Hands on Lab 8 - Azure Data Factory

# Introduction

In this lab we will use Azure Data Factory to execute a Hive query using a pre-created HDInsight cluster. We will push the query output to an Azure SQL Database.

The Azure Data Factory service is a fully managed service for composing data storage, processing, and movement services into streamlined, scalable, and reliable data production pipelines. Developers can use Data Factory to transform semi-structured, unstructured and structured data from on-premises and cloud sources into trusted information. Developers build data-driven workflows (pipelines) that join, aggregate and transform data sourced from their on-premises, cloud-based and internet services, and set up complex data processing through simple JSON scripting.

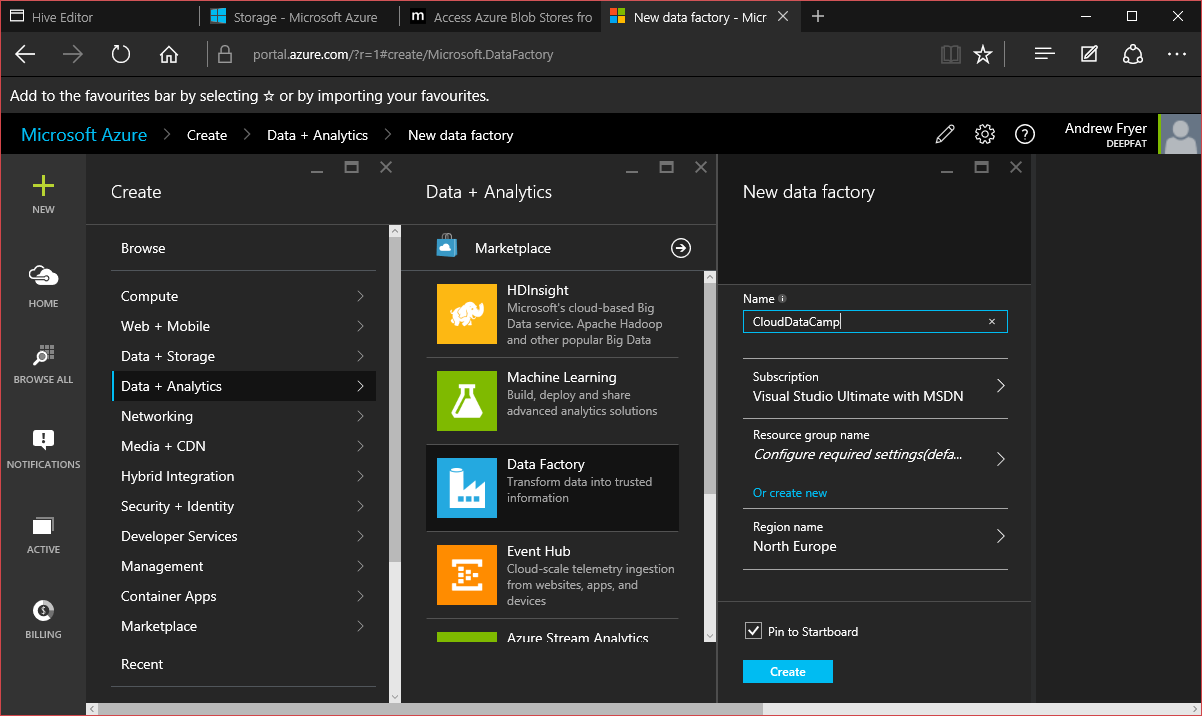
Although it might be compared and considered to like ETL including Informatica and SQL Server Integration Services it works by applying scripts to the sources and target it works on, and at the time of writing has no visual designer. However it is very scalable and resilient with rich monitoring and diagnostics which are graphical and intuitive.

# Pre-requisites

* A SQL Server insance has been created (HOL 1)
* The table readingsByMinute has been created (HOL 1)
* HBase cluster (created during lab HOL 2)

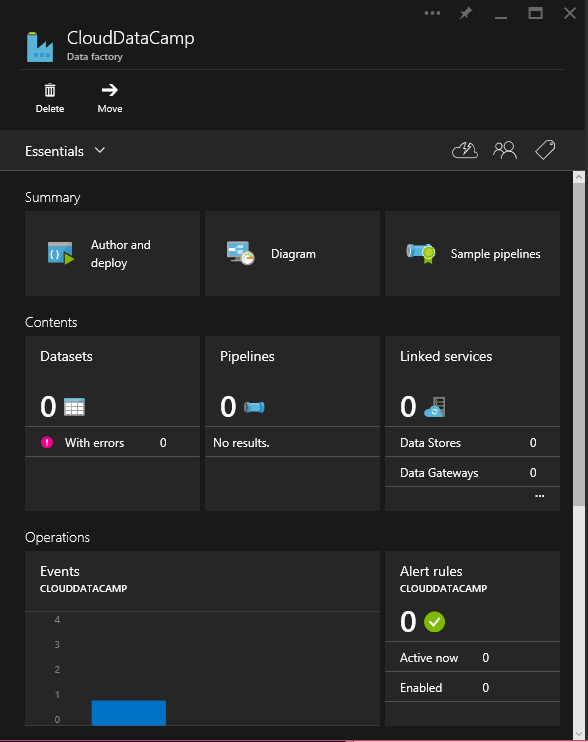
# Create a new Azure Data Factory

1. Navigate to <https://portal.azure.com/> and sign with your Microsoft or Organizational account
2. In the upper left hand corner of the screen Click New -> Data + Analytics -> Data Factory and complete the blade for a new Data Factory ..

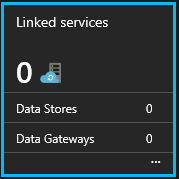


Set the Name and Resource Group for the data factory, currently the region is limited to North Europe which we’ll use and to West US

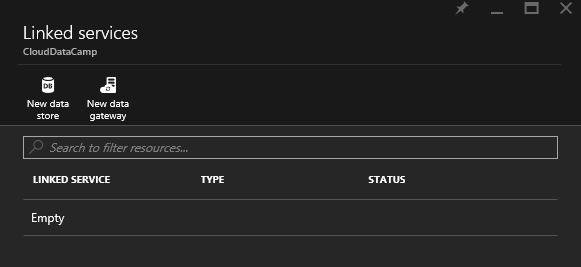
1. Navigate to the data factory once it has been successfully created.

* 

1. We need to create a number of Linked Services that represent Azure Blob Storage and SQL Server instances we have previously created. Click on the linked services area.

