**Shopware Data Pipeline**

**Overview**

The Shopware Data Pipeline is a robust data engineering solution crafted to collect, process, and analyze data from four sources—two streaming (Web Traffic Logs, CRM Interactions) and two batch (POS Data, Inventory Management Data)—to empower decision-making across Shopware teams. It leverages the Medallion Architecture (Bronze, Silver, Gold layers) to transform raw data into actionable insights, supporting KPI tracking and access via ad-hoc querying, dashboards, and data marts.

**High-Level Architecture**

**Project Objectives**

1. **Data Integration**: Seamlessly integrate data from diverse sources, including batch and streaming data.
2. **Data Transformation**: Clean and transform raw data to align with business requirements.
3. **Data Accessibility**: Enable teams to access data through ad-hoc queries, dashboards, or data marts.
4. **Data Storage**: Efficiently organize data using data lakes, warehouses, and marts.
5. **KPI Tracking**: Facilitate tracking of department-specific KPIs.

**Medallion Architecture**

The pipeline adopts a data lakehouse approach with three layers:

1. **Bronze Layer (Raw Data)**: Initial ingestion point for raw data from source systems, chosen for its ability to handle unprocessed data at scale without immediate transformation.
2. **Silver Layer (Processed Data)**: Stores cleaned and validated data, selected for its role in providing a reliable intermediate state for further processing.
3. **Gold Layer (Business Insights)**: Holds aggregated, analytics-ready data in Redshift and S3, preferred for its optimization for business intelligence and reporting needs.

**Data Sources**

The pipeline processes data from four sources:

* **POS Data (Batch, Daily)**: Sales transactions (quantity, revenue, discounts) for Sales, Operations, and Finance teams, managed in batches for structured, periodic updates.
* **Inventory Management Data (Batch, Hourly)**: Real-time inventory and restocking data for Operations and Sales, batched hourly to balance timeliness and processing overhead.
* **Web Traffic Logs (Streaming, Real-Time)**: User behavior and session data for Marketing and Data Analysts, streamed for immediate insights into user activity.
* **CRM Interactions (Streaming, Real-Time)**: Customer interactions and feedback for Marketing and Customer Support, streamed to enable real-time support and engagement analysis.

**Components**

**Detailed Architecture**

**Data Ingestion**

1. **Batch Data (POS, Inventory)**:
   * **S3 Data Lake**: Used as the landing zone for raw batch data in parquet format, chosen for its durability, scalability, and cost-effectiveness for storing large datasets.
   * **Lambda Functions**: Employed to process and move data from Bronze to Silver, selected for its event-driven nature and ability to handle small, specific tasks efficiently.
2. **Streaming Data (Web Traffic, CRM Interactions)**:
   * **API Gateway Webhooks**: Implemented to accept pushed data and forward it to Kinesis via Lambda proxies, chosen for its scalability and ease of integrating external data sources.
   * **ECS Fargate Connectors**: Utilized to poll Web Traffic (/api/web-traffic/) and CRM Interactions (/api/customer-interaction/) endpoints, sending data to Kinesis, selected for its containerized, serverless scalability and management simplicity.
   * All streaming data is processed by Lambda functions for real-time computation, preferred for its low-latency execution.

**Data Storage & Processing**

1. **Kinesis Data Streams**: Manages real-time streaming for Web Traffic and CRM Interactions, chosen for its ability to handle high-throughput data streams with low latency.
2. **Amazon ECS (Fargate)**: Runs Docker containers for batch data connectors, selected for its auto-scaling based on CPU/memory usage and seamless integration with Kinesis.
3. **S3 Data Lake**: Stores data across Bronze, Silver, and Gold layers, preferred for its virtually unlimited storage and support for diverse data formats.
4. **AWS Glue**: Handles ETL transformations and metadata management, chosen for its serverless ETL capabilities and integration with the Medallion Architecture.
5. **AWS Lambda**: Processes streaming data and computes KPIs, selected for its cost efficiency and ability to trigger actions in real-time.
6. **EventBridge and Step Functions (Batch Data)**: EventBridge triggers Step Functions for Bronze-to-Silver and Silver-to-Gold transformations, chosen for their event-driven automation and orchestration of complex workflows.
7. **Amazon Redshift**: Stores KPI results, preferred for its optimized query performance and integration with Power BI for analytics.

