프로그래머스 데이터 엔지니어링 데브코스 1기
파이널 프로젝트

2팀 2조

Github, StackOverFlow
데이터 수집, 분석을 통한
대시보드 구축

오픈소스 트랜드 분석

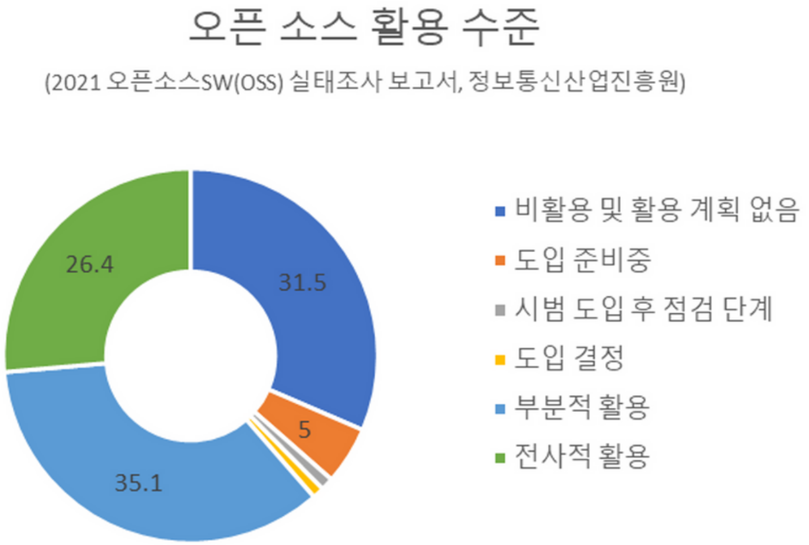
발표자 : 남윤아
팀원 : 김상희, 김혜민, 남윤아

목차

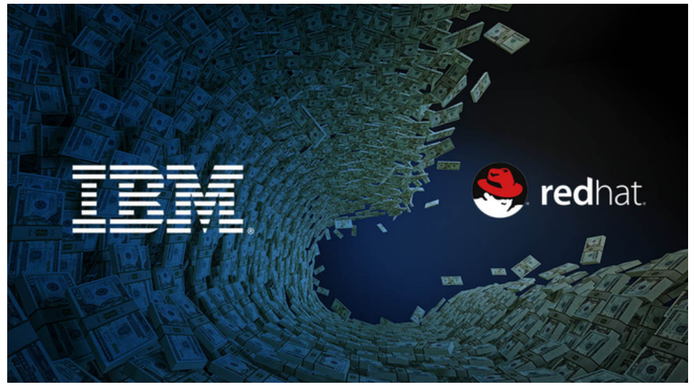
주제 소개

오픈소스 트랜드 분석

1. 오픈 소스 트렌드를 쉽고 자세하게 파악할 수 있도록 데이터 파이프라인 개발과 대시보드 제공



2021 오픈소스SW(OSS) 실태조사 보고서



글로벌 기업의 오픈소스 플랫폼 인수

02. 활용 기술 및 프레임워크

활용 기술과 개발 환경 (요금)

어떻게 하면 한정된 자원 안에 프로젝트를 완수할 수 있을까?

1. **Free Tier 서비스 최대 활용**
   프리티어 안에서 사용할 수 있는 서비스를 최대한 활용하여 비용 절감

2. **불필요한 비용 발생 최소화**
   타 region 간 데이터 이동으로 인해 발생하는 비용 등 불필요한 비용 발생 최소화

3. **오픈 소스 및 Free trial 활용**
   Airflow, Snowflake Free trial을 활용하여 비용 발생 최소화

02. 활용 기술 및 프레임워크

활용 기술과 개발 환경 (요금)

Develop과
Production 환경 구분

환경 및 사용금액 I 기술 스택	Develop	Production	사용 금액 (23.09.03 기준) (AWS Region: US-EAST1)
Apache Spark (Python)	로컬 (Docker)	**Amazon EMR (m5.xlarge)**	USD 0.95
Apache Airflow & monitoring (Python)	로컬 (Docker Compose)	**Amazon EC2 (t3.2xlarge)**	USD 69.93
Data Warehouse	Snowflake	**Snowflake**	0 (Free trial)
Data Lake	Amazon S3	**Amazon S3**	USD 0.42
AWS Monitoring	-	**Amazon CloudWatch**	USD 0.16
Dashboard	Preset	**Preset**	0 (Free trial)
SCM (Software Configuration Management)	Github	**Github**	0
기타		**Amazon SecretManager Amazon Lambda**	USD 4.01
총 금액	-	-	**USD 75.47**