* 

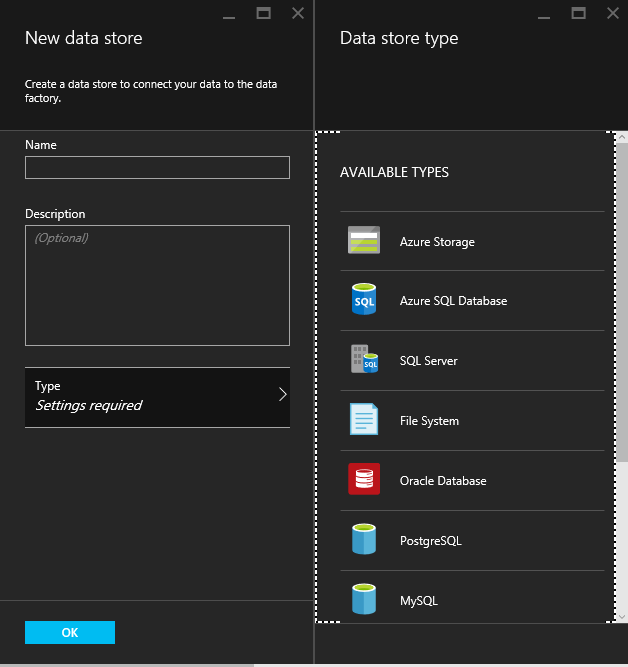
1. This will display the following blade

* 

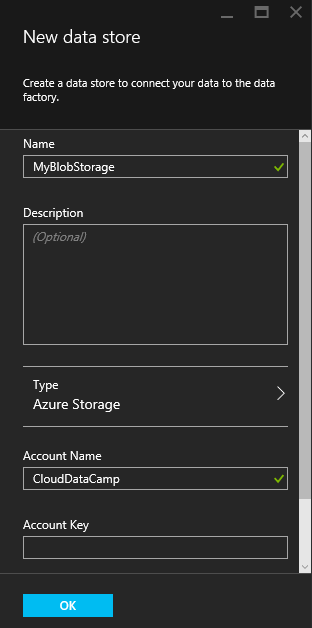
1. Click Add Data Store

* 

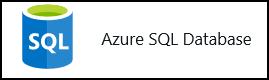
1. Set a name for the data store "MyBlobStore" and expand the type to open the list of possible types that ADF can connect with and set this to Azure Storage

* 

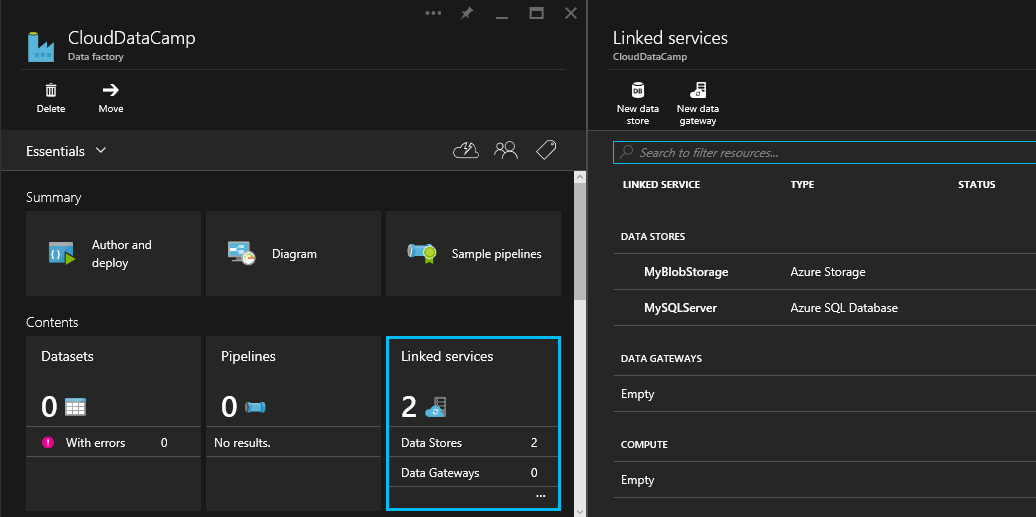
1. The blade will change to allow us to enter the Azure Storage Account Name and one of its Account Keys..



1. Add another data store, this will link to the SQL Server we created in Lab 1 and this time select the Type as Azure SQL Server

* 

1. Enter the credentials for Azure SQL Server we created in Lab 1 and click OK. There should now be two data stores in the Data Factory blade..



## Configuring ADF

We need to create some additional objects, to represent the sources and targets of the pipelines we create in ADF and the pipeline processes. There are three ways we can do this work ..

* 1. We can go into Author and deploy in the DataFactory blade of the Azure portal and work there.
  2. We can interact with the ADF API with Powerhsell and create our objects programmatically
  3. We can use the ADF templates in VS2013 in the VM we have created

It might be good to try each one but whichever method is chosen the json scipts belwo must be edited to reference the Data Stores (aka linked services) we have created and the data sets we are working with.

Note all the code below is in the ..src/ADF folder

1. Open document MyBlobInput.json and update the value for property linkedServiceName to the name of the storage data store you created. Replace mystoragecontainer in folderPath with the container the cluster was created with.

* {  
   "name": "MyBlobInput",  
   "properties":  
   {  
   "location":   
   {  
   "type": "AzureBlobLocation",  
   "folderPath": "mystoragecontainer/input",  
   "linkedServiceName": "MyBlobStore"  
   },  
   "availability":   
   {  
   "frequency": "minute",  
   "interval": 15,  
   "waitonexternal": {}  
   }  
   }  
  }

1. Open MyBlobOutput.json and update the value for property linkedServiceName to the name of the storage data store you created. Replace mystoragecontainer in folderPath with the container the cluster was created with.

* {  
   "name": "MyBlobOutput",  
   "properties":  
   {  
   "location":   
   {  
   "type": "AzureBlobLocation",  
   "folderPath": "mystoragecontainer/output/myaverageByMinute",  
   "linkedServiceName": "MyBlobStore"  
   },  
   "availability":   
   {  
   "frequency": "Minute",  
   "interval": 15  
   }  
   }  
  }

1. Open MyHiveSourceData.json and update the value for property linkedServiceName to the name of the storage data store you created. Replace mystoragecontainer in folderPath with the container the cluster was created with.

