


1단계(3차년도) 주요 결과물

(과제명) 대규모 분산 에너지 저장장치 인프라의 안전한 자율운영
및 성능 평가를 위한 지능형 SW 프레임워크 개발
(과제번호) 2021-0-00077

- 결과물명 : 데이터 주도형 가상 시뮬레이션 시스템(SW)
- 작성일자 : 2023년 11월 20일

과학기술정보통신부 SW컴퓨팅산업원천기술개발사업
“1단계(3차년도) 주요 결과물” 로 제출합니다.

수행기관	성명/직위	확인
한국전자기술연구원	최효섭/책임연구원	

정보통신기획평가원장 귀하



데이터 주도형 가상 시뮬레이션 시스템(SW)

프로젝트

대규모 분산 에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한
지능형 SW 프레임워크 개발

사 용 권 한

본 문서에 대한 서명은 한국전자기술연구원 내부에서 본 문서에 대하여
수행 및 유지관리의 책임이 있음을 인정하는 것임.

본 문서는 작성, 검토, 승인하여 승인된 원본을 보관한다.

작성자 : 박진원

일자 : 2023. 11. 20

검토자 : 김창우


일자 : 2023. 11. 21

승인자 : 최효섭

일자 : 2023. 11. 21


제 · 개정 이력

버전	변경일자	제·개정 내용	작성자
1.0	2023-11-20	최초 등록	박진원

	데이터 주도형 가상 시뮬레이션 시스템(SW)	
	프로젝트	대규모 분산 에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한 지능형 SW 프레임워크 개발

목 차

1. 개요	1
2. 안전AI 분석 SW컨테이너 기반 가상 시뮬레이션 시스템	3
3. 데이터 적응형 분석 조합 시스템	10

	데이터 주도형 가상 시뮬레이션 시스템(SW)	
	프로젝트	대규모 분산 에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한 지능형 SW 프레임워크 개발

1. 개요

□ 목적

- 본 명세서의 목적은 대규모 분산에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한 “지능형 안전SW 프레임워크”를 개발하기 위해 프레임워크의 개요 및 개발 방법을 기록한다.

□ 범위


- 본 설계서는 “지능형 안전SW 프레임워크”의 개발에 필요한 MLOps(Kubeflow) 기반의 데이터 주도형 시뮬레이션 시스템과 이를 활용하기 위한 데이터 적응형 분석 조합 시스템(SW)에 대해서 설명한다.

□ 시스템 개요



〈확장된 MLOps(Kubeflow) 기반 지능형 안전 SW 프레임워크〉

- 가상 환경 및 실증 상황에 필요한 안전AI분석엔진의 SW 구조화 및 에너지 인프라의 운영·관리 목표에 부합하기 위해 안전AI분석엔진 SW의 컨테이너화로 가상 시뮬레이션 시스템을 구축하고 컨테이너 복합 연계 기반 분석·학습 파이프라인 처리 구조 개발로 통합형 프레임워크 기반 안전SW 검증 체계 확보

	데이터 주도형 가상 시뮬레이션 시스템(SW)	
	프로젝트	대규모 분산 에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한 지능형 SW 프레임워크 개발

