OSC - UAV Use Case Guideline

Tags	技術 概念
№ ID	WD-72
Last edited time	@February 6, 2024 8:51 PM

Introduction

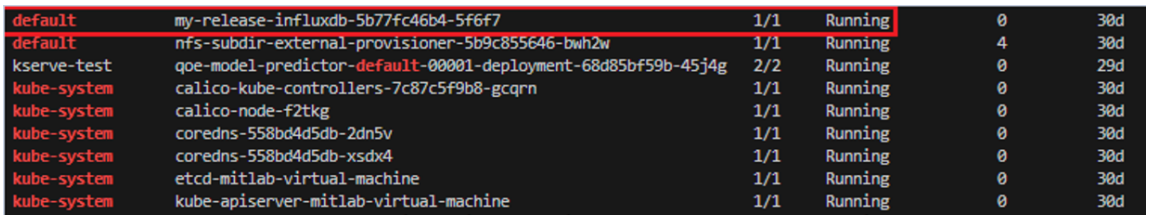
We have developed a new use case for OSC, successfully generating a new model for UAV path prediction by incorporating UAV data and modifying the pipeline.

Step 1. Add UAV data in InfluxDB

▼ 1-1 Create a new bucket

- find influxdb pod

```
kubectl get pods -A
```



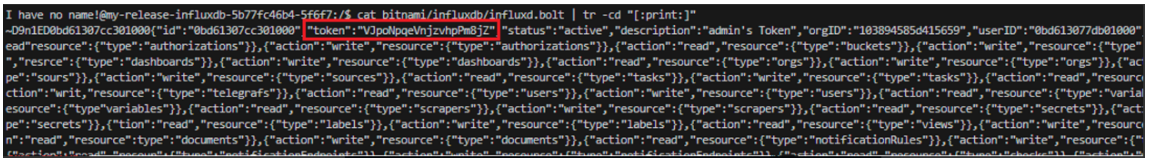
```
default       my-release-influxdb-5b77fc46b4-5f6f7                  1/1   Running   0   30d
default       nfs-subdir-external-provisioner-5b9c855646-bwh2w      1/1   Running   4   30d
kserve-test   qoe-model-predictor-default-00001-deployment-68d85bf59b-45j4g   2/2   Running   0   29d
kube-system   calico-kube-controllers-7c87c5f9b8-gcqrn              1/1   Running   0   30d
kube-system   calico-node-f2tkg                                     1/1   Running   0   30d
kube-system   coredns-558bd4d5db-2dn5v                              1/1   Running   0   30d
kube-system   coredns-558bd4d5db-xsdx4                              1/1   Running   0   30d
kube-system   etcd-mitlab-virtual-machine                           1/1   Running   0   30d
kube-system   kube-apiserver-mitlab-virtual-machine                 1/1   Running   0   30d
```

- get into the influxdb

```
kubectl exec -it my-release-influxdb-5b77fc46b4-5f6f7 -- b
```

- From below command we can get org name and access token

```
cat bitnami/influxdb/influxd.bolt | tr -cd "[:print:]"
```



```
I have no name!@my-release-influxdb-5b77fc46b4-5f6f7:/$ cat bitnami/influxdb/influxd.bolt | tr -cd "[:print:]"
~O9n1ED0bd61307cc301000{"id":"0bd61307cc301000","token":"VJpoNpqeVnjzvhpPm8jZ","status":"active","description":"admin's Token","orgID":"103894585d415659","userID":"0bd613077db01000"
ead"resource":{"type":"authorizations"}},{"action":"write","resource":{"type":"authorizations"}},{"action":"read","resource":{"type":"buckets"}},{"action":"write","resource":{"type"
","resrce":{"type":"dashboards"}},{"action":"write","resource":{"type":"dashboards"}},{"action":"read","resource":{"type":"orgs"}},{"action":"write","resource":{"type":"orgs"}},{"ac
pe":"sours"}},{"action":"write","resource":{"type":"sources"}},{"action":"read","resource":{"type":"tasks"}},{"action":"write","resource":{"type":"tasks"}},{"action":"read","resourc
ction":"writ,"resource":{"type":"telegrafs"}},{"action":"read","resource":{"type":"users"}},{"action":"write","resource":{"type":"users"}},{"action":"read","resource":{"type":"varia
esource":{"type"variables"}},{"action":"read","resource":{"type":"scrapers"}},{"action":"write","resource":{"type":"scrapers"}},{"action":"read","resource":{"type":"secrets"}},{"act
pe":"secrets"}},{"tion":"read","resource":{"type":"labels"}},{"action":"write","resource":{"type":"labels"}},{"action":"read","resource":{"type":"views"}},{"action":"write","resourc
n":"read","resource":type":"documents"}},{"action":"write","resource":{"type":"documents"}},{"action":"read","resource":{"type":"notificationRules"}},{"action":"write","resource":{"
```

