Developing Big Data Solutions with Azure Machine Learning

Lab 4-Building Predictive Big Data Solutions

Overview

In this lab, you will incorporate Azure Machine Learning web services into batch and real-time data processing operations.

What You'll Need

To complete the labs, you will need the following:

- A Microsoft account (for example, an *outlook.com*, *live.com*, or *hotmail.com* address)
- A Microsoft Azure subscription
- A Windows, Linux, or Mac OS X computer
- Azure Storage Explorer
- Node.JS
- The lab files for this course

Note: You must complete the preceding labs in this course before starting this one.

Exercise 1: Using a Predictive Web Service in a Batch Process

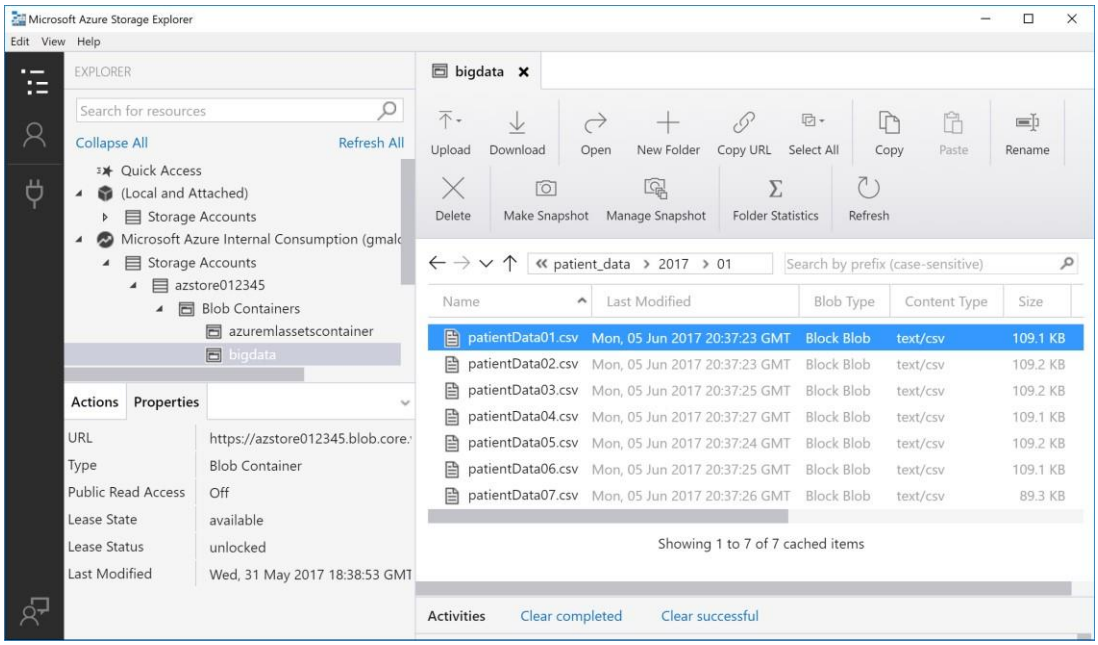
In this exercise, you will implement a data processing pipeline for a clinic that records patient data each day and then uses a nightly batch process to submit the data to the **DiabetesPredictor** web service you created in the previous lab and predict whether or not a patient may be diabetic.

Upload Data to Azure Storage

The patient data is collected each day, processed using a big data processing job, and stored in an Azure storage folder hierarchy that represents the year and month.

1. Start Azure Storage Explorer, and if you are not already signed in, sign into your Azure subscription.

2. Browse to the **bigdata** blob container you created in a previous lab, and in the **Upload** dropdown list, click **Upload Folder**. Then upload the **patient_data** folder from the **Lab04** folder on your local computer to the **bigdata** container.
3. In Azure Storage Explorer, double-click the **patient_data** folder to open it, and note that it contains a folder named **2017**. Open the **2017** folder and note that it contains a folder named **01**, and then open the **01** folder and note that it contains the daily patient data files, which are named in the format **patientData*NN*.csv**, where *NN* is the day of the month as shown here. These files include the daily data for the first seven days in January 2017.



Provision an Azure Data Factory

To orchestrate the data processing, you will use an Azure Data Factory pipeline. This requires an instance of Azure Data Factory.

Note: The Microsoft Azure portal is continually improved in response to customer feedback. The steps in this exercise reflect the user interface of the Microsoft Azure portal at the time of writing but may not match the latest design of the portal exactly.

1. In a web browser, navigate to http://portal.azure.com, and if prompted, sign in using the Microsoft account that is associated with your Azure subscription.
2. In the Microsoft Azure portal, in the menu, click **New**. Then in the **Data + Analytics** menu, click **Data Factory**.
3. In the **New data factory** blade, enter the following settings, and then click **Create**:
    - **Name**: *Enter a unique name*
    - **Subscription**: *Select your Azure subscription*
    - **Resource Group**: *Select the resource group you created previously*
    - **Location**: *Select the location you specified for your storage account (if it is not available, select any other location)*
    - **Pin to dashboard:** Unselected
4. In the Azure portal, view **Notifications** to verify that deployment has started. Then wait for the data factory to be deployed (this can take a few minutes.)

Create Linked Services

Now that you have an Azure Data Factory, you can create the components required to implement the data processing pipeline. The first of these components are the linked services that enable Azure Data Factory to connect to your Azure storage account and your Azure Machine Learning web service.

1. In the Azure portal, browse to the blade for your Azure Storage account, and under **Settings**, click **Access keys**. Then copy the **connection string** for **key 1** to the clipboard and paste it into a new text file using a text editor such as Notepad. You will need this to configure the Azure Data Factory linked service for your Azure Storage account.
2. In the Azure portal, browse to the blade for the **DiabetesPredictor** web service you created in the previous lab, and then click **View this Web Service in Machine Learning Web Service Management** to open the web service management tool in a new browser tab.
3. In the web service management tool, view the **Consume** page, and then copy the **Primary Key** and **Batch Requests** URL to the text file where you pasted the Azure Storage connection string. You will need these to configure the Azure Data Factory linked service for your Azure Machine Learning web service.
4. In the Azure portal, browse to the blade for your Azure Data Factory and click the **Author and Deploy** tile.
5. In the pane on the left, click **New data store**, and select **Azure Storage**.
6. In the JSON definition of the linked service, replace the default connection string with the connection string for your storage account. The resulting JSON should resemble this:

```
{
   "name": "AzureStorageLinkedService",
    "properties": {
      "type": "AzureStorage",
      "typeProperties": {
        "connectionString": "DefaultEndpointsProtocol=..."
      }
    }
}
```

7. Click **Deploy** to deploy the linked service to your Azure Data Factory.
8. In the pane on the left, click **More**, and then click **New compute** and select **Azure ML**.
9. In the JSON definition of the linked service, replace the default **mlEndpoint** URL with the Batch Request URL for your Diabetes Predictor web service, replace the **apiKey** value with the primary key for your Diabetes Predictor web service, and delete the **updateResourceEndpoint**, **servicePrincipalId**, and **servicePrincipalKey** properties. The resulting JSON should resemble this:

```
{
    "name": "AzureMLLinkedService",
    "properties": {
      "type": "AzureML",
      "typeProperties": {
        "mlEndpoint": "https://.../jobs?api-version=2.0",
        "apiKey": "1234567890...=="
        "tenant": "xxxx.onmicrosoft.com"
      }
    }
}
```

10. Click **Deploy** to deploy the linked service to your Azure Data Factory.

Create Datasets

Now that you have defined the linked services for your data processing workflow, you must define datasets for the source patient data and the data generated by your web service.

