Disclaimer: These slides are copyrighted and strictly for personal use only

- This document is reserved for people enrolled into the Ultimate Big Data Masters Program (Cloud Focused) by Sumit Sir

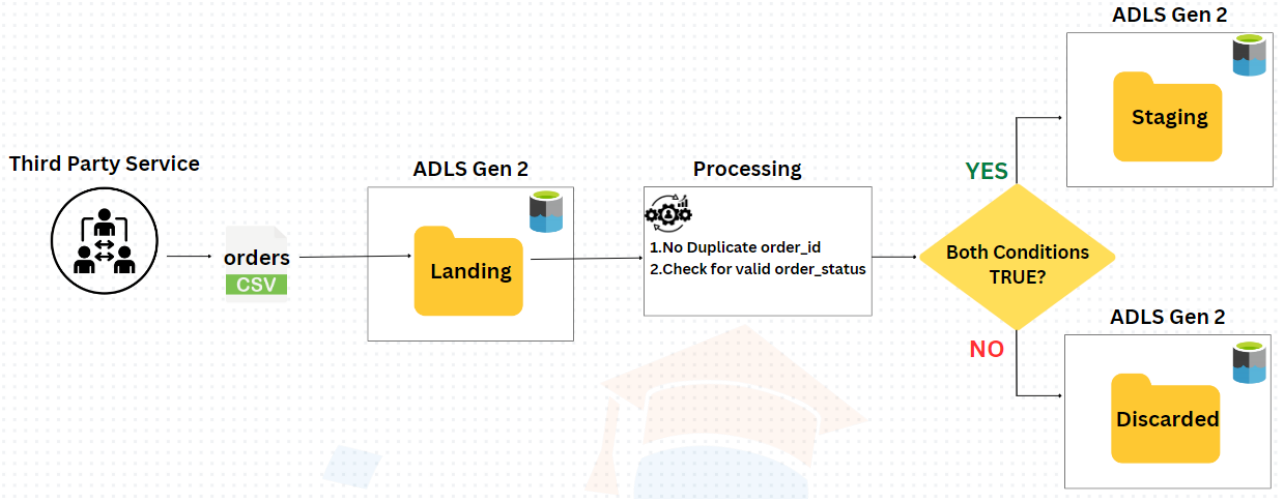
- **Please do not share this document,** it is intended for personal use and exam preparation only, thank you.

- If you've obtained these slides for free on a website that is not the course's website, please reach out to legal@trendytech.in . Thanks!

- All the Best and Happy Learning!

Developing a Data Pipeline

Project Use-case



A third party service drops a file named orders.csv in the landing folder.

Requirement:

As soon as file arrives in landing folder, perform the following checks

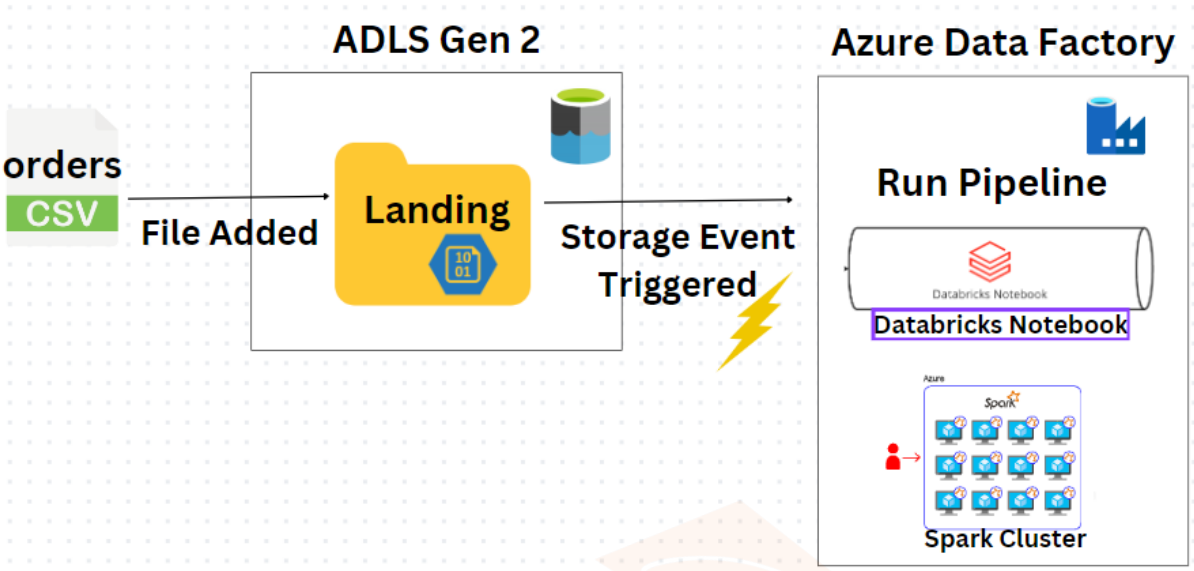
1. Check for duplicate order_id in order_status
2. Check for valid orders_status

If both the conditions are true then move the file to the staging folder else move it to the discarded folder.

