
```

docker-compose.yaml

Dockerfile로 install한 connect를 docker-compose에서 빌드

기본포트는 8083이지만 포트가 겹치는 경우가 발생하여 8081로 변경

kafka cluster private ip를 bootstrap_servers 옵션에 줌

```
services:
  kafka-connect:
    build: .
    container_name: kafka-connect
    ports:
      - 8081:8083
    environment:
      CONNECT_BOOTSTRAP_SERVERS: "172.31.2.88:9092,172.31.5.148:9092,172.31.9.18:9092"
      CONNECT_REST_PORT: 8083
      CONNECT_GROUP_ID: compose-connect-group
      CONNECT_CONFIG_STORAGE_TOPIC: docker-connect-configs
      CONNECT_OFFSET_STORAGE_TOPIC: docker-connect-offsets
      CONNECT_STATUS_STORAGE_TOPIC: docker-connect-status
      CONNECT_KEY_CONVERTER: org.apache.kafka.connect.storage.StringConverter
```

KAFKA TO ELASTIC SEARCH

kafka connect 이용

kafka connect는 RESTful API를 지원하므로 curl -GET -DELETE -PUT으로 조회, 삭제, 수정이 가능함

커넥트 옵션 수정하며 생성

```
curl -X PUT http://localhost:8081/connectors/{커넥트이름}/config -H "Content-Type: application/json" -d '{  
    "connector.class": "io.confluent.connect.elasticsearch.ElasticsearchSinkConnector",  
    "topics": "{본인토픽}",  
    "key.ignore": "true",  
    "schema.ignore": "true",  
    "connection.url": "http://elasticsearch:9200",  
    "type.name": "_doc",  
    "name": "{커넥트이름}",  
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",  
    "value.converter.schemas.enable": "false",  
    "errors.tolerance": "all"  
}'
```

ELASTIC SEARCH AND KIBANA

docker-compose.yaml

```
elasticsearch:
  image: docker.elastic.co/elasticsearch/elasticsearch:8.9.0
  container_name: elasticsearch
  environment:
    - discovery.type=single-node
    - xpack.security.enabled=false
    - xpack.security.enrollment.enabled=false
    - xpack.security.http.ssl.enabled=false
    - xpack.security.transport.ssl.enabled=false
  ports:
    - "9200:9200"
  restart: always

kibana:
  image: docker.elastic.co/kibana/kibana:8.9.0
  container_name: kibana
  ports:
    - "5601:5601"
  environment:
    - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
    - xpack.security.enabled=false
  depends_on:
    - elasticsearch
  restart: always
```

elasticsearch, kibana 이미지 버전을 맞춤

8버전부터 로그인 옵션이 기본이기 때문에
접속에 용이하게 xpack.security
관련 옵션을 false로 줌

각각 기본포트인 9200과5601을 부여

kibana는 elasticsearch에 의존하게 설정

ELASTIC SEARCH

elastic search 0|용

elasticsearch는 RESTful API를 지원하므로

`curl -GET -DELETE -PUT`으로 조회, 삭제, 수정이 가능함

Elastic search index rct_plyf_games 조회

```
curl -X GET http://localhost:9200/rct_plyd_games/_search?pretty
```

kafka topic rct plyf games을 Elastic search에서 확인

KIBANA

{instance3 public ip}:5601로 접속

Index Management

[Index Management docs](#)

[Indices](#) [Data Streams](#) [Index Templates](#) [Component Templates](#)

Update your Elasticsearch indices individually or in bulk. [Learn more.](#)

Include rollup indices

Include hidden indices

rct x [Reload indices](#)

<input type="checkbox"/>	Name	Health	Status	Primaries	Replicas	Docs count	Storage size	Data stream
<input type="checkbox"/>	rct_plyd_games	● yellow	open	1	1	49	29.69kb	

Rows per page: 10 ▾

< 1 >

Elastic search에 추가된 index와 데이터를 확인할 수 있음

5

Overloading Test for the Data Pipeline Built with Docker Containers

도커 컨테이너 환경의 데이터 파이프라인에서 과부하 테스트 하기

KAFKA 과부하 테스트

Docker Container로 구축한 Kafka Cluster가 안정적을 동작하는지 확인하기 위해 과부하 테스트 해보기

Kakfa Cluster가 구축되었지만 클러스터 환경이 잘 동작하는지, 부하를 잘 분산하여 데이터를 스트리밍 할 수 있는지 의구심이 생김

테스트 필요하다고 판단하여 Kafka Topic 여러개에 많은 데이터를 빠르게 전송해보기로 결정 후 Airflow job을 특정 시간에 여러번 동시에 실행시키면서 테스트 진행

현재 파이프라인 구조 상 부하를 가장 많이 받는 곳이 Kafka라고 생각해서 Kakfa Cluster에 과부하 테스트 진행

Cluster 환경이 구축되었기 때문에 데이터가 많이 들어와도 문제없이 잘 데이터를 스트리밍 할 수 있을 것이라고 예상

API Call limit가 있기 때문에 producer 쪽에 for문을 거는 방식으로 진행