1. In the **More** menu, click **New dataset** and select **Azure Blob storage**.
2. Replace the default JSON for the dataset with the following, which you can copy and paste from **patientData.json** in the **Lab04** folder:

```
{
  "name": "patientData",
"properties": {
    "type": "AzureBlob",
    "linkedServiceName": "AzureStorageLinkedService",
    "structure": [],
    "typeProperties": {
      "folderPath": "bigdata/patient_data/{year}/{month}",
      "fileName": "patientData{date}.csv",
      "partitionedBy": [
        {
          "name": "year",
          "value": {
            "type": "DateTime",
            "date": "SliceStart",
            "format": "yyyy"
          }
        },
        {
          "name": "month",
          "value": {
            "type": "DateTime",
            "date": "SliceStart",
            "format": "MM"
          }
        },
        {
          "name": "date",
          "value": {
            "type": "DateTime",
            "date": "SliceStart",
            "format": "dd"
          }
        }
      ],
      "format": {
        "type": "TextFormat",
"firstRowAsHeader": true
      }
    },
    "external": true,
    "availability": {
      "frequency": "Day",
      "interval": 1
```

```
    },
    "policy": {
      "externalData": {
        "retryInterval": "00:01:00",
        "retryTimeout": "00:10:00",
        "maximumRetry": 3
      }
    }
  }
}
```

Note: This JSON defines a dataset for a text file stored as an Azure blob. The path to the blob includes variables for *year*, *month*, and *date* in formats *yyyy*, *MM*, and *dd* respectively.

3. Click **Deploy** to deploy the dataset to your Azure Data Factory.
4. In the **More** menu, click **New dataset** and select **Azure Blob storage**.
5. Replace the default JSON for the dataset with the following, which you can copy and paste from **predictions.json** in the **Lab04** folder:

```
{
  "name": "predictions",
"properties": {
    "type": "AzureBlob",
    "linkedServiceName": "AzureStorageLinkedService",
    "structure": [],
    "typeProperties": {
      "folderPath": "bigdata/predictions",
      "fileName": "diabetes_predictions_{date}.csv",
      "partitionedBy": [
        {
          "name": "date",
          "value": {
            "type": "DateTime",
            "date": "SliceStart",
            "format": "yyyyMMdd"
          }
        }
      ],
        "format":{
          "type": "TextFormat",
"firstRowAsHeader": true
        }
    },
    "external": false,
    "availability": {
      "frequency": "Day",
      "interval": 1
    }
  }
}
```

Note: This JSON defines a dataset for a text file stored as an Azure blob. The path to the blob includes a variable for *date* in the format *yyyyMMdd*.

6. Click **Deploy** to deploy the dataset to your Azure Data Factory.

Create a Pipeline

With the required linked services and datasets defined, you can now use them in a pipeline to automate your data processing workflow.

1. In the **More** menu, click **New pipeline**.
2. Replace the default JSON for the pipeline with the following, which you can copy and paste from **ProcessPatientData.json** in the **Lab04** folder. In this JSON, replace ***<server>*** with the name of your Azure SQL Database server.

```
{
  "name": "Process Patient Data",
  "properties": {
    "description": "Pipeline to predict diebetes",
    "activities": [
      {
        "name": "Predict Diabetes",
        "type": "AzureMLBatchExecution",
        "linkedServiceName": "AzureMLLinkedService",
        "typeProperties": {
          "webServiceInput": "patientData",
          "webServiceOutputs": {
            "output1": "predictions"
          },
          "globalParameters": {
            "Database server name": "<server>.database.windows.net"
          }
        },
        "inputs": [
          {
            "name": "patientData"
          }
        ],
        "outputs": [
          {
            "name": "predictions"
          }
        ],
        "policy": {
          "concurrency": 1,
          "executionPriorityOrder": "OldestFirst",
          "retry": 3,
          "timeout": "01:00:00"
        },
        "scheduler": {
          "frequency": "Day",
          "interval": 1
        }
      }
    ],
    "start": "2017-01-01T00:00:00Z",
    "end": "2017-01-07T23:59:59Z"
  }
```

```
}
```

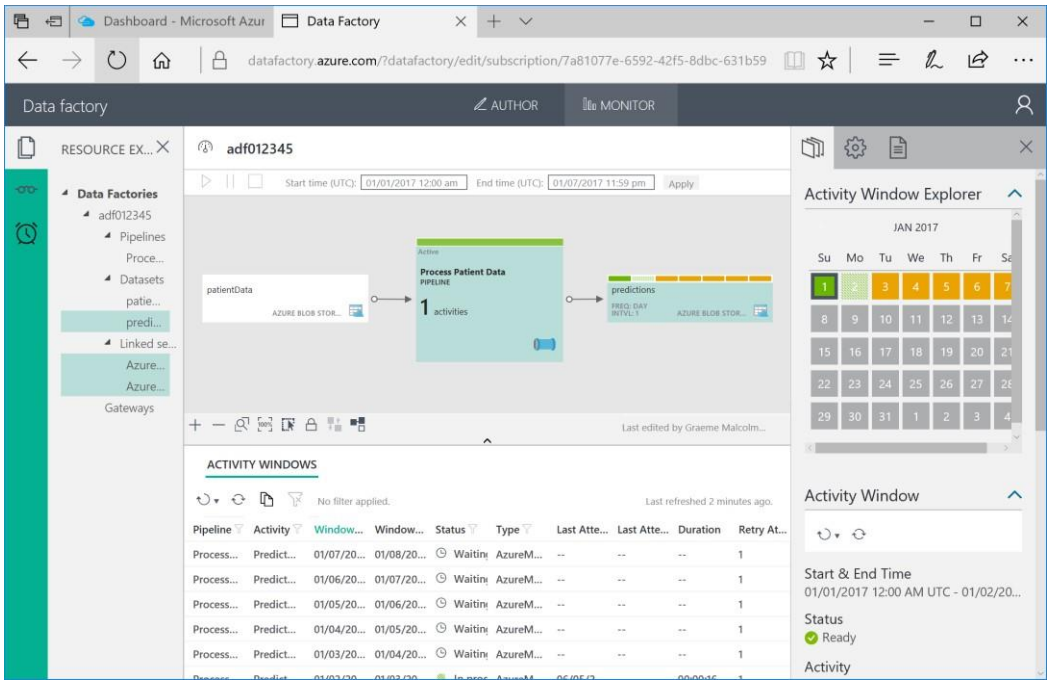
Note: This JSON defines a pipeline that runs daily from January 1st 2017 to January 7th 2017. The pipeline contains a single activity, which submits the daily patient data files to the Diabetes Predictor web service, and writes the web service output to a text file in Azure Storage.