- create a new bucket to store UAVData

```
influx bucket create -n UAVData -o primary -t <token>
```

For example：

```
influx bucket create -n UAVData -o primary -t VJpoNpqeVnjz
```

- check bucket lists

```
influx bucket list --org primary --token VJpoNpqeVnjzvhpPm
```

```
I have no name!@my-release-influxdb-5b77fc46b4-5f6f7:/$ influx bucket list --org primary --token VJpoNpqeVnjzvhpPm8jZ
ID                  Name            Retention       Shard group duration    Organization ID         Schema Type
843ee9b113635d9d    UAVData         infinite        168h0m0s                103894585d415659        implicit
4d219163d016dbfb    UEData          infinite        168h0m0s                103894585d415659        implicit
32cb15b323ef57cf    _monitoring     168h0m0s        24h0m0s                 103894585d415659        implicit
873e1b5d0ea6c982    _tasks          72h0m0s         24h0m0s                 103894585d415659        implicit
7f4bff75d6adf05d    primary         infinite        168h0m0s                103894585d415659        implicit
```

▼ 1-2 Copy and revise recipe file `RECIPE_EXAMPLE/example_recipe_latest_stable.yaml`

- Newfile_name : **example_recipe_latest_stable_copy.yaml**
- change bucket of datalake

```
traininghost:
  ip_address: 192.168.190.140


datalake:
  influxdb:
    host: 192.168.190.140
    port: 8086
    orgname: primary
    bucket: UAVData
    token: VJpoNpqeVnjzvhpPm8jZ
```

- Once updated, follow the below steps for reinstall of some components

```
bin/uninstall.sh
bin/install.sh -f RECIPE_EXAMPLE/example_recipe_latest_sta
```

▼ 1-3 Accessing applications in the cluster using port forwarding to send data

- revise insert.py
- File_path：aimlfw/qp/qp/insert.py

- Newfile_file：Mitlab_insert.py

```
import pandas as pd
from influxdb_client import InfluxDBClient
from influxdb_client.client.write_api import SYNCHRONOUS
import datetime

class INSERTDATA:
   def __init__(self):
        self.client = InfluxDBClient(url = "http://192.168


def explode(df):
     for col in df.columns:
             if isinstance(df.iloc[0][col], list):
                     df = df.explode(col)
             d = df[col].apply(pd.Series)
             df[d.columns] = d
             df = df.drop(col, axis=1)
     return df


def jsonToTable(df):
     df.index = range(len(df))
     cols = [col for col in df.columns if isinstance(df.il
     if len(cols) == 0:
             return df
     for col in cols:
             d = explode(pd.DataFrame(df[col], columns=[co
             d = d.dropna(axis=1, how='all')
             df = pd.concat([df, d], axis=1)
             df = df.drop(col, axis=1).dropna()
     return jsonToTable(df)

def data_normalization(df):
     import numpy as np
```