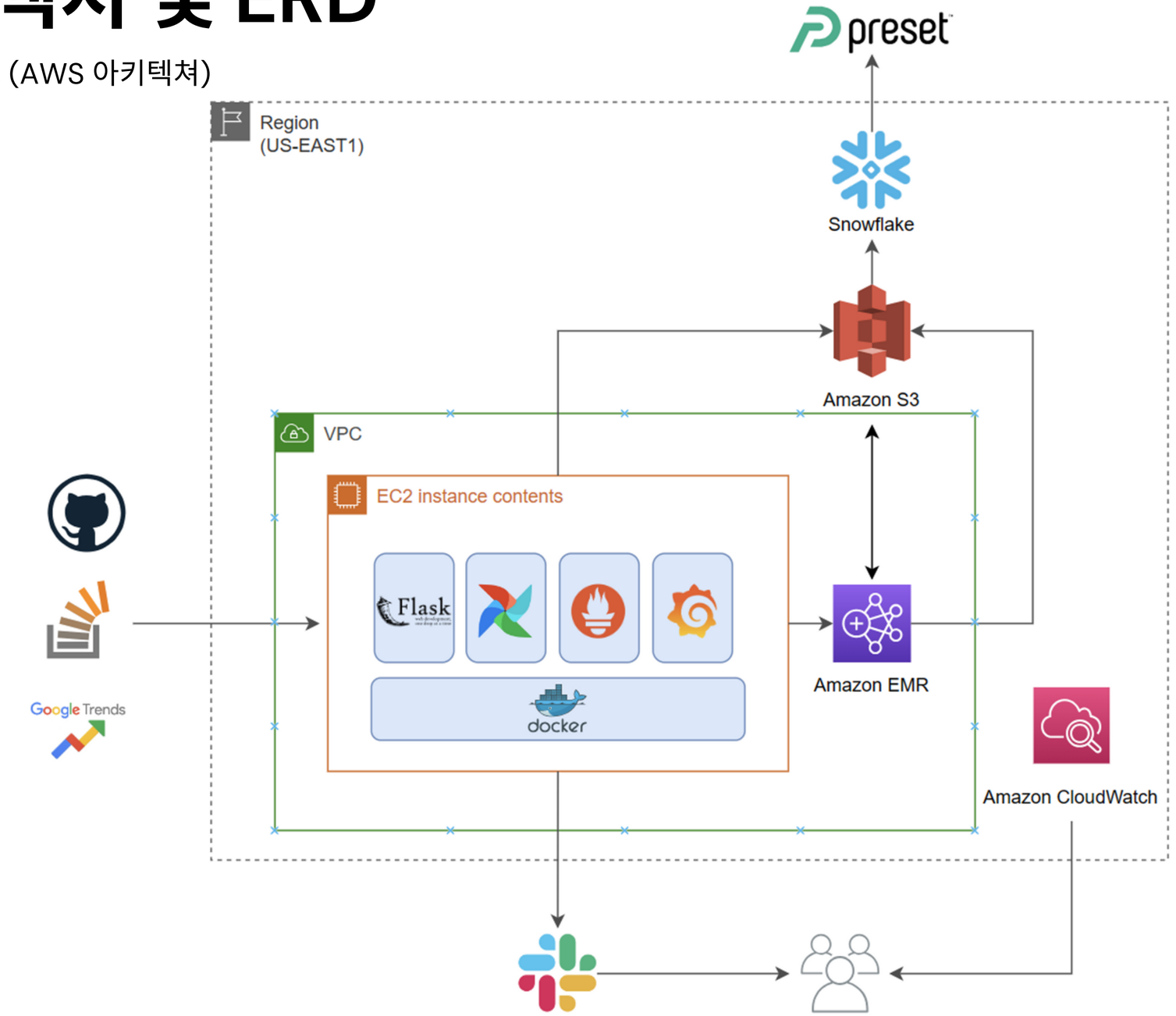
Amazon S3 :

- raw : 수집한 데이터
- analytics : 정제한 데이터
- spark_scripts : Amazon EMR Spark 코드
- cluster_log : Amazon EMR 클러스터 로그

03. 아키텍처 및 ERD

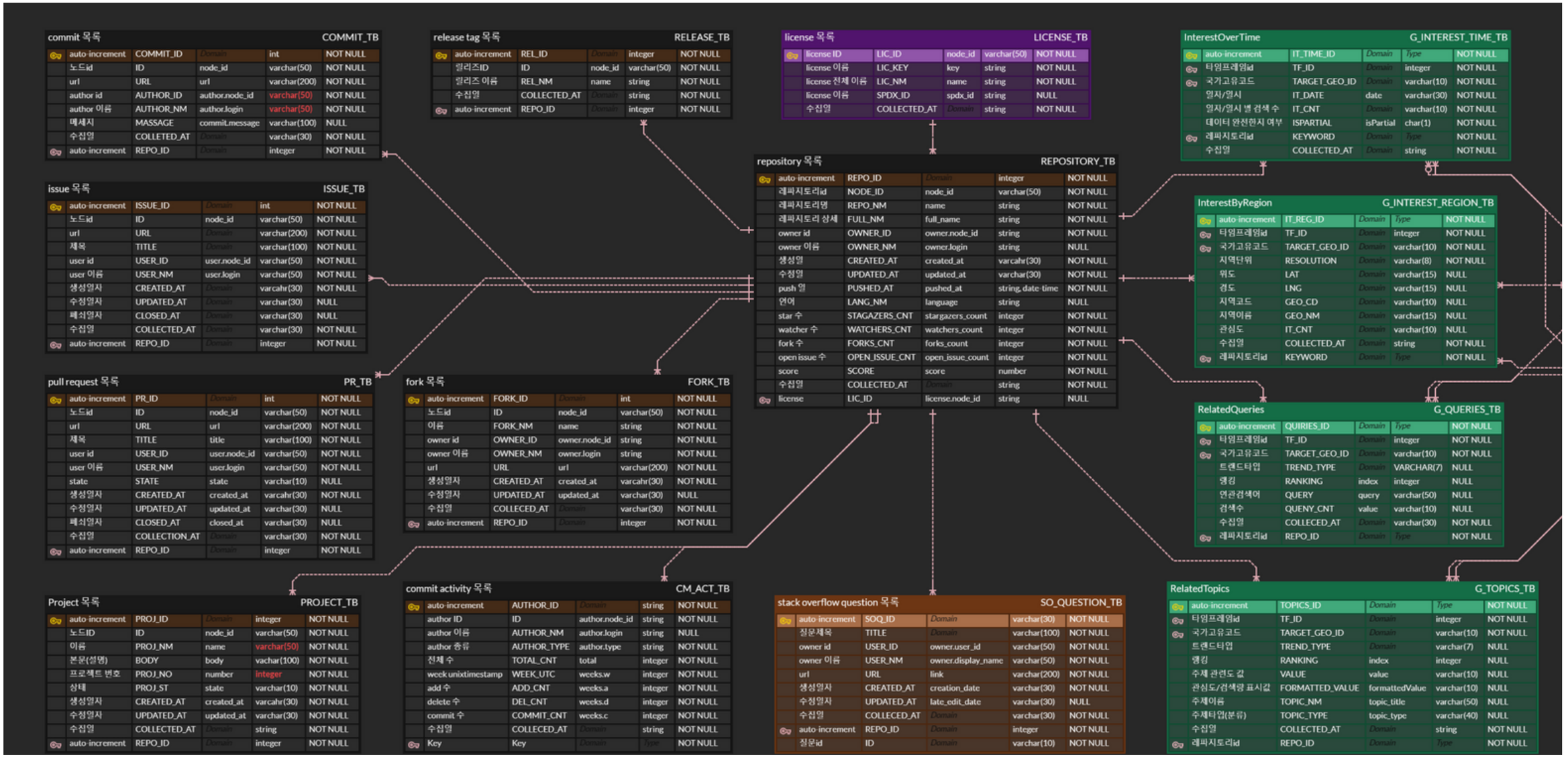
프로젝트 인프라 구성도 (AWS 아키텍쳐)

