## Azure Logic Apps

### James Adair, Deep Azure@McKesson

## **Problem Statement**

Crude oil and natural gas continue to play dominant roles in global and regional energy markets. As with any trade commodity, prices are influenced by supply and demand but that only scratches the surface of the detailed factors that determine the price. An analysis of crude oil and natural gas prices and their possible influences on one each other can serve as a suitable study for the novice energy market investor. This project provides a streamlined means for capturing time series energy data from the U.S. Energy Information Administration (EIA) on a frequency of choice with Azure Logic Apps. We’ll start an exploration of the price data, aligning what we can visualize with assertions that have been put forth by industry experts.

## **Technology Overview**

Azure Logic Apps is an integration platform as a service (iPaaS) that supports the development and execution of simple to complex workflow applications in the cloud. Providing an easy-to-use visual designer, workflow applications are constructed with a wide variety of Microsoft, 3rd party and custom-built connectors. Running Logic Apps is aided with all the benefits of the Azure serverless cloud ecosystem.

## **Technical Solution**

Hardware: Windows 7 64-bit, Dell Precision 7710, Intel Core i7, 16 GB Ram

Software: Azure Logic Apps, Azure Function Apps, Microsoft SQL Server, RStudio

Datasets: <http://api.eia.gov/series/?api_key=YOUR_API_KEY_HERE&series_id=PET.RBRTE.D>

<http://api.eia.gov/series/?api_key=YOUR_API_KEY_HERE&series_id=NG.RNGWHHD.D>

Methodology: Develop Logics Apps that automate the process of (1) capturing energy data once a day (2) transforming JSON representation for easier consumption and (3) importing data into SQL server database. Use RStudio to retrieve data for visualization and analysis.

Lessons: You can cobble together Logic Apps with relative ease but consider all the design, cost and benefit alternatives for achieving the same results. Applying correlation analysis to time series data is a dubious task, beware the trend!

YouTube: 2 Min: <https://youtu.be/9aivyGOm4HY>

15 Min: <https://youtu.be/b_1saRulwvU>

GitHub: <https://github.com/jamespadair/deep_azure_final>

# **Introduction**

Azure Logic Apps, our workflow engine in the cloud, provides building blocks to (1) initiate the application instance with a trigger and (2) specify the step-wise actions to produce the desired workflow application outcome. It is important to realize that actions in Logic Apps take on the form of connectors. These connectors expose APIs to access Azure provisioned services, third party services or those that you create on your own.

The project starts with an explanation of petroleum and natural gas price datasets and how they are obtained using the Energy Information Administration Open Data API. Our application will start with a schedule trigger that will retrieve the time series dataset via the API daily.

We will then create a SQL Server database and a special Azure Function to transform JSON. These are both dependencies we will use in creating the Azure Logic App workflow application.

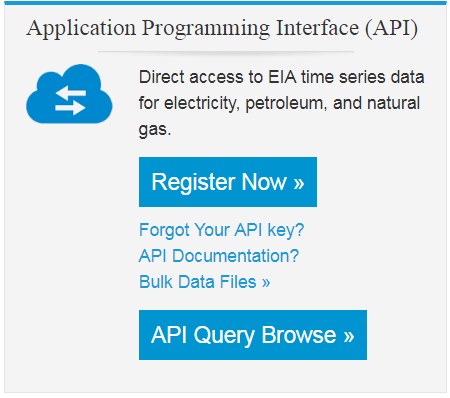
Next, we walk through the creation of our Logic App to illustrate the construction process and the means for tracking its use.

Lastly, we explore dataset visualizations in RStudio.

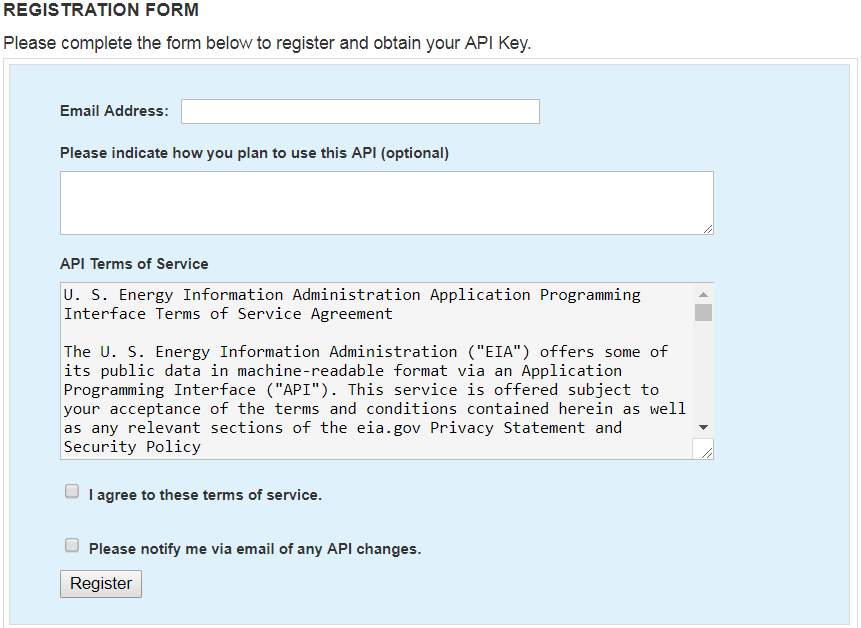
# **EIA Data Source**

Both the Europe Brent Petroleum Spot Price and Henry Hub Natural Gas Spot Price datasets used in this demonstration are accessible via the Open Data API hosted on the U.S. Energy Information site. The EIA API is offered as a free service but requires an API key acquired through a registration process.

1. Get your own API key
2. Open the webpage at <https://www.eia.gov/opendata/>
3. Click on the ‘Register Now’ button found on the right panel



1. Complete the Registration Form and click on the ‘Register’ button. You should expect to receive an API key submitted to the email you provided in the registration.



1. This API key will be required on every API call you make on the EIA site so make note of it for future reference.

EIA API key: xxxxxxxxxxxxxxxxx

The EIA provides a very wide array of datasets for access that you can perused by clicking on the ‘API Query Browse’ button on the Open Data webpage <https://www.eia.gov/opendata/>.

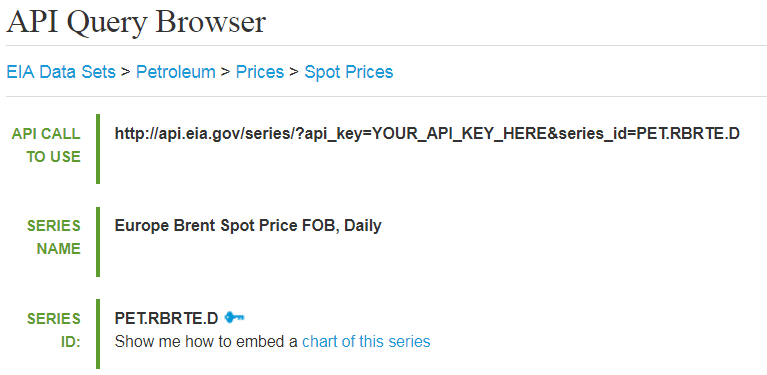
Click through the energy data of interest, for this demo:

Petroleum🡪Prices🡪Spot Prices🡪Europe Brent Spot Price FOB, Daily

You will land on the page which provides the details for making an API call to get the data, which is:

<http://api.eia.gov/series/?api_key=YOUR_API_KEY_HERE&series_id=PET.RBRTE.D>

<http://api.eia.gov/series/?api_key=YOUR_API_KEY_HERE&series_id=NG.RNGWHHD.D>

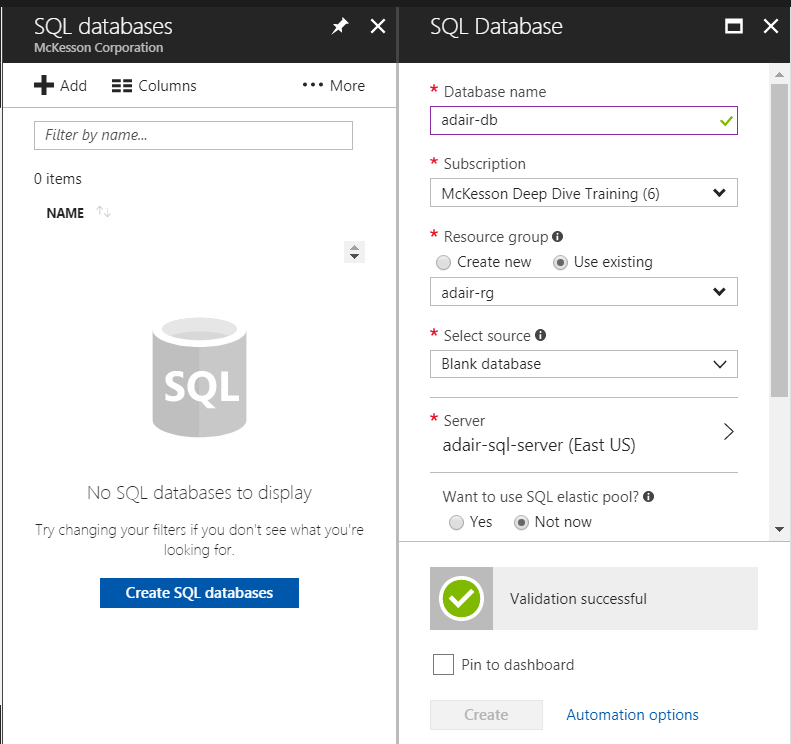


We now turn our attention to the Azure Portal where we will (1) create a SQL Server database, (2) create a helper function to transform JSON payloads and (3) create our Logics App.

# **Provision SQL Server in the Cloud**

Create a SQL Server database in the Azure Portal. We will connect to this database in the ensuing Logic App workflow.

1. Click through from the services list, ‘SQL databases’🡪 +Add
2. Provide database name, a new resource group and change pricing tier to ‘Basic’ on the ‘SQL Database’ blade.
3. Configure a new server providing a server name and credentials on the ‘Server’ and ‘New Server’ blades.



