|  |  |
| --- | --- |
|  | Azure Data Factory Deployment & Automation |
| Deployment & Automation from DEV to UAT Instance | |

| **Document Details** | |
| --- | --- |
| Document Author | Rupesh Shelar |
| Contributors | Agglomerates Scrum |

Table Of Contents

[1 Overview 3](#_Toc192026036)

[2 Project Scope 3](#_Toc192026037)

[2.1.1 Data Flow 1.1 3](#_Toc192026038)

[2.1.2 Automated Pipeline 1.2 3](#_Toc192026039)

[2.2 Post Migration Steps and monitoring 1.2 3](#_Toc192026040)

[2.2.1 Security 1.1.1 4](#_Toc192026041)

[3 Deployment Plan 2 4](#_Toc192026042)

[3.1 Testing 2.1 4](#_Toc192026043)

[4 Azure Data Factory (ADF) Feature Branch Workflow & Deployment Process 5](#_Toc192026044)

[4.1 1. Introduction 5](#_Toc192026045)

[4.2 2. Creating a Feature Branch 5](#_Toc192026046)

[4.3 3. Making Changes and Publishing 5](#_Toc192026047)

[4.4 4. Taking ARM Templates Backup 5](#_Toc192026048)

[4.5 5. Creating a Pull Request (PR) and Merging Changes 5](#_Toc192026049)

[4.6 6. Configuring Pipeline with ARM Template 6](#_Toc192026050)

[4.7 7. Adding Parameters and Resource Configurations 6](#_Toc192026051)

[4.8 8. Running the Pipeline and Merging Changes 6](#_Toc192026052)

[4.9 9. Final Deployment Validation 6](#_Toc192026053)

[4.9.1 Conclusion 6](#_Toc192026054)

# Overview

This document outlines the automation process for the Azure Data Factory pipeline named

**ppl\_ExtractDatafromDataPlatform** for migrating data between **DEV** and **UAT** instances.

The goal is to create a pipeline that automates the extraction, transformation, and loading (ETL) of data from the **StagingDEV** and **MultivueReporting** databases into their corresponding UAT instances.

.

# Project Scope

* **Source Systems**: StagingDEV, MultivueReporting (in the DEV environment)
* **Target System**: UAT Instance (same schemas, database instance)
* **Data Movement Type**: Full or Incremental Load
* **Pipeline Name**: **ppl\_ExtractDatafromDataPlatform**
* **Automation Requirements**: Full automation of the pipeline execution using triggers and parameterization.

### Data Flow 1.1

**Source Database (BDC\_Data\_Platform)** → **Data Extraction** → **Azure Data Factory Pipeline** → **Data Transformation(Inbuilt)** → **UAT Database Instance (StagingUAT, MultivueReporting- SQL Server**

### Automated Pipeline 1.2

* **Triggering Mechanism**: Azure Data Factory triggers will be used to automate the pipeline execution.(TBD)

## Post Migration Steps and monitoring 1.2

1. **Load Data**:
   * Run ADF Pipeline to load data into UAT database tables.
2. **Error Handling**:
   * Use **Azure Monitor** for logging errors and monitoring pipeline execution.
3. **Logging and Monitoring**:
   * Logging mechanisms with Azure Monitor and custom logging for debugging purposes.
   * Email notifications for failed pipeline runs.
4. **Success/Failure Notifications**:
   * Use **Azure Logic Apps** or **Azure Function** to send email notifications on the success or failure of the pipeline run.

### Security 1.1.1

* **Managed Identity**: Use Managed Identity for secure connections to Azure SQL Databases.
* **Azure Key Vault**: Store connection strings, credentials, and sensitive data in Azure Key Vault.

# Deployment Plan 2

* **Stage 1**: Implement and test the pipeline in the **Development** environment.
* **Stage 2**: Deploy and test the pipeline in **UAT** environment.( Deployement Steps in PBI)
* **Stage 3**: Implement the pipeline for **Production** deployment (if required

## Testing 2.1

* **Unit Testing**:
  + Test each activity (e.g., Copy, Data Flow) individually in a non-production environment.
* **Integration Testing**:
  + Test end-to-end data flow to ensure that the pipeline works as expected.
* **UAT Testing**:
  + Allow stakeholders to validate data in the UAT environment before final deployment.

# Azure Data Factory (ADF) Feature Branch Workflow & Deployment Process

## ****1. Introduction****

This document outlines the standard procedure for managing changes in **Azure Data Factory (ADF)** using a feature branch workflow. It includes steps for **creating a feature branch, making changes, publishing updates, handling ARM templates, and merging the changes into the main repository.**

## ****2. Creating a Feature Branch****

* Clone the **main repository** locally.
* Create a new **feature branch**:

git checkout -b feature/<feature-name>

* Ensure the branch is **up-to-date** with the main branch before proceeding.

## ****3. Making Changes and Publishing****

* Make required **modifications** to ADF resources.
* **Validate and test** the changes within the feature branch.
* Publish the changes to **adf\_publish** branch:
* git add .
* git commit -m "Added new ADF feature <feature-name>"

git push origin feature/<feature-name>

## ****4. Taking ARM Templates Backup****

Before making changes, take **ARM template backups** of both **source and target ADF instances** to verify the resource group and subscription details:

* Navigate to **Manage → ARM Templates** in ADF.
* Download the **existing templates**.
* Store them securely for reference.

## ****5. Creating a Pull Request (PR) and Merging Changes****

* Create a **Pull Request (PR)** in the repository.
* Add **PBI commit reference**.
* Request a **code review and approval**.
* Once approved, merge the **feature branch** into the **main branch**.

## ****6. Configuring Pipeline with ARM Template****

* Define a **new pipeline** for ADF deployment.
* Add **artifacts** and reference **ARM template configurations**.
* Ensure the correct **subscription and resource group** are referenced.

## ****7. Adding Parameters and Resource Configurations****

* Modify **ARM template parameters**:
  + ResourceId
  + Subscription
  + Parameter IR (Integration Runtime)
* Ensure parameter values are dynamically updated based on the **target environment**.

## ****8. Running the Pipeline and Merging Changes****

* Execute the pipeline **in a test environment**.
* Validate the changes using **logs and error reports**.
* Merge the pipeline changes into the **target ADF**.

## ****9. Final Deployment Validation****

* Verify the ADF **resource deployment**.
* Check the **data pipeline execution status**.
* Ensure all updates reflect correctly in the **target ADF instance**.

### ****Conclusion****

Following this workflow ensures a **structured and error-free deployment process** for Azure Data Factory. This approach allows for **efficient version control, tracking, and rollback options**, ensuring smooth ADF management in a production environment.

***End of Document***