

Developer's Guide Ingestion Framework

Version V1.0

Version History

#	Date	Description	Updated By	Version
1	01/31/2005	First Version	Ketan Hande	V1.0

Index

Index	3
1 Introduction	5
2 Getting Started	5
2.1 Set Up Databricks Workspace	5
2.2 Access Permissions and User Roles	6
2.3 Access the Ingestion Framework for Templates	6
2.4 Configure Output Paths for Configuration Files	6
3 Standard Process Flow	7
3.1 Prep:	7
3.2 Metadata Configuration:	8
3.3 PAC & Pipelines:	8
4 Data Source Analysis	9
5 Data Ingestion Pipelines	10
5.1 Step 1: Clone or Access the Template	10
5.2 Step 2: Set Up the Delta Live Tables (DLT) Pipeline	10
5.3 Step 3: Configure Workflow	10
6 Source Table Configuration	12
6.1 Generate configuration files	12
7 Deployment	13
7.1 Change management & Branching strategy	13
7.2 Databricks Asset Bundle	14
7.2.1 Logical organization of Databricks Asset Bundles	14
7.2.2 ADB artifacts deployment using Databricks Asset Bundles	16
7.3 Steps to Create Asset Bundle	16
7.3.1 Initialize the Bundle:	16
7.3.2 Configure the Bundle:	16
7.3.3 Develop and Test Locally:	17
7.3.4 Validate the Bundle:	17

7.3.5	Deploy the Bundle:	17
7.3.6	Run the Deployed Assets:	18
7.3.7	Implement CI/CD Pipelines:.....	18
7.3.8	Monitor and Maintain:.....	18
7.4	Asset bundle deployment using DevOps	19
8	TBD: Monitoring & Logging	22
9	TBD: Troubleshooting	22
10	Naming Standards	22
10.1	Catalogs, Schemas, Tables.....	22
10.2	Notebooks and Workflows	22

1 Introduction

This guide provides a step-by-step approach to setting up and configuring a Databricks ETL ingestion framework that leverages Delta Live Tables (DLT) for efficient, scalable data pipelines. You'll learn how to set up a workflow that automates data ingestion, create configurations dynamically from catalog and schema information, and manage pipelines in Databricks. With this guide, developers can quickly deploy a robust data ingestion framework using reusable templates, optimized workflows, and well-organized configurations—empowering teams to seamlessly turn raw data into valuable insights.

2 Getting Started

This section will walk you through the initial setup and prerequisites needed before configuring the Databricks ingestion framework. By following these steps, you'll ensure that your Databricks workspace is ready to support a Delta Live Tables (DLT) pipeline and associated workflows for data ingestion.

2.1 Set Up Databricks Workspace

- **Access the Workspace:** If you don't already have access to the Sandbox or Dev Databricks workspace, request access.
 - The workspaces are behind a VNET so a VPN or AWS Workspace will be needed
 - Databricks permissions are managed via AD Groups, so an advantage account will need to be setup & configured
- **Cluster Configuration:** Ensure you have a cluster available for testing and running your pipeline. These should already be created but tickets or allocation may be limited. A request can be made to increase

2.2 Access Permissions and User Roles

- **User Roles:** Make sure you have the necessary permissions (Workspace Admin or Contributor) to create and run workflows, DLT pipelines, and manage assets within the workspace.
- **Access Control for Data Sources:** Verify that you have access to the data sources that will be ingested. Set up connection credentials and permissions as required for cloud storage (e.g., AWS S3, Azure Blob), databases, or other data repositories.

2.3 Access the Ingestion Framework for Templates

- **Access Template Repository:** Obtain the ETL ingestion template (from GitHub or a shared workspace repo) that contains the Delta Live Tables configuration and workflow setup scripts.

