

포팅 매뉴얼

🕒 생성일	@2023년 11월 16일 오전 10:57
🏷 태그	

삼성 청년 SW 아카데미 서울캠퍼스 9기 자율 프로젝트 포팅 매뉴얼



1. 프로젝트 주제 : MaiL - 스마트홈 AI 조명 솔루션
2. 프로젝트 기간 :
3. 개발 인원 : 류병민(팀장), 박건희, 이건, 채문희, 하제우, 홍익선
4. 담당 코치 : 이치현

MaiL 포팅 매뉴얼 목차

1. 프로젝트 기술 스택
 - A. Device
 - B. BE
 - C. AI
 - D. etc.
 2. 프로퍼티 정의
 - A. 사용 포트
 - C. esp-idf sdkconfig
 3. 도구 설치 및 설정
 - A. AWS EC2 설정
 - B. FastAPI & Uvicorn 설치 및 설정
 - C. influxDB 설치 및 설정
 - D. Nginx 설치 및 설정
 - E. kubernetes-dashboard 설정
 - F. Docker 설치 및 설정
 - G. Kubeflow 설치 및 설정
 7. Kubeflow 1.6.0 설치
 - G. esp-idf, matter 설치
 - H. esp-idf, matter build 환경 설정
 4. 빌드 방법
 - A. Device
 - B. AI
1. kubeflow Dashboard 접속
 2. Notebooks 생성
 3. 노트북 선택
 4. 설정 파일 업로드
 5. pipeline.ipynb 파일 업로드
 6. pipeline 실행

1. 프로젝트 기술 스택

A. Device

- 기술 스택
 - esp-matter 1.2
 - connectedhomeip 1.2.0.1
- 개발 환경
 - Visual Studio Code 1.84.2
 - Vim 8.2.1847
 - ESP-IDF 5.1.1
 - WSL2
 - Ubuntu 22.04.2
- 사용 장치
 - ESP32
 - WROOM-32
 - ESP32 CAM

B. BE

- 기술 스택
 - FastAPI 0.104.1
 - Uvicorn 0.24.0
 - InfluxDB 2.7
 - Nginx 1.25.3
- 개발 환경
 - OpenAPI 3.1.0
 - Vim 8.1
 - Python 3.8

C. AI

- 기술 스택
 - Tensorflow 2.5.0
 - numpy 1.19.5
 - pandas 1.2.4
- 개발 환경
 - Ubuntu 20.04 LTS
 - Kubeflow dashboard 2.6.1
 - Python 3.8
 - Jupyter notebook 1.6.0

- LED
 - Red
- 저항
 - 33 Ω
- 움직임 센서 모듈
 - HC-SR501
- 조도 센서
 - GL5528

D. etc.

• MLOps

- cuDNN 10.1
- Nvidia/cuda 11.4.3
- Kubectl v1.21.10
- Kubernetes v1.21.14
- cilium v1.6
- kustomize v3.2.0
- Kubeflow v1.6.0

• 개발 환경

- AWS EC2 g4dn
- GitLab 16.0.5
- MatterMost 7.8.6

2. 프로퍼티 정의

A. 사용 포트

• Nginx

- nginx 외부 포트 : 8002
- 쿠버네티스 내부 포트
 - Service: 8002
 - Pod: 8002
 - Container: 8002

• FastAPI

- 외부로 개방하진 않음
- 쿠버네티스 내부 포트
 - Service : 8000
 - Pod : 8000
 - Container: 8000

• InfluxDB

- 외부로 개방하진 않음
- 쿠버네티스 내부 포트
 - Service: 8086
 - Pod: 8086
 - Container: 8086

- **Kubernetes-dashboard**

- Service: 8001

C. esp-idf sdkconfig

파일 용량이 큰 관계로 gitlab 참조

- S09P31A305/device/ProjectMAIL/sdkconfig

추가적인 설정이 필요한 경우 다음 명령어를 사용해서 설정

```
// 설정 초기화
idf.py set-target esp32

// 설정 추가 선택
idf.py menuconfig
```

- 설정된 옵션
 - partition 설정
 - flash memory size 확장
 - SRAM1 IRAM 확장
 - IRAM 최적화 설정

3. 도구 설치 및 설정

A. AWS EC2 설정

방화벽 설정 - ufw

- ufw 활성화 / 비활성화

```
$ sudo ufw enable
$ sudo ufw disable
```

- ufw 상태 확인

```
$ sudo ufw status
```

- ufw 상태 및 등록된 rule 확인

```
$ sudo ufw status numbered
```

- 사용할 포트 허용

```
$ sudo ufw allow [PORT]
```

- 등록된 포트 조회

```
$ sudo ufw show added
```

- 등록된 포트 삭제

(중요) 삭제한 정책은 반드시 enable를 수행해야 적용된다.

```
$ sudo ufw delete [PORT]
```

B. FastAPI & Uvicorn 설치 및 설정

개발은 `python:3.8` 를 기반 이미지로 활용합니다.

Pod	container	service	port	namespace	ServiceAccount
fastapi-100	fastapi-100	fastapi-100	x→8000	modeltest	default

1. 이미지 빌드

```
$ cat Dockerfile

FROM python:3.8.10 as requirements-stage

WORKDIR /tmp

RUN pip install poetry
COPY ./pyproject.toml ./poetry.lock* /tmp/
RUN poetry export -f requirements.txt --output requirements.txt --without-hashes

FROM python:3.8.10
WORKDIR /code

WORKDIR /code
COPY --from=requirements-stage /tmp/requirements.txt /code/requirements.txt
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt
RUN pip install python-multipart

COPY . /code/app

CMD ["python3", "-m", "uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]

$ docker build -t forwardz/fastapi-for-mail:1.0.0 . --no-cache
```

2. poetry를 이용한 dependencies 설치

```
# pyproject.toml

[tool.poetry]
name = "mail"
version = "0.1.0"
description = ""
authors = ["Your Name <you@example.com>"]
readme = "README.md"

[tool.poetry.dependencies]
python = "^3.8"
fastapi = "^0.104.1"
uvicorn = "^0.24.0"
influxdb-client = "^1.38.0"
minio = "^7.1.17"

[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
```

3. config.ini

influxdb-001 Pod에 접근하기 위한 설정파일. 이 파일을 참고하여 인증해야함. influxdb 이미지를 생성할 때 환경변수로 설정해둔 값을 사용해야함. 노출 없음

url의 influxdb-001은 kubernetes환경에서 influxdb-001 Service domain name입니다.

```
[influx2]
url=http://influxdb-001:8086
org=Webx
token=8A80B1097C150034D
timeout=6000
verify_ssl=False
```

3. fastapi-100 Pod 생성

fastapi-100이라는 이름으로 컨테이너를 생성하고, 동일한 이름의 Pod를 생성. 이 컨테이너는 이 Pod안에서 실행됨.

```
# FastPod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: fastapi-100
  namespace: modeltest
  labels:
    app: fastapi-100
spec:
  containers:
    - name: fastapi-100
      image: forwartz/fastapi-for-mail:1.0.0
      ports:
        - containerPort: 8000
      resources:
        requests:
          memory: "512Mi"
          cpu: "500m"
        limits:
          memory: "1Gi"
          cpu: "1"
```

4. fastapi-100 Service 생성

fastapi-100이라는 이름으로 서비스를 생성하고, 8000번포트로 요청을 받아서 fastapi-100 Pod의 8000번 포트로 전달.

```
# MailServices.yaml 의 일부

apiVersion: v1
kind: Service
metadata:
  name: fastapi-100
  namespace: modeltest
spec:
  selector:
    app: fastapi-100
  ports:
    - protocol: TCP
      port: 8000
      targetPort: 8000
```

5. Pod, Service 생성하는 파일 실행

```
$ k apply -f FastPod.yaml
$ k apply -f MailServices.yaml
```

6. API 명세서

앞에 **http://[공인IP]:[포트]** 는 공통, **test용 productionName : M16M**

API 이름	기능	경로	메서드	파라미터	Request	R
postSensorData	센서데이터 송신. 상세 서버측에서 는 이 데이터를 influxDB에 저장	/upload/sensor/{productionName}	POST	productionName	{ "MM": int, "DD": int, "HH": int, "Min": int, "Sec": int, "Day": int, "Illuminance": int, "Manual":	종 오 "1

					bool, "Brightness": int, "Movement": bool, "On": bool }	
getTfliteFile	학습파일 다운로드. 상세 서버측에서는 minio 객체 스토리지에 결과물이 나왔나 확인 후, 존재하면 60byte씩 읽어서 전송. 클라이언트 측에서 여러번 요청해야함	/download/tflitefile/{productionName}/{order}	GET	productionName order		존재 여부
getTimeFile	시간 정보 수신	/Mail/gettime	GET			존재 여부

C. influxDB 설치 및 설정

개발은 influxdb:2.7 image를 기반 이미지로 활용합니다.

