記事
Toshihiko Minamoto · 2022年1月25日 9m read

IRISデータベースへのPython ODBC接続 - 2つ目の簡易メモ

キーワード: PyODBC、unixODBC、IRIS、IntegratedML、Jupyterノートブック、Python 3

目的

数か月前、私は「IRISデータベースへのPython JDBC接続
」という簡易メモを書きました。以来、PCの奥深くに埋められたスクラッチパッドよりも、その記事を頻繁に参照しています。 そこで今回は、もう一つの簡易メモで「IRISデータベースへのPython
ODBC接続」を作成する方法を説明します。

ODBCとPyODCBをWindowsクライアントでセットアップするのは非常に簡単なようですが、Linux/Unix系サーバーでunixODBCとPyODBCクライアントをセットアップする際には毎回、どこかで躓いてしまいます。

バニラLinuxクライアントで、IRISをインストールせずに、リモートIRISサーバーに対してPyODBC/unixODBCの配管をうまく行うための単純で一貫したアプローチがあるのでしょうか。

範囲

最近、Linux Docker環境のJupyterノートブック内でゼロからPyODBCデモを機能させるようにすることに少しばかり奮闘したことがありました。
そこで、少し冗長的ではありますが、後で簡単に参照できるように、これをメモに残しておくことにしました。

範囲内

このメモでは、以下のコンポーネントに触れます。

PyODBC over **unixODBC**
TensorFlow 2.2とPython 3を使った**Jupyter Notebook**サーバー
サンプルテストデータを含むIntegratedMLを使った**IRIS2020.3** CEサーバー
この環境内で:

AWS **Ubuntu** 16.04における**docker-compose**によるDockerエンジン
Docker Desktop for **MacOS**、および**Docker Toolbox for Windows** 10もテストされます。

範囲外:

繰り返しになりますが、このデモ環境で機能しない部分は評価されません。
それらは重要なことであり、以下のようにサイト固有の機能である可能性があります。

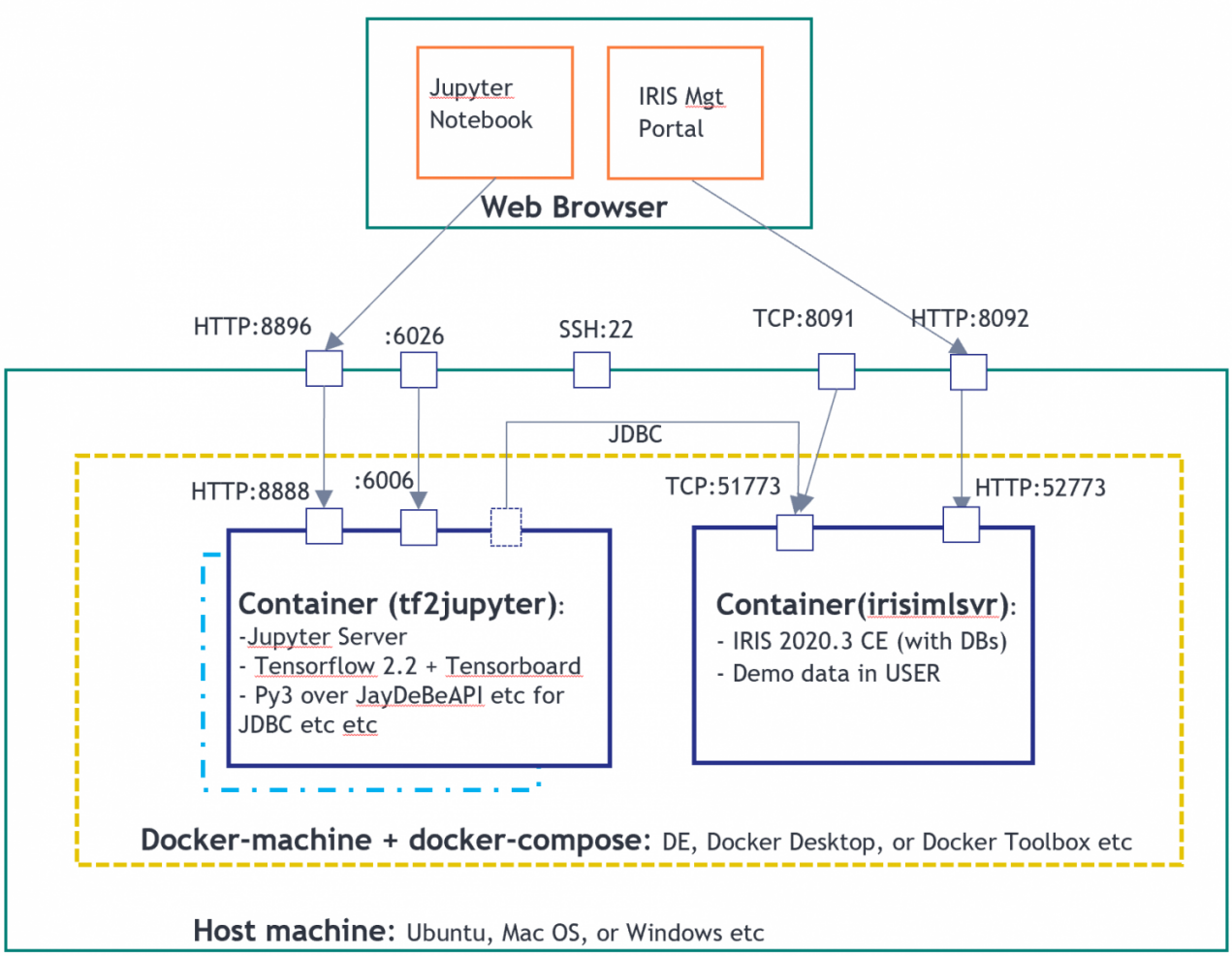
エンドツーエンドのセキュリティと監査
パフォーマンスとスケーラビリティ
ライセンスとサポート可能性など