1. Make note of your resource identities and credentials for future reference, for example:

Resource group: adair-rg

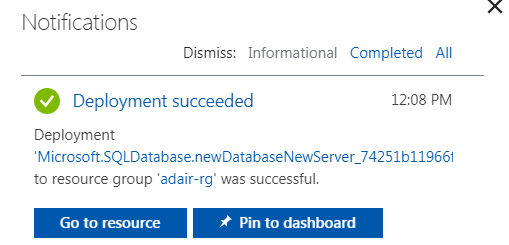
Server name: adair-sql-server.database.windows.net

Database name: adair-db

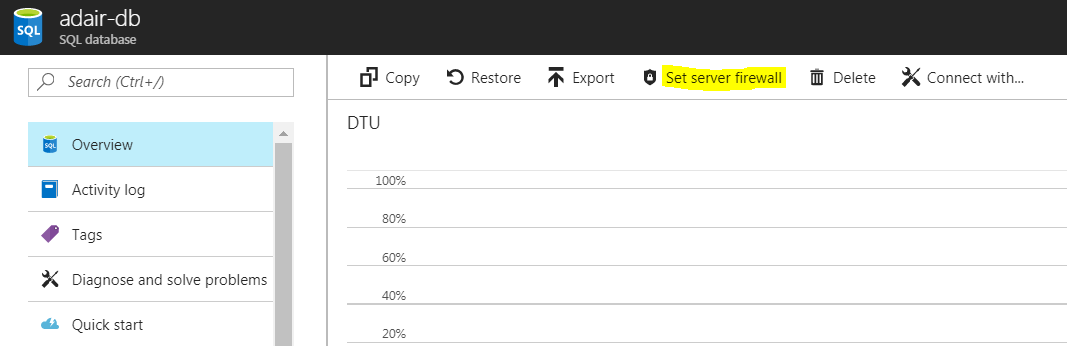
Admin login: jamespadair

Admin password: xxxxxxxx

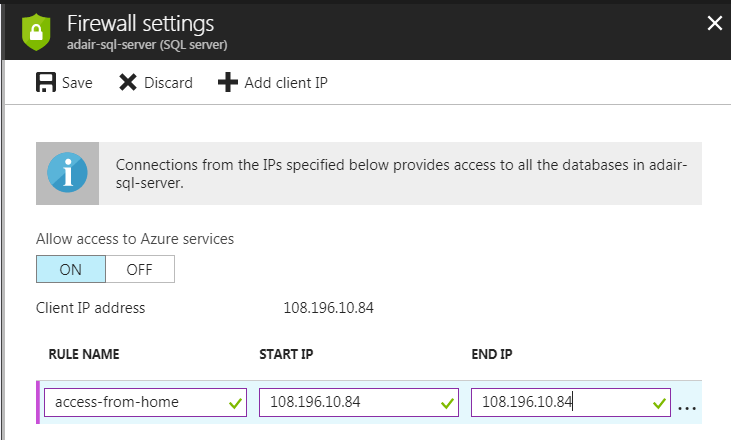
1. Click ‘Create’ to create and deploy the new server and database.



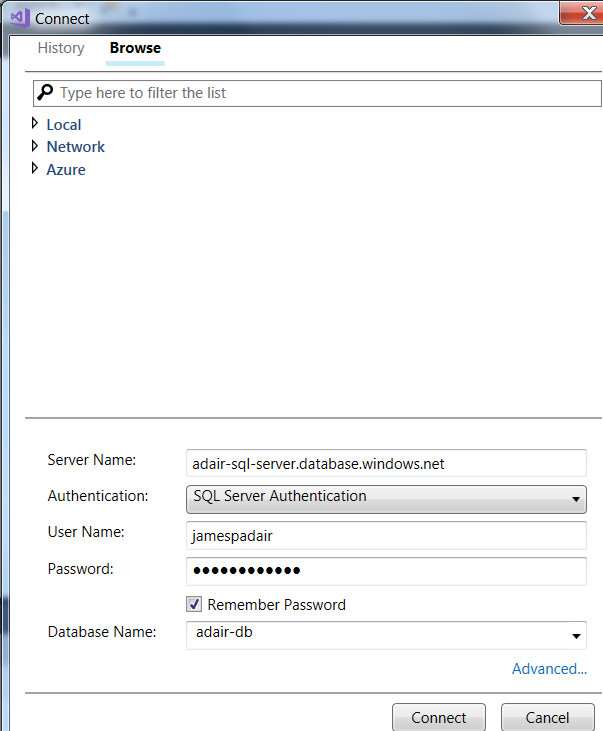
1. Open the SQL Server resource by clicking through from the services list, ‘SQL databases’🡪 ‘adair-sql-server’ and select ‘Set server firewall’ from the summary menu.



1. Create a firewall rule, allowing access from the Client IP and click save. For example, the reported client IP is the ATT provider WAN address I connect to the internet from home.



1. Open Visual Studio 2017, open a New Query window from the Tools🡪SQL Server menu. Connect to the online database.



1. Create table BrentPetroleumPrices as:

USE [adair-db]

GO

-------------------------------------------------------

IF EXISTS (SELECT \* FROM sys.OBJECTS WHERE OBJECT\_ID = OBJECT\_ID(N'dbo.BrentPetroleumPrices') AND TYPE IN (N'U'))

BEGIN

DROP TABLE dbo.BrentPetroleumPrices

END

GO

CREATE TABLE dbo.BrentPetroleumPrices

(

[Day] INT NOT NULL PRIMARY KEY,

[Price] MONEY NOT NULL

)

GO

1. Create stored procedure uspBatchInsertBrentPetroPrices as:

USE [adair-db]

GO

set ANSI\_NULLS ON

GO

set QUOTED\_IDENTIFIER ON

GO

-------------------------------------------------------

IF EXISTS (SELECT \* FROM sys.OBJECTS WHERE OBJECT\_ID OBJECT\_ID(N'dbo.uspBatchInsertBrentPetroPrices') AND TYPE IN (N'P'))

BEGIN

DROP PROCEDURE dbo.uspBatchInsertBrentPetroPrices

END

GO

CREATE PROCEDURE dbo.uspBatchInsertBrentPetroPrices

@jsonObject nvarchar(max)

AS

BEGIN

set NOCOUNT ON;

INSERT INTO dbo.brentPetroleumPrices([day],price)

SELECT [day],price

FROM OPENJSON (@jsonObject, '$.data')

WITH (

day integer '$.day',

price money '$.val'

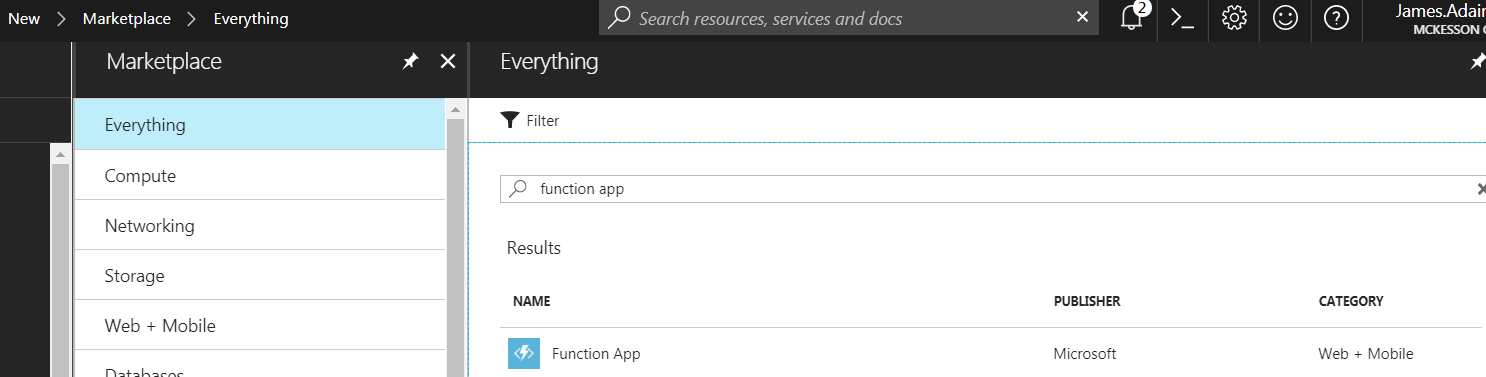
)

END

GO

# **Provision a JSON transformation Function App in the Cloud**

1. Click ‘New’ and enter search for ‘function app’. Select ‘Function App’ from the list.



1. Click ‘Create’ on the Function App blade and enter an app name, the desired resource group, location and allow to create new storage account. For example:

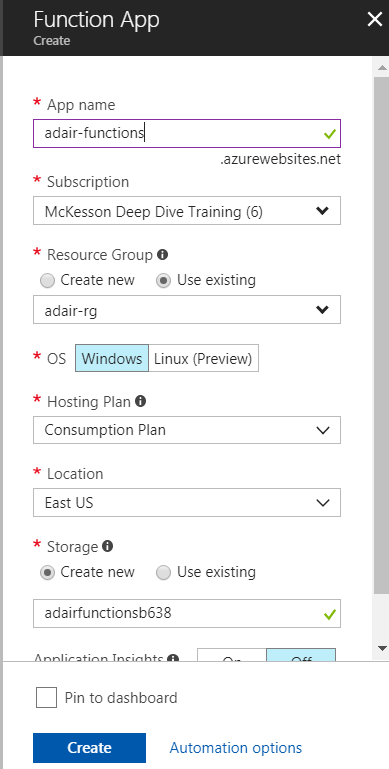
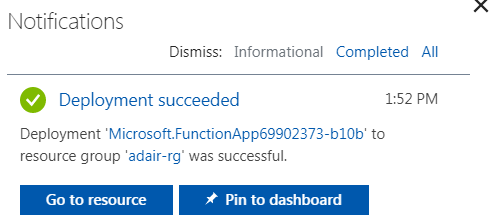
App name: adair-functions

Existing resource group: adair-rg

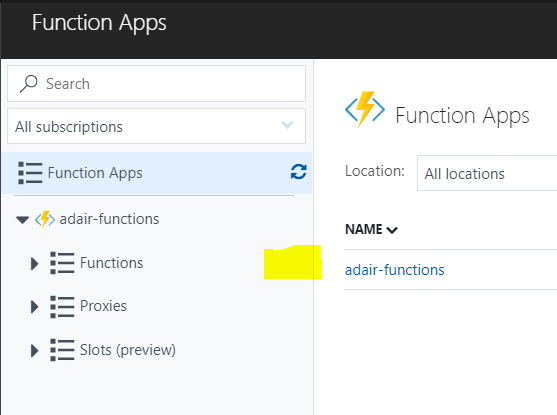
Location: east-us

Storage: adairfunctionsb638

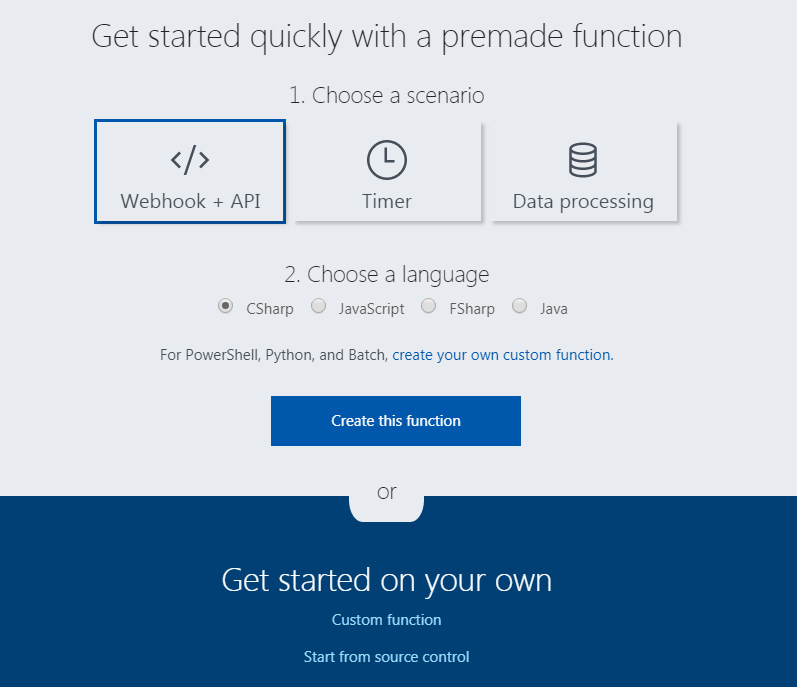
url: adair-functions.azurewebsites.net

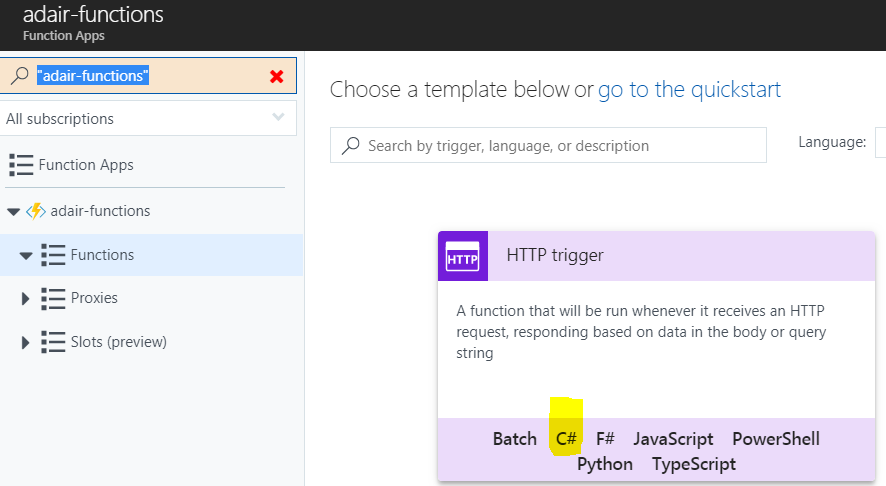
1. Click through from the services list, ‘Function Apps’🡪 expand the ‘adair-functions’ list and hold your mouse over the ‘Functions’ item to see the ‘+’