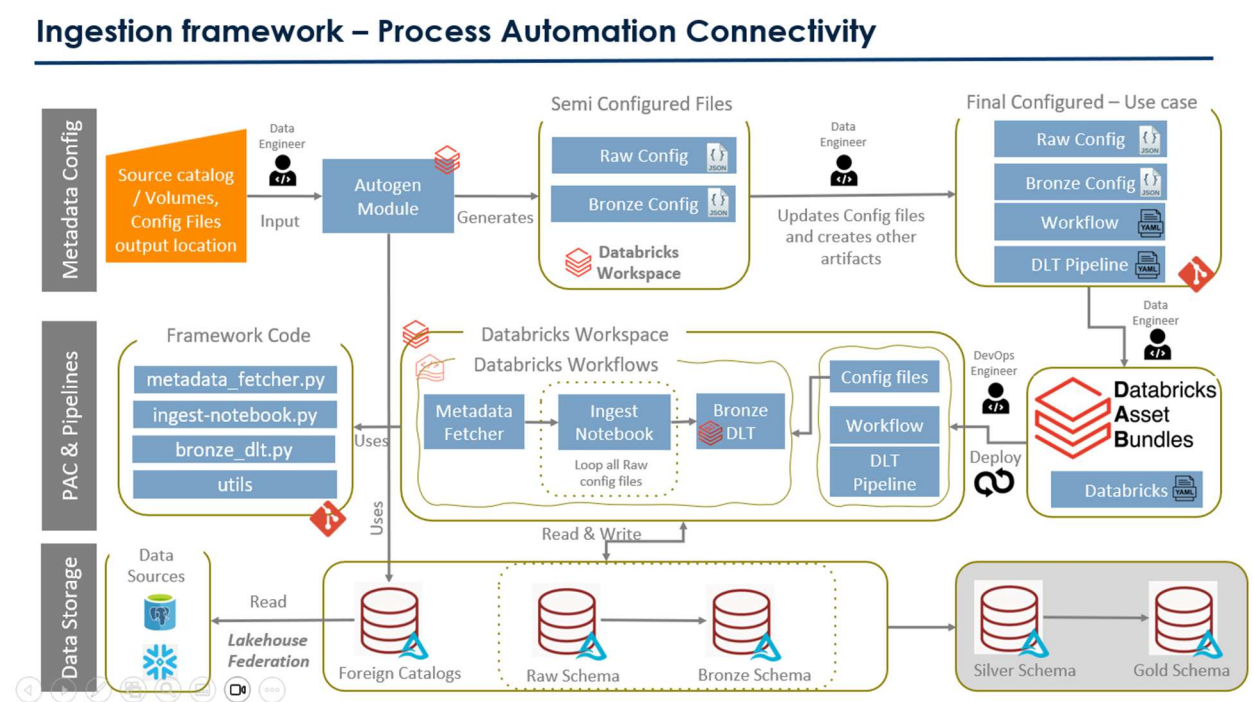
2.4 Configure Output Paths for Configuration Files

- **Define a Directory for Configuration Files:** You'll create configuration files for each dataset as you set up the ingestion framework. Choose a dedicated output path for these files in Volumes (e.g. /Volumes/...) as they'll be referenced in the configuration generation tool.

[Refer – how to use Auto Gen](#)

3 Standard Process Flow

Below diagram illustrates the process to use ingestion framework for your use case.



Note: Some steps can be done ahead of others (i.e. Step 6 can be started alongside step 3)

3.1 Prep:

Step 1: Create a new branch in GitHub for your work. It's important to keep the changes as small as possible to enable quick review. Each change set should contain changes according to a User Story from Azure DevOps.

Step 2: Identify the source catalogs needed

Step 3: Start on Data Source Analysis for the catalogs

3.2 Metadata Configuration:

Step 4: Use the generator tool to jumpstart the table configuration files creation

Step 5: As identified from the Data Source Analysis, manually configure each table's configuration (setting primary key, adjusting target locations if moving is needed, etc..)

Step 6: Place configuration files in the Mass Ingestion Area

3.3 PAC & Pipelines:

Step 7: Create or modify the Databricks Asset Bundle for the group of tables (as identified in Data Source Analysis)

Step 8: Commit all asset bundle changes in GitHub, make any needed adjustments & create a Pull Request.