Note: In future, if the list of valid order_status changes, then we should dynamically be able to incorporate the changes.

Services required to Implement the Solution :

As soon as the data is added to the storage
|
Run the Pipeline (Data Factory - trigger is a Storage Event)
|
Databricks Notebook executes the spark code to perform the necessary checks.



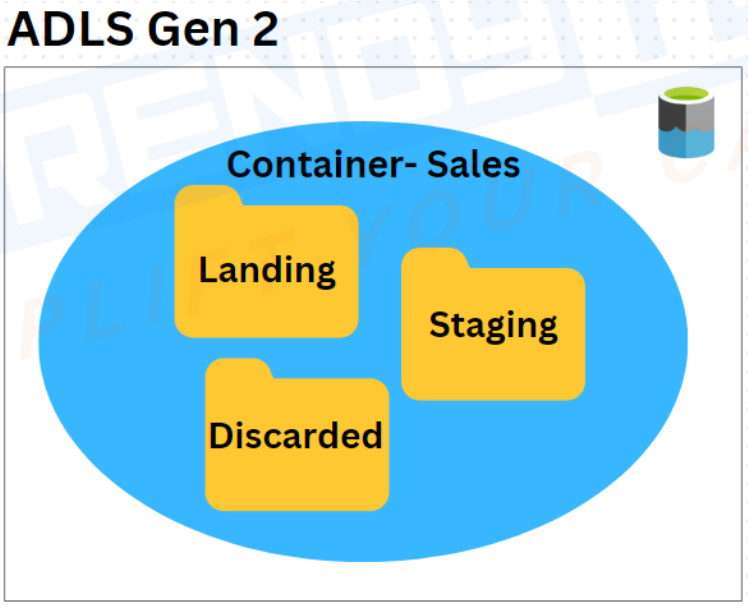
Building the Data Pipeline for the Project

Creating the necessary Resources :

1. **Storage Account**

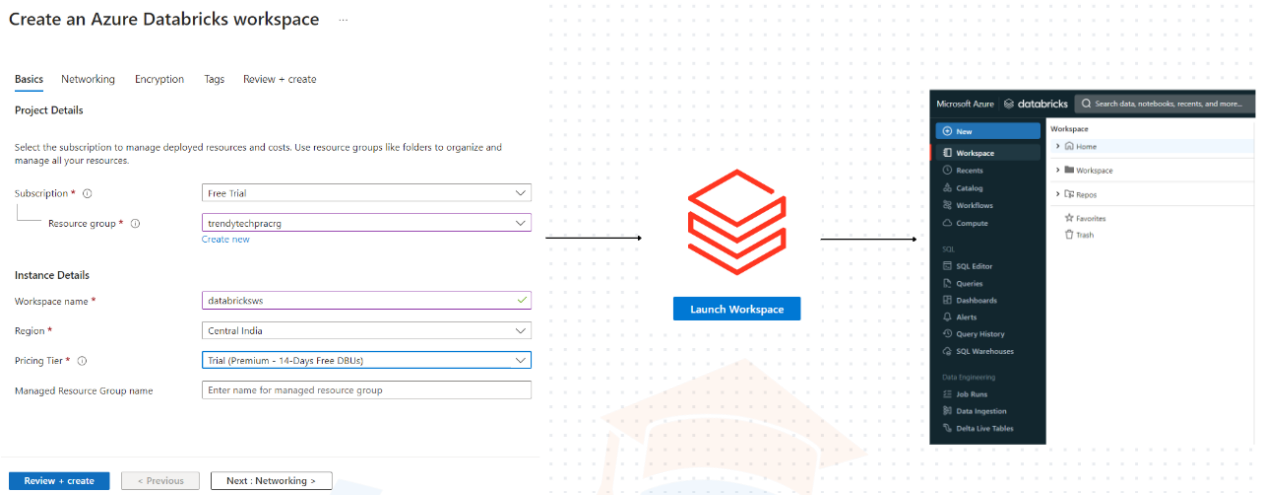
   -> Create a storage account with enabled hierarchical namespace