3. Click **Deploy** to deploy the pipeline to your Azure Data Factory.

Monitor the Pipeline

The pipeline will start to run as soon as it is deployed. You can monitor its progress in the portal.

1. In the Azure portal, in the blade for your Azure Data Factory, click the **Monitor and Manage** tile. This opens a new tab in your browser.
2. In the Monitor and Manage tool, view the pipeline diagram and the list of activity windows beneath it.
3. Above the diagram, click the start time and change it to 0:00 on January 1st 2017, and click **Done**. Then click the end time and change it to 11:59pm on January 7th 2017, and click **Done**. Click **Apply** for the filter to take effect.
4. In the list of activity windows, at the bottom of the screen, select any of the activities. The Activity Window Explorer pane will show the status for each of the daily activity windows as orange for waiting, partially green for running, and solid green for completed; as shown here.



5. In the **Activity Window Explorer** pane, click the **Refresh this panel** button every minute or so to update the status until all the activity windows are complete.

View the Output

Now that the pipeline has processed the daily patient data, you can view the resulting predictions.

1. In Azure Storage Explorer, view the root of the **bigdata** container. It should now contain a new folder named **predictions**.
2. Open the **predictions** folder, and note that it contains a file for each day's predictions with a name such as **diabetes_predictions_20170101.csv**.

3. Double-click any of the files to open it in your default CSV editor, and note that each file contains a row for each patient with the patient ID, a prediction for whether the patient is diabetic, a probability value that indicates the confidence of the prediction, and the name of the patient's physician.

Exercise 2: Retraining a Predictive Model in a Batch Process

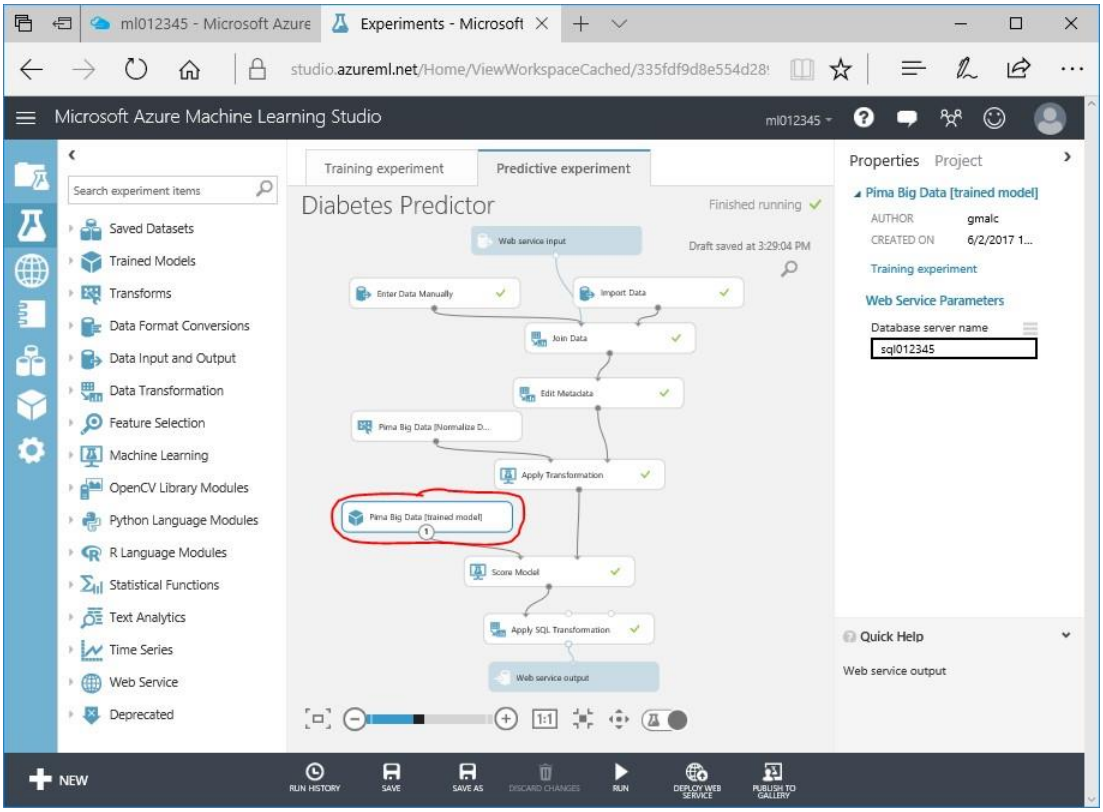
In this exercise, you will extend the batch processing solution for diabetes predictions by automating the retraining of a model and updating the associated web service. In this scenario, the clinic continues to collect patient data, and notes which patient have in fact tested positive for diabetes. This new data is periodically used to retrain the machine learning model used by the **DiabetesPredictor** web service.

Note: The procedures to build a retraining solution for a classic web service are different from those for retraining an Azure Resource Manager web service. This exercise is based on the steps for retraining an Azure Resource Manager web service.