* {  
   "name": "MyHiveSourceData",  
   "properties":  
   {  
   "structure": [  
   { "name": "DeviceType", "type": "String"},  
   { "name": "DateOfReading", "type": "String"},  
   { "name": "RoomNumber", "type": "String"},  
   { "name": "Reading", "type": "Decimal"}  
   ],  
   "location":  
   {  
   "type": "AzureBlobLocation",  
   "folderPath": "mystoragecontainer/output/myaverageByMinute",  
   "fileName": "000000\_0",  
   "linkedServiceName": "MyBlobStore",  
   "format":  
   {  
   "type": "TextFormat",  
   "columnDelimiter": "\t"  
   }  
   },  
   "availability":  
   {  
   "frequency": "Minute",  
   "interval": 15,  
   "waitonexternal": {}  
   }  
   }  
  }

1. Open and update the value for property linkedServiceName to the name of the SQL Server data store you created

* {  
   "name": "MySQLStore",  
   "properties":  
   {  
   "structure":  
   [  
   { "name": "DeviceType", "type": "String"},  
   { "name": "DateOfReading", "type": "String"},  
   { "name": "RoomNumber", "type": "String"},  
   { "name": "Reading", "type": "Decimal"}  
   ],  
   "location":  
   {  
   "type": "AzureSQLTableLocation",  
   "tableName": "readingsByMinute",  
   "linkedServiceName": "MySQLStore"  
   },  
   "availability":   
   {  
   "frequency": "Minute",  
   "interval": 15   
   }  
   }  
  }

1. Open MyHDInsightCluster.json, Modify ClusterUri to point to your cluster, the UserName/Password values are those you used during cluster creation. Update the value for property linkedServiceName to the name of the storage data store you created

* {  
   "Name": "MyHDInsightCluster",  
   "Properties":   
   {  
   "Type": "HDInsightBYOCLinkedService",  
   "ClusterUri": "https://<clusterName>.azurehdinsight.net/",  
   "UserName": "<username>",  
   "Password": "<password>",  
   "LinkedServiceName": "MyBlobStore"  
   }  
  }

1. You should not need to update ADFHivePipeline.json. This pipeline object uses the datasets detailed above into a pipeline composed of activities. Replace mystoragecontainer in scriptpath with the storage account the cluster was created with.

* {  
   "name": "ADFHivePipeline",  
   "properties":  
   {  
   "description" : "Runs a HiveQL query and stores the result in a blob",  
   "activities":  
   [  
   {  
   "name": "RunHiveQuery",  
   "description": "Runs a hive query",  
   "type": "HDInsightActivity",  
   "inputs": [{"name": "MyBlobInput"}],  
   "outputs": [ {"name": "MyBlobOutput"} ],  
   "linkedServiceName": "MyHDInsightCluster",  
   "transformation":  
   {  
   "type": "Hive",  
   "scriptpath": "mystoragecontainer/scripts/queries.hql",  
   "scriptLinkedService": "MyBlobStore"  
   },  
   "policy":  
   {  
   "concurrency": 1,  
   "executionPriorityOrder": "NewestFirst",  
   "retry": 1,  
   "timeout": "01:00:00"  
   }  
   },  
   {  
   "name": "CopyFromBlobToSQL",  
   "description": "Push data from Hive query to SQL database",  
   "type": "CopyActivity",  
   "inputs": [ {"name": "MyHiveSourceData"} ],  
   "outputs": [ {"name": "MySQLStore"} ],   
   "transformation":  
   {  
   "source":  
   {   
   "type": "BlobSource"  
   },  
   "sink":  
   {  
   "type": "SqlSink"  
   }   
   },  
   "Policy":  
   {  
   "concurrency": 1,  
   "executionPriorityOrder": "NewestFirst",  
   "style": "StartOfInterval",  
   "retry": 0,  
   "timeout": "01:00:00"  
   }   
   }  
   ]  
   }  
  }

## Optional Deploying ADF Objects with PowerShell

1. Open Powershell, navigate to src/ADF directory
2. Execute the command to authenticate against Azure and ensure you are in the correct subscription.

* Add-AzureAccount

1. Execute the following command to switch modes

* Switch-AzureMode AzureResourceManager

1. The following commands will deploy the set of objects to the data factory. Substitute <ResourceGroupName> and <DataFactoryName> as appropriate

New-AzureDataFactoryTable –ResourceGroupName <ResourceGroupName> –DataFactoryName <DataFactoryName> -File .\MyBlobInput.json

New-AzureDataFactoryTable –ResourceGroupName <ResourceGroupName> –DataFactoryName <DataFactoryName> -File .\MyBlobOutput.json

New-AzureDataFactoryTable –ResourceGroupName <ResourceGroupName> –DataFactoryName <DataFactoryName> -File .\MyHiveSourceData.json

New-AzureDataFactoryTable –ResourceGroupName <ResourceGroupName> –DataFactoryName <DataFactoryName> -File .\MySQLStore.json

New-AzureDataFactoryLinkedService -ResourceGroupName <ResourceGroupName> -DataFactoryName <DataFactoryName> -File .\MyHDInsightCluster.json

New-AzureDataFactoryPipeline –ResourceGroupName <ResourceGroupName> –DataFactoryName <DataFactoryName> –File .\ADFHivePipeline.json

Set the execution bounds using the command below, this will define the start time/end time of execution of the pipeline. Modify the start/end date string so it starts approximately 5 mins in the future. Note this pipline will be processed every 15 minutes as defined in the availability section of the various inputs/outputs.

