

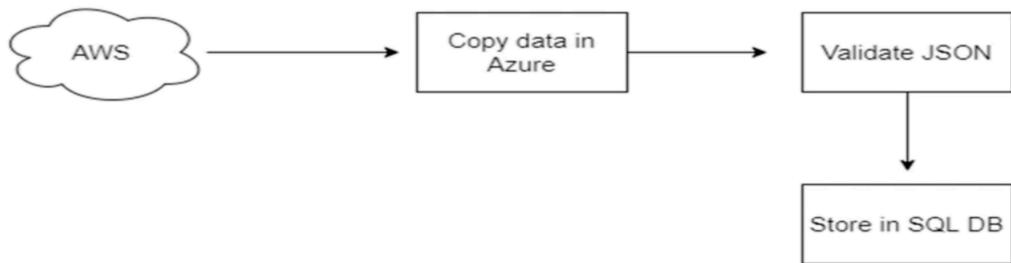
Project 1 Connected vehicles

General Motors is one of the leading heavy vehicle manufacturing company. To improve their services they are planning to rollout lot new features based on IoT.

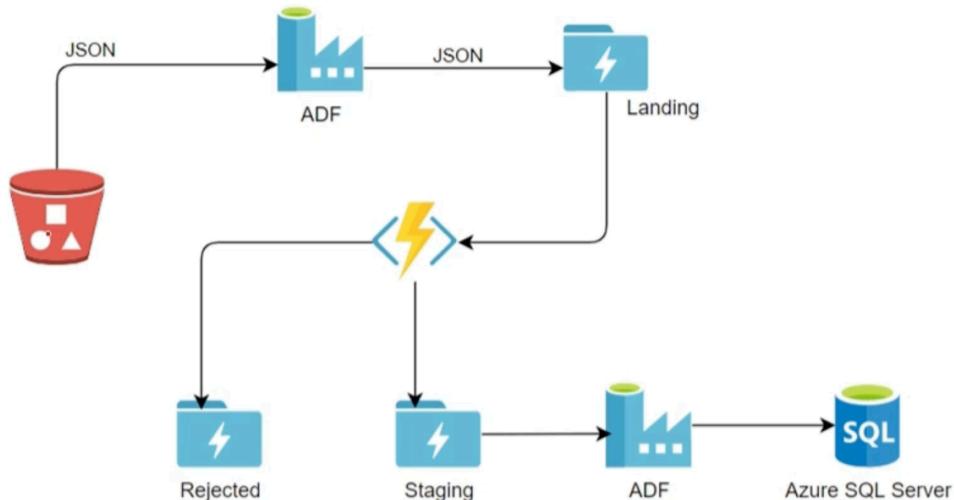
Vehicle has third party IoT device which will send the telemetry data (in JSON format) over the AWS cloud.

Data Engineer tasks:

- 1). Move data from third party AWS to General Motors Azure cloud
- 2). Validate the JSON (sometimes it could be incomplete or wrong JSON which need to be rejected)
- 3). After validation, store the data in SQL database



- 4). Design solution



Architecture Diagram for Connected Vehicle Project

Input source: AWS cloud s3 storage bucket where JSON telemetry data is stored

We can use azure data factory for moving the data from one source to another source and move the data in landing folder

Then we can validate the data (JSON format) using azure function. We can use storage based trigger on the Azure function. As soon as file arrive in landing folder, azure function will trigger automatically. This azure function will read the file and just validate whether its in proper JSON format. If it is in valid format then it'll be moved in Staging folder, otherwise in Rejected folder. Once we have data in Staging folder, we can copy this data from Staging Folder to Azure SQL database using Azure Data Factory

Implementation of Project:

- Create a free AWS account (<https://portal.aws.amazon.com>)
- Create an S3 bucket in AWS to store data from connected vehicles' IoT devices
- Upload a sample data file into the S3 bucket, following a specific folder format for date organization
- The data will be sent on a daily basis and saved in the S3 bucket with a year, month, and day format

The screenshot shows the AWS S3 console. At the top, a green banner indicates "Upload succeeded". Below it, the "Upload: status" page displays a summary table with one succeeded file (1 file, 85.2 KB) and zero failed files (0 files, 0 B). The "Files and folders" tab is selected, showing a table with one item: "Customer_Valid.json" (application/json, 85.2 KB, Succeeded).

