Train AI model in IBM Cloud
&
Deploy on mainframe

Table of Contents

1. Introduction

Across the globe companies are increasingly relying on cloud computing to operate their business. According to Gartner, cloud will be the centerpiece of new digital experiences, and companies will deploy 95% of new digital workloads on cloud-native platforms by 2025.

Cloud offers a great deal of flexibility and scalability and allows one to access data from anywhere. Meanwhile, mainframe computers still plays a central role in the daily operations of most of the world's largest corporations, including 92 of the world's top 100 worldwide banks, 10 out of 10 of the world's largest insurers, 23 of the world's largest airlines & 23 out of 25 of the US's largest retailers (IBM Z®). Mainframes process around 29 billion annual ATM transactions and 12.6 billion transactions per day (The modern mainframe).

Companies are working to have intelligent built into their transaction processing logic, which requires real-time AI in enterprise workloads without impacting SLAs. Meanwhile, batch workloads also face challenges with increasing transaction volume and shrinking processing time window. IBM Z Integrated Accelerator for AI embedded in the IBM Telum™ chip within IBM z16™ can help clients achieve large-scale low-latency model inference for both real-time and batch workloads. Combining the power of cloud and mainframe can bring clients the benefits of both worlds.

2. Use case

To unleash the power of data to better support existing applications and to explore new business initiatives, transactional data on mainframe needs to be made available. Transactional data is highly sensitive and needs to be protected. Security incidents are extremely expensive. Data needs to be secured and governed, and yet can be easily discovered and used in hybrid cloud environment.

Take credit card fraud for example, it is very important to flag and stop the fraudulent transaction while it happens, rather than starting a fraud investigation afterwards. However, it is impossible for financial firms to score all transactions in real-time by sending inference requests to a scoring engine over the network. By collocating the model inference with the transaction processing application on mainframe and utilizing the AI accelerator on z16, large-scale real-time scoring for all the transactions is feasible.

Data scientists do not normally train and tune AI models on mainframe. It is more convenient and cost efficient for data scientists to utilize the rich set of tools and services and pay for only what they consume in cloud.

In this article, we will use credit card fraud detection as a sample use case and discuss how to set up a secure training environment and train a sample AI model in IBM Cloud®, deploy the trained model back to mainframe for real-time inference, and invoke model inference on mainframe.

3. Set up secure environment in IBM Cloud

Although cloud provides many benefits, security is the main concern when using cloud computing services. According to the data breach report 2023, average cost of a data breach for Financial industry is $5.90M. Setting up a secure environment in cloud is critical.

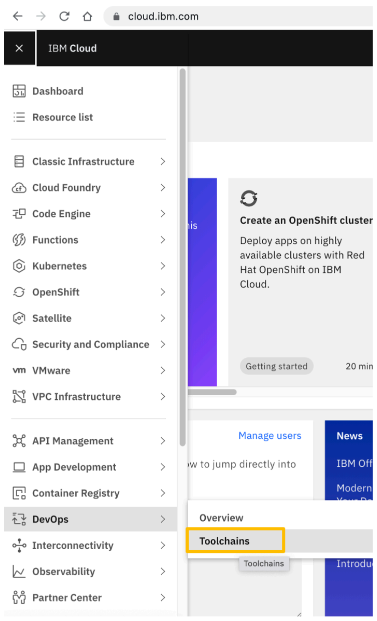
There are many different factors to consider when designing a secure environment in cloud, including network security, identity and access control, application security, and data security. It requires deep knowledge in each area to design the infrastructure and make sure that there are no security holes in the design.

IBM Cloud Framework for Financial Services is designed to build trust and enable a transparent public cloud ecosystem with security, compliance, and resiliency features that financial institutions require. The Framework utilizes services in IBM Cloud to create secure and compliant environment, (for example, IBM Hyper Protect Crypto Services (HPCS), IBM DevOps toolchain, IBM Security and Compliance Center (SCC)), includes reference architectures and best practices. The Security and Compliance Center has a set of profiles with pre-defined controls and polices that will run, monitor, and audit the services. The policies and controls are determined by IBM Cloud Regulatory Council of members who are from risk and compliance of leading financial industry companies and Promontory Financial Group, an IBM subsidiary, to ensure that it is current with new and updated regulations.

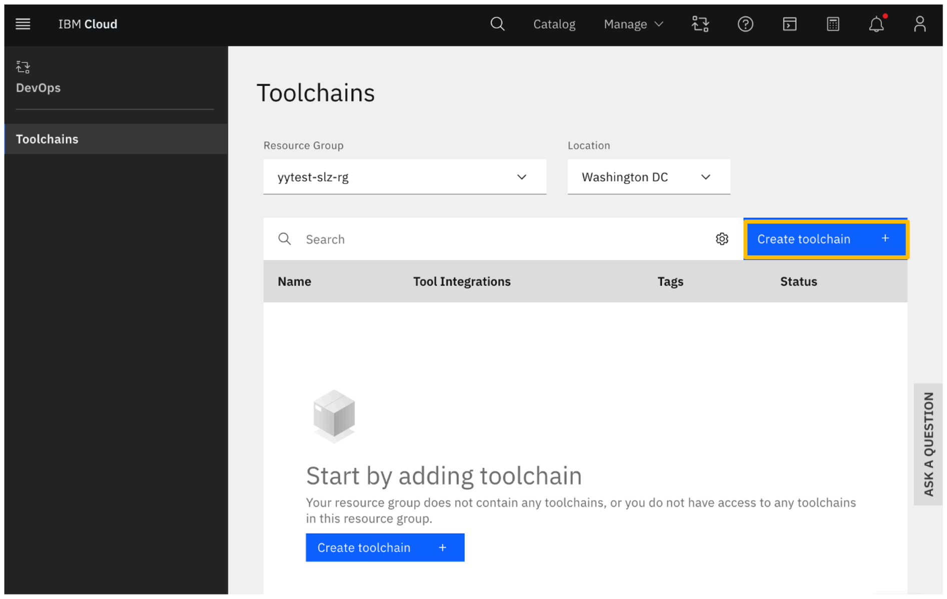
The IBM Cloud Framework for Financial Services defines a few reference architectures, which can be deployed via infrastructure as code DevOps toolchain and used as a basis for meeting the security and regulatory requirements. Sensitive workloads (eg. AI model training jobs) can then be deployed into the environment.

- VPC reference architecture for IBM Cloud for Financial Services
- Satellite reference architecture for IBM Cloud for Financial Services
- IBM Cloud for VMware Regulated Workloads architecture

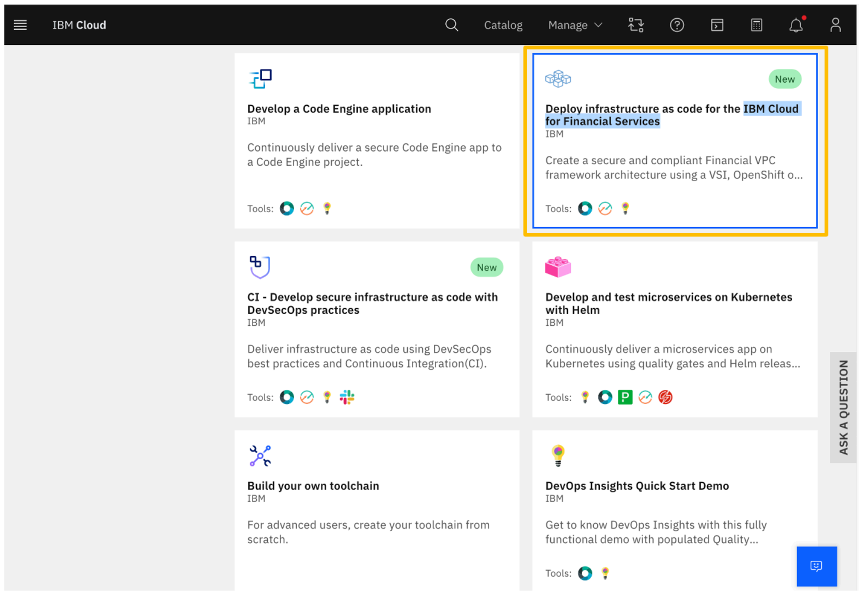
To create DevOps toolchain for IBM Cloud for Financial Services™, log in to IBM Cloud, click on the hamburger menu on the upper-left corner, and then click on DevOps -> Toolchains.



Click ‘Create toolchain’ button.



Look for the tile titled ‘Deploy Infrastructure as code for the IBM Cloud for Financial Services’, which will clone the [Secure Landing Zone toolchain repository]() with Terraform templates to set up the environment. You can update the config files based on your needs before provisioning the environment.