1. Click ‘+’ and you will see the following screen

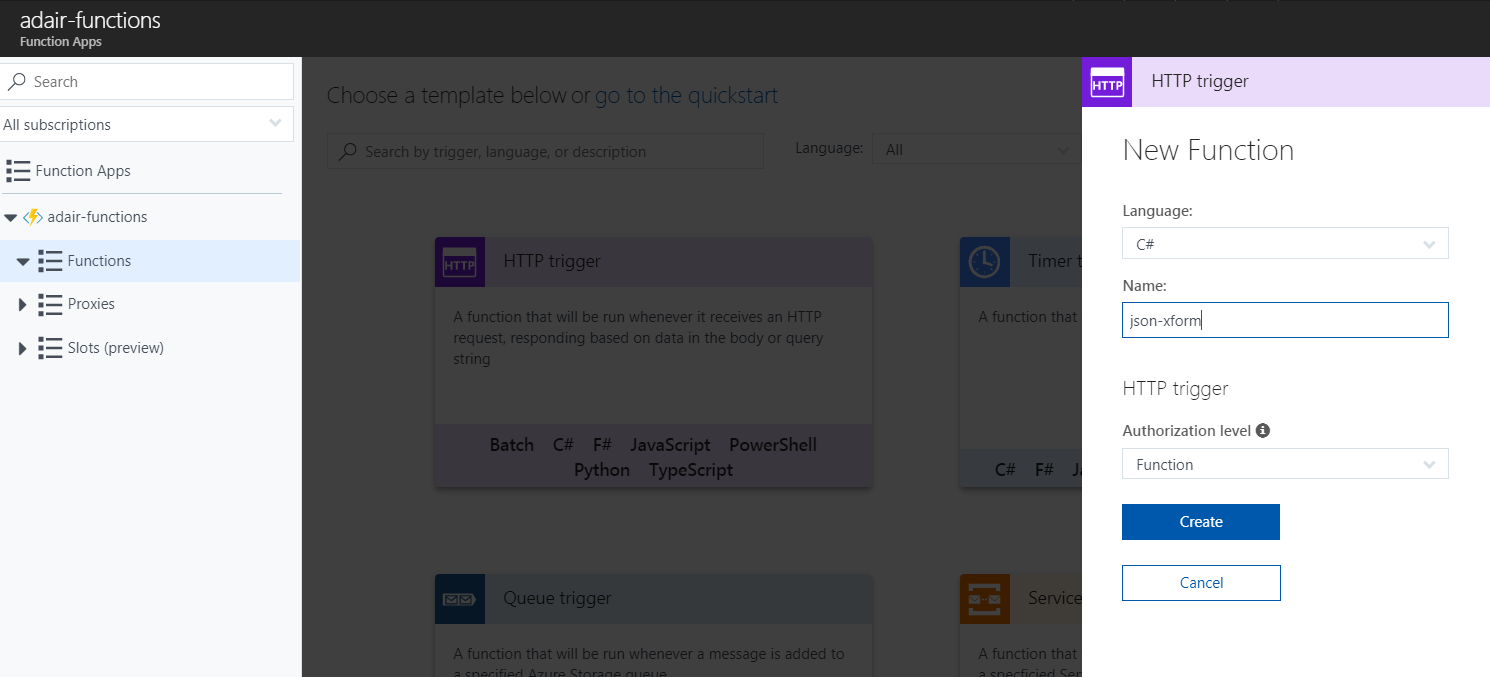


1. Click ‘Custom function’

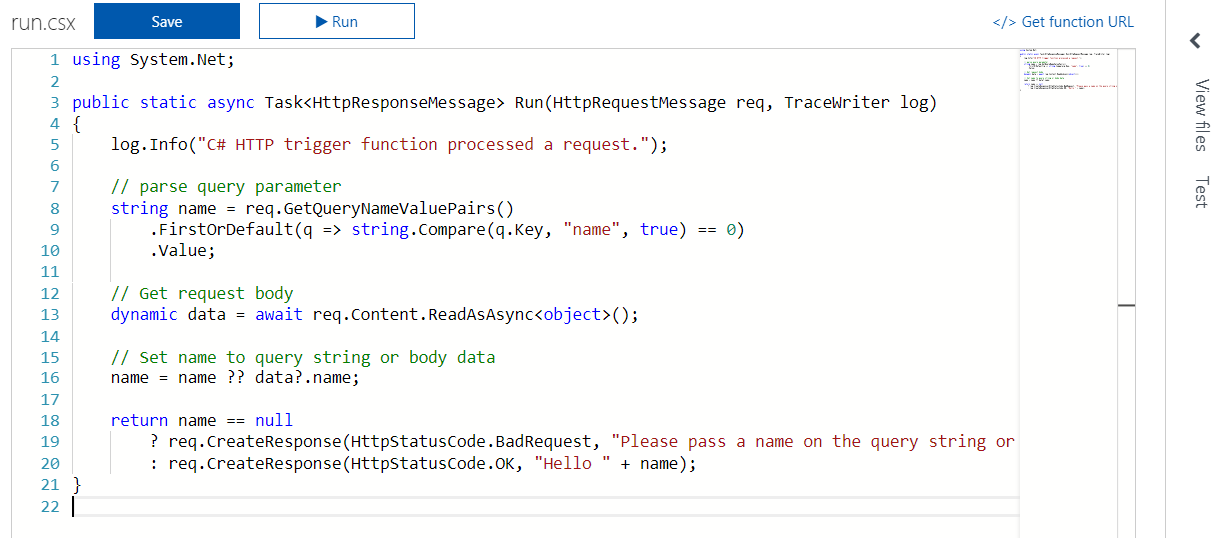


1. Select ‘C#’ in the ‘HTTP trigger’ template and enter function name followed by ‘Create’.

For example, I provided function name: json-xform



1. you will be presented with a run.csx code view containing sample C# template code.



Replace the full content of this code with the following code below, press ‘Save’, then ‘Run’ then close the view.

#r "Newtonsoft.Json"

using System.Net;

using System.Text;

using Newtonsoft.Json;

public static async Task<HttpResponseMessage> Run(HttpRequestMessage req, TraceWriter log)

{

dynamic body = await req.Content.ReadAsStringAsync();

//convert the incoming energy json to EnergyData object

var energyIn = JsonConvert.DeserializeObject<EnergyData>(body as string);

//create outgoing transformation, EnergyDataX

var energyOut = new EnergyDataX();

//preserve the metadata

energyOut.series\_id = energyIn.series[0].series\_id;

energyOut.name = energyIn.series[0].name;

energyOut.units = energyIn.series[0].units;

energyOut.f = energyIn.series[0].f;

energyOut.unitsshort = energyIn.series[0].unitsshort;

energyOut.description = energyIn.series[0].description;

energyOut.copyright = energyIn.series[0].copyright;

energyOut.source = energyIn.series[0].source;

energyOut.start = energyIn.series[0].start;

energyOut.end = energyIn.series[0].end;

energyOut.updated = energyIn.series[0].updated;

// Transform the data array.

// [[x,y]...[x,y]] to [{day=x,val=y}...{day=x,val=y}]

int i=0;

Int32 numDataElems = energyIn.series[0].data.Length;

log.Verbose("len="+ numDataElems.ToString());

energyOut.data = new DataX[numDataElems];

foreach (object[] elem in energyIn.series[0].data)

{

var dataX = new DataX() { day=Convert.ToInt32(elem[0]), val=Convert.ToDouble(elem[1]) };

energyOut.data[i++] = dataX;

}

var jsonOutput = JsonConvert.SerializeObject(energyOut);

return req.CreateResponse(HttpStatusCode.OK, jsonOutput);

}

//incoming object graph

public class Request

{

public string command { get; set; }

public string series\_id{ get; set; }

}

public class Series

{

public string series\_id {get; set;}

public string name {get; set;}

public string units {get; set;}

public string f {get; set;}

public string unitsshort {get; set;}

public string description {get; set;}

public string copyright {get; set;}

public string source {get; set;}

public string start {get; set;}

public string end {get; set;}

public string updated {get; set;}

public object [][] data {get; set;}

}

public class EnergyData

{

public Request request { get; set; }

public Series[] series { get; set; }

}

//transformed outgoing data graph

public class DataX

{

public Int32 day {get; set;}

public Double val {get; set;}

}

public class EnergyDataX

{

public string series\_id {get; set;}

public string name {get; set;}

public string units {get; set;}

public string f {get; set;}

public string unitsshort {get; set;}

public string description {get; set;}

public string copyright {get; set;}

public string source {get; set;}

public string start {get; set;}

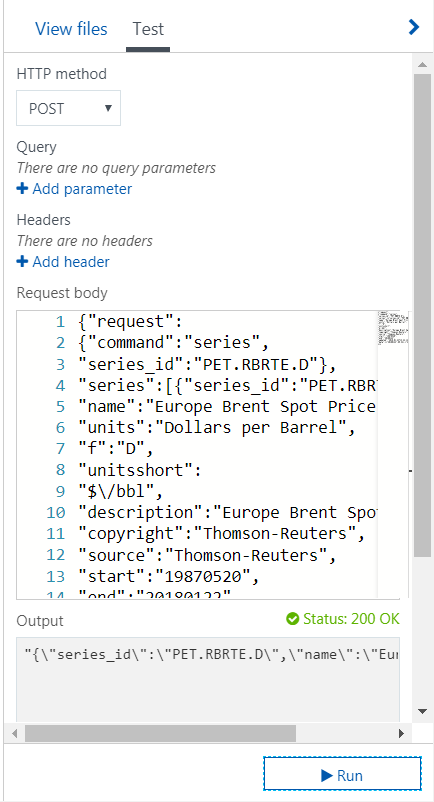
public string end {get; set;}

public string updated {get; set;}

public DataX[] data {get; set;}

}

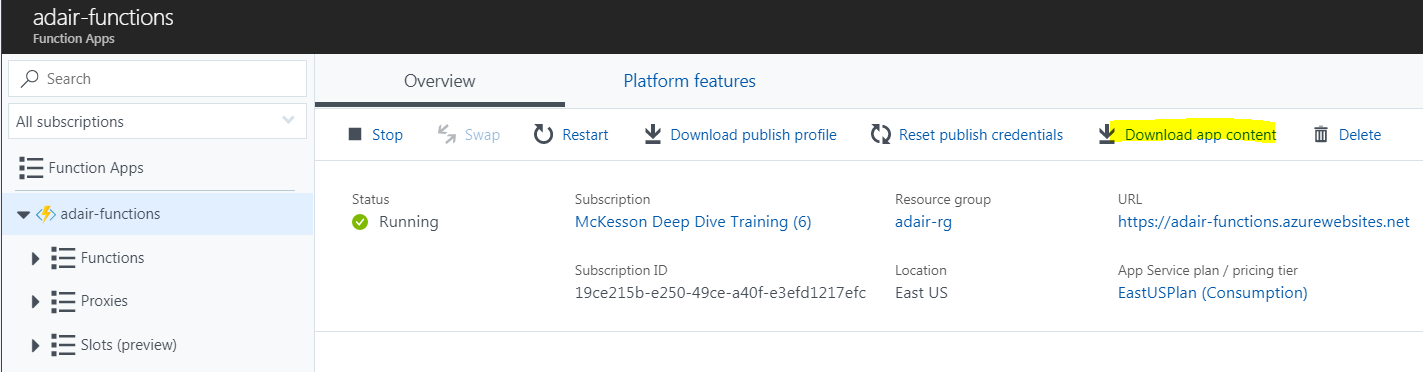
1. Run a test, providing a sample of the resulting dataset JSON in the Request body and clicking ‘Run’. The Output displays ‘Status:200 OK’ and the resulting JSON transformation.

