

# MLOps 를 위한 클라우드 환경에서의 데이터파이프라인 구축 및 ETL 작업 자동화 구축

컴퓨터공학과 2018102238 조인화

## 요 약

고성능 GPU 의 등장과 다량의 데이터 학습 등으로 인공지능의 성능이 폭발적으로 높아짐에 따라 다양한 분야에 머신 러닝이 적용되고 있다. 또한 클라우드 컴퓨팅은 몇 년 전 인공지능 등과 함께 큰 주목을 받으며 등장한 이후, 최근 국내외에서 4 차 산업혁명의 핵심기술로 급부상하고 있다. 따라서 본 문서는 머신 러닝 개발 환경을 위한 클라우드 환경에서의 데이터파이프라인 및 ETL 작업 자동화 구축을 제안하고 구현한다.

## 1. 서론

### 1.1. 연구배경

머신 러닝이란 인간과 같은 학습 능력을 기계를 통해 구현하는 방법이다. 머신 러닝은 다양한 알고리즘 기법을 적용하는 여러 유형의 머신 러닝 모델로 구성된다. 머신 러닝의 알고리즘은 대규모 데이터 셋에서 패턴과 상관관계를 찾고 분석을 토대로 최적의 의사결정과 예측을 수행하도록 훈련된다. 머신 러닝 애플리케이션은 적용을 통해 개선되며 이용 가능한 데이터가 증가할수록 더욱 정확해 진다.

머신 러닝에서의 데이터 수집은 기계 학습 모델을 구축하는 데 매우 중요한 역할을 한다. 데이터 수집은 모델의 성능을 결정하는 중요한 요소 중 하나이기 때문이다. 빅데이터의 출현으로 인해 머신 러닝의 잠재력을 최대치로 끌어내는 것에 대한 실현 가능성이 높아지고 있는 가운데, 빅데이터를 처리하기 위해 병렬처리 기법 등을 이용한 접근 방법이 활용되고 있다.

최근 머신 러닝, 인공지능, 데이터 분석을 위한 클라우드 기반 툴이 증가하고 있다. 클라우드 컴퓨팅은 몇 년 전 빅데이터 등과 함께 큰 주목을 받으며 등장한 이후, 최근 국내외에서 4 차 산업혁명의 핵심기술로 급부상하고 있다. 세계 클라우드 시장은 현재도 성장 중이며, 시장조사기관 마다 규모의 차이는 다소 있으나 공통적으로 높은 성장률을 예측한다.

클라우드 기술을 사용한다면 어느 컴퓨터에서나 중앙 저장소에 로그인할 수 있으며 어디에서나 작업을 할 수 있다. 클라우드에서는 백업과 동기화를 작업으로 처리해주기 때문에 모든 작업이 간소화된다. 또한 클라우드는 사용한 만큼 비용을 지불하기 때문에 컴퓨팅 자원에 대한 시간과 비용을 절감할 수 있다.

이에 본 연구에서는 여러 빅데이터 오픈소스 툴과 컨테이너 기술, NHN Cloud 의 클라우드 상품을 활용하여 머신 러닝 개발 환경을 위한 데이터파이프라인 및 ETL 작업 자동화를 구축을 제안하고자 한다.

## 1.2. 연구목표

본 연구에서는 클라우드 컴퓨팅 기술을 기반으로 Docker, Kubernetes, Airflow 등의 기술을 활용하여 MLOps(Machine Learning Operations)를 위한 클라우드 네이티브 데이터 파이프라인 자동화 및 ETL 작업 자동화를 제안하고 구현하는 것을 목표로 한다. 머신 러닝 개발 환경을 위한 데이터 수집 과정을 자동화하는 것이 주 목적인 만큼 다음 3 가지 연구를 목표로 삼았다.

첫 번째, 컨테이너 오케스트레이션 플랫폼을 클라우드 컴퓨팅 기술과 접목하여 자동 확장 가능하도록 한다. CPU, 메모리, 네트워크 등의 지표를 기반으로 애플리케이션의 인스턴스 수를 자동으로 조절하여야 하며, 더 이상 필요하지 않은 인스턴스를 제거하여 클라우드 사용 비용을 절감한다.

두 번째, 사용자는 여러 배치성 데이터 ETL 작업들을 한 곳에서 관리할 수 있어야 하며 각 작업의 결과를 웹 UI 를 통해 시각적으로 확인할 수 있어야 한다. 배치 작업에 문제가 발생할 경우에는 사용자에게 알림을 보내어 신속한 대응이 가능하도록 한다.