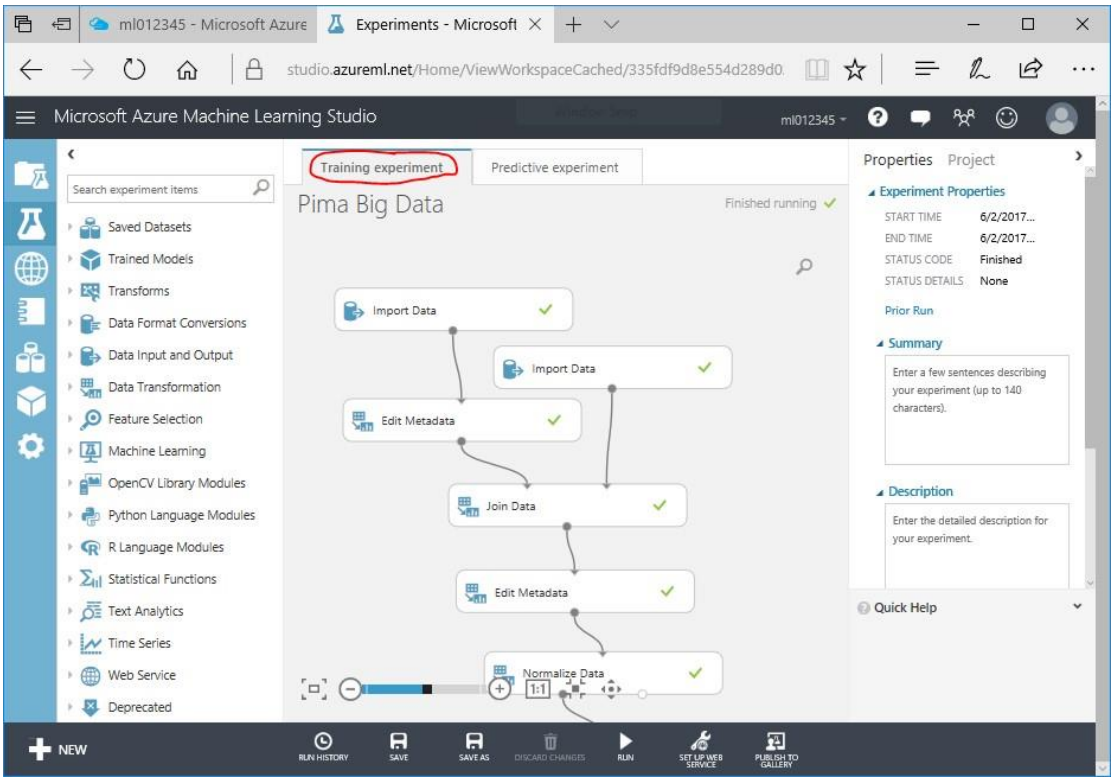
Publish a Training Web Service

The Diabetes Predictor web service is a predictive (or *scoring*) web service – it is used to generate predictions from the machine learning model you trained. To retrain the model, you must also publish the training experiment as a web service.

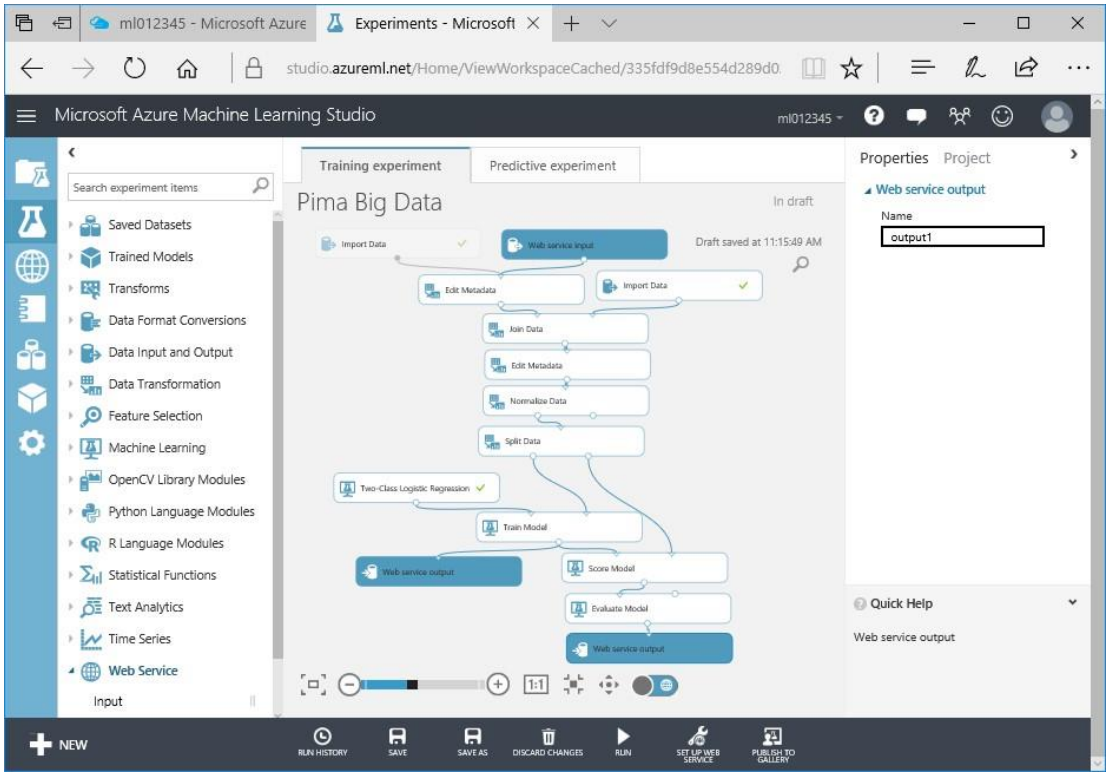
1. In the Azure portal, browse to your machine learning workspace and launch Machine Learning Studio.
2. In Azure Machine Learning Studio, open the **Diabetes Predictor** experiment, and note the name of the trained classification model it contains (which should be **Pima Big Data [trained model]**).



3. Click the **Training experiment** tab, to view the training experiment that the predictive experiment is based on:



4. Run the experiment, and verify that it completes with no errors.
5. Click **Set Up Web Service**, and select **Retraining Web Service**. If a dialog box describing the features of your new web service is displayed, read each page, clicking the **Next** button to navigate through it. Your new web service should contain all the modules from your original training experiment, a new web service input, and two web service outputs, as shown here:



6. Select the **Web service output** that is connected to the output of the **Evaluate Model** module. This is named **output1**, and returns the scored results generated from the test data.
7. Select the **Web service output** that is connected to the output of the **Train Model** module. This is named **output2**, and returns the trained model as a *.ilearner* file.
8. Run the experiment, and verify that it completes without error.
9. Click **Setup Web Service** and select **Deploy Web Service [New] Preview**.
10. Deploy the web service with the following settings:
    - **Web Service Name**: TrainDiabetesModel • **Storage Account**: Your Azure storage account.
    - **Price plan**: Your Azure Machine learning plan.
11. When the web service has been deployed, click the **Consume** tab. Then copy the **Primary Key** and **Batch Requests** URL to a text file – you will need these later.

Gather Predictive Web Service Information

After using the **TrainDiabetesModel** web service to generate a new model, your batch process will need to call the **DiabetesPredictor** web service, passing it the .ilearning model file to update the model it uses to predict diabetic patients. You will therefore need to gather some details for the **DiabetesPredictor** Web service.