環境

以下の構成とテスト手順では、任意のバニラLinux
Dockerイメージを使用できますが、以下のようにすると、そのような環境を5分で簡単にセットアップできます。

1. このデモテンプレートをGit **clone**します。
2. クローンされた、docker-compose.ymlファイルを含むディレクトリで"**docker-compose up -d**"を実行します。

以下のトポロジーのように、2つのコンテナーのデモ環境が作成されます。
1つはPyODBCクライアントとしてのJupyter Notebookサーバー用で、もう1つはIRIS2020.3 CEサーバー用です。



上記の環境では、tf2jupyterには"Python over
JDBC"クライアント構成しか含まれておらず、ODBCまたはPyODBCクライアント構成はまだ含まれていません。

そこで、わかりやすくするために、以下の手順を実行して、Jupyter
Notebook内から直接それらをセットアップしましょう。

手順

AWS Ubuntu 16.04サーバーで、以下の構成とテストを実行しました。
私の同僚の@Thomas.Dyarは、MacOSで実行しました。 また、Docker Toolbox for
Windowsでも簡単にテストされていますが、 何らかの問題に遭遇した場合は、お知らせください。

以下の手順は、Dockerfileに簡単に自動化できます。
ここでは、どのように行われたかを数か月後に忘れてしまった場合に備えて、手動で記録しました。

1. 公式ドキュメント

IRISのODBCサポート
UnixにおけるODBCデータソースの定義
IRISのPyODBCサポート

2. Jupyterサーバーに接続する

私はローカルのPuttyのSSHトンネリングを使用してリモートのAWS
Ubuntuポート22に接続し、上記のトポロジーのようにポート8896にマッピングしました。

（ローカルのdocker環境では、たとえば、直接dockerマシンのIP:8896にHTTP接続することもできます。）

3. Jupyterノートブック内からODBCインストールを実行する

Jupyterのセル内から直接以下を実行します。
!apt-get update
!apt-get install gcc
!apt-get install -y tdsodbc unixodbc-dev
!apt install unixodbc-bin -y
!apt-get clean -y

上記は、次の手順でPyODBCドライバーをリコンパイルするために必要なgcc（g++を含む）、FreeTDS、unixODBC、およびunixodbc-devをインストールします。

この手順はネイティブWindowsサーバーまたはPCでのPyODBCインストールには必要ありません。

4. Jupyter内からPyODBCインストールを実行する

!pip install pyodbc

```
Collecting pyodbc
  Downloading pyodbc-4.0.30.tar.gz (266 kB)
     |????????????????????????????????| 266 kB 11.3 MB/s eta 0:00:01
Building wheels for collected packages: pyodbc
  Building wheel for pyodbc (setup.py) ... done
  Created wheel for pyodbc: filename=pyodbc-4.0.30-cp36-cp36m-linux_x86_64.whl size=2
73453 sha256=b794c35f41e440441f2e79a95fead36d3aebfa74c0832a92647bb90c934688b3
  Stored in directory: /root/.cache/pip/wheels/e3/3f/16/e11367542166d4f8a252c031ac3a4
163d3b901b251ec71e905
Successfully built pyodbc
Installing collected packages: pyodbc
Successfully installed pyodbc-4.0.30
```

