2팀 2조 프로젝트 보고서

오픈소스 트랜드 분석

> **프로그래머스 데이터 엔지니어링 데브코스 1기 파이널 프로젝트**

1. 팀원 (2팀 2조) : 김상희, 김혜민, 남윤아

2. 프로젝트 기간 : 2023-08-01 ~ 2023-09-03

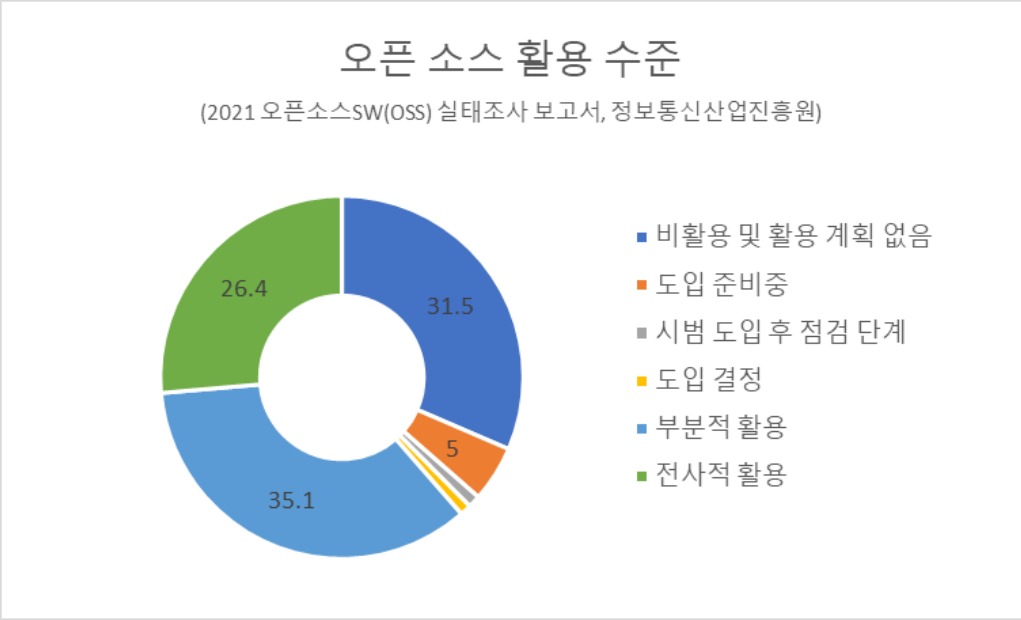
3. 목차

오픈소스 트랜드 분석
01. 주제
02. 주제 선정 이유
03. 기대 효과
04. 활용 기술 및 프레임워크
    ✔️ 1) 활용 기술과 개발 환경 (요금)
    ✔️ 2) 아키텍처 및 ERD
05. 주요 구현 사항
    ✔️ 1) Airflow 환경 구성
    ✔️ 2) 수집 (Airflow)
        2-1. **수집 데이터 및 DAG 종류**
        2-3. 세부 구현 사항
        2-4. **Airflow Configuration 및 파라미터 튜닝**
        2-5. Raw Data 적재
    ✔️ 3) 정제 및 적재
        3-1. Amazon EMR Cluster 구성
        3-2. Amazon EMR 구현
        3-2. S3-To-Snowflake
    ✔️ 4) 시각화 (Preset)
    ✔️ 5) 모니터링
06. 커뮤니케이션 전략
07. 참여자 정보 및 각 역할

01. 주제

오픈 소스에 대한 릴리즈, 이슈, 관련 질문, 관심도 등을 파악할 수 있도록 시각화 하여 대시 보드를 제공합니다.

02. 주제 선정 이유



정보통신산업진흥원의 '2021 오픈소스SW(OSS) 실태조사 보고서' 따르면, 국내 기업의 오픈 소스 활용 비율은 61.5%로 많은 것을 확인할 수 있습니다. Microsoft가 Github를, IBM이 Red Hat을 인수하는 등 오픈 소스가 더욱 중요해지고 있습니다. 오픈 소스를 도입하려면, 새로운 오픈 소스가 무엇이 있는지 기존에 도입한 오픈 소스의 이슈 혹은 버그 수정 여부 등을 지속적으로 확인해야 합니다.

따라서, 이번 프로젝트에는 오픈 소스 트랜드 및 상세 정보를 파악할 수 있는 대시보드를 제공하는 것을 주제로 선정하였습니다.

03. 기대 효과

1. repository top 10

   star 수를 기준으로 현재 가장 관심 있어하는 repository를 확인할 수 있습니다.

2. repository 상세 내용
   release, commit, issue 등 repository 상세 내용을 확인할 수 있습니다.

3. google trend 관련 키워드
   repository와 연관된 키워드를 확인할 수 있습니다.

4. 관련 stack overflow 최신 질문 목록
   repository와 관련된 stack overflow 질문 목록을 확인할 수 있습니다.

현재 가장 관심 있어하는 repository 순위를 파악할 수 있으며, repository의 최근 동향 및 관련된 키워드와 질문 목록을 확인할 수 있습니다.

04. 활용 기술 및 프레임워크

✔️ 1) 활용 기술과 개발 환경 (요금)