Summary		
Destination	Succeeded	Failed
s3://iotedataproject1/2023/01/23/	(1) 1 file, 85.2 KB (100.00%)	(0) 0 files, 0 B (0%)

Files and folders (1 Total, 85.2 KB)								
Name	Folder	Type	Size	Status	Error			
Customer_Valid.json	-	application/json	85.2 KB	(S) Succeeded	-			

- The goal is to create an ADF to pull data from this S3 bucket.

1. Create Azure data storage account

The screenshot shows the "Create a storage account" wizard in the Microsoft Azure portal. The "Project details" step is selected, showing fields for "Subscription" (Azure subscription 1) and "Resource group" ((New) Project1). The "Instance details" step is partially visible below, showing fields for "Storage account name" (dataengineerproject1), "Region" (US East US), and "Performance" (Standard: Recommended for most scenarios). The "Review" button is at the bottom left, and the "2. Select" label is on the right.

subscription and resource group.

3. Enter unique storage account name and select standard performance.

The screenshot shows the 'Create a storage account' wizard in the Microsoft Azure portal. The 'Advanced' tab is selected. The configuration includes:

- Require secure transfer for REST API operations: Enabled
- Allow enabling public access on containers: Enabled
- Enable storage account key access: Enabled
- Default to Azure Active Directory authorization in the Azure portal: Disabled
- Minimum TLS version: Version 1.2
- Permitted scope for copy operations (preview): From any storage account

Data Lake Storage Gen2

The Data Lake Storage Gen2 hierarchical namespace accelerates big data analytics workloads and enables file-level access control lists (ACLs). [Learn more](#)

Enable hierarchical namespace: Enabled

At the bottom, there are buttons for 'Review' (highlighted), '< Previous', 'Next : Networking >', and 'Give feedback'.

If we enable hierarchical namespace then this account is converted to adls gen2 account.

4. Click "Review + Create" and wait for storage account creation.
5. Click on "Go to resource" to access storage account.
6. In the storage account go to the “Data Lake Storage”

The screenshot shows the 'dataengineerproject1' storage account overview in the Microsoft Azure portal. The 'Data Lake Storage' section displays the following configuration:

Setting	Value
Hierarchical namespace	Enabled
Default access tier	Hot
Blob public access	Enabled
Blob soft delete	Enabled (7 days)
Container soft delete	Enabled (7 days)
Versioning	Disabled
Change feed	Disabled
NFS v3	Disabled
SFTP	Disabled
Large file share	Disabled

Other sections visible include 'Properties', 'Monitoring', 'Capabilities (5)', 'Recommendations (0)', 'Tutorials', 'Developer Tools', 'Security', 'Networking', and 'File service'.

7. Create a container named "input" inside the storage account.

8. Create a folder named "landing" inside the "input" container.
9. "landing" folder will be the landing folder for all files transferred from AWS.

Create Azure Data Factory Account Setting Up for Data Transfer:

1. Creating a new ADF account in the Azure portal
2. Selecting subscription, resource group, region, and giving a suitable name to the account
3. Skipping Git configuration for now and keeping networking settings default
4. Reviewing the settings and creating the ADF account
5. Accessing the ADF account and opening up the Azure Data Factory studio to create pipelines

The screenshot shows the Azure Data Factory Studio interface. At the top, it displays the project details: **dataengineerproject1** (Data factory (V2)). The **Essentials** section provides key information: Resource group ([move](#)) : Project1, Status : Succeeded, Location : East US, Subscription ([move](#)) : Azure subscription 1, and Subscription ID : e60d5a57-13f1-4a20-bb6f-327c27fe06eb. Below this is the **Azure Data Factory Studio** logo and a **Launch studio** button. The interface is divided into several sections: **Quick Starts**, **Tutorials**, **Template Gallery**, **Training Modules**, **PipelineRuns**, and **ActivityRuns**. At the bottom, there are navigation links: **Review + create**, **< Previous**, **Next : Git configuration >**, and **Give feedback**.