세 번째, 데이터 ETL 과정을 위한 코드는 빈번한 수정이 일어날 수 있다. 이에 Jenkins 웹 훅을 활용하여 Airflow 의 DAG 와 GitHub 를 연동하여 git-sync 를 구현할 수 있도록 한다.

## 2. 관련연구

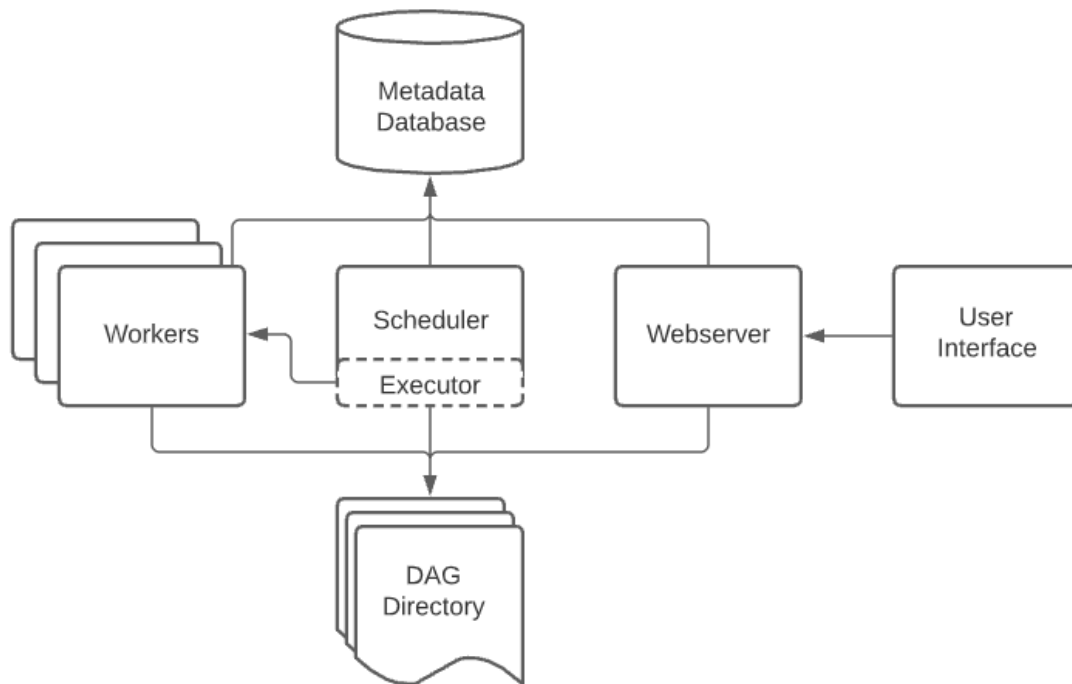
### 2.1. NHN Cloud

NHN Cloud는 오픈 스택 기반 클라우드 서비스로 유연한 클라우드 인프라를 제공한다. 인프라에 기반한 플랫폼 중심 클라우드이며 인프라, 콘텐츠, 분석, 게임, 보안, 알림 메시지, 기타 애플리케이션을 운영할 때 필요한 각종 기능을 제공한다. NHN Kubernetes Service, NHN Container Registry 등 컨테이너 관련 기능과 여러 AI 서비스 기능 또한 제공하고 있다. 또한 사용한 만큼 요금을 지불하는 pay as you go 클라우드이므로 합리적인 가격으로 클라우드 서비스를 사용할 수 있다. 몇몇 보조 서비스들을 무료로 사용 가능하기도 한다. NHN Cloud는 웹 브라우저 상에서 간단한 클릭만으로 서비스를 사용할 수 있어 누구나 쉽게 사용할 수 있는 클라우드 서비스로 평가받고 있다.

### 2.2. Airflow

Airflow는 Airbnb에서 개발한 데이터 파이프라인 자동화 오픈소스 플랫폼이며, 현재는 아파치

재단에서 관리중이다. Airflow는 DAG(Directed Acyclic Graph)라는 개념을 사용하여 작업 간의 의존성을 정의하고, 이를 바탕으로 지정한 일정에 따라 작업을 실행하고 모니터링한다. Airflow는 다양한 데이터 소스로부터 데이터를 추출하고, 변환하며, 적재할 수 있는 풍부한 연결성을 가진다. 또한 Airflow는 Python으로 작성되어 있기 때문에, 개발자들은 Python을 사용하여 각 작업을 편리하게 커스텀 하여 구성할 수 있다.