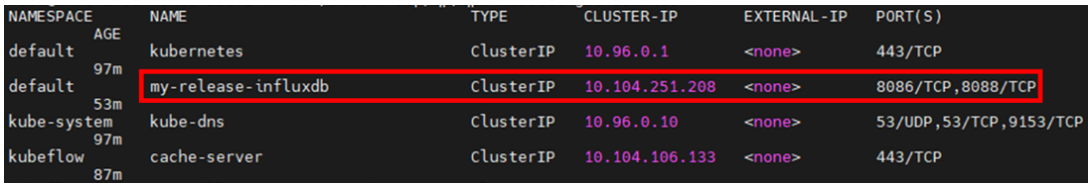
```
     df.drop('time', axis=1, inplace=True)
     # Normalization
     std = []
     mean = []

     # Iterate over columns
     for col in df.columns:
          std_val = df[col].std()
          mean_val = df[col].mean()
          std.append(std_val)
          mean.append(mean_val)
          df[col] = (df[col] - mean_val) / std_val
     return df

def time(df):
     df.index = pd.date_range(start=datetime.datetime.now(
     df['time'] = df['time'].apply(lambda x: str(x))
     return df

def populatedb():
     df = pd.read_csv('/home/mitlab/osc/aimlfw-dep/qp/qp/o
     df = jsonToTable(df)
     df = time(df)
     df = data_normalization(df)
     print(df)
     db = INSERTDATA()
     write_api = db.client.write_api(write_options=SYNCHRO
     write_api.write(bucket="UAVData",record=df, data_fram

populatedb()
```

```python
from influxdb_client import InfluxDBClient
from influxdb_client.client.write_api import SYNCHRONOUS
import datetime

# num=0

class INSERTDATA:
    def __init__(self):
        self.client = InfluxDBClient(url = "http://192.168.190.140:8086", token="VJpoNpqeVnjzvhpPm8jZ")


def explode(df):
    for col in df.columns:
        if isinstance(df.iloc[0][col], list):
            df = df.explode(col)
        d = df[col].apply(pd.Series)
        df[d.columns] = d
        df = df.drop(col, axis=1)
    return df


def jsonToTable(df):
    df.index = range(len(df))
    cols = [col for col in df.columns if isinstance(df.iloc[0][col], dict) or isinstance(df.iloc[0][col], list)]
    if len(cols) == 0:
        return df
    for col in cols:
        d = explode(pd.DataFrame(df[col], columns=[col]))
        d = d.dropna(axis=1, how='all')
        df = pd.concat([df, d], axis=1)
        df = df.drop(col, axis=1).dropna()
    return jsonToTable(df)

def data_normalization(df):
    import numpy as np
    df.drop('time', axis=1, inplace=True)
    # Normalization
    std = []
    mean = []

    # Iterate over columns
    for col in df.columns:
        std_val = df[col].std()
        mean_val = df[col].mean()
        std.append(std_val)
        mean.append(mean_val)
        df[col] = (df[col] - mean_val) / std_val
    return df

def time(df):
    df.index = pd.date_range(start=datetime.datetime.now(), freq='10ms', periods=len(df))
    df['time'] = df['time'].apply(lambda x: str(x))
    return df

def populatedb():
    df = pd.read_csv('/home/mitlab/osc/aimlfw-dep/qp/qp/original_dataset.csv')
    df = jsonToTable(df)
    df = time(df)
    df = data_normalization(df)
    print(df)
    db = INSERTDATA()
    write_api = db.client.write_api(write_options=SYNCHRONOUS)
    write_api.write(bucket="UAVData",record=df, data_frame_measurement_name="liveCell",org="primary")

populatedb()
```

- Follow below command to port forward to access Influx DB

- **Step 1：Check influx service name and port**

```
kubectl get service -A
```



NAMESPACE	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
default	kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	97m
default	my-release-influxdb	ClusterIP	10.104.251.208	<none>	8086/TCP,8088/TCP	53m
kube-system	kube-dns	ClusterIP	10.96.0.10	<none>	53/UDP,53/TCP,9153/TCP	97m
kubeflow	cache-server	ClusterIP	10.104.106.133	<none>	443/TCP	87m

- **Step 2：Open new terminal and follow below command to port forward to Influx DB**

```
kubectl port-forward svc/<Your influxDB service name> 8
```

For example：

```
kubectl port-forward svc/my-release-influxdb 8086:8086
```

- **Step 3：Back to the terminal and run this command to insert data**

```
python3 Mitlab_insert.py
```

- To check inserted data in Influx DB, execute below command inside the Influx DB container:
- **Step 1. Get into influxdb pod**

```
kubectl exec -it my-release-influxdb-5b77fc46b4-5f6f7 -
```

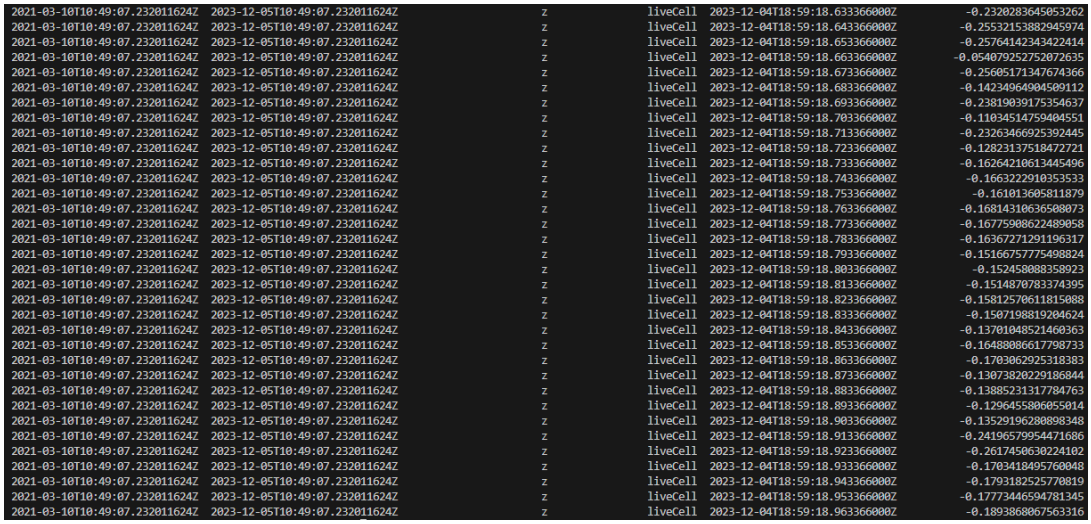
- **Step 2. Check the data in the container**

```
influx query  'from(bucket: "UAVData") |> range(start:
```

For example:

```
influx query  'from(bucket: "UAVData") |> range(start:
```

and you will see the information like this figure.