Create Azure Key Vault & Add the secrets: (Create a new Azure Key Vault to store Access Keys and Access ID. Use the Azure Key Vault to manage and store credentials and secrets):

- Store all the secrets within the Azure Key Vault to allow easy access to Azure portal.
- Create a new Azure Key Vault by selecting a subscription and resource group and giving it a unique name.
- Keep all default settings and create the Azure Key Vault.
- Once the Azure Key Vault is created, store the other credential in the secrets of the Azure Key Vault.
- Go to the secrets and click on Generate.
- Copy the Access Key and Access Secret Key for AWS (go to AWS Services —> click on IAM —> click on “Manage Access Keys” —> Create New Access Keys —> copy the Access Key ID —> go to KeyVault & paste the Access Key ID in value -> & Secret Access Key).
- Create a secret for each key by pasting the Access Key and Access Secret Key in the Azure Key Vault.

- Assign access policies to the Azure Key Vault by clicking on Add Access Policy and

Name	Type	Status	Expiration date
s3secretkey	String	✓ Enabled	
s3accesskeyid	String	✓ Enabled	

selecting the Azure Data Factory account as the principal.

- Verify that the keyboard is accessible by the user and the Azure Data Factory account.
- Use the stored Access Key and Access Secret Key to connect to AWS S3 account in a pipeline.

Name	Email	Key Permissions	Secret Permissions	Certificate Permissions
dataengineerproject1		Get, List, Update, Create, Import, D...	Get, List, Set, Delete, Recover, Back...	Get, List, Update, Create, Import, D...
Promila Sharan	promiladala1@gmail.com#EXT#@p...	Get, List, Update, Create, Import, D...	Get, List, Set, Delete, Recover, Back...	Get, List, Update, Create, Import, D...

Create pipeline for moving the data from AWS s3 bucket to landing folder in azure & add trigger to run on daily basis:

1). In Azure Data Factory, go to the author tab and clicked on “Linked services” for creating link service for connecting to AWS S3 (where data is stored by the third party device; Source of the data).

In this we'll use the credentials using Azure Key Vault where we stored secret key & secret key id. But before that we need to creat AKV linked service which we can utilise also for creating linked

service for connecting to ADLS. We'll test connection, if it's successful then we can see it at the bottom right corner "Connection successful"

2). Connect to Azure Data Lake Gen2 Storage

In a similar way we'll create linked service for making connection of Azure Data Factory with Azure Data Lake

The screenshot shows the Microsoft Azure Data Factory interface. On the left, there is a sidebar with various options like General, Connections, and Source control. The main area is titled 'Linked services' and shows a list of existing linked services: 'AzureKeyVault1' (Azure Key Vault) and 'S3_LS' (Amazon S3). A 'New' button is visible. On the right, a form is open for creating a new linked service for 'Azure Data Lake Storage Gen2'. The 'Name' field is set to 'Azure_Data_Lake_LS'. The 'Description' field contains the text 'This is link service to storage account'. Under 'Connect via integration runtime', 'AutoResolveIntegrationRuntime' is selected. The 'Authentication type' is set to 'Account key'. Under 'Account selection method', 'From Azure subscription' is selected, and 'Data Engineering (e60d5a57-13f1-4a20-bb6f-327c27fe06eb)' is chosen from the dropdown. The 'Storage account name' is set to 'dataengineerproject1'. The 'Test connection' button is highlighted with a green checkmark and the text 'Connection successful'. At the bottom, there are 'Create', 'Back', 'Test connection', and 'Cancel' buttons.

3). Create pipeline for pulling the data from AWS S3 bucket to Azure Data Lake Gen2 . We'll name the pipeline "Ingestion_S3_to_ADLS". For this we'll use Copy activity. We need to provide source & sink. When we click on source (at this moment we don't have any dataset available so click on "+" for creating the dataset and the source side. We know that Amazon S3 as our source, so we have to create the dataset for Amazon S3, so select S3 and file format to JSON. Then we have to select the link service, which we had created so select that. When we click on browse-it will allow us to select the root folder that is in S3 bucket, select root folder. In our case, data folder is in format YYYY-MM-DD hierarchy. If folder path is changing on daily basis and we need to run this pipeline on daily basis as well then we have to make a pipeline in such a way that'll dynamically pull the data from dynamically selected folders. If we want to make a pipeline that'll dynamically pull the data from dynamically selected foldersPath. We can do this by parametrising the dataset Go to the parameter & give the name "folderPath" as dynamic parameter —> go to the connection —> Directory & add dynamic content—>select parameter which is folderPath

Microsoft Azure | Data Factory > dataengineerproject1

Tell us what you think, share your feedback in this quick survey about Azure Data Factory

Factory Resources

- Pipelines
 - Ingestion_S3_to_A... (1)
 - Change data capture (preview) (0)
 - Datasets (0)
 - Data flows (0)
 - Power Query (0)

Activities

- copy
- Move & transform
 - Copy data

Set properties

Name: S3_JSON_DS

Linked service: S3_LS

File path: iotdataproject1 / [Directory] / [File name]

Import schema: From connection/store (radio button selected)

Advanced: Open this dataset for more advanced configuration with parameterization.