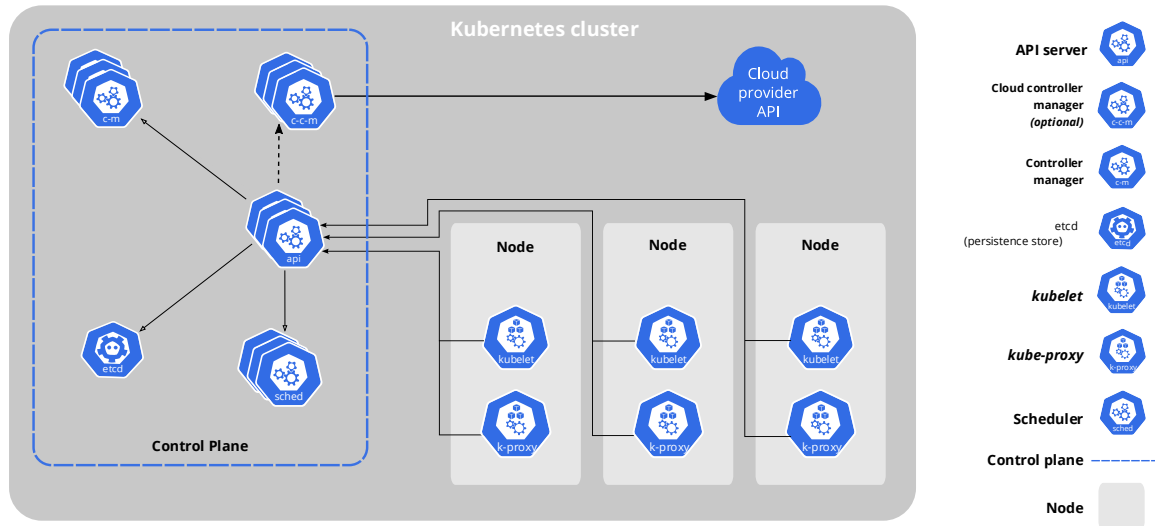
- Webserver: Airflow UI를 제공하며, DAG 및 작업 실행 정보를 시각화한다.
- Scheduler: 정의된 DAG를 실행하는 역할을 한다.
- Metadata Database: Airflow 설정과 DAG 상태 정보 등을 저장한다.
- Executor: Airflow가 작업을 실행하는 방식을 결정한다. LocalExecutor, CeleryExecutor, KubernetesExecutor 등이 있다.
- Worker: 실제 Task를 실행하는 주체이다.

주기적으로 실행하는 것은 Linux Cron과 유사하지만, Cron은 하나의 애플리케이션만 지정 가능하고 각 태스크를 연결하는 것이 불가능한 반면 Airflow는 복잡한 작업을 쉽게 구성할 수 있고, 각 태스크 별로 연결할 수 있어 전체 파이프라인 구성에 적합하다.

## 2.3. Kubernetes

Kubernetes는 컨테이너 오케스트레이션 도구로, 구글에서 개발된 오픈소스 프로젝트이다. Kubernetes는 컨테이너화 된 애플리케이션을 관리하고 배포하기 위한 플랫폼으로, 대규모 분산

시스템에서 일어날 수 있는 문제들을 자동으로 처리하고, 애플리케이션을 수평 확장하며, 서비스 디스커버리 및 로드밸런싱 등을 제공한다.



- 컨테이너 오케스트레이션: 컨테이너를 배치하고 스케줄링 하며, 컨테이너가 죽었을 때 자동으로 다시 시작하는 등의 기능을 제공한다.
- 서비스 디스커버리: 컨테이너화 된 애플리케이션의 인스턴스들을 자동으로 발견하고 로드밸런싱을 수행하여 애플리케이션의 가용성과 안정성을 높인다.
- 스토리지 오케스트레이션: 컨테이너화 된 애플리케이션에서 사용하는 스토리지 리소스를 관리하고, 스토리지 볼륨을 동적으로 할당하여, 데이터의 안정성과 지속성을 보장한다.
- 자동 스케일링: 컨테이너화 된 애플리케이션의 부하량에 따라 자동으로 인스턴스 수를 늘리거나 줄여서, 애플리케이션의 가용성과 성능을 최적화 한다.

Kubernetes는 대규모 분산 시스템에서 사용될 수 있도록 설계되었으며, 클라우드, 온프레미스, 하이브리드 등 다양한 환경에서 사용될 수 있다. 현재 많은 국내외 기업들이 Kubernetes를 클라우드 네이티브 애플리케이션 개발 및 운영에 활용하고 있다.