Step 9: Inform the Data Engineering team of the new Pull Request & a review will be started

4 Data Source Analysis

This table provides an overview of essential configuration parameters for managing a data pipeline. Each parameter plays a specific role in defining how data is extracted, loaded, transformed, and stored across different stages of the pipeline (such as raw, bronze, and silver layers). By configuring these parameters, you can control aspects like data extraction strategies, handling of historical and incremental data, partitioning for optimized storage, and transformation requirements.

Parameter	Description	Supported Framework Values
Source Data extraction Strategy	Data read approach from source	Full, Incremental
Target Load Strategy Raw	Layer	cdc-append
Target Load Strategy Bronze	Bronze layer	SCD 2
Watermark Column	Column to identify the incremental data	MODIFIED_TIMESTAMP/ LOAD_TIMESTAMP
Watermark Value	Historical data period for fetching source data	Minimum date for historical data load
Primary Key (Merge Key to identify new records)	Records	Eg. product_code, id
Data Volume- Incremental	Incremental data volume expected	
Data Volume-Historical	Historical data volume expected	
CDC Patterns	Records	PK-NPK-HASHING/FULL-COLUMN-HASHING
Catalog Name	Catalog name for each target layer	
Schema Name	Schema name for each target layer	
Table Name	Table Name for each target layer	

5 Data Ingestion Pipelines

This guide will walk you through setting up an asset bundle on Databricks. We will set up a workflow that automates the execution of a Delta Live Tables (DLT) pipeline, leveraging an existing template to streamline setup.

5.1 Step 1: Clone or Access the Template

From where to clone?

1. **Locate the Template:**
 - a. It can be found within the workspace in a shared project repository.
2. **Clone the Template:**
 - a. Clone or download the templates to your workspace .
 - b. Ensure you have files:
 - i. databricks.yml
 - ii. workflows/workflow.yml
 - iii. pipelines/pipeline.yml
 - iv. shemas/schema.yml
3. **Review the Template:**
 - a. Review the DLT pipeline code to ensure it meets your data processing needs. Make adjustments if necessary.

5.2 Step 2: Set Up the Delta Live Tables (DLT) Pipeline

Note that some values may already be set as a template default.

1. Configure the pipeline :
 - a. Name the pipeline
 - b. Pipeline mode: Triggered
 - c. Cluster: TBD
 - d. Target: \${var.schema}
 - e. Notebooks: Leave as a reference to the main-repos notebook
2. Customize if needed
3. Save & validate

5.3 Step 3: Configure Workflow

1. Configure the workflow from template:
 - a. Name the workflow

- b. Add or update the DLT pipeline task to match the previous pipeline details
 - c. Configure the job settings
 - i. Schedule: TBD
 - ii. Cluster Configuration: TBD
 - iii. Retry: TBD
 - d. Add additional tasks if needed
 - 2. Save
 - 3. Validate

6 Source Table Configuration

6.1 Generate configuration files

Using Auto Gen module developers can generate semi configured Json schema. Please are the details of Auto Gen Notebooks and required parameters.

Notebook Path	/Workspace/Shared/dnap-ingestion-framework_18_12_2024/tools/json_config_generator
Parameters	Description
metadata_file_path	File path for saving landing to raw jsons or raw to bronze jsons
source_catalog	source catalog name: Landing catalog in case of raw jsons and raw catalog in case of bronze jsons
source_schema	source schema name
table_name	Name of the table for which we want to generate json
target_catalog	Target Catalog Name
target_schema	Target Schema Name