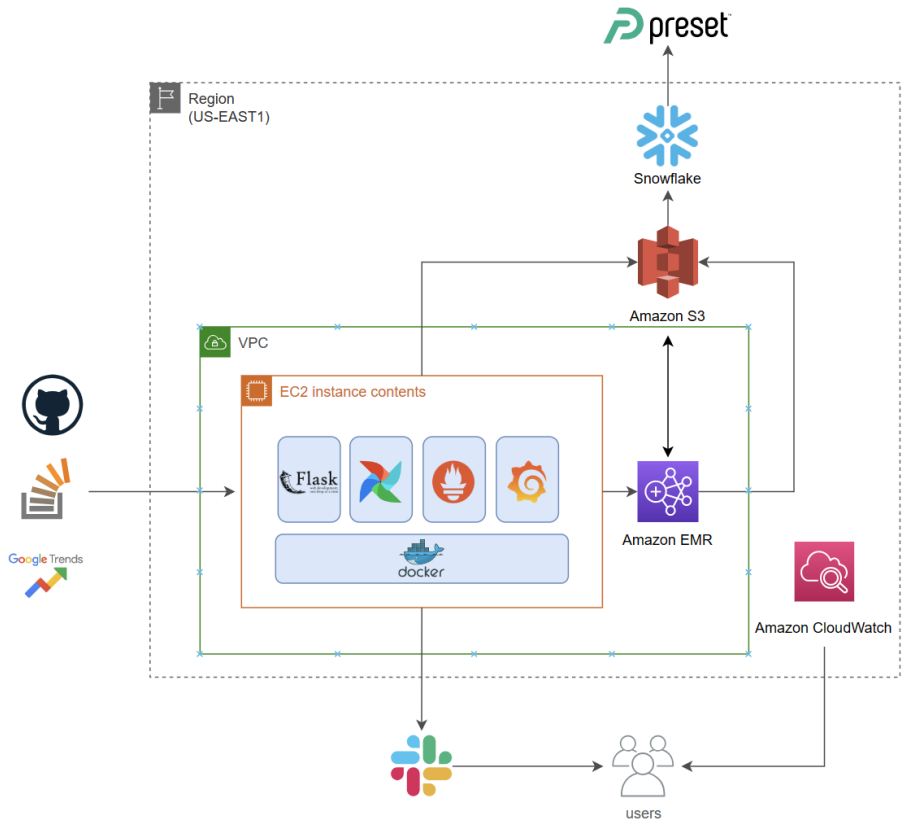
환경 및 사용금액 | 기술 스택	Develop	Production	**사용 금액 (23.09.04 기준) (AWS Region: US-EAST1)**
Apache Spark (Python)	로컬 (Docker)	Amazon EMR (m5.xlarge)	USD 0.95
Apache Airflow & monitoring (Python)	로컬 (Docker Compose)	Amazon EC2 (t3.2xlarge)	USD 69.93
Data Warehouse	Snowflake	Snowflake	0 (Free trial)
Data Lake	Amazon S3	Amazon S3	USD 0.42
AWS Monitoring	-	Amazon CloudWatch	USD 0.16
Dashboard	Preset	Preset	0 (Free trial)

환경 및 사용금액 | 기술 스택	Develop	Production	사용 금액 (23.09.04 기준) (AWS Region: US-EAST1)
SCM (Software Configuration Management)	Github	Github	0
기타		Amazon SecretManager Amazon Lambda	USD 4.01
총 금액	-	-	**USD 75.47**

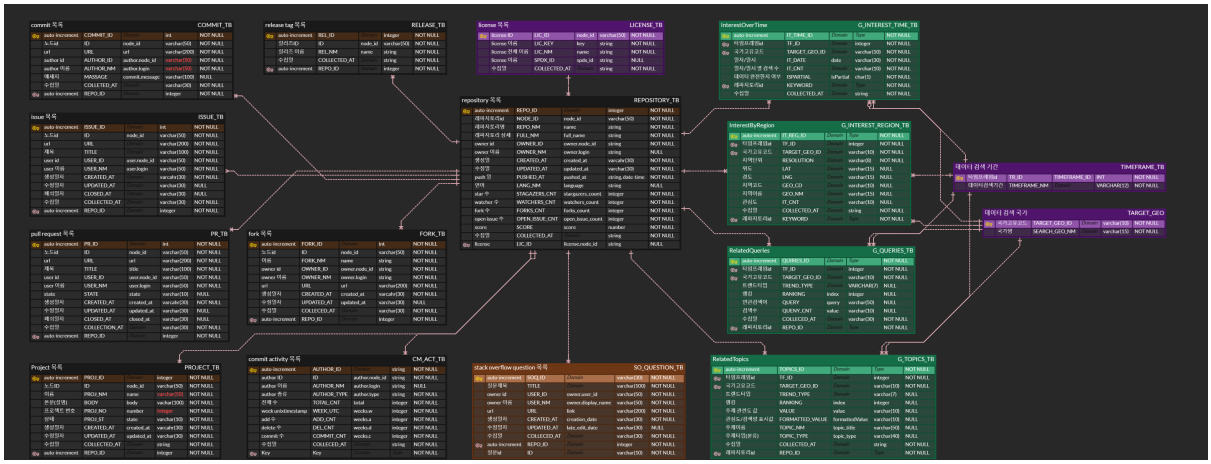
Develop 환경과 **Production** 환경을 구분하여 구현함

- **Develop** : Airflow는 로컬에서 Docker Compose로 환경 구축하여 개발한 후 Github Repository에 DAG를 저장하고 관리함
- **Production**: `Amazon EC2(t3.2xlarge)` 위에 Airflow와 모니터링을 위한 Grafana, Prometheus등을 정의한 docker-compose파일을 빌드하여 환경 구축
  - ▼ **Datalake - Amazon S3** : 버킷 내에 오브젝트 유형별로 prefix로 두어 관리함
    - `raw` : 수집한 데이터
    - `analytics` : 정제한 데이터
    - `spark_scripts` : Amazon EMR Spark 코드
    - `cluster_log` : Amazon EMR 클러스터 로그

✔️ 2) 아키텍처 및 ERD



프로젝트 인프라 구성도 (AWS 아키텍쳐)



ERD

05. 주요 구현 사항

✔️ 1) Airflow 환경 구성

Docker Compose 주요 서비스 목록

- ***airflow-scheduler, airflow-worker 등***
  (CeleryExecutor 이용)
- ***redis***
  (Airflow 메시지 브로커)
- ***chrome***
  (웹 스크래핑 작업)
- ***flask-api***
  (임시 데이터를 Redis에 저장하기 위한 Flask REST-API)
- ***postgres***
  (Airflow 메타데이터베이스)
- ***statsd-exporter***
  (Airflow Metric 데이터 수집 서비스)
- ***prometheus***
  (Airflow Metric 저장 시계열 데이터베이스)
- ***grafana***
  (Airflow 모니터링 대시보드)

✔️ 2) 수집 (Airflow)

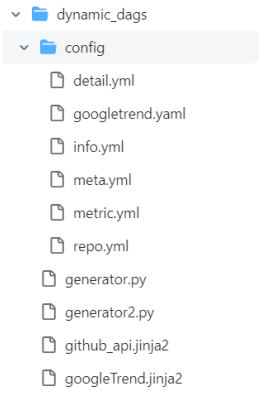
2-1. 수집 데이터 및 DAG 종류

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

DAG 명	스케줄링	데이터 소스	수집 항목 및 설명	수집 기간
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 ~