## 2.4. 빅데이터 병렬 처리

### 2.4.1. Apache Hadoop

Hadoop 은 하나의 컴퓨터를 Scale up 하여 데이터를 처리하는 대신 적당한 성능의 범용 컴퓨터 여러 대를 Scale out 한 후, 클러스터화 하여 큰 크기의 데이터를 클러스터에서 병렬로 동시에 처리한다. 이를 통해 처리 속도를 높이는 것을 목적으로 두는 오픈소스 프레임워크로 아래와 같은 모듈로 구성된다.

- Hadoop Common: Hadoop 의 다른 모듈을 지원하기 위한 공통 컴포넌트 모듈
- Hadoop MapReduce: 대용량 데이터 처리를 분산 병렬 컴퓨팅에서 처리하기 위한 목적으로 제작된 소프트웨어 프레임워크
- Hadoop HDFS: 분산 저장을 처리하기 위한 모듈, 여러 개의 서버를 하나의 서버처럼 묶어서 데이터를 저장
- Hadoop YARN: 병렬 처리를 위한 클러스터 자원관리 및 스케줄링 담당
- Hadoop Ozone: Hadoop 을 위한 오브젝트 저장소

Hadoop 은 시스템을 중단하지 않고 장비의 추가가 용이하며 일부 장비에 장애가 발생하더라도 전체 시스템 사용성에 영향이 적다는 장점이 있다. 그렇지만 데이터를 DISK 기반으로 처리하기 때문에 데이터 처리 시간 외에도 read/write 연산에 추가 시간이 소요된다. 또한 동일 데이터에 대해서 작업할 때, 매번 read 연산이 필요하다.

#### 2.4.2. Apache Spark

Spark 는 대규모 데이터 처리를 위한 오픈소스 분산 컴퓨팅 시스템이다. Spark 는 클러스터 컴퓨팅 환경에서 데이터 처리 작업을 병렬로 처리하여 대규모 데이터 집합을 빠르게 처리할 수 있다. 대량의 데이터 집합을 대상으로 인메모리 기반 데이터 프로세싱을 지원한다. Hadoop 의 MapReduce 는 작업의 중간 결과를 디스크에 써서 IO 로 인해 작업 속도에 제약이 생기는 반면 Spark 는 메모리에 중간결과를 저장하여 반복 작업의 처리 효율이 높다. 또한 Spark 는 데이터 처리 작업을 단순화하며, 컴퓨팅 리소스를 효율적으로 활용할 수 있도록 도와준다.

Spark 의 핵심 기능 중 하나로 Resilient Distributed Datasets(RDD)가 있다. RDD 는 다수의 노드에 분산된 데이터를 메모리에 로드하여 처리할 수 있는 추상화된 데이터셋이다. 이러한 RDD 는 불변하며, 분산처리 환경에서 안정적으로 동작할 수 있다.

Spark 는 다양한 언어에서 사용 가능하다. Java, Scala, Python, R 등에서 Spark 를 지원한다. Spark 는 Hadoop, Cassandra, HBase 등의 다양한 데이터 소스와 연동될 수 있어 확장성이 높다. 또한 기계 학습 작업을 위한 MLlib, 그래프 처리를 위한 GraphX, 실시간 데이터 처리를 위한 Spark Streaming 등의 다양한 라이브러리를 제공한다. 또한 다양한 클러스터 매니저를 지원하여 YARN, Mesos, Kubernetes 등 다양한 클러스터에서 작동이 가능하다.

### 3. 프로젝트 내용

#### 3.1. 시나리오

1. CI/CD 를 통해 사용자의 코드를 자동으로 ETL 처리 서버에 적용한다.
2. Airflow 는 코드를 분석하여 Task 간 의존성 관계를 파악하고 배치 작업으로 등록한다.
3. 배치 작업 수행 시 적절하게 Task 를 쪼개어 여러 Worker 로 분산 처리 한다.
4. Worker 는 Kubernetes Pod 형태로 동적으로 생성되고 Task 가 완료되면 Pod 를 삭제된다.
5. Worker 를 동적으로 생성하고 제거하는 과정에서 컴퓨팅 자원 지표를 분석하여 자동으로 필요한 인스턴스를 생성하거나 필요하지 않은 인스턴스를 제거한다.