1. In the web services management page, click ← **Web Services**.
2. In the list of web services, click **DiabetesPredictor**.
3. On the **Consume** tab, copy the **Primary Key** and **Batch Requests** URL to your text file. Make sure you clearly note which URL and key belong to the **TrainDiabetesModel** web service, and which belong to the **DiabetesPredictor** web service.
4. Switch back to the web browser tab for the Azure portal, and view the blade for the **DiabetesPredictor** web service. Note that this blade displays the **Resource group** and **Subscription name** associated with the web service, and then copy these values to your text file. You will need these to construct the *update resource* endpoint URL used to apply the retrained model to your web service.
5. Verify that your text file now contains the following six items:
    - The Batch URL for the **TrainDiabetesModel** web service
    - The API Key for the **TrainDiabetesModel** web service
    - The Batch URL for the **DiabetesPredictor** web service
    - The API Key for the **DiabetesPredictor** web service
    - The name of the resource group for the **DiabetesPredictor** web service
    - The Subscription ID for the Azure subscription containing the resource group

Create a Service Principal

Your batch process will update the predictive web service by calling the Azure resource management API. To do this successfully, the batch process must use an account that has sufficient permissions to update the resource. You will therefore create an Azure Active Directory (AAD) service principal for the batch process to use, and give it permission to access the resource group.

Note: You must be an administrator in your Azure subscription to create a service principal.

1. In the Azure Portal, in the menu, click **Azure Active Directory**.
2. In the Azure Active Directory blade, click **App registrations**.

3. Click **+ New application registration**, and create a new application with the following settings:
    - **Name**: DiabetesUpdater
    - **Application type**: Web app / API
    - **Sign-on URL**: The Batch Request URL for your **DiabetesPredictor** web service.
4. In the **App registrations** blade, search for the **DiabetesUpdater** app you just created and open its blade. Then copy its **Application ID** to your text file.
5. In the blade for the **DiabetesUpdater** app, click **Keys**. Then enter the following details and click **Save** to generate a key value.
    - **Description**: Diabetes Key
    - **Expires**: In 1 year.
6. Copy the generated key value to your text file. Note that you cannot retrieve the value after closing this blade!)
7. In the Azure portal menu, click **Resource groups**, and select the resource group in which your **DiabetesPredictor** web service is defined.
8. In the blade for your resource group, click **Access control (IAM)**.
9. In the **Access control (IAM)** blade, click **+ Add**.
10. In the **Add permissions** blade, select the following settings and click **Save**:
    - **Role**: Contributor
    - **Select**: DiabetesUpdater
11. Verify that your text file now contains the following six items:
    - The Batch URL for the **TrainDiabetesModel** web service
    - The API Key for the **TrainDiabetesModel** web service
    - The Batch URL for the **DiabetesPredictor** web service
    - The API Key for the **DiabetesPredictor** web service
    - The name of the resource group for the **DiabetesPredictor** web service
    - The Subscription ID for the Azure subscription containing the resource group
    - The Application ID of the **DiabetesUpdater** service principle
    - The Key for the **DiabetesUpdater** service principle

Upload Training Data to Azure Storage

The training data will be updated each month as new patient data is collected. It must be stored in a text file that matches the input schema expected by the training web service.

1. In Azure Storage Explorer, browse to the **bigdata** blob container you created in a previous lab, and in the **Upload** drop-down list, click **Upload Folder**. Then upload the **training** folder from the **Lab04** folder on your local computer to the **bigdata** container.
2. In Azure Storage Explorer, double-click the **training** folder to open it, and note that it contains a file named **training.csv**. This file contains training data for the **Pima Big Data [trained model]** classification model in the **DiabetesPredictor** web service.

Create Linked Services in Azure Data Factory

Now that you have prepared the web services and service principal, and uploaded the training data, you can create linked services in your Azure Data Factory to retrain the machine learning model. You will need two linked services: one to call the **TrainDiabetesModel** web service, and one to update the **DiabetesPredictor** web service.

1. In the Azure portal, browse to the blade for your Azure Data Factory and click the **Author and Deploy** tile.
2. In the pane on the left, click **More**, then click **New compute** and select **Azure ML**.
3. In the JSON definition of the linked service, change the name to **TrainWS** and replace the default **mlEndpoint** URL with the Batch Request URL for your **TrainDiabetesModel** web service, replace the **apiKey** value with the primary key for your **TrainDiabetesModel** web service, and delete the **updateResourceEndpoint**, **servicePrincipalId**, and **servicePrincipalKey** properties. The resulting JSON should resemble this:

```
{
    "name": "TrainWS",
    "properties": {
      "type": "AzureML",
      "typeProperties": {
        "mlEndpoint": "https://.../jobs?api-version=2.0",
        "apiKey": "1234567890...=="
        "tenant": "xxxx.onmicrosoft.com"
      }
    }
}
```

4. Click **Deploy** to deploy the linked service to your Azure Data Factory.
5. In the pane on the left, click **More**, then click **New compute** and select **Azure ML**.
6. In the JSON definition of the linked service, make the following changes: 1) Change the name to **UpdateWS**.
    2) Replace the default **mlEndpoint** URL with the Batch Request URL for your **DiabetesPredictor** web service.
    3) Replace the **apiKey** value with the primary key for your **DiabetesPredictor** web service.
    4) Replace the **updateResourceEndpoint** with the following URL (including your *subscription ID* and *resource group name* where indicated):

       https://management.azure.com/subscriptions/***subscription_ID***/resourceGroups/***resource_group***/providers/Microsoft.MachineLearning/webServices/DiabetesPredictor?api-version=2016-05-01-preview

    5) Replace the **servicePrincipalId** with the Application ID for your **DiabetesUpdater** service principal.
    6) Replace the **servicePrincipalKey** with the key for your **DiabetesUpdater** service principal.

    The resulting JSON should resemble this:

```
{
    "name": "UpdateWS",
    "properties": {
        "type": "AzureML",
        "typeProperties": {
          "mlEndpoint": "https://.../jobs?api-version=2.0",
          "apiKey": "abcdefg...==",
          "updateResourceEndpoint": "https://management... ",
          "servicePrincipalId": "1234567890...",
          "servicePrincipalKey": "12345...==",
          "tenant": "xxxx.onmicrosoft.com"
```

```
        }
      }
}
```

7. Click **Deploy** to deploy the linked service to your Azure Data Factory.

Create Datasets

Now that you have defined the linked services to retrain and update the predictive model, you must define datasets for the source training data and the trained model generated by your training web service. Additionally, you must define a "dummy" dataset to represent the output of the operation to update the predictive web service (even though this operation does not in fact generate any output).