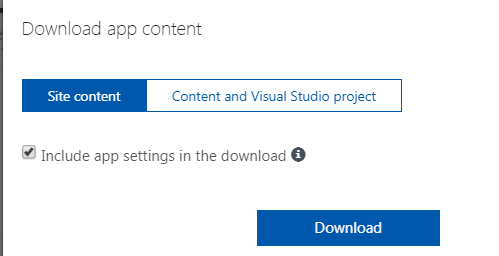


1. On attempt to close the view, you might be prompted with a message that you have unsaved changes but if you’ve saved and run the application you can proceed with the closure. Now navigate to your azure function view again and you will find that it is running.

Click through from the services list, ‘Function Apps’🡪 ‘adair-functions’

1. Click on ‘Download app content’





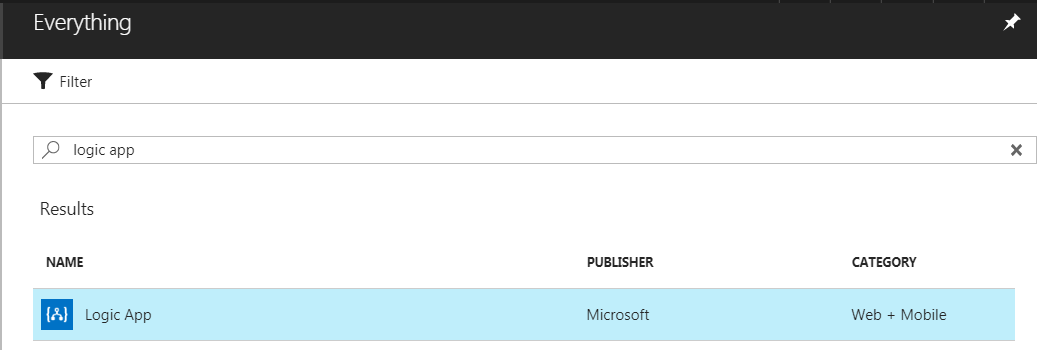
Save the downloaded zip file to a convenient location for future reference.

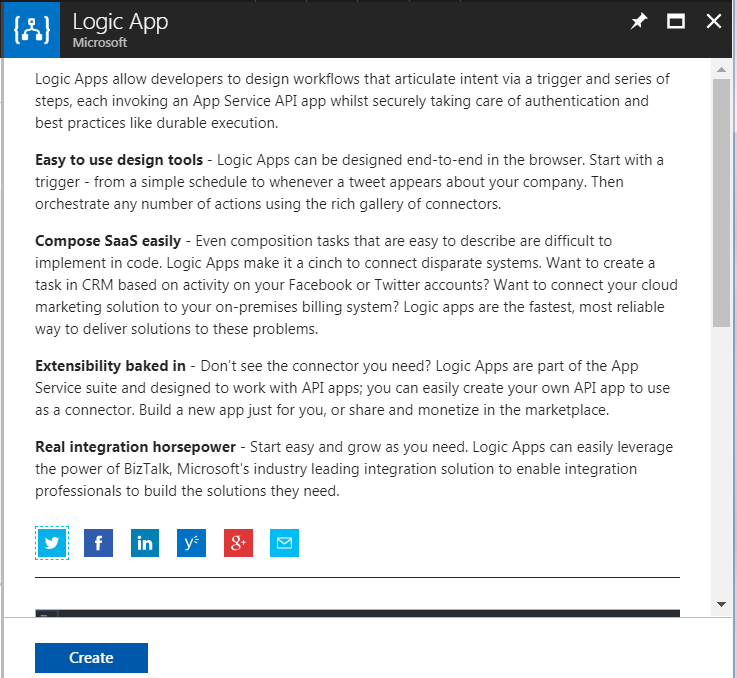
For example: adair-functions.zip

# **Create a workflow application in the cloud with Logic Apps**

Here is the creation of our Logic App, ‘adair-logic-app’, an application that runs as a serverless application in the Azure cloud ecosystem once a day to capture our data, parse it, import it into SQL server for visualization in RStudio.

1. Click ‘New’ and enter search for ‘logic app’. Select ‘Logic App’ from the list.



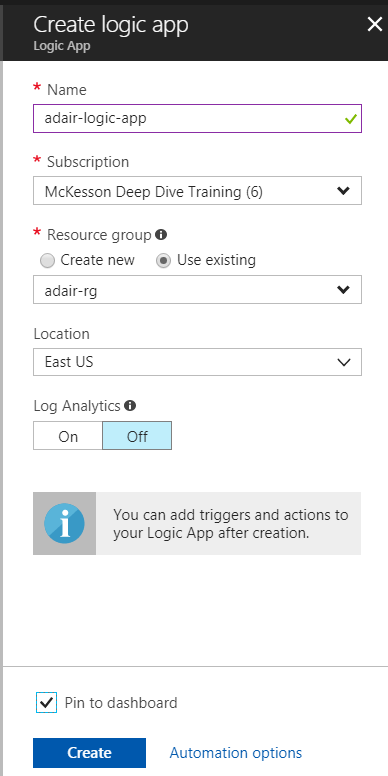


1. Enter name, resource group and click ‘Create’.

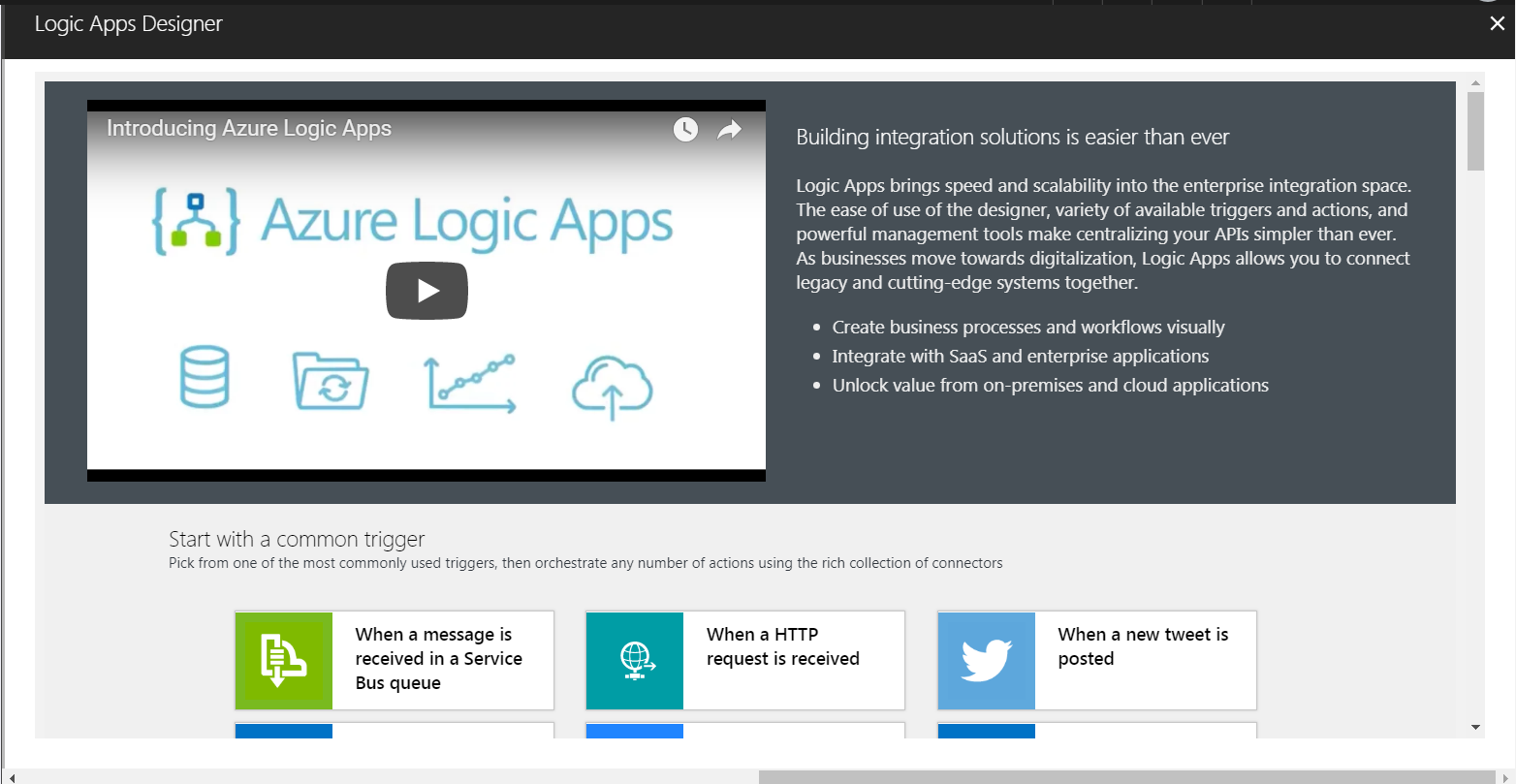
For example,

Name: adair-logic-app

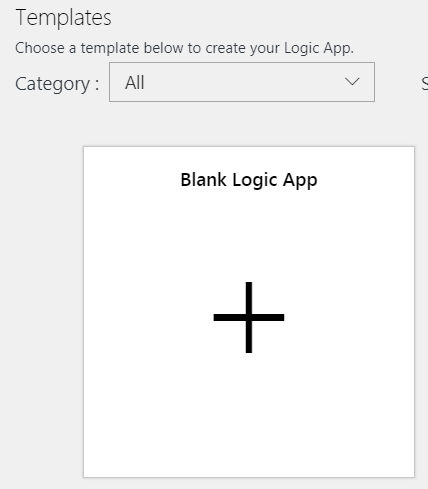
Resource group: adair-rg



1. You will be presented with the Logics Apps Designer blade



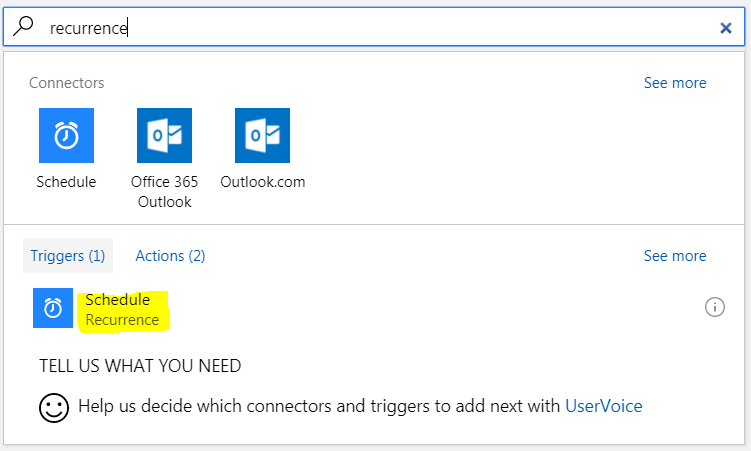
1. Here is where you get a sampling of the various ways you can create your logic application as provided in a gallery of connectors which are trigger and action templates. Scroll down the view and select the ‘Blank Logic App’ template.



