PRJ-SAP-016: Serverless Data Analytics Pipeline

Certification: AWS Certified Solutions Architect – Professional
Domain: Data & Analytics

1. Business Context

Modern organizations generate massive volumes of data from various sources, including application logs, IoT devices, and transactional databases. Traditional data warehousing solutions often struggle to scale efficiently and can be cost-prohibitive due to the need to provision and manage underlying infrastructure.

This project addresses these challenges by implementing a **Serverless Data Analytics Pipeline** on AWS. By leveraging a data lake architecture centered around Amazon S3, organizations can ingest, store, process, and analyze vast amounts of data without managing any servers. This approach decouples storage from compute, allowing each to scale independently, optimizing costs, and enabling agile data exploration.

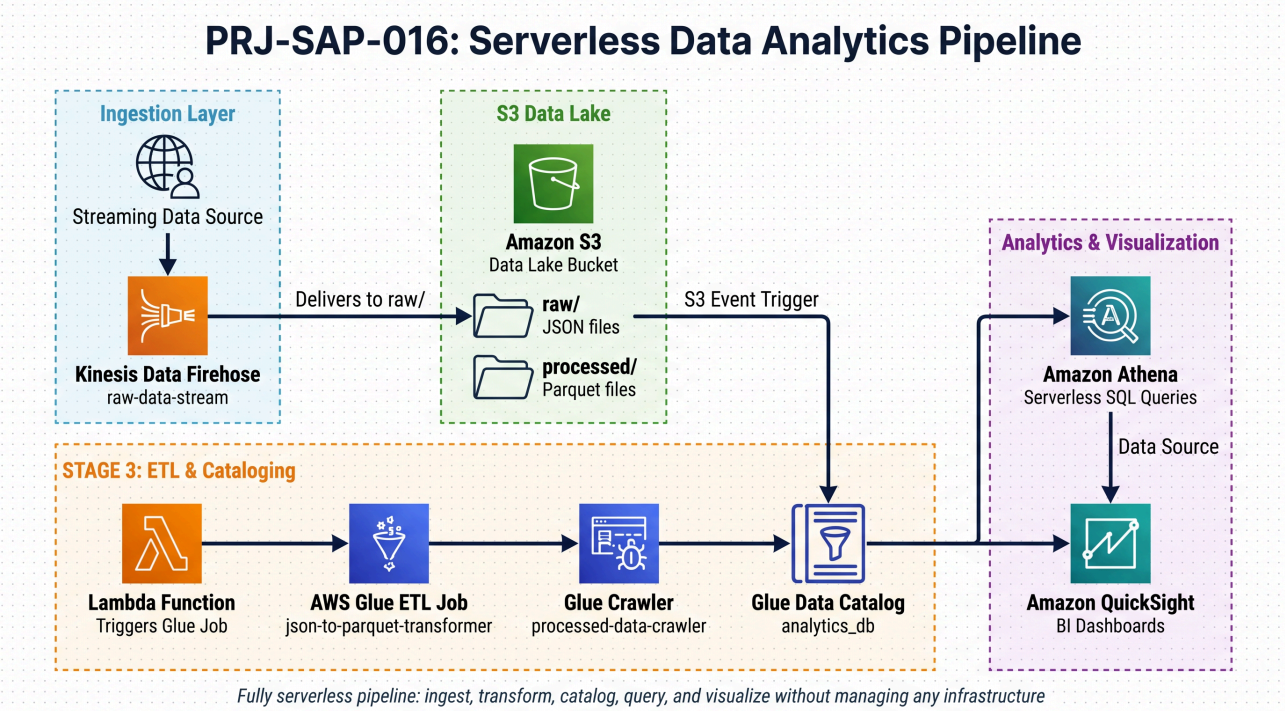
GRC Mapping (Governance, Risk, and Compliance)