2-3. 세부 구현 사항

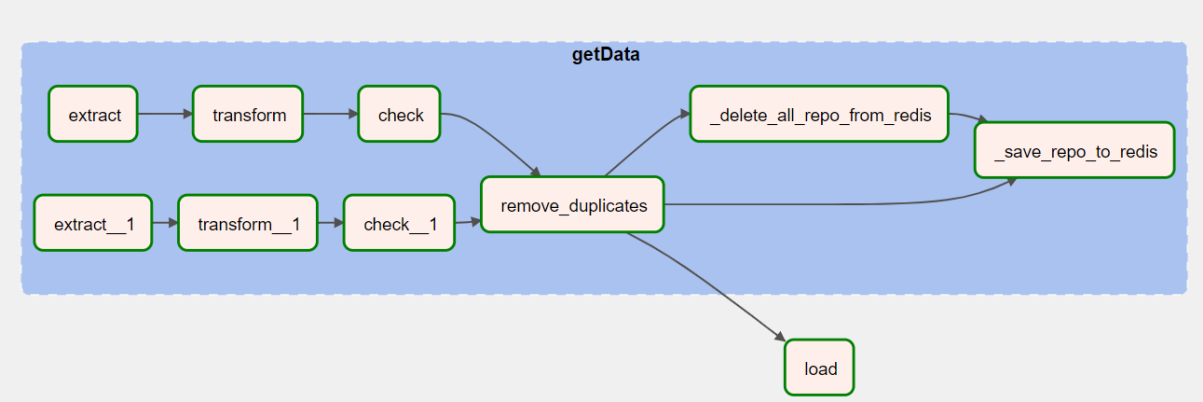
Dynamic DAG 작성



- Dynamic DAG 생성 : 데이터 수집 시 동일한 로직이 반복되어, 필요한 정보를 yml 파일로 정의하고 generate 하여 dag를 생성하였습니다.
  - api 호출 시 필요한 url, header, params
  - 수집할 column 목록
  - 데이터 포맷

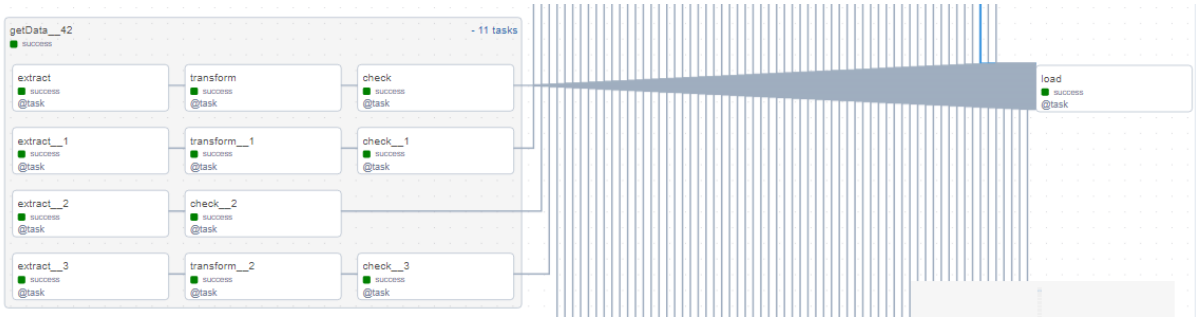
ETL DAG

01. Github Repository 데이터 수집



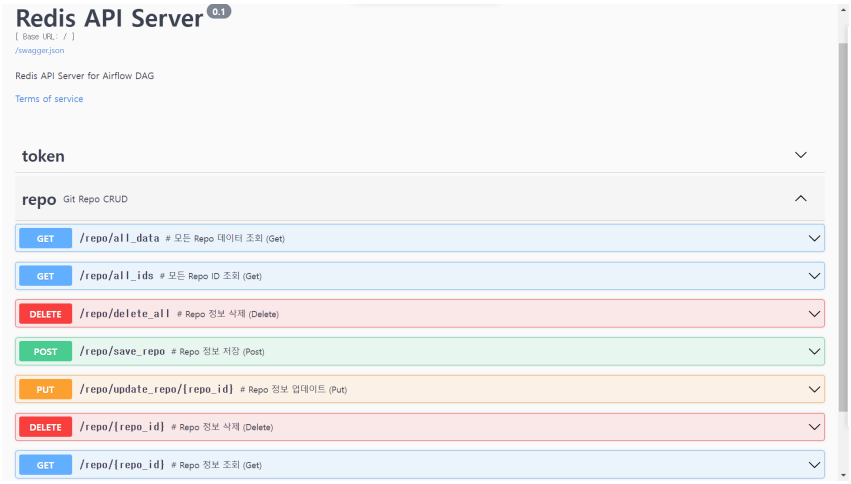
1. star순과 fork 순으로 API 호출하여 얻은 Repository 목록에서 필요한 필드만 추출한 후 중복을 제거한다.
2. Redis에 저장된 Repository 정보를 삭제 후 새로 저장한다.
3. S3 버킷에 업로드한다.

02. Github Repository 데이터 수집



1. Redis에서 Repository 정보를 불러온다.
2. 각 API 별로 데이터 추출, 정제, 데이터 검증한 후에 Amazon S3 버킷으로 업로드한다.

Flask REST-API 개발



Flask REST-API

Github의 Metric, Info, Detail 정보를 수집하기 위해서는 API 요청 시 Github Repository 정보가 필요하다. 해당 정보를 **Flask REST API**를 활용하여 **Redis**에 저장한다.

- Repository의 `고유 ID`와 `OWNER/REPO` 정보를 Redis에 저장함
- Amazon S3에 저장 시 네트워크 비용이 발생할 수 있으므로, **Read/Write 속도가 빠른 Key-Value 형식의 Redis** 선택함
- 비교적 가벼운 **Flask** 프레임워크로 구축함
- 현재 비용 문제로 Redis는 Docker 컨테이너에서 실행 중이지만, 향후 Redis 서버를 별도로 구축하는 것을 고려하여 **REST API**를 개발함

Plugin

자주 쓰이는 기능은 plugin으로 파일 구분함

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

2-4. 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
```

- `AIRFLOW__CORE__PARALLELISM` : 최대 512개의 task만 동시 실행 가능
- `AIRFLOW__CELERY__WORKER_CONCURRENCY` : task를 실행시키는 Celery의 큐 수 512개
- `AIRFLOW__SCHEDULER__MIN_FILE_PROCESS_INTERVAL` : 각 DAG 파일 40초 단위로 구문 분석
- `AIRFLOW__SCHEDULER__DAG_DIR_LIST_INTERVAL` : DAG 폴더에서 새 파일 검색하는 빈도 60초
- `AIRFLOW__SCHEDULER__PARSING_PROCESSES` : DAG 구문 분석을 병렬로 실행할 수 있는 프로세스 수 4개 (Amazon EC2 t3.2xlarge 기준 vCPU수 8개)
- `AIRFLOW__CORE__DEFUALT_TIMEZONE` : 타임존을 UTC로 통일함

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:
```