Set-AzureDataFactoryPipelineActivePeriod `

-ResourceGroupName MyNewResourceGroup `

-DataFactoryName mynewdatafactory `

-StartDateTime 2015-02-10T16:00:00Z `

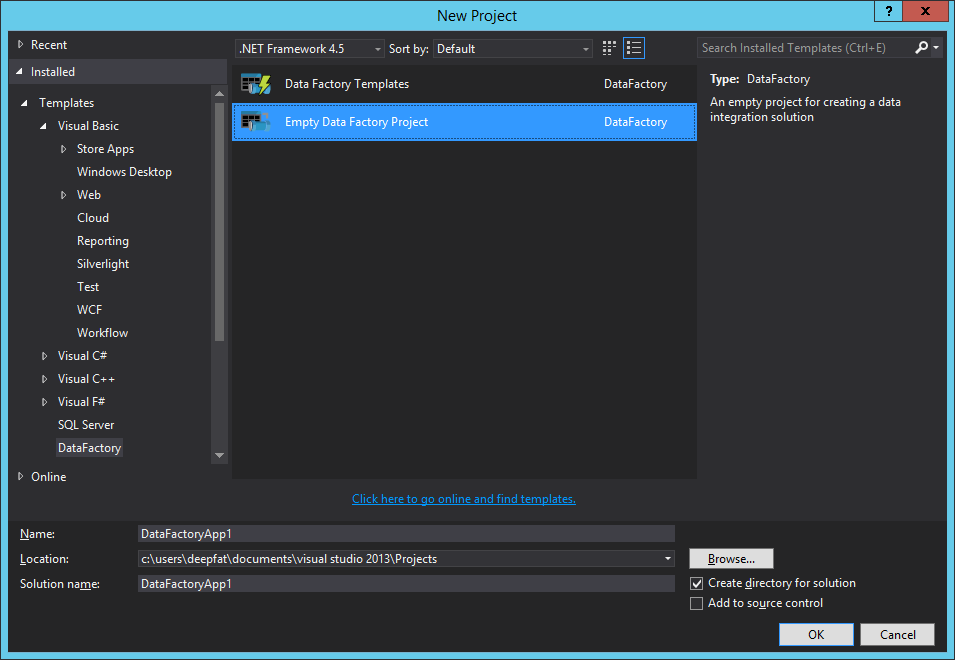
–EndDateTime 2015-02-10T21:30:00Z `

–Name ADFHivePipeline

## Developing Data Factories in Visual Studio - optional

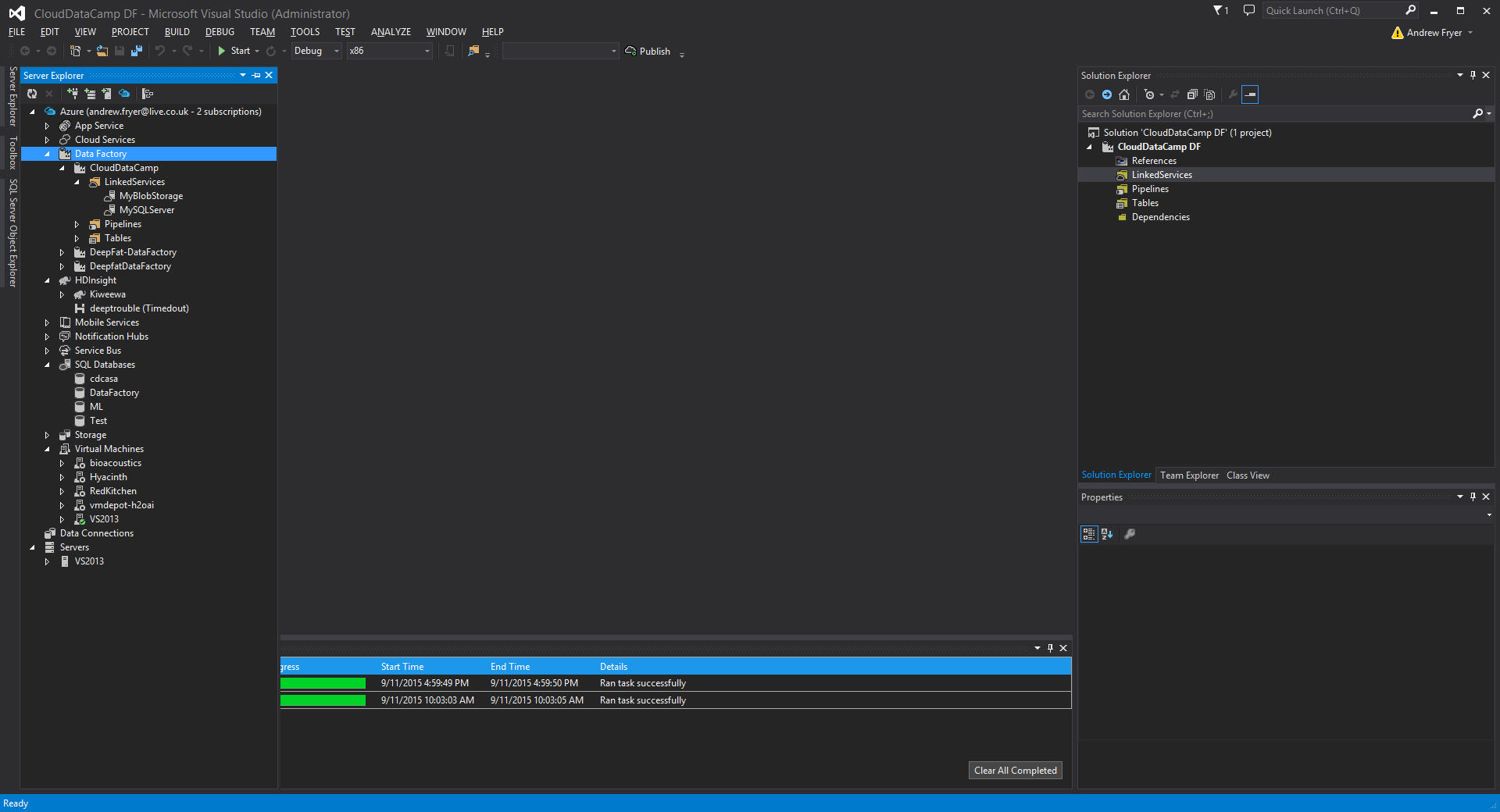
The ADF team recently developed templates for Visual Studio 2013 which are included in the Azure VM we have created.

1. Open Visual Studio and create a new project (File->New Project). Expand Templates and at the end of the list select Data Factory. Select and empty Data Factory Project