1. In the **More** menu, click **New dataset** and select **Azure Blob store**.
2. Replace the default JSON for the dataset with the following, which you can copy and paste from **trainingData.json** in the **Lab04** folder:

```
{
  "name": "trainingData",
  "properties": {
    "type": "AzureBlob",
    "linkedServiceName": "AzureStorageLinkedService",
    "structure": [],
    "typeProperties": {
      "folderPath": "bigdata/training/",
      "fileName": "training.csv",
      "format": {
        "type": "TextFormat",
"firstRowAsHeader": true
      }
    },
    "external": true,
    "availability": {
      "frequency": "Month",
      "interval": 1
    },
    "policy": {
      "externalData": {
        "retryInterval": "00:01:00",
        "retryTimeout": "00:10:00",
        "maximumRetry": 3
      }
    }
  }
}
```

Note: This JSON defines a dataset for the **training.csv** file in the **training** folder in Azure Storage.

3. Click **Deploy** to deploy the dataset to your Azure Data Factory.
4. In the **More** menu, click **New dataset** and select **Azure Blob store**.
5. Replace the default JSON for the dataset with the following, which you can copy and paste from **trainedModel.json** in the **Lab04** folder:

```
{
  "name": "trainedModel",
```

```
  "properties": {
    "type": "AzureBlob",
    "linkedServiceName": "AzureStorageLinkedService",
    "typeProperties": {
      "folderPath": "bigdata/training/model",
      "fileName": "model.ilearner",
      "format": {
        "type": "TextFormat"
      }
    },
    "availability": {
      "frequency": "Month",
      "interval": 1
    }
  }
}
```

Note: This JSON defines a dataset for the .ilearner model file generated by your training web service.

6. Click **Deploy** to deploy the dataset to your Azure Data Factory.
7. In the **More** menu, click **New dataset** and select **Azure Blob store**.
8. Replace the default JSON for the dataset with the following, which you can copy and paste from **trainingOutput.json** in the **Lab04** folder:

```
{
  "name": "trainingOutput",
  "properties": {
    "availability": {
      "frequency": "Month",
      "interval": 1
    },
    "type": "AzureBlob",
    "linkedServiceName": "AzureStorageLinkedService",
    "typeProperties": {
      "folderPath": "bigdata/training/output",
      "format": {
        "type": "TextFormat"
      }
    }
  }

}
```

Note: All activities in an Azure Data Factory pipeline require an output dataset. Updating a web service with a retrained model does not generate any data, so this JSON defines a dummy output dataset for the activity.

9. Click **Deploy** to deploy the dataset to your Azure Data Factory.

Create a Pipeline

With the required linked services and datasets defined, you can now use them in a pipeline to automate your model retraining workflow.

4. In the **More** menu, click **New pipeline**.
5. Replace the default JSON for the pipeline with the following, which you can copy and paste from **UpdateModel.json** in the **Lab04** folder.

```
{
  "name": "Update Diabetes Model",
  "properties": {
    "description": "Update Diabetes Prediction Model",
    "activities": [
      {
        "name": "Retrain Model",
        "type": "AzureMLBatchExecution",
        "inputs": [
          {
            "name": "trainingData"
          }
        ],
        "outputs": [
          {
            "name": "trainedModel"
          }
        ],
        "typeProperties": {
          "webServiceInput": "trainingData",
          "webServiceOutputs": {
            "output2": "trainedModel"
          }
        },
        "linkedServiceName": "TrainWS",
        "policy": {
          "concurrency": 1,
          "executionPriorityOrder": "OldestFirst",
          "retry": 1,
          "timeout": "02:00:00"
        }
      },
      {
        "name": "Update Predictive Web Service",
        "type": "AzureMLUpdateResource",
        "linkedServiceName": "UpdateWS",
        "typeProperties": {
          "trainedModelDatasetName": "trainedModel",
          "trainedModelName": "Pima Big Data [trained model]"
        },
        "inputs": [
          {
            "name": "trainedModel"
          }
        ],
        "outputs": [
          {
            "name": "trainingOutput"
          }
        ],
```

```
        "policy": {
          "concurrency": 1,
          "executionPriorityOrder": "OldestFirst",
          "retry": 3,
          "timeout": "01:00:00"
        },
        "scheduler": {
          "frequency": "Month",
          "interval": 1
        }
      }
    ],
    "start": "2017-02-01T00:00:00Z",
    "end": "2017-02-01T23:59:59Z"
  }
}
```

Note: This JSON defines a pipeline that runs monthly from February 1st 2017 (the schedule is set so only one run will occur in this lab – in a real solution you would set the end date to a future date so that the pipeline runs every month.) The pipeline contains two activities: one to call the **TrainDiabetesModel** web service and generate a trained model, and one to submit the new model to the **DiabetesPredictor** web service and replace the **Pima Big Data [Trained Model]** classification model that it contains.

6. Click **Deploy** to deploy the pipeline to your Azure Data Factory.

Monitor the Pipeline

The pipeline will start to run as soon as it is deployed. You can monitor its progress in the portal.

1. In the Azure portal, in the blade for your Azure Data Factory, click the **Monitor and Manage** tile. This opens a new tab in your browser.
2. In the Monitor and Manage tool, view the pipeline diagram and the list of activity windows beneath it.
3. Above the diagram, click the start time and change it to 0:00 on January 1st 2017, and click **Done**. Set the end time to 23:59 on February 2nd 2017 and click **Done**. Then click **Apply** for the filter to take effect.
4. Right-click the **Update Diabetes Pipeline** and click **Open Pipeline** to see the datasets and activities the pipeline contains.
5. Select the **Retrain Model** activity and note that this highlights the activity window. The status of the activity window is indicated as a colored bar at the top of the activity in the pipeline diagram.
6. Select the **Update Predictive Web Service** activity and note that its activity window is selected.
7. Refresh the **Activity Windows** list until the status of all activity windows is **ready**.

View the Output

Now that the pipeline has finished, the model has been retrained and the predictive web service has been updated. You can view the files that were generated during this process.

1. In Azure Storage Explorer, refresh the view of the **training** folder in the **bigdata** container. It should now contain a new folder named **model**.

2. Open the **model** folder, and note that it contains a file named **model.ilearner**. This is the trained model generated by your training web service. The folder also includes a file with an autogenerated name, which contains details of the trained model.

Exercise 3: Using a Predictive Web Service in a Realtime Process

In this exercise, you will use a predictive web service in a real-time streaming process. The diabetes clinic has decided to research the effect of fitness on diabetes treatment and needs to monitor patient exercise. A streaming process will capture fitness monitor data from patients undergoing exercise, and uses the **CaloriePredictor** predictive web service to estimate the calories burned over one-minute intervals.

Create a Database Table for Calorie Predictions

The patient fitness data will be used to predict calories burned each minute, and these predictions will be stored in a database table for reporting. You must therefore create a table in the Azure SQL database instance you created previously.