DAG 설정 - concurrency, max_active_runs

동시 다발적인 DAG의 실행은 시스템 성능 저하로 이어지기에, 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)
```

Task 설정 - pool

	Pool	Slots	Running Slots	Queued Slots	Scheduled Slots
	api_pool	60	0	60	673

Airflow Pool 지정

Github API 및 Pytrend를 호출하는 Task에 `pool`(*api_pool*)을 두어 API 동시 호출 수(=Task 수) 제한을 둠

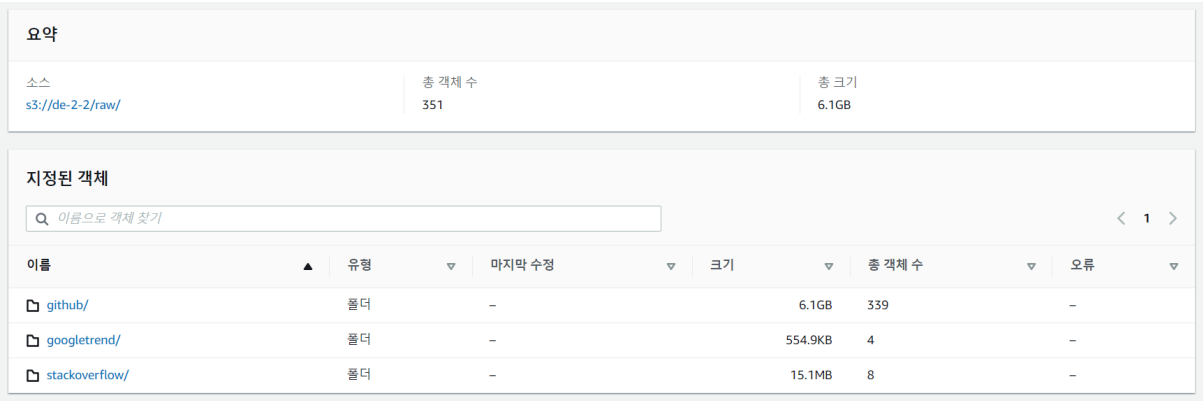
2-5. Raw Data 적재

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

> **수집한 데이터**
> **(23.09.02 기준)**



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

✔️ 3) 정제 및 적재

3-1. 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",
}
```

Amazon EMR 클러스터 구성

1. 클러스터당 처리해야할 데이터의 크기가 100mb 이내이기 때문에 워커 노드를 사용하지 않고 단일 마스터 노드로 구성
2. cluster의 log 정보는 S3에 저장
3. instance의 종류는 AWS에서 추천하는 m5.xlarge 사용
4. Cluster 생성 후 데이터 정제 Step으로 넘어가게끔 설정

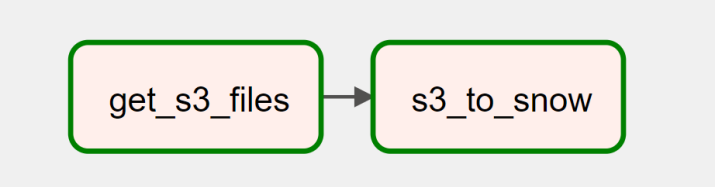
3-2. Amazon EMR 구현

Amazon EMR workflow

Amazon EMR 클러스터 목록

1. raw data를 정제하고 Snowflake에 적재할 Table양식에 맞게 변환함
2. S3 저장 및 Snowflake 적재 시 데이터의 속도 및 크기를 고려하여 parquet 형식으로 S3에 저장
3. 데이터 정재 및 변환 중 발생할 비용을 최소화하기 위해 EMR Cluster를 생성 후 정제 및 적재 과정이 끝나면 EMR Cluster를 종료하는 과정을 자동화 함
    a. EmrJobFlowSensor를 통해 cluster 생성 실패 또는 오류 발생 시 cluster가 종료하도록 설정

3-2. S3-To-Snowflake

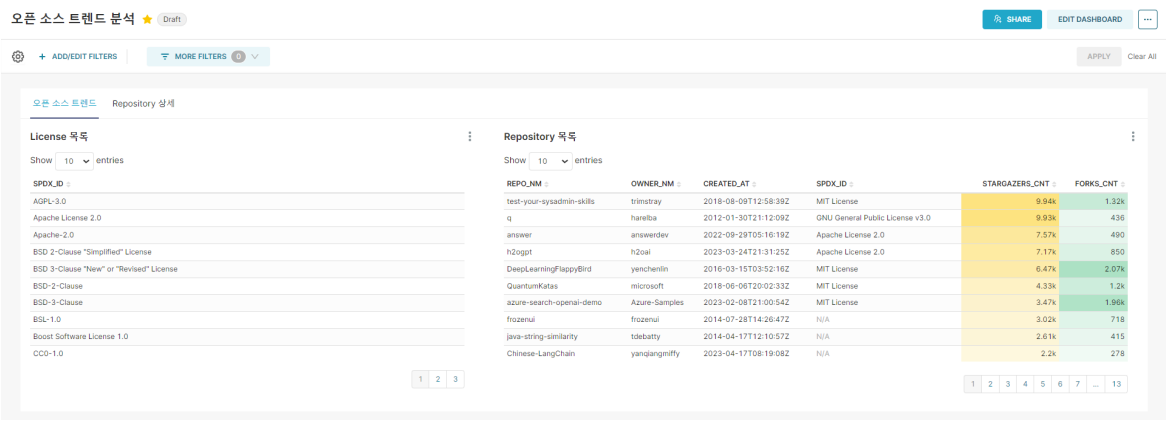


매일 00:15분에 Amazon S3 Analytics의 `execution_date` 기준 전날의 데이터를 불러와 한 파일씩 Upsert를 하여 Snowflake의 테이블에 데이터를 적재한다.

- Temporary Table을 사용하여 S3 Analytics의 특정 날짜 Prefix의 Parquet 파일을 하나씩 불러온다.
- 이후 Merge Into 쿼리문을 통해 `ID` 및 `collected_at` 필드 비교를 하여 Upsert를 진행한다.

