Shopping Card Data

Overview

This project implements a robust, real-time data pipeline for a shopping cart microservice. Leveraging AWS, Kafka, and Apache Airflow, it ensures data resilience, scalability, and actionable insights through a Medallion Data Lake architecture. By integrating real-time ingestion, transformation, and analytics, the pipeline enhances business intelligence and minimizes revenue loss due to data discrepancies.

The scope of this document is to present the architectural design, implementation details, and testing strategy for a data pipeline supporting a shopping cart microservice. By leveraging the Medallion architecture, the pipeline ensures data integrity, resilience, and analytics-readiness across its layers.

The proposed solution integrates a Medallion Data Lake architecture (Bronze, Silver, and Gold layers) with modern cloud technologies like Kafka, DynamoDB, and S3. This architecture ensures that raw purchase data is transformed into actionable insights in near real-time, enabling improved decision-making across business operations.

Throughout this document, we explain the tools and interactions required to meet the project's demands, detailing how each component contributes to the pipeline's objectives. To support the implementation, **Python scripts for orchestration and SQL models for transformation are provided as attachments**. These scripts and models enable automation, data validation, and seamless transitions between stages. Additionally, an **architectural diagram accompanies this document, visually illustrating the system's components, interactions, and data flow, and is referenced to provide clarity on the workflow.**

This document is structured as follows:

- Introduction: Business context, objectives, and scope.
- Architecture Overview: High-level description of the data pipeline and its components.
- Technical Implementation: Detailed explanation of the data flow, resilience mechanisms, and orchestration.
- Testing Strategy: Methods to validate pipeline reliability and performance.
- Conclusion: Summary of findings.

Objectives

- The primary objective is to design and implement a scalable, resilient, and analytics-ready data pipeline. Specific goals include:
    - Real-Time Ingestion: Capturing purchase data via API Gateway and Kafka with zero critical data loss.
    - Secure Persistence: Leveraging AWS DynamoDB for low-latency, scalable storage of raw purchase data.
    - Data Transformation: Applying the Medallion architecture to standardize, clean, and enrich data through Bronze, Silver, and Gold layers.
    - Resilience: Incorporating fallback mechanisms, retries, and alert systems to minimize data loss and downtime.
    - Actionable Insights: Delivering analytics-ready data to Amazon Redshift for visualization in tools like Power BI.

Architecture Overview

High-Level Data Flow

The data pipeline integrates several components to ensure a seamless, reliable, and scalable workflow:

- API Gateway: Receives and validates purchase data payloads.
- Kafka: Acts as the message broker, ensuring fault tolerance and scalability in data ingestion.
- DynamoDB: Temporarily stores raw data for immediate retrieval and further processing by Airflow DAGs.
- Apache Airflow: Orchestrates data transitions across the Medallion architecture:
    - Bronze Layer: Moves raw data from DynamoDB to S3 for initial storage.
    - Silver Layer: Cleanses and standardizes data using DBT models.
    - Gold Layer: Performs aggregations and enrichments for analytics-ready datasets.
- Amazon Redshift: Hosts the final Gold layer data for reporting and visualization via tools like Power BI.

Key Technologies

- Kafka: Chosen for its high durability and scalability in real-time processing.
- DynamoDB: Provides low-latency, highly scalable NoSQL storage for temporary data persistence.
- S3: Central storage backbone, optimized for cost-effective and long-term data storage in Parquet format.
- DBT: Manages data cleaning, transformation, and enrichment for the Silver and Gold layers.
- Amazon Redshift: Stores analytics-ready data for consumption by BI tools such as Power BI.

Data Flow and Orchestration:

- The pipeline uses Apache Airflow to orchestrate five primary DAGs that implement the Medallion architecture:
    - Kafka to DynamoDB: Consumes messages from Kafka and stores them in DynamoDB.
    - DynamoDB to Bronze Layer: Moves data to S3's Bronze layer for raw storage.
    - Bronze to Silver Layer: Cleanses and standardizes raw data using DBT.
    - Silver to Gold Layer: Performs enrichments and aggregations to prepare analytics-ready datasets.
    - Gold to Redshift: Loads Gold layer data into Redshift for reporting.
- Airflow's TriggerDagRunOperator connects DAGs, enabling seamless transitions and near real-time processing across the pipeline.