#### 3.2. 요구사항

##### 3.2.1. 지속적 통합 및 지속적 배포(CI/CD)에 대한 요구사항

- 코드가 업데이트되면 자동으로 코드를 배포하여 빠르게 사용자에게 적용 가능해야 한다.
- 사용자가 필요한 코드를 GitHub 에 Push 하면 서버에 자동으로 적용되어야 한다.
- Jenkins 웹 혹은 Git Push 를 감지하고 자동으로 코드를 업데이트할 수 있어야 한다.

##### 3.2.2. 클라우드 컴퓨팅 자원 자동 운영에 대한 요구사항

- CPU, 메모리, 네트워크 등의 컴퓨팅 자원 지표를 분석할 수 있어야 한다.
- 해당 지표를 기반으로 애플리케이션의 인스턴스 수를 자동으로 조절하여야 한다.
- 더 이상 필요하지 않은 인스턴스를 제거하여 클라우드 비용을 절감할 수 있어야 한다.

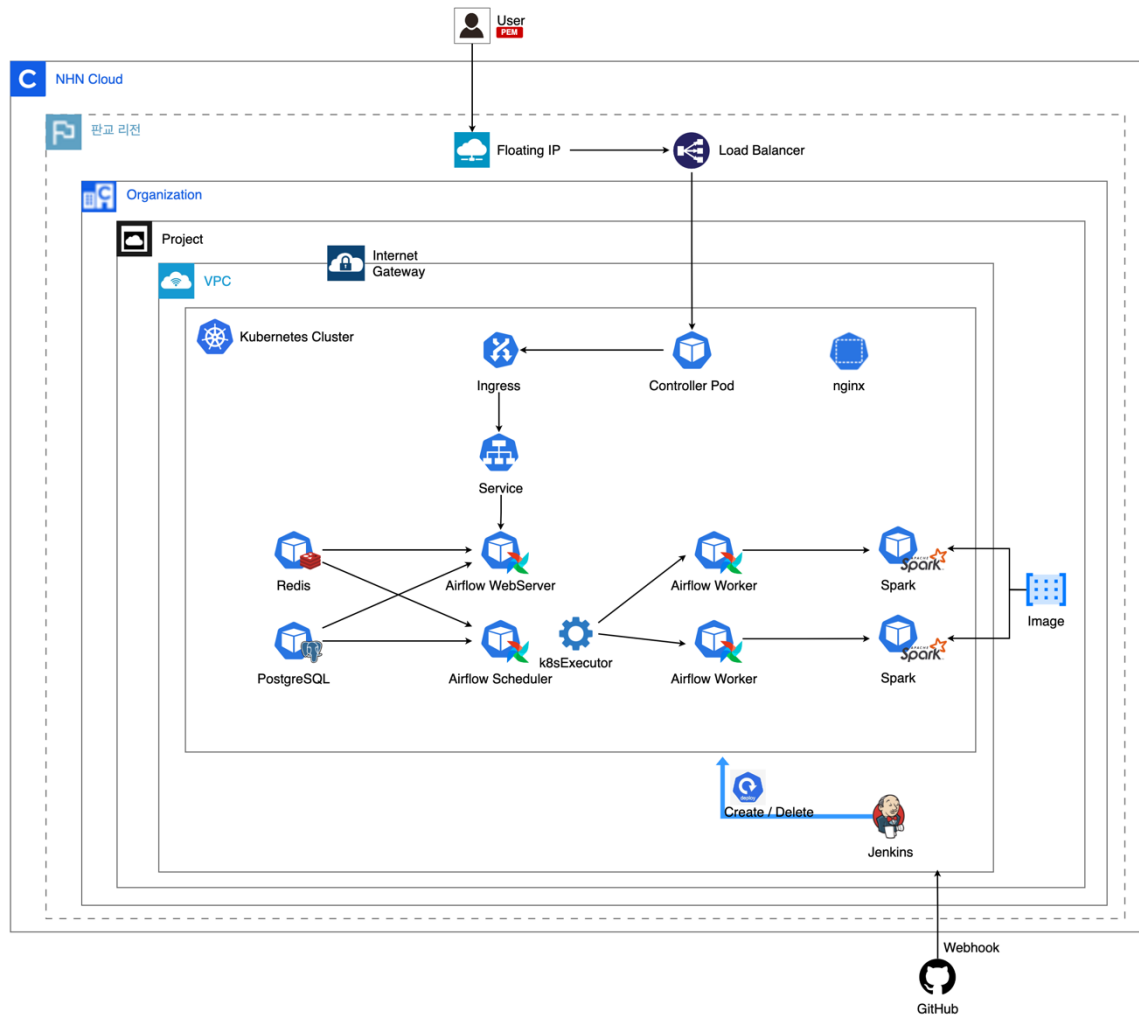
##### 3.2.3. 데이터 병렬 처리에 대한 요구사항

- 매우 큰 데이터 셋에 대해 하나의 Worker 에서 모든 처리를 진행하게 된다면 매우 오랜 시간이 걸릴 뿐더러, 큰 부하가 생기게 되고 이는 서버 다운의 원인이 될 수 있다.
- 적절한 크기의 데이터 셋으로 분해하여 여러 Worker 로 분산처리 가능해야 한다.
- Worker 는 필요할 때에만 동적으로 생성되었다가 제거되어야 한다.