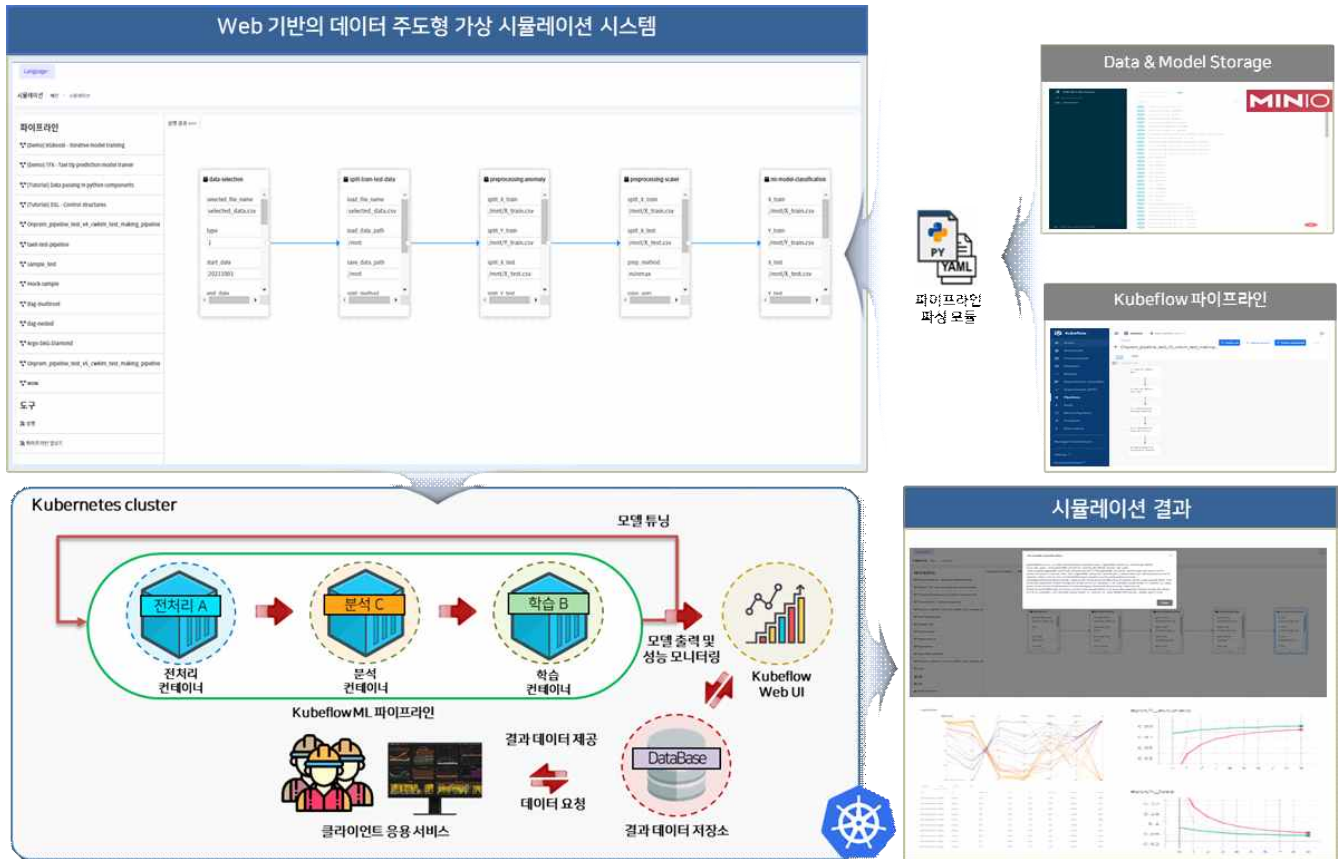
□ 관련 계획 및 표준

o 본 설계서는 아래 계획 및 표준을 참고한다.

구분	식별자	세부 내용	설명
설계서	ISO/IEC 9126	9126-1 (품질 모델) 9126-2 (외부 품질) 9126-3 (내부 품질) 9126-4 (사용 품질)	품질 특성 및 측정기준을 제시 소프트웨어의 기능성, 신뢰성, 사용성, 효율성, 유지보수 용이성, 이식성
	ISO/IEC 14598	14598-1 (개요) 14598-2 (계획과 관리) 14598-3 (개발자용 프로세스) 14598-4 (구매자용 프로세스) 14598-5 (평가자용 프로세스) 14598-6 (평가 모듈)	ISO 9126에 따른 제품 평가 표준: 반복성, 공정성, 객관성, 재생산성
	ISO/IEC 12119	소프트웨어 패키지 -제품설명서 -사용자문서 -프로그램과 데이터	패키지 SW 품질 요구사항 및 테스트
계획서	-	지능형 안전 SW 프레임워크 요구사항 정의서	대규모 분산 에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위해 개발된 솔루션을 가상 환경에서 시뮬레이션 분석을 도와주는 프 레임워크 요구사항을 정의한 문서

2. 안전AI 분석 SW컨테이너 기반 가상 시뮬레이션 시스템

□ 가상 시뮬레이션 시스템 기술 개발



< 데이터 주도형 가상 시뮬레이션 시스템 >

- 사용자 친화적 인터페이스 개발을 위한 설정파일 기반 파이프라인 기술 개발
 - Kubeflow SDK를 사용하여 개발된 파이프라인을 웹 기반의 데이터 주도형 가상 시뮬레이션 시스템에 연동하기 위한 모듈 개발
 - 아티팩트 저장소 연동: MinIO는 객체 저장소로서, Kubeflow 파이프라인의 아티팩트(결과물)를 저장하는 데 사용
 - 파이프라인 정보 조회: Kubeflow 파이프라인에서 생성한 파이프라인 정보를 효율적으로 조회