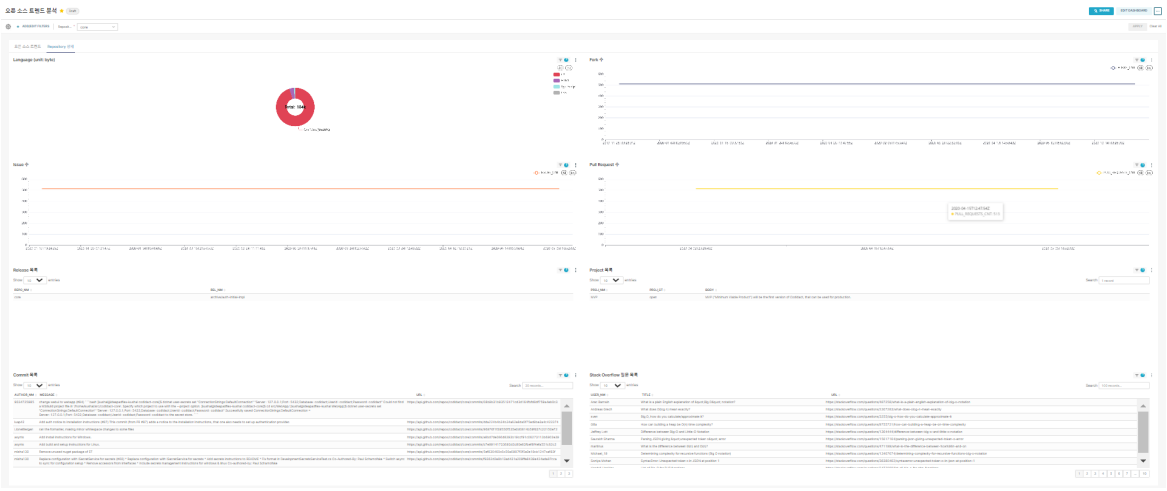
✔️ 4) 시각화 (Preset)

1. 오픈 소스 트랜드



- license 목록 및 star/fork 수 기준으로 정렬된 repository 목록을 확인할 수 있습니다.

2. Repository 상세



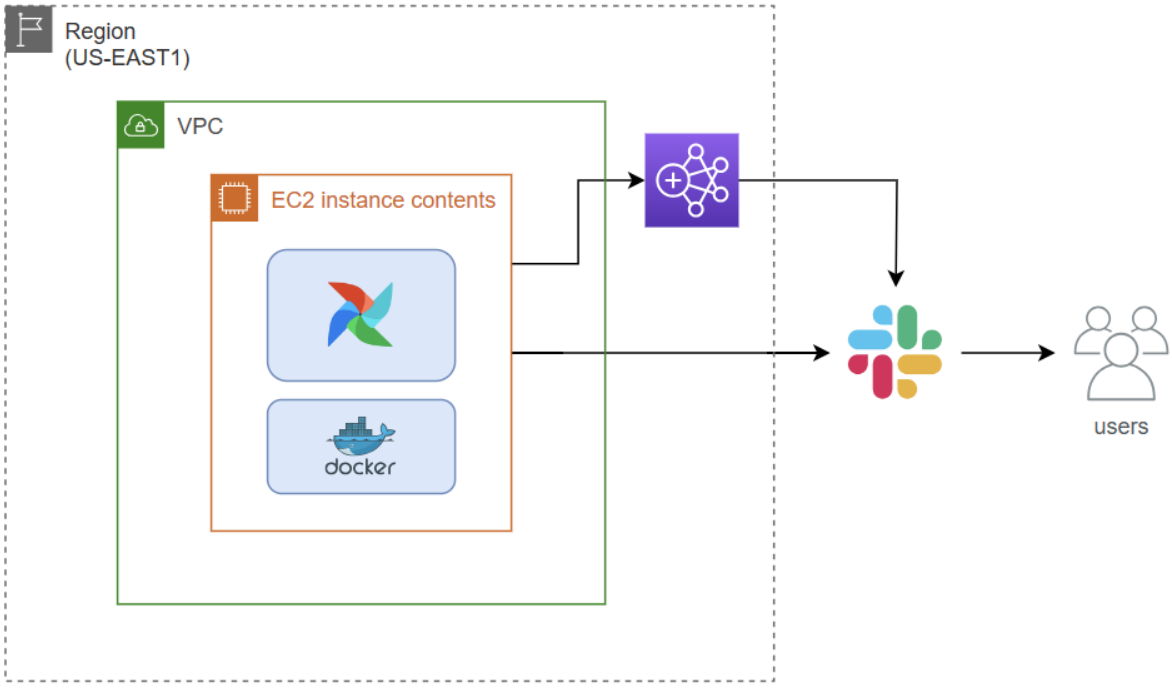
Repository와 연관된 정보들을 확인할 수 있습니다.

- 작성된 언어 분포
- issue, pull request, fork 수 변화
- release, project, commit 목록
- Stack Overflow 질문 목록