Develop a Code Engine application
IBM
Continuously deliver a secure Code Engine app to a Code Engine project.
Tools:

New
Deploy infrastructure as code for the IBM Cloud for Financial Services
IBM
Create a secure and compliant Financial VPC framework architecture using a VSI, OpenShift o...
Tools:

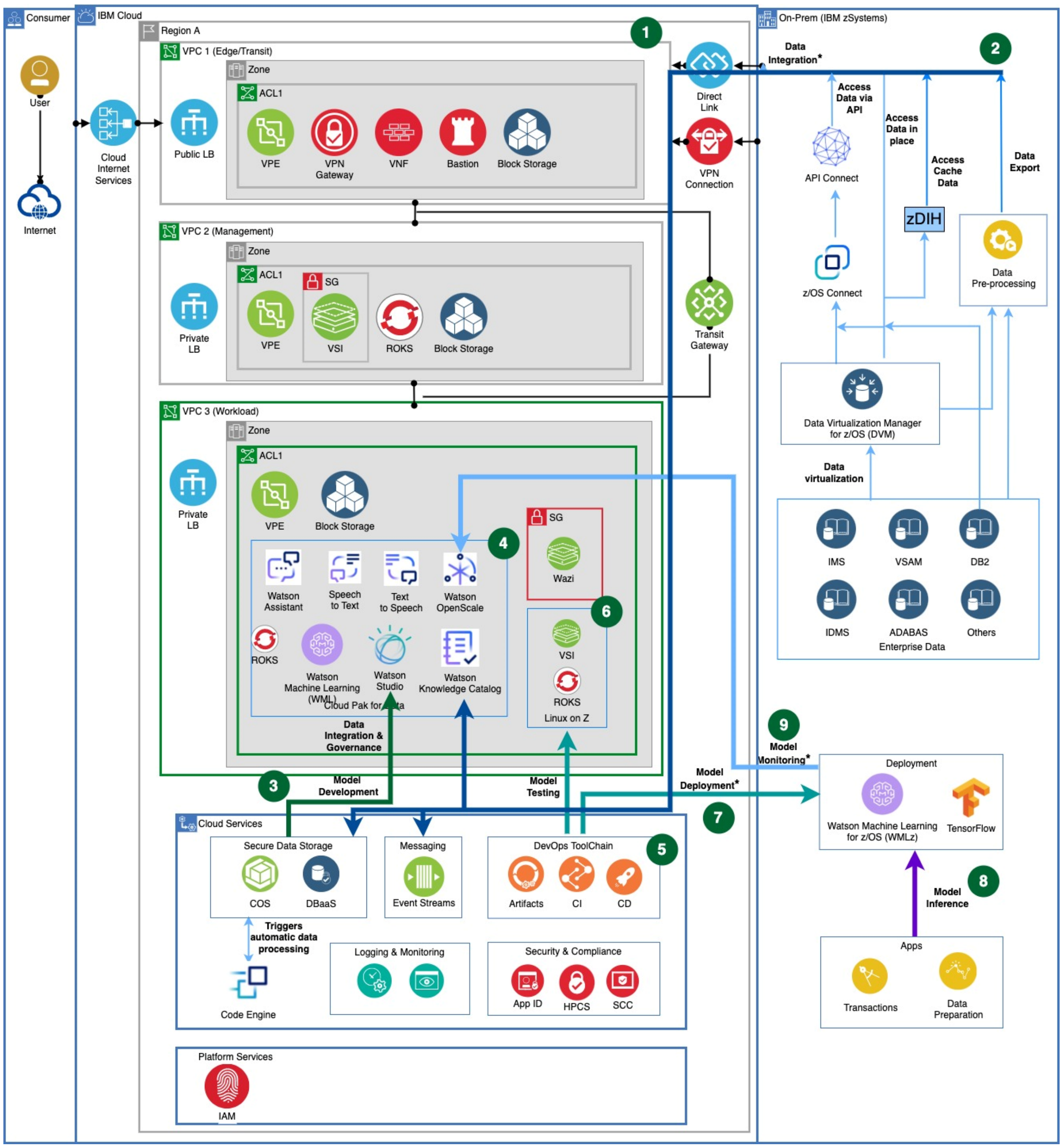
New
CI - Develop secure infrastructure as code with DevSecOps practices
IBM
Deliver infrastructure as code using DevSecOps best practices and Continuous Integration(CI).
Tools:

Develop and test microservices on Kubernetes with Helm
IBM
Continuously deliver a microservices app on Kubernetes using quality gates and Helm releas...
Tools:

Build your own toolchain
IBM
For advanced users, create your toolchain from scratch.

DevOps Insights Quick Start Demo
IBM
Get to know DevOps Insights with this fully functional demo with populated Quality...
Tools:

Here is a VPC based reference architecture for AI pipeline. This article focuses on the AI model building, deployment, and inference.



1. Set up the infrastructure by creating DevOps toolchain for "Deploy infrastructure as Code for the IBM Cloud for Financial Services", which provides a secure environment for model training jobs. For more information, refer to reference architecture for IBM Cloud for Financial Services.
2. Make the enterprise data on-prem accessible in refer to data integration blog cloud. There are different ways to expose the data on-prem.
3. Understand, prepare, and manage the governance of the data with tools in IBM Cloud Pak® for Data, for example, organizing and managing data with Watson Knowledge Catalog.

4. Training jobs can be run in the workload VPC. Various tools in IBM Cloud Pak for Data can be used. IBM Watson® Studio can be used as the model development environment. Different machine learning platforms and tools can be used to train the model (example, TensorFlow, pyTorch, SnapML, etc.). Model can be tested in Machine Learning. For conversational AI, tools like Watson Assistant, Speech-to-Text, and Text-to-Speech can be used.
5. IBM DevOps Toolchains can be used to manage the model source codes and tested models.
6. Linux® on IBM Z or IBM Wazi as a Service can be used to test the AI models and AI applications before pushing them to on-premises IBM Z.
7. Tested AI models are deployed on IBM Machine Learning for z/OS®.
8. AI model inference is invoked to gain insights from the data.
9. AI models are continuously monitored via IBM Watson OpenScale, which is an enterprise-grade environment for AI applications that provides enterprise visibility into how your AI is built and used, and delivers return on investment. Its open platform enables businesses to operate and automate AI at scale with transparent, explainable outcomes that are free from harmful bias and drift.

4. AI model building in IBM Cloud

AI model building is an iterative process. It normally starts with understanding the business problem on hand, and iterate through data understanding and preparation, model training and testing, model deployment, model inference, and model monitoring. We will focus on model building in IBM Cloud in this section.

In the last section, we set up a secure environment in cloud, now we can deploy our favorite training framework into the environment. This demo uses Cloud Pak for Data (CP4D) as the training environment and CP4D is set up in the workload VPC. Refer to the CP4D documentation if you need to install CP4D.

In this demo, we will build a sample model that analyzes historic credit card transaction data to predict whether a new credit card transaction is fraudulent. The source code is in ai-on-z-fraud-detection github repository, and we will use one of the sample models to demonstrate the model development and deployment process.

4.1 AI model building steps

You can choose your favorite framework to build the AI model. In this demo, TensorFlow framework is used to build a deep learning model, and Watson Studio in CP4D is used as the development environment.

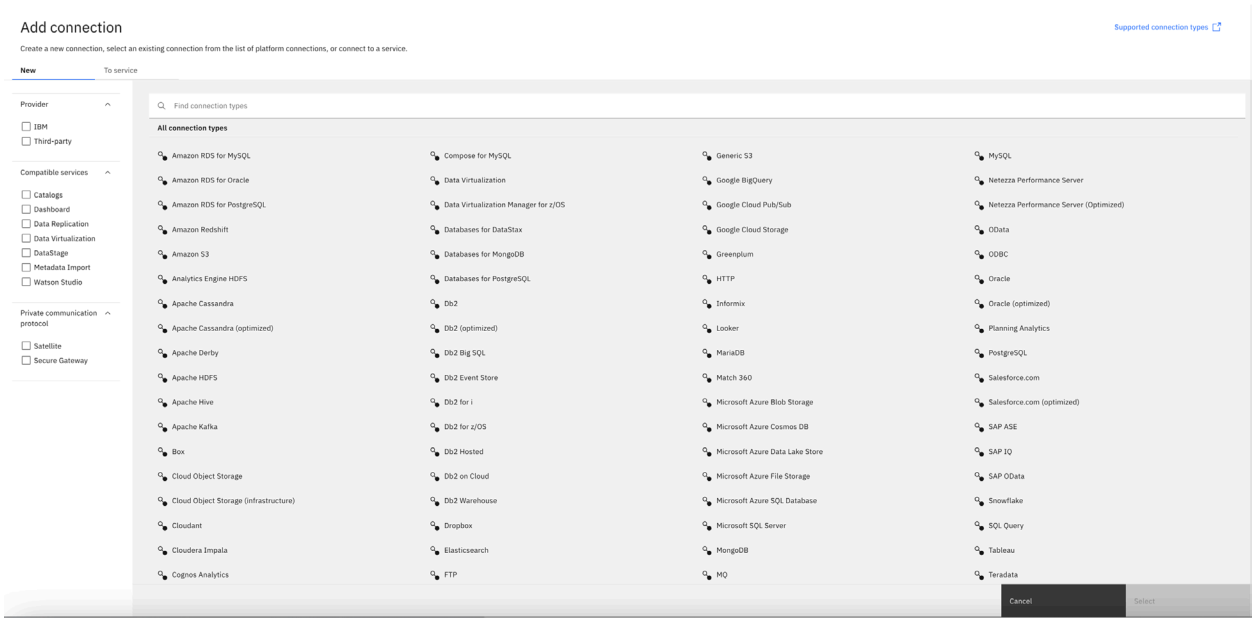
The following steps are needed to build and persist the model:

- Project setup – Watson Studio project is the logical organization of resources and can be used to manage access to the resources.
- Data connection – Watson Studio supports data connectors to various data sources. In our case, the batch data from on-prem is uploaded to IBM Cloud Object Storage (COS), which is protected by IBM Cloud Hyper Protect Crypto Services (HPCS) keys. HPCS is an industry-leading key management and hardware security module service built with mainframe-level encryption and offered as a service in the cloud. It can be integrated with other IBM Cloud services to protect digital assets in Cloud. Data in motion, data at rest, and data in use are all fully protected in IBM Cloud for Financial Services environment.
- Model training – Data scientists can play with the data, use the framework of their choice, and build AI models. In this demo, TensorFlow framework is used to train the demo model.
- Model persistence – Trained model is persisted, so that it can be brought to the deployment environment for model inference. In this demo, the model is persisted in a couple of different ways, TensorFlow format and Open Neural Network Exchange (ONNX) format, and we will show how to deploy each of them on the mainframe.
- Model conversion – ONNX defines a common set of operators - the building blocks of machine learning and deep learning models - and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers. By converting the model into ONNX format, the model can be hosted in a framework that is independent of the model training framework, so that data scientists have more flexibility when they choose the framework they are most familiar with to train the model.