파이프라인 연동 코드
<pre> import requests import os import kfp from dotenv import load_dotenv from pprint import pprint from git import Repo import yaml </pre>

```

from minio import Minio
load_dotenv()

USERNAME = os.environ.get("USERNAME")
PASSWORD = os.environ.get("PASSWORD")
NAMESPACE = os.environ.get("NAMESPACE")
HOST = os.environ.get("HOST")

S3_ENDPOINT = os.environ.get("S3_ENDPOINT")
S3_ACCESS_KEY = os.environ.get("S3_ACCESS_KEY")
S3_SECRET_KEY = os.environ.get("S3_SECRET_KEY")

KFP_URL = "https://kubeflow.keti-dp.re.kr"
PIPELINE_NAME = "Onprem_pipeline_test_v4_cwkim_test_making_pipeline"

def download_pipeline_details_from_minio(endpoint, access_key, secret_key, pipeline_id):
    # MinIO 클라이언트를 생성
    minioClient = Minio(endpoint,
                        access_key=access_key,
                        secret_key=secret_key,
                        secure=False)

    # 파이프라인 정의가 저장된 버킷과 객체의 이름을 지정
    bucket_name = 'mlpipeline'
    pipeline_file = 'pipelines/' + pipeline_id

    # MinIO에서 파이프라인 파일을 가져옵니다.
    data = minioClient.get_object(bucket_name, pipeline_file)

    # 데이터를 바이트 스트림에서 문자열로 변환
    data_str = ''
    for d in data.stream(32*1024):
        data_str += d.decode('utf-8')

    # 문자열을 파이썬 딕셔너리로 변환
    data_dict = yaml.safe_load(data_str)

    # 딕셔너리를 YAML 형식으로 저장
    with open(PIPELINE_NAME + '.yaml', 'w') as file_data:
        yaml.dump(data_dict, file_data, default_flow_style=False, sort_keys=False)

```

```
def commit_and_push_file(repo_path, file_path, commit_message):
    #try:
    repo = Repo(repo_path)
    repo.git.add(file_path)
    repo.git.commit('-m', commit_message)
    origin = repo.remote(name='origin')
    origin.push()
    print("File committed and pushed successfully.")

    # TODO: exception 발생할 수 있는 것들로 변경
    # except Exception as e:
    #     print(f"Error: {str(e)}")

if __name__ == "__main__":
    session = requests.Session()
    response = requests.get(f"{KFP_URL}/apis/v1beta1/runs")
    headers = {
        "Content-Type": "application/x-www-form-urlencoded",
    }
    user_data = {"login": USERNAME, "password": PASSWORD}
    session.post(response.url, headers=headers, data=user_data)
    session_cookie = session.cookies.get_dict()["authservice_session"]

    # Connect My KubeFlow Client
    client = kfp.Client(host=f"{HOST}/pipeline",
                       namespace=f"{NAMESPACE}",
                       cookies=f"authservice_session={session_cookie}",
    )

    # pipeline 목록
    pipelines = client.list_pipelines()
    pipe_dict = {}
    for pipe in pipelines.pipelines:
        pipe_name = pipe.name
        pipe_dict[pipe_name] = pipe.id

    pprint(pipe_dict)

    # TODO: pipeline 이름을 argument로 받을것인지 고민하기
    try:
        pipe_id = pipe_dict[PIPELINE_NAME]
        #pipe_info = client.pipelines.get_pipeline(pipe_id) # issue1. metadata 정보는 주지 않고, 기본
```


적인 정보

except KeyError:

pprint(f“존재하지 않는 Pipeline 입니다. Pipeline의 이름을 확인해주세요.“)