OK Back Cancel

Microsoft Azure | Data Factory > dataengineerproject1

Tell us what you think, share your feedback in this quick survey about Azure Data Factory

Factory Resources

- Pipelines
 - Ingestion_S3_to_ADL... (1)
 - Change data capture (preview) (0)
- Datasets
 - S3_JSON_DS (1)
 - Data flows (0)
 - Power Query (0)

Properties

General Related (1)

Name: S3_JSON_DS

Description:

Annotations

Preview experience: Off

Microsoft Azure | Data Factory > dataengineerproject1

Tell us what you think, share your feedback in this quick survey about Azure Data Factory

Factory Resources

- Pipelines
 - Ingestion_S3_to_ADL... (1)
 - Change data capture (preview) (0)
- Datasets
 - S3_JSON_DS (1)
 - Data flows (0)
 - Power Query (0)

Pipeline expression builder

Add dynamic content below using any combination of expressions, functions and system variables.

@dataset().FolderPath

Clear contents

Parameters Functions

FolderPath

Connection Schema Parameters

Linked service: S3_LS

Test connection Edit + New Learn more

File path: iotdataproject1 / [Directory] / Add dynamic [Alt+Shift+]

Compression type: None

Encoding: Default(UTF-8)

OK Cancel

Whole path will be created dynamically based on parameter value which we pass in folder path. Now we'll go the pipeline and we can see that in source we're getting the option of folderPath. Now we know that we have to create the folder path in YYYY-MM-DD format so for that we'll use formatDate function. For ex. If we give this function:

The screenshot shows the Microsoft Azure Data Factory Pipeline expression builder. In the top right, there's a search bar and a user profile. Below it, the pipeline name 'Ingestion_S3_to_A...'. The main area has tabs for 'Activities' (selected), 'Move & transform', and 'Copy data'. A preview window shows the expression: `@concat(formatDateTime.UtcNow(), 'yyyy', '/', formatDateTime.UtcNow(), 'MM'), '/', formatDateTime.UtcNow(), 'dd')`. Below this, there are tabs for 'Parameters', 'System variables', 'Functions', and 'Variables', with a search bar and a '+' button.

It'll extract yyyy MM & DD from the date. We'll use this to create our dynamic folder and then we'll use the concat function to concatenate all of them.

The screenshot shows the Microsoft Azure Data Factory Pipeline activities. The pipeline 'Ingestion_S3_to_A...' is selected. The 'Source' tab is active for the 'Copy data' activity. The 'Source dataset' is set to 'S3_JSON_DS'. Under 'Dataset properties', 'folderPath' is configured with the expression: `@concat(formatDateTime.UtcNow(), 'yyyy', '/', formatDateTime.UtcNow(), 'MM'), '/', formatDateTime.UtcNow(), 'dd')`. The 'File path type' is set to 'Wildcard file path' with the value 'iotdataproject1 / Wildcard folder path / *.json'. At the bottom, there are 'Start time (UTC)' and 'End time (UTC)' fields.

In

The screenshot shows the Microsoft Azure Data Factory interface. In the top navigation bar, it says "Microsoft Azure | Data Factory > dataengineerproject1". The main area is titled "Pipeline expression builder". A message at the top says "Tell us what you think, share your feedback in this quick survey about Azure Data Factory". Below this, there's a search bar and a "Publish all" button.

The left sidebar shows "Factory Resources" with sections for Pipelines (1 item), Datasets (2 items), and others. The "Azure_DL_DS" dataset is selected.

The main workspace shows a "JSON" dataset named "Azure_DL_DS". The "Parameters" tab is active, showing a parameter named "landingFolder" with the expression "@dataset().landingFolder".