   -> Create a container named “sales”. Inside the sales container create directories as: landing, staging, discarded



2. Databricks

-> Create a Databricks Workspace



3. Data Factory

-> Create a Datafactory Service

-> Two components that has to be connected to Azure Data Factory

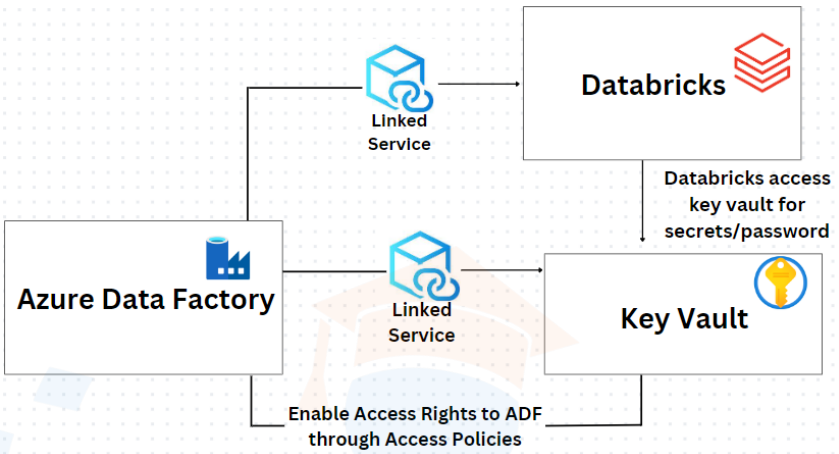
1. **ADLS Gen2 (Storage)**
2. **Databricks (Compute)**

-> During this process, we will use a **Key Vault** to store any of the passwords/ secret keys.

-> **Linked Services** need to be created between the datafactory and the 2 components - ADLS Gen2 & Databricks.

-During the creation of linked service for databricks, we cannot directly add the access token, instead store the access token in key vault and use that name by moving the option to “Azure key vault”

-While trying to connect with key vault in linked service of databricks, a key vault linked service should also be created and to create a linked service to key vault we need to allow datafactory to access the key vault by enabling it in access policies.



Totally, 3 linked services will be created ->

1. **ADLS GEN2 | 2. Databricks | 3. Key Vault**

Creating Azure SQL DB : Lookup Table to maintain the List of valid order_status

ON_HOLD

PAYMENT_REVIEW

PROCESSING

CLOSED

SUSPECTED_FRAUD

COMPLETE

PENDING

CANCELLED

PENDING_PAYMENT

To keep the above list in a lookup table, Azure SQL database is required along with the following database details

Database-Name

Server-Name

Username-Name

Password (Sql-password is stored in key vault)

-> Once the database is ready, allow the Azure services(like Databricks) to access this database by enabling the client access through firewall settings.

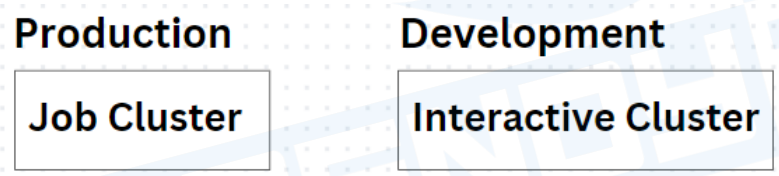
-> Create a **lookup table** - valid_order_status

```
create table valid_order_status(status_name varchar(50))
```

-> Inserting values into the table

```
insert into valid_order_status
values('ON_HOLD'),('PAYMENT_REVIEW'),('PROCESSING'),('CLOSED'),('SUSPECTED_FRAUD'),('COMPLETE'),('PENDING'),('CANCELED'),('PENDING_PAYMENT')select * from valid_order_status
```

Developing Logic using Interactive Databricks Cluster



-> Create an interactive cluster in databricks that will execute the databricks notebook.

```
dbutils.fs.mount(source='wasbs://sales@trendtechsa.blob.core.windows.net',
mount_point = '/mnt/sales',

extra_configs={'fs.azure.account.key.trendtechsa.blob.core.windows.net':'MQsEdge/QEAbx95+IIlFcujt5AnU7Q9ErfjTiDEEXkBv4jHFNfsTEozFawfr8KUUrkd3qf9/LSDZ+AStDaWbXw==''})
```

sales@trendtechsa : “sales” - Container name
“trendtechsa” - Storage account name

fs.azure.account.key.trendtechsa.blob.core.windows.net :
“trendtechsa” - **Storage Account Name**

Perform the following in the Databricks Notebook :

1. Create a **Mount Point**.

2. Create a **Dataframe.**

3. Load the orders.csv file into a Dataframe and perform validation(checking for duplicates order_id in order_status)

4. Establish a connection with Azure SQL Server from databricks. During this process, a secret scope has to be created so that databricks can access the sql-password which is stored in the key vault.

