IMPLEMENTANY ONE OF CICD PIPLINE ACTIVIRY IN AZURE DEVOPS & AZURE DATA FACTORY

Theophila Murphy S

DE-Batch 6

**CI/CD Pipeline in Azure DevOps**

**Introduction**

Continuous Integration and Continuous Deployment (CI/CD) in Azure DevOps allows development teams to automatically build, test, and deploy applications. Using Azure DevOps, developers can ensure that any change to the application code is reliably tested and deployed to multiple environments (Development, Test, Production) with minimal manual effort.

**Prerequisites**

Before starting the pipeline implementation, ensure the following:

1. An **Azure DevOps organization** and project is created.
2. The **application source code** is stored in Azure Repos or GitHub.
3. Target deployment environment is prepared, such as **Azure Web App, Virtual Machines, or AKS**.
4. An **Azure Service Connection** is created in Azure DevOps with **Contributor access** to the target resources.

**Step-by-Step Implementation**

**Step 1: Organize the Repository**

* Create a folder structure:

/src → Application source code

/tests → Unit or integration tests

/pipelines → YAML files for CI/CD pipeline

**Step 2: Build Pipeline (Continuous Integration)**

* Navigate to **Pipelines → New Pipeline → YAML** in Azure DevOps.
* Select your repository and configure the pipeline to trigger on changes to the main branch.

**Step 3: Release Pipeline (Continuous Deployment)**

* Create a release pipeline with multiple stages: **Development → Test → Production**.
* Each stage should:
  + Take artifacts from the build pipeline.
  + Deploy to the target environment (Azure Web App, VM, or AKS).
  + Include environment-specific configurations.
* Configure **manual approvals** for production deployment to ensure safety.

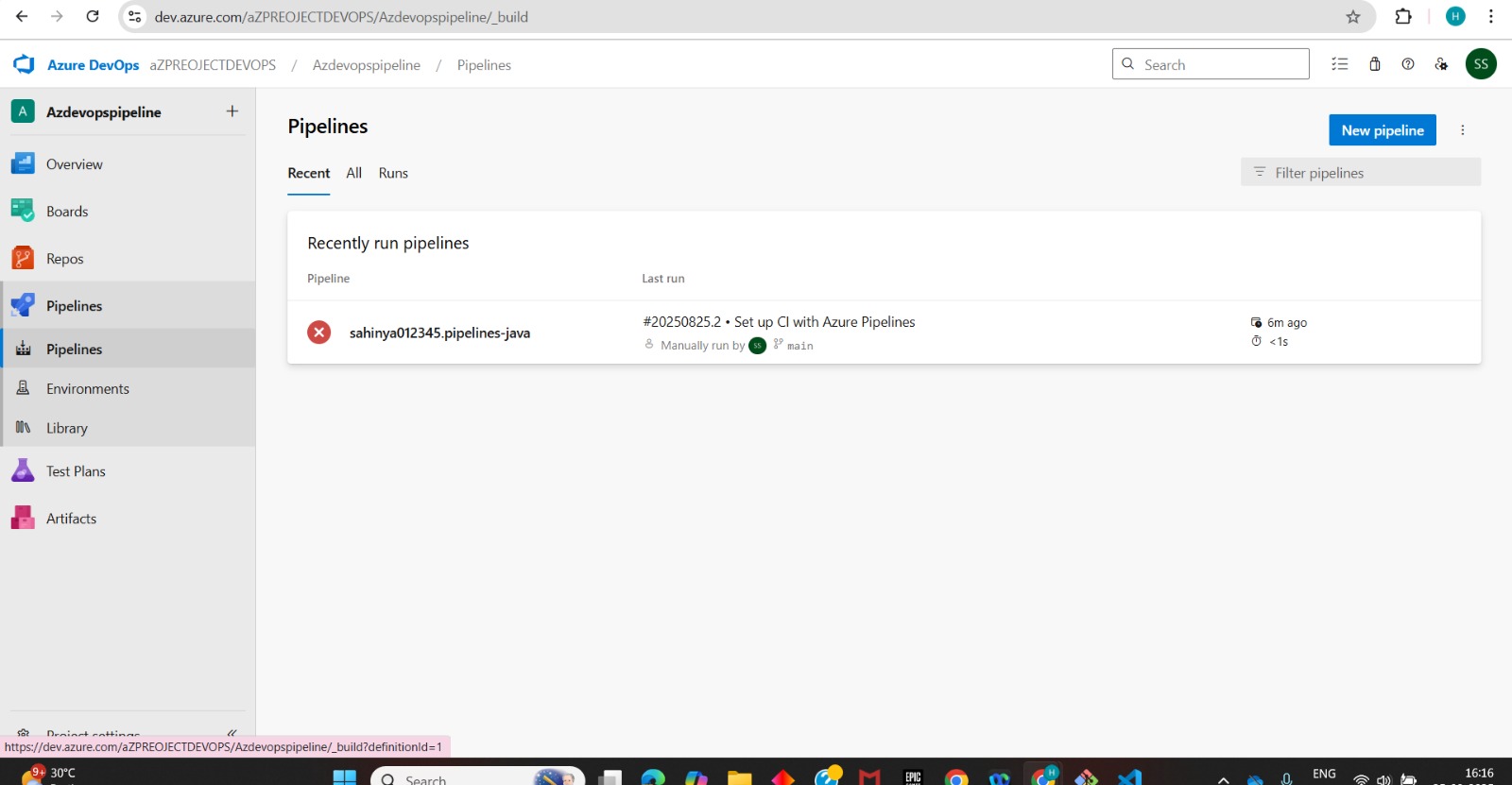
**Step 4: Configure Variables and Secrets**

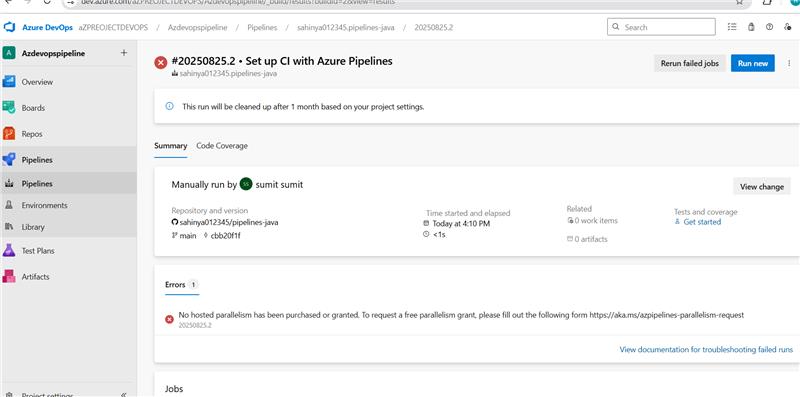
* Define variables for environment-specific settings like connection strings or service endpoints.
* Secure sensitive data in **Azure Key Vault** and link it to the pipeline.