Pod	container	service	port	namespace	ServiceAccount
influxdb-001	influxdb-001	influxdb-001	x → 8086	modeltest	default

1. 이미지 빌드

```
$ cat Dockerfile

FROM influxdb:2.7

ENV DOCKER_INFLUXDB_INIT_MODE=setup
ENV DOCKER_INFLUXDB_INIT_USERNAME=PANZER
ENV DOCKER_INFLUXDB_INIT_PASSWORD=Webxa305#
ENV DOCKER_INFLUXDB_INIT_ORG=Webx
ENV DOCKER_INFLUXDB_INIT_BUCKET=Mail
ENV DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=8A80B1097C150034D      #토큰 인증시 필요. 노출 금지
ENV DOCKER_INFLUXDB_INIT_RETENTION=1w                        #데이터 보유 기간

$ docker build -t forwartz/influxdb-for-mail:0.0.1 . --no-cache
```

2. Properties

```
bucket = Mail
measurement = SensorData
tag = M16M
field 1 = MM          , (int)
field 2 = DD          , (int)
field 3 = HH          , (int)
field 4 = Min         , (int)
field 5 = Sec         , (int)
field 6 = Day         , (int)
field 7 = Illuminance , (int)
field 8 = Manual      , (bool)
field 9 = Brightness  , (int)
field 10 = Movement   , (bool)
field 11 = On         , (bool)
```

3. influxdb-001 Pod 생성

influxdb-001이라는 이름으로 컨테이너를 생성하고, 동일한 이름의 Pod를 생성. 이 컨테이너는 이 Pod안에서 실행됨.

```
# FluxPod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: influxdb-001
  namespace: modeltest
  labels:
    app: influxdb-001
spec:
  containers:
    - name: influxdb-001
      image: forwartz/influxdb-for-mail:0.0.1
      ports:
        - containerPort: 8086
      resources:
        requests:
          memory: "512Mi"
          cpu: "500m"
        limits:
          memory: "1Gi"
          cpu: "1"
```

4. influxdb-001 Service 생성

influxdb-001이라는 이름으로 서비스를 생성하고, 8086번포트로 요청을 받아서 influxdb-001 Pod의 8086번 포트로 전달.

```
# MailServices.yaml 의 일부

apiVersion: v1
kind: Service
metadata:
  name: influxdb-001
  namespace: modeltest
spec:
  selector:
    app: influxdb-001
  ports:
    - protocol: TCP
      port: 8086
      targetPort: 8086
```

5. Pod, Service 생성하는 파일 실행

```
$ k apply -f FluxPod.yaml
$ k apply -f MailServices.yaml
```

D. Nginx 설치 및 설정

개발은 `nginx:latest` 이미지를 기반으로 활용합니다.

Pod	container	service	port	namespace	ServiceAccount
mginx-010	mginx-010	mginx-010	8002 → 8002	modeltest	default

2. 이미지 빌드

```
$ cat Dockerfile

FROM nginx:latest
COPY test.conf /etc/nginx/conf.d/test.conf
RUN rm /etc/nginx/conf.d/default.conf

$ docker build -t mginx:0.1.0 . --no-cache
```

1. nginx 설정파일 작성


```
# test.conf

server {
    listen      8001;
    listen  [::]:8001;
    server_name localhost;

    access_log /var/log/nginx/host.access.log main;

    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
    }
    location /docs {                                #-> fastapi swagger를 사용하기 위한 경로
        proxy_pass http://fastapi-100:8000/docs;

        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
    location /openapi.json {                        #-> /docs로 연결시, 리디렉트되는 경로
        proxy_pass http://fastapi-100:8000/openapi.json;

        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
    location ~ ^/upload/sensor/(\w+)/ {            #-> 센서데이터를 수신하는 경로
        proxy_pass http://fastapi-100:8000/upload/sensor/$1;
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
    location ~ ^/download/tflitefile/(\w+)/(\d+) {  #.tflite파일을 송신해주기 위한 경로
        proxy_pass http://fastapi-100:8000/download/tflitefile/$1/$2;
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
    location /Mail/gettime {                        # esp32에게 시각을 알려주는 경로
        proxy_pass http://fastapi-100:8000/Mail/gettime;
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    #error_page 404                /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root /usr/share/nginx/html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ \.php$ {
    #    proxy_pass http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ \.php$ {
    #    root            html;
    #    fastcgi_pass    127.0.0.1:9000;
    #    fastcgi_index   index.php;
    #    fastcgi_param   SCRIPT_FILENAME /scripts$fastcgi_script_name;
    #    include          fastcgi_params;
    #}

    # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
    #location ~ /\.ht {
    #    deny all;
    #}

```

```
}  
#}  
}
```

proxy_pass되는 경로로 사용되는 “fastapi-100”은 쿠버네티스 상에서 fastapi Pod와 연결된 Service이름입니다.

2. 이미지 빌드

```
$ cat Dockerfile  
  
FROM nginx:latest  
COPY test.conf /etc/nginx/conf.d/test.conf  
RUN rm /etc/nginx/conf.d/default.conf  
  
$ docker build -t mginx:0.1.0 . --no-cache
```

3. mginx-010 Pod 생성

mginx-010이라는 이름으로 컨테이너를 생성하고, 동일한 이름의 Pod를 생성. 이 컨테이너는 이 Pod안에서 실행됨.

```
# Mginx.yaml  
  
apiVersion: v1  
kind: Pod  
metadata:  
  name: mginx-010  
  namespace: modeltest  
  labels:  
    app: mginx-010  
spec:  
  containers:  
    - name: mginx-010  
      image: mginx:0.1.0  
      ports:  
        - containerPort: 8002  
      resources:  
        requests:  
          memory: "512Mi"  
          cpu: "500m"  
        limits:  
          memory: "1Gi"  
          cpu: "1"
```

4. mginx-010 Service 생성

mginx-010이라는 이름으로 서비스를 생성하고, 8002번포트로 요청을 받아서 mginx-010 Pod의 8002번 포트에 전달.

```
# MailServices.yaml 의 일부  
  
apiVersion: v1  
kind: Service  
metadata:  
  name: mginx-010  
  namespace: modeltest  
spec:  
  selector:  
    app: mginx-010  
  ports:  
    - protocol: TCP  
      port: 8002  
      targetPort: 8002
```

5. Pod, Service 생성하는 파일 실행

```
$ k apply -f Mginx.yaml  
$ k apply -f MailServices.yaml
```

5. Gateway 연결

```
$ kubectl port-forward --address=0.0.0.0 svc/mginx-010 -n modeltest 8002:8002
```

6. EC2 인바운드 규칙 8002번 포트 해제

사용중인 EC2 인스턴스 보안그룹에서, 8002번 포트로 TCP통신이 가능하도록 인바운드 규칙을 추가합니다.

sgr-0040859bd963c21fd	8002	TCP	0.0.0.0/0	A305_VGA_Server_1 
-----------------------	------	-----	-----------	---

7. 결과

공인IP:8002 포트로 접속 가능

13.125.12.50:8002

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

E. kubernetes-dashboard 설정

1. 쿠버네티스 대시보드 UI 배포

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.6.1/aio/deploy/recommended.yaml
```

2. 대시보드 서비스 수정

```
$ kubectl edit services kubernetes-dashboard -n kubernetes-dashboard
```

```
...
spec:
  ...
  ports:
    - port: 8001      # 외부에서 접속하려는 포트로 변경
      ...
    ...
  type: ClusterIP # ClusterIP로 변경
  ...
  ...
```

Esc + w로 저장 후 나오면 자동 적용

3. 외부 접속을 위한 Gateway 설정

```
$ kubectl port-forward --address=0.0.0.0 svc/kubernetes-dashboard -n kubernetes-dashboard 8001:8001
```

nginx의 경우와 마찬가지로 EC2 인바운드 규칙수정을 통해 설정해준 포트도 허용해야함
그러면 https://[공인IP]:[설정해준 포트인 8001] 로 접속가능

Kubernetes Dashboard

☒ 토큰
 모든 서비스 어카운트는 시크릿을 가지고 있고, 시크릿에는 대시보드에 로그인할 때 사용할 수 있는 유효한 베어러(Bearer) 토큰이 있습니다. 베어러(Bearer) 토큰을 설정 및 사용하는 방법은 [인증](#) 섹션에서 알 수 있습니다.

☐ Kubeconfig
 클러스터에 접근을 설정하기 위해 생성한 kubeconfig 파일을 선택하세요. kubeconfig 파일을 설정 및 사용하기 위한 방법은 [멀티 클러스터에 접근 설정하기](#) 섹션에서 확인할 수 있습니다.

토큰 입력 *

로그인

4. 권한 설정

쿠버네티스 대시보드에서 모든 네임스페이스의 리소스를 확인할 수 있도록 권한을 부여해야한다. 그래서 다음과 같은 yaml파일 생성 후 실행