Control Area	Implementation Details
Data Retention & Lifecycle	Amazon S3 lifecycle policies can be applied to automatically transition older data to cheaper storage classes (e.g., S3 Glacier) or delete it according to compliance requirements.
Access Control	AWS IAM policies and S3 bucket policies ensure that only authorized services (like Glue and Athena) and users have access to raw and processed data.
Data Encryption	S3 Server-Side Encryption (SSE-S3 or SSE-KMS) ensures data is encrypted at rest. TLS is used for data in transit via Kinesis and API calls.
Audit Logging	AWS CloudTrail logs all API actions across S3, Glue, Athena, and QuickSight, providing a comprehensive audit trail for compliance monitoring.

---

2. Architecture Overview

The architecture is a modular, event-driven pipeline that leverages serverless components for scalability and cost-effectiveness.



Fully serverless pipeline: ingest, transform, catalog, query, and visualize without managing any infrastructure

Pipeline Stages:

1. **Ingestion:**
    - **Streaming Data: Kinesis Data Firehose** provides a fully managed service to capture, transform, and load streaming data. It automatically batches, compresses, and delivers data to S3.
2. **Storage (Data Lake):**
    - **Amazon S3** is used as the central data lake.
    - **Raw Zone:** The `raw/` prefix stores the data in its original, unaltered format (e.g., JSON).
    - **Processed Zone:** The `processed/` prefix stores the data after it has been cleaned, transformed, and partitioned, converted to a columnar format like **Apache Parquet**.
3. **ETL & Cataloging:**
    - An S3 `PutObject` event in the raw zone triggers an **AWS Lambda function**.
    - The Lambda function starts an **AWS Glue ETL job** (serverless Apache Spark) to perform the transformation (JSON to Parquet).
    - An **AWS Glue Crawler** runs to infer the schema and update the **AWS Glue Data Catalog**.
4. **Analytics & Visualization:**
    - **Amazon Athena:** A serverless query service using the Glue Data Catalog to run standard SQL queries directly on the Parquet files in S3.
    - **Amazon QuickSight:** A cloud-native BI service connecting to Athena to create interactive dashboards.

