Data Pipeline Architecture Design Document

Problem statement

Part 1: Design (Estimated Time: 1 hour)

Deliverable: A short design document (1–2 pages, PDF or Markdown)

Prompt: Design a data pipeline architecture that:

- Ingests JSON event data from multiple sources (e.g., mobile, web, APIs)
- Processes and transforms the data (e.g., enrich, clean, deduplicate)
- Stores the data in a format suitable for analytics and ML
- Supports scalability, monitoring, and fault tolerance

Include:

- Cloud services (AWS or Azure) you would use and why
- Data storage choices (e.g., S3, Redshift, Delta Lake)
- Technologies for orchestration and transformation (e.g., Airflow, Spark, dbt)
- DevOps considerations (CI/CD, testing, observability)
- A simple architecture diagram (optional but encouraged)

Solution for Part 1

Overview

This document outlines a scalable and robust data pipeline architecture for ingesting, processing, and storing JSON event data from multiple sources such as mobile apps, web apps, and third-party APIs. The solution leverages Azure services including Azure Databricks, Azure Data Lake Storage (ADLS), and Azure Key Vault. It ensures data is processed efficiently and stored in a format suitable for analytics and machine learning, while also providing scalability, monitoring, and fault tolerance. I am calling this project **Digital Finance**

Architecture Components

1. **Data Ingestion**
    - **Azure Event Hubs**: Acts as the ingestion layer to capture and stream JSON event data from multiple sources. It is scalable and can handle high-throughput data streams.
    - **Azure IoT Hub**: If your sources include IoT devices, Azure IoT Hub can be used to connect, monitor, and manage billions of IoT assets.
    - **Azure API Management (APIM)**: Fully managed service that enables organizations to publish, secure, transform, maintain, and monitor APIs at scale. It acts as a gateway for backend services, allowing developers to expose their APIs to external and internal consumers securely and efficiently. We can control the number of calls to restrict over use.

2. **Data Processing and Transformation**
    - **Azure Databricks**: Provides a collaborative environment for processing and transforming data. It is used to:
        - **PySpark Streaming**: Processes streaming data in real-time from Azure Event Hubs, enabling continuous data flow and transformation.
        - **PySpark UDFs (User-Defined Functions)**: Custom functions written in Python to perform complex transformations and enrichments on data.
        - Cleanse data: Handle missing values, remove duplicates, etc.
        - Enrich data: Augment data with additional information as needed.
        - Transform data: Convert data into a format suitable for analytics and ML.
    - **Delta Lake**: Utilized within Databricks for managing data in a reliable and performant manner, providing ACID transactions and scalable metadata handling.

3. **Data Storage**
    - **Azure Data Lake Storage (ADLS) Gen2**: Acts as the storage layer for both raw and processed data. It supports hierarchical namespaces and is optimized for big data analytics workloads.
    - **Delta Tables**: Store processed data in ADLS. They provide efficient query performance and support features like time travel and schema evolution.
4. **Machine Learning**

    Both Azure Databricks and Azure Machine Learning are powerful platforms for building and deploying machine learning models, each serving distinct purposes within the Azure ecosystem. Below is an overview of each service and how they can be used in the context of machine learning.

5. **Security and Access Management**
    - **Azure Key Vault**: Securely stores and manages access to sensitive information such as API keys, secrets, and credentials used in the pipeline.
    - **Azure Active Directory**: Manages authentication and authorization, ensuring secure access to resources.
6. **Monitoring and Fault Tolerance**
    - **Azure Monitor**: Provides monitoring capabilities for all Azure resources, including metrics, logs and alerts for the data pipeline.
    - **Azure Databricks /Apache Airflow Jobs**: Allows scheduling and monitoring of data processing tasks, with automatic retries and failure notifications. Airflow for orchestrating external applications and their dependencies while processing data.
7. **Disaster Recovery and Scalability**
    - **Geo-Redundant Storage (GRS)**: Ensures data in ADLS is replicated across regions for disaster recovery.
    - **Azure Databricks Autoscaling**: Automatically adjusts the number of workers based on the workload, ensuring cost efficiency and scalability.

8. **Dev Ops**

Continuous Integration and Continuous Deployment (CI/CD) with Jenkins in a data pipeline architecture can significantly enhance the development lifecycle by automating integration and deployment processes. Here's a more detailed explanation of how you can set up CI/CD for a data pipeline using Jenkins, particularly in the context of Azure.

CI/CD with Jenkins for a Data Pipeline

Overview

Continuous Integration (CI): Automates the integration of code changes from multiple contributors into a single project. It involves automated testing and validation to ensure that new changes do not break the existing functionality.

Continuous Deployment (CD): Automates the deployment of validated code changes to production or other environments. This ensures that new features and fixes are delivered to users efficiently and reliably.

Jenkins Setup

Jenkins Server Setup:

Installation: Deploy Jenkins on a virtual machine (ubuntu) in Azure, such as an Azure Virtual Machine.