```
$ cat kubernetes-dashboard-service-account.yaml

apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kube-system

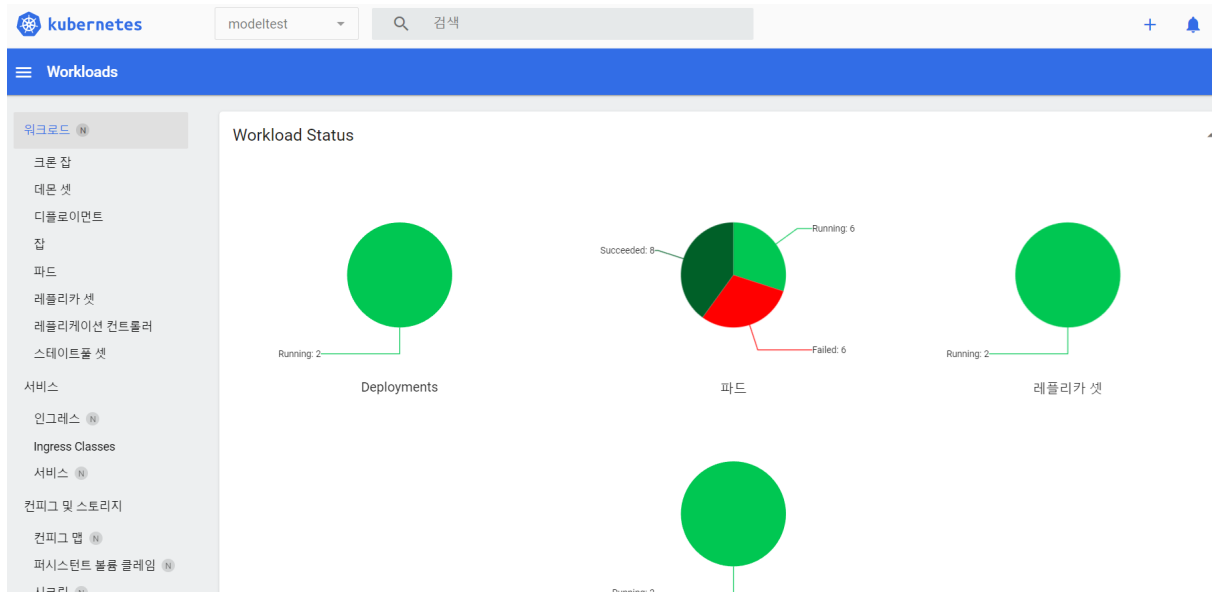
$ k apply -f kubernetes-dashboard-service-account.yaml
```

다음의 명령어로, admin-user의 토큰 출력

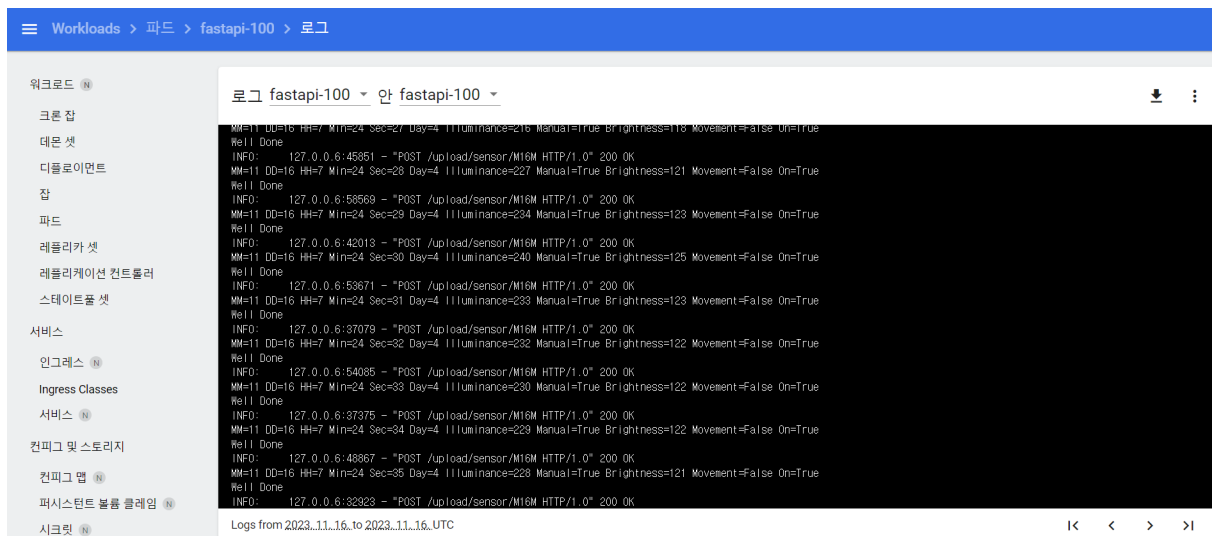
```
$ kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')
```

5. 토큰값 이용해서 로그인

좌측 상단에 네임스페이스를 골라서 리소스 확인이 가능하고 shell접속이나 log도 확인 가능



로그



E. Docker 설치 및 설정

Docker Engine 설치

1. 오래된 버전 삭제

```
$ for pkg in docker.io docker-doc docker-compose podman-docker containerd runc; do sudo apt-get remove $pkg; done
```

2. apt repository 셋업

```

# Update the apt package index and install packages to allow apt to use a repository over HTTPS
$ sudo apt-get update
$ sudo apt-get install ca-certificates curl gnupg

# Add Docker's official GPG key
$ sudo install -m 0755 -d /etc/apt/keyrings
  
```

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
$ sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Set up the repo
$ echo \
"deb [arch=$(dpkg --print-architecture)] signed-by=/etc/apt/keyrings/docker.gpg https://download.docker.com/linux/ubuntu \
"$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Update apt package index
$ sudo apt-get update
```

3. Docker Engine 설치

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

4. 설치 확인

```
$ sudo docker run hello-world
```

Docker 설정

- 컨테이너 찾기

```
$ sudo docker ps -a
```

- 컨테이너 접속

```
$ sudo docker exec -it [컨테이너 이름] /bin/bash
```

- 컨테이너 접속종료

```
# exit
```

- 컨테이너 중지

```
$ sudo docker stop [컨테이너 이름]
```

- 컨테이너 삭제

```
$ sudo docker rm -f [컨테이너 이름]
```

- 이미지 생성 (Dockerfile이 있는 디렉토리에서)

```
$ sudo docker build -t [이미지 이름] .
```

- 이미지 찾기

```
$ sudo docker images
```

- 이미지 실행

```
$ sudo docker run --name [생성할 컨테이너 이름] -p [host 포트]:[docker 포트] [이미지 이름]
```

- 이미지 삭제

```
$ sudo docker rmi [이미지 이름]
```

F. Kubeflow 설치 및 설정

1. cuDNN 설치

AWS 스펙 상 NVIDIA GPU(g4dn)을 사용하기 때문에, GPU 가속화를 위해 cuDNN을 설치한 후 nvidia Docker(2)를 이용해서 GPU 가속을 할 것이다.

Working Directory는 Home Directory에서 진행한다.

먼저 다음 Shell script를 실행한다.

```
#!/bin/bash

sudo apt-get -y update
sudo apt-get -y remove --purge '^nvidia-.*'
sudo apt-get -y remove --purge 'cuda-.*'
sudo apt-get -y install nvidia-cuda-toolkit
sudo apt-get -y install nvidia-cuda-toolkit
nvcc -V
whereis cuda
mkdir ~/nvidia
cd ~/nvidia
CUDNN_TAR_FILE="cudnn-10.1-linux-x64-v7.6.5.32.tgz"
wget https://developer.download.nvidia.com/compute/redist/cudnn/v7.6.5/${CUDNN_TAR_FILE}
tar -xzf ${CUDNN_TAR_FILE}

sudo cp cuda/include/cudnn.h /usr/lib/cuda/include/
sudo cp cuda/lib64/libcudnn* /usr/lib/cuda/lib64/
sudo chmod a+r /usr/lib/cuda/lib64/libcudnn*
sudo chmod a+r /usr/lib/cuda/include/cudnn.h
echo "export LD_LIBRARY_PATH=/usr/lib/cuda/lib64:$LD_LIBRARY_PATH" >> ~/.bashrc
export "LD_LIBRARY_PATH=/usr/lib/cuda/include:$LD_LIBRARY_PATH" >> ~/.bashrc
source ~/.bashrc
ubuntu-drivers devices
sudo apt install -y nvidia-driver-470
echo "reboot...."
```

Docker 설치

위 설치가 완료되면 다음 Shell script를 실행한다.

```
#!/bin/bash

sudo apt-get install -y apt-transport-https ca-certificates curl gnupg lsb-release
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
echo "[deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable]" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update -y
sudo apt-get install -y docker-ce docker-ce-cli containerd.io
sudo docker run hello-world
sudo usermod -aG docker $USER && newgrp docker
sudo service docker restart
```

Docker 설치가 끝나면, 재부팅을 해준다.

```
$ sudo reboot
```

2. Nvidia Docker 설치

reboot를 완료했다면, 다음 shell script를 활용해서 nvidia docker를 설치해준다.

```
#!/bin/bash

distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list
sudo apt-get -y update
sudo apt-get -y install nvidia-docker2
sudo systemctl restart docker
sudo docker run --runtime nvidia nvidia/cuda:11.4.3-base-ubuntu20.04 /usr/bin/nvidia-smi
sudo bash -c 'cat <<EOF > /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "data-root": "/mnt/storage/docker_data",
  "storage-driver": "overlay2",
  "default-runtime": "nvidia",
  "runtimes": {
    "nvidia": {
      "path": "/usr/bin/nvidia-container-runtime",
      "runtimeArgs": []
    }
  }
}
EOF'
sudo systemctl restart docker
```

```
ca1778b69356: Pull complete
3b595136a210: Pull complete
b0cbedff9be8: Pull complete
569ff44c1698: Pull complete
1c34507dbc74: Pull complete
Digest: sha256:bee557a9785dc70cc3c5084e9b358ad1d64f4b1fc5d87e0fe82c540355a5eba1
Status: Downloaded newer image for nvidia/cuda:11.4.3-base-ubuntu20.04
Wed Nov 1 01:55:04 2023

+-----+
| NVIDIA-SMI 470.199.02   Driver Version: 470.199.02   CUDA Version: 11.4   |
+-----+
| GPU   Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                                           MIG M.           |
+-----+
|  0   Tesla T4             Off   | 00000000:00:1E:0 Off |                    |
| N/A   38C    P0      26W / 70W   |  0MiB / 15109MiB |      0%      Default  |
+-----+

+-----+
| Processes: |
| GPU   GI    CI          PID    Type   Process name                      GPU Memory |
| ID   ID     ID              |                 | Usage     |
+-----+
| No running processes found |
+-----+
```

3. k8s 설치 및 초기화

nvidia-smi를 통해 지표가 동작함을 확인했다면, 다음 shell script를 실행시킨다.

```
#!/bin/bash

sudo swapoff -a
sudo sed -i 's/ swap / s/^(.*)$/#\1/g' /etc/fstab

sudo apt-get install -y iptables arptables ebttables
sudo apt-get update && sudo apt-get install -y apt-transport-https curl
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
```



```

sudo apt-get update
sudo apt-get install -y kubelet=1.21.10-00 kubeadm=1.21.10-00 kubectl=1.21.10-00 --allow-downgrades --allow-change-held-packages
sudo apt-mark hold kubelet kubeadm kubectl
kubeadm version
kubelet --version
kubectl version --client

```

```

kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
kubeadm version: &version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.10", GitCommit:"a7a32748b5c60445cd7ee904caf01b91f2db671", GitTreeState:"clean", BuildDate:"2022-02-16T11:22:49Z", GoVersion:"go1.16.14", Compiler:"gc", Platform:"linux/amd64"}
kubeadm version: &version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.10", GitCommit:"a7a32748b5c60445cd7ee904caf01b91f2db671", GitTreeState:"clean", BuildDate:"2022-02-16T11:22:49Z", GoVersion:"go1.16.14", Compiler:"gc", Platform:"linux/amd64"}
Client Version: &version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.10", GitCommit:"a7a32748b5c60445cd7ee904caf01b91f2db671", GitTreeState:"clean", BuildDate:"2022-02-16T11:22:49Z", GoVersion:"go1.16.14", Compiler:"gc", Platform:"linux/amd64"}
shuntu@ip-10-0-1-10:~$ kubectl version --client
Client Version: v1.21.10
Kubernetes Version: v1.21.10