**Step 5: Run Pipeline and Verification**

* Commit code changes to the repository; this will trigger the build pipeline automatically.
* Verify the build artifacts are created successfully.
* Deploy to Development → Test → Production environments sequentially.
* Perform smoke tests to ensure the application works as expected in each environment.

**OUTPUTS:**

****

****

**CI/CD Pipeline in Azure Data Factory**

**Introduction**

ADF CI/CD pipelines to automate deployment of data integration pipelines, datasets, and linked services from development to production. By combining ADF Git integration with Azure DevOps pipelines, teams can maintain version control, deploy reliably, and manage secrets securely, ensuring data workflows operate consistently across environments.

**Steps to Create CI/CD Pipeline in Azure Data Factory**

**Step 1: Connect ADF to Git**

1. Open **Azure Data Factory Studio → Manage → Git Configuration**.
2. Select **Azure DevOps Git**.
3. Provide Organization, Project, Repository, and Collaboration Branch (main).
4. Set the root folder for ADF artifacts (e.g., /adf\_root).
5. Click **Apply**.

**Step 2: Develop in Feature Branch**

1. Create pipelines, datasets, and linked services in the **Development ADF instance**.
2. Work in **feature branches**.
3. Create a **Pull Request (PR)** to merge changes into the main branch.

**Step 3: Publish ARM Templates**

After merging PR into **main**, click **Publish** in ADF Studio.

**Step 4: Create Build Pipeline in Azure DevOps**

1. Go to **Azure DevOps → Pipelines → New Pipeline → YAML**.
2. Point it to the repository and main branch.
3. Trigger the pipeline on changes in /adf\_publish/\*\*.
4. Add steps to:
   * Copy ARM templates to artifact staging.
   * Publish build artifacts.

**Step 5: Create Release Pipeline**

1. Add stages: **Test → Prod**.
2. Each stage deploys ARM templates to the corresponding ADF environment using the **AzureResourceManagerTemplate** task.
3. Add **manual approval** for the Prod stage.