MinIO의 pipelines에 접근하여, pipeline 정보 yaml로 다운로드

```
download_pipeline_details_from_minio(endpoint=S3_ENDPOINT,
                                     access_key=S3_ACCESS_KEY,
                                     secret_key=S3_SECRET_KEY,
                                     pipeline_id=pipe_id)
```

git에 yaml 파일 업로드

```
#commit_and_push_file('/mnt/c/Users/KETI/git/MyProject',      'kubeflow/taeil-test-pipeline.yaml',
'Update')
```

o 공개SW 서비스를 위한 기계학습 워크플로우 연동 처리 기술 개발

- 웹 기반의 파이프라인 DAG(Directed Acyclic Graph) 표시를 위한 DAG 파싱 기술 개발
 - DAG 순서: 파이프라인 정보를 포함하는 YAML파일의 entrypoint, dependencies, condition 등을 기반으로 DAG 순서 파싱
 - DAG 파라미터: 각 DAG에 들어가는 파라미터 값 파싱



〈웹 기반의 데이터 주도형 가상 시뮬레이션 시스템〉

DAG 파싱 코드

```
import yaml
import re
from pprint import pprint

def read_pipeline(file_path):
```

```

with open(file_path, 'r') as f:
    yaml_data = yaml.load(f, Loader=yaml.FullLoader)
return yaml_data

def convert_to_dict(args):
    result_dict = {}
    key = None
    for arg in args:
        if arg.startswith("--"):
            if key is not None:
                if len(result_dict[key]) == 1:
                    result_dict[key] = result_dict[key][0]
                key = arg
                result_dict[key] = []
            else:
                result_dict[key].append(arg)
    if key is not None and len(result_dict[key]) == 1:
        result_dict[key] = result_dict[key][0]
    return result_dict

def condition_parse_string(input_str):
    # Regular expression to match the task name
    pattern = r"{{tasks\.(.*?)\.outputs\.parameters\..*}}"
    match = re.search(pattern, input_str)
    if match:
        return match.group(1)
    else:
        return None

if __name__ == "__main__":
    #pipeline = read_pipeline("./Onprem_pipeline_test_v4_cwkim_test_making_pipeline.yaml")
    #pipeline = read_pipeline("./pipeline1.yaml")
    #pipeline = read_pipeline("./control_structures.yaml")