上記は、このDockerデモ用に最小化されたpip installです。公式ドキュメントには、「MacOS Xインストール」用のより詳細なpip installが提供されています。

5. LinuxでODBC INIファイルとリンクを再構成する

以下を実行して、**odbcinst.ini**と**odbc.ini**リンクを再作成します。
!rm /etc/odbcinst.ini
!rm /etc/odbc.ini
!ln -s /tf/odbcinst.ini /etc/odbcinst.ini
!ln -s /tf/odbc.ini /etc/odbc.ini

注意:
上記を行う理由は、
**手順3と4では
通常2つの空の（したがって
無効な）ODBCファイルが\etc\ディレクトリに作成され
る**
ためです。Windowsインストールとは異なりこれらの空のiniファイルは問題を生じるため、まずそれらを削除してから、マッピングされたDockerボリュームに提供されている実際のiniファイル（/tf/odbcinst.ini, and /tf/odbc.ini）へのリンクを再作成してください。

これらの2つのiniファイルをチェックしましょう。この場合、Linux ODBC構成の最も単純な形式です。
```
!cat /tf/odbcinst.ini
```

```
[InterSystems ODBC35]
UsageCount=1
Driver=/tf/libirisodbcu35.so
Setup=/tf/libirisodbcu35.so
SQLLevel=1
FileUsage=0
DriverODBCVer=02.10
ConnectFunctions=YYN
APILevel=1
DEBUG=1
CPTimeout=&lt;not pooled>
```

```
!cat /tf/odbc.ini
```

```
[IRIS PyODBC Demo]
Driver=InterSystems ODBC35
Protocol=TCP
Host=irisimlsvr
Port=51773
Namespace=USER
UID=SUPERUSER
Password=SYS
Description=Sample namespace
Query Timeout=0
Static Cursors=0
```

上記のファイルは事前構成済みであり、マッピングされたドライブで提供されています。
IRISサーバーのコンテナーインスタ
ンスからも取得できるドライバーファイル**libirisodbcu35.so**を参照しています。

したがって、上記のODBCインストールを機能させる
には、
**これらの3つのファイ
ルがマッピングされたドライブ（または任
意のLinuxドライブ）に存在**し、**適切なファイルアクセス権が適用されている**ことが必要です。

libirisodbcu35.so
odbcinst.ini
odbc.ini

6. PyODBCのインストールを検証する
```
!odbcinst -j
```

```
unixODBC 2.3.4
DRIVERS............: /etc/odbcinst.ini
SYSTEM DATA SOURCES: /etc/odbc.ini
FILE DATA SOURCES..: /etc/ODBCDataSources
USER DATA SOURCES..: /root/.odbc.ini
SQLULEN Size.......: 8
SQLLEN Size........: 8
SQLSETPOSIROW Size.: 8
```

```
 import pyodbc
print(pyodbc.drivers())
```

```
['InterSystems ODBC35']
```