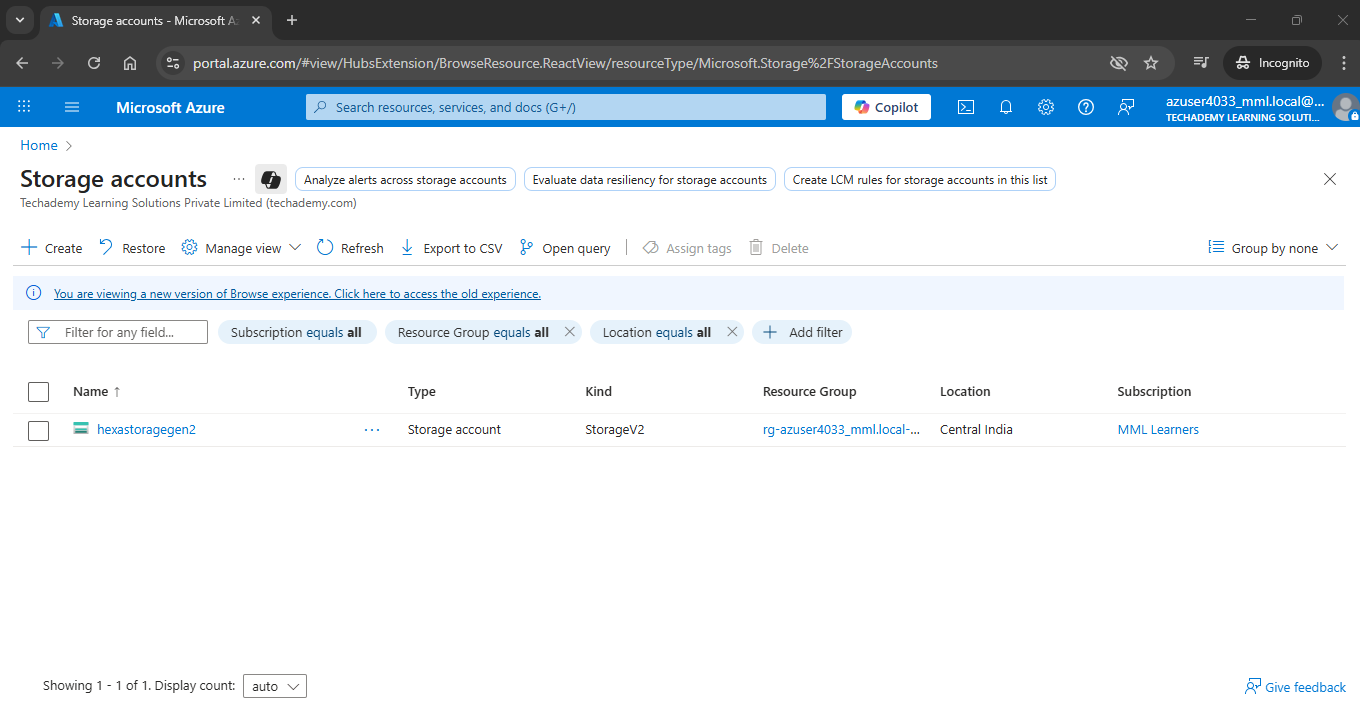
**Step 6: Configure Variables**

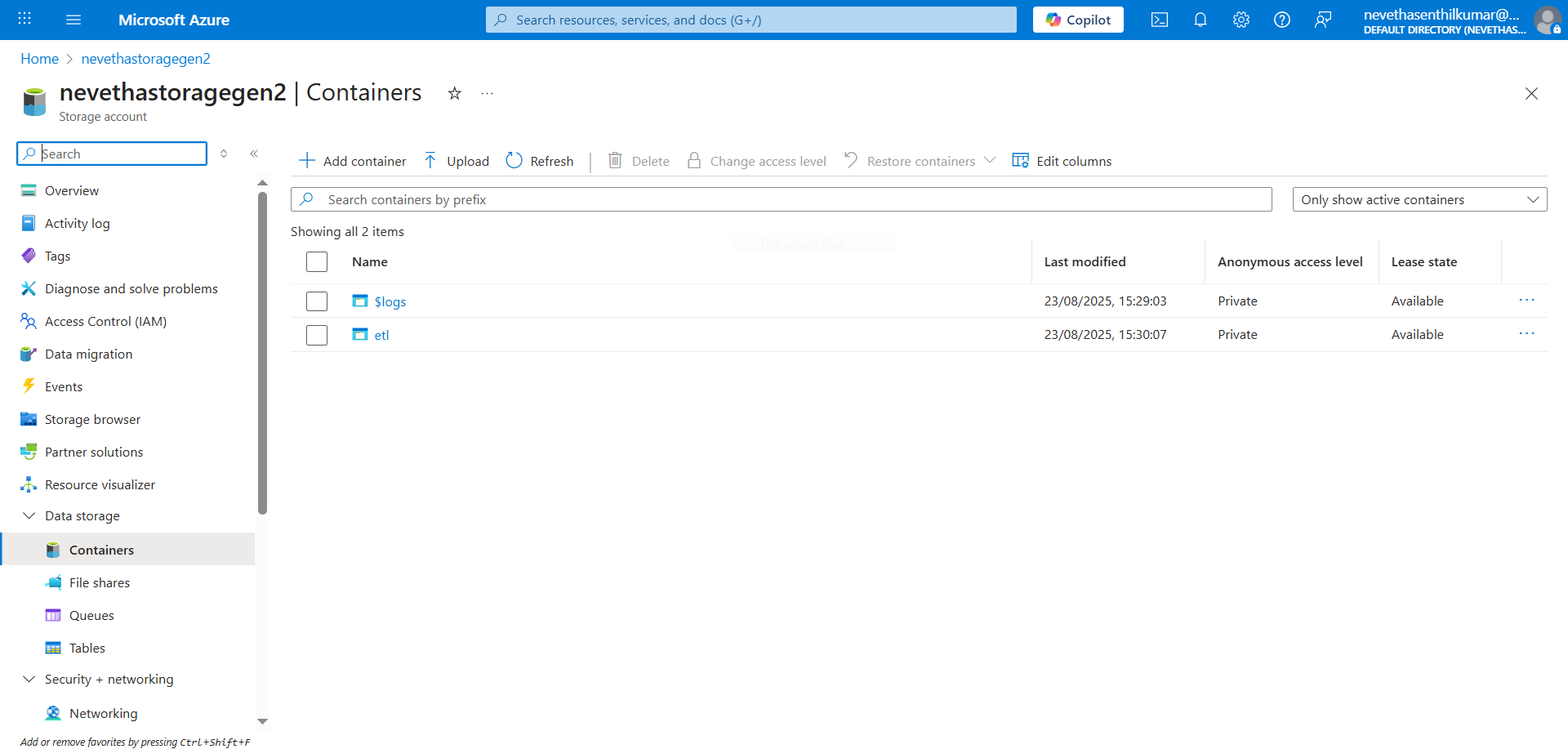
1. Set environment-specific variables: Factory name, Resource group, Storage account, etc.
2. Use **Azure Key Vault** references for secrets in linked services.