2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.633366000Z	-0.232028364505326
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.643366000Z	-0.25532153882945974
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.653366000Z	-0.25764142343422414
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.663366000Z	-0.05407925275207263
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.673366000Z	-0.25605171347674366
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.683366000Z	-0.14234964904509112
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.693366000Z	-0.23819039175354637
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.703366000Z	-0.11034514759404551
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.713366000Z	-0.23263466925392445
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.723366000Z	-0.12823137518472721
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.733366000Z	-0.16264210613445496
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.743366000Z	-0.16632229103535533
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.753366000Z	-0.161013605811879
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.763366000Z	-0.16814310636508073
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.773366000Z	-0.16775908622489058
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.783366000Z	-0.16367271291196317
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.793366000Z	-0.15166757775498824
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.803366000Z	-0.152458088358923
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.813366000Z	-0.15148707833374395
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.823366000Z	-0.15812570611815088
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.833366000Z	-0.1507198819204624
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.843366000Z	-0.13701048521460363
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.853366000Z	-0.16488086617798733
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.863366000Z	-0.1703062925318383
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.873366000Z	-0.13073820229106844
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.883366000Z	-0.13885231317784763
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.893366000Z	-0.1296455806055014
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.903366000Z	-0.13529196280898348
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.913366000Z	-0.24196579954471686
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.923366000Z	-0.26174506302241O2
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.933366000Z	-0.17034184957600048
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.943366000Z	-0.17931825257708l9
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.953366000Z	-0.17773446594781345
2021-03-10T10:49:07.232011624Z	2023-12-05T10:49:07.232011624Z	z	liveCell	2023-12-04T18:59:18.963366000Z	-0.1893868067563316

Step 2. Create a pipeline

▼ 2-1 create a new pipeline

- Revise OSC qoe.iphpy
- File_path：connect 32088 port
- Newfile_path：Mitlab_pipeline.iphpy

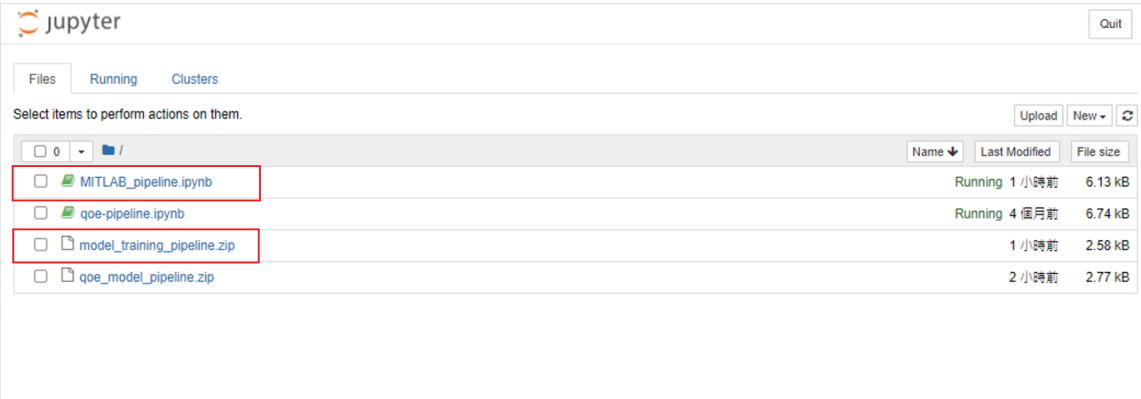
```
import kfp
import kfp.components as components
import kfp.dsl as dsl
from kfp.components import InputPath, OutputPath

def train_export_model(trainingjobName: str, epochs: str,

    import tensorflow as tf
```

```
from numpy import array
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dr
import numpy as np
print("numpy version")
print(np.__version__)
import pandas as pd
import os
from featurestoresdk.feature_store_sdk import FeatureS
from modelmetricsdk.model_metrics_sdk import ModelMetr

import requests
from tensorflow import keras
from sklearn.model_selection import train_test_split

fs_sdk = FeatureStoreSdk()
mm_sdk = ModelMetricsSdk()

features = fs_sdk.get_features(trainingjobName, ['x',
features = features.astype('float64')
print("Dataframe:")
print(features)
print(features.dtypes)

features = features.to_numpy()
print("features.dtype:")
print(features.dtype)

one_set_data_count = 2500
set_count = 1
input_data_series_count = 100  # hyper parameter
separate_data = []
series = []
label = []
for i in range(set_count):
    separate_data.append(features[i*one_set_data_count
```

```
            for j in range(one_set_data_count-input_data_serie
                series.append(separate_data[i][j:j+input_data_
                label.append(separate_data[i][j+input_data_ser
        separate_data = np.array(separate_data)
        series = np.array(series)
        label = np.array(label)

        # make train and test
        x_train, x_test, y_train, y_test = train_test_split(se
        x_test, x_validate, y_test, y_validate = train_test_sp
        print("x_train:", x_train.dtype)
        print("y_train:", y_train.dtype)
        print("x_validate:", x_validate.dtype)
        print("y_validate:", y_validate.dtype)



        # Define the model architecture
        model = Sequential()
        model.add(LSTM(units=64, input_shape=(input_data_serie
        model.add(Dropout(0.3))
        model.add(Dense(32, activation='relu'))
        model.add(Dense(3, activation='linear'))

        # Compile the model
        model.compile(loss='mean_squared_error', optimizer='ad

        # Train the model on your data
        batch_size = 4
        model.fit(x_train, y_train, batch_size=batch_size, epo

        model.save("./")
        mm_sdk.upload_model("./", trainingjobName, version)

BASE_IMAGE = "traininghost/pipelineimage:latest"

def train_and_export(trainingjobName: str, epochs: str, ve
```

```
    trainOp = components.func_to_container_op(train_export
    # Below line to disable caching of pipeline step
    trainOp.execution_options.caching_strategy.max_cache_s
    trainOp.container.set_image_pull_policy("IfNotPresent"

@dsl.pipeline(
    name='MITLAB_training_pipeline',
    description='UAVData_pipeline'
)
def super_model_pipeline(
    trainingjob_name: str, epochs: str, version: str):

    train_and_export(trainingjob_name, epochs, version)

pipeline_func = super_model_pipeline
file_name = "model_training_pipeline"
kfp.compiler.Compiler().compile(pipeline_func, '{}.zip'.fo

import requests
pipeline_name="MITLAB_training_pipeline"
pipeline_file = file_name+'.zip'
requests.post("http://tm.traininghost:32002/pipelines/{}/u
```