1. Again, a presentation of more gallery options.

Search for ‘recurrence’ and select ‘Schedule Recurrence’ under the Triggers section.

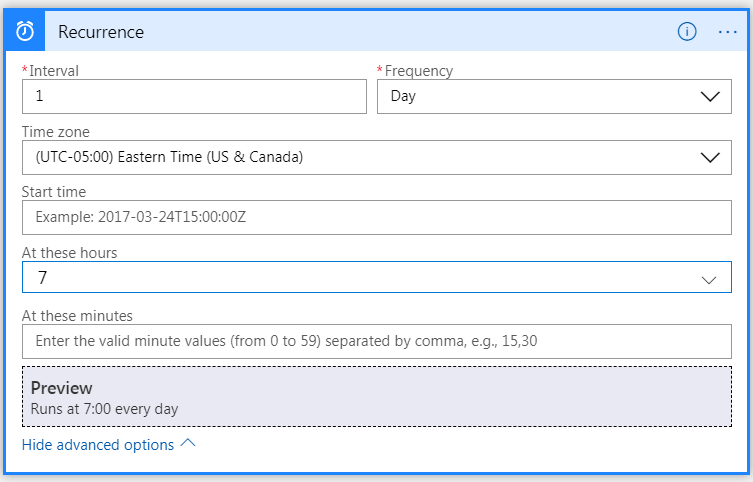
WORFLOW STEP 1, TRIGGER – a time clock event to start our application, once a day



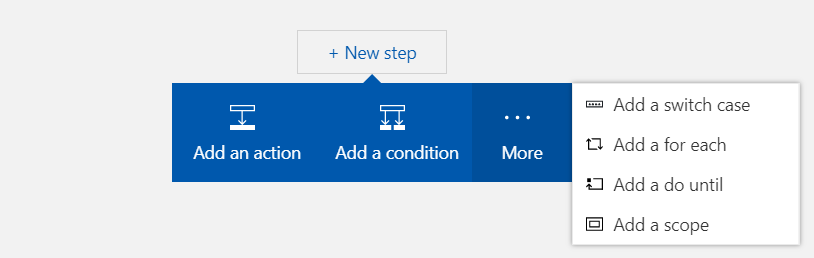
1. We are effectively triggering the invocation of our application instance with a time schedule event.

For example: runs at 7:00am every day

Where specification for interval, frequency and time zone are selected.

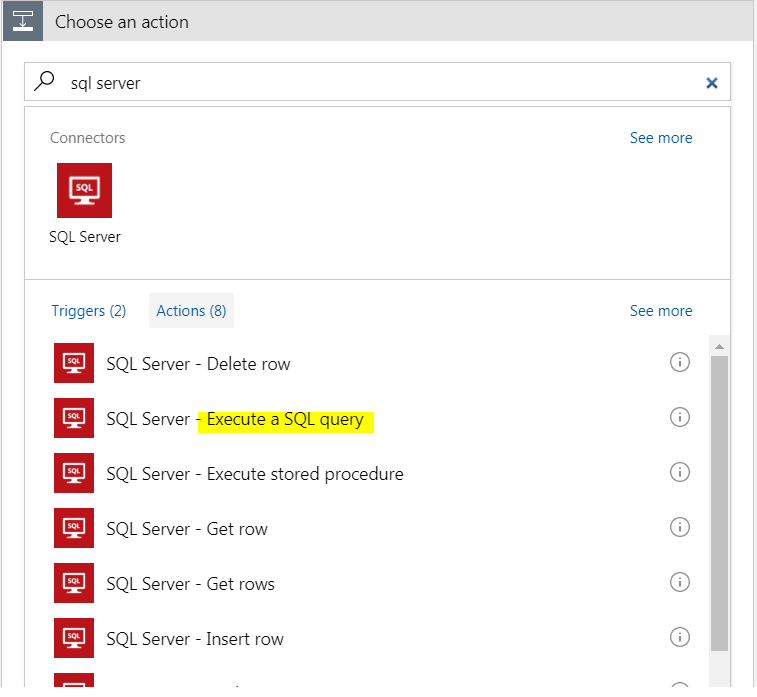


1. We now start adding actions by clicking ‘New step’.



1. Click ‘Add an action’ and search for ‘sql server’.

WORFLOW STEP 2, ACTION - query the database for the next day, to be used in the subsequent dataset API HTTP request. This aligns our request for data with the data we’ve already stored at any given time.

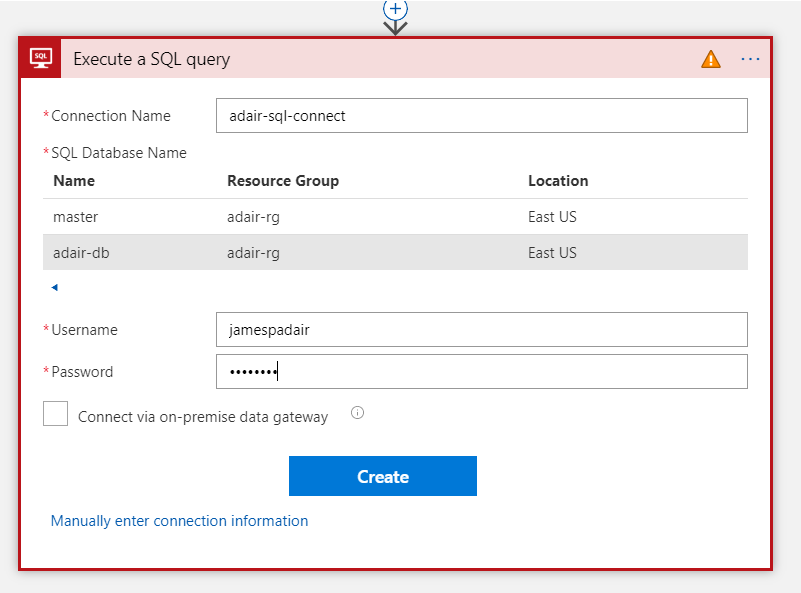


1. Provide the details to create a connection to the desired SQL server instance. For example,

Connection name: adair-sql-connect

SQL server: adair-sql-server

SQL database name: adair-db



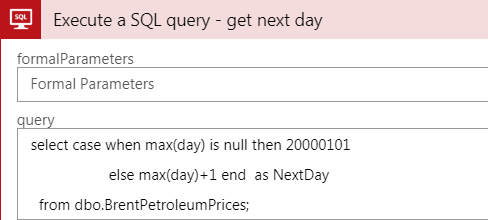
1. We are performing a query to determine what start date we should request from the dataset endpoint. When the database is initially empty, we will set the start date to 20000101, otherwise, the start date will be the current max day plus 1.

Enter the query and rename this step to reflect its purpose.

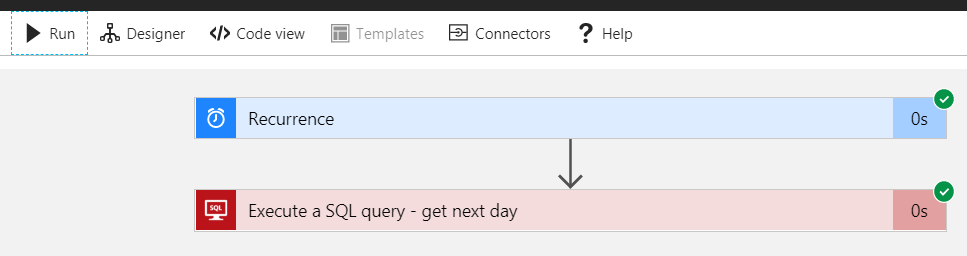
select case when max(day) is null then 20000101

else max(day)+1 end as NextDay

from dbo.BrentPetroleumPrices;



1. We can test steps 1 and 2 by first clicking ‘Save’ and then ‘Run’. The green check boxes indicate success. Return to the designer to continue.



1. Click ‘New step’, followed by ‘Add an action’ and search for ‘HTTP’. You will be presented with an ‘HTTP’ connector that you should select and then select the ‘HTTP HTTP’ action from the list.

WORKFLOW STEP 3, ACTION – invoke the dataset endpoint API call.



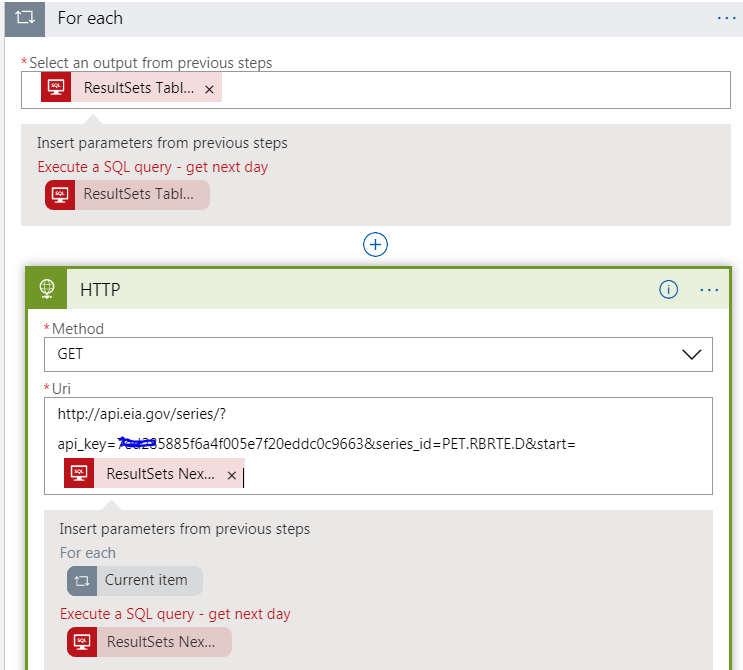
Adding an action in a Logic Apps workflow provides for capturing tokens from *any* previous action in the flow. At this new HTTP action, the engine senses that the previous action produces a result set and presents a ‘Dynamic content’ popup for us to select ‘ResultSets NextDay’. Because the previous action assumes a result set outcome and not the expected scalar result, it automatically injects a ‘For each’ conditional wrapper around our new HTTP action. This works for us where the result is perceived to be 1 row in a result set.

We provide the details of our HTTP call, the GET method and the Uri where we provide our EIA API key as well as a ‘start’ query parameter assigned the ‘ResultSets NextDay’ value from the previous step. ‘ResultSets Table’ and ‘ResultSets NextDay’ are tokens in the workflow designer.