If you want to learn more about how to build a model in Watson Studio in CP4D environment, this tutorial has detailed instructions.

4.2 Data connection

IBM CP4D provides a suite of services to connect, catalog, discover, and govern the data. CP4D supports data connectors to various data sources, regardless of the physical location of the data. Watson Knowledge Catalog helps user to catalog and understand the data and control access of the data. Here is a sample screenshot of the various data sources that one can connect from Watson Studio. The list may change over time.



Add connection

Create a new connection, select an existing connection from the list of platform connections, or connect to a service.

Supported connection types

New | To service

Provider
- [ ] IBM
- [ ] Third-party

Compatible services
- [ ] Catalogs
- [ ] Dashboard
- [ ] Data Replication
- [ ] Data Virtualization
- [ ] DataStage
- [ ] Metadata Import
- [ ] Watson Studio

Private communication protocol
- [ ] Satellite
- [ ] Secure Gateway

Find connection types

All connection types

Amazon RDS for MySQL	Compose for MySQL	Generic S3	MySQL
Amazon RDS for Oracle	Data Virtualization	Google BigQuery	Netezza Performance Server
Amazon RDS for PostgreSQL	Data Virtualization Manager for z/OS	Google Cloud Pub/Sub	Netezza Performance Server (Optimized)
Amazon Redshift	Databases for DataStax	Google Cloud Storage	OData
Amazon S3	Databases for MongoDB	Greenplum	ODBC
Analytics Engine HDFS	Databases for PostgreSQL	HTTP	Oracle
Apache Cassandra	Db2	Informix	Oracle (optimized)
Apache Cassandra (optimized)	Db2 (optimized)	Looker	Planning Analytics
Apache Derby	Db2 Big SQL	MariaDB	PostgreSQL
Apache HDFS	Db2 Event Store	Match 360	Salesforce.com
Apache Hive	Db2 for i	Microsoft Azure Blob Storage	Salesforce.com (optimized)
Apache Kafka	Db2 for z/OS	Microsoft Azure Cosmos DB	SAP ASE
Box	Db2 Hosted	Microsoft Azure Data Lake Store	SAP IQ
Cloud Object Storage	Db2 on Cloud	Microsoft Azure File Storage	SAP OData
Cloud Object Storage (infrastructure)	Db2 Warehouse	Microsoft Azure SQL Database	Snowflake
Cloudant	Dropbox	Microsoft SQL Server	SQL Query
Cloudera Impala	Elasticsearch	MongoDB	Tableau
Cognos Analytics	FTP	MQ	Teradata

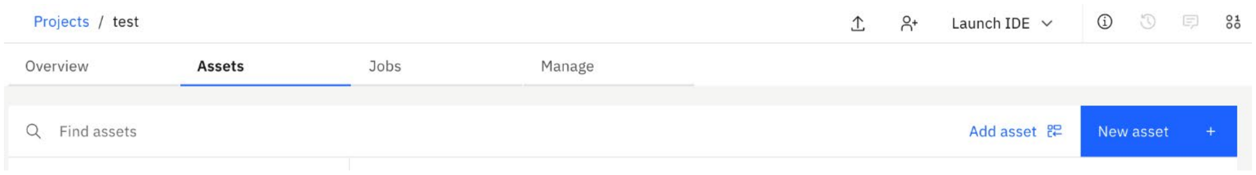
Cancel | Select

Data connection to different data sources

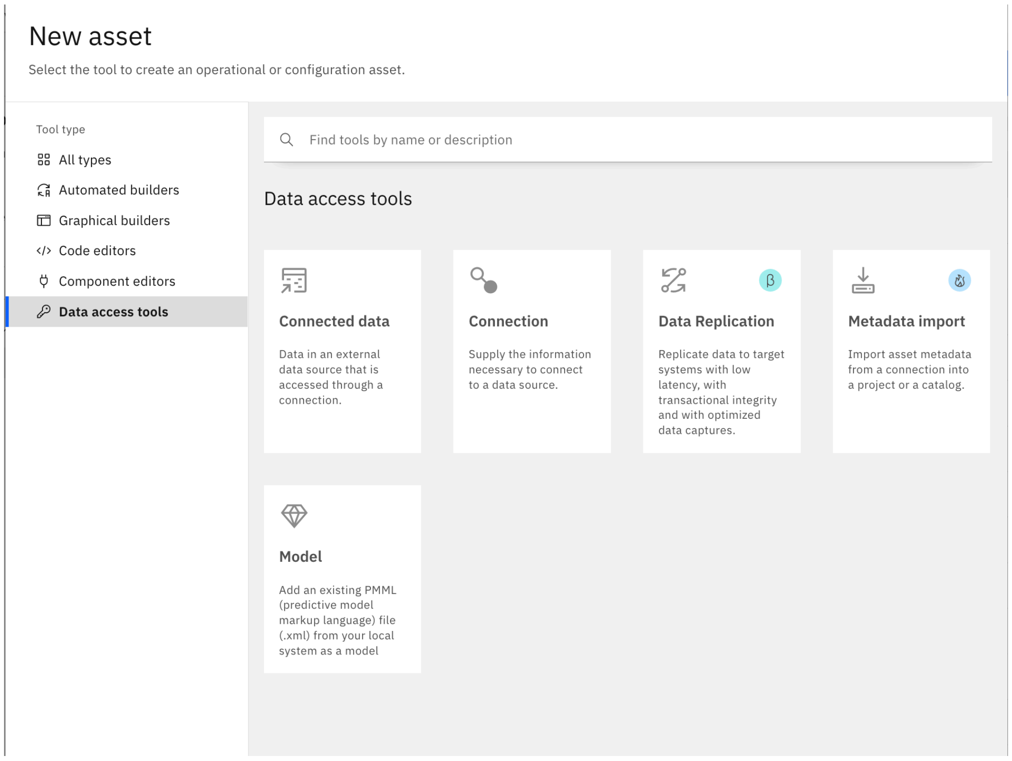
For this demo, the data is stored in Hyper Protect Crypto Service (HPCS) protected Cloud Object Storage (COS) bucket. IBM Cloud HPCS is a key management service and cloud Hardware Security Module (HSM) that is built on FIPS 140-2 Level 4-certified hardware which supports clients with Keep Your Own Key (KYOK). The synthesized data of this demo can be found in this github repository.

You can use direct link or create VPN to connect on-prem to IBM Cloud. You can also config IBM Cloud Functions or IBM Code Engine to listen on COS write events and process the data automatically when it arrives.

Once you have the connection created, Watson Studio can generate code to access data from the given connection. For example, you can first create connection to the data source, in this case COS bucket. Under 'Assets', click 'New asset'.



In the New asset window, under 'Data access tools' on the left, click on 'Connection' tile.


New asset

Select the tool to create an operational or configuration asset.

Tool type
- All types
- Automated builders
- Graphical builders
- Code editors
- Component editors
- Data access tools

Find tools by name or description

Data access tools

Connected data — Data in an external data source that is accessed through a connection.

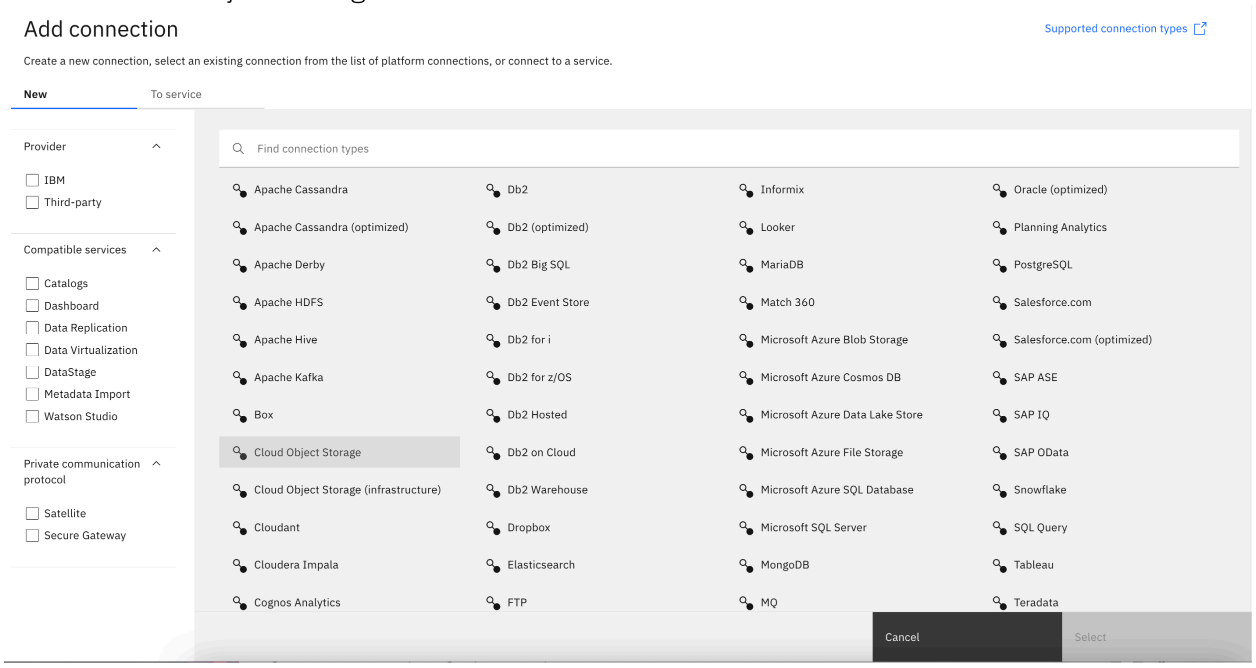
Connection — Supply the information necessary to connect to a data source.