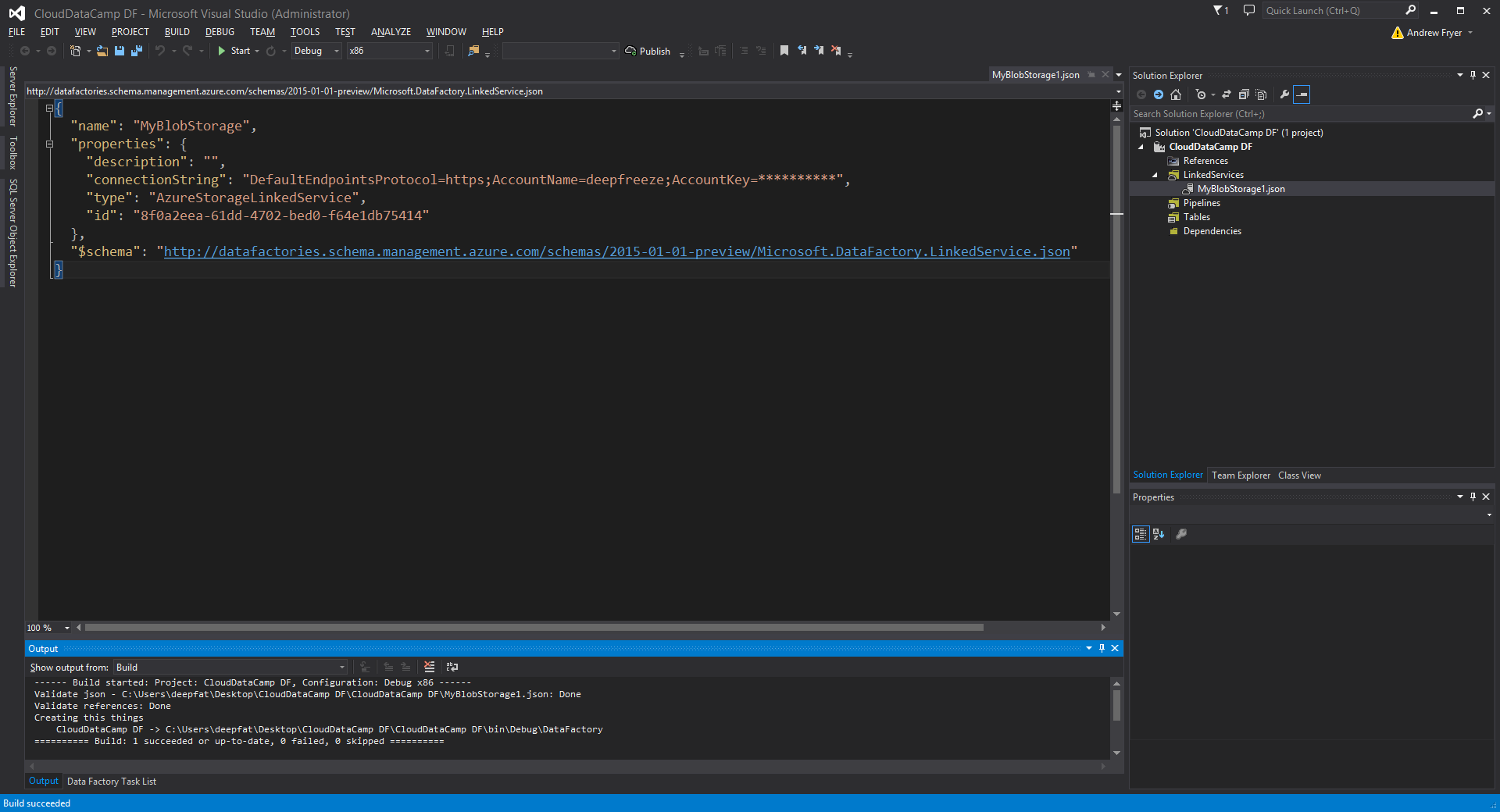


The first thing we might want to do is to add the work we did earlier specifically the Data Stores/Linked Services we created.

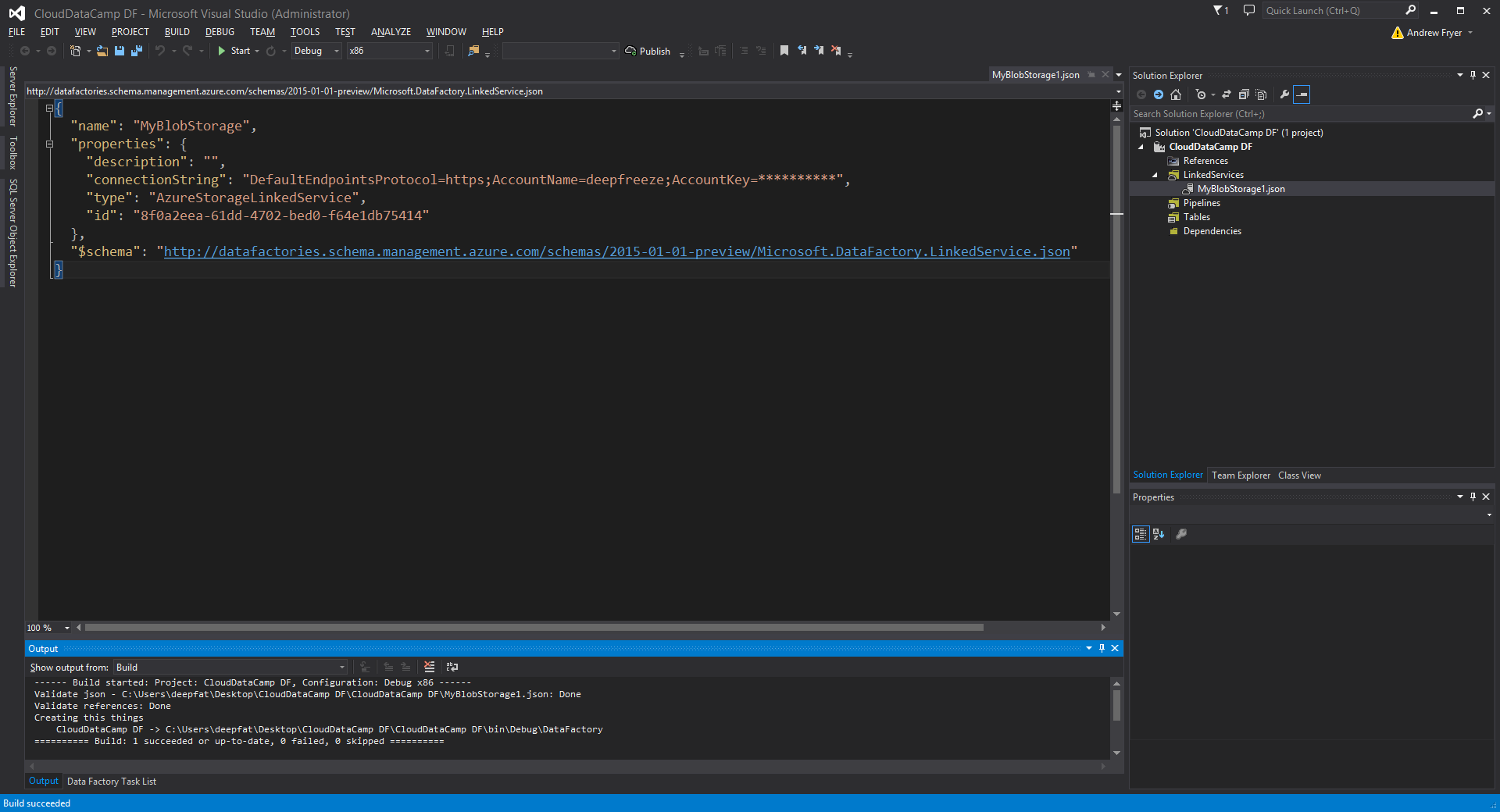
1. Open Server Explorer on the Left of VS2013 refresh our Azure subscription and expand out the Data Factory Node..



Right click on the MyBlobStorage linked service and select Add to Data Factory Project. The Linked Service is now in our project..