#### 3.3. 시스템 아키텍처

편집기 클라이언트에서 업로드한 이미지를 Node.js 이미지 저장 서버로 전송한다. Node.js 이미지 저장 서버에서 이미지 데이터를 AWS S3 에 업로드하여 사용자가 업로드한 이미지를 보관할 수 있도록 하고 사용자가 원할 때 언제든지 이미지를 다시 불러올 수 있도록 한다.


### 3.3.1. 시스템 아키텍처



[그림 1] 시스템 아키텍처

### 3.3.2. Spark 인터페이스

Apache Spark 웹 UI 를 사용하여 실행 중인 Spark 애플리케이션을 모니터링하고 디버그할 수 있습니다. Spark UI 를 사용하면 각 작업에 대해 다음을 확인할 수 있습니다.

**Spark Master at spark://10.200.179.153:7077**

URL: spark://10.200.179.153:7077  
REST URL: spark://10.200.179.153:6066 (cluster mode)  
Alive Workers: 2  
Total Cores: 12 / 12  
    ◦ default: 12 / 12  
Total Memory: 2.0 GB / 26.4 GB  
    ◦ default: 2.0 GB / 26.4 GB  
Applications: 1 Running, 1 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

Worker Id	Address	State	Cores	Memory
<a href="#">worker-20180105193611-10.200.179.153-58829</a>	10.200.179.153:58829	ALIVE	Total: 6 / 6 • default: 6 / 6	Total: 1024.0 MB / 13.2 GB • default: 1024.0 MB / 13.2 GB
<a href="#">worker-20180105193616-10.200.179.152-51071</a>	10.200.179.152:51071	ALIVE	Total: 6 / 6 • default: 6 / 6	Total: 1024.0 MB / 13.2 GB • default: 1024.0 MB / 13.2 GB

**Running Applications**

Application ID	Name	Workpool	Cores	Memory per Executor	Submitted Time	User	State	Duration
<a href="#">app-20180111194527-0001</a>	(kill) <a href="#">Spark shell</a>	default	12	1024.0 MB	2018/01/11 19:45:27	anonymous	RUNNING	19 s


**Completed Applications**

Application ID	Name	Workpool	Cores	Memory per Executor	Submitted Time	User	State	Duration
<a href="#">app-20180105193837-0000</a>	SparkSQL::10.200.179.152	default	0	1024.0 MB	2018/01/05 19:38:37	anonymous	FINISHED	1.0 min

[그림 2] Spark UI

### 3.3.3. Airflow 인터페이스

Apache Airflow 웹 UI 를 사용하여 DAG 를 실행할 수 있으며, 언제 실행되고 어떻게 실행되는 지 모니터링할 수 있다. 또한 각 Task 의 세부 정보와 실행 로그를 확인할 수 있으며 Graph 기능을 통해 DAG 의 동작 순서를 가시적으로 확인할 수 있다.