✔️ 5) 모니터링

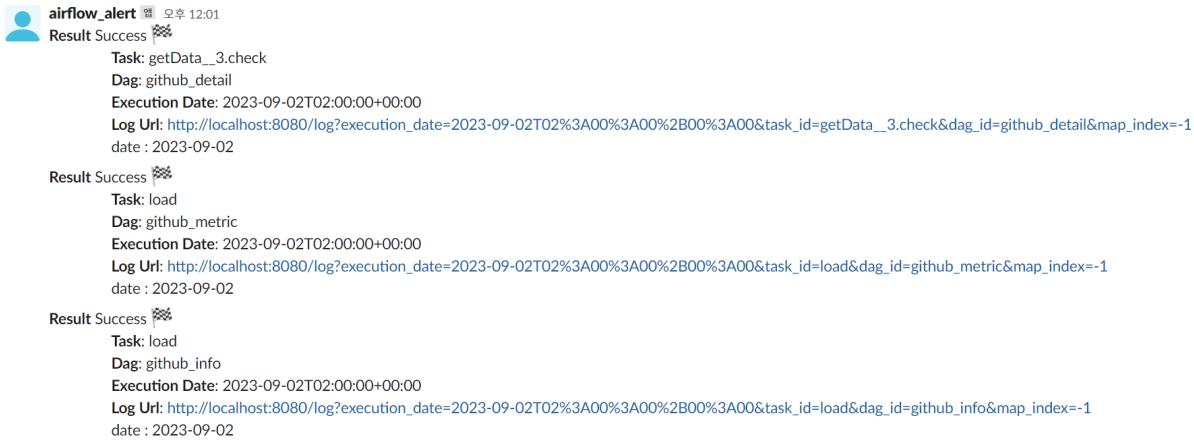
1. 에러

a. slack 알림 기능



Airflow Slack 알림

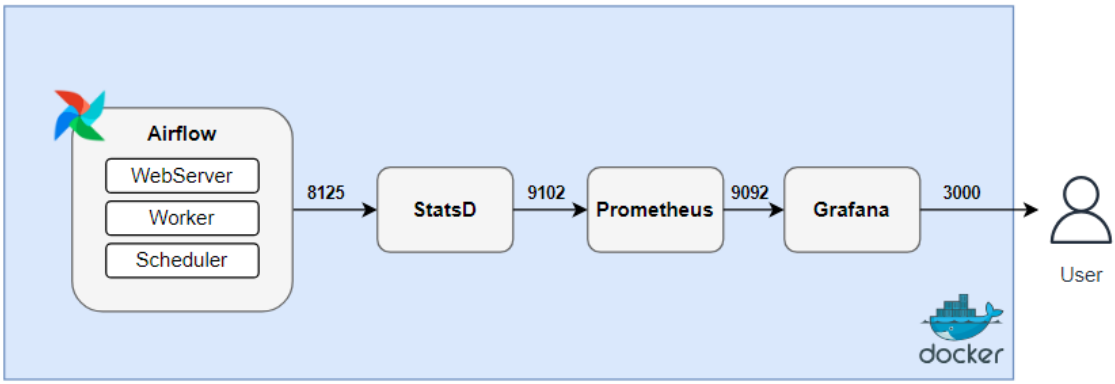
- EMR Data Process 과정을 실시간으로 확인하고 에러 발생 시 빠르게 대응하기 위해 **Slack 알림** 기능 구현



Slack 알림

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

> **b. Grafana 대시 보드 구성**
> **StatsD + Prometheus + Grafana**



Airflow Monitoring

`StatsD`로 수집한 Metric 정보를 시계열 데이터 베이스 `Prometheus`에 저장하고 대시보드 툴인 `Grafana`로 Prometheus의 Metric 데이터를 시각화한다.

- Airflow는 내부적으로 **StatsD**를 통해 Metric을 외부로 전송 가능함
- **Prometheus** 는 PromQL 언어로 쿼리할 수 있는 시계열 데이터베이스

  ⇒ Airflow의 수집기 StatsD가 Push한 Metric 정보를 Promethus는 Pull하여 가져옴

https://prod-files-secure.s3.us-west-2.amazonaws.com/6dcbf0f0-7d29-4be0-a940-8bc3b7e73757/edfa902a-8ccd-4a0a-8653-409f49bd4212/%EB%85%B9%ED%99%94_2023_09_02_18_42_44_539.mp4

Grafana 실행영상

https://drive.google.com/file/d/1VsNsWHem9AHrZ_-9xWtYgcblDy-9Ri8I/view?usp=sharing

Grafana 대시보드 접속 시 Scheduler 상태 및 Task, Job, DAG 상태 (성공/실패 개수 및 기간별 조회)를 차트를 통해 확인 가능하다.

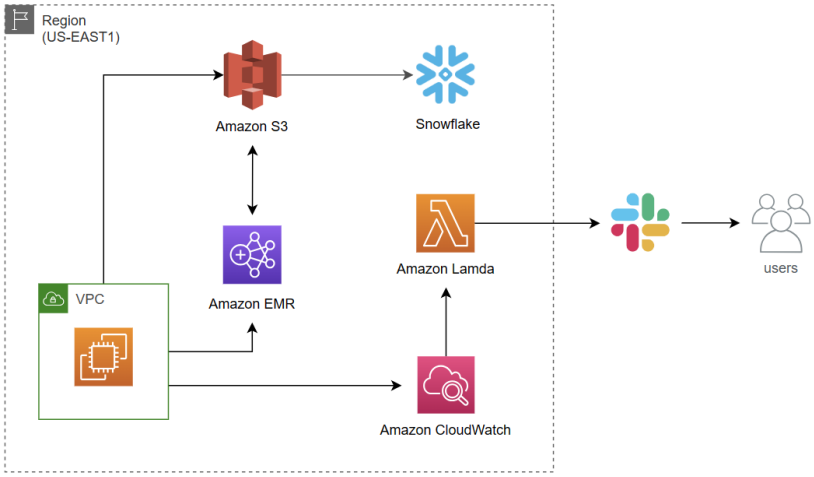
2. Usage

a. Amazon CloudWatch

> 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.

Airflow Web UI - Scheduler 이슈

- ec2에 할당된 용량을 모두 사용할 시 airflow가 동작하지 않음
- 이를 방지하기 위해 할당된 용량의 85% 도달 시 slack 알림이 가도록 함