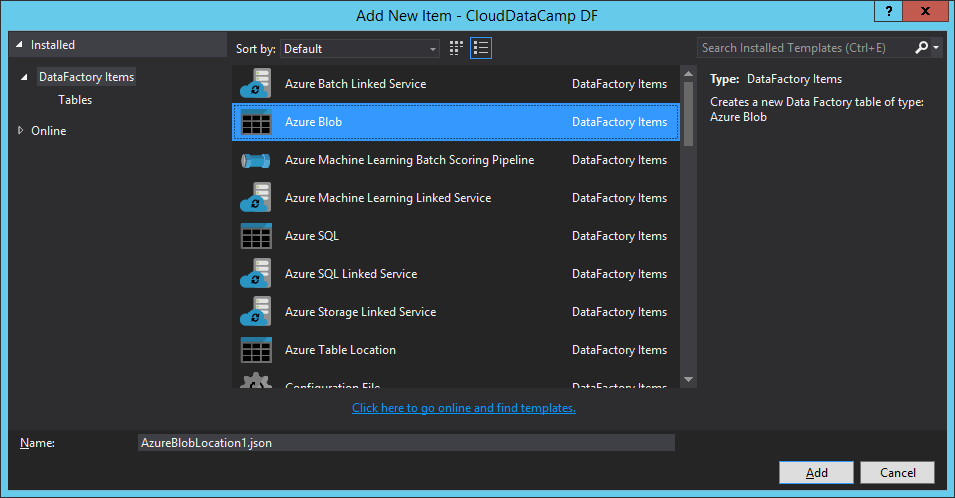


1. We can also see the json source..

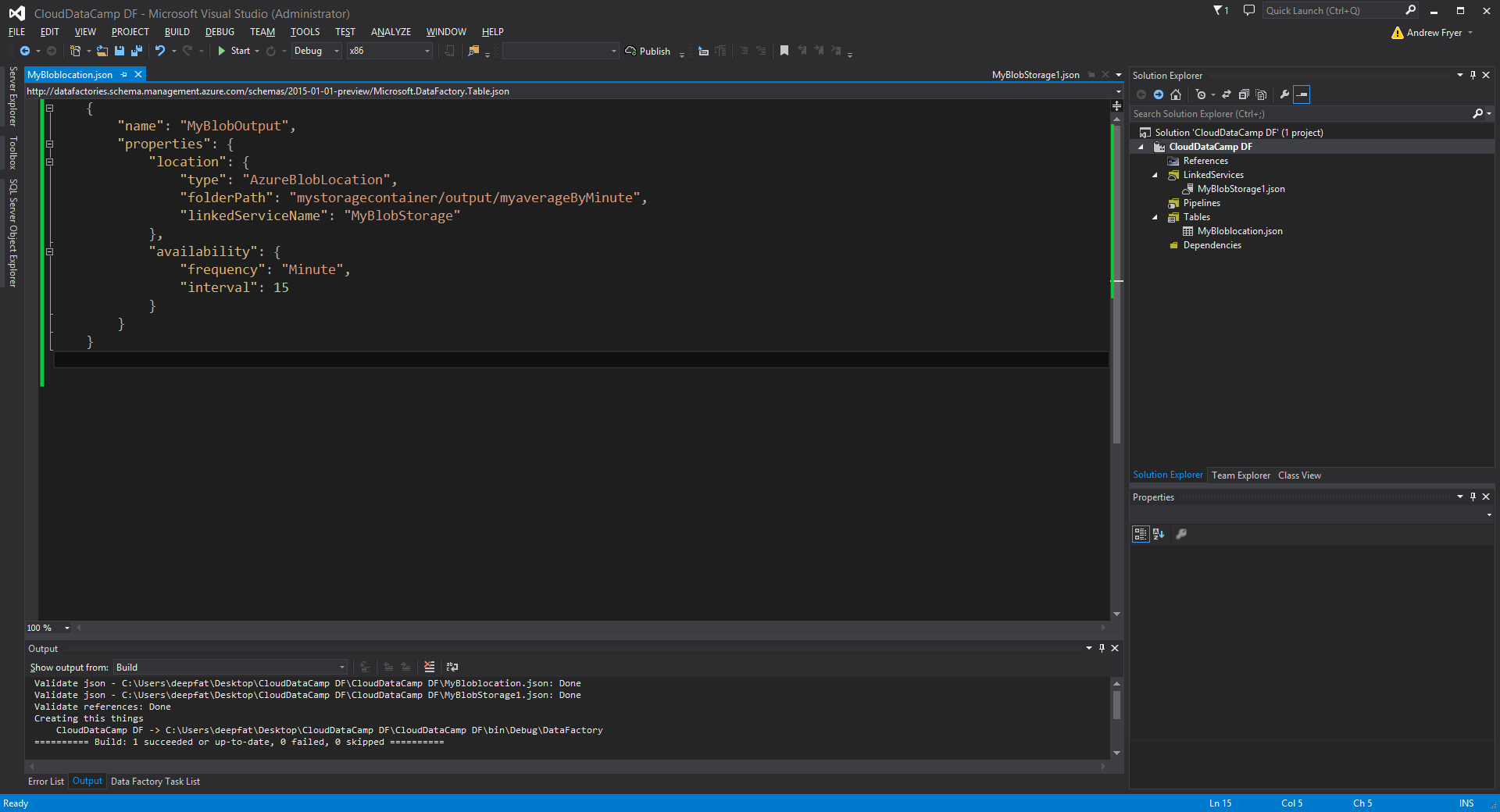


Note that we can add a description if we want and that the storage account key is obfuscated so we would need to add it back in if we were to redeploy.

Now we know where the services we want to connect are we can add in the data sets we want by selecting tables -> Add -> New Item and selecting the appropriate object, Azure Blob..



1. We get a template we can build from but in our case we can just cut in the code inn section 3.2 in this case from para 2.. and editing it and renaming the object to file we created to reflect its name..