Airflow

DAGs   Datasets   Security   Browse   Admin   Docs

17:22 KST (+09:00)   AA

**DAGs**

All 45   Active 0   Paused 46

Filter DAGs by tag   Search DAGs   Auto-refresh

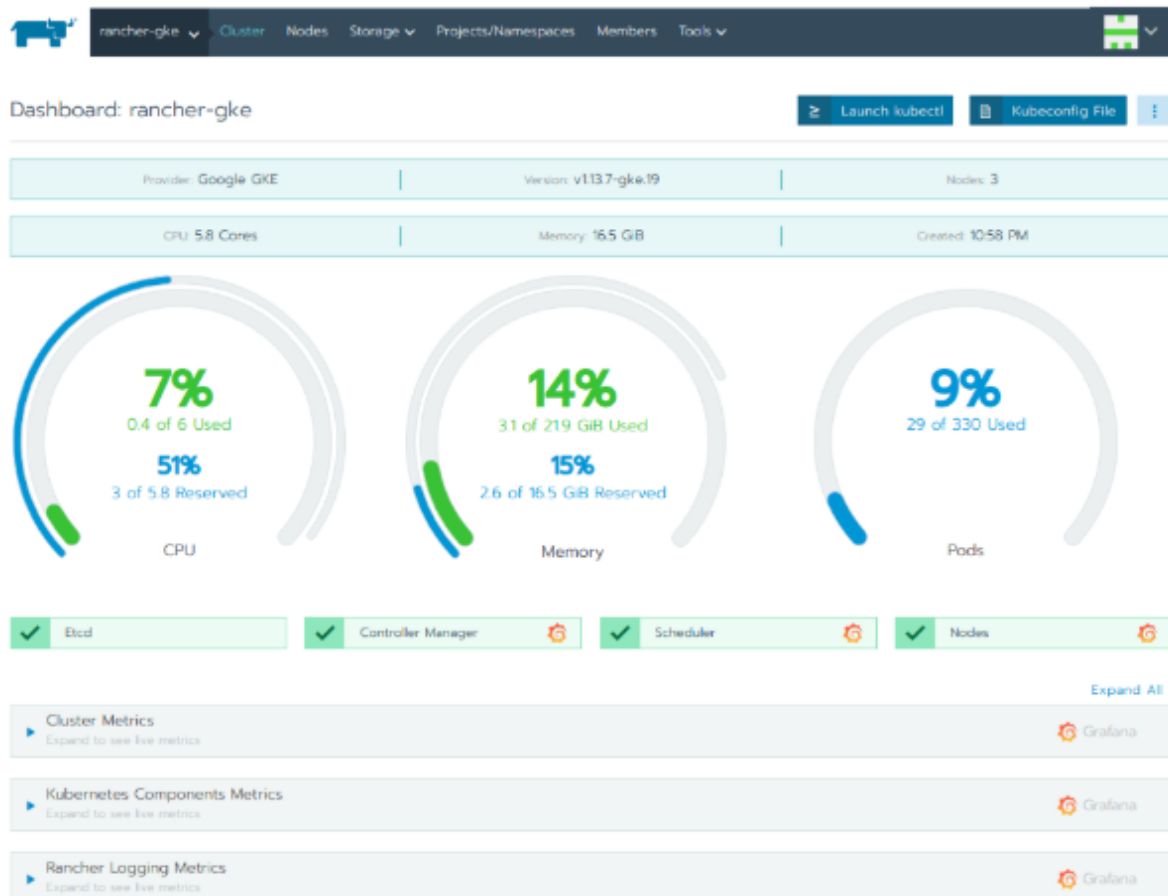
DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
<a href="#">dataset_consumes_1</a> consumes dataset-scheduled	airflow	○○○○	Dataset		On s3://dag1/output_1.txt	○○○○○○○○○○○○○○○○○○○○	▶ □	...
<a href="#">dataset_consumes_1_and_2</a> consumes dataset-scheduled	airflow	○○○○	Dataset		0 of 2 datasets updated	○○○○○○○○○○○○○○○○○○○○	▶ □	...
<a href="#">dataset_consumes_1_never_scheduled</a> consumes dataset-scheduled	airflow	○○○○	Dataset		0 of 2 datasets updated	○○○○○○○○○○○○○○○○○○○○	▶ □	...
<a href="#">dataset_consumes_unknown_never_scheduled</a> dataset-scheduled	airflow	○○○○	Dataset		0 of 2 datasets updated	○○○○○○○○○○○○○○○○○○○○	▶ □	...
<a href="#">dataset_produces_1</a> dataset-scheduled produces	airflow	○○○○	@daily		2023-03-17, 09:00:00	○○○○○○○○○○○○○○○○○○○○	▶ □	...
<a href="#">dataset_produces_2</a> dataset-scheduled produces	airflow	○○○○	None			○○○○○○○○○○○○○○○○○○○○	▶ □	...

[그림 3] Airflow UI



### 3.3.4. Rancher 인터페이스

Rancher 는 퍼블릭 클라우드와 온프레미스 상의 쿠버네티스의 설치 및 통합 관제를 가능하게 해준다. Rancher UI 를 통해 여러 쿠버네티스 클러스터를 한눈에 관리 및 모니터링할 수 있다. 기본적인 Node 정보 및 상세 정보, API 서버의 지표들을 확인할 수 있다. Grafana 와 연동 또한 가능하다.



