**Creating Data Pipeline Using Azure**

The first and foremost step to create a data pipeline in Azure is to sign in to the portal with your credentials.

**Creating Resource Group**

Once you have signed in to the portal, click on the resource group and create a resource group where you will be running all the services required to create a pipeline. In this case, I gave the name of the resource group as “resources01” while rest of the settings were set to default.

A screenshot of a computer

Description automatically generated

**Creating Storage Account**

First, navigate to "resources01" and click on "Create," which will take you to the Azure marketplace. In the search bar, type "storage account by Azure" and select it to begin creating a storage account. The resource group you created earlier, "resources01," will be automatically listed. Next, provide a globally unique name for the storage account; for this project, the name used is "storageproject0123." Choose **Locally-Redundant Storage (LRS)**, which offers a cost-effective solution for ensuring local redundancy within a single region. Then, enable the **Hierarchical Namespace** feature, which transforms the Azure Data Lake Storage Gen2 into a fully managed hierarchical file system, providing better directory-based management, enhanced performance, and advanced features such as ACLs and atomic file operations. After reviewing all configurations, click "Create" to create the storage account.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Creating Containers under the Storage Account**

For this project, we will be creating five containers: raw-containers, orders, customers, order-customer-joined, aggregated-data. After we have created all the containers, we will upload the two Json format dataset at raw-container. This will be our data source. We will further discuss about why we require five containers as we proceed to create a pipeline.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Creating Data Factory**

After setting up the storage account, the next step is to create an Azure Data Factory within the same resource group. Navigate to **resource01** and click **Create**. This action will open the Azure Marketplace. Search for the **Data Factory** service and select it. Then, click **Create** again. Provide a name for the Data Factory—for instance, "projectdf0123"—and proceed with the review. Once everything is verified, click **Create** to complete the process.

A screenshot of a computer

Description automatically generated

A screenshot of a software

Description automatically generated

Now we have successfully created a storage account and a Data factory under a resource group.

A screenshot of a computer

Description automatically generated

**Launching a Data Factory Studio and creating connector**

In the next step, go to projectdf0123 and click on launch studio.

A screenshot of a computer

Description automatically generated

Before creating any pipelines, it is essential to establish a connection between the Data Factory and the Data Lake. This connection allows seamless access to source data and facilitates storing output in the Data Lake.

To set this up, first, navigate to the **Manage** section in Azure Data Factory and select **Linked Services**, which acts as the connector between services. Click **New**, then choose **Azure Data Lake Storage Gen2** as the data source. Proceed to configure the connection accordingly.

A screenshot of a computer

Description automatically generated

**Creating Pipelines**

The first pipeline we will create is called pipeline-metadata, designed to retrieve metadata for both the orders.json and customers.json files. To start, drag the Get Metadata activity from the toolbox into the pipeline canvas. Configure it by selecting the raw-container, which contains both datasets, as the source. Specify the file path within the container for the activity to access the datasets. Next, select the field list child item to retrieve the child item from the dataset. Once configured, validate the pipeline setup and use the Debug feature to identify and resolve any errors before proceeding further.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Next, drag the **For Each** activity from the toolbox and drop it onto the canvas. Connect the **For Each** activity to the **Get Metadata** activity by linking the **success** output of **Get Metadata** to the **For Each** activity. Afterward, navigate to the **Settings** tab within the **For Each** activity. Enable the **Sequential** option, then add a dynamic content expression. Select **Get Metadata -> childItems**, and Azure Data Factory will automatically generate the appropriate expression for you.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Within the **For Each** activity, navigate to the **Activities** section and add an **If Condition** activity. Configure the condition with the expression @contains(item().name, 'orders.json') to check if the current child item’s name includes orders.json. Under the **If Condition**, define two activities: for the **TRUE** branch, set up a **Copy Data** activity to copy the orders.json file to the **"orders"** container; for the **FALSE** branch, configure another **Copy Data** activity to copy the customers.json file to the **"customers"** container. This ensures that files are dynamically routed to the correct destinations based on their names. However, when you store the data on the output container, I changed the the file to .CSV format for easy readability and aggregation.

Lastly, validate and debug the code to check if there is any errors.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Check both orders and customers containers if the data was copy to the containers respectively from the conditions we created.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Creating Data flow**

1. **Dataflow1**

Next, we will create a Data Flow to join two datasets based on a common column and store the resulting data in a joined container. Start by navigating to Data Flow, provide a name for the data flow, and add a new source. Select the datasets from the orders and customers containers as sources. Before joining the datasets, navigate to the Projection tab for each source and update the data types of all columns as needed. Once both sources are configured, click the plus (+) icon and add a Join transformation. Specify the join condition using the common column between the datasets. After joining, add a Sink transformation to define the storage location for the joined dataset. Set the destination as the order-customer-joined container in the storage account.