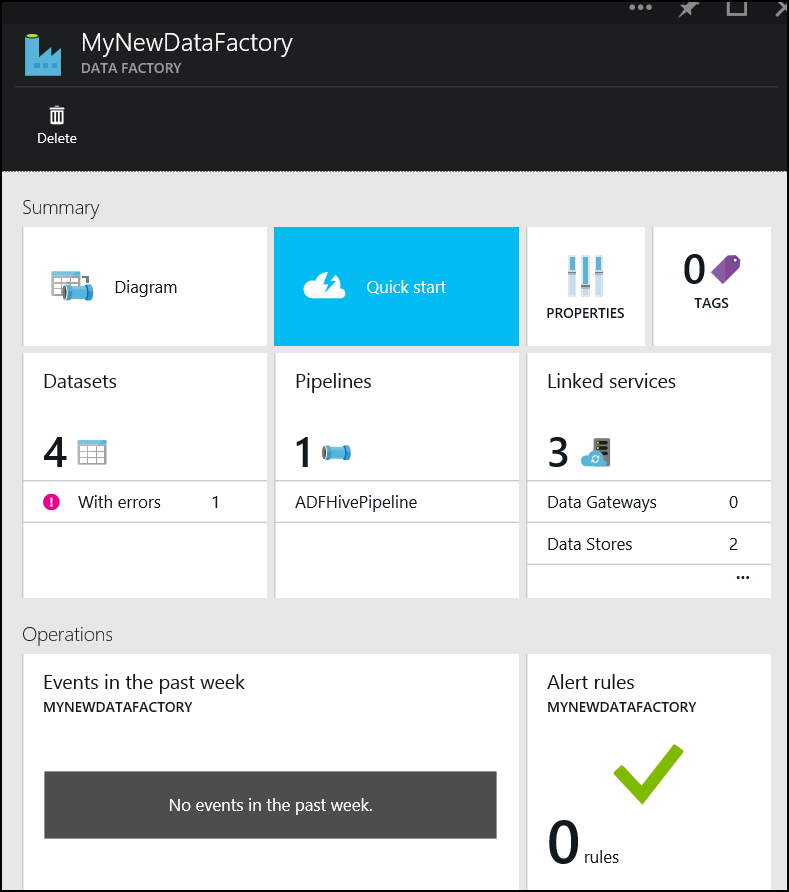
VS2013 will also allow us to check consistency by building the project as we would any other solution.

It’s also possible to export an existing ADF to VS2013 and then this could be replayed to a new ADF on a different subscription which would be much more labour intensive otherwise.

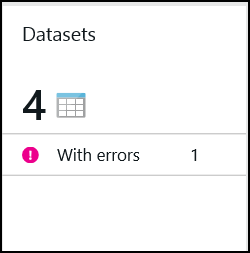
## Troubleshooting

You view the progress of the pipeline in the portal.

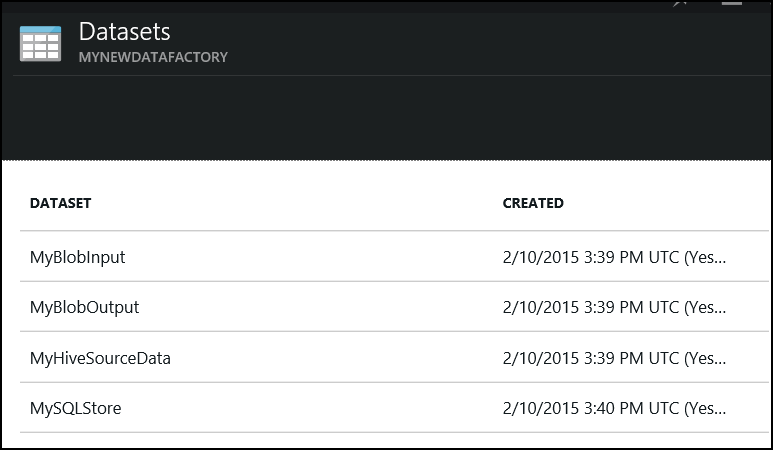
1. Navigate to the data factory in the portal

* 

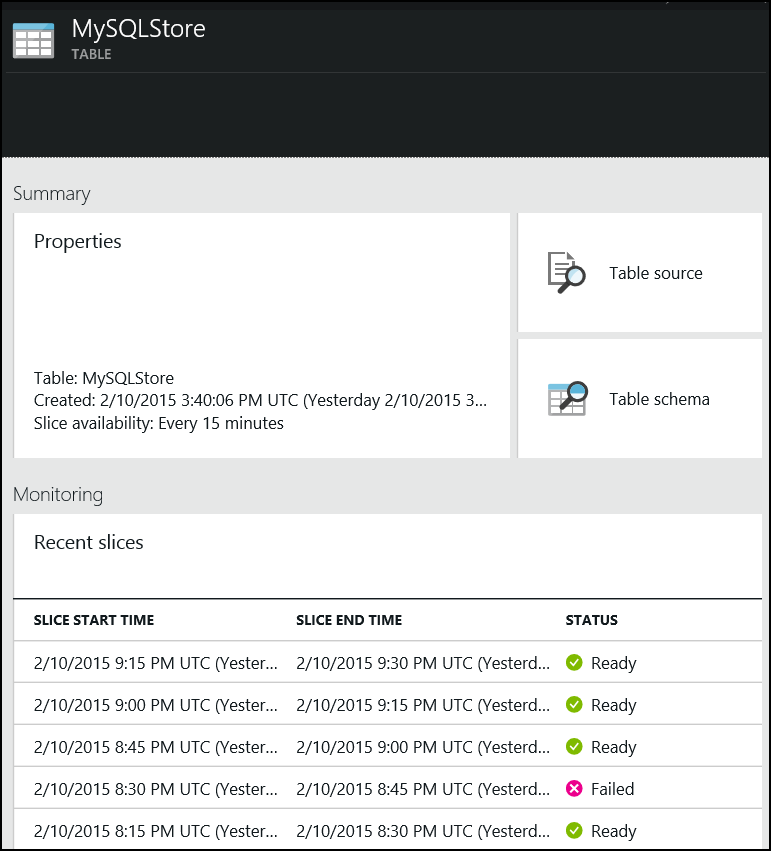
1. Choose option Data Sets

* 

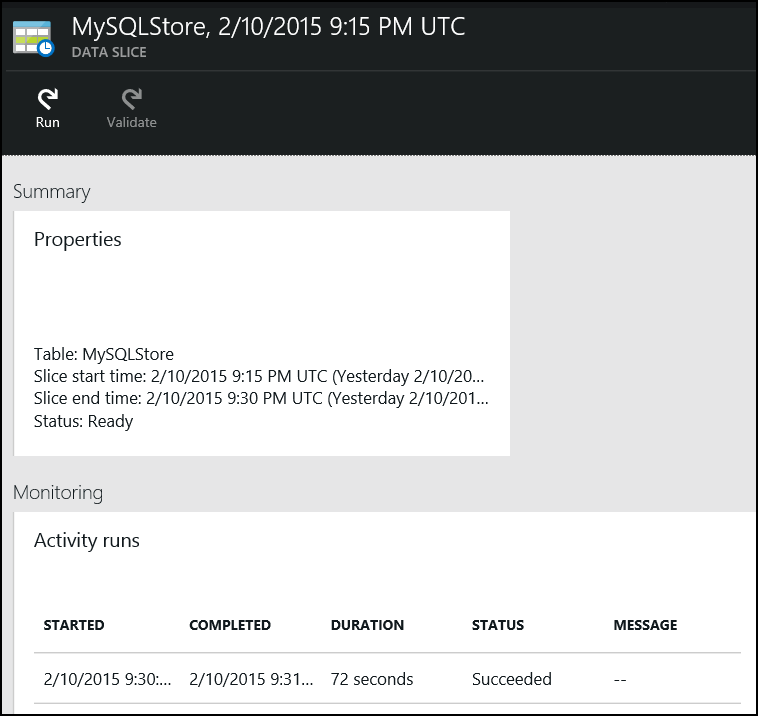
1. These objects equate to the inputs/outputs to the pipeline, select MySQLStore