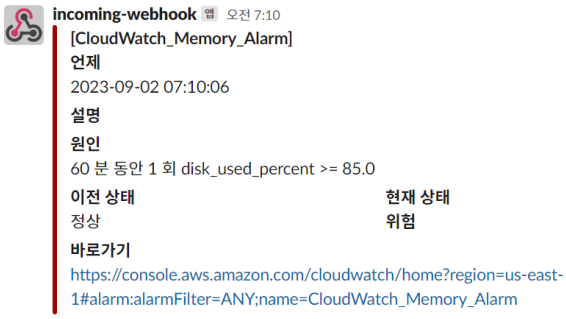
Amazon Clouwatch의 Usage 알림

- Amazon CloudWatch 경보를 트리거로 설정하여 경보 내용이 Slack으로 전달 하게끔 lambda함수를 생성함

▼ lambda 함수

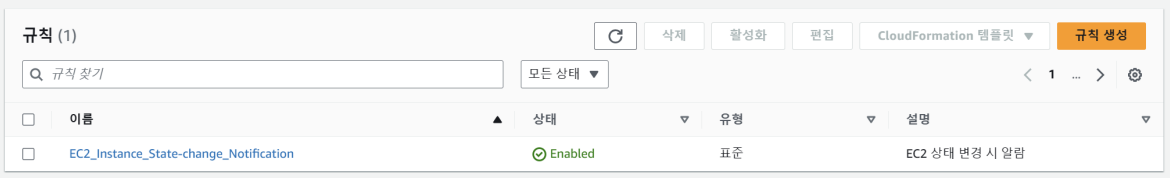
```
import os
import json
import http.client
from datetime import datetime, timedelta
from urllib.parse import urlencode, quote_plus, urlparse

from dateutil import parser
import http.client as http_client

status_colors_and_message = {
    "ALARM": {"color": "danger", "message": "위험"},
    "INSUFFICIENT_DATA": {"color": "warning", "message": "데이터 부족"},
    "OK": {"color": "good", "message": "정상"}
}

comparison_operator = {
    "GreaterThanOrEqualToThreshold": ">=",
    "GreaterThanThreshold": ">",
    "LowerThanOrEqualToThreshold": "<=",
    "LessThanThreshold": "<"
}

def export_region_code(arn):
    return arn.replace("arn:aws:cloudwatch:", "").split(":")[0]

def create_link(data):
    region_code = export_region_code(data["AlarmArn"])
    encoded_name = quote_plus(data["AlarmName"])
    return f"https://console.aws.amazon.com/cloudwatch/home?region={region_code}#alarm:alarmFilter=ANY;name={encoded_name}"

def get_cause(data):
    trigger = data["Trigger"]
    evaluation_periods = trigger["EvaluationPeriods"]
    minutes = trigger["Period"] // 60

    if "Metrics" in trigger:
        return build_anomaly_detection_band(data, evaluation_periods, minutes)

    return build_threshold_message(data, evaluation_periods, minutes)

def build_anomaly_detection_band(data, evaluation_periods, minutes):
    metrics = data["Trigger"]["Metrics"]
    metric = next(metric["MetricStat"]["Metric"]["MetricName"] for metric in metrics if metric["Id"] == "m1")
    expression = next(metric["Expression"] for metric in metrics if metric["Id"] == "ad1")
    width = expression.split(",")[1].replace(")", "").strip()

    return f"{evaluation_periods * minutes} 분 동안 {evaluation_periods} 회 {metric} 지표가 범위(약 {width}배)를 벗어났습니다."

def build_threshold_message(data, evaluation_periods, minutes):
    trigger = data["Trigger"]
    threshold = trigger["Threshold"]
    metric = trigger["MetricName"]
    operator = comparison_operator[trigger["ComparisonOperator"]]
```

```
    return f"{evaluation_periods * minutes} 분 동안 {evaluation_periods} 회 {metric} {operator} {threshold}"


def to_yyyymmddhhmmss(time_string):
    if not time_string:
        return ""

    utc_date = parser.parse(time_string)
    kst_date = utc_date + timedelta(hours=9)

    return kst_date.strftime("%Y-%m-%d %H:%M:%S")


def post_slack(message, slack_url):
    parsed_url = urlparse(slack_url)
    headers = {
        "Content-Type": "application/json"
    }
    conn = http.client.HTTPSConnection(parsed_url.netloc)
    conn.request("POST", parsed_url.path, json.dumps(message), headers)
    response = conn.getresponse()
    conn.close()



def lambda_handler(event, context):
    webhook = os.environ["webhook"]
    sns_message = json.loads(event["Records"][0]["Sns"]["Message"])
    post_data = build_slack_message(sns_message)
    post_slack(post_data, webhook)

def build_slack_message(data):
    new_state = status_colors_and_message[data["NewStateValue"]]
    old_state = status_colors_and_message[data["OldStateValue"]]
    execute_time = to_yyyymmddhhmmss(data["StateChangeTime"])
    description = data["AlarmDescription"]
    cause = get_cause(data)

    return {
        "attachments": [
            {
                "title": f"[{data['AlarmName']}]",
                "color": new_state["color"],
                "fields": [
                    {"title": "언제", "value": execute_time},
                    {"title": "설명", "value": description},
                    {"title": "원인", "value": cause},
                    {"title": "이전 상태", "value": old_state["message"], "short": True},
                    {"title": "현재 상태", "value": f"*{new_state['message']}*", "short": True},
                    {"title": "바로가기", "value": create_link(data)}
                ]
            }
        ]
    }
```



Usage 관련 Slack 알람

b. Amazon Eventbridge



Amazon Eventbridge 규칙

```
{
  "source": ["aws.ec2"],
  "detail-type": ["EC2 Instance State-change Notification"],
  "detail": {
    "state": ["shutting-down", "stopping", "stopped"],
    "instance-id": ["i-03401a842fb0939d0"]
  }
}
```

Amazon Lambda 함수

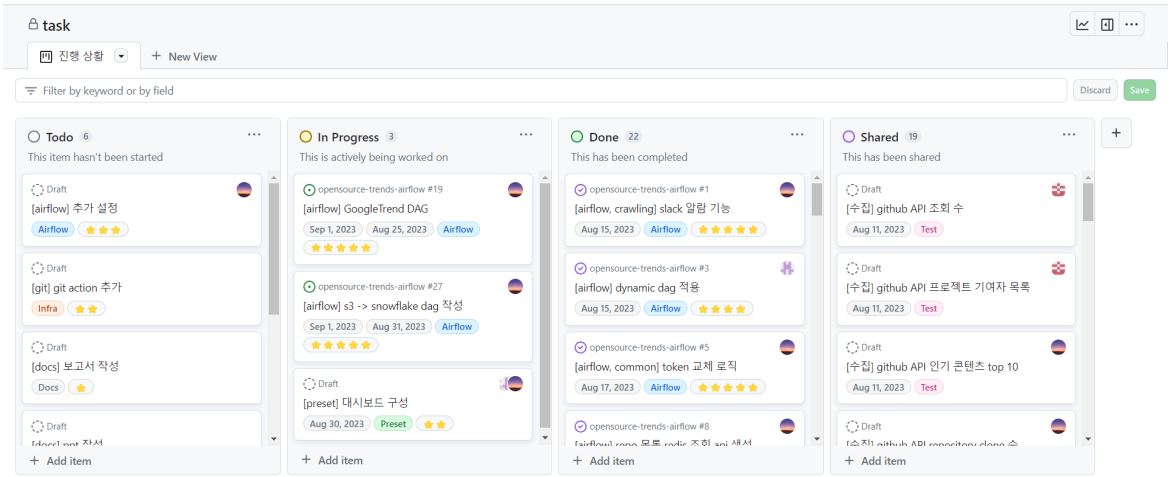
- EC2의 상태가 “shutting-dowm”, “stopping”, “stopped” 시 Slack 알림이 가도록 Amazon EventBridge 규칙을 생성함
- Amazon EventBridge 규칙을 트리거로 설정하여 EC2 상태 변경 시 Slack으로 전달 하게끔 Amazon Lambda함수를 생성함

▼ lambda 함수

```
import json, random
import httplib
from botocore.vendored import requests

def lambda_handler(event, context):
    slack_url = 'https://hooks.slack.com/services/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'

    payloads = {
            "attachments":[{
                "pretext": "AWS EC2 관리 봇",
                "color":"#0099A6",
                "fields": [{
                    "title": "EC2 상태가 변경되었습니다.",
                    "value": json.dumps(event["detail"]["state"]), #EC2 state 정보
                    "short": False
                }]
            }]
        }
    response = requests.post(
        slack_url, data=json.dumps(payloads),
        headers={'Content-Type': 'application/json'}
    )
    if response.status_code != 200:
        raise ValueError(
            'Request to slack returned an error %s, the response is:\n%s'
            % (response.status_code, response.text)
        )
```

06. 커뮤니케이션 전략

1. 태스크 관리
    - 우선 순위가 높은 태스크를 할당하여 관리할 수 있도록 다음과 같이 구성하였습니다.



a. 우선 순위

- 시급도, 시작일, 종료일 등을 통해 작업의 우선 순위를 확인할 수 있습니다.