Below the parameters, there are tabs for "Connection", "Schema", and "Parameters". Under "Connection", the "Linked service" is set to "Azure_Data_Lake_LS". Under "Parameters", the "landingFolder" parameter is defined with a value of "@dataset().landingFolder".

A modal window is open in the center, titled "Sink dataset *". It has tabs for "General", "Source", "Sink", and "Mapping". The "Sink" tab is selected. It shows a "Sink dataset *" dropdown with a "New" button. At the bottom of the modal are "OK", "Back", and "Cancel" buttons.

This screenshot shows the "Properties" pane for the "Azure_DL_DS" dataset. The "General" tab is selected, showing the "Name" field set to "Azure_DL_DS". There are also fields for "Description" and "Annotations".

The left sidebar shows the same "Factory Resources" structure as the previous screenshot.

The main workspace shows the "Azure_DL_DS" dataset with its "Parameters" tab selected. It lists a single parameter "landingFolder" of type String with a default value of "Value".

the source, if we have to pull multiple files then we'll select wildcard file path. It'll pull all files which are in json format

Now go to the “sink” (which is ADLS)→ create sink dataset→ in new dataset search for Azure Data Lake Storage Gen2 → select the file format JSON → select the name & select link service → create folder structure→ just select input folder for dynamic creation of folder (same procedure as we followed for source)

So whatever value we give to this dynamic parameter, our file path will be selected accordingly. For creating the folder structure in the similar pattern like YYYY-MM-DD, we'll use concat function. First we'll give the path as 'landing'/yyyy/mm/dd

Properties

General

- Name * Azure_DL_DS
- Description
- Annotations

Connection

- Linked service * Azure_Data_Lake_LS
- Test connection
- File path * input / @dataset().landingFolder / File name
- Compression type None
- Encoding Default(UTF-8)

Pipeline expression builder

Add dynamic content below using any combination of **expressions**, **functions** and **system variables**.

```
@concat('landing', formatDateTime.UtcNow(), 'yyyy', '/', formatDateTime.UtcNow(), 'MM', '/', formatDateTime.UtcNow(), 'dd')
```

Clear contents

Parameters **System variables** **Functions** **Variables**

Search

General **Source** **Sink** **Mapping**

Sink dataset * sink

Copy behavior

Max concurrent connections

Block size (MB)

OK **Cancel**

Now, our pipeline is almost ready, so we'll click on "debug". It'll only pull the data from the current date. We can see pipeline run is successful. We can check whether our data have been copied from the S3 bucket to Azure data lake storage or not. We can see that data has been copied from the AWS

The screenshot shows the Microsoft Azure Data Factory interface. On the left, the 'Factory Resources' sidebar lists 'Pipelines' (Ingestion_S3_to_ADLS), 'Datasets' (Azure_DL_DS, S3_JSON_DS), and 'Data flows'. The main workspace displays a pipeline named 'Ingestion_S3_to_ADLS' with three stages: 'S3_JSON_DS' (selected), 'Copy data1' (highlighted with a green checkmark), and 'Azure_DL_DS'. Below the pipeline, the 'Output' tab of the pipeline run details shows a successful run with ID 6f88e6e4-4c6b-401f-9f0d-34ccb74d1a2d, starting at 2023-02-23T21:37:28.6514 and duration 0:00:14. A status table indicates 'Copy data1' was a 'Copy data' type activity that succeeded.

Name	Type	Run start	Duration	Status
Copy data1	Copy data	2023-02-23T21:37:28.6514	0:00:14	Succeeded

S3 bucket to Azure Data lake get 2 container

The screenshot shows the Microsoft Azure Storage Explorer interface. The left sidebar shows 'input' (Container) selected. The main area displays two containers: 'input' and 'landing'. Under 'input', there is one blob named 'Customer_Valid.json'. The 'landing' container is selected, and its properties show it was created on 2/23/2023 at 9:37:40 PM with a size of 85.25 KiB and an 'Available' lease state.