* 

1. This will display a blade that shows successful and unsuccessful runs.

* 

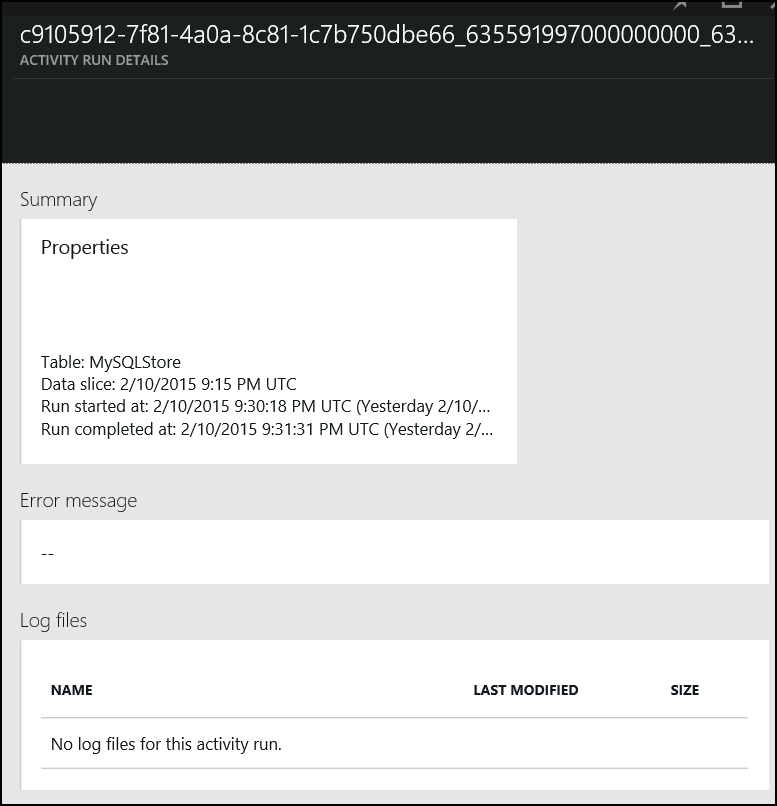
1. Clicking on a successful run will display the following blade, this provides additional information on the run.

* 

1. Click the individual run details

* createDataFactoryImg24.png

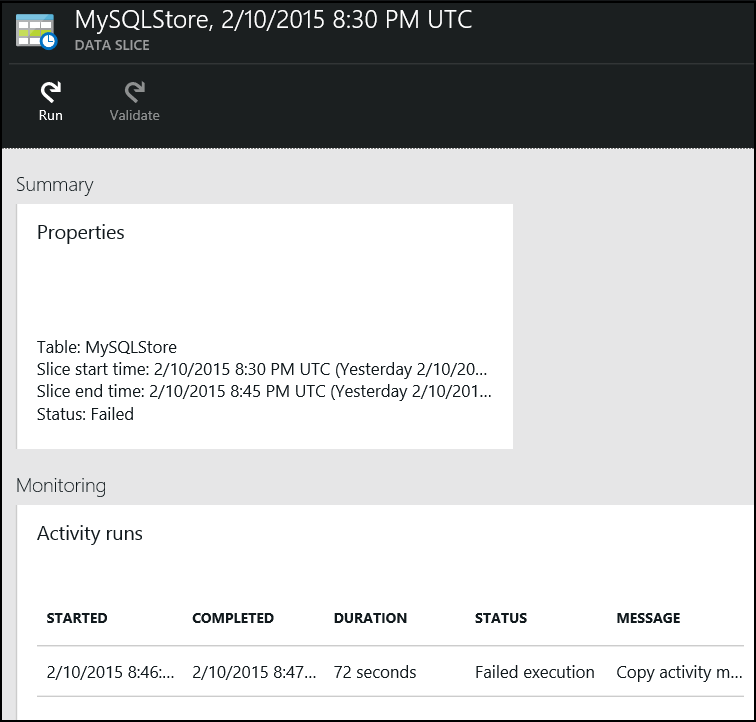
1. This will open an additional blade with will show any logs.

* 

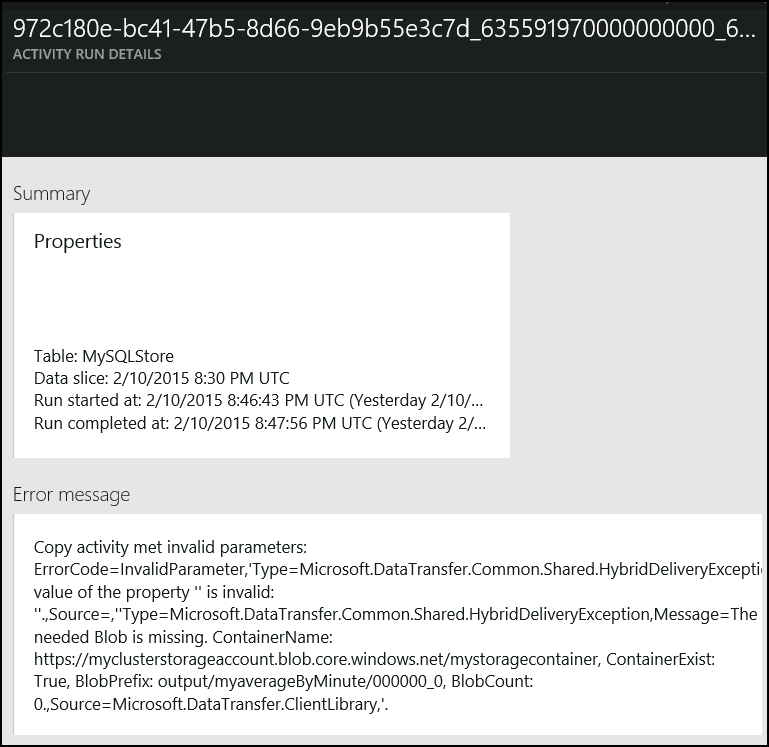
1. For instance we can choose a failed run

* createDataFactoryImg21.png

1. We can get the details.

* 

1. And get the log to help diagnose the issue.

* 

1. We can do the above with any of the Data Sets we created.