데이터 레이크,
데이터 웨어하우스 구축



03. 아키텍처 및 ERD

Entity Relationship Diagram



ETL

04. 주요 구현 사항

04. 주요 구현 사항 - 수집 (Airflow)

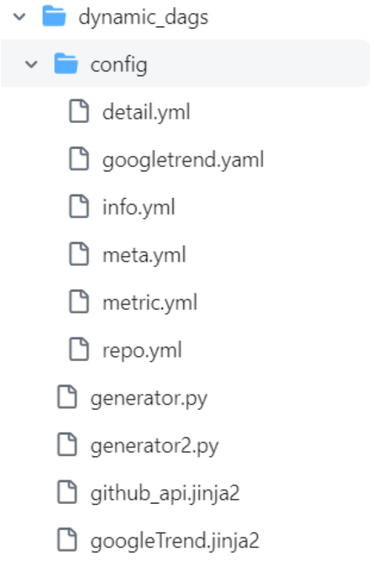
1) 수집 데이터 및 DAG 종류

DAG 명	스케줄링	데이터 소스	수집 항목 및 설명	수집 기간
github_repo	1일	Github	Star/Fork 순으로 각 100개의 Repository 목록	8. 28~9. 3
github_meta	1일	Github	license 목록	8. 28~9. 3
github_detail	1시간	Github	repository의 issue, pull request, commit 정보	8. 28~9. 3
github_info	1시간	Github	repository의 release, project, language, fork 정보	8. 28~9. 3
github_metric	1시간	Github	repository의 contributor 별 활동 (additions, deletions, commits)	8. 28~9. 3
stackoverflow_question_list	1일	StackOverFlow	repository와 관련된 질문 목록 수	8. 28~9. 3
emr_repo	1일	S3 - Raw Data	Repository 목록 중복 제거 후 Repository_TB에 맞게 변환한 뒤 parquet로 S3에 저장	8. 28~9. 3
emr_meta	1일	S3 - Raw Data	license 목록 중복 제거 후 LICENSE_TB에 맞게 변환한 뒤 parquet로 S3에 저장	8. 28~9. 3
emr_detail	1시간	S3 - Raw Data	issue, pull request, commit 정보 중복 제거 후 각 TABLE 에 맞게 변환한 뒤 parquet로 S3에 저장	8. 28~9. 3
emr_info	1시간	S3 - Raw Data	release, project, language, fork 정보 중복 제거 후 각 TABLE에 맞게 변환한 뒤 parquet로 S3에 저장	8. 28~9. 3
emr_metric	1시간	S3 - Raw Data	contributor 별 활동 정보 중복 제거 후 CM_ACT_TB에 맞게 변환한 뒤 parquet로 S3에 저장	8. 28~9. 3
s3→snowflake	하루	S3 - Analytics	S3에 있는 데이터를 snowflake로 적재	8.28 ~

04. 주요 구현 사항 – 수집 (Airflow)