Error Handling and Resilience

- Retries and Fallbacks:
    - Airflow tasks include retry configurations to mitigate transient failures.
    - Kafka implements a Dead Letter Queue (DLQ) for failed messages, allowing reprocessing without data loss.
- Alerts:
    - Airflow sends email notifications for task or DAG failures to ensure rapid issue resolution.
    - Amazon CloudWatch monitors metrics and logs, triggering alerts for anomalies or critical errors.
- Data Durability:
    - S3 stores Parquet files with lifecycle policies for cost optimization and versioning for recovery.
    - Kafka supports replay capabilities for failed events.
- Monitoring and Observability:
    - Logs and metrics are aggregated in Amazon CloudWatch for centralized monitoring.
    - Power BI dashboards visualize pipeline performance and data quality metrics.

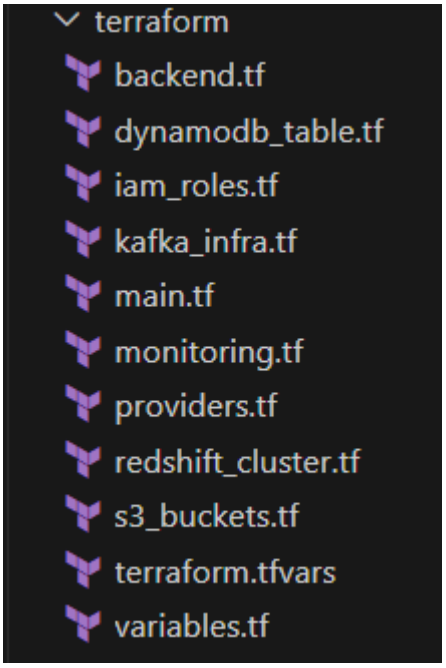
Resilience Strategies

- Asynchronous Processing: Ensures scalability and avoids bottlenecks from API Gateway through Kafka and DynamoDB.
- Infrastructure as Code (IaC): Terraform guarantees consistent, reproducible infrastructure deployments.
- Data Replay: Kafka and DLQ mechanisms allow reprocessing of failed records without full data re-ingestion.

Infrastructure as Code (IaC) with Terraform

Overview

Infrastructure as Code (IaC) ensures the consistency, scalability, and reproducibility of the infrastructure by defining and provisioning resources programmatically. This project uses Terraform to manage the deployment of critical components such as S3, DynamoDB, Kafka, IAM roles, monitoring, and Redshift. Below is an overview of the functionalities implemented in the project, with the Terraform file structure shown in the image below:



Provisioned Resources

- S3 Buckets (s3_buckets.tf) - Configures S3 buckets for the Bronze, Silver, and Gold layers with lifecycle policies for cost optimization and versioning for data protection.
- DynamoDB Table (dynamodb_table.tf) - Provisions a DynamoDB table with streams enabled for real-time data processing and auto-scaling to handle workload variations.
- Kafka Infrastructure (kafka_infra.tf) - Sets up Kafka topics for real-time data pipelines, ensuring fault tolerance and message durability through replication and Dead Letter Queues (DLQs).
- IAM Roles and Policies (iam_roles.tf) - Implements secure roles and policies to manage access to AWS resources while following the principle of least privilege.
- Redshift Cluster (redshift_cluster.tf) - Configures an analytics-focused Redshift cluster with scalable node types and restricted public access for data security.
- Monitoring and Alerting (monitoring.tf) - Deploys CloudWatch monitoring and alerting mechanisms to track system health and notify administrators of critical failures.
- Terraform Backend (backend.tf) - Manages Terraform state storage in S3 with locking enabled via DynamoDB to ensure safe collaboration and state consistency.
- Provider Configuration (providers.tf) - Specifies the AWS provider and region, serving as the foundation for managing infrastructure in Terraform.

Technical Implementation

Data Preparation: DynamoDB Configuration and Monitoring

Objective: Prepare DynamoDB for real-time data ingestion by enabling streams, setting up auto-scaling, and implementing monitoring to ensure reliability and performance.

Pipeline Stage

- Enable DynamoDB Streams:
    - DynamoDB Streams are configured to capture changes in the purchase_orders table, including INSERT and MODIFY events.
    - The stream is set to deliver the NEW_IMAGE view for complete data records.
- Set Up Auto-Scaling:
    - Auto-scaling policies are applied to manage read and write capacity dynamically, ensuring the table can handle varying workloads.
- Validate and Enforce Schema:
    - Records written to the DynamoDB table are validated at the API Gateway level to ensure schema compliance.
- Monitor Streams and Capacity:
    - CloudWatch metrics and alarms are configured to monitor ThrottledReads, ThrottledWrites, and StreamViewLag.