    pipeline = read_pipeline("./[Demo] XGBoost - Iterative model training.yaml")

    # pipeline의 componenet 조회
    pipeline_dict = {}
    entrypoint_tasks = []
    entrypoint = pipeline["spec"]["entrypoint"]

    for component in pipeline["spec"]["templates"]:
        dag_list = []
    
```

```
#pprint(component["name"])

# 1-1: entrypoint 기준으로 dependencies 획득
if component["name"] == entrypoint:
    for tasks in component["dag"]["tasks"]:
        if tasks["name"] not in pipeline_dict:
            pipeline_dict[tasks["name"]] = {}

    entrypoint_tasks.append(tasks["name"])

    try:
        pipeline_dict[tasks["name"]]["dependencies"] = tasks["dependencies"]
    except KeyError:
        #print(f"dag-tasks 안에 컴포넌트들에서 dependencies라는 key값이 존재하지
않습니다.")
        continue

else:
    # 1-2: DAG 기록
    try:
        for tasks in component["dag"]["tasks"]:
            if component["name"] not in pipeline_dict:
                pipeline_dict[component["name"]] = {}
            dag_list.append(tasks["name"])
            pipeline_dict[component["name"]]["dag"] = dag_list
    except KeyError:
        pass

# 1-3: entrypoint 와 같은 이름 이외에 DAG이 존재할 경우 (dependencies, condition 파싱)
try:
    for tasks in component["dag"]["tasks"]:
        if tasks["name"] not in pipeline_dict:
            pipeline_dict[tasks["name"]] = {}

        try:
            pipeline_dict[tasks["name"]]["dependencies"] = tasks["dependencies"]
        except KeyError:
            pass

        # Condition에 따른 DAG 표현을 위함
        if "when" in tasks:
            #task_name = condition_parse_string(tasks["when"])
            pipeline_dict[tasks["name"]]["condition"] = True
```

```

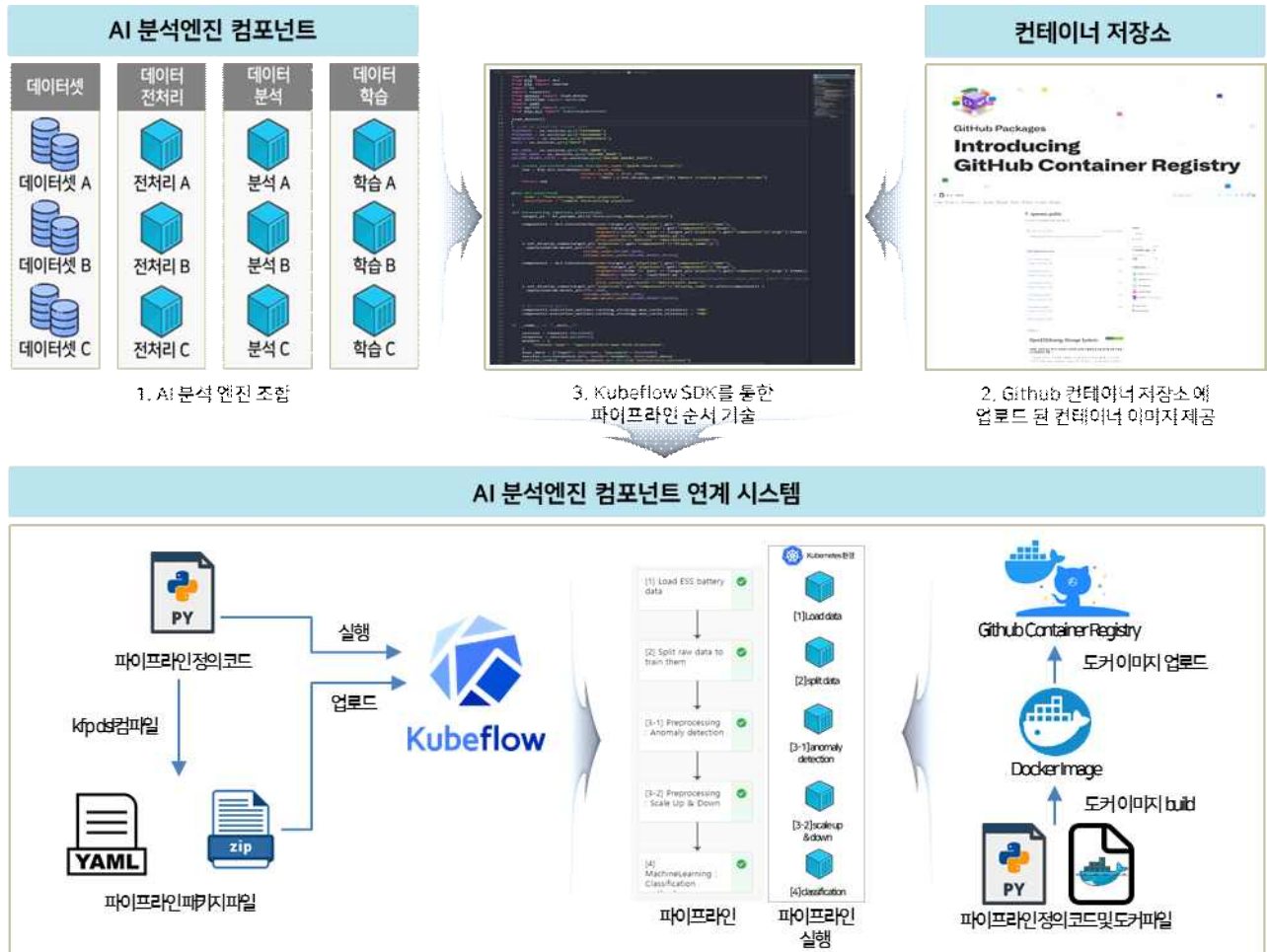
except KeyError:
    pass # dag의 key error 예외처리

# 1-4: task들의 args 획득
for component in pipeline["spec"]["templates"]:
    try:
        pipeline_dict[component["name"]]["args"] =
convert_to_dict(component["container"]["args"])
    except KeyError:
        pass