Data Replication — Replicate data to target systems with low latency, with transactional integrity and with optimized data captures.

Metadata import — Import asset metadata from a connection into a project or a catalog.

Model — Add an existing PMML (predictive model markup language) file (.xml) from your local system as a model

Choose Cloud Object Storage and fill in the connection information to the COS instance.


Add connection

Supported connection types

Create a new connection, select an existing connection from the list of platform connections, or connect to a service.

New | To service

Provider
- IBM
- Third-party

Compatible services
- Catalogs
- Dashboard
- Data Replication
- Data Virtualization
- DataStage
- Metadata Import
- Watson Studio

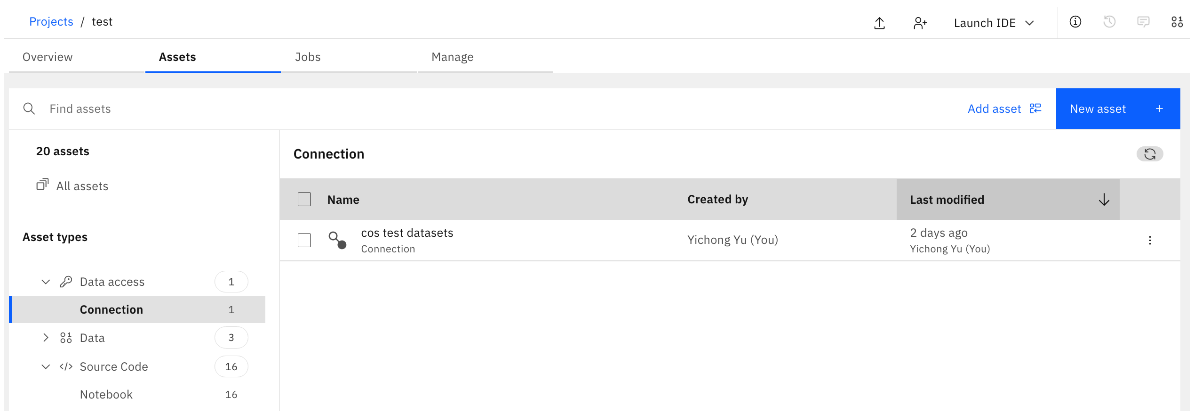
Private communication protocol
- Satellite
- Secure Gateway

Find connection types

Apache Cassandra	Db2	Informix	Oracle (optimized)
Apache Cassandra (optimized)	Db2 (optimized)	Looker	Planning Analytics
Apache Derby	Db2 Big SQL	MariaDB	PostgreSQL
Apache HDFS	Db2 Event Store	Match 360	Salesforce.com
Apache Hive	Db2 for i	Microsoft Azure Blob Storage	Salesforce.com (optimized)
Apache Kafka	Db2 for z/OS	Microsoft Azure Cosmos DB	SAP ASE
Box	Db2 Hosted	Microsoft Azure Data Lake Store	SAP IQ
Cloud Object Storage	Db2 on Cloud	Microsoft Azure File Storage	SAP OData
Cloud Object Storage (infrastructure)	Db2 Warehouse	Microsoft Azure SQL Database	Snowflake
Cloudant	Dropbox	Microsoft SQL Server	SQL Query
Cloudera Impala	Elasticsearch	MongoDB	Tableau
Cognos Analytics	FTP	MQ	Teradata

Cancel | Select

The new connection should be under the Connection section.


Projects / test

Launch IDE

Overview | Assets | Jobs | Manage

Find assets | Add asset | New asset +

20 assets

All assets

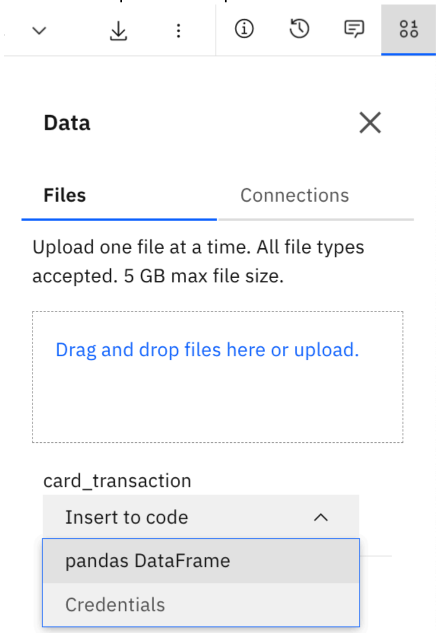
Asset types

- Data access 1
  - Connection 1
- Data 3
- Source Code 16
  - Notebook 16

Connection

Name	Created by	Last modified
cos test datasets Connection	Yichong Yu (You)	2 days ago Yichong Yu (You)

Now in notebook, Watson Studio can help insert pandas Dataframe to the code.


Data
Files
Connections
Upload one file at a time. All file types accepted. 5 GB max file size.
Drag and drop files here or upload.
card_transaction
Insert to code
pandas DataFrame
Credentials

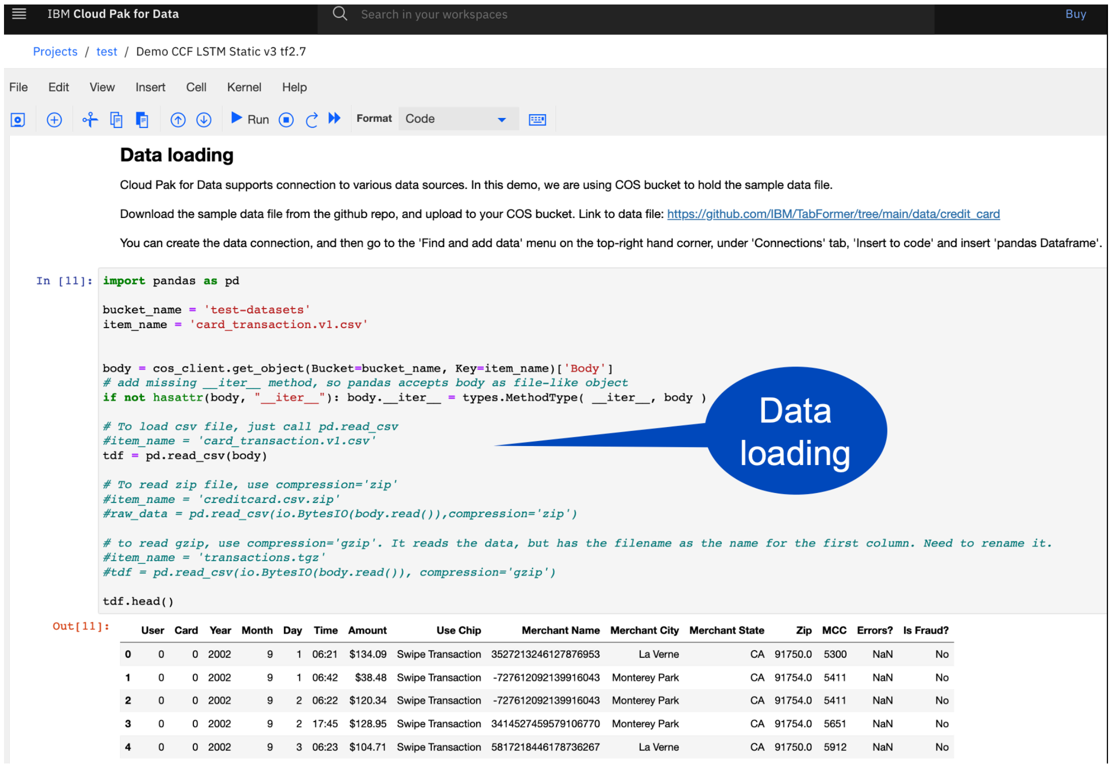
Insert code to access data

If you want to write your own codes to connect to COS buckets via APIs, here are sample codes to do so. Please make sure to update the COS connection information and API keys.

```
# TODO: Replace with your API Key
COS_API_KEY = 'YOUR_API_KEY'
# TODO: Replace with your COS bucket endpoint
COS_ENDPOINT = 'https://YOUR_COS_ENDPOINT'
# TODO: Replace with your COS resource CRN
COS_RESOURCE_CRN = 'YOUR_COS_RESOURE_CRN'

# TODO: Replace with your COS location
location = 'YOUR_COS_LOCATION'

# IAM endpoint
IAM_AUTH_ENDPOINT = 'https://iam.cloud.ibm.com/oidc/token'
```

```python
import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share the notebook.
cos_client = ibm_boto3.client(service_name='s3',
    ibm_api_key_id=COS_API_KEY,
    ibm_auth_endpoint=IAM_AUTH_ENDPOINT,
    config=Config(signature_version='oauth'),
    endpoint_url=COS_ENDPOINT)
```

Then you can read data in.