1. In the Azure portal, browse to the blade for the **DiabetesData** database you created in a previous lab, and then on the **Tools** menu, click **Query editor** to open the web-based query interface for your database.
2. In the **Query editor** blade, click **Login** and then log in to your Azure SQL database server using **SQL server authentication** with the username and password you specified when provisioning the database.
3. After you have been authenticated, copy and paste the code in the **CreateCalorieTable.txt** script file in the **Lab04** folder into the empty query editor. This script creates a table named **dbo.Calories**.
4. Click **Run**, and then wait for the query to complete. Then when the query has completed, close the query editor blade, discarding the changes to your script.

Upload User Profile Data

The fitness devices only emit physiological readings such as body temperature and heart rate, as well as a unique device user identifier that correlates to the patient to whom the device is assigned. The streaming process must therefore look up the user profile data required by the **CaloriePredictor** web service based on the device user ID.

1. In Azure Storage Explorer, browse to the **bigdata** blob container you created in a previous lab, and in the **Upload** drop-down list, click **Upload Folder**. Then upload the **device_users** folder from the **Lab04** folder on your local computer to the **bigdata** container.
2. In Azure Storage Explorer, double-click the **device_users** folder to open it, and note that it contains a file named **profiles.csv**. This file contains patient details (ID, gender, age, height, and weight) for each user assigned to one of ten fitness devices.

Create an Event Hub

The patient fitness data will be ingested into an event hub for processing. You will therefore need to create an event hub in a service bus namespace, and define a shared access policy that enables the fitness app to connect and submit data.

1. In the Microsoft Azure portal, in the Hub Menu, click **New**. Then in the **Internet of Things** menu, click **Event Hubs**.

2. In the **Create namespace** blade, enter the following settings, and then click **Create**:
    - **Name**: *Enter a unique name*
    - **Pricing tier**: Basic
    - **Subscription**: *Select your Azure subscription*
    - **Resource Group**: *Select the resource group you created previously*
    - **Location:** *Select any available region*
    - **Pin to dashboard**: *Not selected*
3. In the Azure portal, view **Notifications** to verify that deployment has started, and wait for the namespace to be deployed (this can take a few minutes.) Then browse to the blade for the namespace.
4. In the blade for your namespace, click **Add Event Hub**.
5. In the **Create Event Hub** blade, enter the following settings (other options may be shown as unavailable) and click **Create**:
    - **Name**: *Enter a unique name*
    - **Partition Count**: 2
6. In the Azure portal, wait for the notification that the event hub has been created. Then, in the blade for your namespace, select the event hub you just created.
7. In the blade for your event hub, click **Shared access policies**.
8. On the **Shared access policies** blade for your event hub, click **+ Add**. Then add a shared access policy with the following settings:
    - **Policy name**: DeviceAccess
    - **Claim**: Send
9. Wait for the shared access policy to be created, then select it, and note that primary and secondary connection strings have been created for it. Copy the **primary connection string** to the clipboard - you will use it to connect to your event hub from simulated fitness devices later in this exercise.
10. On the **Shared access policies** blade for your event hub, click **+ Add**. Then add a second shared access policy with the following settings:
    - **Policy name**: StreamAccess
    - **Claim**: Listen

    You will use this shared access policy to connect to your event hub from an Azure Stream Analytics job to read the data submitted to the event hub by the fitness devices.

Simulate Exercise Readings

You will use a Node.JS script to simulate ten concurrent fitness devices.

1. Open a command line console and navigate to the **fitness_device** folder in the **Lab04** folder where you extracted the lab files.
2. Enter the following command, and press RETURN to accept all the default options. This creates a package.json file for your application:

    ```
    npm init
    ```

3. Enter the following command to install the Azure Event Hubs package:

    ```
    npm install @azure/event-hubs
    ```

4. Use a text editor to edit the **exercise.js** file in the **fitness_device** folder.
5. Modify the script to set the **connStr** variable to reflect your shared access policy connection string, as shown here:

```
var EventHubClient = require('@azure/event-hubs').Client;
 var connStr =
'<EVENT_HUB_CONNECTION_STRING>';
 var client = EventHubClient.fromConnectionString(connStr)
client.createSender()     .then(function (tx) {
setInterval(function(){             usr =
String(Math.floor((Math.random() * 10) + 1));             hr =
String(100 - (Math.random() * 5));             bt = String(35 +
Math.random());             t = new Date().toISOString();
            console.log("User:" + usr + ", Heart: " + hr + ", Temp: " + bt);
tx.send({eventtime: t, userid: usr, heartrate: hr, bodytemp: bt});
         }, 200);
    });
```

Note: This script sends a random set of fitness readings for one of ten users every 200 milliseconds. The data emitted by the script consists of a JSON object containing the fields **eventtime**, **userid**, **heartrate**, and **bodytemp**.

6. Save the script and close the file.
7. Keep the command line window open at the **fitness_device** folder.

Create a Stream Analytic Job

You will use an Azure Stream Analytics job that will read the data in the event hub, join it to the user profile data in azure storage, and write the predicted calories to Azure SQL Database.

1. In the Microsoft Azure portal, in the Hub Menu, click **New**. Then in the **Internet of Things** menu, click **Stream Analytics job**.
2. In the **New Stream Analytics Job** blade, enter the following settings, and then click **Create**:
   - **Name**: *Enter a unique name*
   - **Subscription**: *Select your Azure subscription*
   - **Resource Group**: *Select the resource group containing your existing resources*
   - **Location:** *Select any available region*
   - **Pin to dashboard**: *Not selected*
3. In the Azure portal, view **Notifications** to verify that deployment has started, and wait for the job to be deployed (this can take a few minutes.) Then browse to the Stream Analytics job you created previously.
4. In the blade for your Stream Analytics job, in the **Job Topology** section, click the **Inputs** tile.
5. In the **Inputs** blade, click **+ Add**.
5. In the **New input** blade, enter the following settings to define an input for the fitness device data in the event hub, and then click **Create**:
   - **Input alias**: DeviceData
   - **Source Type**: Data stream
   - **Source**: Event hub
   - **Import option:** Use event hub from current subscription
   - **Service bus namespace**: *Select your service bus namespace*
   - **Event hub name:** *Select your event hub*

- **Shared Access Policy Name:** StreamAccess
- **Consumer group:** *Leave blank to use the $Default consumer group*
- **Event serialization format:** JSON
- **Encoding:** UTF-8

6. Wait for the input to be created and tested.
7. In the **Inputs** blade, click **+ Add**.
8. In the **New input** blade, enter the following settings to define an input for the patient profile data in Azure Storage, and then click **Create**:
    - **Input alias**: PatientData
    - **Source Type**: Reference data
    - **Import option:** Use blob storage from current subscription
    - **Storage account**: *Select your storage account*
    - **Storage account key:** *This will be selected automatically*
    - **Container**: bigdata
    - **Path pattern:** device_users/profiles.csv
    - **Date format**: *Not enabled*
    - **Time format**: *Not enabled*
    - **Event serialization format:** CSV
    - **Delimiter**: comma (,)
    - **Encoding:** UTF-8
9. Wait for the input to be created and tested.
10. Close the **Inputs** blade, and in the blade for your Stream Analytics job, in the **Job Topology** section, click the **Outputs** tile.
11. In the **Outputs** blade, click **+ Add**.
12. In the **New output** blade, enter the following settings to create an output for the **dbo.Calories** Azure SQL database table, and then click **Create**:
    - **Output alias**: CaloriePredictions
    - **Sink**: SQL database
    - **Import option:** Use SQL database from current subscription
    - **Database**: DiabetesData
    - **Server name:** *Your server will be selected automatically*
    - **Username:** *Your SQL login*
    - **Password:** *Your SQL login password*
    - **Table:** dbo.Calories
13. Wait for the output to be created and tested. Then close the **Outputs** blade.