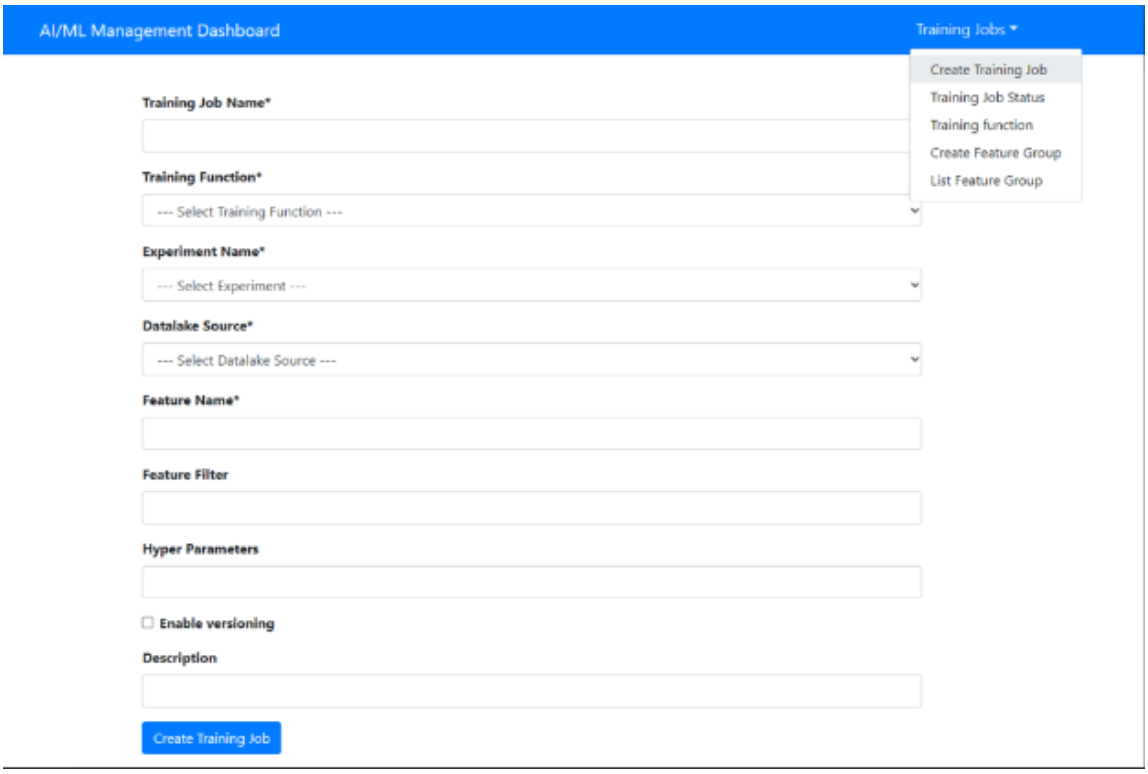
- Produce a new pipeline



Step 3. Creating training job

▼ 3-1 Creating an new training job

- Create an new training job on aiml-dashboard

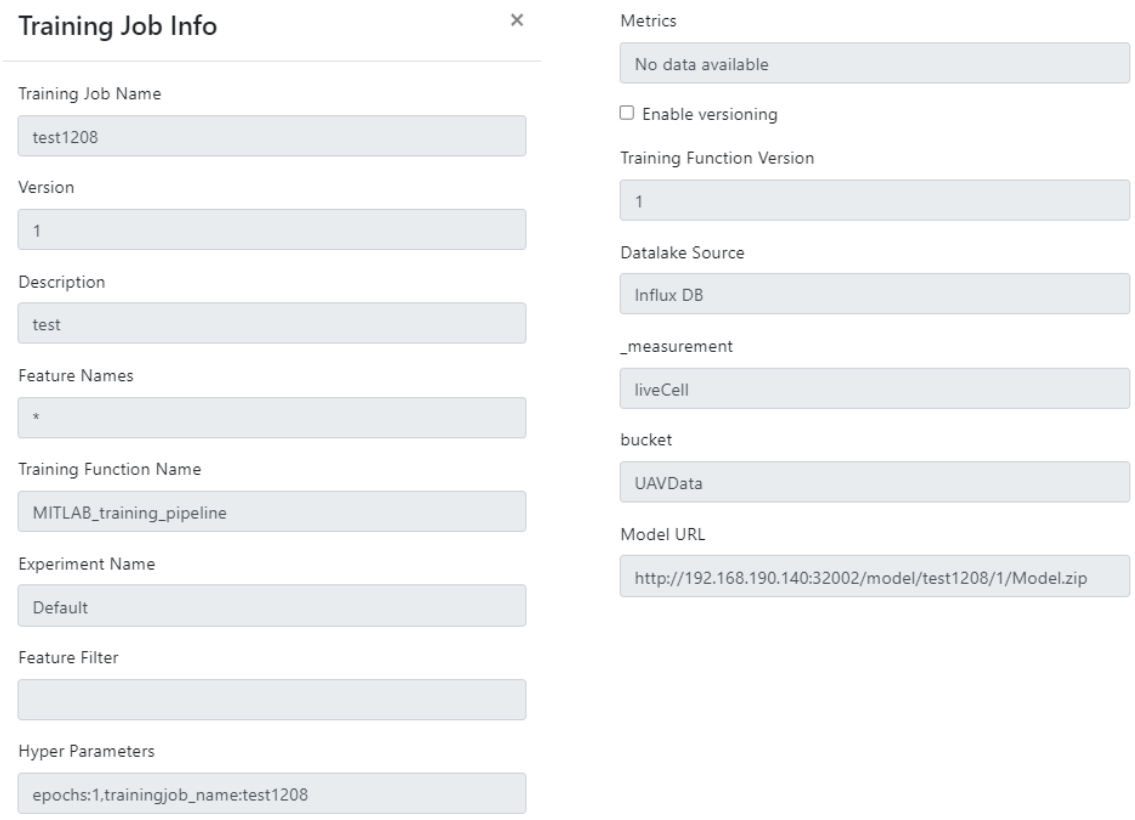


- Use the default parameter

Parameter	Value
Training Job Name	test1208