**Analytics & Visualization**

1. **Amazon Athena**: Enables SQL queries on S3 data, chosen for its serverless query capability without managing infrastructure.
2. **Power BI**: Connects to Redshift for KPI dashboards, selected for its powerful visualization tools and enterprise-grade reporting features.
3. **Data Marts**: Provides team-specific aggregated data, implemented to enhance accessibility and focus on relevant metrics.

**Key Performance Indicators (KPIs)**

**Sales Team (via POS, Inventory Data)**

* Total Sales by Region/Product
* Stock Availability
* Product Turnover Rate

**Marketing Team (via Web Traffic, CRM Interactions)**

* Customer Engagement Score
* Session Duration & Bounce Rate
* Loyalty Activity Rate

**Operations Team (via Inventory, POS)**

* Inventory Turnover
* Restock Frequency
* Stockout Alerts

**Customer Support Team (via CRM Interactions)**

* Feedback Score
* Interaction Volume by Type
* Time-to-Resolution

**Setup Instructions**

**Prerequisites**

* AWS Account with appropriate permissions
* AWS CLI configured locally
* Docker installed locally
* Terraform installed locally
* Python 3.11 or higher

**Deployment Steps**

1. **Clone the Repository**:
2. git clone https://github.com/Amoako419/Shopware.git

cd shopware

1. **Build and Push Docker Images**:
2. # For CRM Logs Connector
3. cd api-gw-webhooks/crm-logs-infra/scripts
4. ./build\_push\_ecr.sh
5. # For Web Logs Connector
6. cd api-gw-webhooks/web-logs-infra/scripts

./build\_push\_ecr.sh

1. **Deploy Infrastructure with Terraform**:
2. # For CRM Logs Infrastructure
3. cd api-gw-webhooks/crm-logs-infra/terraform
4. terraform init
5. terraform apply
6. # For Web Logs Infrastructure
7. cd api-gw-webhooks/web-logs-infra/terraform
8. terraform init

terraform apply

1. **Deploy Glue Jobs and Step Functions**:
   * Upload Glue scripts to S3.
   * Create and configure Glue jobs and Step Functions using the AWS Console or Terraform.
   * Configure EventBridge rules to trigger Step Functions for batch data processing.
2. **Set Up Redshift and Power BI**:
   * Create a Redshift cluster and configure access.
   * Connect Power BI to Redshift for dashboard creation.
3. **Verify Deployment**:
   * Check AWS Console to ensure all resources are created.
   * Test webhook endpoints for streaming data.
   * Monitor CloudWatch logs for connector applications and Step Functions.

**Configuration**

Key configuration files:

* api-gw-webhooks/crm-logs-infra/terraform/terraform.tfvars: CRM pipeline configuration
* api-gw-webhooks/web-logs-infra/terraform/terraform.tfvars: Web traffic pipeline configuration
* api-gw-webhooks/crm-logs-infra/connector/.env: CRM connector environment variables
* api-gw-webhooks/web-logs-infra/connector/.env: Web connector environment variables

**Data Flow**

**Bronze Layer (Data Ingestion)**

1. **Batch Data Collection (POS, Inventory)**:
   * Lambda moves data from the landing bucket in parquet format:
     + POS: Daily updates.
     + Inventory: Hourly updates.
   * Data is sent to Kinesis Data Streams and stored in S3 Bronze buckets.
2. **Streaming Data Collection (Web Traffic, CRM Interactions)**:
   * **ECS Fargate Connectors**: Poll web traffic data from /api/web-traffic/ and CRM interaction data from /api/customer-interaction/, sending to Kinesis, chosen for their scalability and reliability.
   * **API Gateway Webhooks**: Accept pushed data and forward via Lambda proxies to Kinesis, selected for their flexibility with external integrations.
   * Data is processed by Lambda and stored in S3 Bronze buckets.

**Silver Layer (Data Processing)**

1. **Batch Data**:
   * EventBridge detects new data in the Bronze layer.
   * Triggers a Step Function to run AWS Glue jobs for cleaning, validation, and transformation, chosen for their orchestrated workflow management.
   * Results are stored in S3 Silver buckets.
2. **Streaming Data**:
   * AWS Lambda processes streaming data in real-time, selected for its low-latency processing.
   * Data is cleaned and transformed, then stored in Redshift.