Create a Function for the Azure Machine Learning Web Service

The Azure Stream Analytics job will use a function to call the **CaloriePredictor** web service and get the predicted calorie burn amounts.

1. In the blade for your Stream Analytics job, click **Functions**.
2. On the **Functions** blade, click **+ Add**.
3. In the **New function** blade, enter the following details and click **Create**:
    - **Function Alias**: PredictCalories
    - **Function Type**: Azure ML

- **Import option**: Select from the same subscription
- **URL**: CaloriePredictor
- **Key**: *This will be selected automatically*

Create a Query

The Azure Stream Analytics job will run a perpetual query that reads the stream of device data from the event hub, joins it to the patient profile data, calls the **PredictCalories** function to get calorie predictions from your Azure Machine Learning **CaloriePredictor** web service, and writes the results to Azure SQL Database.

1. In the blade for your Stream Analytics job, click **Overview**. Then click the **Query** tile.
2. In the blank query window, enter the following code, which you can copy and paste from **StreamQuery.txt** in the **Lab04** folder:

```
WITH [exercise] AS (
    SELECT userid,
           DateAdd(minute ,-1,System.TimeStamp) AS WindowStart,
           System.TimeStamp AS WindowEnd,
           AVG(CAST(heartrate AS float)) AS AvgHeartRate,
           AVG(CAST(bodytemp AS float)) AS AvgBodyTemp
    FROM [DeviceData] TIMESTAMP BY eventtime
    GROUP BY userid, TumblingWindow(minute, 1)
)

SELECT pat.PatientID,
ex.WindowStart,        ex.WindowEnd,
       DATEDIFF(minute, ex.WindowStart, ex.WindowEnd) AS Duration,
ex.AvgHeartRate,        ex.AvgBodyTemp,
       PredictCalories(pat.PatientID,
pat.Gender,                        pat.Age,
pat.Height,
pat.Weight,
                       DATEDIFF(minute, ex.WindowStart, ex.WindowEnd),
ex.AvgHeartRate,
                       ex.AvgBodyTemp) AS CaloriesBurned
INTO [CaloriePredictions]
FROM [exercise] AS ex
JOIN [PatientData] AS pat ON ex.userid = pat.UserID
```

Note: This query creates a common table expression called **exercise** that defines a one-minute temporal window and includes the average heart rate and body temperature for each user in that period. The rest of the query joins the aggregated exercise data to the patient profile data to retrieve the patient ID and the features used by the **CaloriePredictor** machine learning web service, and then uses the **PredictCalories** function to call the web service and retrieve the predicted calorie burn amount for the period for each user. The results are sent to the output for the **dbo.Calories** table in Azure SQL Database.

3. Click **Save** to save the query, and click **Yes** when prompted to confirm the changes. Then close the query blade.

Simulate Patient Exercise

Now you are ready to run the streaming query and simulate patient exercise.

1. In the blade for your Stream Analytics job, click **Start**. Then on the **Start job** blade, verify that **Now** is selected and click **Start**.
2. Wait a minute or so for the notification that the job has started, and verify that the status is shown as **Running**.
3. In the command line console, which should be open in the **fitness_device** folder, enter the following command to run the exercise simulation:

   ```
   node exercise
   ```

4. Observe that the simulated device readings are displayed in the console window – these are the readings that will be submitted to your event hub.
5. Wait for five minutes or so to allow the streaming query to capture and process a reasonable volume of real-time data.
6. With the blade for your Stream Analytics job open, refresh the browser page until you can see some input and output events in the **Monitoring** graph.
7. In the blade for your Stream Analytics job, click **Stop** and click **Yes** to confirm you want to stop the job.
8. In the console window, type CTRL+C to stop the script.
9. In the Azure portal, browse to the blade for the **DiabetesData** database, and then on the **Tools** menu, click **Query editor** to open the web-based query interface for your database.
10. In the **Query editor** blade, click **Login** and then log in to your Azure SQL database server using **SQL server authentication** with the username and password you specified when provisioning the database.
11. After you have been authenticated, enter the following query to retrieve the captured calorie predictions:

    ```
    SELECT * FROM dbo.Calories;
    ```

12. Click **Run**, and then wait for the query to complete. Then review the results in the bottom pane. These include the **CaloriesBurned** value predicted by your Azure Machine Learning web service for each one-minute window.
13. Modify the query to retrieve the average-per-minute and total calories predicted for each patient as shown here:

    ```
    SELECT PatientID,
           DATEDIFF(minute, MIN(WindowStart), MAX(WindowEnd)) AS Duration,
           AVG(CaloriesBurned) AS AvgCalories,
           SUM(CaloriesBurned) AS SumCalories
    FROM dbo.Calories
    GROUP BY PatientID
    ```

14. Click **Run**, and then wait for the query to complete. Then review the results in the bottom pane. These show the total number of minutes of exercise, the average calories per one-minute window, and the total calories burned for each user.
15. After you have been authenticated, enter the following query to delete the captured calorie predictions, resetting the table for the next exercise period:

    ```
    TRUNCATE TABLE dbo.Calories;
    ```

16. Click **Run**, and then wait for the query to complete. Then close the query editor blade, discarding the changes to your script.

Cleaning Up

You have now completed all the labs in this course. Hopefully you've learned a great deal about using Azure Machine Learning to create predictive web services, and using them in big data processing solutions.

You may continue to experiment with the Azure resources you have created, but be aware that they will continue to incur charges against your subscription (or use up your free credits if you are using a trial or free account).

When you no longer require the resources used in this course, use the following steps to delete them:

1. In the Azure portal, view the Resource Groups blade.
2. Click the ellipses (…) for the resource group in which you created the resources for these labs, and then click Delete.
3. When prompted, enter the name of the resource group, and click **Delete**.