IBM Cloud Pak for Data
Search in your workspaces
Buy
Projects / test / Demo CCF LSTM Static v3 tf2.7
File Edit View Insert Cell Kernel Help
Run
Format Code
Data loading

Data loading

Cloud Pak for Data supports connection to various data sources. In this demo, we are using COS bucket to hold the sample data file.

Download the sample data file from the github repo, and upload to your COS bucket. Link to data file: https://github.com/IBM/TabFormer/tree/main/data/credit_card

You can create the data connection, and then go to the 'Find and add data' menu on the top-right hand corner, under 'Connections' tab, 'Insert to code' and insert 'pandas Dataframe'.

```python
In [11]: import pandas as pd

bucket_name = 'test-datasets'
item_name = 'card_transaction.v1.csv'


body = cos_client.get_object(Bucket=bucket_name, Key=item_name)['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

# To load csv file, just call pd.read_csv
#item_name = 'card_transaction.v1.csv'
tdf = pd.read_csv(body)

# To read zip file, use compression='zip'
#item_name = 'creditcard.csv.zip'
#raw_data = pd.read_csv(io.BytesIO(body.read()),compression='zip')

# to read gzip, use compression='gzip'. It reads the data, but has the filename as the name for the first column. Need to rename it.
#item_name = 'transactions.tgz'
#tdf = pd.read_csv(io.BytesIO(body.read()), compression='gzip')

tdf.head()
```

Out[11]:

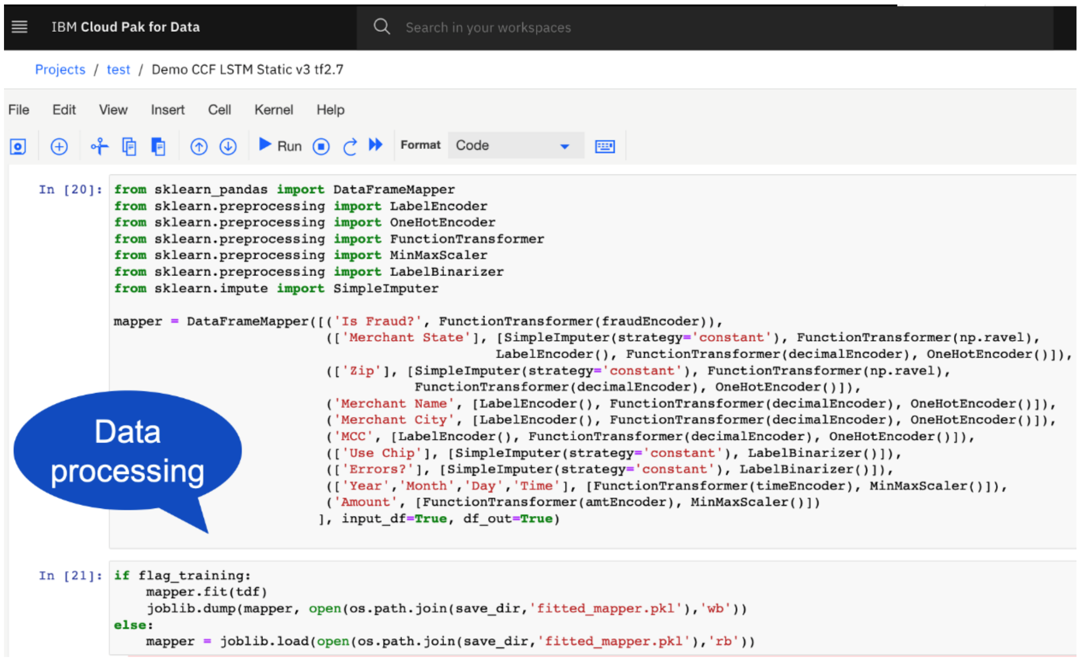
	User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zip	MCC	Errors?	Is Fraud?
0	0	0	2002	9	1	06:21	$134.09	Swipe Transaction	3527213246127876953	La Verne	CA	91750.0	5300	NaN	No
1	0	0	2002	9	1	06:42	$38.48	Swipe Transaction	-727612092139916043	Monterey Park	CA	91754.0	5411	NaN	No
2	0	0	2002	9	2	06:22	$120.34	Swipe Transaction	-727612092139916043	Monterey Park	CA	91754.0	5411	NaN	No
3	0	0	2002	9	2	17:45	$128.95	Swipe Transaction	3414527459579106770	Monterey Park	CA	91754.0	5651	NaN	No
4	0	0	2002	9	3	06:23	$104.71	Swipe Transaction	5817218446178736267	La Verne	CA	91750.0	5912	NaN	No

Here are sample helper functions to download file from and upload file to COS bucket.

```
In [7]: # define functions that will be used later to upload and download models
        def get_item(cos_resource, bucket_name, item_name, file_name):
            print("Retrieving item from bucket: {0}, key: {1}".format(bucket_name, item_name))
            try:
                item = cos_resource.Object(bucket_name, item_name)
                item.download_file(file_name)
            except ClientError as be:
                print("CLIENT ERROR: {0}\n".format(be))
            except Exception as e:
                print("Unable to retrieve file contents: {0}".format(e))


        def upload_large_file(cos_cli, bucket_name, item_name, file_path):
            print("Starting large file upload for {0} to bucket: {1}".format(item_name, bucket_name))

            # set the chunk size to 5 MB
            part_size = 1024 * 1024 * 5

            # set threadhold to 5 MB
            file_threshold = 1024 * 1024 * 5

            # set the transfer threshold and chunk size in config settings
            transfer_config = ibm_boto3.s3.transfer.TransferConfig(
                multipart_threshold=file_threshold,
                multipart_chunksize=part_size
            )

            # create transfer manager
            transfer_mgr = ibm_boto3.s3.transfer.TransferManager(cos_cli, config=transfer_config)

            try:
                # initiate file upload
                future = transfer_mgr.upload(file_path, bucket_name, item_name)

                # wait for upload to complete
                future.result()

                print ("Large file upload complete!")
            except Exception as e:
                print("Unable to complete large file upload: {0}".format(e))
            finally:
                transfer_mgr.shutdown()
```

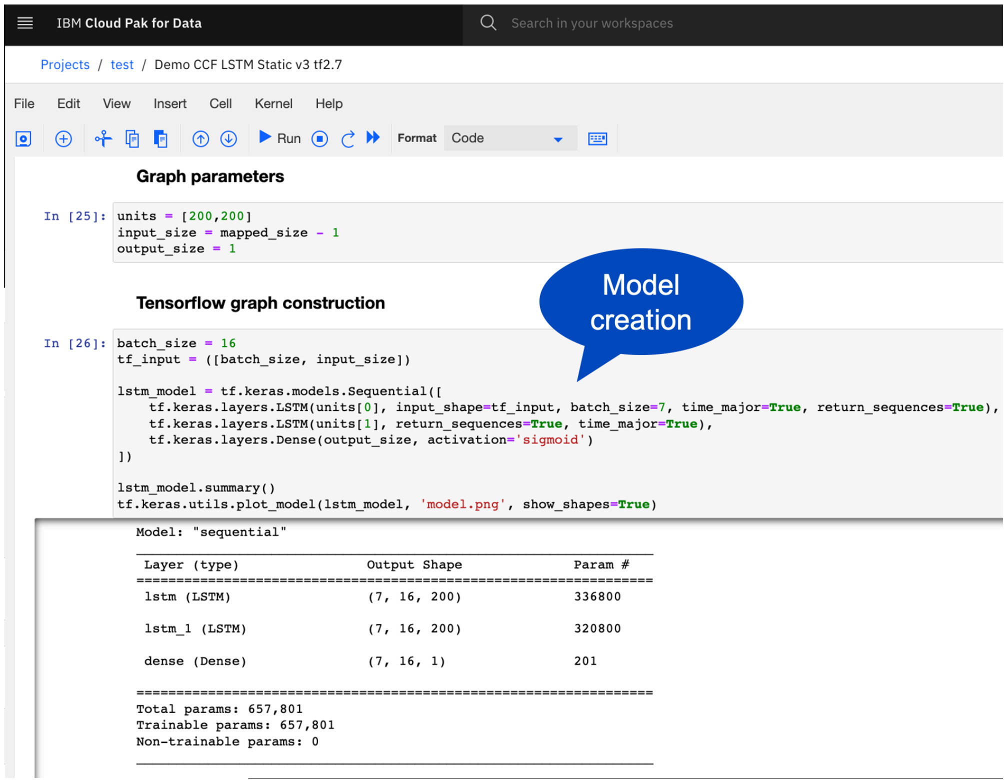
4.3 AI model building

When building AI models, it is critical to understand the data. You can inspect the data distribution and data quality (e.g. missing, duplicate data, etc.). You may need to map categorical data to numerical and normalize numerical data. Here are the data transformations on the demo data set.