```

3. 데이터 적응형 분석 조합 시스템

□ 가상 시뮬레이션 시스템 기반의 분석 조합 시스템



< 데이터 적응형 분석 조합 시스템 >

- o Github packages를 활용한 컴포넌트 저장 및 관리
 - 컴포넌트 구성 코드와 버전 관리를 통한 일관성 유지
 - Github의 README를 사용하여 컴포넌트 사용방법, 예제, 버전 기록 등 문서화
- o ESS/EV 안전 분석 AI 분석엔진 컴포넌트 개발
 - 컨소시엄내 안전 분석 AI 분석엔진에 대한 컴포넌트 개발
 - 최종 목표인 선택형 이전에 고정형으로 분석 파이프라인과 컴포넌트를 관리 및 실행할 수 있는 YAML 파일 및 SDK 방법 개발

파이프라인과 컴포넌트 기술된 YAML 예
<pre># Component "get_dataset_10minute" : &get_dataset_10minute { "name": "Get_data_10minute", "image": "ketidp/openess:keti.get_data_10minute-0.2",</pre>

```

"args": {
  "--site" : 1,
  "--bank" : 1,
  "--rack": 1,
  "--date": "2022-10-30 00:00:00",
  "--sel": "min_vol",
  "--save_data_path" : "/mnt/"
},
"display_name" : "Get_dataset_10minute"
}

"forecasting_10minute" : &forecasting_10minute {
  "name": "forecasting_10minute",
  "image": "ketidp/openess:keti.forecasting_10minute-0.2",
  "args": {
    "--model": "linear",
    "--sel": "min_vol",
    "--data_load_path" : "/mnt/"
  },
  "display_name" : "Forecasting_10minute"
}

# Pipeline

## 10 분 후 예측-v0.1
"forecasting_10minute_pipeline": {
  "user_pipeline_func" : "forecasting_10minute_pipeline",

  "pipeline" : {
    "component1" : *get_dataset_10minute,
    "component2" : *forecasting_10minute
  }
}

```

파이프라인과 컴포넌트가 기술된 YAML을 활용하여 작성한 코드

```

import kfp
import kfp.components as comp
from kfp import dsl
from kfp import onprem
import os

import requests

```

```

from dotenv import load_dotenv
from random import randrange
from datetime import datetime
import yaml
from pprint import pprint
import logging

load_dotenv()

# Load My KubeFlow Client info
USERNAME = os.environ.get("USERNAME")
PASSWORD = os.environ.get("PASSWORD")
NAMESPACE = os.environ.get("NAMESPACE")
HOST = os.environ.get("HOST")

PVC_NAME = os.environ.get("PVC_NAME")
VOLUME_NAME = os.environ.get("VOLUME_NAME")
VOLUME_MOUNT_PATH = os.environ.get("VOLUME_MOUNT_PATH")

def create_persistent_volume_func(pvol_name="jpark-shared-volume"):
    vop = kfp.dsl.VolumeOp(name = pvol_name,
                           resource_name = pvol_name,
                           size = '10Gi',).set_display_name("[0] Import Creating persistent
volume")
    return vop

def init_pipeline(sel_pl):
    make_cont = dsl.ContainerOp(
        name=sel_pl["name"],
        image=sel_pl["image"],
        arguments=[item for pair in sel_pl["args"].items() for item in pair],
        command=['python', '/app/main.py'],
    ).set_display_name(sel_pl["display_name"]) \
    .apply(onprem.mount_pvc(PVC_NAME,
                           volume_name=VOLUME_NAME,
                           volume_mount_path=VOLUME_MOUNT_PATH))

    return make_cont

def make_pipeline(prev_cont, sel_pl):
    make_cont = dsl.ContainerOp(
        name=sel_pl["name"],
        image=sel_pl["image"],
        arguments=[item for pair in sel_pl["args"].items() for item in pair],
    
```

```

        command=['python', '/app/main.py'],
    ).set_display_name(sel_pl["display_name"]).after(prev_cont) \
        .apply(onprem.mount_pvc(PVC_NAME,
                                volume_name=VOLUME_NAME,
                                volume_mount_path=VOLUME_MOUNT_PATH))

    return make_cont

@kfp.dsl.pipeline(
    name = "forecasting_10minute_pipeline",
    description = "sample forecasting pipeline"
)

def forecasting_10minute_pipeline():
    target_pl = kf_params_dict["forecasting_10minute_pipeline"]

    component1 = init_pipeline(sel_pl=target_pl["pipeline"].get("component1"))

    component2 = make_pipeline(prev_cont=component1,
                               sel_pl=target_pl["pipeline"].get("component2"))

    # No caching parts
    component1.execution_options.caching_strategy.max_cache_staleness = 'P0D'
    component2.execution_options.caching_strategy.max_cache_staleness = 'P0D'


if __name__ == "__main__":

    session = requests.Session()
    response = session.get(HOST)
    headers = {
        "Content-Type": "application/x-www-form-urlencoded",
    }
    user_data = {"login": USERNAME, "password": PASSWORD}
    session.post(response.url, headers=headers, data=user_data)
    session_cookie = session.cookies.get_dict()["authservice_session"]

    # Connect My KubeFlow Client
    client = kfp.Client(host=f"{HOST}/pipeline",
                       namespace=f"{NAMESPACE}",
                       cookies=f"authservice_session={session_cookie}",
    )

    # User's KF pipeline file written by YAML