[그림 4] Rancher UI

## 3.4. 구현

### 3.4.1. Spark Master

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: spark-master
spec:
  replicas: 1
  selector:
    matchLabels:
      component: spark-master
  template:
    metadata:
      labels:
        component: spark-master
    spec:
      containers:
        - name: spark-master
          image: spark-hadoop:3.2.0
          command: ["/spark-master"]
          ports:
            - containerPort: 7077
            - containerPort: 8080
          resources:
            requests:
              cpu: 100m
```

[그림 5] Spark Master 배포 파일

### 3.4.2. Spark Worker

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: spark-worker
spec:
  replicas: 2
  selector:
    matchLabels:
      component: spark-worker
  template:
    metadata:
      labels:
        component: spark-worker
    spec:
      containers:
        - name: spark-worker
          image: spark-hadoop:3.2.0
          command: ["/spark-worker"]
          ports:
            - containerPort: 8081
          resources:
            requests:
              cpu: 100m
```

[그림 6] Spark Worker 배포 파일

### 3.4.2. Airflow

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: worker
  kubeadmConfigPatches:
  - |
    kind: JoinConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "node=worker_1"
  extraMounts:
  - hostPath: ./data
    containerPath: /tmp/data
- role: worker
  kubeadmConfigPatches:
  - |
    kind: JoinConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "node=worker_2"
  extraMounts:
  - hostPath: ./data
    containerPath: /tmp/data
- role: worker
  kubeadmConfigPatches:
  - |
    kind: JoinConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "node=worker_3"
  extraMounts:
  - hostPath: ./data
    containerPath: /tmp/data
```

[그림 7] Airflow Cluster 생성 파일

## 4. 프로젝트 결과

프로젝트 설계 시 계획했던 요구사항 중 데이터 병렬 처리, 클라우드 컴퓨팅 자원 자동에 대한 기능을 일부 구현하였고 고도화 중에 있다. Kubernetes 환경 구성에 대한 내용을 모두 .yaml 파일을 통해 코드로 관리하였다. 이후 Jenkins 를 도입하여 지속적 통합 및 지속적 배포(CI/CD)에 대한 요구사항도 구현할 예정이다.

기존 EMR, YARN 기반 Spark 데이터 분석 컴퓨팅은 Spark 실행을 위한 구성요소들이 잘 갖춰져 있고 제각기 다른 사이즈 작업에 대해서도 잘 실행할 수 있었다. 하지만 오래 실행되는 Spark job 의 driver 가 인스턴스 종료로 인해 죽게 되고, Spark SQL 서버가 알 수 없는 원인으로 종종 hang 이 걸리는 문제점이 있었다. 해당 프로젝트는 Spark job 을 Kubernetes 에서 띄우고 실행하며, Auto Scaling 기능을 통해 기존 문제를 해결하였다.

## 5. 결론 및 기대효과

본 연구에서 제안하는 솔루션은 사용자가 컴퓨팅 자원이나 인스턴스간 네트워크 설정, 배치 작업 자동화 등에 대한 노력을 줄이고 필요한 개발에만 집중할 수 있는 플랫폼을 제공한다. 편리하고 직관적인 웹 UI 를 함께 제공함으로써 손쉽고 편리하게 배치 작업을 등록하고 데이터 처리 결과를 확인할 수 있다.

머신 러닝에 대한 도입이 활성화되고 있는 요즘, 빅데이터 수집 및 처리에 대한 수요 또한 급증하고 있기 때문에 이를 필요로 하는 사용자에게 유용한 도구가 될 것이다.

## 6. 참고문헌

- [1] 박지훈, 「빅데이터 시스템의 데이터 수집 및 저장에 관한 연구」, 『2017 년 추계학술발표대회 논문집 제 24 권 제 2 호』, 2017.
- [2] 인포매티카, 「빅 데이터(Big Data)의 폭발적 증가 - 빅 데이터를 큰 비즈니스 기회로 연결시키는 Informatica 9.1 플랫폼」, 2011.
- [3] 임수중, 민옥기, 「빅데이터 활용을 위한 기계학습 기술 동향」, 2012.
- [4] hs\_seo 저, 빅데이터 - 하둡, 하이브로 시작하기
- [5] hs\_seo 저, 빅데이터 - 스칼라, 스파크로 시작하기
- [6] 신호승, 강성원, 이지현, 「확장형 실시간 데이터 파이프라인 시스템 아키텍처 설계」, 『정보과학회논문지』, 2015.
- [7] 류우석, 「스파크를 이용한 머신러닝의 분산 처리 성능 요인」, 『한국전자통신학회 논문지』, 2021.