IBM Cloud Pak for Data
Search in your workspaces
Projects / test / Demo CCF LSTM Static v3 tf2.7
File Edit View Insert Cell Kernel Help
Run Format Code
Data processing

```
In [20]: from sklearn_pandas import DataFrameMapper
         from sklearn.preprocessing import LabelEncoder
         from sklearn.preprocessing import OneHotEncoder
         from sklearn.preprocessing import FunctionTransformer
         from sklearn.preprocessing import MinMaxScaler
         from sklearn.preprocessing import LabelBinarizer
         from sklearn.impute import SimpleImputer

         mapper = DataFrameMapper([('Is Fraud?', FunctionTransformer(fraudEncoder)),
                                   (['Merchant State'], [SimpleImputer(strategy='constant'), FunctionTransformer(np.ravel),
                                                        LabelEncoder(), FunctionTransformer(decimalEncoder), OneHotEncoder()]),
                                   (['Zip'], [SimpleImputer(strategy='constant'), FunctionTransformer(np.ravel),
                                             FunctionTransformer(decimalEncoder), OneHotEncoder()]),
                                   ('Merchant Name', [LabelEncoder(), FunctionTransformer(decimalEncoder), OneHotEncoder()]),
                                   ('Merchant City', [LabelEncoder(), FunctionTransformer(decimalEncoder), OneHotEncoder()]),
                                   ('MCC', [LabelEncoder(), FunctionTransformer(decimalEncoder), OneHotEncoder()]),
                                   (['Use Chip'], [SimpleImputer(strategy='constant'), LabelBinarizer()]),
                                   (['Errors?'], [SimpleImputer(strategy='constant'), LabelBinarizer()]),
                                   (['Year','Month','Day','Time'], [FunctionTransformer(timeEncoder), MinMaxScaler()]),
                                   ('Amount', [FunctionTransformer(amtEncoder), MinMaxScaler()])
                                  ], input_df=True, df_out=True)

In [21]: if flag_training:
             mapper.fit(tdf)
             joblib.dump(mapper, open(os.path.join(save_dir,'fitted_mapper.pkl'),'wb'))
         else:
             mapper = joblib.load(open(os.path.join(save_dir,'fitted_mapper.pkl'),'rb'))
```

This demo builds a deep learning model, which has an input layer, an output layer, and a couple of hidden layers.


IBM Cloud Pak for Data
Search in your workspaces
Projects / test / Demo CCF LSTM Static v3 tf2.7
File Edit View Insert Cell Kernel Help
Run Format Code
Model creation

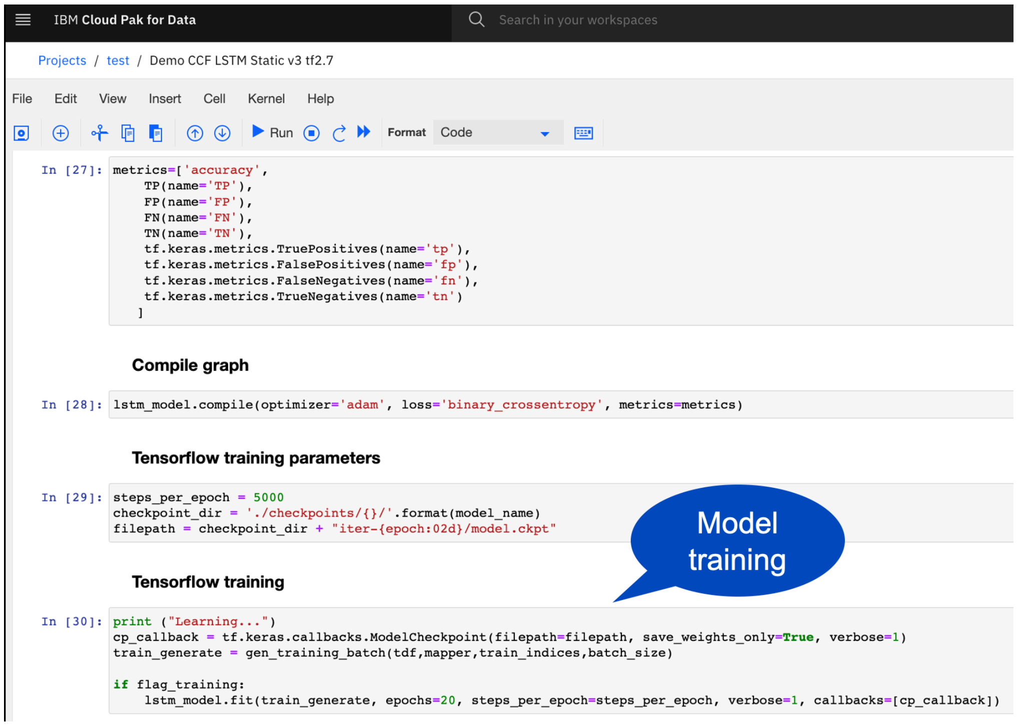
Graph parameters

```
In [25]: units = [200,200]
         input_size = mapped_size - 1
         output_size = 1
```

Tensorflow graph construction

```
In [26]: batch_size = 16
         tf_input = ([batch_size, input_size])

         lstm_model = tf.keras.models.Sequential([
             tf.keras.layers.LSTM(units[0], input_shape=tf_input, batch_size=7, time_major=True, return_sequences=True),
             tf.keras.layers.LSTM(units[1], return_sequences=True, time_major=True),
             tf.keras.layers.Dense(output_size, activation='sigmoid')
         ])

         lstm_model.summary()
         tf.keras.utils.plot_model(lstm_model, 'model.png', show_shapes=True)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (7, 16, 200)              336800

 lstm_1 (LSTM)               (7, 16, 200)              320800

 dense (Dense)               (7, 16, 1)                201

=================================================================
Total params: 657,801
Trainable params: 657,801
Non-trainable params: 0
_________________________________________________________________
```


IBM Cloud Pak for Data
Search in your workspaces
Projects / test / Demo CCF LSTM Static v3 tf2.7
File Edit View Insert Cell Kernel Help
Run Format Code
Model training

```
In [27]: metrics=['accuracy',
             TP(name='TP'),
             FP(name='FP'),
             FN(name='FN'),
             TN(name='TN'),
             tf.keras.metrics.TruePositives(name='tp'),
             tf.keras.metrics.FalsePositives(name='fp'),
             tf.keras.metrics.FalseNegatives(name='fn'),
             tf.keras.metrics.TrueNegatives(name='tn')
            ]
```

Compile graph

```
In [28]: lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=metrics)
```

Tensorflow training parameters

```
In [29]: steps_per_epoch = 5000
         checkpoint_dir = './checkpoints/{}/'.format(model_name)
         filepath = checkpoint_dir + "iter-{epoch:02d}/model.ckpt"
```

Tensorflow training

```
In [30]: print ("Learning...")
         cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=filepath, save_weights_only=True, verbose=1)
         train_generate = gen_training_batch(tdf,mapper,train_indices,batch_size)

         if flag_training:
             lstm_model.fit(train_generate, epochs=20, steps_per_epoch=steps_per_epoch, verbose=1, callbacks=[cp_callback])
```

4.4 AI model persistence

TensorFlow model can be saved via
    model.save(save_dir)
Refer to TensorFlow documentation for more details.

You can save the trained model to repository of your choice, and later picked up by the CI/CD pipeline. In this demo, the TensorFlow model is zipped and uploaded to COS bucket.

```
!zip -r '{model_name}.zip' ./saved_models
```

```
    adding: saved_models/ (stored 0%)
    adding: saved_models/ccf_220_keras_lstm_public/ (stored 0%)
    adding: saved_models/ccf_220_keras_lstm_public/1/ (stored 0%)
    adding: saved_models/ccf_220_keras_lstm_public/1/wts.index (deflated 66%)
    adding: saved_models/ccf_220_keras_lstm_public/1/saved_model.pb (deflated 91%)
    adding: saved_models/ccf_220_keras_lstm_public/1/keras_metadata.pb (deflated 89%)
    adding: saved_models/ccf_220_keras_lstm_public/1/wts.data-00000-of-00001 (deflated 7%)
    adding: saved_models/ccf_220_keras_lstm_public/1/checkpoint (deflated 35%)
    adding: saved_models/ccf_220_keras_lstm_public/1/variables/ (stored 0%)
    adding: saved_models/ccf_220_keras_lstm_public/1/variables/variables.index (deflated 67%)
    adding: saved_models/ccf_220_keras_lstm_public/1/variables/variables.data-00000-of-00001 (deflated 7%)
    adding: saved_models/ccf_220_keras_lstm_public/1/fitted_mapper.pkl (deflated 61%)
    adding: saved_models/ccf_220_keras_lstm_public/1/assets/ (stored 0%)
```

```
# Upload model to COS bucket
local_file_name = '{}.zip'.format(model_name)
cos_file_name = local_file_name

upload_large_file(cos_client, bucket_name, cos_file_name, local_file_name)
```

```
  Starting large file upload for ccf_220_keras_lstm_public.zip to bucket: test-datasets
  Large file upload complete!
```

4.5 AI model conversion

ONNX is an open format for AI models, allowing interchangeability of models between various AI frameworks and tools. Models developed using different frameworks can be saved or converted in ONNX format. For example, tf2onnx converts TensorFlow (tf-1.x or tf-2.x), keras, tensorflow.js and tflite models to ONNX via command line or python api. Refer to the tf2onnx information here.

Here is the sample code to convert demo TensorFlow model to ONNX format:

```
import tf2onnx

# convert to onnx
spec = (tf.TensorSpec((7, 16, 220), tf.float32, name="input"),)
onnx_model_tf2onnx_filename = '{}_tf2onnx.onnx'.format(model_name)

onnx_model_tf2onnx = tf2onnx.convert.from_keras(new_model, spec, output_path=onnx_model_tf2onnx_filename)
```

ONNX model can be validated.

```
# check onnx model
import onnx

# Preprocessing: load the ONNX model
model_path = onnx_model_tf2onnx_filename
onnx_model = onnx.load(model_path)

#print('The model is:\n{}'.format(onnx_model))

# Check the model
try:
    onnx.checker.check_model(onnx_model)
except onnx.checker.ValidationError as e:
    print('The model is invalid: %s' % e)
else:
    print('The model is valid!')
```

In the demo, TensorFlow model was converted to ONNX format, and it is also uploaded to COS bucket.