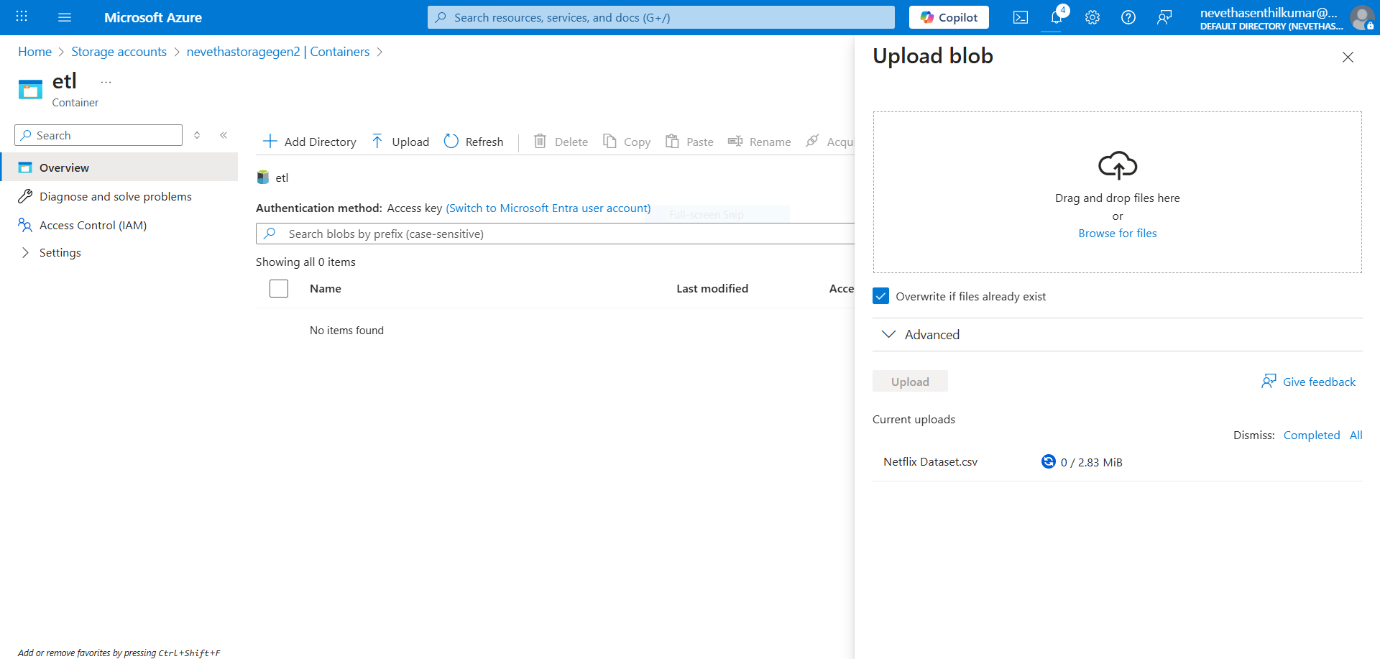
**Step 7: Trigger Pipeline and Verify**

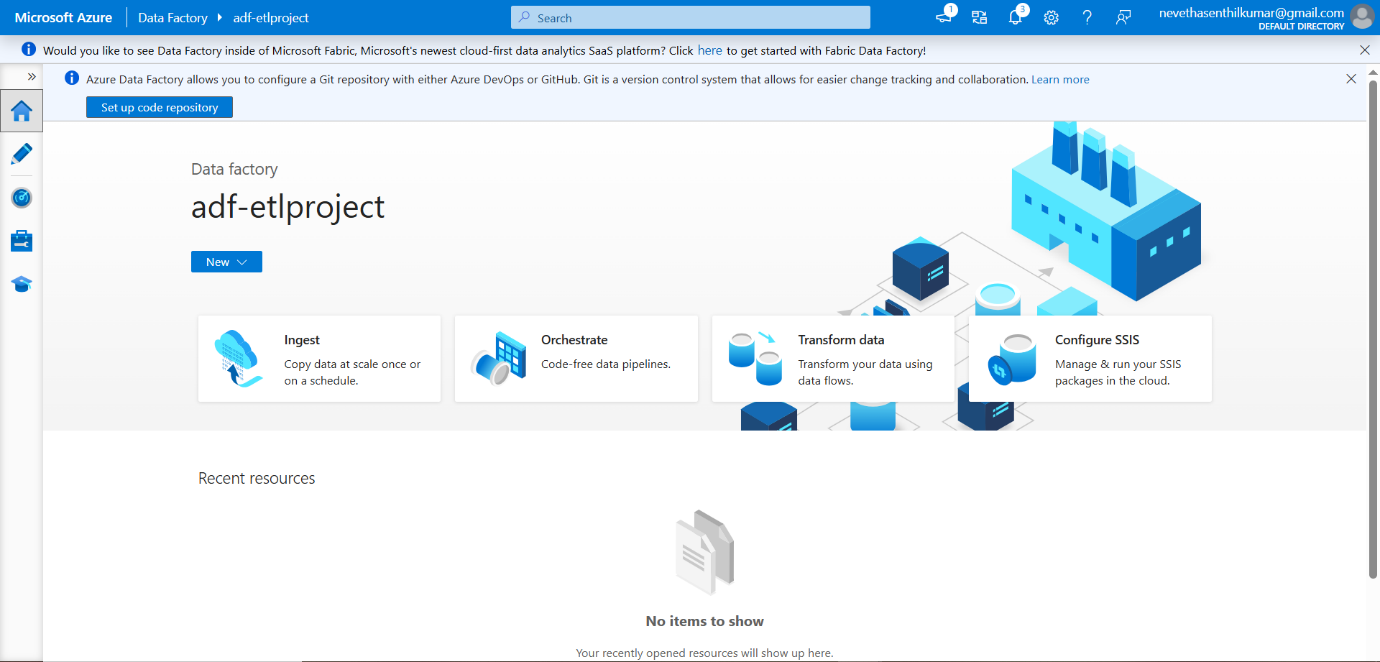
1. Merge code → triggers **Build pipeline**.
2. Ensure build artifacts are created successfully.
3. Deploy to **TEST** environment → validate pipelines, datasets, and triggers.
4. After approval → deploy to **PROD** environment.
5. Run a **sample pipeline** to confirm deployment success.