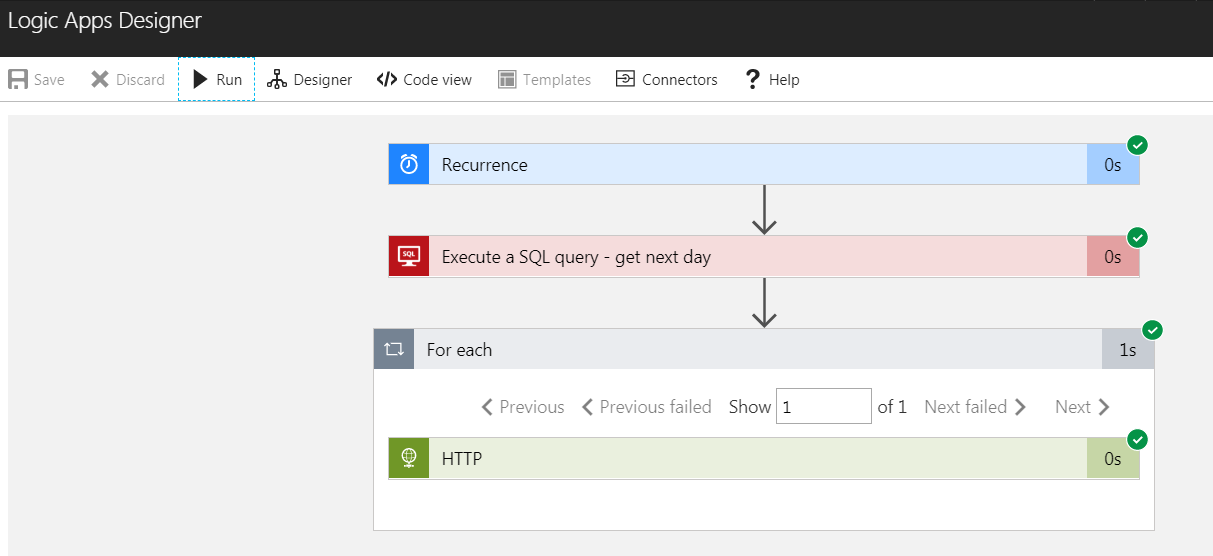
Behind the scenes the Uri is:

[http://api.eia.gov/series/?api\_key=[APIKEY]&series\_id=PET.RBRTE.D&start=@{items('For\_each')?['NextDay']}](http://api.eia.gov/series/?api_key=%5bAPIKEY%5d&series_id=PET.RBRTE.D&start=@%7bitems('For_each')?%5b'NextDay'%5d%7d)

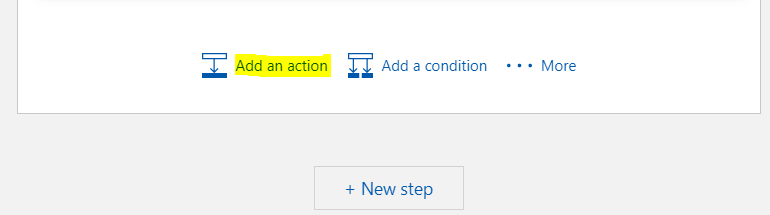
This shows the actual Logic Apps Workflow Definition Language representation of the token, an alternative that we can use instead of selecting tokens in the designer.



1. Let’s test steps 1-3, ‘Save’ and ‘Run’. Note the ‘Showing 1 of 1’, effectively 1 row is returned in the result set as we anticipated. Return to the designer to continue.



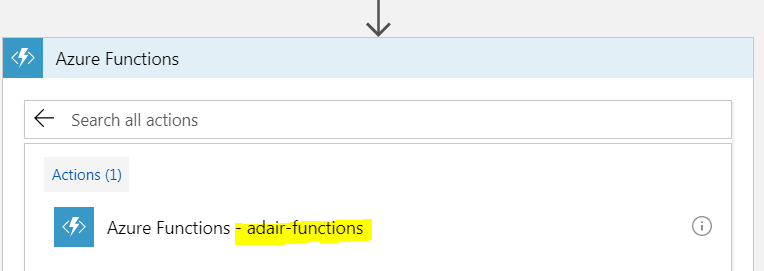
1. Now, instead of clicking ‘New step’, click ‘Add an action’ from within the ‘For each’ wrapper. This has the effect of maintaining the application scope instead of the ‘For each’ instead of continuing with steps outside of the ‘For each’.

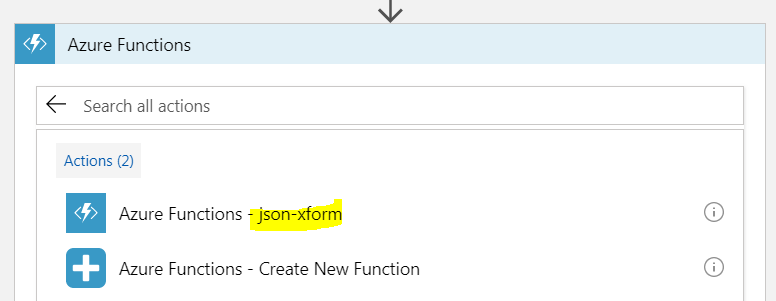


Search for ‘function’. You will be presented with an ‘Azure Function’ connector that you should select and then select the azure function created earlier in the tutorial.

WORKFLOW STEP 4, ACTION – transform the dataset JSON

For example: ‘Azure Functions – adair-functions’ followed by selection of the ‘json-xform’ action





1. In the ‘json-xform’ action, click the mouse pointer in the empty Request Body field and the options for inserting previous step tokens will be presented. Select ‘Body’ which is the dataset content returned in the HTTP response.

Here is a sample of the JSON returned by the dataset endpoint. Note the two-dimensional array representation of the ‘data’ value.

Exhibit 15-A

{"request":

{"command":"series",

"series\_id":"PET.RBRTE.D"},

"series":[{

"series\_id":"PET.RBRTE.D",

"name":"Europe Brent Spot Price FOB, Daily",

"units":"Dollars per Barrel",

"f":"D",

"unitsshort":

"$\/bbl",

"description":"Europe Brent Spot Price FOB",

"copyright":"Thomson-Reuters",

"source":"Thomson-Reuters",

"start":"19870520",

"end":"20180122",

"updated":"2018-01-24T13:22:12-0500",

"data":[["20180122",69.32],["20180119",68.56]]

}]

}

The transformation performed in the ‘json-xform’ action will remove the ‘request’ object, promote the and ‘series’ contents to root level fields and convert the two-dimensional ‘data’ array object to a single array as:

Exhibit 15-B

{

"series\_id":"PET.RBRTE.D",

"name":"Europe Brent Spot Price FOB, Daily",

"units":"Dollars per Barrel",

"f":"D",

"unitsshort":

"$\/bbl",

"description":"Europe Brent Spot Price FOB",

"copyright":"Thomson-Reuters",

"source":"Thomson-Reuters",

"start":"19870520",

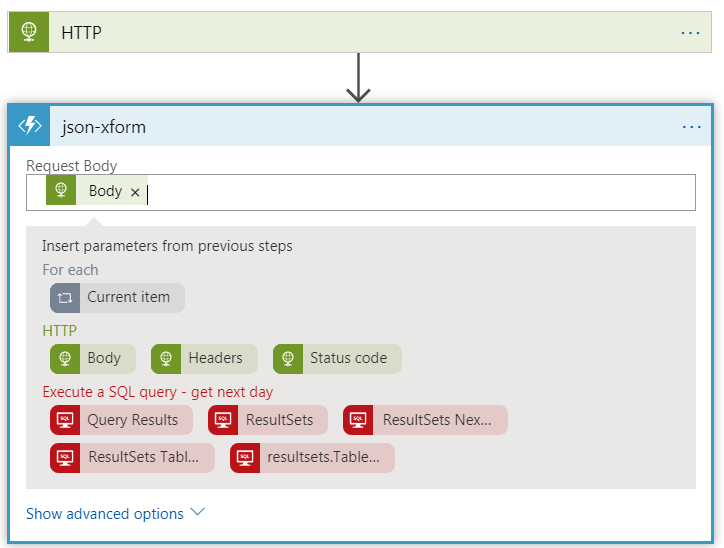
"end":"20180122",

"updated":"2018-01-24T13:22:12-0500",

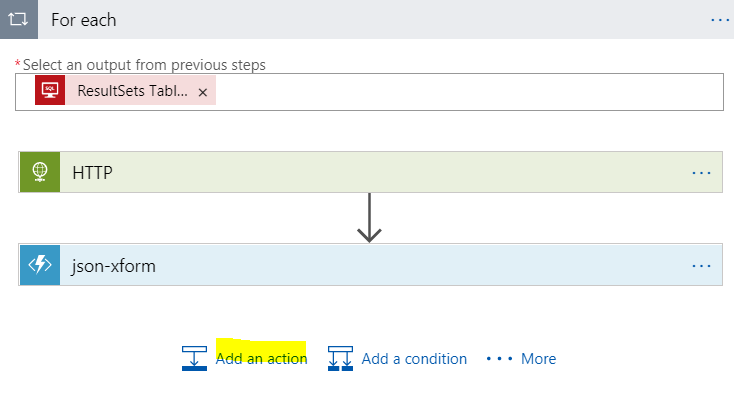
"data":[{"day":20180122,"val":69.32},{"day":20180119,"val":68.56}]

}

Here is a sample of the JSON returned by the dataset endpoint. Note the two-dimensional array representation of the ‘data’ value.

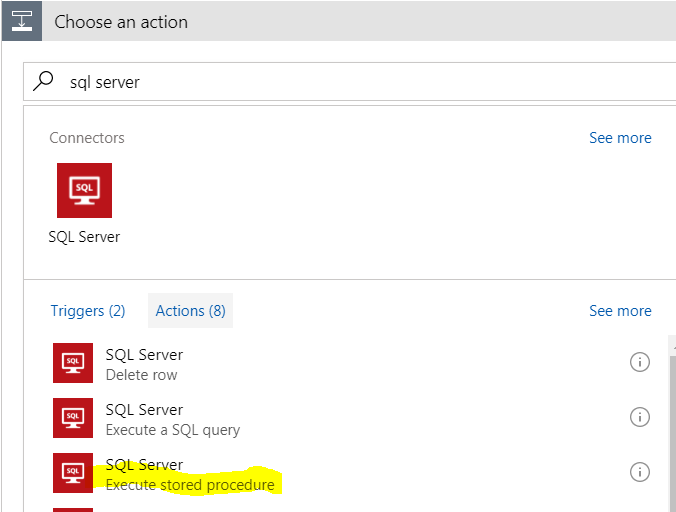


1. Click the ‘Add an action’, again from within the ‘For each’ wrapper.

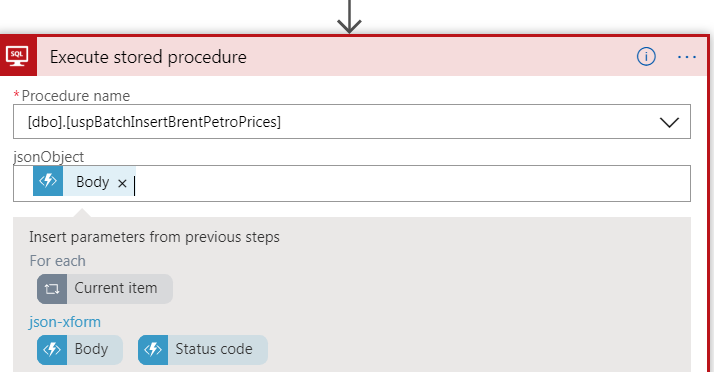


Search for ‘sql server’ and select ‘Execute stored procedure’

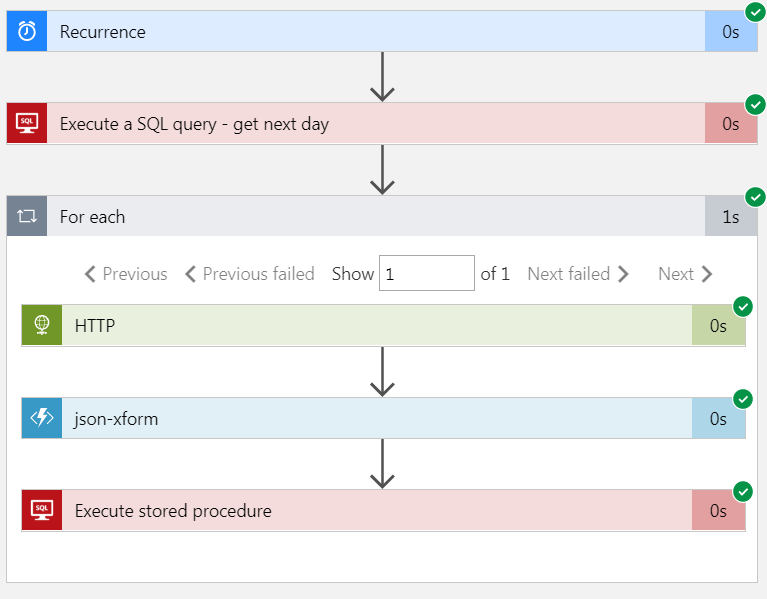
WORKFLOW STEP 5, ACTION – invoke call to stored proc passing in the json payload in order to insert the contents into the table.