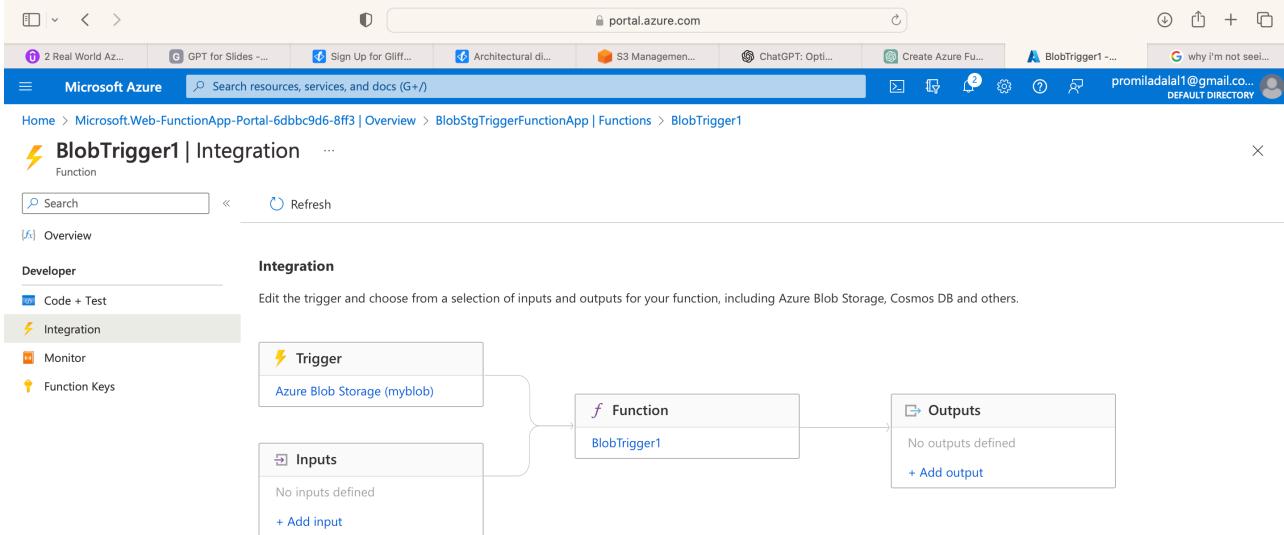
Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
Customer_Valid.json	2/23/2023, 9:37:40 PM	Hot (Inferred)		Block blob	85.25 KiB	Available

Create Azure Function with blob storage logic:

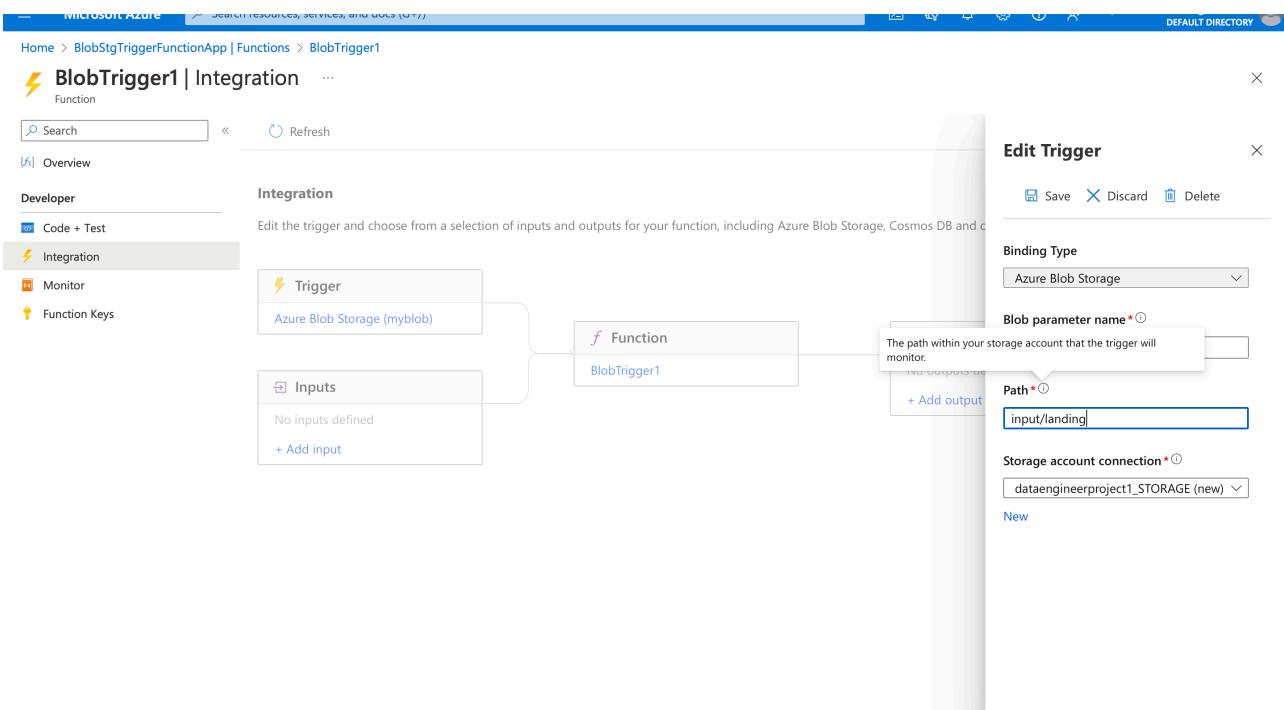
To create the azure function which'll validate the json & it'll going to run as soon as file is landed up in landing folder from amazon s3, it'll be triggered up. This ADF'll be run on daily basis & this function'll decide whether incoming file should move to staging folder or rejected folder.