Plugins: Install necessary plugins for integration with Azure services, such as the Azure Credentials Plugin, the Azure CLI Plugin, and the Jenkins DevOps Plugin.

Source Code Management:

Use a version control system like Git, hosted on platforms such as Azure Repos or GitHub, to manage your data pipeline scripts and configurations.

Pipeline as Code:

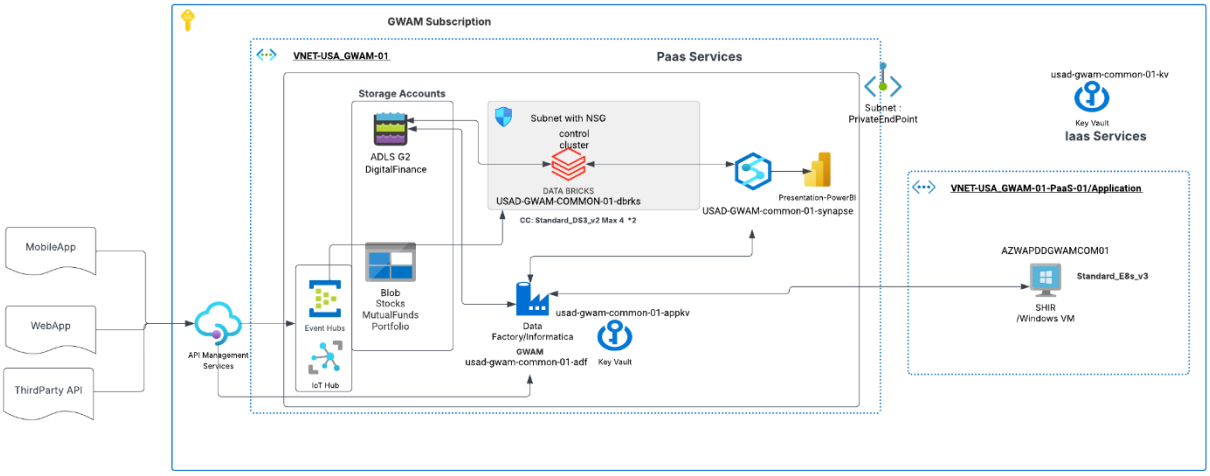
Use Jenkins Pipelines, defined via a Jenkinsfile, to specify the CI/CD process. This file is stored in the same repository as the code, making it easy to version control and manage.

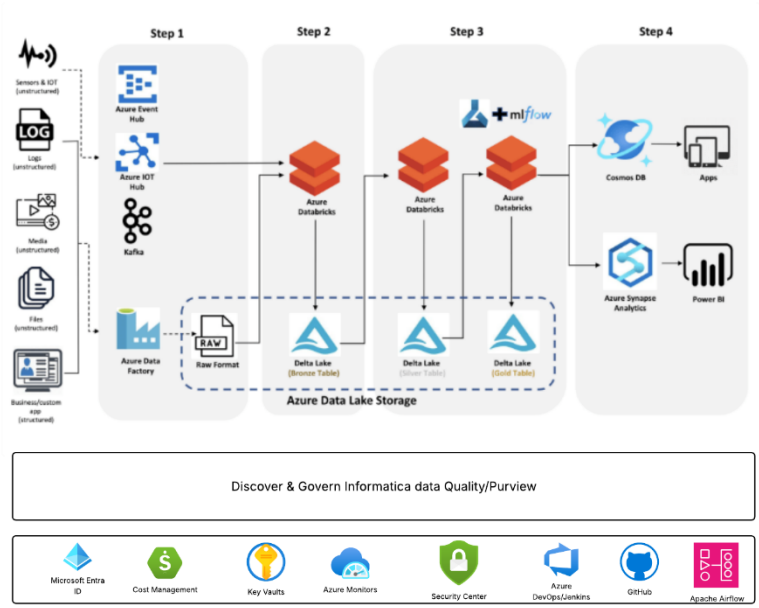
Data Flow

1. **Ingestion**: JSON event data is sent from various sources to Azure Event Hubs.
2. **Processing**:
    - **PySpark Streaming** in Azure Databricks reads data from Event Hubs in real-time, continuously processing the incoming stream.
    - The data is transformed and enriched using **PySpark UDFs** for custom logic.
    - Enriched and transformed data is written to Delta Tables.
3. **Storage**: Processed data is stored in ADLS Gen2 as Delta Tables, ready for analytics and machine learning.
4. **Access**: Data scientists and analysts access the data through Databricks Notebooks or other analytics tools connected to ADLS.

Architecture diagram

In this diagram, we can use informatica as ETL tool to connect with tools like markit edm.





Problem statement

Part 2: Implementation (Estimated Time: 2–3 hours)

Deliverable: A GitHub repo or zip file with code and a README

Prompt: Implement a simplified version of your pipeline using Python or PySpark. You may simulate data ingestion using sample JSON files.