2) 세부 구현 사항

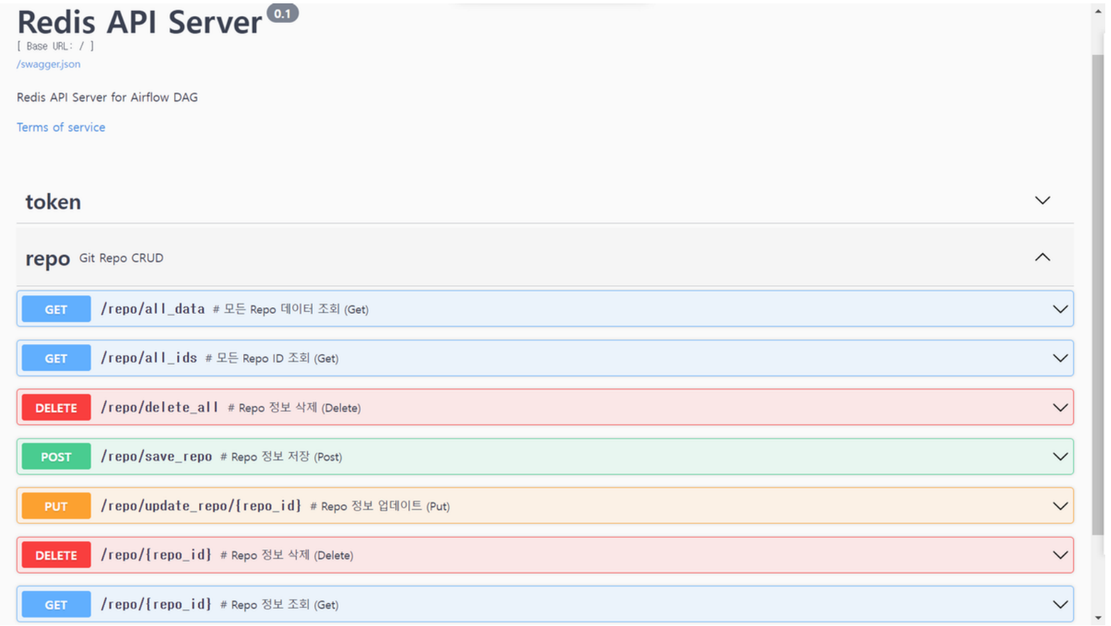
1 Dynamic DAG 작성



2 Plugin 파일

- AWS 관련
  (AWS Secret Manager, AWS S3)
- Slack 호출
- Flask API 호출
- JSON 저장
- Github API 및 Pytrend 호출 등

3 Flask REST-API 개발



Repository의 고유 ID와 OWNER/REPO 정보를 Redis에 저장함

Read/Write 속도가 빠른 *Key-Value* 형식의 *Redis*
비교적 가벼운 *Flask* 프레임워크
향후 *Redis* 서버를 별도로 구축하는 것을 고려하여 *REST API* 개발

04. 주요 구현 사항 – 수집 (Airflow)

3) Airflow Configuration 및 파라미터 튜닝

Airflow 관점

AIRFLOW__CORE__PARALLELISM: '512'
AIRFLOW__CELERY__WORKER_CONCURRENCY: 512

AIRFLOW__SCHEDULER__MIN_FILE_PROCESS_INTERVAL: 40
AIRFLOW__SCHEDULER__DAG_DIR_LIST_INTERVAL: 60
AIRFLOW__SCHEDULER__PARSING_PROCESSES: 4

AIRFLOW__CORE__DEFUALT_TIMEZONE: utc

- 최대 512개의 Task만 동시 실행 가능
- Task를 실행시키는 Celery의 큐 수 512개
- DAG 구문 분석을 병렬로 실행할 수 있는 프로세스 수 4개
  (Amazon EC2 t3.2xlarge 기준 vCPU수 8개)

DAG 관점

```python
with DAG(
    dag_id="github_repo",
    start_date=datetime(2023, 8, 29),
    schedule='50 23,11 * * *',
    catchup=False,
    on_success_callback=send_slack_message().success_alert,
    on_failure_callback=send_slack_message().fail_alert,
    concurrency = 90, # 해당 DAG에 대해 최대 동시에 실행 가능한 Task 수
    max_active_runs = 3  #  해당 DAG에 대해 동시에 RUN 가능한 DAG 수
) as dag:
```

DAG별로 최대 실행 가능한 DAG 및 Task 수에 제한을 둠

Task 관점

```python
@task(trigger_rule=TriggerRule.ONE_FAILED, retries=1, pool="api_pool")
def extract(kind, url, params=None):
    from plugins.github_api import get_request

    response = get_request(kind=kind, url=url, params=params)
```

Task에 Pool을 두어 API 동시 호출 수(=Task 수) 제한함

04. 주요 구현 사항 - 수집 (Airflow)

4) Raw Data 적재

1. Amazon S3 Partitioning

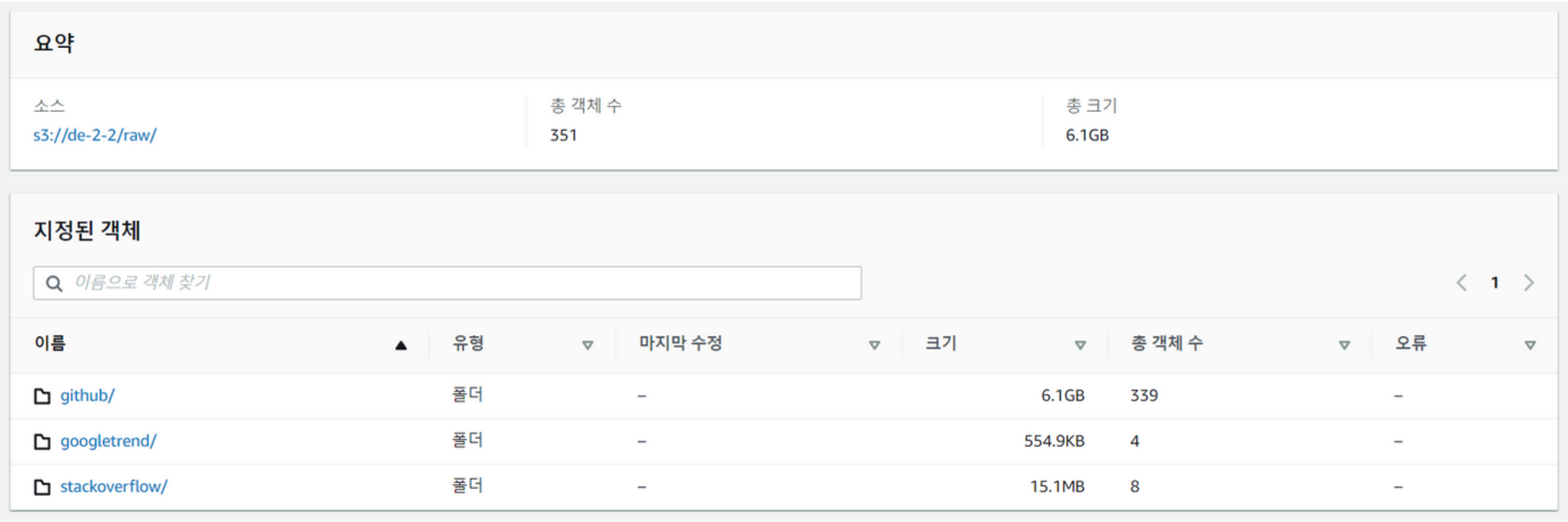
```
# 원본 데이터
/raw/github/{api명}/2023/08/15
/raw/googletrend/2023/08/15
/raw/stackoverflow/2023/08/15
# 분석 데이터
/analytics/github/{api명}/2023/08/15
/analytics/googletrend/2023/08/15
/analytics/stackoverflow/2023/08/15
```