---

3. Infrastructure as Code (IaC) Templates

Option A: Terraform Template (`main.tf`)

```
provider "aws" {
  region = "us-east-1"
}

variable "project_name" {
  default = "prj-sap-016"
}

# S3 Data Lake Bucket
resource "aws_s3_bucket" "data_lake" {
  bucket_prefix = "${var.project_name}-datalake-"
  force_destroy = true
}

resource "aws_s3_object" "raw_folder" {
  bucket = aws_s3_bucket.data_lake.id
  key    = "raw/"
}

resource "aws_s3_object" "processed_folder" {
  bucket = aws_s3_bucket.data_lake.id
  key    = "processed/"
}

# IAM Role for Firehose
resource "aws_iam_role" "firehose_role" {
  name = "${var.project_name}-firehose-role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Action = "sts:AssumeRole"
      Effect = "Allow"
      Principal = { Service = "firehose.amazonaws.com" }
    }]
  })
}

resource "aws_iam_role_policy" "firehose_policy" {
  name = "${var.project_name}-firehose-policy"
  role = aws_iam_role.firehose_role.id
```

```
  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Action = ["s3:AbortMultipartUpload", "s3:GetBucketLocation",
"s3:GetObject", "s3:ListBucket", "s3:ListBucketMultipartUploads",
"s3:PutObject"]
      Effect = "Allow"
      Resource = [aws_s3_bucket.data_lake.arn,
"${aws_s3_bucket.data_lake.arn}/*"]
    }]
  })
}

# Kinesis Firehose Delivery Stream
resource "aws_kinesis_firehose_delivery_stream" "raw_stream" {
  name        = "${var.project_name}-raw-stream"
  destination = "s3"

  s3_configuration {
    role_arn           = aws_iam_role.firehose_role.arn
    bucket_arn         = aws_s3_bucket.data_lake.arn
    prefix             = "raw/"
    buffer_interval    = 60
    buffer_size        = 1
  }
}

# IAM Role for Glue
resource "aws_iam_role" "glue_role" {
  name = "${var.project_name}-glue-role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Action = "sts:AssumeRole"
      Effect = "Allow"
      Principal = { Service = "glue.amazonaws.com" }
    }]
  })
}

resource "aws_iam_role_policy_attachment" "glue_service" {
  role       = aws_iam_role.glue_role.name
  policy_arn = "arn:aws:iam::aws:policy/service-role/AWSGlueServiceRole"
}

resource "aws_iam_role_policy" "glue_s3_policy" {
```

```
  name = "${var.project_name}-glue-s3-policy"
  role = aws_iam_role.glue_role.id
  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Action = ["s3:GetObject", "s3:PutObject", "s3:DeleteObject",
"s3:ListBucket"]
      Effect = "Allow"
      Resource = [aws_s3_bucket.data_lake.arn,
"${aws_s3_bucket.data_lake.arn}/*"]
    }]
  })
}

# Glue Database
resource "aws_glue_catalog_database" "analytics_db" {
  name = "${var.project_name}_analytics_db"
}

# Glue Crawler
resource "aws_glue_crawler" "processed_crawler" {
  database_name = aws_glue_catalog_database.analytics_db.name
  name          = "${var.project_name}-processed-crawler"
  role          = aws_iam_role.glue_role.arn

  s3_target {
    path = "s3://${aws_s3_bucket.data_lake.bucket}/processed/"
  }
}

output "s3_bucket_name" {
  value = aws_s3_bucket.data_lake.bucket
}
output "firehose_stream_name" {
  value = aws_kinesis_firehose_delivery_stream.raw_stream.name
}
```

Option B: CloudFormation Template (`template.yaml`)

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: 'PRJ-SAP-016: Serverless Data Analytics Pipeline'

Resources:
  DataLakeBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: !Sub 'prj-sap-016-datalake-${AWS::AccountId}'

  FirehoseRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: firehose.amazonaws.com
            Action: sts:AssumeRole
      Policies:
        - PolicyName: FirehoseS3Policy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action:
                  - s3:AbortMultipartUpload
                  - s3:GetBucketLocation
                  - s3:GetObject
                  - s3:ListBucket
                  - s3:ListBucketMultipartUploads
                  - s3:PutObject
                Resource:
                  - !GetAtt DataLakeBucket.Arn
                  - !Sub '${DataLakeBucket.Arn}/*'

  RawDataStream:
    Type: AWS::KinesisFirehose::DeliveryStream
    Properties:
      DeliveryStreamName: prj-sap-016-raw-stream
      S3DestinationConfiguration:
        BucketARN: !GetAtt DataLakeBucket.Arn
        Prefix: 'raw/'
```

```
      RoleARN: !GetAtt FirehoseRole.Arn
      BufferingHints:
        IntervalInSeconds: 60
        SizeInMBs: 1

GlueRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service: glue.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-role/AWSGlueServiceRole
    Policies:
      - PolicyName: GlueS3Policy
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
            - Effect: Allow
              Action:
                - s3:GetObject
                - s3:PutObject
                - s3:DeleteObject
                - s3:ListBucket
              Resource:
                - !GetAtt DataLakeBucket.Arn
                - !Sub '${DataLakeBucket.Arn}/*'

AnalyticsDatabase:
  Type: AWS::Glue::Database
  Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseInput:
      Name: prj_sap_016_analytics_db

ProcessedCrawler:
  Type: AWS::Glue::Crawler
  Properties:
    Name: prj-sap-016-processed-crawler
    Role: !GetAtt GlueRole.Arn
    DatabaseName: !Ref AnalyticsDatabase
    Targets:
```

```
        S3Targets:
          - Path: !Sub 's3://${DataLakeBucket}/processed/'