```

설치가 완료되었다면, 다음 shell script를 실행한다.

```

#!/bin/bash
# init k8s
sudo kubeadm init --pod-network-cidr=10.217.0.0/16

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

kubectl cluster-info

# CNI
kubectl create -f https://raw.githubusercontent.com/cilium/cilium/v1.6/install/kubernetes/quick-install.yaml
kubectl get pods -n kube-system --selector=k8s-app=cilium

kubectl taint nodes --all node-role.kubernetes.io/master-
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/1.0.0-beta6/nvidia-device-plugin.yml

#test GPU
kubectl -n kube-system get pod -l name=nvidia-device-plugin-ds
kubectl -n kube-system logs -l name=nvidia-device-plugin-ds

# default storageclass
kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provisioner/master/deploy/local-path-storage.yaml
kubectl get storageclass
kubectl patch storageclass local-path -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
kubectl get sc

# install kustomize# if [ ! -f /usr/local/bin/kustomize ]
then
echo "kustomize"
wget https://github.com/kubernetes-sigs/kustomize/releases/download/v3.2.0/kustomize_3.2.0_linux_amd64
mv ./kustomize_3.2.0_linux_amd64 kustomize
sudo chmod 777 kustomize
sudo mv kustomize /usr/local/bin/kustomize
fi

# autocomplete k8s
shellname=`echo $SHELL | rev | cut -d '/' -f1 | rev`
shellconf=`echo ~/.\${shellname}rc`
grep -n "kubectl completion" $shellconf

if [ $? = 1 ]
then
echo 'install autocomplete k8s'
sudo apt-get install bash-completion -y
echo 'source <(kubectl completion '$shellname')' >>$shellconf
echo 'alias k=kubectl' >>$shellconf
echo 'complete -F __start_kubectl k' >>$shellconf
fi
$SHELL

```

CNI는 cilium 1.6 이상은 제대로 동작하지 않기 때문에 사용하지 않는 것을 추천 (안 되면 cilium 말고 calico로 설치하면 잘 된다)

kustomize는 3.2.0 이상 버전은 지원이 안되기 때문에, 이상의 버전 사용을 권장하지 않는다.

4. kubeflow 설치

kubeflow 설치를 위해 kubeflow github에 들어가서 manifests를 다운받아야 한다.

```
$ cd ~
$ git clone https://github.com/kubeflow/manifests.git

$ cd manifests
$ git checkout v1.5.1# kubeflow 1.5.1 설치를 위해
```

일단 이전에 github에 써 있는 내용 중, cert-manager를 먼저 설치하고, 나머지는 single command로 설치할 것이다.

```
$ kustomize build common/cert-manager/cert-manager/base | kubectl apply -f -

# 확인
$ watch -n1 kubectl get pod -A
```

```
Every 1.0s: kubectl get pod -A ip-172-31-39-103: Wed Nov 1 04:01:37 2023

NAMESPACE      NAME                                READY   STATUS    RESTARTS   AGE
cert-manager    cert-manager-7b8c77d4bd-r48c5      1/1     Running   0           81m
cert-manager    cert-manager-cainjector-7c744f57b5-sr4kf  1/1     Running   0           81m
cert-manager    cert-manager-webhook-fcd445bc4-2c21h  1/1     Running   0           81m
kube-system     cilium-operator-59db6bffd9-j8v2j     1/1     Running   0           90m
kube-system     cilium-tqpg9                        1/1     Running   0           90m
kube-system     coredns-558bd4d5db-brpc5            1/1     Running   0           90m
kube-system     coredns-558bd4d5db-zg4xf            1/1     Running   0           90m
kube-system     etcd-ip-172-31-39-103               1/1     Running   0           91m
kube-system     kube-apiserver-ip-172-31-39-103      1/1     Running   0           91m
kube-system     kube-controller-manager-ip-172-31-39-103  1/1     Running   0           91m
kube-system     kube-proxy-6hr75                    1/1     Running   0           90m
kube-system     kube-scheduler-ip-172-31-39-103      1/1     Running   0           91m
kube-system     nvidia-device-plugin-daemonset-dvslm  1/1     Running   0           90m
local-path-storage local-path-provisioner-849cb58dff-t15r9  1/1     Running   0           90m
```

pod을 보았을 때, 모두 Running 상태면 다음 단계를 진행하면 된다.

```
$ kustomize build common/cert-manager/kubeflow-issuer/base | kubectl apply -f -

# 위 작업이 완료가 되면, 아래 싱글 커맨드를 진행
$ while ! kustomize build example | kubectl apply -f -; do echo "Retrying to apply resources"; sleep 10; done

# 다시 확인
$ watch -n1 kubectl get pod -A
```

이후 전체 pod가 running 상태가 될 때까지 대기한다.

5. Kubeflow 접속

이제 Kubeflow 설치가 완료가 되었고, Kubeflow에 접근하기 위해 port forwarding을 진행해주어야 한다.

```
# istio-system이 있는 지 확인
$ kubectl get namespaces

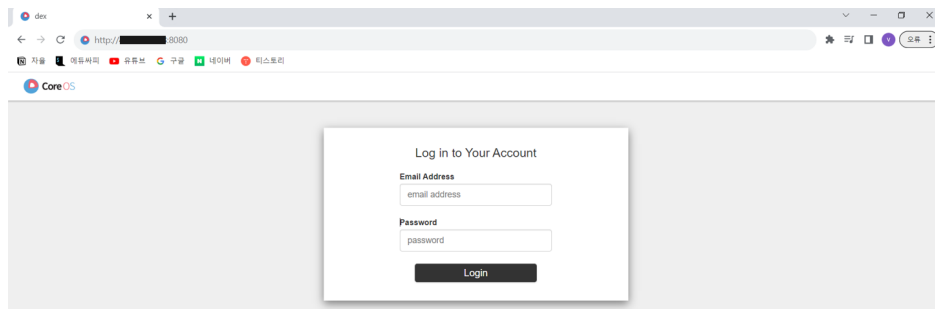
# istio-ingressgateway PORT 확인
$ kubectl get svc -n istio-system

# 원하는 포트(8080)로 port-forwarding 진행
$ kubectl port-forward --address=0.0.0.0 svc/istio-ingressgateway -n istio-system 8080:80
```

```
ubuntu@ip-172-31-39-103:~/manifests$ watch -n1 kubectl get pod -A
ubuntu@ip-172-31-39-103:~/manifests$ kubectl get namespaces
NAME                STATUS    AGE
auth                Active    12m
cert-manager        Active    96m
default             Active    106m
istio-system        Active    12m
knative-eventing    Active    12m
knative-serving     Active    12m
kserve              Active    12m
kube-node-lease     Active    106m
kube-public         Active    106m
kube-system         Active    106m
kubeflow            Active    12m
kubeflow-user-example-com Active    3m51s
local-path-storage  Active    106m
ubuntu@ip-172-31-39-103:~/manifests$ kubectl get svc -n istio-system
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
authservice         ClusterIP    10.96.129.138    <none>            8080/TCP          16m
cluster-local-gateway ClusterIP    10.111.127.181   <none>            15020/TCP,80/TCP  16m
istio-ingressgateway NodePort     10.105.191.124   <none>            15021:31424/TCP,80:30497/TCP,443:31834/TCP,31400:30356/TCP,15443:31849/TCP  16m
istiod              ClusterIP    10.102.191.61    <none>            15010/TCP,15012/TCP,443/TCP,15014/TCP  16m
knative-local-gateway ClusterIP    10.102.9.232     <none>            80/TCP            16m
ubuntu@ip-172-31-39-103:~/manifests$ kubectl port-forward --address=0.0.0.0 svc/istio-ingressgateway -n istio-system 8080:80
Forwarding from 0.0.0.0:8080 -> 8080
```

forwarding 명령어 실행 중인 상태

그럼 지금 상태로는, http에 본인 public ip 주소와 8080포트로 kubeflow에 접근이 가능하다.



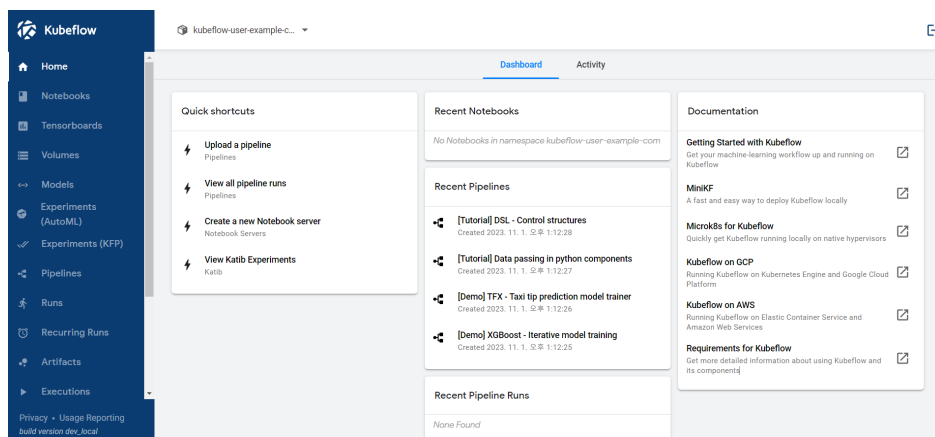
Kubeflow 초기 화면

처음 이메일 및 패스워드는 다음과 같다.