1). Creation of Function app

2). Creation of blob storage trigger function:



3). Setting up trigger path



4). Create folders for rejected & staging folders

The screenshot shows the Azure Functions blade for a function named 'BlobTrigger1'. The 'Integration' tab is selected. The trigger section shows 'Azure Blob Storage (myblob)'. The function section shows 'BlobTrigger1'. The outputs section lists 'Azure Blob Storage (rejectedFolder)' and 'Azure Blob Storage (stagingFolder)'. A diagram illustrates the flow: Trigger (myblob) → Function (BlobTrigger1) → Outputs (rejectedFolder, stagingFolder).

5). code+test

Click on Test/run & mention the folder path (i.e., landing in our case)

The screenshot shows the Azure Functions blade for 'blobstoragetriggerapp1' under 'Code + Test'. The 'index.js' file contains the following code:

```
module.exports = async function (context, myBlob) {
    context.log("JavaScript blob trigger function processed blob with Blo
    bName: " + myBlob.name);
    var result = true;
    try {
        context.log(myBlob.toString());
        JSON.parse(myBlob.toString().trim().replace("\n", " "));
    } catch(exception){
        context.log(exception);
        result =false;
    }
    if(result){
        context.bindings.stagingFolder = myBlob.toString();
        context.log("*****File Copied to Staging Folder Success");
    } else{
        context.bindings.rejectedFolder = myBlob.toString();
        context.log("*****Inavlid JSON File Copied to Rejected
        Folder");
    }
}
```

The 'Input' tab is selected, showing the input path 'input/landing/2023/02/26/Customer_Valid.json'. The 'Output' tab is also visible. At the bottom, there are 'Run' and 'Close' buttons.

The screenshot shows the Microsoft Azure Functions portal. The URL is <https://portal.azure.com/#view/WebsitesExtension/FunctionMenuBlade/-/code/re...>. The left sidebar has 'Code + Test' selected. The main area shows the code for blobstoragetriggerapp1 \ BlobTrigger1 \ index.js:

```

1 module.exports = async function (context, myBlob) {
2   context.log("JavaScript blob trigger function processed blob \n Blob:");
3   context.log("*****Azure Function Started*****");
4   var result = true;
5   try{
6     context.log(myBlob.toString());
7     JSON.parse(myBlob.toString().trim().replace("\n", ' '));
8   }catch(exception){
9     context.log(exception);
10    result = false;
11  }
12  if(result){
13    context.bindings.stagingFolder = myBlob.toString();
14    context.log("*****File Copied to Staging Folder Successfully*****");
15  } else{
16    context.bindings.rejectedFolder = myBlob.toString();
17    context.log("*****Inavlid JSON File Copied to Rejected Folder Successfully*****");
18  }
19}

```

The logs pane shows the message: "Connected! You are now viewing logs of Function runs in the current Code + Test panel. To see all the logs for this Function, please go to 'Monitor' from the Function menu."