- 데이터 저장할때 가능한 **64MB** 이상으로 저장되게끔 구성
  - 데이터 스키마 형식에 맞게 필요한 칼럼 정보만 추출 후 **json** 파일로 저장
  - S3 API 비용 줄이기 위함
  - 데이터 소스와 수집 주기가 비슷한 API끼리 묶음
- **Partitioning** 적용하여 Read/Write 성능을 높임

---

2. 수집한 데이터 (23.09.03 기준)

- 수집 기간 : 2023.08.28 ~ 2023.09.03 (7일)
- 총 데이터 크기 : **6.1 GB**



04. 주요 구현 사항 - 정제 및 적재

2팀 2조

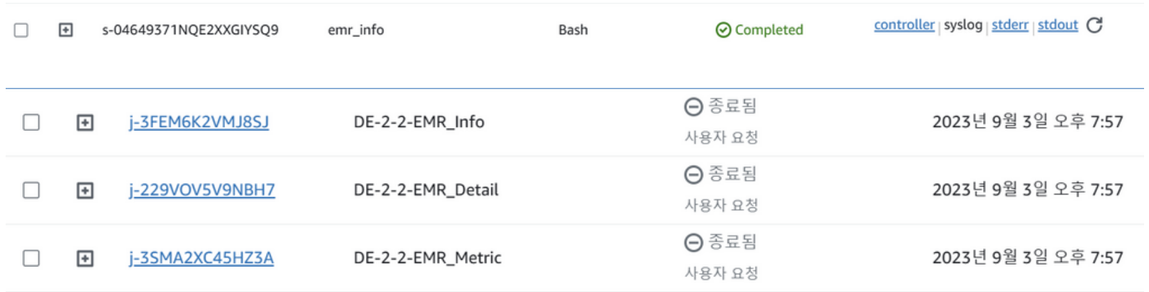
Amazon EMR

1. Amazon EMR Cluster 구성

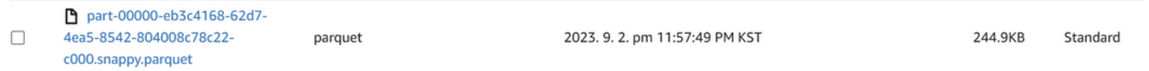
```
JOB_FLOW_OVERRIDES = {
    "Name": "DE-2-2-EMR_Metric",
    "ReleaseLabel": "emr-6.12.0",
    "Applications": [{"Name": "Spark"}],
    "LogUri": "s3://de-2-2/cluster-log",
    "Instances": {
        "InstanceGroups": [
            {
                "Name": "Primary node",
                "Market": "ON_DEMAND",
                "InstanceRole": "MASTER",
                "InstanceType": "m5.xlarge",
                "InstanceCount": 1,
            },
        ],
        "KeepJobFlowAliveWhenNoSteps": True,
        "TerminationProtected": False,
        'Ec2SubnetId': 'subnet-0ff43f8d26bd81499',
    },

    "Steps": SPARK_STEPS,
    "JobFlowRole": "Amazone-DE-2-2-EMR_EC2_DefaultRole",
    "ServiceRole": "Amazone-DE-2-2-EMR_DefaultRole",
}
```

- 워커 노드를 사용하지 않고 단일 마스터 노드로 구성
  - 클러스터당 처리해야할 데이터의 크기가 100mb 이내
- cluster의 log 정보는 S3에 저장
- Cluster 생성 후 데이터 정제 Step으로 넘어가게끔 설정

2. Amazon EMR 구현 사항



- 비용을 최소화하기 위해 EMR Cluster 생성부터 Step 실행, Cluster 종료까지의 과정을 자동화 함
  - EmrJobFlowSensor를 통해 cluster 생성 실패 또는 오류 발생 시 cluster가 종료하도록 설정