- **Email Address** : user@example.com

- **Password** : 12341234

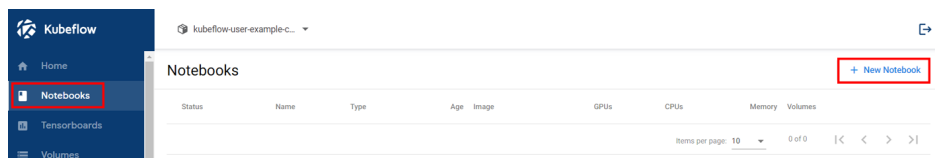
- **Namespace** : kubeflow-user-example-com



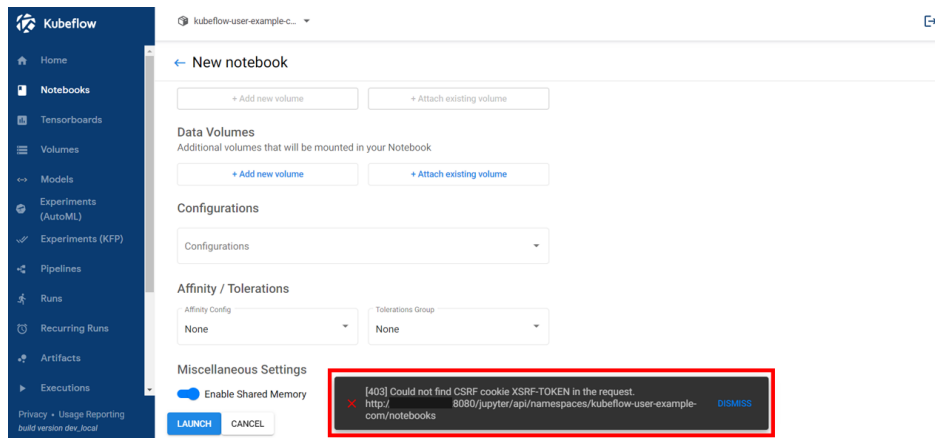
로그인 성공 화면

6. Kubeflow 환경 설정

테스트하기 위해 notebook을 새로 만들어보자



해당 페이지에서, 노트북 이름과 이미지 설정만 한 다음, Launch 버튼을 누르면 다음과 같은 **CSRF 오류**가 난다.



두 yaml 파일을 작성해서 배포하면 된다.

```
! gateway.yaml X
1      .yaml
2  apiVersion: networking.istio.io/v1alpha3
3  kind: Gateway
4  metadata:
5    name: kubeflow-gateway
6    namespace: kubeflow
7  spec:
8    selector:
9      istio: ingressgateway
10   servers:
11     - hosts:
12         - "*"
13       port:
14         name: http
15         number: 80
16         protocol: HTTP
17       # Upgrade HTTP to HTTPS
18       tls:
19         httpsRedirect: true
20     - hosts:
21         - "*"
22       port:
23         name: https
24         number: 443
25         protocol: HTTPS
26       tls:
27         mode: SIMPLE
28         credentialName: kubeflow-ingressgateway-certs
```

```
! certificate.yaml X
1  apiVersion: cert-manager.io/v1alpha2
2  kind: Certificate
3  metadata:
4    name: kubeflow-ingressgateway-certs
5    namespace: istio-system
6  spec:
7    commonName: example.com #Domain name
8    issuerRef:
9      kind: ClusterIssuer
10     name: kubeflow-self-signing-issuer
11     secretName: kubeflow-ingressgateway-certs
12
```

```
$ kubectl apply -f gateway.yaml
$ kubectl apply -f certificate.yaml
```

이렇게 두 파일을 적용을 해주면 https로 접근할 수가 있다.

```
# gateway가 443이 https로 되어있으므로, 443으로 port forwarding 진행
$ kubectl port-forward --address=0.0.0.0 svc/istio-ingressgateway -n istio-system 8080:443
```

이후에 아가와 같이 접속을 진행한다.

연결이 비공개로 설정되어 있지 않습니다.

공격자가 [redacted]에서 정보(예: 비밀번호, 메시지, 신용카드 등)를 도용하려고 시도 중일 수 있습니다. [자세히 알아보기](#)

NET::ERR_CERT_AUTHORITY_INVALID

Chrome에서 가장 강력한 보안 기능을 사용하려면 [향상된 보호 모드](#)를 사용 설정하세요.

세부정보 숨기기

안전한 페이지로 돌아가기

이 서버가 [redacted]임을 입증할 수 없으며 컴퓨터의 운영체제에서 신뢰하는 보안 인증서가 아닙니다. 서버를 잘못 설정했거나 불법 사용자가 연결을 가로채고 있기 때문일 수 있습니다.

[redacted] (안전하지 않음)(으)로 이동

인증서가 자체 인증서기 때문에 실제로는 그냥 들어가지지는 않고, 안전하지 않음을 눌러 이동한다.

이제 노트북을 생성하면, CSRF 오류가 뜨지 않는다.

Status	Name	Type	Age	Image	GPUs	CPUs	Memory	Volumes	
✓	test2		11 mins ago	jupyter-scipy:v1.5.0	0	100m	1Gi		CONNECT

Items per page: 10 1 - 1 of 1 < >

7. Kubeflow 1.6.0 설치

kubeflow 1.6.0을 설치하기 위해서는 kubeflow 초기화를 해주어야 한다.

```
$ sudo kubeadm reset
```

```

bunt@bunt:~$ sudo kubeadm reset
[reset] Reading configuration from the cluster...
[reset] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[reset] WARNING: Changes made to this host by 'kubeadm init' or 'kubeadm join' will be reverted.
[reset] Are you sure you want to proceed? [y/N]: y
[preflight] Running pre-flight checks
[reset] Removing info for node "bunt" from the ConfigMap "kubeadm-config" in the "kube-system" Namespace
[reset] Stopping the kubeadm service
[reset] Unmounting mounted directories in "/var/lib/kubelet"
[reset] Deleting contents of config directories: [/etc/kubernetes/manifests /etc/kubernetes/pki]
[reset] Deleting files: [/etc/kubernetes/admin.conf /etc/kubernetes/kubelet.conf /etc/kubernetes/bootstrap-kubelet.conf /etc/kubernetes/controller-manager.conf /etc/kubernetes/scheduler.conf]
[reset] Deleting contents of stateful directories: [/var/lib/etcd /var/lib/kubelet /var/lib/docker/shim /var/run/kubernetes /var/lib/cni]

The reset process does not clean CNI configuration. To do so, you must remove /etc/cni/net.d

The reset process does not reset or clean up iptables rules or IPVS tables.
If you wish to reset iptables, you must do so manually by using the "iptables" command.

If your cluster was setup to utilize IPVS, run iptables -F (or similar)
to reset your system's IPVS tables.

The reset process does not clean your kubeconfig files and you must remove them manually.
Please, check the contents of the $HOME/.kube/config file
  
```

config file(빨간 박스)까지 지워야 k8s가 초기화된다.

```
$ rm $HOME/.kube/config
```

이후 3번 과정에 있는 k8s initiation을 다시 진행한다.

그리고 후속 과정 중, v1.6.0 tag로 checkout 후 같은 과정을 진행하면 된다.

G. esp-idf, matter 설치

- esp-idf 설치

```
sudo apt-get update
sudo apt-get upgrade

// 패키지 설치
sudo apt-get install git wget flex bison gperf python3 python3-venv cmake ninja-build ccache libffi-dev libssl-dev dfu-util libusb-1.0-

// idf 설치
mkdir -p ~/esp
cd ~/esp
git clone -b v5.1 --recursive https://github.com/espressif/esp-idf.git

// idf 환경 설치
cd ~/esp/esp-idf
./install.sh esp32
```

- esp-matter 설치

```
cd
git clone --recurse-submodules https://github.com/project-chip/connectedhomeip.git
cd connectedhomeip
git pull
git submodule update --init
```

H. esp-idf, matter build 환경 설정

```
cd cd S09P31A305/device/ProjectMAIL

// esp-idf 가져오기
. $HOME/esp-idf/export.sh

// esp-matter 가져오기
. $HOME/esp-matter/export.sh

// build
idf.py build

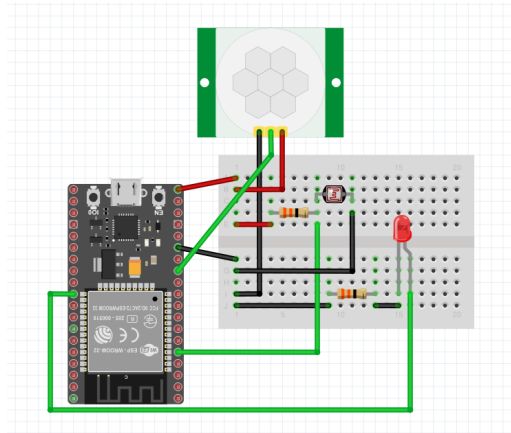
// flash
idf.py -p /dev/ttyUSB0 flash

// monitor
idf.py -p /dev/ttyUSB0 monitor
```

4. 빌드 방법

A. Device

하드웨어 구성



- LED 핀 : GPIO 5번
- 조도센서 : GPIO 34번
- 모션센서 : GPIO 14번

빌드

```
cd cd S09P31A305/device/ProjectMAIL

// esp-idf 가져오기
. $HOME/esp-idf/export.sh

// esp-matter 가져오기
. $HOME/esp-matter/export.sh

// build
idf.py build

// flash
idf.py -p /dev/ttyUSB0 flash

// monitor
idf.py -p /dev/ttyUSB0 monitor
```

허브 연결

SamrtThings Hub를 네트워크(wifi, 유선랜)에 연결합니다.