Training Function	model_training_pipeline
Training Function Version Name	1
Experiment Name	Default
Datalake Source	Influx DB
_measurement	liveCell
bucket	UAVData
Feature Name	*

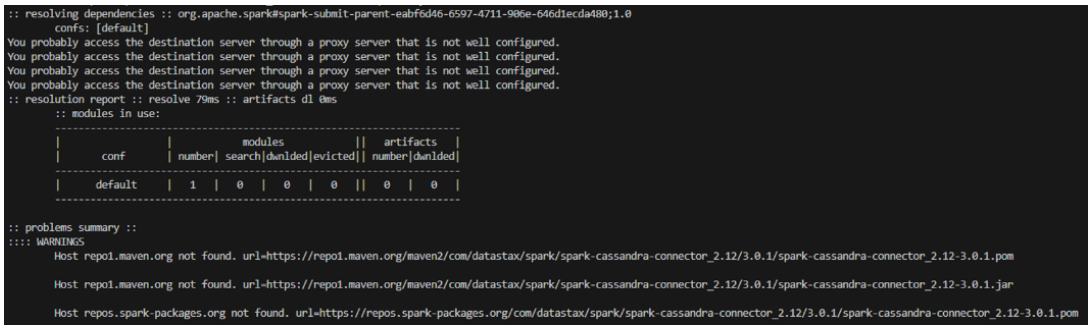
Feature Filter	
Hyper Parameters	epochs:1
Description	test