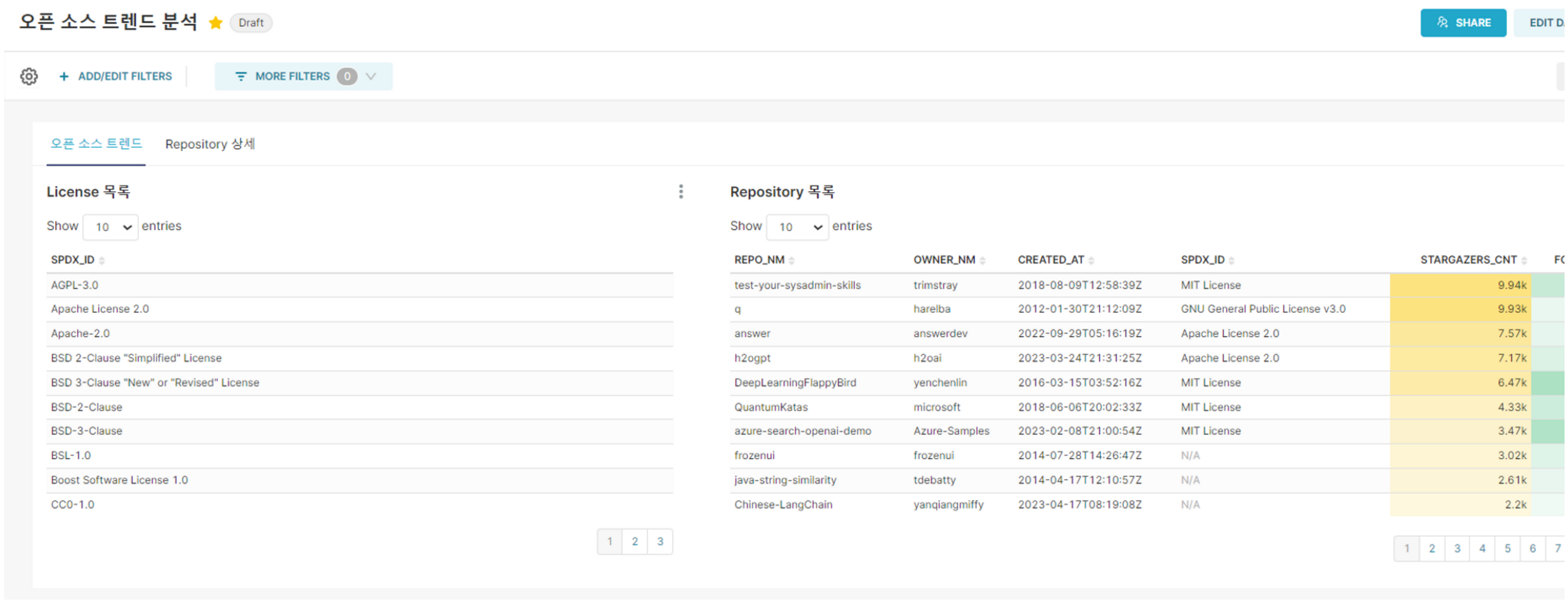


- Snowflake 및 S3 적재 시 데이터의 속도 및 크기를 고려하여 parquet 형식으로 저장

04. 주요 구현 사항 – 시각화

2팀 2조

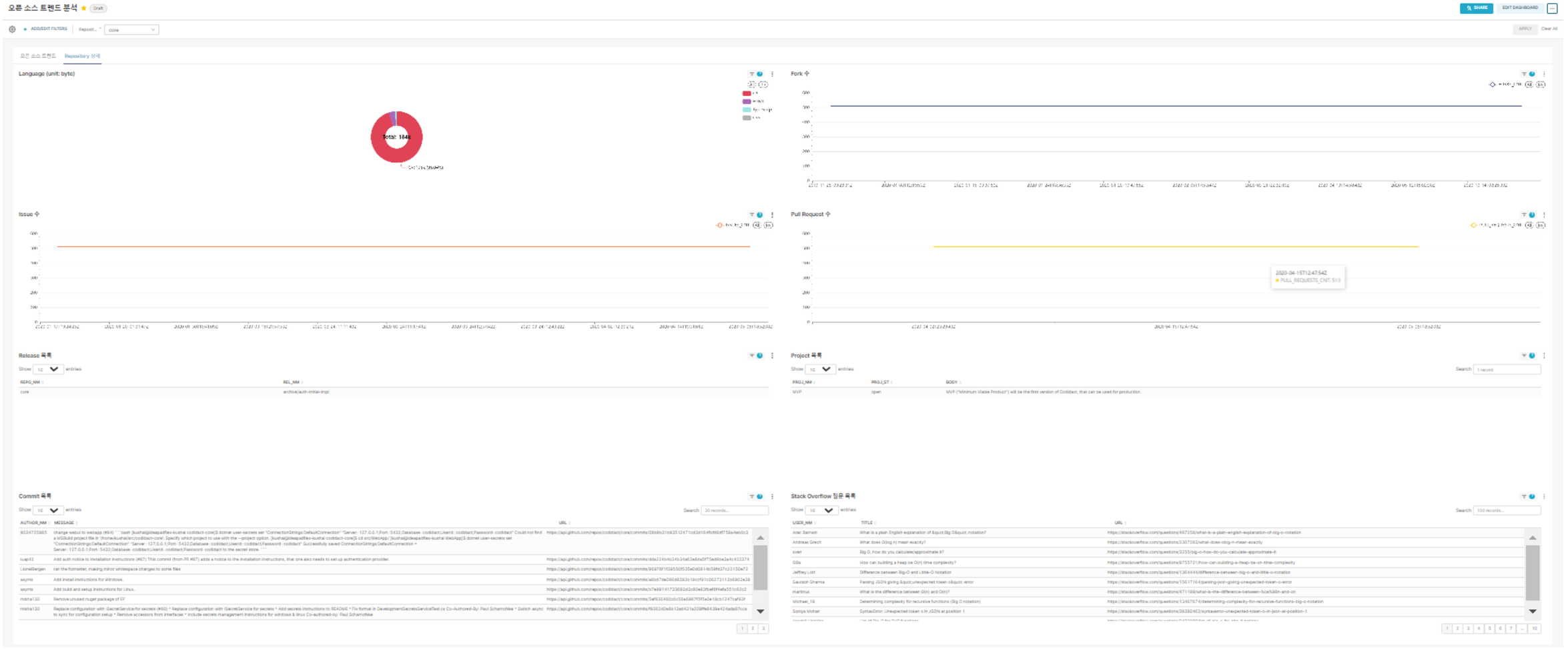
1) Preset



04. 주요 구현 사항 - 시각화

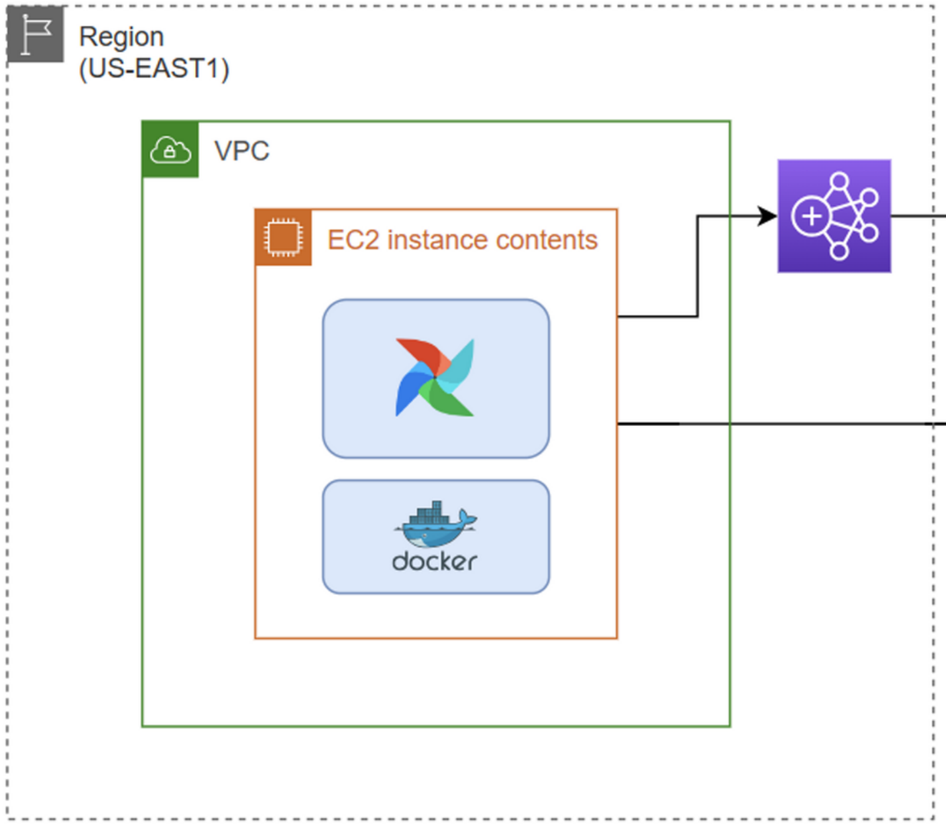