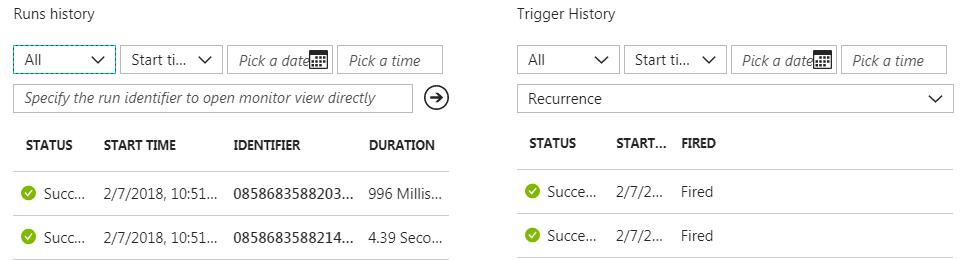
1. Select the procedure, uspBatchInsertBrentPetroPrices in the ‘Procedure Name’ field and select json-xform ‘Body’ in the ‘jsonObject’ field



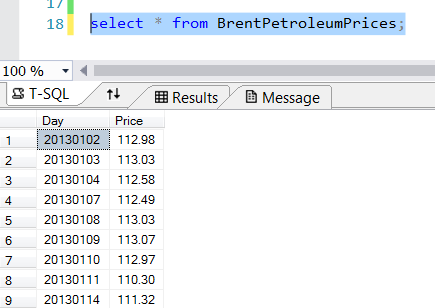
1. Let’s save and test all steps, ‘Save’ and ‘Run’ against an empty database table.

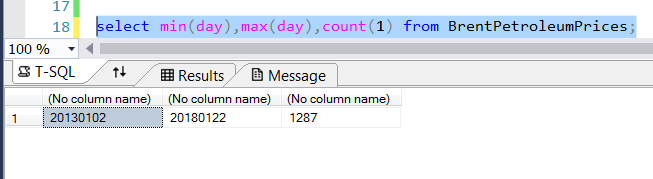


Checked status that status and elapsed time.



1. Let’s also query the table via SQL Studio or Visual Studio 2017 to see the imported data.





1. We can also the ‘Code View’, providing the JSON definition of the workflow.

{

"$connections": {

"value": {

"sql": {

"connectionId": "/subscriptions/19ce215b-e250-49ce-a40f-e3efd1217efc/resourceGroups/adair-rg/providers/Microsoft.Web/connections/sql-4",

"connectionName": "sql-4",

"id": "/subscriptions/19ce215b-e250-49ce-a40f-e3efd1217efc/providers/Microsoft.Web/locations/eastus/managedApis/sql"

}

}

},

"definition": {

"$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",

"actions": {

"Execute\_a\_SQL\_query\_-\_get\_next\_day": {

"inputs": {

"body": {

"query": "select case when max(day) is null then 20000101 \n else max(day)+1 end as NextDay\n from dbo.BrentPetroleumPrices;"

},

"host": {

"connection": {

"name": "@parameters('$connections')['sql']['connectionId']"

}

},

"method": "post",

"path": "/datasets/default/query/sql"

},

"runAfter": {},

"type": "ApiConnection"

},

"For\_each": {

"actions": {

"Execute\_stored\_procedure": {

"inputs": {

"body": {

"jsonObject": "@{body('json-xform')}"

},

"host": {

"connection": {

"name": "@parameters('$connections')['sql']['connectionId']"

}

},

"method": "post",

"path": "/datasets/default/procedures/@{encodeURIComponent(encodeURIComponent('[dbo].[uspBatchInsertBrentPetroPrices]'))}"

},

"runAfter": {

"json-xform": [

"Succeeded"

]

},

"type": "ApiConnection"

},

"HTTP": {

"inputs": {

"method": "GET",

"uri": "http://api.eia.gov/series/?api\_key=7ed285885f6a4f005e7f20eddc0c9663&series\_id=PET.RBRTE.D&start=@{items('For\_each')?['NextDay']}"

},

"runAfter": {},

"type": "Http"

},

"json-xform": {

"inputs": {

"body": "@body('HTTP')",

"function": {

"id": "/subscriptions/19ce215b-e250-49ce-a40f-e3efd1217efc/resourceGroups/adair-rg/providers/Microsoft.Web/sites/adair-functions/functions/json-xform"

}

},

"runAfter": {

"HTTP": [

"Succeeded"

]

},

"type": "Function"

}

},

"foreach": "@body('Execute\_a\_SQL\_query\_-\_get\_next\_day')?['resultsets']?['Table1']",

"runAfter": {

"Execute\_a\_SQL\_query\_-\_get\_next\_day": [

"Succeeded"

]

},

"type": "Foreach"

}

},

"contentVersion": "1.0.0.0",

"outputs": {},

"parameters": {

"$connections": {

"defaultValue": {},

"type": "Object"

}

},

"triggers": {

"Recurrence": {

"recurrence": {

"frequency": "Day",

"interval": 1,

"schedule": {

"hours": [

"7"

]

},

"timeZone": "Eastern Standard Time"

},

"type": "Recurrence"

}

}

}

}

# **Create line plots of Energy Price Time Series data in RStudio**

### Download and install R from <http://archive.linux.duke.edu/cran/>

1. Download and install RStudio Desktop (Open Source License) from <https://www.rstudio.com/products/rstudio/download/>

1. Open RStudio and create a ‘New Project’ specifying the location you wish to store your R scripts.
2. Install the following R packages by executing these commands:

install.packages("odbc")

install.packages("lubridate")

install.packages("xts")

install.packages("forecast")

1. In RStudio create and run the following R script, preShaleCorrelation.R:

==========

library(odbc)

#connect to SQL server, database adair-db

cn <- dbConnect(odbc::odbc(), .connection\_string = "Driver={SQL Server};server=adair-sql-server.database.windows.net;database=adair-db;uid=deepazureguest;pwd=4Tr%&~RKVCw4xdB;")

#retrieve energy price data

startDate <- '20000101'

endDate <- '20081231'

sql <- paste('select Day, PetroPrice, NGasPrice from dbo.vwPetroNaturalGasPrices where DayNum between ', startDate, " and ", endDate)

energy <- dbGetQuery(cn, sql)

energy$Day = as.POSIXct(strptime(energy$Day, format="%Y-%m-%d","GMT"))

#increase margins bottom,left,top,right

par(mar=c(4,5,4,5))

#plot petroleum price series

plot(energy$Day, energy$PetroPrice, type="l", xlab="", ylab="", col="black", main="Prices in the Pre-Shale Period")

mtext("Brent Petroleum Price ($ per Bbl", side=2, line=2.5)

#prepare to plot on same graph

par(new=TRUE)

#plot natural gas price series

plot(energy$Day, energy$NGasPrice, type="l", xlab="", ylab="", col="blue", axes=FALSE)

mtext("HenryHub Natural Gas Prices ($ per Mil Btu)", side=4, col="blue", line=4)

axis(4, col="blue", col.axis="blue", las=1)

#add legend

legend("topleft", legend=c("Petroleum","Natural Gas"), text.col=c("black","blue"), bty="n")

#get correlation coefficient and stat summary of Petroleum and Gas Prices over a few date ranges

corVal <- cor(energy$PetroPrice, energy$NGasPrice)

cat("correlation=",corVal)

print(' ');

summaryVal <- summary(energy)

print(summaryVal)

> source('C:/Users/adair/OneDrive/DA/Final/Project/Energy Analysis/preShaleCorrelation.R')

correlation= 0.7206092[1] " "

Day PetroPrice NGasPrice

Min. :2000-01-04 00:00:00 Min. : 16.51 Min. : 1.690

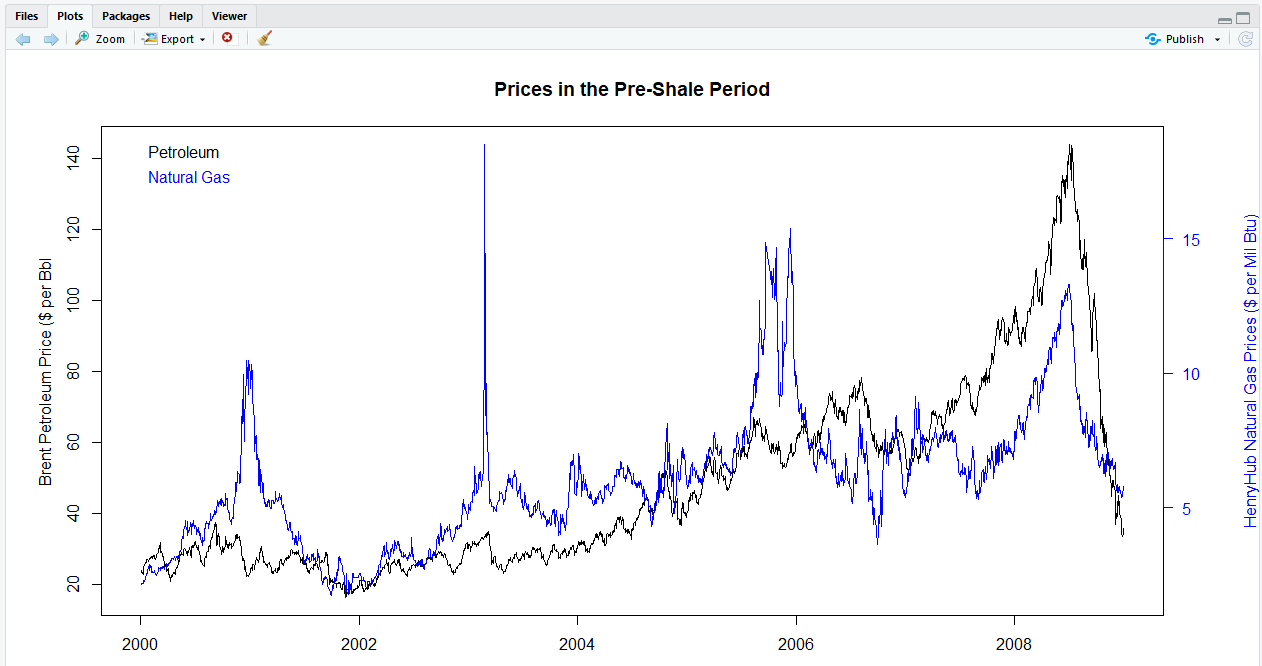
1st Qu.:2002-04-03 00:00:00 1st Qu.: 27.53 1st Qu.: 4.400