- training job info



▼ 3-2 Data extraction

- **(Problem) The module cannot successfully downloaded in the data exaction pod.**
  - Data extraction pod error message (**CoreDNS Problem**)

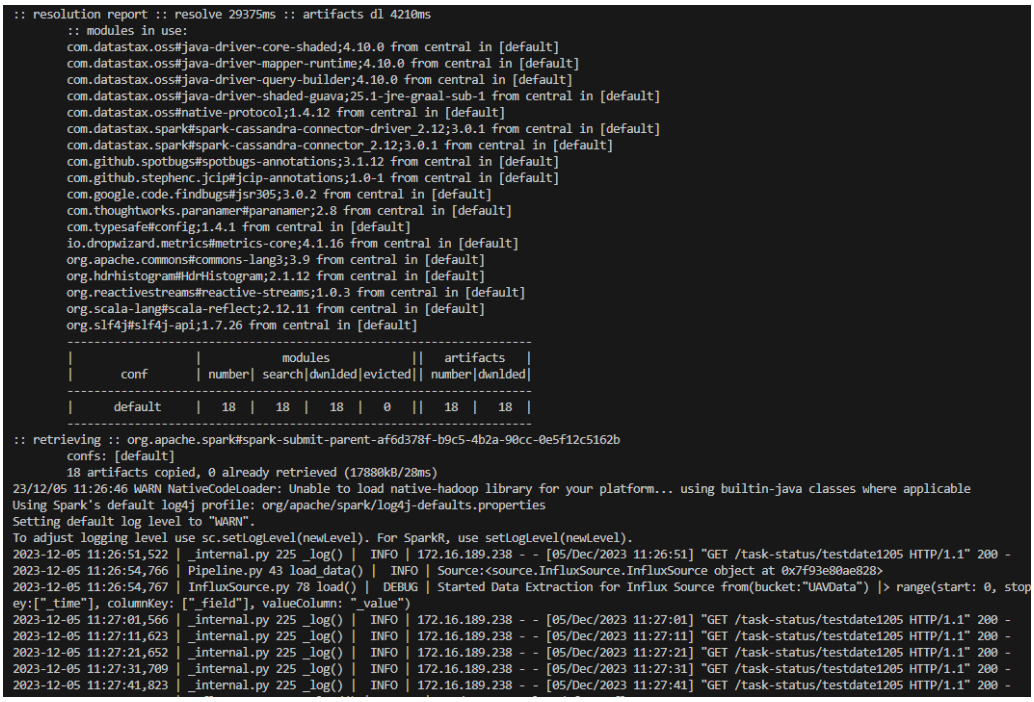


- To reslove **CoreDNS Problem** in kubernetes:
  - Step 1. Enter the data extraction podand **add nameserver 8.8.8.8**(Google's DNS server) to /etc/resolv.conf in the pod ,restart the data extraction pod and restart the training job again to download the essential module.

```
kubectl exec -it --namespace=traininghost data-extra
```

```
cat << EOF > /etc/resolv.conf
nameserver 8.8.8.8
nameserver 10.96.0.10
search traininghost.svc.cluster.local svc.cluster.lo
options ndots:5
EOF
```

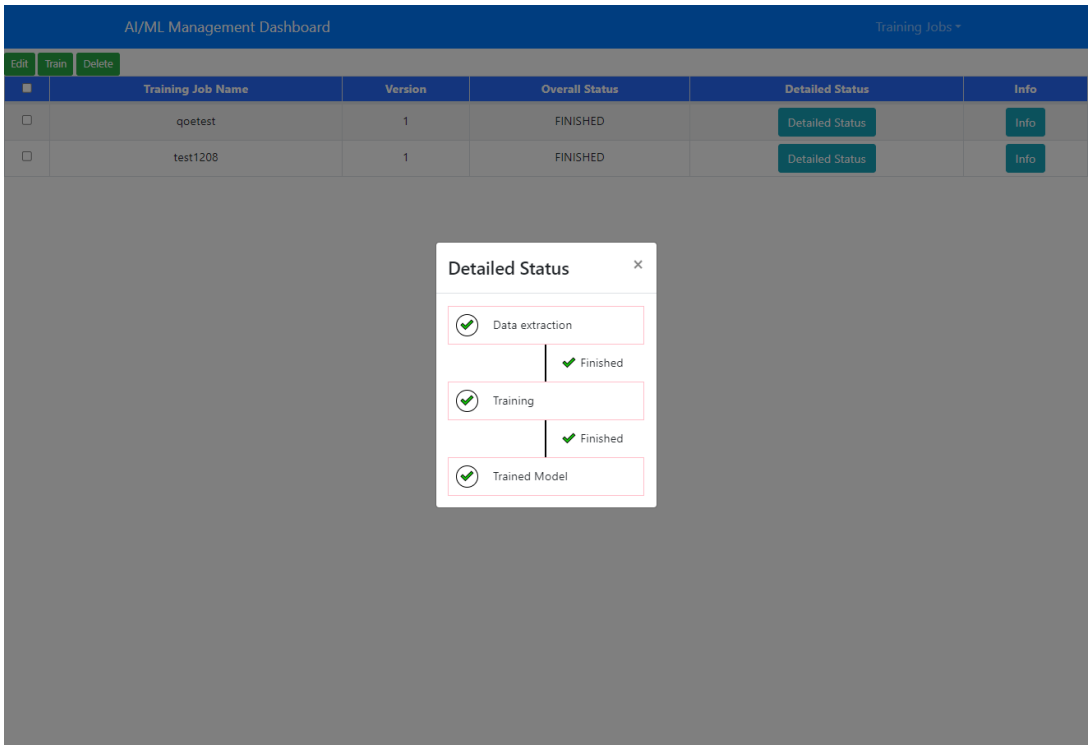
    check the modules already downloaded or not.