We can see that file copied to staging folder successfully

The screenshot shows the Microsoft Azure Storage Explorer. The URL is <https://portal.azure.com/#view/WebsitesExtension/FunctionMenuBlade/-/code/re...>. The left sidebar has 'Container' selected. The main area shows the 'input' container details and a list of blobs:

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
[..]						...
29b0e08f-7467-4589-9efc-a4538c...	2/26/2023, 11:26:55 ...	Hot (Inferred)		Block blob	85.25 KiB	Available
3f132ce7-f143-4456-87ba-be27c...	2/26/2023, 4:55:49 PM	Hot (Inferred)		Block blob	85.25 KiB	Available
c55dbffd-ab54-4ca0-bf86-54bb4...	2/26/2023, 4:30:37 PM	Hot (Inferred)		Block blob	85.25 KiB	Available

Add JSON format in the Azure function

-in 'Path' add .json for both stagingFolder & rejectedFolder outputs

The screenshot shows the Microsoft Azure Functions portal. The URL is <https://portal.azure.com/#view/WebsitesExtension/FunctionMenuBlade/-/code/re...>. The left sidebar has 'Integration' selected. The main area shows the integration setup for BlobTrigger1:

- Trigger:** Azure Blob Storage (myBlob)
- Function:** BlobTrigger1
- Outputs:**
 - Azure Blob Storage (stagingFolder)
 - Azure Blob Storage (rejectedFolder)

The 'Edit Output' dialog is open for the stagingFolder output:

- Binding Type:** Azure Blob Storage
- Blob parameter name:** stagingFolder
- Path:** input/staging/{rand-guid}.json
- Storage account connection:** dataengineerproject1_STORAGE

Create Azure SQL Server & Database

We'll use Azure data factory for creating a pipeline that will use copy activity for moving data from Azure data lake storage Gen2 to Azure SQL db. So for this purpose we have created SQL database & SQL server.

Create Pipeline for moving data from ADLS to Azure SQL DB:

- Most of the steps are similar to the steps we utilised for creating pipeline for copying data from amazon s3 bucket to ADLS (with few exception)

End-to-end project execution:

Pipeline 1: for copying data from amazon s3 to ADLS (succeeded)

The screenshot shows the Microsoft Azure Data Factory interface. On the left, the 'Factory Resources' sidebar lists 'Pipelines' (Ingestion_S3_to_ADLS, transfer_adls_to_sql), 'Datasets' (adls_to_sql, Azure_datalake_storage_DS, AzureSqlTable1, s3_json_DS), and other sections like 'Data flows' and 'Power Query'. The main workspace displays the 'Ingestion_S3_to_ADLS' pipeline. It contains a single 'Copy' activity named 'Copy data1'. The 'Output' tab is selected, showing a table with one row: Name (Copy data1), Type (Copy data), Run start (2023-02-26T17:21:57.7104), Duration (00:00:11), and Status (Succeeded). The top right corner shows the email 'promiladala1@gmail.com' and 'DEFAULT DIRECTORY'.

Pipeline 2: for copying data from adls to SQL DB

The screenshot shows the Microsoft Azure Data Factory interface, similar to the previous one but with a different pipeline selected. The 'Factory Resources' sidebar shows the same resources. The main workspace displays the 'transfer_adls_to_sql' pipeline. It also contains a single 'Copy' activity named 'Copy data1'. The 'Output' tab is selected, showing a table with one row: Name (Copy data1), Type (Copy data), Run start (2023-02-26T16:37:19.7680), Duration (00:00:13), and Status (Succeeded). The top right corner shows the email 'promiladala1@gmail.com' and 'DEFAULT DIRECTORY'.

We can see that data has been copied to Azure SQL database

The screenshot shows the Microsoft Azure portal interface for a SQL database named 'Project1SQLDB'. The left sidebar contains navigation links for Overview, Activity log, Tags, Diagnose and solve problems, Getting started, and the currently selected 'Query editor (preview)'. The main area displays a 'Query 1' window with the following details:

- Toolbar:** Includes 'Run', 'Cancel query', 'Save query', 'Export data as', and 'Show only Editor' buttons.
- Query Editor:** A code editor containing the SQL query: `1 select * from [dbo].[VehicleData]`.
- Results:** A table showing the results of the query. The table has columns: vehicleID, latitude, longitude, City, temperature, and speed. There are three rows of data, all with the same values: vehicleID -78, latitude -78, longitude -78, City Mérignac, temperature 191, and speed 191.
- Status Bar:** Shows the message 'Query succeeded | 1s'.