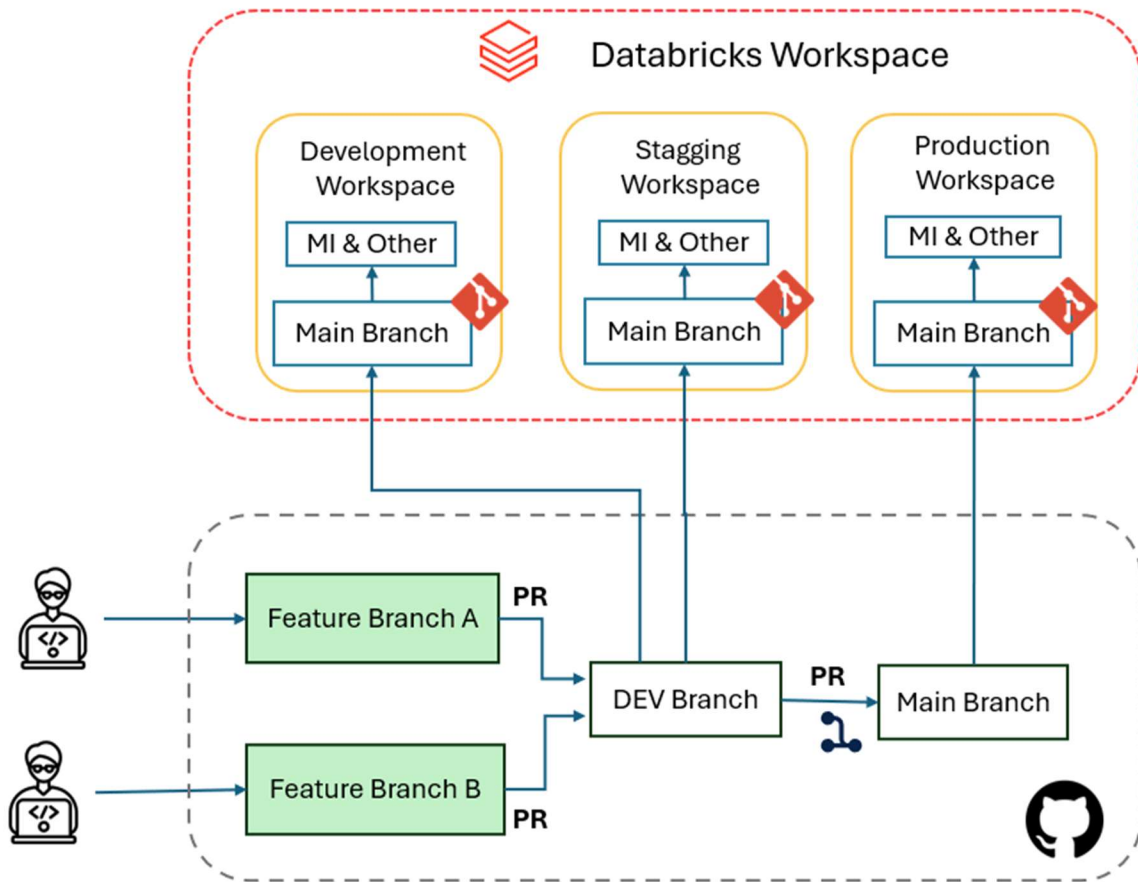
Once the config file is populated Developer needs to update certain fields such as we primary key, watermark column and modify the target table name etc. and everything else will be auto populated, source schema details, target schema details, datatype etc.

Place them in the Mass Ingestion location (TBD)

Test

7 Deployment

7.1 Change management & Branching strategy



Changes made to Mass Ingestion repository should be in an individual branch

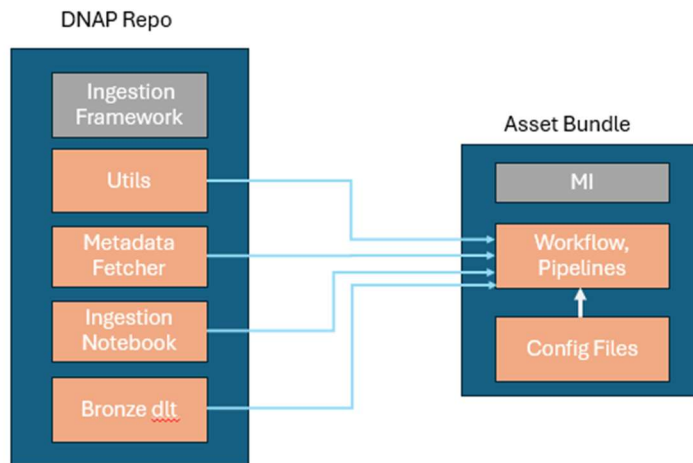
When ready, the changes can be manually deployed to Dev for testing and then a PR will be created for review

On merge from working branch to Main via the PR, the files and workflow configuration will be deployed to production

7.2 Databricks Asset Bundle

7.2.1 Logical organization of Databricks Asset Bundles

We have an ingestion framework designed for reuse across multiple use cases, such as a shared library. Create asset bundles as needed for the use case and reference the ingestion framework's utilities, notebooks, and other components within the asset bundle wherever required.