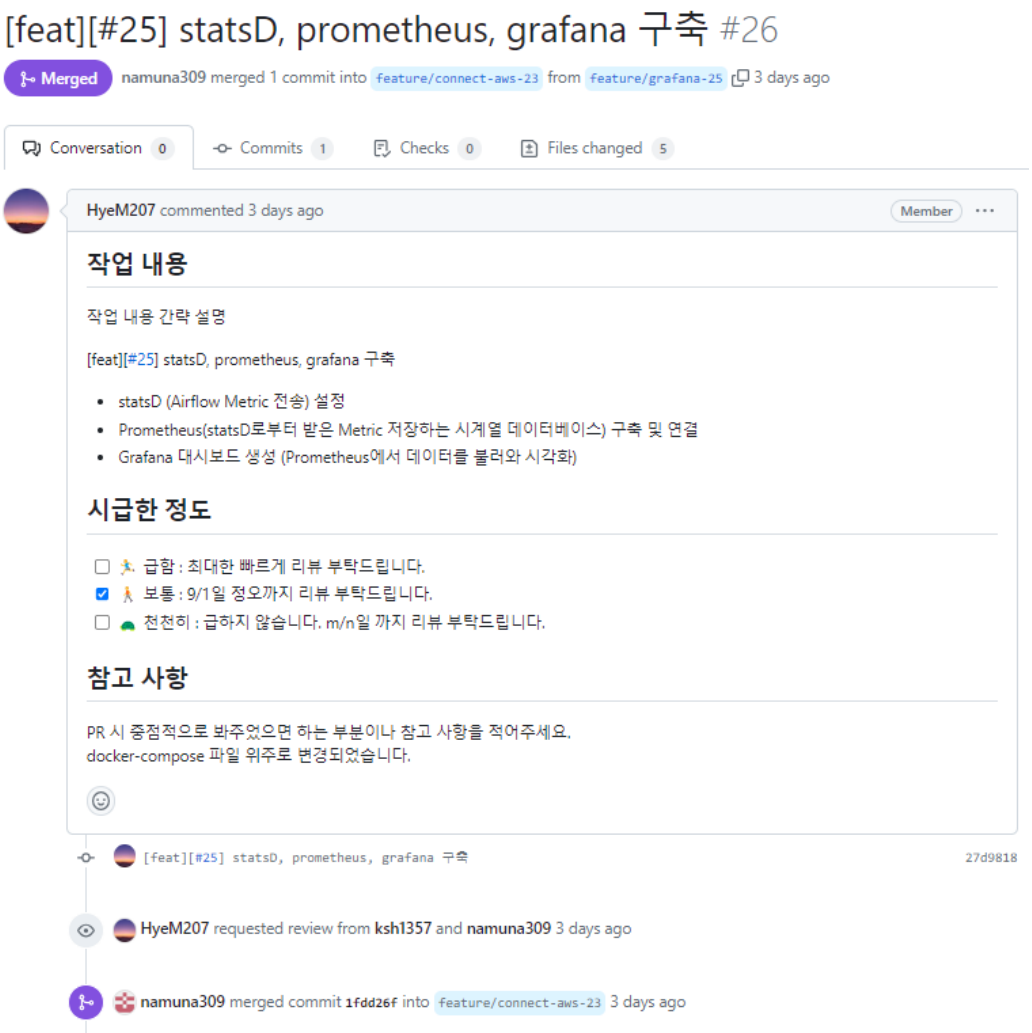
b. 작업의 상태

- 작업의 상태를 4단계로 관리하여, 다른 사람의 진행 중인 작업을 파악할 수 있습니다.

Todo	시작 전인 작업
In Progress	진행 중인 작업
Done	완료된 작업
Shared	코드 리뷰 혹은 내용 공유를 완료한 작업

2. 브랜치 관리 전략

- task를 issue로 변경하여 브랜치를 생성하여 해당 task의 작업을 수행했습니다.
- 테스트가 완료된 경우, pull request를 요청하여 1명 이상의 승인을 받아 develop 브랜치에 merge하였습니다.



3. 커뮤니케이션
   - 현재 진행 상황, 이슈 등 공유할 사항이 있으면 팀 채널에 메세지를 보내거나, 메세지로 공유하기 힘든 경우 허들을 통해 소통하였습니다.
   - 오전 스크럼을 통해 현재 진행 상황을 공유하였고, 그 외 추가적으로 회의가 필요한 경우 따로 시간을 정해 논의했습니다.
4. 문서 정리
   - 회의록, 자료 조사, 공유할 내용 등을 노션에 문서로 정리하였습니다.

07. 참여자 정보 및 각 역할

이름	종류	역할
김상희	Airflow	• web crawling을 통한 데이터 수집 DAG 개발 • Github 데이터 수집 DAG 개발 • 데이터 정합성 확인 DAG 개발
	Preset	• 대시보드 및 차트 생성
김혜민	Airflow	• Docker-Compose 환경 구축 • Github 데이터 수집 DAG 수정 및 GoogleTrend 데이터 수집 DAG 개발 • Plugin 개발 (Slack, Pytrend 호출 등) • Airflow Monitoring 환경 구축 (Grafana)
	Snowflake	• 데이터 마트 구축 DAG 개발 (Amazon S3 → Snowflake)
	Flask	• Redis 연결 Flask REST-API 개발

이름	종류	역할
남윤아	Airflow	• Plugin 개발(Slack, AWS S3 read and write 등) • StackOverflow 데이터 수집 DAG 개발 • EMR Data Process 자동화 DAG 개발
	Spark	• Raw Data 정제 및 S3 적재 모듈 생성
	AWS	• AWS EC2, EMR 관리 • EC2 관리를 위한 CloudWatch 경보, EventBridge 규칙, Lambda 함수 개발