文章
Hao Ma · 三月 25, 2021 阅读大约需 8 分钟

将 Python ODBC 连接到 IRIS 数据库 - 第 2 条快速笔记

关键字：PyODBC，unixODBC，IRIS，IntegratedML，Jupyter Notebook，Python 3

目的

几个月前，我简单谈到了关于 "将 Python JDBC 连接到 IRIS " 的话题。我后来频繁提起它， 因此决定再写一篇 5 分钟的笔记，说明如何 " 将 Python ODBC 连接到 IRIS " 。

在 Windows 客户端中通常很容易设置 ODBC 和 PyODBC，不过我每次在 Linux/Unix 风格的服务器中设置 unixODBC 和 PyODBC 客户端时，都会遇到一些麻烦。

有没有一种简单连贯的方法，可以不安装任何 IRIS，在原版 Linux 客户端中让 PyODBC/unixODBC 针对远程 IRIS 服务器运行？

范围

最近，我花了点时间研究如何在 Linux Docker 环境的 Jupyter Notebook 中从头开始让一个 PyODBC 演示运行起来， 记录下这篇稍微有些繁琐的笔记，以供日后快速参考。

范围内：

这篇笔记将涉及以下组件：

PyODBC over **unixODBC**
安装了 TensorFlow 2.2 和 Python 3 的 **Jupyter Notebook** 服务器
带有 IntegratedML 的 **IRIS2020.3** CE 服务器，包括示例测试数据。
在此环境中

安装了 **Docker-compose** over AWS **Ubuntu** 16.04 的 Docker Engine
Docker Desktop for **MacOS** 和 **Docker Toolbox for Windows** 10 也经过了测试

范围外：

同样，在此演示环境中不评估非功能性方面。 它们很重要，并且可以针对特定站点，如：

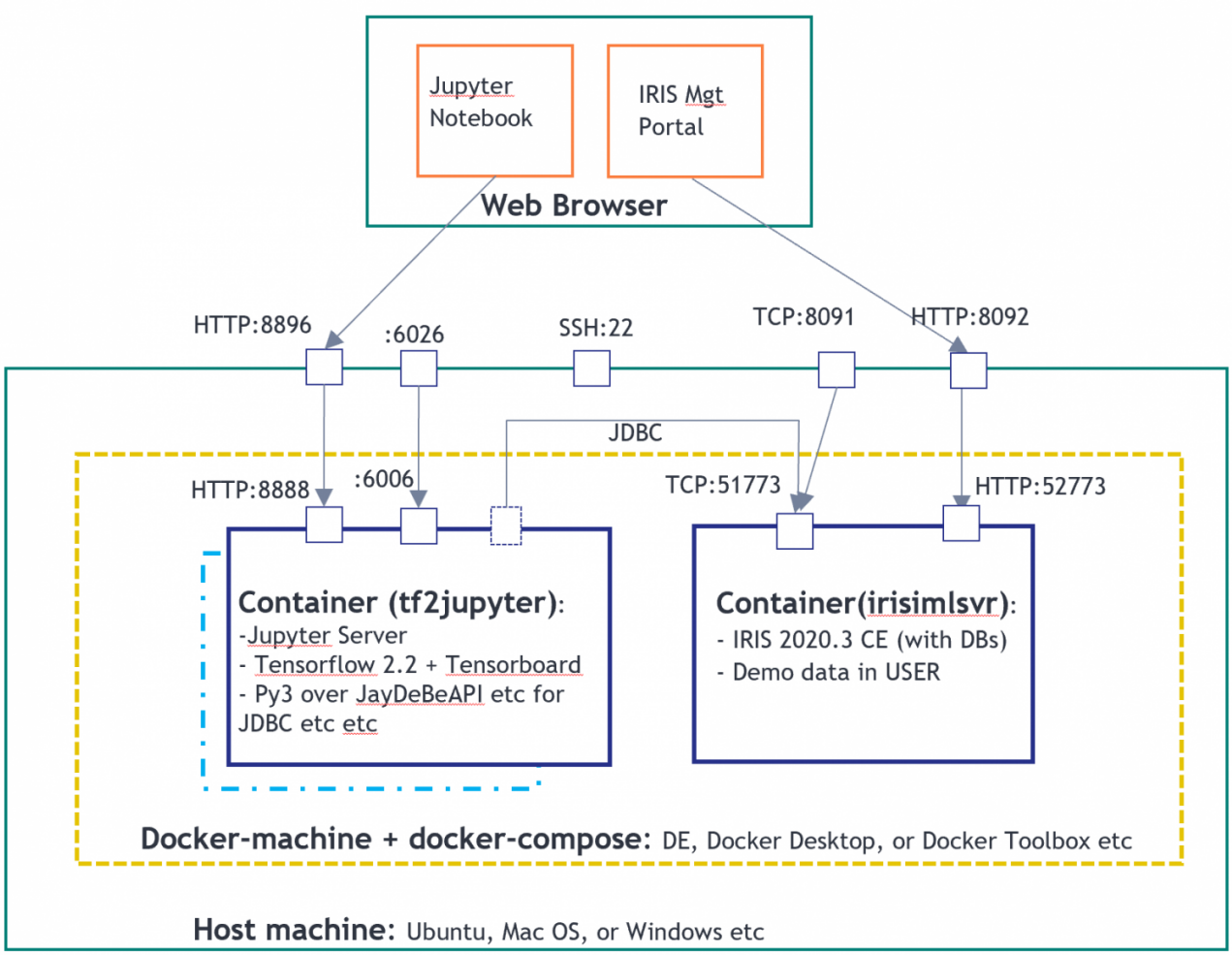
端到端安全和审核
性能和可扩展性
许可和可支持性等

环境

任何原版 Linux Docker 镜像都可以用于以下配置和测试步骤，但有一个简单的方法可以在 5 分钟内设置这样的环境：

1. Git **克隆**此演示模板
2. 在包含 docker-compose.yml 文件的克隆目录中运行 "**docker-compose up -d** " 。

这将创建一个演示环境，如下面的拓扑所示，其中包含 2 个容器。 一个用于 Jupyter Notebook 服务器作为 PyODBC 客户端，另一个用于 IRIS2020.3 CE 服务器。