Databricks Asset Bundle directory structure

The following is an example of a directory structure for Databricks Asset Bundles. Each source domain is organized as a separate asset bundle, containing its own configuration files, workflows, pipelines, and the asset bundle configuration file (databricks.yml).

bundles/

```
--- boost_retail/
  --- definitions/
    --- <dev>
      --- raw/
      --- bronze/
    --- <prod>
      --- raw/
      --- bronze/
  --- workflows/
    --- boost_retail_workflow.yml
  --- pipelines/
    --- boost_retail_bronze_pipeline.yml
  --- schemas/
    --- boost_retail_schema.yml
  --- databricks.yml
```

Config Files

Databricks workflow

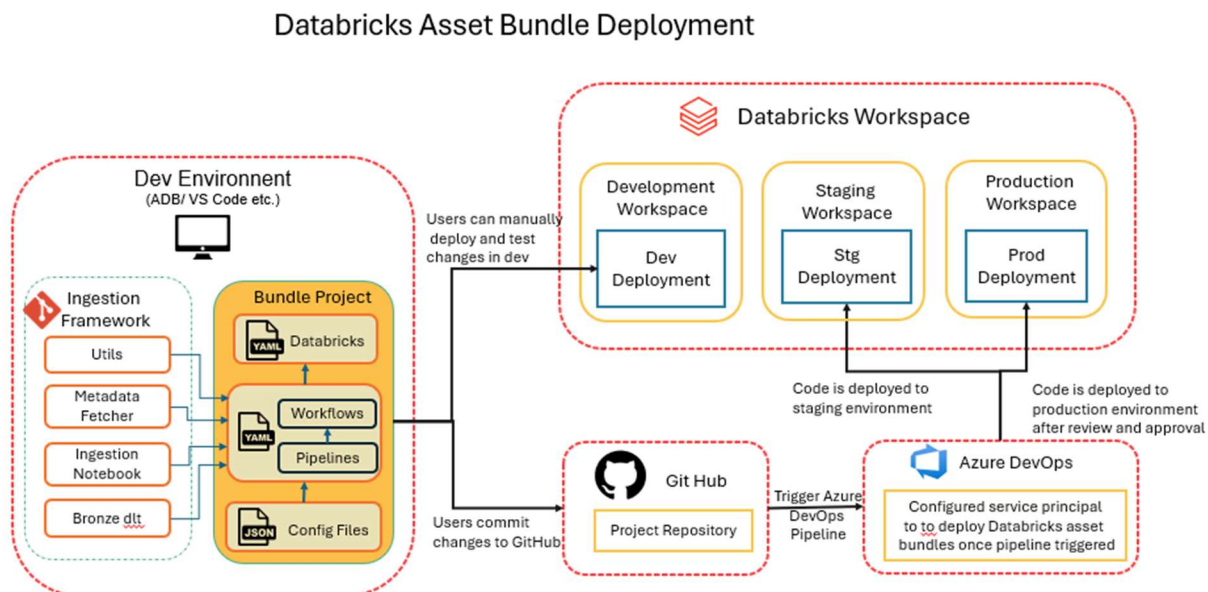
dlt pipeline

Databrick Bundle config

```
--- jet_pts/
  --- definitions/
    --- <dev>
      --- raw/
      --- bronze/
    --- <prod>
      --- raw/
      --- bronze/
  --- workflows/
    --- jet_pts_workflow.yml
  --- pipelines/
    --- jet_pts_bronze_pipeline.yml
  --- schemas/
    --- jet_pts_schema.yml
  --- databricks.yml
```