```


	데이터 주도형 가상 시뮬레이션 시스템(SW)	
	프로젝트	대규모 분산 에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한 지능형 SW 프레임워크 개발

```

kf_params_file_dir = "../kf_params.yaml"

try:
    with open(kf_params_file_dir) as f:
        kf_params_dict = yaml.load(f, Loader=yaml.FullLoader) #yaml.safe_load(f)

except FileNotFoundError as e:
    raise e

pipeline_func_name

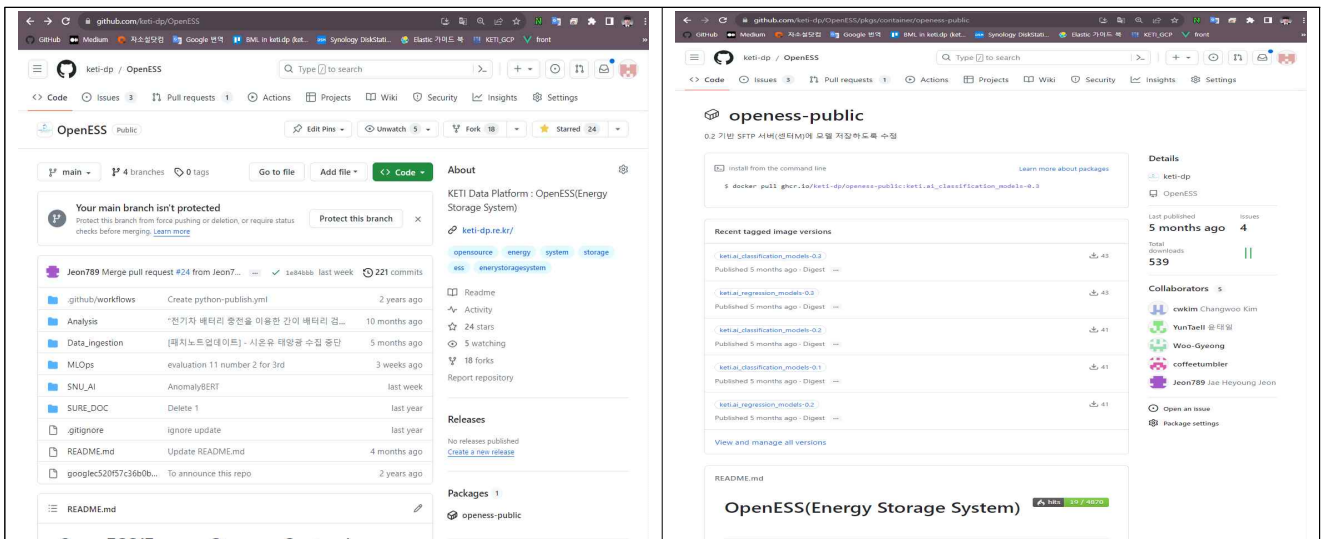
kf_params_dict["forecasting_10minute_pipeline"].get('user_pipeline_func')
pipeline_func = eval(pipeline_func_name)

now = datetime.now()
dt_string = now.strftime("%d/%m/%Y_%H:%M:%S")
print("[KF_pipeline_log] Start time : ", dt_string)

run_name = pipeline_func.__name__ + '-run-' + dt_string

# Submit pipeline directly from pipeline function
run_result = client.create_run_from_pipeline_func(pipeline_func,
                                                    run_name=run_name,
                                                    namespace=NAMESPACE,
                                                    arguments={})

```



〈Github packages를 활용한 컴포넌트 저장소〉