Rice Simple Feature Store Demo

December 7, 2021

1 Rice Simple Feature Store Demo

This is a demo for the rice-sfs system. In this notebook, we would see how to use the system to manage the features. Specifically, we would ingest some features into the system and retrieve them later. Through this process, you will see the workflow of a typical user of the system.

1.1 Setup

This section help you setup the environment needed in order to run the notebook.

We perform all the computation and storing in you local machine.

1.1.1 Local DynamoDB

You have to have a local version of DynamoDB server running on your machine.

To run a DynamoDB locally, first go to AWS DynamoDB Developer Guide , download the archive, and put the extracted `DynamoDBLocal.jar` and `DynamoDBLocal_lib/` in `dynamodb/` directory.

Then run:

```
bash dynamodb/start_dynamo_local.sh
```

Use aws-cli to access DynamoDB:

```
aws dynamodb --endpoint-url http://localhost:8000 <command> [args]
```

1.1.2 Local PostgreSQL Server

You have to have a local version of PostgreSQL server running on your machine.

You can download a PostgreSQL for your OS here and start the server in the backgroud.

By default the PostgreSQL server would run on `localhost:5432`.

After getting PostgreSQL running, you should create a user and a database:

```
# create user "rice"
createuser --createdb rice

# create database "sfs"
createdb --username rice --no-password sfs
```

We would use the user `rice` to access the database `sfs`.

1.1.3 Local Spark

You have to have a local Spark cluster running on your machine.

You can find instructions on intalling and deploying a Spark on your local machine here.

After getting spark running in the background, you can use `spark-shell` to start a Scala shell:

```
spark-shell --packages io.delta:delta-core_2.12:1.1.0 --conf "spark.sql.extensions=io.delta.sq
```

Note that we add Delta package so that later we can directly query delta lake.

1.1.4 Download Dataset

We use the MovieLens 100K Dataset. You can download it [here[(https://grouplens.org/datasets/movielens/100k/).

After you download the dataset, please put the extracted `ml-100k/` in `example/data/` directory.

1.1.5 Start the services

Go to the root directory of the project. Run `mvn clean package` to package the program.

Go the `sfs-registry/` directory. Run `mvn spring-boot:run` to start the feature registry service. By default it will listen to port `8081`.

Start another terminal, go to the `sfs-serving/` directory. Run `mvn spring-boot:run` to start the feature serving service. By default it will listen to port `8082`.

[8]:
```
import pandas as pd
import requests
import os

%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

1.2 Feature Registry

We first register some features in the system.

[2]:
```
registry_endpoint = "http://localhost:8081/api/v1"
serving_endpoint = "http://localhost:8082/api/v1"
```

[6]:
```
# register the user entity
requests.post(registry_endpoint + "/entities", json={"name":
 ↪"user","description":"user in the system"}).json()
```

[6]:
```
{'name': 'user', 'description': 'user in the system'}
```

```
[7]: # register some features
     requests.post(registry_endpoint + "/features",
                   json={"name":"user_age","description":"user age","valueType":
      ↪"INT","deltaTableColName":"age","dynamoTableColName":"age"}).json()
     requests.post(registry_endpoint + "/features",
                   json={"name":"user_gender","description":"user␣
      ↪gender","valueType":"STRING","deltaTableColName":
      ↪"gender","dynamoTableColName":"gender"}).json()
     requests.post(registry_endpoint + "/features",
                   json={"name":"user_occupation","description":"user␣
      ↪occupation","valueType":"STRING","deltaTableColName":
      ↪"occupation","dynamoTableColName":"occupation"}).json()
```

```
[7]: {'name': 'user_occupation',
      'description': 'user occupation',
      'valueType': 'STRING',
      'deltaTableColName': 'occupation',
      'dynamoTableColName': 'occupation'}
```