README.md

7.2.2 ADB artifacts deployment using Databricks Asset Bundles.



7.3 Steps to Create Asset Bundle

Consider the following high-level steps:

7.3.1 Initialize the Bundle:

Begin by creating a new bundle using the Databricks CLI. This involves selecting an appropriate template that suits your project's requirements. For example, to use the default Python template, execute

```
%bash
```

```
databricks bundle init default-python
```

7.3.2 Configure the Bundle:

Define your bundle's settings and resources in the `databricks.yml` file. This configuration should specify details such as workspace parameters, artifact names, file locations, job definitions, and pipeline configurations. Ensure that you include deployment targets (e.g., development, staging, production) with appropriate settings for each environment.

7.3.3 Develop and Test Locally:

Develop your code and test it in your local environment to ensure functionality before deployment. This step helps in identifying and resolving issues early in the development process.

7.3.4 Validate the Bundle:

Before deploying, validate your bundle configuration to ensure all definitions are correct. Use the following command:

```
%bash
```

```
databricks bundle validate
```

This command checks for any configuration errors that need to be addressed prior to deployment

7.3.5 Deploy the Bundle:

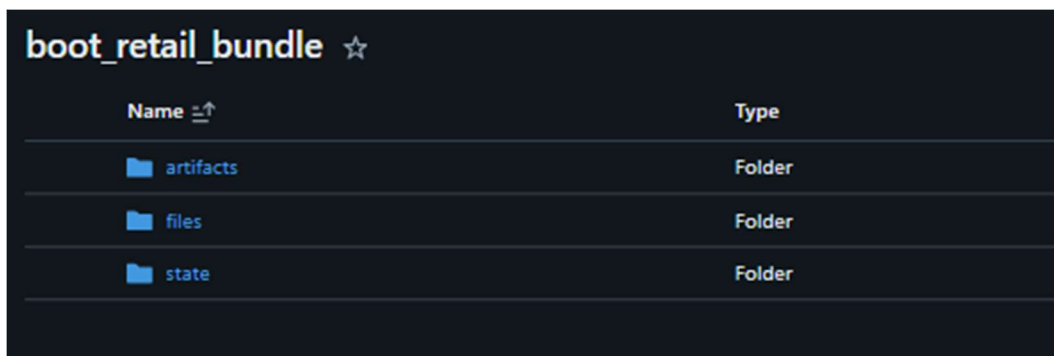
Deploy your bundle to the specified target environment using the Databricks CLI:




```
%bash
```

```
databricks bundle deploy --target <target-name>
```

Replace <target-name> with the appropriate target defined in your configuration (e.g., dev, staging, prod)

Upon successful deployment, you will find all the below folders in the destination of the deployment.



Name 	Type
 artifacts	Folder
 files	Folder
 state	Folder

All the deployed assets will be available under files folder



The screenshot shows a file explorer window titled 'files' with a star icon. It contains a table with two columns: 'Name' and 'Type'. The table lists the following items:

Name	Type
definitions	Folder
pipelines	Folder
schemas	Folder
workflows	Folder
.gitignore	File
databricks.yml	File

7.3.6 Run the Deployed Assets:

Execute the deployed jobs or pipelines as needed:

```
%bash
```

```
databricks bundle run <job-name> --target <target-name>
```

Ensure that the <job-name> corresponds to the job defined in your bundle configuration.

7.3.7 Implement CI/CD Pipelines:

Integrate your deployment process into a Continuous Integration/Continuous Deployment (CI/CD) pipeline using Azure DevOps. This setup automates validation, deployment, and execution of your bundles upon code changes, ensuring consistent and reliable deployments.

7.3.8 Monitor and Maintain:


After deployment, monitor the performance and health of your jobs and pipelines. Regular maintenance and updates to your bundles should be managed through your version control and CI/CD systems to maintain alignment with best practices.







By following these steps, you can establish a robust deployment process using Databricks Asset Bundles that adheres to Databricks standards and facilitates efficient development and deployment workflows.