在上面的环境中，tf2jupyter 仅包含 " Python over JDBC " 客户端配置；它尚不包含任何 ODBC 或 PyODBC 客户端配置。

因此，我们将直接在 Jupyter Notebook 内部运行以下设置步骤，以使其易于说明。

步骤

以下配置和测试由我在 AWS Ubuntu 16.04 服务器中运行， 由我的同事 @Thomas Dyar 在 MacOS 中运行。 另外在 Docker Toolbox for Windows 中也进行了简单的测试。 不过，如果您遇到任何问题，还是请告诉我们。

以下步骤可以自动化到其 Dockerfile。 我在这里特别记录一下，以防几个月后忘记。

1. 官方文档：

IRIS 的 ODBC 支持
在 Unix 上定义 ODBC 数据源

IRIS 的 PyODBC 支持

2. 连接到 Jupyter 服务器

我用本地 Putty 的 SSH 隧道连接到远程 AWS Ubuntu 端口 22，然后按照上述拓扑结构映射到端口 8896。

（举个例子，在本地 Docker 环境中，也可以直接直接 http 到 Docker 机器的 IP:8896。）

3. 从 Jupyter Notebook 中运行 ODBC 安装

直接在 Jupyter 单元格中运行以下代码：
!apt-get update
!apt-get install gcc
!apt-get install -y tdsodbc unixodbc-dev
!apt install unixodbc-bin -y
!apt-get clean -y

它将安装 gcc（包括 g++）编译器、FreeTDS、unixODBC 和 unixodbc-dev，以在下一步重新编译 PyODBC 驱动程序。

在原生 Windows 服务器或 PC 上安装 PyODBC 不需要这一步。

4. 从 Jupyter 中运行 PyODBC 安装

!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
```

以上是这个 Docker 演示的最简化 pip 安装。 在官方文档中，为 “ MacOS X 安装 ” 提供了更详细的 pip 安装。

5 在 Linux 中重新配置 ODBC INI 文件和链接：

运行以下命令重新创建 **odbcinst.ini** 和 **odbc.ini** 链接
!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：这样的原因是，**第 3 步和第 4 步通常会在 \etc\directory 下创建 2 个空白（因此无效）的 ODBC 文件。**与 Windows 安装不同，这里的空白 ini 文件会导致问题。因此需要先将其删除，然后重新创建一个链接来指向映射的 Docker 卷中提供的真实 ini 文件：/tf/odbcinst.ini 和 /tf/odbc.ini

看一看这两个 ini 文件。在这种情况下，它们是 Linux ODBC 配置的最简形式：
!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
```

以上文件都已预先配置，位于映射的驱动器中。 引用的是驱动程序文件 **libirisodbcu35.so**，可以从 IRIS 服务器的容器实例中获取该文件（在其 {iris-installation}/bin 目录下）。

要使上述 ODBC 安装正常运行，**这 3 个文件**必须存在于**具有正确文件权限**的映射驱动器（或任何 Linux 驱动器）中：

libirisodbcu35.so
odbcinst.ini
odbc.ini

6. 验证 PyODBC 安装

```
!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']
```

以上输出将表明 ODBC 驱动程序目前具有有效链接。

我们应该能够在 Jupyter Notebook 中运行一些 Python ODBC 测试

7. 运行将 Python ODBC 连接到 IRIS 的示例：

```
 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
```

这里有一些陷阱：

1. **cnxn = pyodbc.connect()** - 在 Linux 环境下，此调用中传递的连接字符串必须正确无误，不能有任何空格。
2. 正确设置连接编码，例如使用 utf8。 在这里默认值对字符串不起作用。
3. **libirisodbcu35.so** - 理想情况下，此驱动程序文件应与远程 IRIS 服务器的版本保持一致。

未来计划

这样就得到一个带有 Jupyter Notebook 的 Docker 环境，包括 Python3 和 TensorFlow 2.2（无 GPU），通过 PyODBC（以及 JDBC）连接到远程 IRIS 服务器。 所有定制的 SQL 语法应该都可以适用，比如 IRIS Integrated ML 专有的 SQL 语法。那么何不多研究一下 IntegratedML 的功能，用它驱动 ML 生命周期的 SQL 方法以进行一些创新？

另外，我希望接下来能介绍或总结出在 IRIS Native 甚至是 Python 环境中的魔法 SQL 上最简单的 IRIS 服务器挂接方法。 而且，现在有出色的 Python Gateway，我们甚至可以直接从 IRIS 服务器内部调用外部 Python ML 应用和服务。我希望我们也能在这方面多做些尝试。

附录

上面的笔记本文件也将被迁入此 Github 存储库以及 Open Exchange 中。

#AI #分析 #机器学习 #InterSystems IRIS

源 URL:https://cn.community.intersystems.com/post/%E5%B0%86-python-odbc-%E8%BF%9E%E6%8E%A5%E5%88%B0-iris-%E6%95%B0%E6%8D%AE%E5%BA%93-%E7%AC%AC-2-%E6%9D%A1%E5%BF%AB%E9%80%9F%E7%AC%94%E8%AE%B0