Requirements:

- Read JSON files from a local or cloud storage location
- Perform basic transformations (e.g., flattening, timestamp parsing, deduplication)
- Write the transformed data to a target format (e.g., Parquet or CSV)
- Include unit tests for at least one transformation function
- Provide a README.md with setup instructions and a brief explanation of your approach

Solution

Our project, DIGITAL FINANCE, utilizes Databricks and PySpark for efficient financial data processing. We implement a three-tier architecture—Bronze, Silver and Gold layers—to manage data effectively.

Architecture Overview

- Bronze Layer: This is the raw data ingestion zone, capturing unprocessed data to ensure traceability.
- Silver Layer: Here, data is cleaned and standardized, making it suitable for preliminary analytics.
- Gold Layer: Data is enriched and aggregated for advanced analytics and business reporting.

Implementation

- Data Ingestion: Automated data ingestion from various sources into the bronze layer using Databricks.
- Processing with PySpark: Leverages PySpark for distributed processing, transforming data through silver and gold layers.
- Collaboration and Scalability: Databricks provides a collaborative and scalable environment, enabling efficient teamwork and handling of large datasets.

Benefits

- Scalability and Performance: Ensures high performance and scalability for large data volumes.
- Data Quality: Maintains data integrity and quality through structured processing.
- Business Value: Transforms raw data into actionable insights, aiding informed decision-making.

This approach optimizes data workflows and maximizes data value for the organization.

Problem Statement

Part 3: Reflection (Estimated Time: 30 minutes)

Deliverable: A short write-up (half-page to 1 page)

Prompt: Reflect on your design and implementation:

- What trade-offs did you make?
- What would you improve with more time or resources?
- How would you scale this solution for real-time ingestion?

Solution

Trade-offs

- **Complexity vs. Flexibility**: Using PySpark Streaming and UDFs provides flexibility to perform custom transformations, but it adds complexity to the pipeline. Managing real-time streaming and ensuring low latency can be challenging.
- **Cost vs. Performance**: Autoscaling in Databricks ensures performance during high loads but can lead to increased costs. Balancing these aspects is crucial for cost-effective operations. We can use horizontal and vertical scaling aspects of different services to enhance performance based on SLAs.
- **Real-time vs. Batch Processing**: While the solution supports real-time processing, it may require additional tuning and resources to ensure latency remains low as data volumes grow.
- **Azure :** Choosing Azure simplifies integration with its suite of services, unlike AWS or GCP, which often demand more extensive networking adjustments and security compliance configurations

Improvements with More Time or Resources

- **Enhanced Monitoring and Alerting**: Implement more granular monitoring and alerting mechanisms using Azure Monitor to detect anomalies and potential issues quickly.
- **Optimization of PySpark Jobs**: Continuously optimize PySpark jobs and UDFs for better performance, especially under high data throughput scenarios.

- **Data Quality Framework**: Build and integrate a data quality framework to validate data consistently and ensure high quality before it is used for analytics.
- **Automated Testing**: Develop automated testing for the pipeline components to ensure reliability and correctness in data processing.

Scaling for Real-time Ingestion

- **Partitioning and Sharding**: Optimize Event Hubs and Delta Tables by implementing effective partitioning and sharding strategies to handle increased data volumes.
- **Elastic Scaling of Event Hubs**: Use Event Hubs' auto-inflate feature to dynamically adjust the throughput units based on data load.
- **Advanced Caching Techniques**: Implement caching mechanisms to speed up data retrieval and processing. Azure Redis Cache could be integrated for this purpose.
- **Distributed Data Processing**: Leverage the full potential of distributed computing in Databricks by efficiently utilizing clusters and optimizing resource allocation.
- **Micro-batching**: Fine-tune PySpark Streaming to efficiently process micro-batches, adjusting the batch interval based on workload requirements to minimize latency.

Conclusion

This architecture provides a robust, scalable, and secure solution for ingesting, processing, and storing event data. By leveraging Azure's cloud services, it ensures high availability, disaster recovery, and efficient data management suitable for advanced analytics and machine learning applications. The use of PySpark Streaming and UDFs allows for real-time processing and complex data transformation tailored to specific business needs.

Advance features for Disaster recovery and High availability

Azure supports disaster recovery on ADLS Gen2 storage with geo-redundant storage (GRS) and read-access geo-redundant storage (RA-GRS), in which cases, data will be copied from the primary region to the secondary region asynchronously. Microsoft will determine the failover for a disaster in a region. Data will be accessible from the secondary region once the DR failover is completed.

RTO: 1 hour

RPO: 15 minutes

Data Factory

Azure Data Factory supports automatic DR failover. ADF metadata (code) such as pipelines, datasets, etc. will be restored as well as the Azure Integration Runtime (Auto Resolve IR).

Synapse Analytics

Requirement for RTO is 1 day and RPO is 4 hours

Azure support Synapse DR with snapshots and restore points only. Database snapshots are taken automatically every 4 hours. Those snapshots can be used for database restore points to DR failover.

SQL pool will be restored from snapshot backup during the failover.

RTO: 1 day

RPO: 4 hours

Similarly, we can prepare the DR plan for other services.