Airflow DAG: Monitor DynamoDB Streams

(dag/monitor_dynamodb_streams.py)
Responsibilities:

- Listen to DynamoDB Streams for new data events.
- Trigger the DynamoDB to Bronze DAG upon detecting new records.

Trigger:

- Events in the DynamoDB stream automatically notify the Airflow sensor, initiating the next stage of the pipeline.

Connections Required:

- Airflow to DynamoDB Streams: An aws_default connection is used to poll DynamoDB Streams.
- DynamoDB to CloudWatch: Metrics are integrated into CloudWatch for monitoring table and stream performance.

Data Ingestion: API Gateway to Kafka

Objective: Ingest purchase data through API Gateway, validate it with a Lambda function, and forward it to Kafka for reliable queuing. Kafka ensures fault tolerance and prevents data loss during high loads.

Pipeline Stage:

- API Gateway receives HTTP requests containing purchase data (buyer_id, product_id, total_amount, etc.).
- The API Gateway triggers an AWS Lambda function to validate the payload and send it to a Kafka topic (purchase_orders).

Airflow DAG: Kafka to DynamoDB (***dag/1-kafka_to_dynamo.py***)

- Responsibilities:
    - Consume messages from the Kafka topic.
    - Validate message schemas to ensure correctness.
    - Write validated data into DynamoDB.
    - Trigger the DynamoDB to Bronze DAG.

Connections Required:

- API Gateway to Lambda: API Gateway is configured to trigger a Lambda function upon receiving HTTP requests.
- Lambda to Kafka: Kafka bootstrap servers, security protocols, and authentication must be defined in the Lambda configuration.
- Airflow to Kafka: The kafka_default connection in Airflow specifies Kafka's broker details for consuming messages.
- Airflow to DynamoDB: An aws_default connection in Airflow provides the necessary credentials for writing to DynamoDB.

Data Persistence: Kafka to DynamoDB

Objective: Export validated data from DynamoDB to the Bronze Layer in Amazon S3.

Pipeline Stage:

- DynamoDB Streams detects new entries in the **purchase_orders** table.
- An Airflow DAG monitors DynamoDB Streams, extracts the data, and writes it to the Bronze Layer as Parquet files in S3.

Airflow DAG: DynamoDB to Bronze (***dag/2-dynamodb_to_bronze_dag.py***)

- Responsibilities:
    - Monitor DynamoDB for new data using sensors.
    - Export data to the Bronze Layer bucket in S3.
    - Trigger the Bronze to Silver DAG upon completion.

Connections Required:

- Airflow to Kafka: Configure a **kafka_default** connection in the Airflow with the Kafka broker details.
- Airflow to DynamoDB: An **aws_default** connection in Airflow is used to write Parquet files to the Bronze Layer bucket.

Data Transformation: Bronze to Silver

Objective: Transform raw data in the Bronze Layer into a structured format in the Silver Layer

Pipeline Stage

- An Airflow DAG triggers DBT models to clean and standardize data from the Bronze Layer.
- The transformed data is written to the Silver Layer bucket in S3.

Airflow DAG: Bronze to Silver (***dag/3-bronze_to_silver_dag.py***)

- Responsibilities:
    - Run DBT models to clean and structure raw data.
    - Validate the output using DBT tests.
    - Trigger the Silver to Gold DAG after successful validation

Connections Required:

- Airflow to DBT: Airflow uses the BashOperator to trigger DBT commands (dbt run and dbt test).
- DBT to S3: DBT transformations are configured to read from the Bronze Layer bucket and write to the Silver Layer bucket.

Data Aggregation: Silver to Gold

Objective: Generate analytics-ready data by performing aggregations and creating metrics.

Pipeline Stage:

- An Airflow DAG triggers advanced DBT models to calculate metrics and prepare datasets.
- The results are saved in the Gold Layer bucket in S3.

Airflow DAG: Silver to Gold (***dag/4-silver_to_gold_dag.py***)

- Responsibilities:
    - Run DBT models for aggregations and calculations.
    - Validate the Gold Layer data with DBT tests.
    - Trigger the Gold to Data Warehouse DAG.