**Gold Layer (Analytics)**

1. **Batch Data**:
   * EventBridge detects processed data in the Silver layer.
   * Triggers a Step Function to compute KPIs using AWS Glue jobs, preferred for its batch processing efficiency.
   * Results are stored in Redshift and S3 Gold buckets.
2. **Streaming Data**:
   * AWS Lambda computes KPIs for Marketing and Customer Support teams, chosen for its real-time capability.
   * Results are stored in Redshift.
3. **Analytics Access**:
   * Amazon Athena for SQL queries on S3 data.
   * Power BI for dashboards connected to Redshift.
   * Notebooks for data science workflows via SageMaker.

**Monitoring and Maintenance**

1. **CloudWatch Monitoring**: Tracks logs and metrics for Kinesis, Lambda, ECS, and Step Functions, chosen for its comprehensive monitoring and alerting capabilities.
2. **Error Handling**: Implements automatic retries, dead-letter queues, and SNS notifications, selected for their robustness in managing failures.
3. **Scaling**: ECS scales based on usage, Kinesis supports resharding, and Glue adjusts DPUs, preferred for their adaptive performance management.

**Troubleshooting**

**Common Issues**

1. **Connector Not Sending Data**:
   * Check CloudWatch logs for errors
   * Verify network connectivity to source APIs
   * Ensure IAM permissions are correctly configured
2. **Glue Job Failures**:
   * Check job logs in CloudWatch
   * Verify input data schema matches expectations
   * Check for sufficient IAM permissions
3. **Missing Data in Analytics**:
   * Verify Glue crawlers have run successfully
   * Check S3 bucket permissions
   * Ensure data partitioning is correctly configured

**CI/CD with GitHub Actions**

Our pipeline uses GitHub Actions for continuous integration and deployment, automating critical processes:

**Infrastructure Deployment**

* **Terraform Validation**: Validates Terraform configurations on pull requests, chosen to catch errors early.
* **Infrastructure Deployment**: Deploys AWS changes after merging to main, selected for automation efficiency.
* **Security Scanning**: Runs checkov and tfsec, preferred for identifying security vulnerabilities.

**Code Quality**

* **Python Linting**: Enforces style with flake8 and black, chosen for code consistency.
* **Type Checking**: Validates type hints with mypy, selected for type safety.
* **Unit Tests**: Runs pytest suite, preferred for verifying functionality.
* **Integration Tests**: Tests end-to-end data flow on staging, chosen for system validation.

**Container Management**

* **Docker Image Building**: Builds images for ECS Fargate connectors, selected for containerized deployment.
* **Image Security Scanning**: Scans with Trivy, chosen for vulnerability detection.
* **ECR Publishing**: Pushes validated images to Amazon ECR, preferred for secure distribution.

**Data Quality**

* **Schema Validation**: Validates data schemas, chosen to ensure data integrity.
* **Data Test Suite**: Runs dbt tests, selected for transformation accuracy.
* **Documentation**: Auto-generates and publishes data docs, preferred for maintainability.

Workflows are defined in .github/workflows/ and triggered on:

* Pull request creation/updates
* Merges to main branch
* Scheduled runs for security scanning
* Manual triggers for emergency fixes

**Contact**

For questions or support, please contact the data engineering team.

**Contributing**

1. Fork the repository
2. Create your feature branch (git checkout -b feature/AmazingFeature)
3. Commit your changes (git commit -m 'Add some AmazingFeature')
4. Push to the branch (git push origin feature/AmazingFeature)
5. Open a Pull Request

**Code Style**

* Python code should follow PEP 8
* Use meaningful variable and function names
* Include docstrings for all functions and classes
* Add type hints where applicable

**Testing**

* Write unit tests for new features
* Ensure all tests pass before submitting PR
* Include integration tests where necessary

**Security**

* Never commit sensitive credentials
* Use AWS Secrets Manager for sensitive data
* Follow least privilege principle for IAM roles

**Authors**

* **Amoako Heskey** - *Contributor*
* **Andrew Marfo** - *Contributor*
* **Charles Adu Nkansah** - *Contributor*
* **Ann-Vanessa Lartey** - *Contributor*

**License**

This project is licensed under the MIT License - see the LICENSE file for details

**Acknowledgments**

* AWS Documentation
* Medallion Architecture Best Practices
* Data Engineering Community