Median :2004-07-01 00:00:00 Median : 37.11 Median : 5.910

Mean :2004-07-04 12:24:13 Mean : 48.34 Mean : 6.026

3rd Qu.:2006-10-10 00:00:00 3rd Qu.: 62.95 3rd Qu.: 7.220

Max. :2008-12-31 00:00:00 Max. :143.95 Max. :18.480



1. In RStudio create and run the following R script, shaleCorrelation.R:

========

library(odbc)

#connect to SQL server, database adair-db

cn <- dbConnect(odbc::odbc(), .connection\_string = "Driver={SQL Server};server=adair-sql-server.database.windows.net;database=adair-db;uid=deepazureguest;pwd=4Tr%&~RKVCw4xdB;")

#retrieve energy price data

startDate <- '20090101'

endDate <- '20181231'

sql <- paste('select Day, PetroPrice, NGasPrice from dbo.vwPetroNaturalGasPrices where DayNum between ', startDate, " and ", endDate)

energy <- dbGetQuery(cn, sql)

energy$Day = as.POSIXct(strptime(energy$Day, format="%Y-%m-%d","GMT"))

#increase margins bottom,left,top,right

par(mar=c(4,5,4,5))

#plot petroleum price series

plot(energy$Day, energy$PetroPrice, type="l", xlab="", ylab="", col="black", main="Prices in the Shale Periods")

mtext("Brent Petroleum Price ($ per Bbl", side=2, line=2.5)

#prepare to plot on same graph

par(new=TRUE)

#plot natural gas price series

plot(energy$Day, energy$NGasPrice, type="l", xlab="", ylab="", col="blue", axes=FALSE)

mtext("HenryHub Natural Gas Prices ($ per Mil Btu)", side=4, col="blue", line=4)

axis(4, col="blue", col.axis="blue", las=1)

#add legend

legend("topright", legend=c("Petroleum","Natural Gas"), text.col=c("black","blue"), bty="n")

#get correlation coefficient and stat summary of Petroleum and Gas Prices over a few date ranges

corVal <- cor(energy$PetroPrice, energy$NGasPrice)

cat("correlation=",corVal)

print(' ');

summaryVal <- summary(energy)

print(summaryVal)

|  |
| --- |
| > source('C:/Users/adair/OneDrive/DA/Final/Project/Energy Analysis/shaleCorrelation.R')  correlation= 0.3769287[1] " "  Day PetroPrice NGasPrice  Min. :2009-01-02 00:00:00 Min. : 26.01 Min. :0.000  1st Qu.:2011-04-12 18:00:00 1st Qu.: 53.13 1st Qu.:2.840  Median :2013-07-31 12:00:00 Median : 77.16 Median :3.420  Mean :2013-07-26 20:19:51 Mean : 79.82 Mean :3.475  3rd Qu.:2015-11-09 06:00:00 3rd Qu.:108.49 3rd Qu.:4.070  Max. :2018-02-05 00:00:00 Max. :128.14 Max. :8.150 |
|  |
| |  | | --- | |  | |

1. In RStudio create following R script, energyPriceAnalysis.R, which displays both pre-shale and shale period time series on the same plot:

==========

library(odbc)

#connect to SQL server, database adair-db and

cn <- dbConnect(odbc::odbc(), .connection\_string = "Driver={SQL Server};server=adair-sql-server.database.windows.net;database=adair-db;uid=deepazureguest;pwd=4Tr%&~RKVCw4xdB;")

#retrieve energy price data

startDate <- '20000101'

endDate <- '20181231'

sql <- paste('select Day, PetroPrice, NGasPrice from dbo.vwPetroNaturalGasPrices where DayNum between ', startDate, " and ", endDate)

energy <- dbGetQuery(cn, sql)

energy$Day = as.POSIXct(strptime(energy$Day, format="%Y-%m-%d","GMT"))

#increase margins bottom,left,top,right

par(mar=c(4,5,4,5))

#plot petroleum price series

plot(energy$Day, energy$PetroPrice, type="l", xlab="", ylab="", col="black", main="Prices in the Pre-Shale and Shale Periods")

mtext("Brent Petroleum Price ($ per Bbl", side=2, line=2.5)

abline(v=energy$Day[2230], col="red")

#prepare to plot on same graph

par(new=TRUE)

#plot natural gas price series

plot(energy$Day, energy$NGasPrice, type="l", xlab="", ylab="", col="blue", axes=FALSE)

mtext("HenryHub Natural Gas Prices ($ per Mil Btu)", side=4, col="blue", line=4)

axis(4, col="blue", col.axis="blue", las=1)

#add legend

legend("topright", legend=c("Petroleum","Natural Gas"), text.col=c("black","blue"), bty="n")

#get correlation coefficient and stat summary of Petroleum and Gas Prices over a few date ranges

corVal <- cor(energy$PetroPrice, energy$NGasPrice)

cat("correlation=",corVal)

print(' ');

summaryVal <- summary(energy)

print(summaryVal)

> source('C:/Users/adair/OneDrive/DA/Final/Project/Energy Analysis/energyPriceAnalysis.R')

correlation= 0.1097[1] " "

Day PetroPrice NGasPrice

Min. :2000-01-04 00:00:00 Min. : 16.51 Min. : 0.000

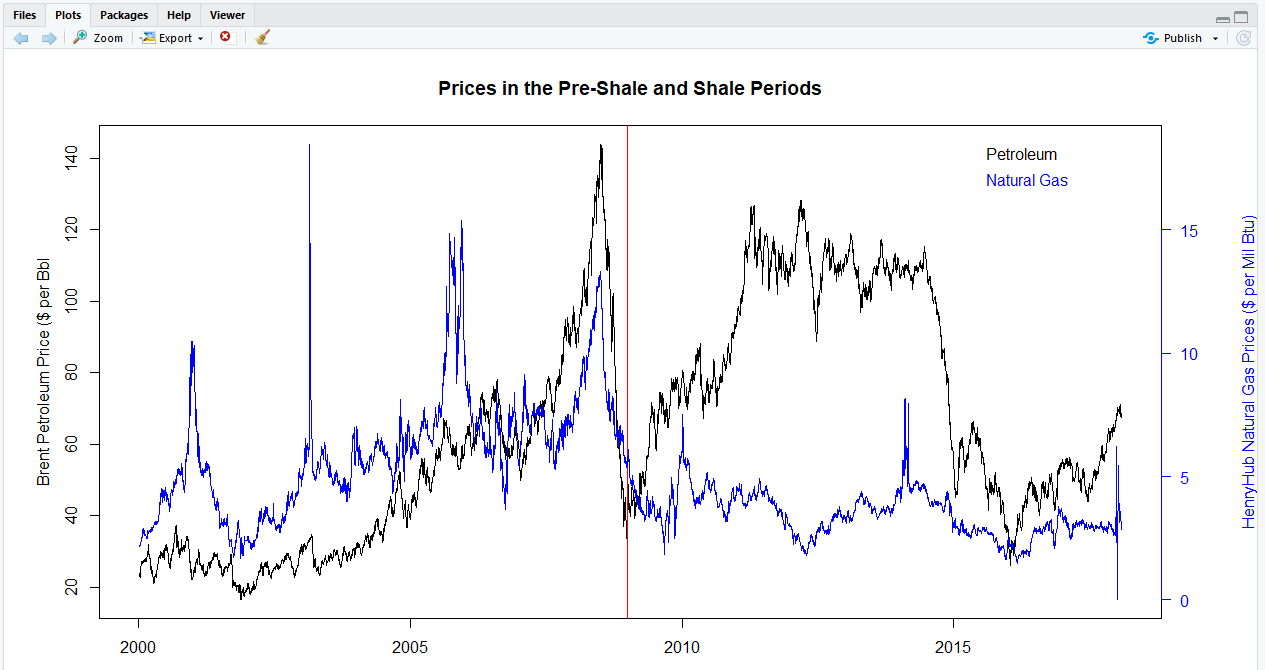
1st Qu.:2004-07-27 00:00:00 1st Qu.: 35.62 1st Qu.: 3.080

Median :2009-02-20 00:00:00 Median : 58.28 Median : 4.150

Mean :2009-02-07 16:22:03 Mean : 64.32 Mean : 4.731

3rd Qu.:2013-08-23 00:00:00 3rd Qu.: 91.59 3rd Qu.: 5.920

Max. :2018-02-05 00:00:00 Max. :143.95 Max. :18.480



1. Energy Analysis Summary

The conventional wisdom gained by the energy experts is that crude oil and natural gas price correlation can be divided into a pre-shale and shale eras with the widespread use of fracking techniques used to extract oil and gas from sedimentary rock.

In the pre-shale period, gas price movement generally followed crude oil because gas extraction was a by-product of crude oil extraction where crude oil has been a well-established global commodity. Natural gas fracking extraction opened new regional markets with independent price supply and demand factors that cause its price to vary differently than crude oil.

|  |  |  |
| --- | --- | --- |
|  | Correlation coefficient | My analysis |
| 1997-2008 (pre-shale) | 0.9 | correlation= 0.7206092[1] " "  Day PetroPrice NGasPrice  Min. :2000-01-04 00:00:00 Min. : 16.51 Min. : 1.690  1st Qu.:2002-04-03 00:00:00 1st Qu.: 27.53 1st Qu.: 4.400  Median :2004-07-01 00:00:00 Median : 37.11 Median : 5.910  Mean :2004-07-04 12:24:13 Mean : 48.34 Mean : 6.026  3rd Qu.:2006-10-10 00:00:00 3rd Qu.: 62.95 3rd Qu.: 7.220  Max. :2008-12-31 00:00:00 Max. :143.95 Max. :18.480 |
| 2009-present | 0.54 | correlation= 0.3768616[1] " "  Day PetroPrice NGasPrice  Min. :2009-01-02 00:00:00 Min. : 26.01 Min. :0.000  1st Qu.:2011-04-08 18:00:00 1st Qu.: 52.97 1st Qu.:2.840  Median :2013-07-24 12:00:00 Median : 77.28 Median :3.420  Mean :2013-07-19 15:24:43 Mean : 79.87 Mean :3.475  3rd Qu.:2015-10-28 18:00:00 3rd Qu.:108.51 3rd Qu.:4.070  Max. :2018-01-22 00:00:00 Max. :128.14 Max. :8.150 |

Note that correlation coefficient calculations on time series data (versus cross section data) in general is not a reliable measure. This is due the possible trending in the data, for instance in our case, season changes in the price, that might indicate a false correlation.

This is a lesson learned in my initial analysis of the energy price data. From my initial study of trending and how it can impact time series correlation, it is advised to model the trend in each time series and use the model to remove it. See the article ‘Avoiding Common Mistakes with Time Series’ by Tom Fawcett at <https://svds.com/avoiding-common-mistakes-with-time-series/> This would be an exercise as a next step beyond the scope of this project but my inclination would be to apply the knowledge learned in the class lecture on Machine Learning and especially the application of linear regression and modeling.