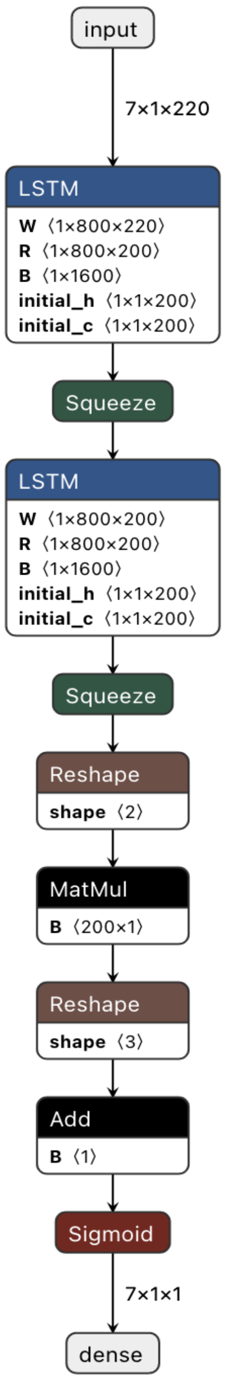
```
local_file_name = onnx_model_tf2onnx_filename
cos_file_name = onnx_model_tf2onnx_filename

upload_large_file(cos_client, bucket_name, cos_file_name, local_file_name)
```

```
Starting large file upload for ccf_220_keras_lstm_public_tf2onnx.onnx to bucket: test-datasets
Large file upload complete!
```

4.6 AI model visualization

Netron is an open-source multi-platform visualizer and editor for artificial intelligence models. It supports many extensions for deep learning, machine learning and neural network models. You can find more information here.

Here is the view of our demo model.



5. AI model deployment on mainframe

There are different ways to deploy AI models on IBM Z, for example:

- Deploying with IBM Machine Learning for z/OS (MLz)
- Deploying with MLz Online Scoring Community Edition (OSCE)
- Deploying with TensorFlow Serving
- IBM zDNN plugin for TensorFlow

IBM Machine Learning for z/OS (MLz) is an enterprise-grade and production-ready platform that enables embedding ML and DL models into transactional applications for real-time insights. With MLz, organizations can build, deploy, and run models on IBM Z and leverage several essential enterprise-grade ML features, such as model versioning, auditing, and monitoring. Models trained on other platforms can be converted to PMML or ONNX format and deployed on MLz. Refer to IBM Machine Learning for z/OS for details.

The MLz Online Scoring Community Edition (OSCE) is a special no-charge version of MLz that is intended for simple, non-production testing of the real-time scoring function of pretrained ONNX models. MLz OSCE can be used for rapid use case evaluation of embedding DL models in transactional z/OS applications while leveraging the Integrated Accelerator for AI. MLz OSCE is packaged as an s390x Docker container image that is easily deployed in IBM z/OS Container Extensions (zCX). IBM zCX enables clients to deploy Linux applications as Docker containers on z/OS as part of a z/OS workload.

TensorFlow Serving is an open-source, high-performance deployment option that is a good fit for enterprises that are heavily invested in the TensorFlow ecosystem or have complex model pipelines. TensorFlow Serving is available as a container image in the IBM Z Container Image Registry, and it can be used in a zCX or a Linux on IBM Z environment. Any z/OS application can access the TensorFlow model by using a REST API call.

IBM zDNN Plugin for TensorFlow can also be used to deploy TensorFlow model and utilize the IBM Integrated Accelerator on Z. IBM-zDNN-Plugin will detect the operations in your model that are supported by the Integrated Accelerator for AI and transparently target them to the device.

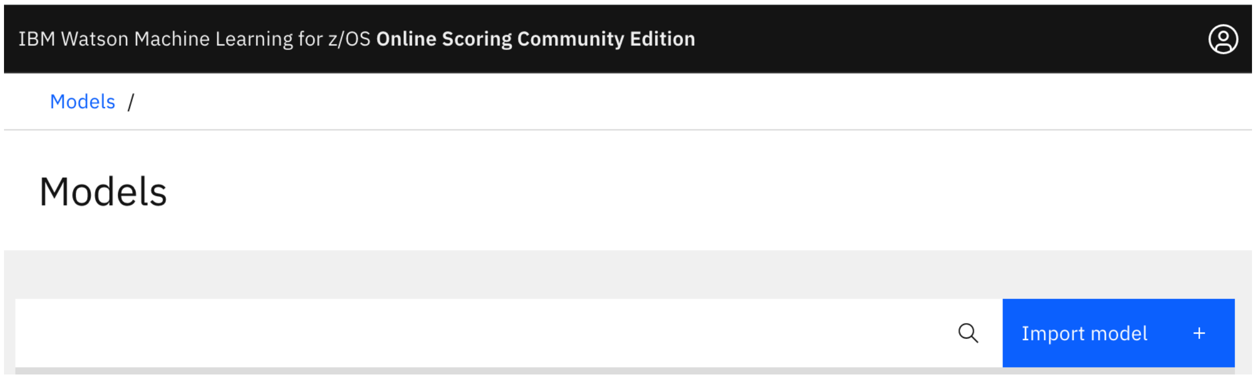
We will demonstrate how to deploy the TensorFlow model built in the previous section to MLz OSCE edition and TensorFlow Serving container.

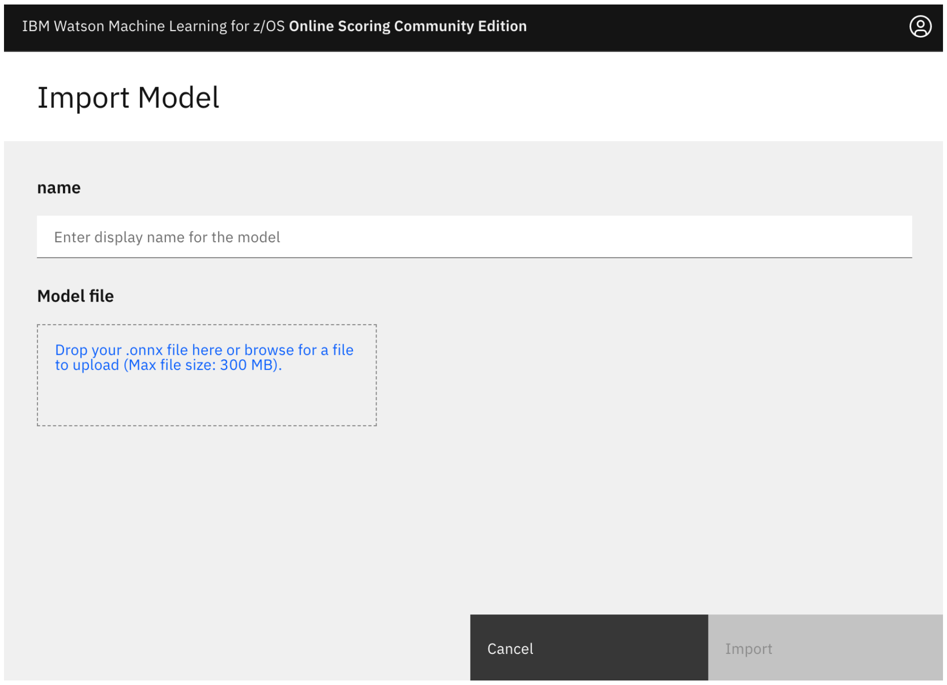
5.1 Deploy AI model in MLz OSCE

IBM Machine Learning for z/OS (MLz) is an enterprise machine learning solution that runs on IBM Z and Red Hat® OpenShift® Container Platform (OCP). It delivers predictive analytics capabilities to the platform and enables the generation of real-time insights at the source. MLz OSCE offers a no-charge option to test the MLz feature of real-time inferencing of pre-trained DL ONNX models.

You can set up IBM Machine Learning for z/OS Online Scoring Community Edition for testing purpose, and IBM Machine Learning for z/OS for production usage. Once LMz is setup, it has a link to the portal that you can use to manage your model.

(a) Import ONNX model
    Go to the MLz link and import the ONNX model.


IBM Watson Machine Learning for z/OS Online Scoring Community Edition
Models /
Models
Import model +


IBM Watson Machine Learning for z/OS Online Scoring Community Edition
Import Model
name
Enter display name for the model
Model file
Drop your .onnx file here or browse for a file to upload (Max file size: 300 MB).
Cancel
Import

(b) Test the model

Once the model is imported, you can click on the model, and inspect details. The General tab has the link to the scoring endpoint. The Schema tab has information about the input and output data. The Test tab allows you to give some sample payload and do a quick test on the model.

```
curl --location --request POST 'http://$SERVER_IP:15552/v1/models/saved_model:predict' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--data-raw '{"instances": [ ... test data ...]}'
```

5.2 Deploy AI model in TensorFlow serving container

TensorFlow Serving is an open-source, high-performance serving system for AI models. TensorFlow Serving is available as a container image in the IBM Z Container Image Registry, and it can be used in a zCX or a Linux on IBM Z environment.

TensorFlow models can be deployed in TensorFlow Serving container. TensorFlow models that are trained in TensorFlow 2.7 or later can be deployed on IBM Z without endian conversion, otherwise the byte order of the models need to be converted using this endian converter tool.

You can also find the IBM Z Accelerated Serving for TensorFlow container in the IBM Z Container Image Registry, which will leverage new inference acceleration capabilities that transparently target the IBM Integrated Accelerator for AI through the IBM z Deep Neural Network (zDNN) library.