```
[12]: # register a feature table
      path = os.getcwd()
      pj_root = os.path.join(path, os.pardir)
      delta_root = os.path.join(pj_root, "delta/")
      body = {
          "name":"user_feat",
          "description":"user feature",
          "entity":"user",
          "features":["user_age","user_gender"],
          "dynamoTableName":"user_feat",
          "deltaTablePath":os.path.abspath(os.path.join(delta_root, "user_feat/"))
      }
      requests.post(registry_endpoint + "/featureTables", json=body).json()
```

```
[12]: {'name': 'user_feat',
       'description': 'user feature',
       'entity': {'name': 'user', 'description': 'user in the system'},
       'features': [{'name': 'user_age',
         'description': 'user age',
         'valueType': 'INT',
         'deltaTableColName': 'age',
         'dynamoTableColName': 'age'},
        {'name': 'user_gender',
         'description': 'user gender',
         'valueType': 'STRING',
         'deltaTableColName': 'gender',
         'dynamoTableColName': 'gender'}],
       'deltaTablePath': '/Users/freemso/dev/cloud-536/rice-sfs/delta/user_feat',
 'dynamoTableName': 'user_feat'}
```

1.3 Feature Ingestion

We would ingest some user features stored in the `ml-100k/u.user` file.

```
[ ]: !spark-submit \
       --master "local[4]" \
       --deploy-mode client \
       --class edu.rice.sfs.ingest.BatchIngestApp \
       --name BatchIngestApp \
       --packages com.audienceproject:spark-dynamodb_2.12:1.1.2 \
       --packages io.delta:delta-core_2.12:1.1.0 \
       --conf "spark.driver.extraJavaOptions=-Daws.dynamodb.endpoint=http://
     ↪localhost:8000/" \
       --conf "spark.sfs.source.batch.options=delimiter:|" \
       --conf "spark.sfs.source.batch.format=csv" \
       --conf "spark.sfs.source.batch.path=$(pwd)/data/ml-100k/u.user" \
       --conf "spark.sfs.source.batch.cols=id, age, gender, occupation, zip_code" \
       --conf "spark.sfs.source.batch.feature-table=user_feat" \
       --conf "spark.sfs.source.batch.entity-col=id" \
       --conf "spark.sfs.source.batch.feature-col-map=user_age:age,user_gender:
     ↪gender" \
       --driver-java-options "-Daws.dynamodb.endpoint=http://localhost:8000/" \
       --driver-memory 1024M \
       --driver-cores 1 \
       --executor-memory 1G \
       $(dirname `pwd`)/sfs-ingest/target/sfs-ingest-0.0.
     ↪1-SNAPSHOT-jar-with-dependencies.jar
```

Now you can use spark-shell to see if the feature is actually ingested.

1.4 Feature Serving

We now query the serving service to get the online feature from DynamoDB.

```
[14]: # first register a feature table view.
      body = {
          "name":"user_feat_view",
          "featureTableName":"user_feat",
          "featureNames":["user_age"] # only care about the ages
      }
      requests.post(registry_endpoint + "/featureTableViews", json=body).json()
```

```
[14]: {'name': 'user_feat_view',
       'featureTableName': 'user_feat',
       'featureNames': ['user_age']}
```

[15]:
```
requests.get(serving_endpoint + "/getFeature", params={"featureTableView":␣
↪"user_feat_view", "entity": "42"}).json()
```

[15]:
```
{'age': 30}
```

[16]:
```
# if we care more features, we can use a different view
body = {
    "name":"user_feat_view_2",
    "featureTableName":"user_feat",
    "featureNames":["user_age", "user_gender"] # only care about the ages
}
requests.post(registry_endpoint + "/featureTableViews", json=body).json()
```

[16]:
```
{'name': 'user_feat_view_2',
 'featureTableName': 'user_feat',
 'featureNames': ['user_age', 'user_gender']}
```

[17]:
```
requests.get(serving_endpoint + "/getFeature", params={"featureTableView":␣
↪"user_feat_view_2", "entity": "42"}).json()
```

[17]:
```
{'gender': 'M', 'age': 30}
```