Outputs:
  BucketName:
    Value: !Ref DataLakeBucket
  FirehoseStream:
    Value: !Ref RawDataStream
```

---

4. Step-by-Step Deployment Guide

Step 4.1: Deploy Infrastructure

You can deploy the base infrastructure using either the Terraform or CloudFormation templates provided above. This will create the S3 bucket, IAM roles, Kinesis Firehose stream, Glue Database, and Glue Crawler.

Step 4.2: Create the Glue ETL Job

1. Go to the **AWS Glue Console -> ETL jobs -> Add job**.
2. **Name:** `json-to-parquet-transformer`
3. **IAM Role:** Select the `prj-sap-016-glue-role` created by IaC.
4. **Type:** Spark
5. **Glue version:** Glue 3.0 or later.
6. **Script:** Replace the boilerplate script with the following Python code:

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ["JOB_NAME"])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Replace <YOUR_DATA_LAKE_BUCKET> with the actual bucket name from IaC
outputs
bucket_name = "<YOUR_DATA_LAKE_BUCKET>"

# Read from the Raw S3 bucket
datasource0 = glueContext.create_dynamic_frame.from_options(
    "s3",
    {"paths": [f"s3://{bucket_name}/raw/"], "recurse": True},
    "json"
)

# Convert to Parquet and write to the Processed S3 bucket
datasink2 = glueContext.write_dynamic_frame.from_options(
    frame=datasource0,
    connection_type="s3",
    connection_options={"path": f"s3://{bucket_name}/processed/"},
    format="parquet"
)

job.commit()
```

7. Save the job.

Step 4.3: Test the Pipeline

1. **Send Data to Kinesis:** Use the AWS CLI to send a sample JSON record to your Firehose stream.

```
aws firehose put-record --delivery-stream-name prj-sap-016-raw-stream
--record "{\"Data\":\"
{\\\"sensorId\\\":\\\"A123\\\",\\\"temperature\\\":25.5}\"}"
```

2. **Verify Raw Data:** After about 60 seconds, check your `raw/` S3 folder. You should see a new file containing the JSON record.
3. **Run the Glue ETL Job:** Go to the Glue console and manually run your `json-to-parquet-transformer` job.
4. **Verify Processed Data:** Once the job succeeds, check your `processed/` S3 folder for the new Parquet file.
5. **Run the Glue Crawler:** Go to the Glue console and run `prj-sap-016-processed-crawler`.
6. **Verify Catalog:** After the crawler finishes, check **AWS Glue -> Tables**. You should see a new table in `prj_sap_016_analytics_db`.

Step 4.4: Query with Athena and Visualize with QuickSight

1. **Query with Athena:**
    - Go to the **Amazon Athena Console**.
    - **Settings:** Set a query result location in S3 (e.g., `s3://<YOUR_DATA_LAKE_BUCKET>/athena-results/`).
    - Select `prj_sap_016_analytics_db` as the database.
    - Run a query: `SELECT * FROM "processed" LIMIT 10;`
2. **Visualize with QuickSight:**
    - Go to the **Amazon QuickSight Console** (requires Enterprise edition).
    - **Manage QuickSight -> Security & permissions:** Ensure QuickSight has access to Athena and your S3 data lake bucket.
    - **Datasets -> New dataset -> Athena.**
    - **Data source name:** `analytics_data`
    - Select the database and the `processed` table.

- Click **Edit/Preview data** and then **Save & visualize** to build your dashboard.

---

5. Cleanup

To avoid ongoing charges:

1. **Delete the QuickSight dashboard and dataset.**
2. **Empty the S3 data lake bucket** (Terraform/CloudFormation will fail to delete it if it's not empty).
3. **Delete the Glue ETL job.**
4. **Destroy the IaC stack:**
   - Terraform: `terraform destroy`
   - CloudFormation: Delete the stack via the console or CLI.