Connections Required:

- Airflow to DBT: Similar to the Bronze to Silver process, Airflow triggers DBT models for transformation.
- DBT to S3: The Silver Layer serves as the input for DBT, and the Gold Layer bucket stores the output

Reporting: Gold Layer to Data Warehouse and Power BI

Objective: Make Gold Layer data available for business intelligence reporting.

Pipeline Stage:

- The Gold Layer data is ingested into Amazon Redshift using Airflow.
- Power BI connects directly to Redshift for real-time reporting and visualization.

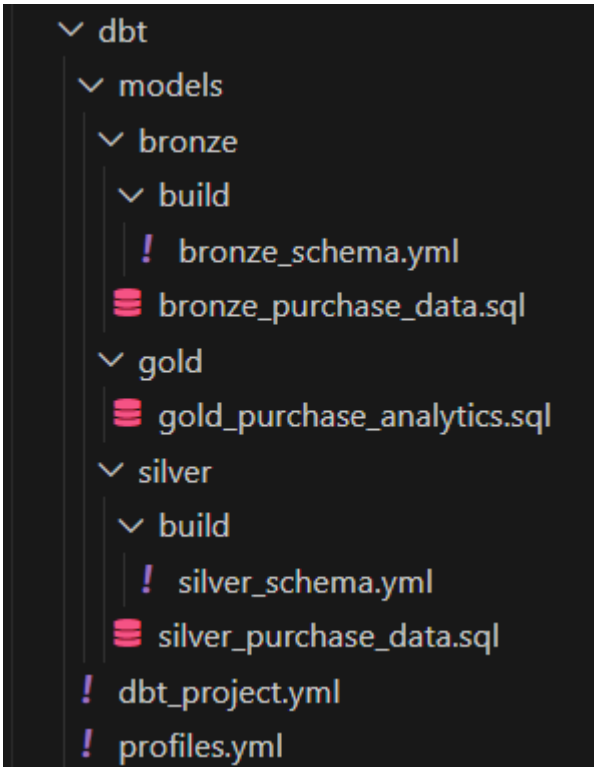
Airflow DAG: Gold to Data Warehouse (***dag/5-gold_to_redshift_dag.py***)

- Responsibilities:
    - Use Amazon Redshift's COPY command to load data from S3 into Redshift.
    - Notify stakeholders or update logs after successful ingestion.

Connections Required:

- Airflow to Redshift: A redshift_default connection in Airflow allows SQL operations on Redshift.
- Power BI to Data Warehouse: Power BI is configured to connect directly to Redshift using native connectors.

Repository Structure:



The files corresponding to the SQL scripts are attached to this document, along with the previously mentioned diagram.

Orchestration

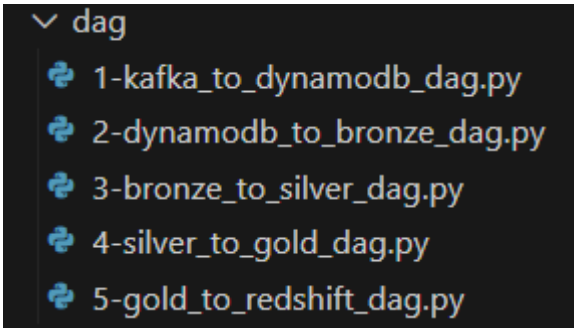
Each stage is represented by a dedicated DAG, orchestrated with TriggerDagRunOperator for seamless transitions:

1. Kafka to DynamoDB DAG:
    a. Consumes Kafka messages and writes to DynamoDB.
    b. Triggers the DynamoDB to Bronze DAG upon completion.
2. DynamoDB to Bronze DAG:
    a. Exports DynamoDB data to the Bronze Layer in S3.
    b. Triggers the Bronze to Silver DAG upon completion.
3. Bronze to Silver DAG:
    a. Processes raw data into a structured Silver Layer using DBT.
    b. Triggers the Silver to Gold DAG upon completion.
4. Silver to Gold DAG:
    a. Performs aggregations and saves analytics-ready data in the Gold Layer.
    b. Triggers the Gold to Redshift DAG.
5. Gold to Redshift DAG:
    a. Loads data into Redshift or makes it available via Athena for Power BI.

Global Connections:

- Airflow to Kafka: Use the KafkaHook and a kafka_default connection with broker details.
- Airflow to AWS Services: Use the aws_default connection for DynamoDB, S3, and Redshift.
- Airflow to DBT: Integrate DBT commands within Airflow DAGs using BashOperator.
- Power BI to Data Warehouse: Establish a direct connection to Redshift or Athena for reporting.