Note:

In the case of Datafactory, it can directly access the key vault by creating a linked service but for databricks, a secret scope is needed to access the key vault.

While creating the secret scope, edit the url till .net and add "#secrets/createScope"

https://<databricks-instance>#secrets/createScope

Eg:
https://adb-8497524354556085.5.azuredatabricks.net/#secrets/createScope

5. First Validation Check - If there are any duplicate order_ids. Once the validation is successful, create a **Spark Table** from Dataframe

```
errorFlg = False

ordersCount = ordersDf.count()
distinctOrdersCount = ordersDf.select('order_id').distinct().count()

if ordersCount != distinctOrdersCount
        errorFlg = True
if errorFlg :
        dbutils.fs.mv('/mnt/sales/landing/orders.csv','/mnt/sales/discarded')
        dbutil.notebook.exit('{"errorFlg": "true", "errorMsg": "Orderid is
        repeated"}')

ordersDf.createOrReplaceTempView("orders")
```

6. Then the second validation(check for valid orders_status) is evaluated.

For this validation, we need to connect to Azure SQL DB from the databricks notebook. The following details are required for connecting to the SQL DB

dbServer
dbPort
dbName
dbUser
dbPassword
databricksScope

```
connectionUrl =
'jdbc:sqlserver://{}.database.windows.net:{};database={};user={};'.
format(dbServer, dbPort, dbName, dbUser)

dbPassword = dbutils.secrets.get(scope = databricksScope,
key='sql-password')

connectionProperties = {
        'password' : dbPassword,
        'driver': 'com.microsoft.sqlserver.jdbc.SQLServerDriver'
}
```

Note: Databricks cannot directly access the key vault (where the database password is stored) directly. A secret scope needs to be created in Databricks in order to access the database password present in the key vault.

In the URL of the Databricks workspace, add the following to **Create a secret scope (databricksScope) :**
add #secrets/createScope

Ex:
https://adb-5015943971662126.6.azuredatabricks.net/onboarding?o=5015943971662126#secret/createScope

Connecting to the SQL Database for order_status

```
validStatusDf = spark.read.jdbc(url = connectionUrl, table =
'dbo.valid_order_status', properties = connectionProperties
```

7. Once the second validation is also successful, notebook will exit by giving the following message and the file is moved to the staging folder.

Notebook exited: {"errorflag:"false","errorMsg:"All Good"}

Creating a Pipeline in Azure Data Factory (Automating the pipeline execution)

-> Add the **Databricks Notebook** to the pipeline.

-> In the Settings tab, Select the Notebook to execute

-> Add Trigger -> Type (Storage Event Trigger) -> Storage Account Name -> Container Name (Ex: Sales) -> Path Begins with (Ex: Landing) -> Path Ends with(Ex: orders.csv) -> Continue.

-> **Monitor** tab gives an overview of all the triggered pipelines

-> **Process**

Once the data is uploaded in the specified container in the storage account, Storage event triggers the pipeline execution by launching the cluster and the databricks notebook will be executed. Once the execution is complete, the cluster deployed for the pipeline execution will be terminated.

Scenario 2

Problem Statement : Automatically trigger a pipeline execution when any file (not a specific hardcoded filename - orders.csv as in the previous example) is added to the Storage Account Container.

Solution :

Adopting a parameterized approach to dynamically read files, moving away from hardcoding specific filenames.

TriggerBody —> Pipeline —> databricks Notebook

[Code]

```
filename = dbutils.widgets.get('filename') # The filename will be taken from the pipeline
fnamewithoutExt = filename.split('.')[0]
print(filename)

OrdersDf = spark.read.csv('/mnt/sales/landing/{}'.format(filename), inferSchema = True , header = True)
```

Note:

-> Trigger(First point of contact) should have the capability to dynamically capture the filenames of the newly added files in the Storage Account and pass it to the Pipeline.