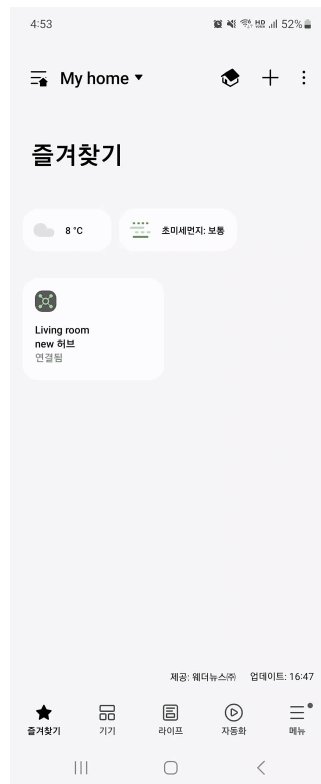
Device Wifi 설정

```
// idf를 가져와야 합니다. build 환경설정 참조
idf.py menuconfig
```

/ : 검색 기능

/ → SSID 검색 → wifi id와 password 설정 (hub와 동일)

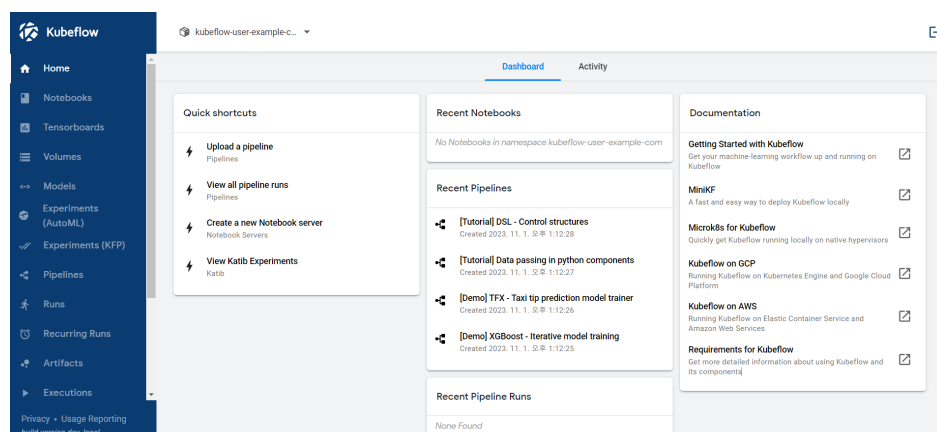
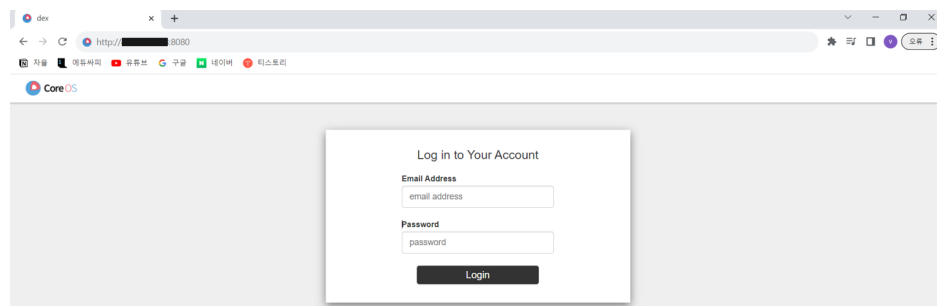
SmartThigns APP에 Device 연결



B. AI

1. kubeflow Dashboard 접속

kubeflow를 웹 브라우저를 통해 접속하고, 앞선 과정에서 인가된 계정으로 접속한다.



2. Notebooks 생성

Kubeflow Central Dashboard

주요 요약 | [https://\[redacted\]/jupyter/?ns=modeltest](https://[redacted]/jupyter/?ns=modeltest)

Kubeflow modeltest (Owner)

Notebooks

[+ New Notebook](#)

Status	Name	Type	Age	Last activity	Image	GPUs	CPUs	Memory	Volumes	
	a305-pro...		2 d...	32 minutes ago	jupyter_nb...	1	2	8Gi	⋮	CONNECT
	gputest		11 ...	-	jupyter_nb...	1	1	4Gi	⋮	CONNECT

Items per page: 10 1 - 2 of 2

해당 버튼을 클릭하여 사용할 노트북을 생성한다.

Kubeflow
modeltest (Owner)

New notebook

Name

Name

Namespace

modeltest

Docker Image

☐ Custom Image

jupyterlab

1

2

Image

kubeflownotebookswg/jupyter-scipy:v1.6.0

Advanced Options

CPU / RAM

Requested CPUs

0.5

Requested memory in Gi

1

Advanced Options

GPUs

Number of GPUs

None

GPU Vendor

Workspace Volume

Volume that will be mounted in you home directory.

New volume

-volume, Empty, 10Gi

다음과 같은 설정들이 있는데, 세부 설정들은 다음과 같다.

① Docker Image 선택

Notebook은 container image를 미리 생성한 후, 이를 호출하는 방식이다.

Image

kubeflownotebookswg/jupyter-scipy:v1.6.0

kubeflownotebookswg/jupyter-pytorch-full:v1.6.0

kubeflownotebookswg/jupyter-pytorch-cuda-full:v1.6.0

kubeflownotebookswg/jupyter-tensorflow-full:v1.6.0

kubeflownotebookswg/jupyter-tensorflow-cuda-full:v1.6.0

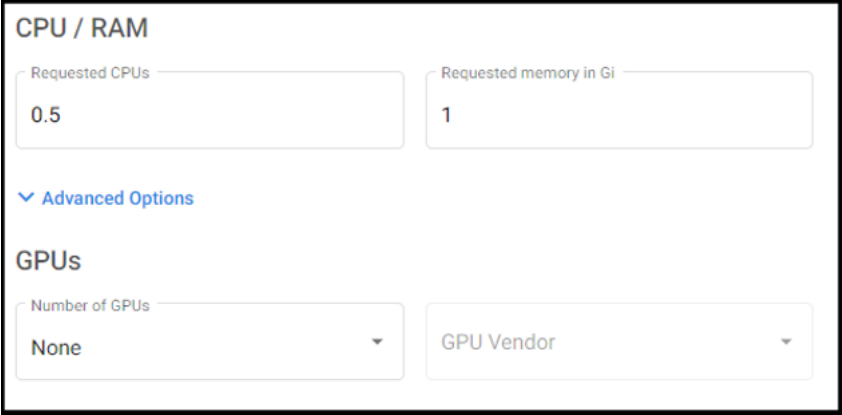
cuda가 있다면 GPU를 사용한다는 것이고, cuda가 없다면 CPU를 사용한다는 것이다.

포팅 매뉴얼

26

자신의 환경에 맞게 선택한다.

② CPU / GPU 할당



CPU는 소수점 단위로 할당 가능하고, GPU는 정수 단위(1, 2, 4, 8)로 할당 가능하다.

얼마나 할당할 지는 본인의 **할당 가능한 자원**을 살펴보면 된다.

할당을 잘 못할 경우, **0/[n] nodes available: insufficient [resource]** 와 같은 오류를 볼 수 있다.

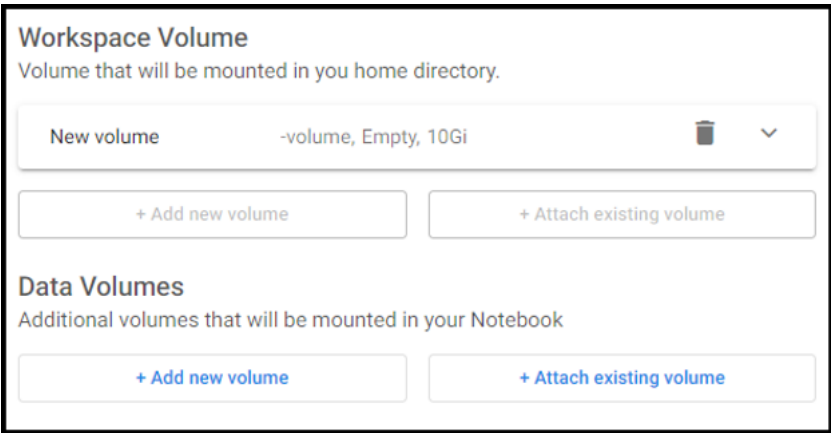
```
# node의 자원 사용 정보를 확인할 수 있다.  
$ kubectl describe nodes
```

```
Allocated resources:  
(Total limits may be over 100 percent, i.e., overcommitted.)  
Resource           Requests          Limits  
-----  
cpu                 5315m (66%)      74800m (935%)  
memory             7556Mi (23%)     45922176204800m (138%)  
ephemeral-storage  0 (0%)           0 (0%)  
hugepages-1Gi      0 (0%)           0 (0%)  
hugepages-2Mi      0 (0%)           0 (0%)  
nvidia.com/gpu     1                 1
```

위의 명령어를 통해 현재 요청한 CPU 양과, 제한된 CPU를 알 수 있다. / Core 1개는 1000m을 의미한다.

다른 곳에서 GPU 등 자원을 사용하고 있다면, GPU가 할당이 되지 않음을 기억할 것

③ Volume 설정



Workspace Volume의 경우, Notebook은 기본적으로 PV가 생성된다.

Data Volume에서 **[Attach existing volume]** 을 통해 기존에 만들어두었던 PV를 bounding시킬 수 있다.

Data Volumes

Additional volumes that will be mounted in your Notebook

Existing volume

workspace

Type

Kubernetes Volume

☐ Readonly

Name

workspace

Mount path

/home/jovyan/vol-1

Mount path는 Notebook에 mount될 volume의 위치를 나타내고 있다.

```
# 원하는 namespace에 있는 pod 확인
$ kubectl get pod -n {namespace}

# 원하는 pod의 정보 확인
$ kubectl describe pod -n {namespace} {pod name}
```

```
ubuntu@ip-10-0-1-10:~/kubeflow/section1_install$ kubectl get pod -n namespace1
NAME                                READY   STATUS    RESTARTS   AGE
ml-pipeline-ui-artifact-7cd897c59f-xnpst    2/2     Running   0           75m
ml-pipeline-visualizationserver-795f7db965-zdd4x  2/2     Running   0           75m
testcpu-0                                  2/2     Running   0           15m
testgpu-0                                  2/2     Running   0           18m
```

get pod 명령어 실행 시 나타나는 화면

Notebook 생성 후 Volume의 status를 보면,

workspace의 volume이 pending 상태에서 binding 상태로 바뀐 것을 확인할 수 있다.