Repository Structure:
The DAG scripts within the repository are structured as shown below:



This structure corresponds to the pipelines visualized in the attached diagram. The DAG scripts are numbered (1 to 5) to match the respective processes in the diagram.

The files corresponding to the DAG scripts are attached to this document, along with the previously mentioned diagram.

Testing Strategy

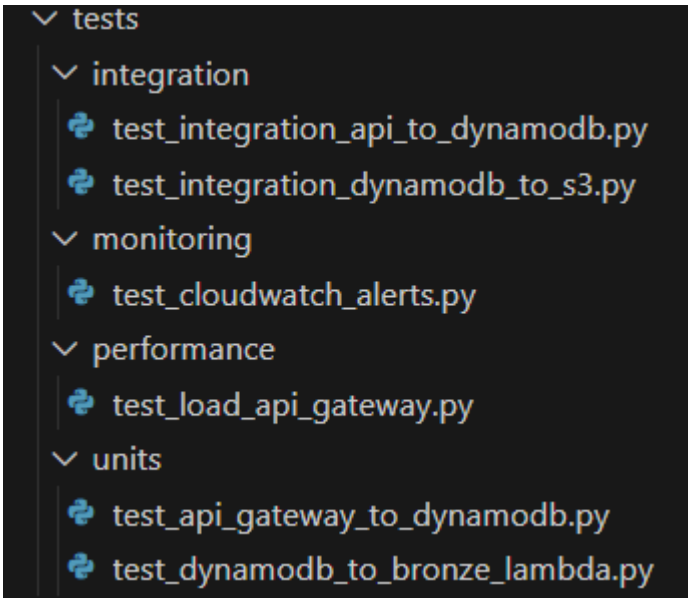
Overview

To ensure the reliability, resilience, and accuracy of the pipeline, a comprehensive testing strategy is implemented. This includes unit tests, integration tests, data validation, resilience testing, performance benchmarks, and monitoring validation. Testing guarantees that the system meets functional and non-functional requirements, minimizes failures, and ensures smooth operation under different conditions.

The testing structure is organized into the following well-defined categories:

- Integration Tests: Validate the interaction between services, such as API Gateway, DynamoDB, S3, and Redshift.
- Monitoring Tests: Ensure CloudWatch alerts, logs, and SNS triggers are correctly configured for observability.
- Performance Tests: Assess system throughput, latency, and other performance metrics for critical components.
- Unit Tests: Test isolated functionalities, such as Lambda error handling or specific logic.
- Security Tests: Validate IAM roles, S3 bucket policies, and KMS key rotations to adhere to security standards.
- End-to-End Tests: Simulate the complete data pipeline to validate data flow and ensure the accuracy and integrity of the entire workflow.

Additionally, the testing strategy can follow a structured organization like the one shown below:



The attached diagram also highlights specific testing points within the architecture:

Number 7 corresponds to the **test_api_gateway_to_dynamodb** unit test, which validates the integration of API Gateway with DynamoDB.

Number 8 corresponds to the **test_dynamodb_to_bronze_lambda** unit test, which verifies the data ingestion flow from DynamoDB to the Bronze layer through AWS Lambda.

Conclusion

This proposal outlines an end-to-end data pipeline that emphasizes scalability, resilience, and near real-time data availability. By adopting the **Medallion Architecture**—dividing data into Bronze, Silver, and Gold layers—the pipeline ensures the transformation of raw purchase data into structured, analytics-ready datasets optimized for business intelligence.

Each component in the pipeline is strategically selected to fulfill specific roles:

- Apache Kafka ensures reliable message queuing, fault tolerance, and scalability during data ingestion.
- AWS DynamoDB and S3 provide robust storage solutions for temporary and long-term data persistence.
- DBT enables efficient data cleaning, validation, and enrichment to support high-quality transformations.
- Amazon Redshift integrates seamlessly with tools like Power BI, delivering actionable insights for decision-making.

The process is **orchestrated by Apache Airflow**, which ensures fault-tolerant execution, automated retries, and smooth transitions across pipeline stages. Its modular DAG design not only facilitates debugging and maintenance but also enables easy extensibility as business requirements evolve.

This architecture delivers a seamless flow of data from ingestion to reporting, ensuring that business intelligence tools consistently have fresh, accurate, and reliable datasets. The implementation prioritizes resilience, scalability, and performance, equipping businesses with the insights needed to make timely and informed decisions while minimizing data-related risks.