2팀 2조

1) Preset



04. 주요 구현 사항 - 모니터링

1) Slack 알람 기능

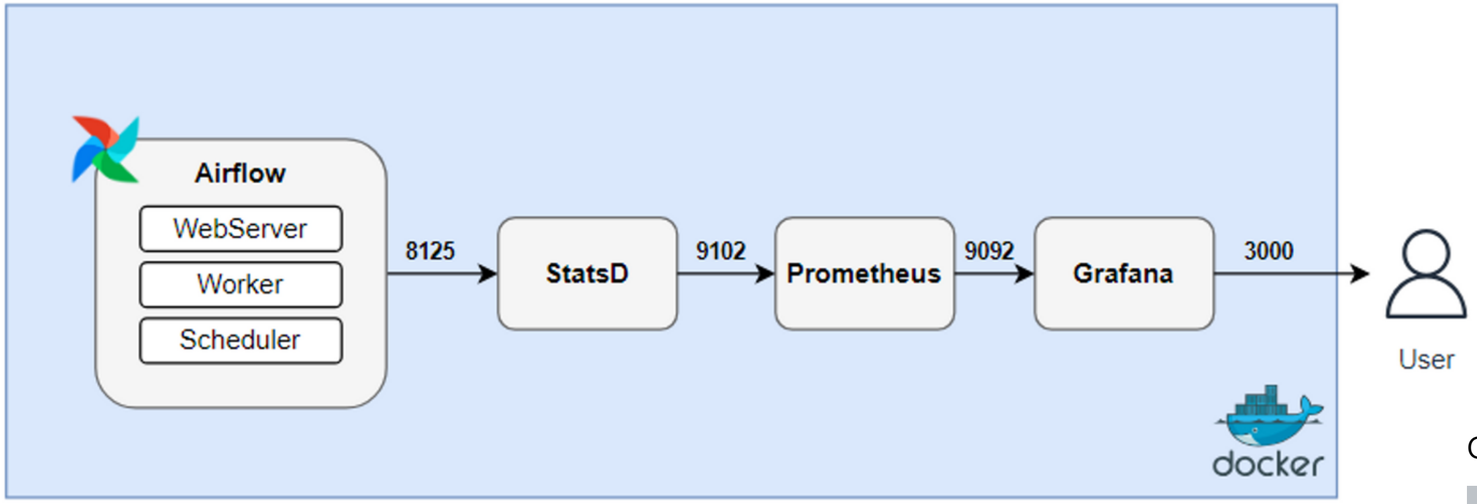


- Raw 데이터 수집과 EMR을 통한 데이터 정제 및 적재를 위한 **Dag 실행 결과와 에러 내용**을 Slack으로 발송
- 수신자가 실시간으로 데이터 파이프라인을 관리할 수 있게 함