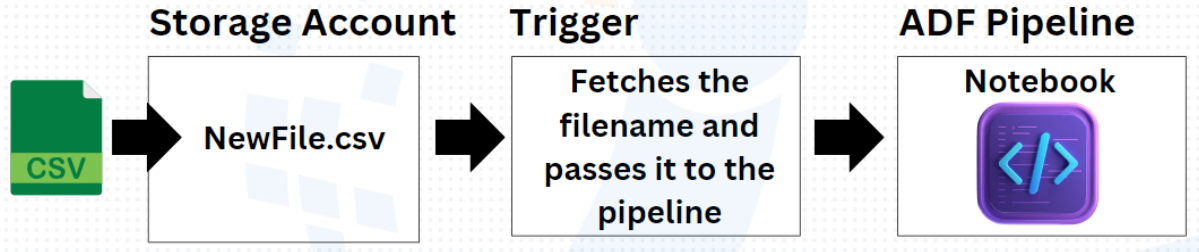
In this context, the file names are no longer hardcoded; instead, they are dynamically retrieved from the pipeline and stored in the "filename" variable. The DataFrame is then created using the “filename” variable.

-> We are retrieving the filename from the triggerbody (**@triggerBody().filename)**

-> Changes to be made in the Notebook

```
filename = dbutils.widgets.get('filename')

Fnamewithoutext = filename.split('.')[0]
```



Scenario 3

Problem Statement : Enhance the Pipeline by creating Generalized Pipelines with **Generic Mount Code**.

Solution : Logic of Mount Code should be developed in a way that if the mount point is already set, then no mounting should take place else the storage should be mounted.

```
alreadyMounted = False

for x in dbutils.fs.mounts() :

        if x.mountPoint == "/mnt/sales" :

                alreadyMounted = True

                break

        else

                alreadyMounted = False

print(alreadyMounted)
```

```
if not alreadyMounted :
        dbutils.fs.mount(
        source='wasbs://sales@trendtechsa.blob.core.windows.net',
        mount_point = '/mnt/sales',
        extra_configs={'fs.azure.account.key.trendtechsa.blob.core.windows.net
':'MQsEdge/QEAbx95+IIlFcujt5AnU7Q9ErfjTiDEEXkBv4jHFNfsTEozFawfr8KU
Urkd3qf9/LSDZ+AStDaWbXw==')
alreadyMounted = True
print("Mounting done successfully")
else
print("Already mounted")
```

Key Points : Enhancements to the Pipeline

1. **Dynamically picking the Filenames using the Parameterization approach**
2. **Generic code for mounting the storage**
3. **A secure way of accessing the Storage Account Key with Key Vault.**

Background Activity : Mimicking the scenario - data provided by Third Party

1. order_items data (Present in Amazon S3 in Json Format)

A third party service is adding the order_items file in Json format in a bucket in Amazon S3 (Need to mimic this scenario)

Requirement - Getting the data from Amazon S3 to ADLS Gen2 using Azure Data Factory

[You would need an AWS account for this activity. Create an Amazon S3 bucket and a folder to upload the order_items data file in Json format. An IAM role and secret key needs to be generated for external services to access this Json file in the S3 bucket.

If you need to access the file from Azure, The Secret Key and Id values should be added as secrets in the Key Vault.]

Accessing file in Amazon S3 Bucket from Azure

Create a Linked Service for **AWS S3 -** provide the details of S3 (Secret Key and ID present in the Azure Key Vault)

Tasks to be performed as soon as file (orders.csv) arrives in the Landing folder of ADLS Gen2

a. Get the data file- **order_items.json** from Amazon S3 and load it to ADLS Gen2 :

Create a Data Pipeline the ADF with source as Amazon S3(file in Json format) and the Sink as ADLS Gen2(file to be loaded in CSVformat)

b. Execute the Databricks Notebook

2. customers data

   A third party agency will be publishing this data in Azure SQL DB(Need to mimic this scenario)

Requirement - Getting the data from Azure SQL DB to ADLS Gen2 using Azure Data Factory

-Upload customers.csv file to a folder in ADLS Gen2

-Create an Azure SQL DB and a table with the schema of customers data file.

-Create two Linked Services ->

   a. Linked Service pointing to Azure SQL DB
   b. Linked Service pointing to Storage ADLS Gen2

-Create a Pipeline in Azure Data Factory with **Copy Data Activity**

   Source : customers.csv in ADLS Gen2

   Sink : customers table in Azure SQL DB

   (Schema Mapping - Ensure that the schema of Source and Sink are matching)

Project Requirement

1. **No.of orders placed by each customer**
2. **Amount spent by each customer**

Solution : Join the tables (orders, order_item, customers) to calculate the no.of orders placed and the amount spent by each customer.