☰

Kubeflow

namespace1 (Owner) ▼

🔗

Volumes

+ New Volume

Status	Name	Age	Size	Access Mode	Storage Class	
✔	test-gpu-volume	56 mins ago	10Gi	ReadWriteOnce	local-path	🗑
✔	testcpu-volume	17 mins ago	10Gi	ReadWriteOnce	local-path	🗑
✔	testgpu-volume	19 mins ago	10Gi	ReadWriteOnce	local-path	🗑
✔	workspace	58 mins ago	20Gi	ReadWriteOnce	local-path	🗑

Items per page: 10
1 - 4 of 4
|< < > >|

3. 노트북 선택

노트북이 생성되었다면 다음과 같은 창이 나온다.

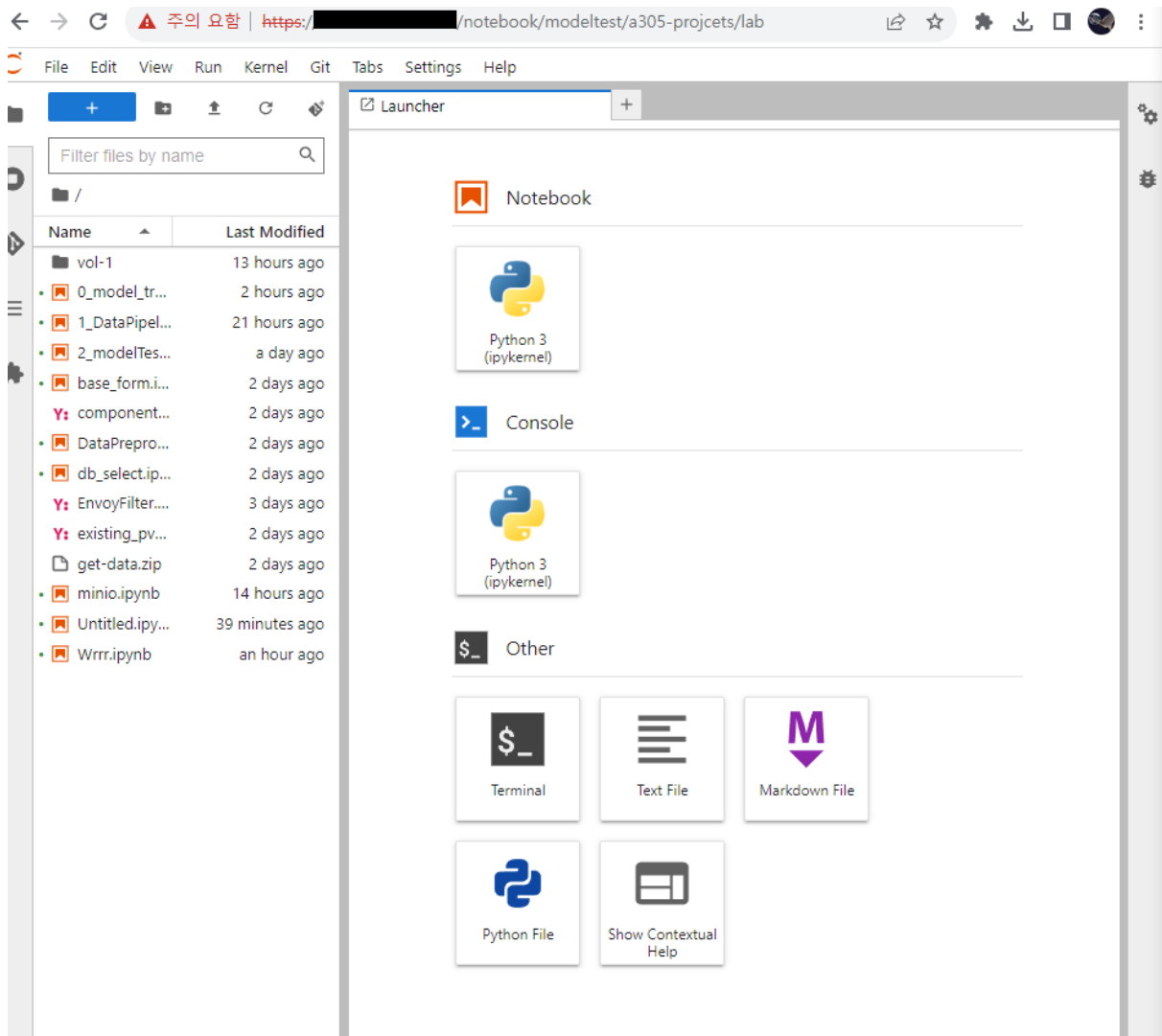
Notebooks

+ New Notebook

Status	Name	Type	Age	Image	GPUs	CPUs	Memory	Volumes	
✔	test2		11 mins ago	jupyter-scipyv1.5.0	0	100m	1Gi	⋮	CONNECT

Items per page: 10
1 - 1 of 1
|< < > >|

CONNECT 를 클릭하여 다음 창으로 들어간다.



4. 설정 파일 업로드

volume mount를 설정해 주면, 기본적으로 `vol-1` 와 같은 볼륨 디렉토리가 함께 생성된다.

해당 파일에 `influxDB` 접속을 위한 설정이 담긴 `config.ini` 파일을 생성한다.

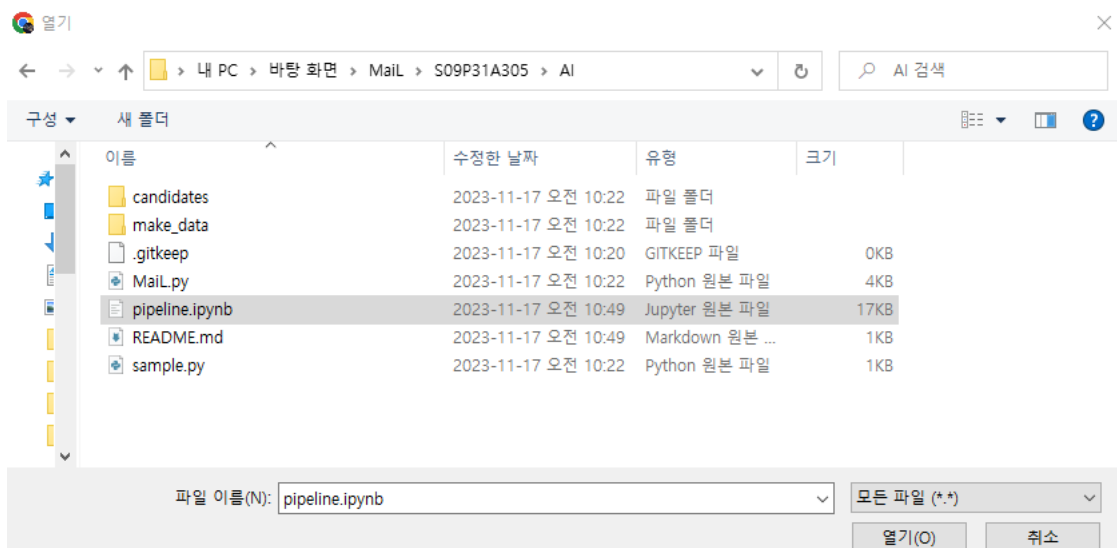
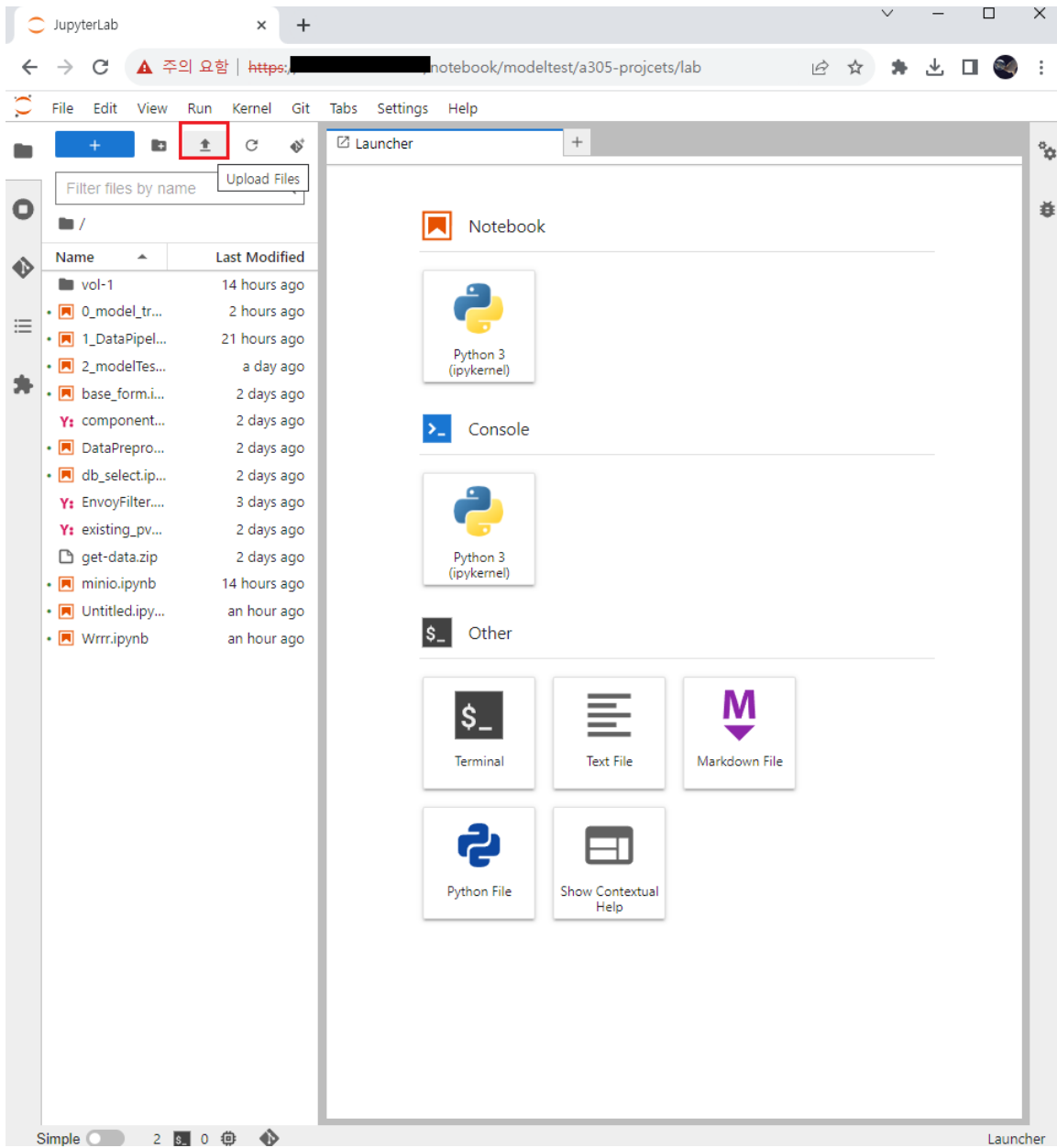
`config.ini` 파일 형식은 다음과 같다.

```
[influx2]
url=
org=
token=
timeout=
verify_ssl=False
```