- Step 2. After the pod successfully downloads the module, enter the data extraction pod and **restore /etc/resolv.conf**.

```
cat << EOF > /etc/resolv.conf
nameserver 10.96.0.10
search trainninghost.svc.cluster.local svc.cluster.lo
options ndots:5
EOF
```

- Re-execute the training job, wait for minutes then the model is complete.



Step 4. Test predictions using model deployed on Kserve

▼ 4-1 Deploy trained UAV prediction model on Kserve

- Create `uav.yaml` file with below contents

```
apiVersion: "serving.kserve.io/v1beta1"
kind: "InferenceService"
metadata:
  name: uav-model
spec:
  predictor:
    tensorflow:
      storageUri: "http://192.168.190.140:32002/model/test
      runtimeVersion: "2.5.1"
      resources:
        requests:
          cpu: 1
          memory: 0.5Gi
```

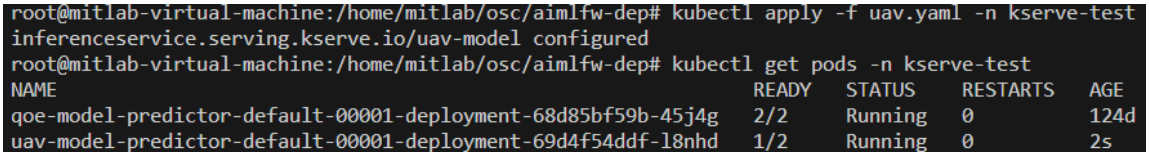
```
        limits:
          cpu: 2
          memory: 0.5Gi
```

- deploy uav-model

```
kubectl apply -f uav.yaml -n kserve-test
```

- Check running state of pod

```
kubectl get pods -n kserve-test
```



▼ 4-2 (problem) Failed calling webhook "inferenceservice.kserve-webhook-server.defaulter"

- deleting 2 webhook config

```
kubectl delete mutatingwebhookconfigurations.admissionregi
kubectl delete validatingwebhookconfigurations.admissionre
```

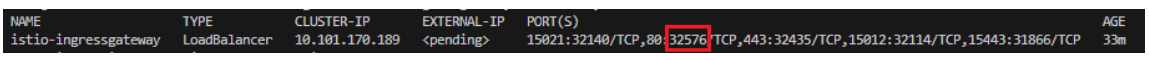
- inferenceservice.serving.kserve.io/uav-model configured

```
kubectl apply -f uav.yaml -n kserve-test
```

▼ 4-3 Test predictions using model deployed on Kserve

- Use below command to obtain Ingress port for Kserve.

```
kubectl get svc istio-ingressgateway -n istio-system
```



- Create `uav.sh` file with following contents

```
nano uav.sh
```

- Copy the below content and update the **“IP of host”** where Kserve is deployed and ingress **“port”** of Kserve obtained using above method.

```
model_name=uav-model
curl -v -H "Host: $model_name.kserve-test.example.com" htt
```

For example:

```
model_name=uav-model
curl -v -H "Host: $model_name.kserve-test.example.com" htt
```

- After complete update, create sample data for predictions in file **input_uav.json**.

```
nano input_uav.json
```

Add the following content in `input_uav.json` file.

```
{"signature_name": "serving_default", "instances": [[[0.74
       [0.9348491377517286, 0.5797269721343681, 0.15480809
       [0.32685562293198867, 0.40325520693639316, 0.379231
```

- Use command below to trigger predictions.

```
source uav.sh
```