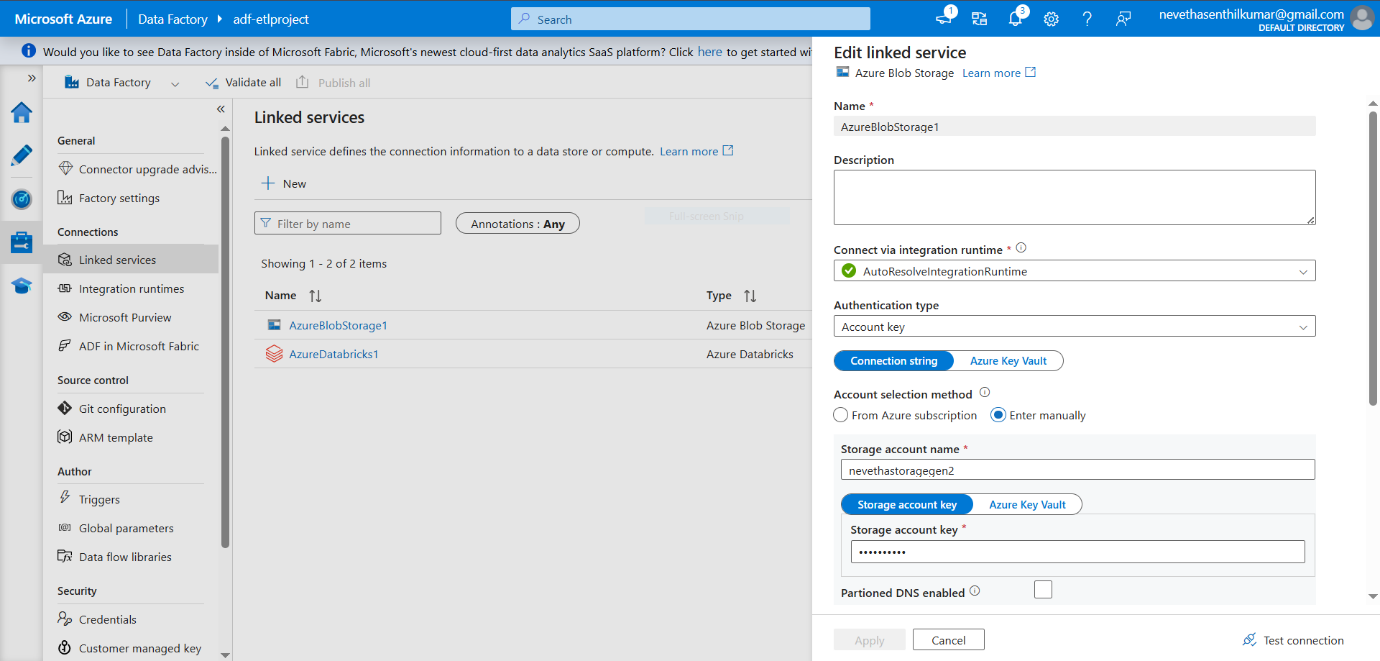
**OUTPUTS:**

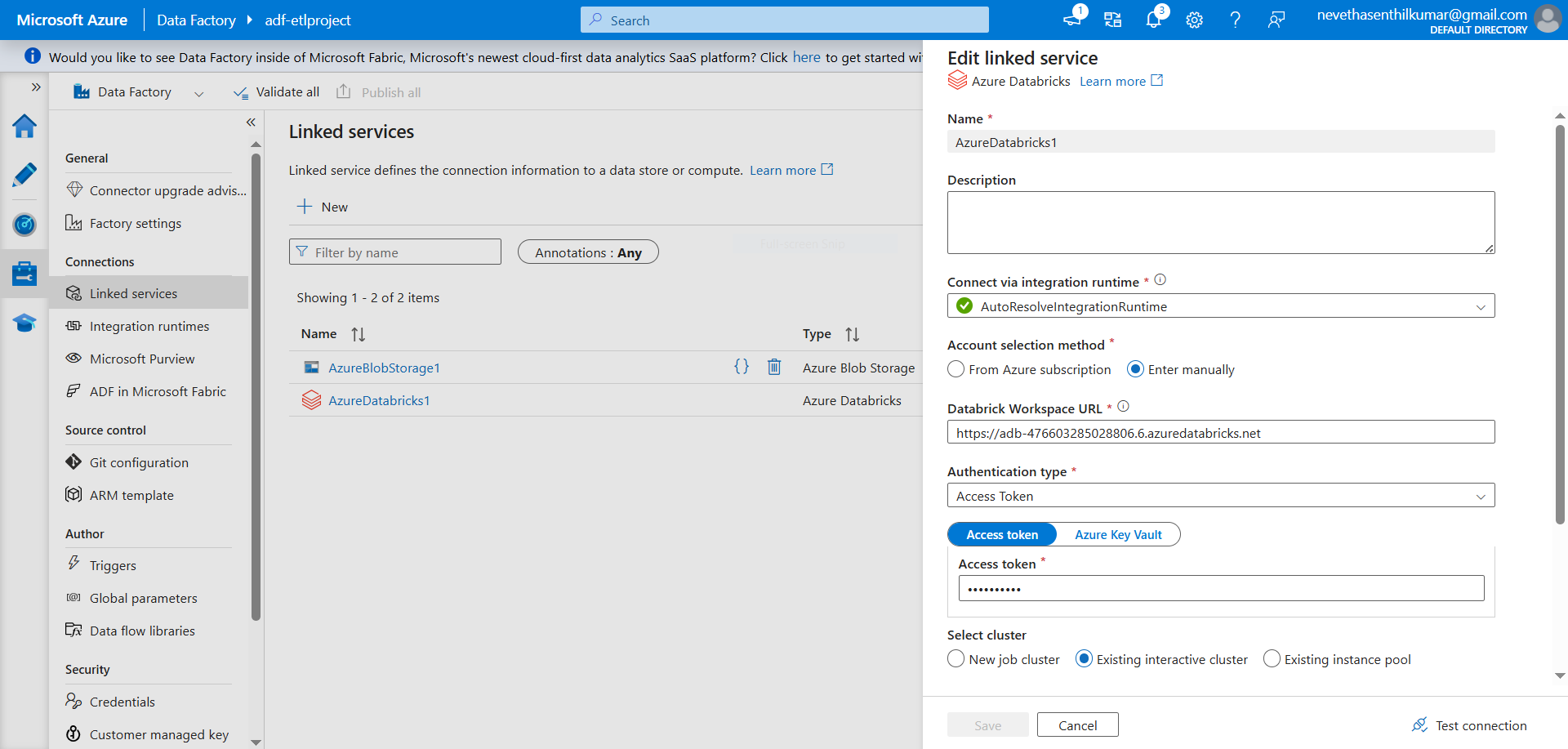


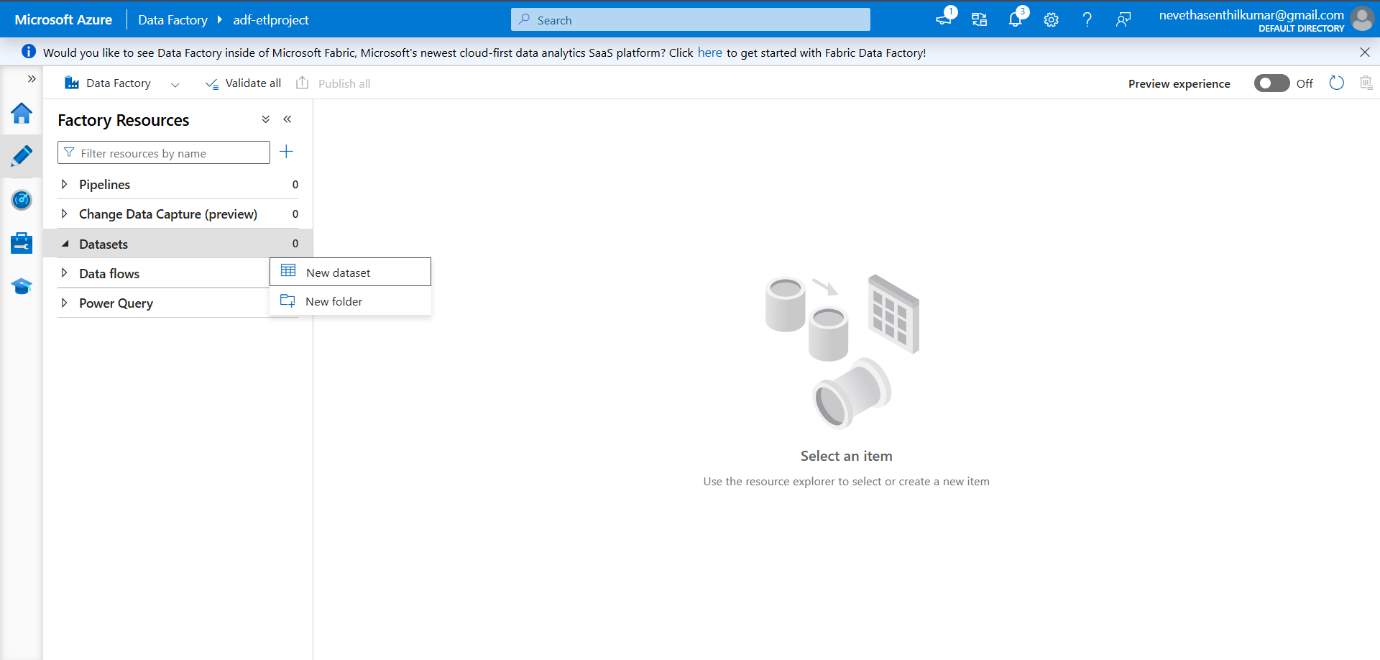


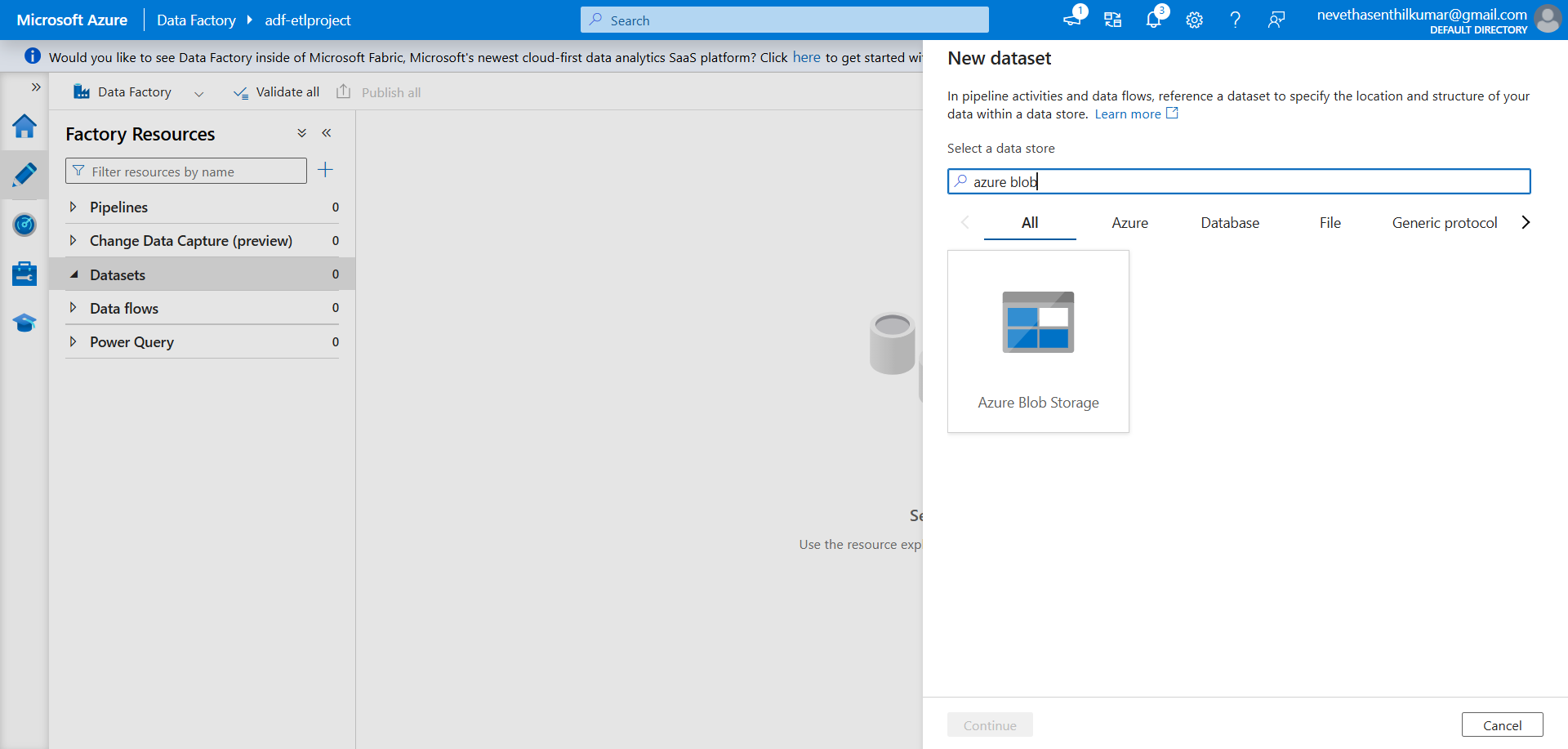


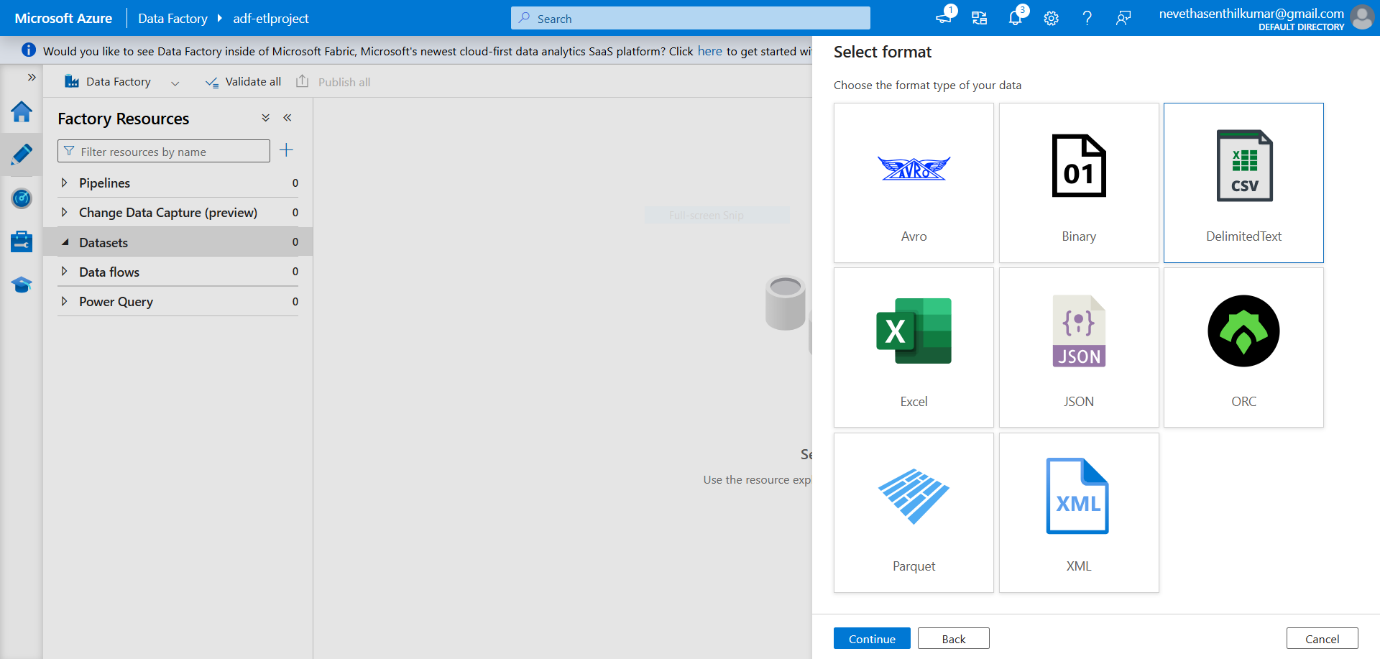