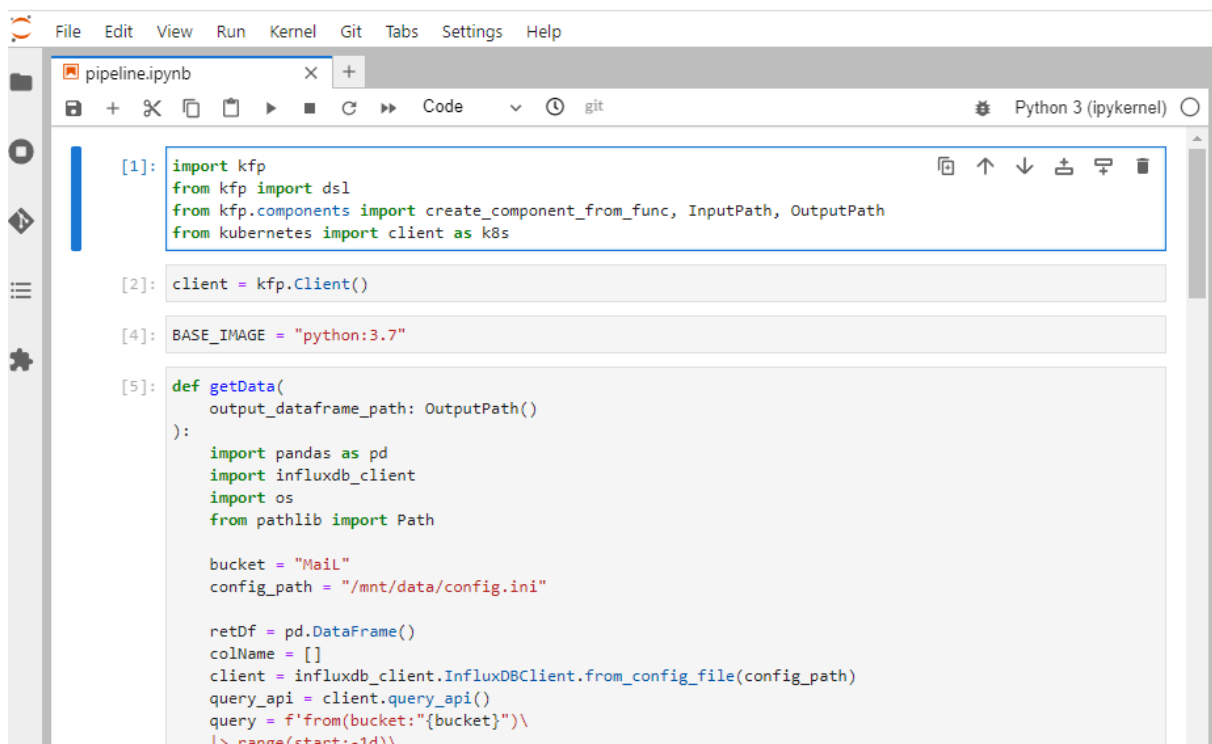
5. `pipeline.ipynb` 파일 업로드

MLOps를 위한 파이썬인 노트북 파일이 현재 프로젝트에 포함되어 있다.

`S09P31A305/AI/Pipeline.ipynb` 파일을 해당 경로에 업로드한다.



6. pipeline 실행



The screenshot shows a Jupyter Notebook window titled 'pipeline.ipynb'. The interface includes a top menu bar with 'File', 'Edit', 'View', 'Run', 'Kernel', 'Git', 'Tabs', 'Settings', and 'Help'. Below the menu is a toolbar with icons for saving, undo, redo, and other editing functions. The notebook contains five code cells:

```
[1]: import kfp
      from kfp import dsl
      from kfp.components import create_component_from_func, InputPath, OutputPath
      from kubernetes import client as k8s

[2]: client = kfp.Client()

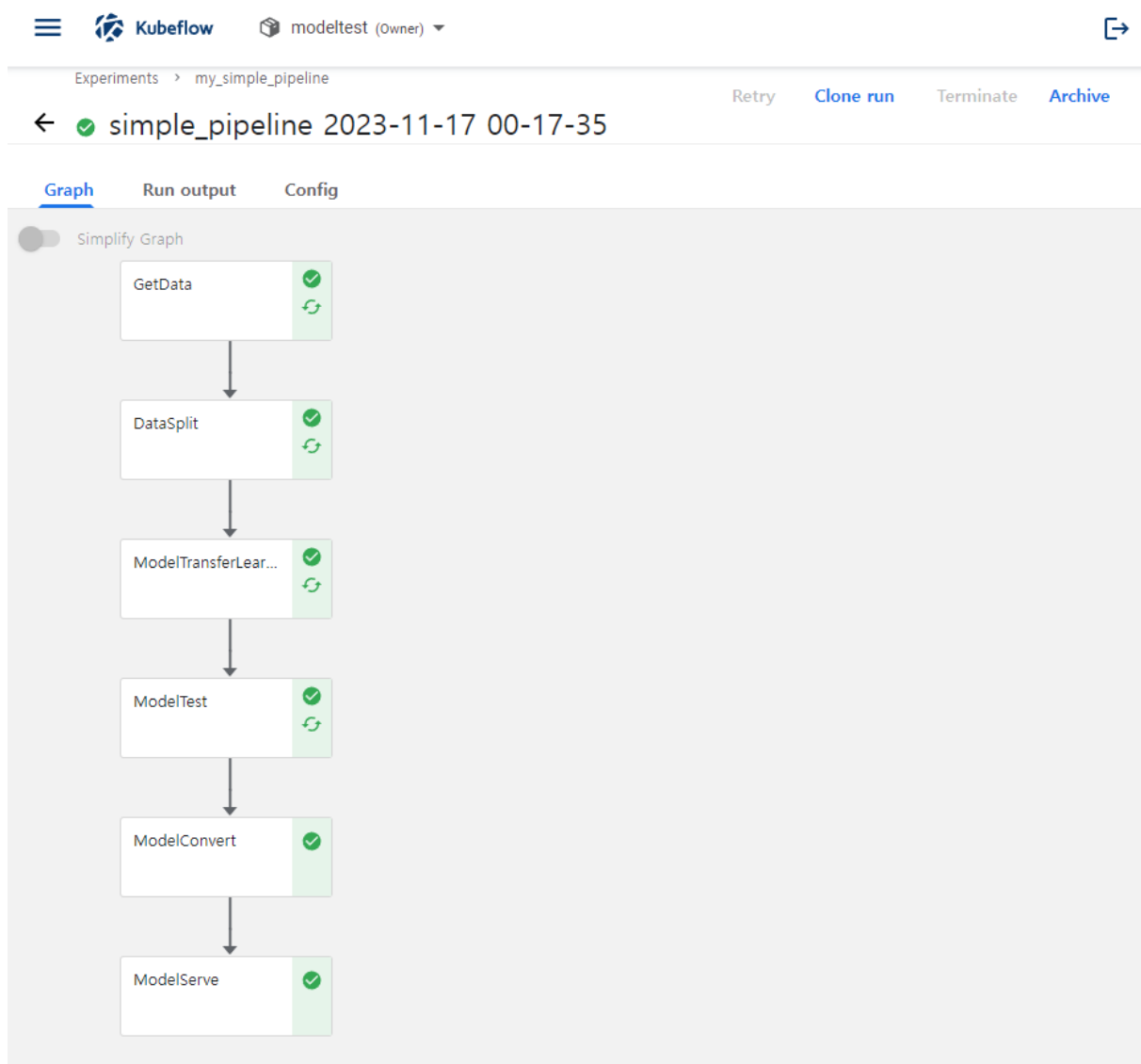
[4]: BASE_IMAGE = "python:3.7"

[5]: def getData(
      output_dataframe_path: OutputPath()
    ):
      import pandas as pd
      import influxdb_client
      import os
      from pathlib import Path

      bucket = "Mail"
      config_path = "/mnt/data/config.ini"

      retDf = pd.DataFrame()
      colName = []
      client = influxdb_client.InfluxDBClient.from_config_file(config_path)
      query_api = client.query_api()
      query = f'from(bucket:"{bucket}")\
      |> range(start:-1d)\'
```

해당 파일의 블록들을 전부 실행해 주면, 다음과 같이 kubeflow pipeline을 확인할 수 있다.



컴포넌트들은 하나의 함수로 구성되어 있어, 개발자의 필요에 의해 수정, 추가 혹은 제거를 할 수 있다.