Here are the steps to deploy model in TensorFlow Serving container:

(a) Get docker image
Here is the link to IBM image registry for z/OS, which has trusted container images for z/OS. Find the 'tensorflow-serving' entry and run the 'docker pull' command as documented.

(b) Start docker container

Here is the command to start docker container and serve the model.

```
docker run -t --rm -p $MODEL_PORT:$MODEL_PORT \
   -v "$MODEL_PATH:/models/$MODEL_NAME" \
   -e MODEL_NAME=$MODEL_NAME $TF_SERVING &
```

Here is sample setting for our demo:

```
MODEL_BASEDIR=/path/to/model
MODEL_NAME=ccf_220_keras_lstm_public
MODEL_PATH=$MODEL_BASEDIR/$MODEL_NAME
MODEL_PORT=8501
```

(c) Test the model
Once the model is deployed, a curl command can be used to validate the deployment.

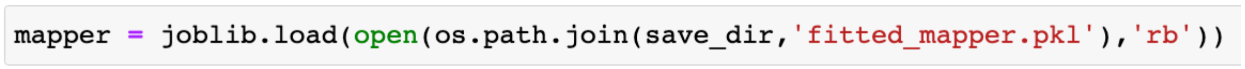
```
# invoke predict on the model
curl -X POST http://localhost:$MODEL_PORT/v1/models/$MODEL_NAME:predict -d '{"instances": [ … data payload … ]}'
```

6. AI model Inference

Now model is deployed, it can be invoked for real-time inference. Just like model training, data needs to be prepared and transformed to the format before invoking the model. The data transformations should be consistent with the ones that are done at model training time.

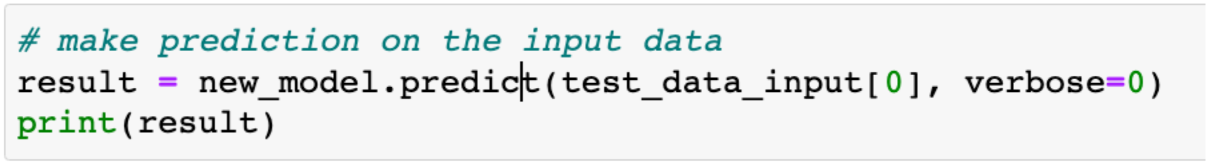
Take the credit card fraud detection sample for example, when the model is trained, past 7 transactions from the user is grouped together for model to learn the behavior of the user. At model inference time, when a new transaction is received, the past few transactions from the user should be looked up. Then the data is transformed to the format that can be fed to the model. Transformation could include encoding strings to numbers, normalizing numbers, etc. The data preparation and transformation application can be deployed in application that is collocated with the model.

The demo loads the saved information into a mapper and use it to transform the input data.



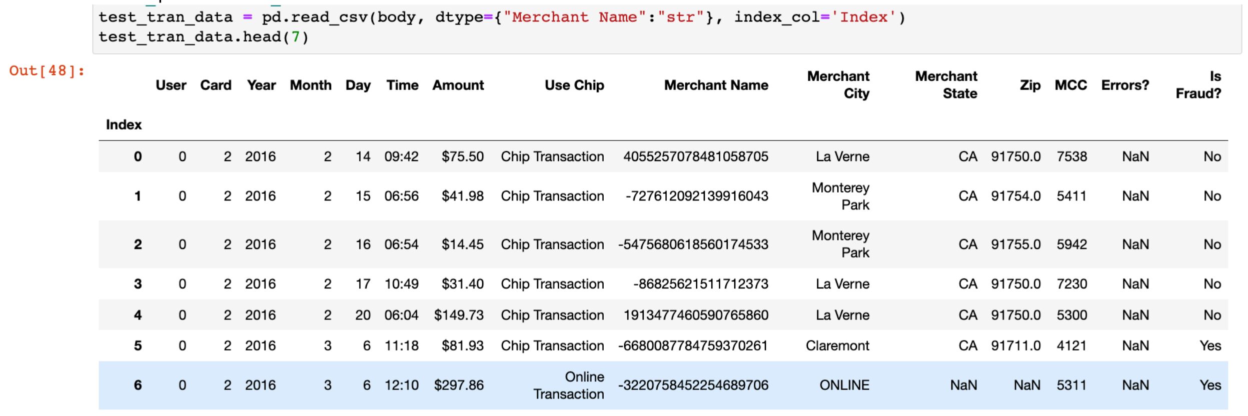
```
mapper = joblib.load(open(os.path.join(save_dir,'fitted_mapper.pkl'),'rb'))
```

The transformed data is then used for prediction.



```
# make prediction on the input data
result = new_model.predict(test_data_input[0], verbose=0)
print(result)
```

For this sample test data:

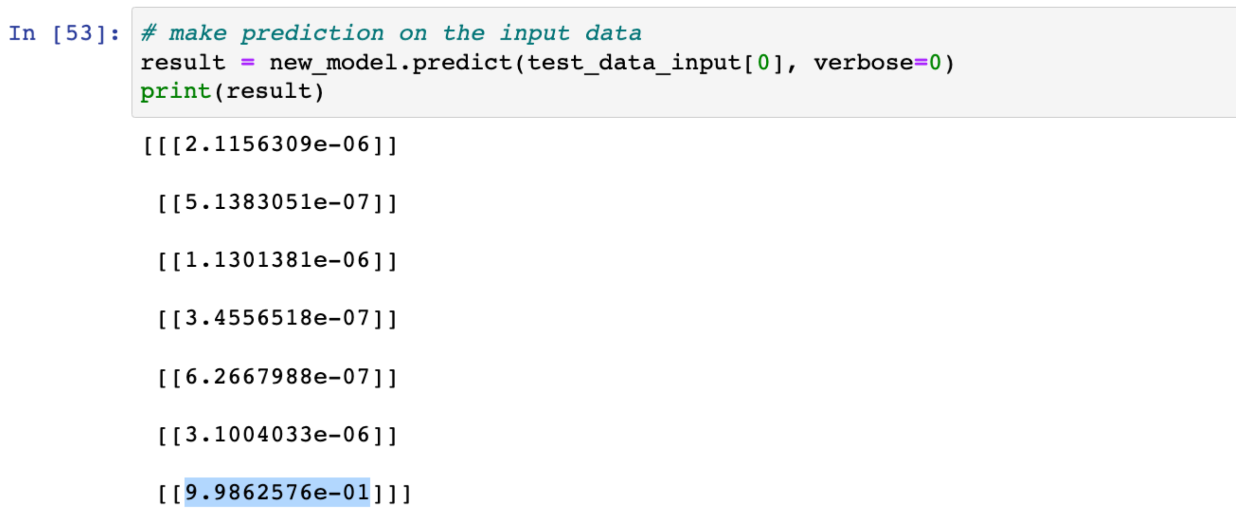


```
test_tran_data = pd.read_csv(body, dtype={"Merchant Name":"str"}, index_col='Index')
test_tran_data.head(7)
```

Out[48]:

Index	User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zip	MCC	Errors?	Is Fraud?
0	0	2	2016	2	14	09:42	$75.50	Chip Transaction	4055257078481058705	La Verne	CA	91750.0	7538	NaN	No
1	0	2	2016	2	15	06:56	$41.98	Chip Transaction	-727612092139916043	Monterey Park	CA	91754.0	5411	NaN	No
2	0	2	2016	2	16	06:54	$14.45	Chip Transaction	-5475680618560174533	Monterey Park	CA	91755.0	5942	NaN	No
3	0	2	2016	2	17	10:49	$31.40	Chip Transaction	-86825621511712373	La Verne	CA	91750.0	7230	NaN	No
4	0	2	2016	2	20	06:04	$149.73	Chip Transaction	1913477460590765860	La Verne	CA	91750.0	5300	NaN	No
5	0	2	2016	3	6	11:18	$81.93	Chip Transaction	-6680087784759370261	Claremont	CA	91711.0	4121	NaN	Yes
6	0	2	2016	3	6	12:10	$297.86	Online Transaction	-3220758452254689706	ONLINE	NaN	NaN	5311	NaN	Yes

Here is the sample inference, and you can see that the last transaction has high probability to be a fraudulent transaction.



```
In [53]: # make prediction on the input data
         result = new_model.predict(test_data_input[0], verbose=0)
         print(result)
```

```
[[[2.1156309e-06]]

 [[5.1383051e-07]]

 [[1.1301381e-06]]

 [[3.4556518e-07]]

 [[6.2667988e-07]]

 [[3.1004033e-06]]

 [[9.9862576e-01]]]
```

7. Conclusion

Companies are moving to cloud. Data scientists can train, tune, and test AI models in cloud, and benefit from the flexibility, scalability, and pay-as-you-go model. For highly regulated industries, eg. Financial industry, it is critical to protect the sensitive data and meet the regulatory requirements. IBM Cloud for Financial Services is designed to build trust and enable a transparent public cloud ecosystem with the features for security, compliance, and resiliency that financial institutions require. Financial institutions can confidently host their mission-critical applications in the cloud and transact quickly and efficiently.

Mainframe is still the main transaction processing engine for enterprises for unbeatable performance, reliability, and security. Companies try to build intelligence into their transaction processing logic for real-time workloads. They also try to process large volume of data with shrinking processing time. This requires low-latency large-scale AI model inference. The IBM Integrated Accelerator for AI Telum chip on z16 can help companies to achieve this goal.

Combining the secure environment in IBM Cloud with the low-latency inference capability on mainframe, companies can benefit from both worlds and achieve their goals in Hybrid cloud environment.