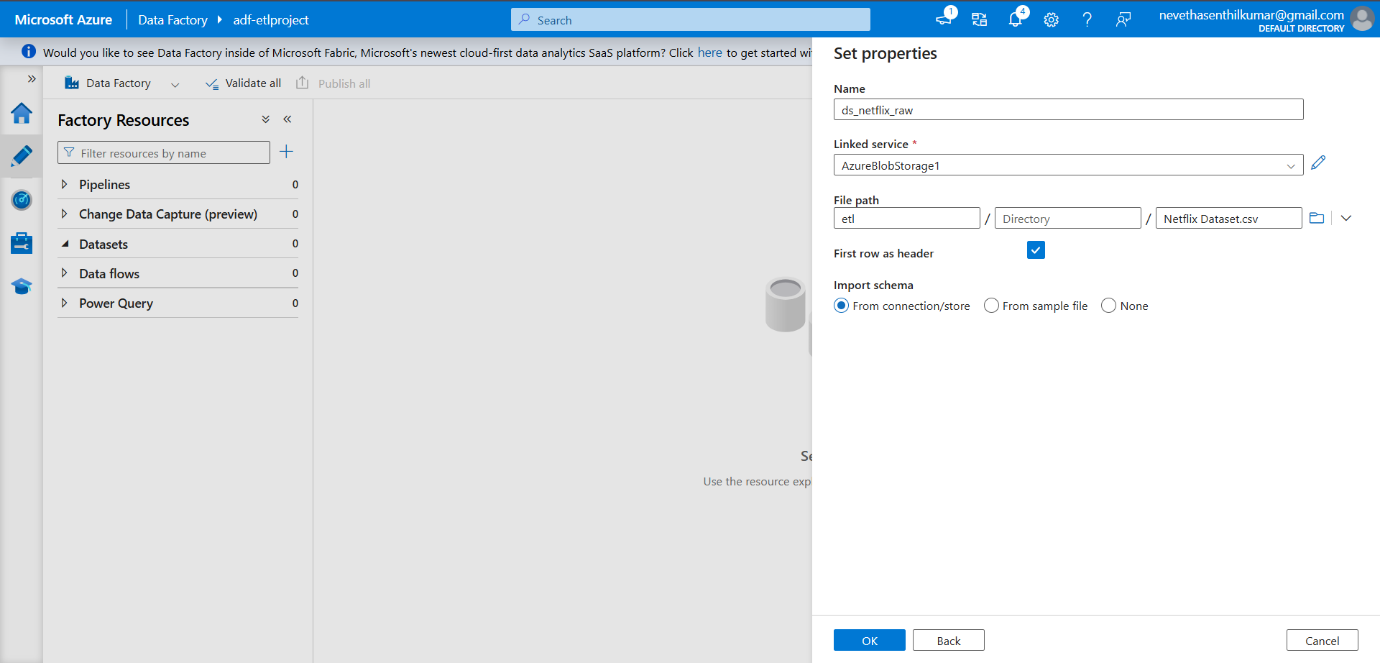


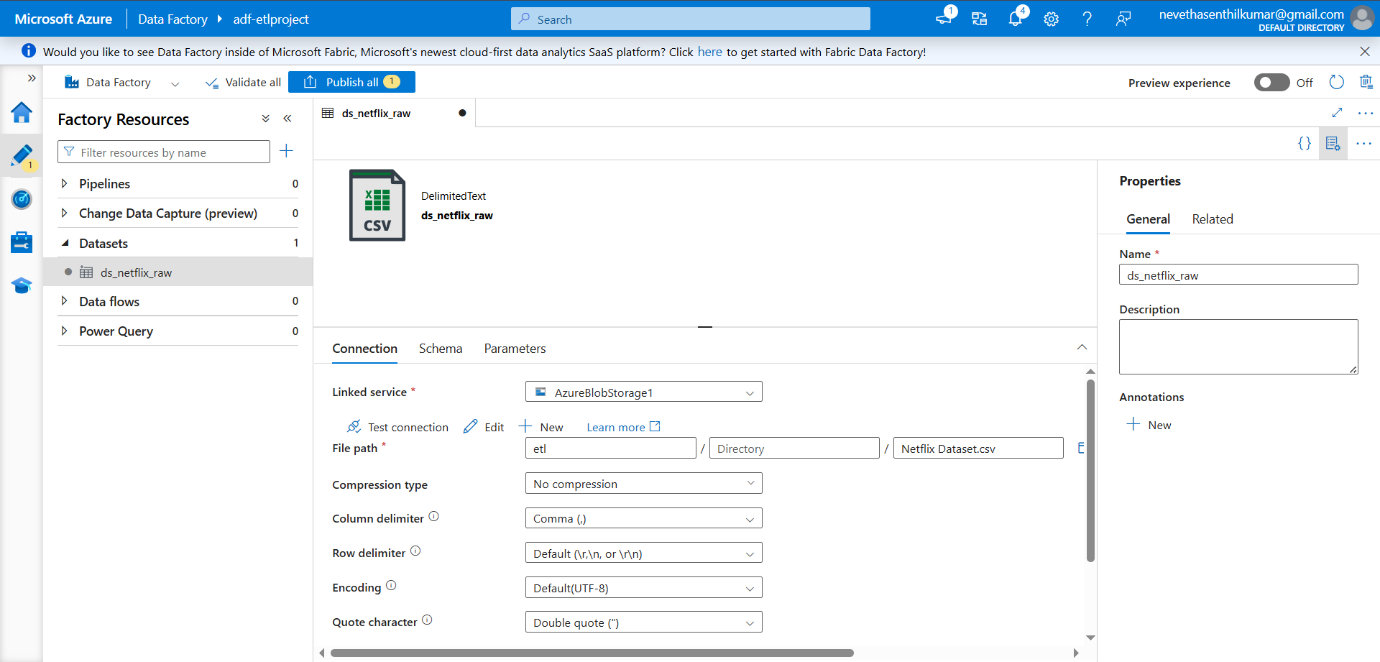


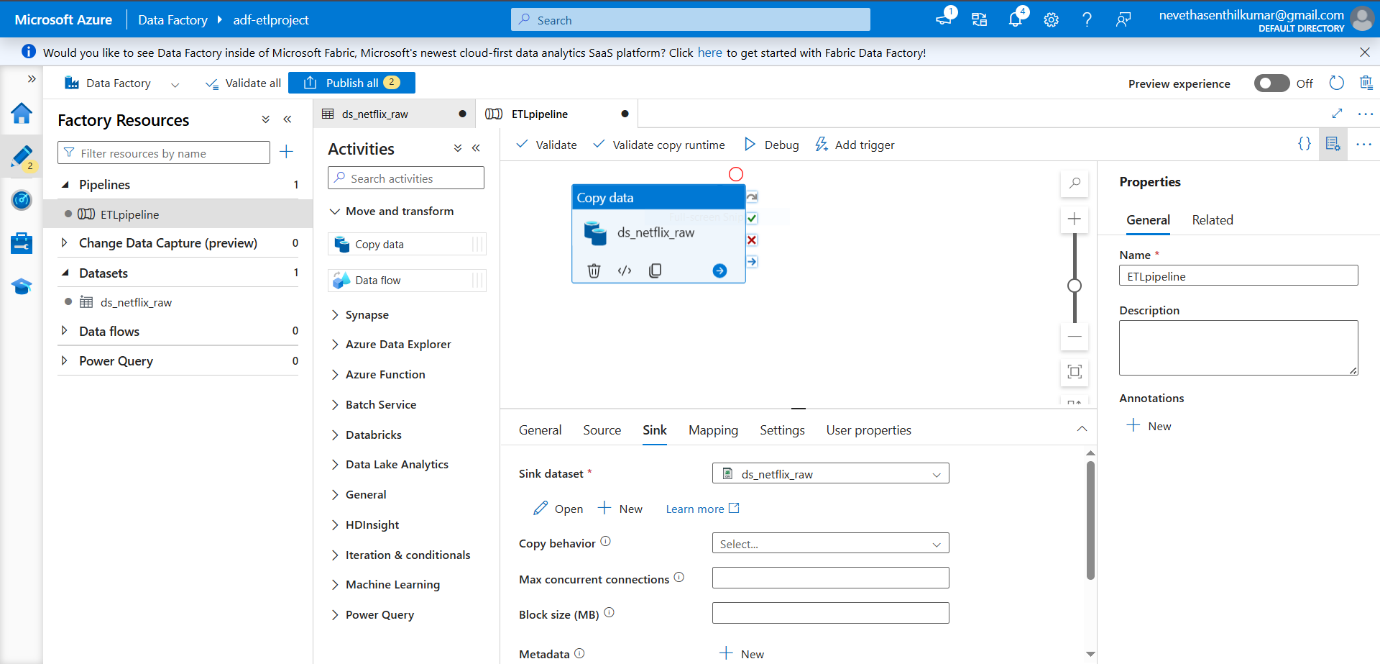




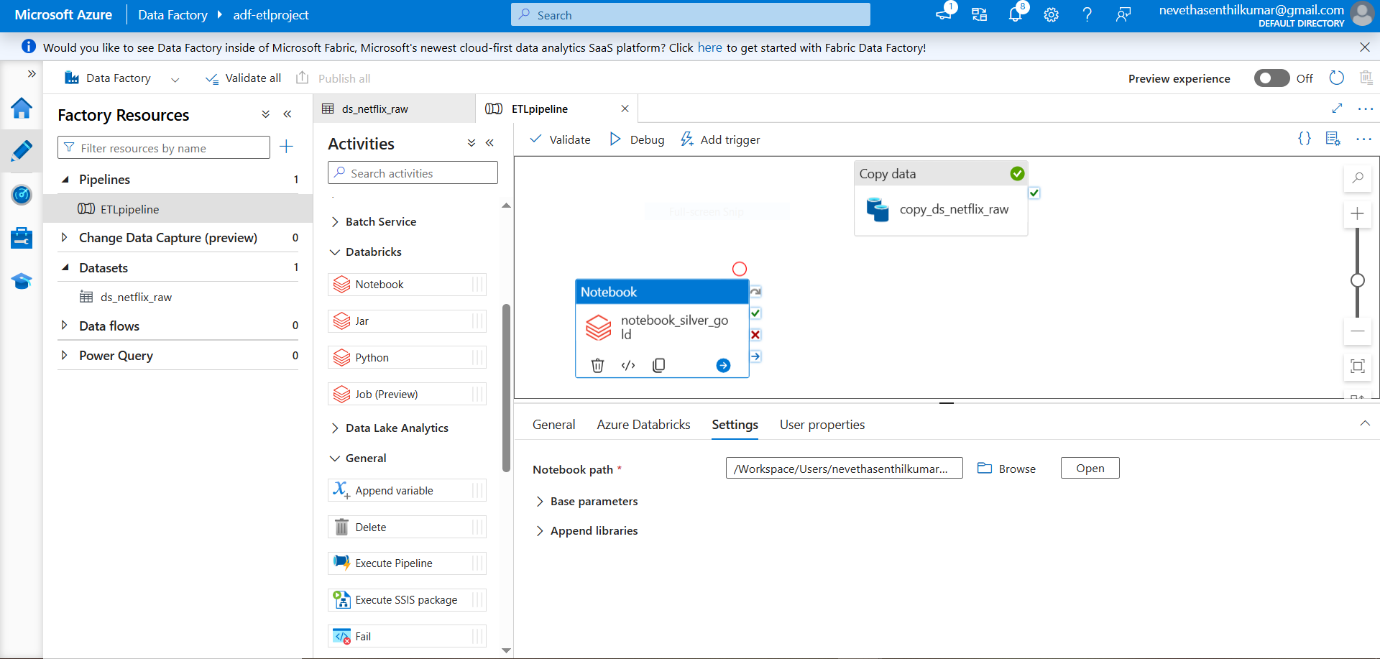








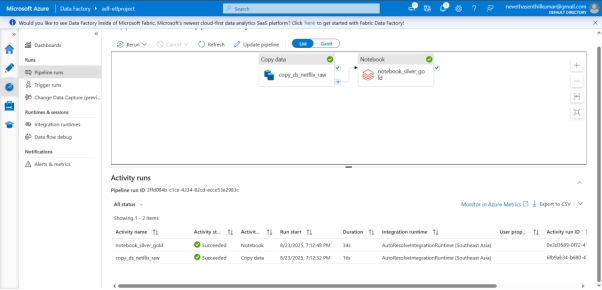
**Copy Data Activity –** Moves raw data from Blob Storage into the staging/target dataset.



**Pipeline Trigger –** The pipeline is triggered manually to start the orchestration process.

**Successful Output Generated:**

* Validate and Debug the pipeline created to see the results of the execution.
* It shows the Activity Status as Succeeded which means our CSV file is converted into the Parquet file in the destination folder after the successful ETL Process.

****