04. 주요 구현 사항 - 모니터링

2) Grafana 대시 보드 구성

StatsD + Prometheus + Grafana



- **StatsD**로 수집한 Metric 정보를 시계열 데이터 베이스 **Prometheus**에 저장하고 대시보드 툴인 **Grafana**로 Prometheus의 Metric 데이터를 시각화한다.
  - Airflow는 내부적으로 StatsD를 통해 Metric을 외부로 전송 가능함
  - Prometheus 는 PromQL 언어로 쿼리할 수 있는 시계열 데이터베이스

- Grafana 대시보드 : Scheduler 상태 및 Task, Job, DAG 상태 등

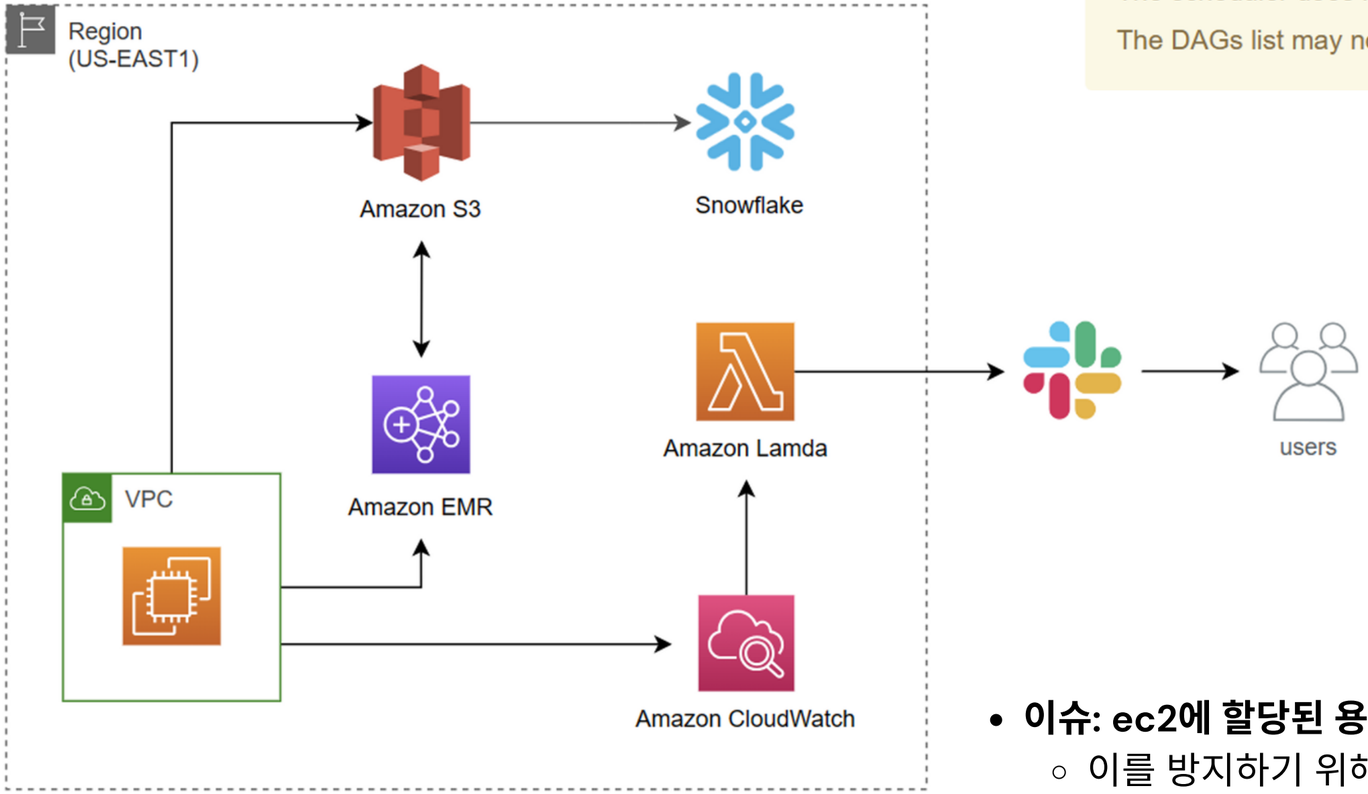
Grafana 영상 (구글 드라이브 링크)

04. 주요 구현 사항 - 모니터링

2팀 2조

3) Amazon CloudWatch & EventBridge

Amazon Clouwatch의 Usage 알림



Airflow Web UI - Scheduler 이슈

The scheduler does not appear to be running. Last heartbeat was received 30 minutes ago.
The DAGs list may not update, and new tasks will not be scheduled.

Usage 관련 Slack 알람

incoming-webhook 앱 오전 7:10
[CloudWatch_Memory_Alarm]
언제
2023-09-02 07:10:06
설명
원인
60 분 동안 1 회 disk_used_percent >= 85.0
이전 상태 **현재 상태**
정상 **위험**
바로가기
https://console.aws.amazon.com/cloudwatch/home?region=us-east-1#alarm:alarmFilter=ANY;name=CloudWatch_Memory_Alarm

- **이슈: ec2에 할당된 용량을 모두 사용할 시 airflow가 동작하지 않음**
  - 이를 방지하기 위해 할당된 용량의 85% 도달 시 slack 알림
- **Amazon CloudWatch** 경보를 트리거로 설정하여 경보 내용이 Slack으로 전달 하게끔 **Lambda**함수를 생성함
- EC2의 상태가 “shutting-dowm”, “stopping”, “stopped” 시 Slack 알림이 가도록 **Amazon EventBridge** 규칙을 생성함

감사합니다