```
for game in recently_played_games:
    key = str(game['appid']).encode('utf-8')
    value = json.dumps(game).encode('utf-8')
    try:
        for i in range(1000):
            producer.produce(KAFKA_TOPIC, key=key, value=value, callback=delivery_report)
            print(f"Kafka에 최근 플레이한 게임 데이터 전송 완료. {value}")
    except Exception as e:
        print(f"Kafka에 메시지 전송 실패: {str(e)}")
producer.flush()
```

KAFKA 과부하 테스트

문제: Kafka Cluster에만 집중한 나머지 다른 환경 고려 못함

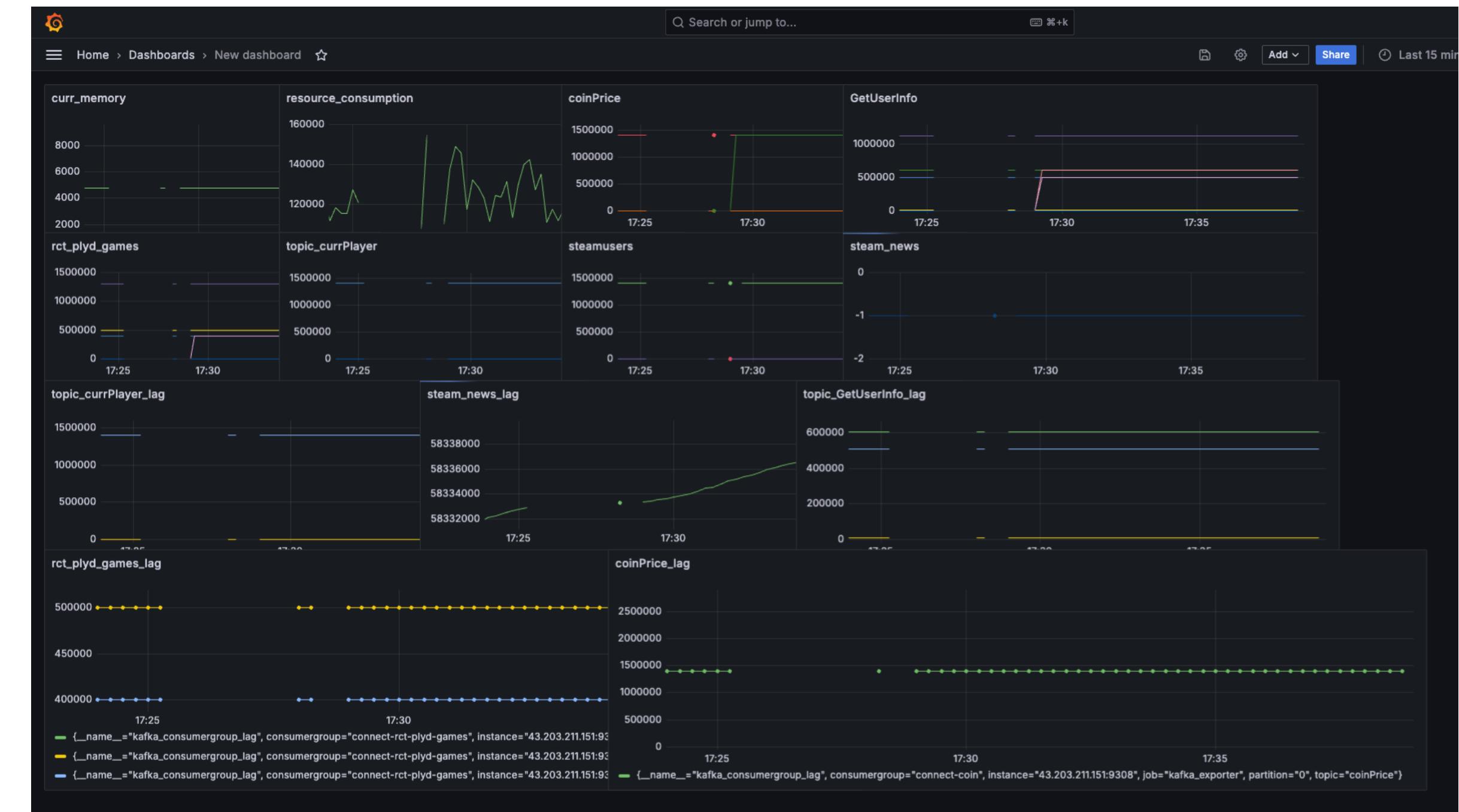
특정 인스턴스(2번)에만 스토리지가 빠르게 차오르기 시작

- 파티션을 제대로 생성하지 않아서 topic 데이터들이 일부 파티션에 몰리는 문제 발생
- 추후 파티션을 여러개 추가하여 topic 데이터 분산저장 처리