上記の出力では、現在ODBCドライバーに有効なリンクがあることが示されています。

Jupyter NotebookでPython ODBCテストを実行できるはずです。

7. IRISサンプルへのPython ODBC接続を実行する

```
 import pyodbc
import time
 ### 1. Get an ODBC connection
#input("Hit any key to start")
dsn = 'IRIS PyODBC Demo'
server = 'irisimlsvr' # IRIS server container or the docker machine's IP
port = '51773' # or 8091 if docker machine IP is used
database = 'USER'
username = 'SUPERUSER'
password = 'SYS'
 #cnxn = pyodbc.connect('DSN='+dsn+';') # use the user DSN defined in odbc.ini, or use the connection string
below
cnxn = pyodbc.connect('DRIVER={InterSystems
ODBC35};SERVER='+server+';PORT='+port+';DATABASE='+database+';UID='+username+';PWD='+ password)
 ###ensure it reads strings correctly.
cnxn.setdecoding(pyodbc.SQL_CHAR, encoding='utf8')
cnxn.setdecoding(pyodbc.SQL_WCHAR, encoding='utf8')
cnxn.setencoding(encoding='utf8')
 ### 2. Get a cursor; start the timer
cursor = cnxn.cursor()
start= time.clock()
 ### 3. specify the training data, and give a model name
dataTable = 'DataMining.IrisDataset'
dataTablePredict = 'Result12'
dataColumn = 'Species'
dataColumnPredict = "PredictedSpecies"
modelName = "Flower12" #chose a name - must be unique in server end
 ### 4. Train and predict
#cursor.execute("CREATE MODEL %s PREDICTING (%s) FROM %s" % (modelName, dataColumn, dataTable))
#cursor.execute("TRAIN MODEL %s FROM %s" % (modelName, dataTable))
#cursor.execute("Create Table %s (%s VARCHAR(100), %s VARCHAR(100))" % (dataTablePredict,
dataColumnPredict, dataColumn))
#cursor.execute("INSERT INTO %s SELECT TOP 20 PREDICT(%s) AS %s, %s FROM %s" % (dataTablePredict,
modelName, dataColumnPredict, dataColumn, dataTable))
#cnxn.commit()
 ### 5. show the predict result
cursor.execute("SELECT * from %s ORDER BY ID" % dataTable) #or use dataTablePredict result by IntegratedML
```

```
if you run step 4 above
row = cursor.fetchone()
while row:
  print(row)
  row = cursor.fetchone()
  ### 6. CLose and clean
cnxn.close()
end= time.clock()
print ("Total elapsed time: ")
print (end-start)
```

```
(1, 1.4, 0.2, 5.1, 3.5, 'Iris-setosa')
(2, 1.4, 0.2, 4.9, 3.0, 'Iris-setosa')
(3, 1.3, 0.2, 4.7, 3.2, 'Iris-setosa')
(4, 1.5, 0.2, 4.6, 3.1, 'Iris-setosa')
(5, 1.4, 0.2, 5.0, 3.6, 'Iris-setosa')
... ...
... ...
... ...
(146, 5.2, 2.3, 6.7, 3.0, 'Iris-virginica')
(147, 5.0, 1.9, 6.3, 2.5, 'Iris-virginica')
(148, 5.2, 2.0, 6.5, 3.0, 'Iris-virginica')
(149, 5.4, 2.3, 6.2, 3.4, 'Iris-virginica')
(150, 5.1, 1.8, 5.9, 3.0, 'Iris-virginica')
Total elapsed time:
0.023873000000000033
```

ここにはいくつかの落とし穴があります。

1. **cnxn = pyodbc.connect() ** -
   Linux環境では、この呼び出しに渡される接続文字列は、スペースなしで文字通り正しい必要があります。
2. 接続エンコーディングをたとえばuft8などで適切に設定してください。
   この場合は、文字列のデフォルトエンコーディングは機能しません。
3. **libirisodbcu35.so** - 理想的には、このドライバーファイルはリモートIRISサーバーのバージョンと緊密に連携する必要があります。

今後の内容

これで、リモートIRISサーバーへのPyODBC（およびJDBC）接続による、Python 3とTensorFlow 2.2（GPUなし）を含むJupyterノートブックのDocker環境を得られました。 IRIS IntegratedML固有のSQL構文など、特別に設計されたすべてのSQL構文で機能するはずです。そこで、IntegratedMLの機能をもう少し探り、MLライフサイクルを駆動するSQL手法を工夫してみてはどうでしょうか。

また、次回は、Python環境でIRISネイティブSQLまたはマジックSQLを使用してIRISサーバーに接続する上で、最も単純なアプローチに触れられればと思います。 また、今では優れたPython Gateway を使用できるため、外部のPython MLアプリケーションとサービスをIRISサーバー内から直接呼び出してみることも可能です。これについてもさらに詳しく試せればいいなと思っています。

付録

上記のノートブックファイルは、こちらのGitHubリポジトリとOpen Exchangeにチェックインされます。

#AI #分析 #機械学習 #InterSystems IRIS

BC%E3%82%B9%E3%81%B8%E3%81%AEpython-odbc%E6%8E%A5%E7%B6%9A-2%E3%81%A4%E7%9B%AE%E3%81%AE%E7%B0%A1%E6%98%93%E3%83%A1%E3%83%A2