7.4 Asset bundle deployment using DevOps

Create a Azure DevOps pipeline `azure-pipelines.yml` and integrate GitHub repository with the Azure DevOps pipeline where the code resides. The Azure DevOps pipelines are running on a self hosted agent and each environment has a separate agent pool configured.

Define pipeline level variables and their values for each environment in the variable group under `pipelines.library`. Also a bash script is added for installing all the dependencies required for running the ADO pipeline under the `.ado` folder.

Library >  ADV-ADB-MI-DEPLOY-PROD*

Variable group |  Save  Clone  Security  Pipeline permissions  Approvals and checks  Help

Properties

Variable group name

ADV-ADB-MI-DEPLOY-PROD

Description

This variable group is for Mass Ingestion team's Databricks deployment to production

☒ Link secrets from an Azure key vault as variables ⓘ

Variables

Name ↑	Value
DATABRICKS_HOST	https://adb-7622183527867515.15.azure.databricks.net
env	prod

Usage of service principal

For databricks asset bundles deployment through Azure DevOps pipeline we are creating a service principal and granting contributor permission at subscription level where the Databricks workspace is provisioned. The jobs/DLT pipelines/schemas will be provisioned under the service principal name. The service principal needs to be added manually to the respective databricks workspace. We need to provide admin access to the service principal at the workspace level.

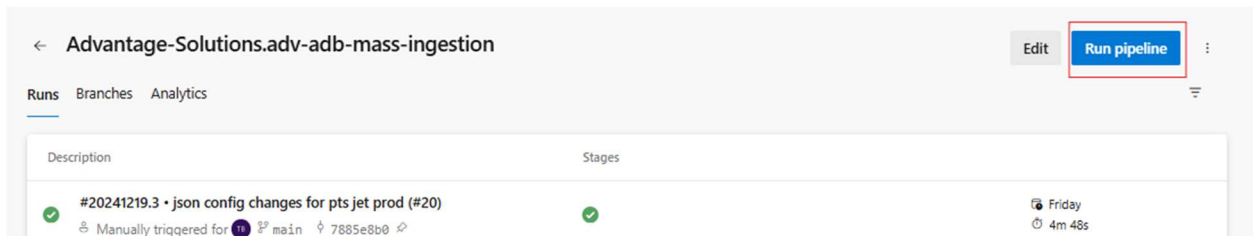
The created service principal is also configured in the service connections under the Azure DevOps project settings.

Permissions

The service principal should have USE_CATALOG, CREATE SCHEMA etc permissions in order to successfully validate and deploy the asset bundles, else it will fail at validate DAB or deploy DAB step.

Additionally the service principal is also provided admin permission in the databricks workspace level.

Go to Azure DevOps, select the pipelines and proceed with the necessary pipeline.



Enter the required parameters that are required to run the Azure DevOps pipeline and then click on Run.

Run pipeline

Select parameters below and manually run the pipeline

Branch/tag

main

Select a branch from the list or enter the name of a tag as refs/tags/<tagname>

Commit

version

3.8

environment

☒ DEV

☐ PROD

subPath

☒ bundles/boost_retail

☐ bundles/pts_jet

Advanced options

Variables

This pipeline has no defined variables

☐ Enable system diagnostics

CancelRun

Once the pipeline is triggered then these are the below steps which would be executed in order to complete the deployment successfully.

Deployment		
✓	DAB_Deployment	4m 25s
✓	Initialize job	5s
✓	Checkout repository	3s
✓	Environment / Conte...	2s
✓	Use Python 3.8	17s
✓	Install addition...	2m 10s
✓	Fetch Databricks To...	10s
✓	Install Databricks CLI	4s
✓	Configure Databrick...	1s
✓	Validate DAB	4s
✓	Deploy DAB	1m 23s
✓	Post-job: Checkout...	<1s
✓	Finalize Job	<1s

8 TBD: Monitoring & Logging

9 TBD: Troubleshooting

10 Naming Standards

10.1 Catalogs, Schemas, Tables

10.2 Notebooks and Workflows