```
ubuntu@instance2:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       96G   80G   17G  83% /
tmpfs           7.9G    0    7.9G  0% /dev/shm
tmpfs           3.2G  4.1M  3.2G  1% /run
tmpfs           5.0M    0   5.0M  0% /run/lock
/dev/xvda16     881M 133M  687M 17% /boot
/dev/xvda15     105M  6.1M  99M  6% /boot/efi
shm              64M    0   64M  0% /run/container
tmpfs            64M    0   64M  0% /run/containers
```

인스턴스 메모리 부족(3번)

- 메모리 제한을 걸지 않고 Elastic Search, Kibana 등 같이 실행되고 있는 앱들에 메모리 제한을 걸지 않고 전부 실행
- 메모리 부족으로 3번 인스턴스에 있는 Kafka Cluster도 사망 Grafana로 문제상황 모니터링
- 다른 앱들의 메모리 제한 등 적절한 설정 이후 복구 성공



REVIEW

개인별 후기

다양한 시도를 해보며 학습해 보는 것이 팀의 목표였기에 전체적인 프로젝트의 완성도 자체는 높지는 않으나, 여러 문제상황에 직면하고 해결하며 경험이 많이 쌓여 다음 번에 이런 프로젝트를 한다면 정말 잘할 수 있을 것 같습니다.

leader

김민혁

팀원들 덕분에 한번도 다뤄보지 못해본 플랫폼들을 다뤄보고 이렇게 저렇게 시도해보고 강의 시간 동안 배우지 못했던, 못하는 것들을 배울 수 있는 소중한 기회였다. 언제 또 클라우드 환경에서 인스턴스 여러 개 띄워놓고 우리 마음대로 데이터 파이프라인을 구축해보겠나. 언제 또 비용, 리소스 걱정 덜하면서 마음껏 삽질해보겠나. 언제 또 서버 터뜨려보겠나.

이주홍

프로젝트 시작 이전엔 아는 것이 별로 없었는데 프로젝트를 통해 많이 배울 수 있어서 기쁩니다. 또한 열정적인 팀원 분들을 만나게 되어 평일 주말 없이 진심으로 프로젝트로 몰입하는 좋은 경험을 겪어 행운이라고 생각합니다.

팀원들과 시행착오를 공유하며 시간을 아끼고 자신이 원하는 경험과 도전을 자유롭게 시도해 보았습니다.

그동안 정리 되지 않았던 학습 내용들이 이해되고 연결되며 폭발적인 성장이 가능했던 프로젝트로 너무 소중한 시간이었습니다.

주말에도 모두 12시간 인스턴스 제한을 모두 소비할 정도로 열정적이었고 이 과정이 앞으로의 커리어에 든든한 기반일 될 것 같습니다.

도커는 어렵다는 생각이 지배적이였는데 할 수 있다고 확실하게 말해주시고 친절하게 설명해주세요서 따라갈 수 있었습니다.

혼자였다면 시도도 못해봤을 상황이었지만, 팀원들이 있어서 믿고 따라갈 수 있었습니다.

변준호**서용원****이시형**

한계, 아쉬운 점, 이후 프로젝트 발전 방향

중간에 개인 사정으로 잠깐 참여를 못한게 아쉬웠고,

kubernetes의 전환도

시간이 부족해서 못 한거라 아쉽습니다.

leader 김민혁

"아는 만큼 보인다"를 씁쓸하지만 다시금 깨닫는 시간이었다. 다양한 애플리케이션을 이용해 데이터 파이프라인을 구축해본 것은 만족하지만, 구축하면서 여러 가지 부분을 다루지 못해 아쉽다. 마지막으로, 시간이 더 있었다면 쿠버네티스에서 겪었던 장벽들을 넘을 수 있었을까 의문이 들면서도 넘고 싶다는 욕심이 든다.

이주홍

쿠버네티스에서 구축하던 kafka cluster를 마무리 하지 못한 것이 가장 아쉽지만 이후 프로젝트에서는 꼭 구축 할 수 있도록 더 노력하고자 마음먹게 되었습니다.

쿠버네티스를 완성하기에 시간이 조금 모자랐던 부분이 가장 아쉽습니다.

쿠버네티스 클러스터 내&외부 네트워크 설정을 마무리하고 Kubernetes on Cloud 시스템을 완벽히 구축 해보고고 싶습니다.

컨테이너화한 도구들도 느낀 점이 정말 많지만 최대한 짧게 압축해보자면

호환성 / 종속성 / 마이그레이션 안에서 발생하는 잠재적인 문제들이 수도 없이 많고

결국 앞으로도 다양한 환경에서 사례를 해결하고 연구하는 충분한 시간을 지속적으로 반복하고 경험 해봐야만 한다!

내가 생각하지 못했던 방향으로 의견을 조율하고 따라가는 것이 쉽지 않았습니다. 타인의 의견에 적극 공감하고 공동의 목표에 집중하려는 노력의 필요성을 느꼈습니다.



변준호

서용원

이시형