After that validate and debug, once that’s done, create another pipeline-joined and drag the dataflow to pipeline-joined and the validate and debug the pipeline. And then check the orders-customers-joined containers to check see if the joined data was in the container.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A close-up of a computer screen

Description automatically generated

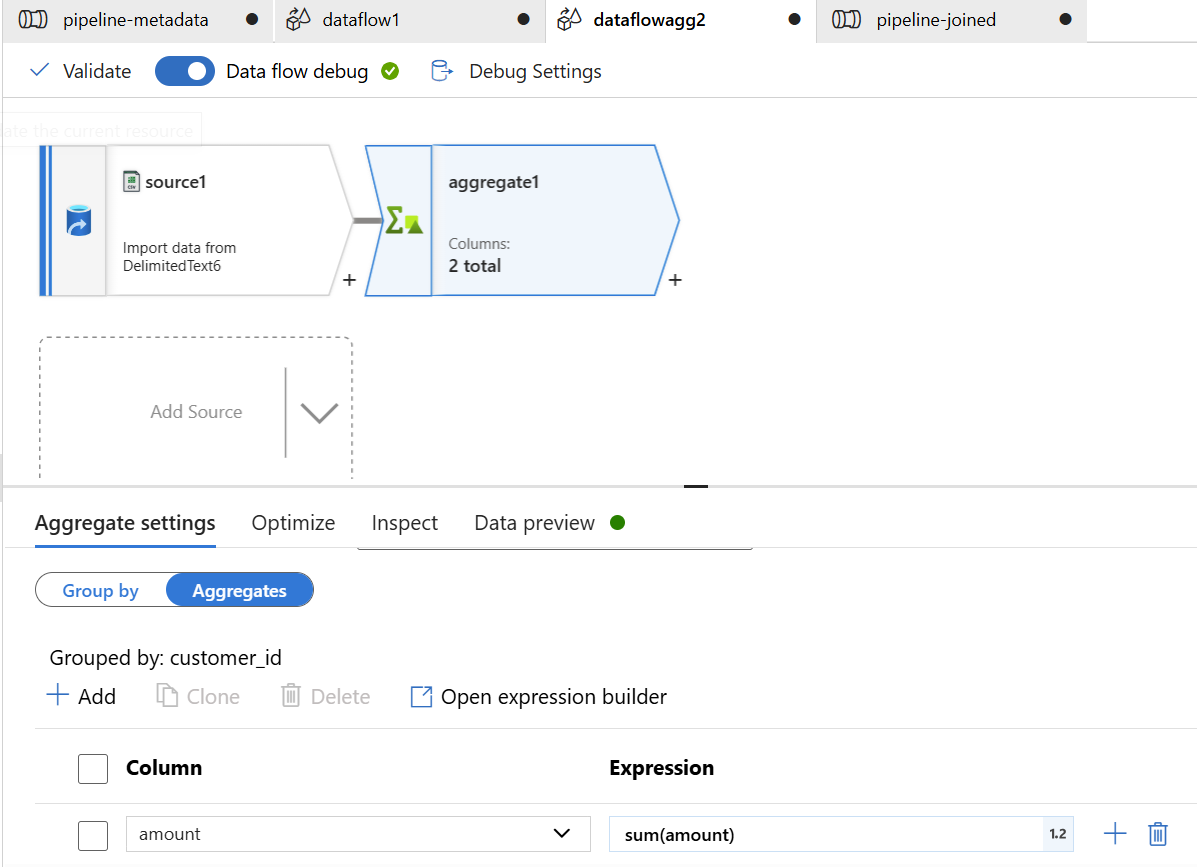
1. **dataflowagg2**

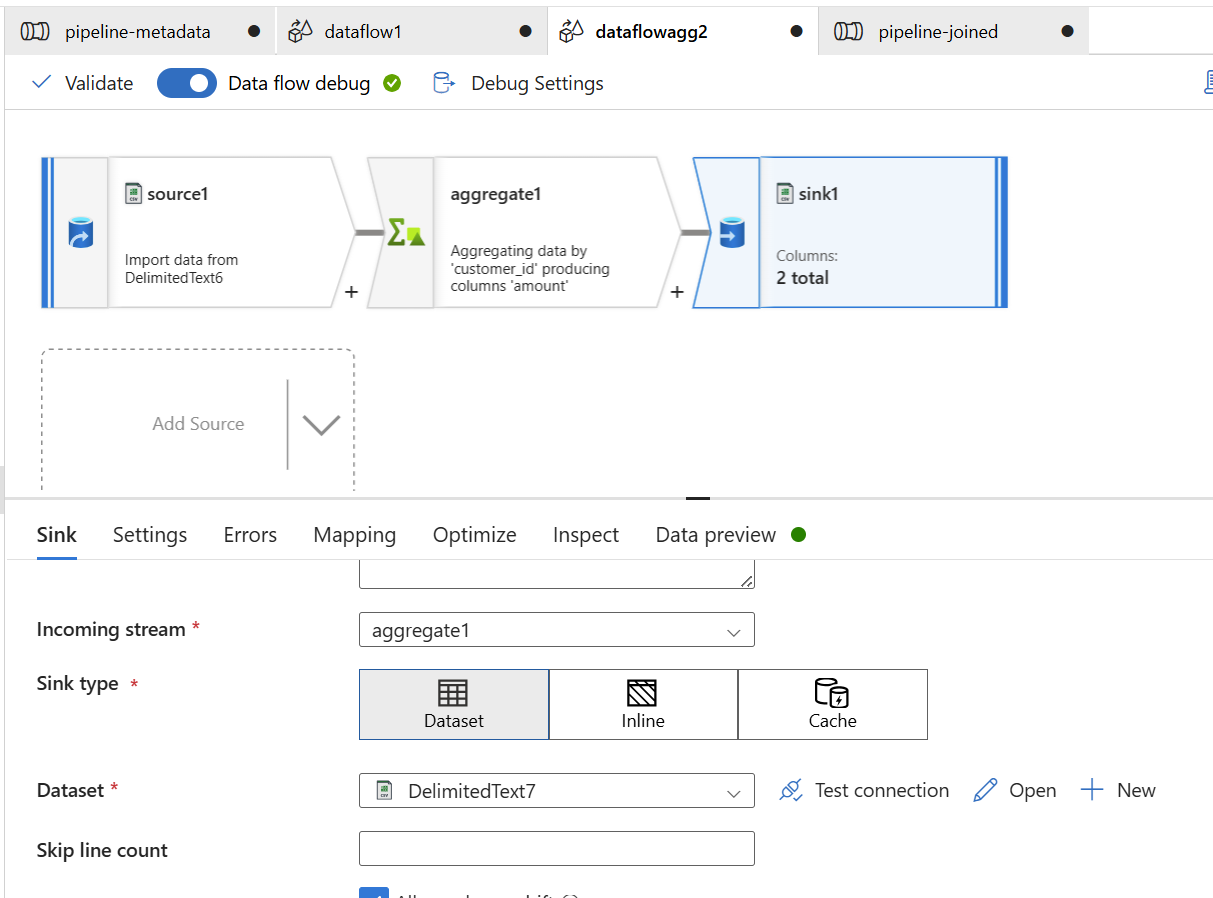
Next, add a source from the orders-customers-joined dataset and adjust the data types for the columns as needed. Then, click the plus (+) icon and select the Aggregation option. In the aggregation settings, group the data by customerId and calculate the sum of salary. Once the aggregation is complete, add a Sink transformation and specify the path to the aggregated-data container where the aggregated data will be stored.

After configuring the data flow, validate and debug it to check for any errors. Then, create a new pipeline called pipeline-aggregated, drag the dataflowagg2 into this pipeline, and validate and debug it again to ensure there are no issues. And check the aggregated-data container if the output is stored properly.

A screenshot of a computer

Description automatically generated





A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Finally, after validating and debugging all executions, publish the work to ensure that all changes are saved and deployed. Once the pipelines are published, clone **pipeline-joined** and **pipeline-aggregated** to create a flow. After creating the flow, add a trigger to the flow to automate its execution according to the defined schedule or event-based condition. This ensures that the entire data processing workflow runs smoothly and on time.

A screenshot of a computer

Description automatically generated

Lastly, to avoid unnecessary charges, make sure to remove all the resources that are no longer needed. This includes deleting any unused pipelines, data flows, datasets, triggers, and storage accounts that were created during the process. You can do this by navigating to the **Azure Portal**, selecting the resources in your resource group, and deleting them. This helps ensure that you are not incurring additional costs for resources that are not in use.