**Streaming Data into BigQuery using Cloud Run**

A solution to ingest real-time streaming data into BigQuery using serverless computing

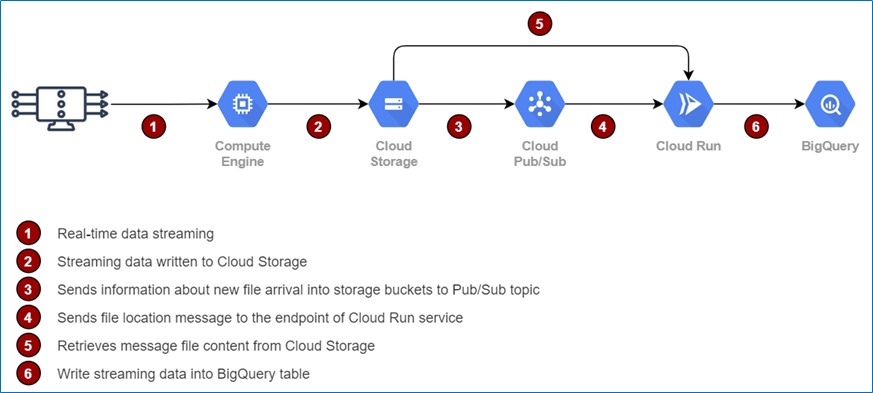
Solution

This solution is designed to ingest streaming data into BigQuery using Google cloud serverless platform without worrying about the underlying infrastructure. It also enables synchronous data read/write capabilities and contributes linear capacity planning & data accuracy. An additional hop (compute engine) used in this solution to syndicate multiple messages into a single file before it written to cloud storage to enhance storage efficacy.

Pub/Sub notifications sends information about new message/file arrival into storage buckets to Pub/Sub topic. As soon as pub/sub receives a notification, it pushes messages to the endpoint of Cloud Run service.

Cloud Run process the messages received from pub/sub, retrieve file contents from cloud storage and writes to BigQuery table.

**Architecture**



Data Flow Diagram for Streaming Data Ingestion

**Step-1: Create BigQuery Dataset and Table**

This solution allows flexible schema definition without source code change, but it must adhere to steaming data attributes. In the below example, only three columns represented for data ingestion process and as per requirement it can be extended.

*# Create Dataset*

*$ bq mk --dataset <project-name>:employee*

*# Create Table*

*$ bq mk -t employee.txn\_msg year:STRING,month:STRING,sales:FLOAT*

BigQuery Dataset and Table

**Step-2: Create Service Account and Add Roles**

This solution requires a service account with BigQuery admin, Pub/Sub editor, Cloud Storage admin and Cloud Run admin roles.

*# Create Service Account*

*$ gcloud iam service-accounts create svc-cloudrun-api --display-name "Cloud Run Streaming API"*

*# Add Roles to Service Account*

*$ gcloud projects add-iam-policy-binding data-lab \*

*--member serviceAccount:svc-cloudrun-api@data-lab.iam.gserviceaccount.com \*

*--role roles/bigquery.admin*

*$ gcloud projects add-iam-policy-binding data-lab \*

*--member serviceAccount:svc-cloudrun-api@data-lab.iam.gserviceaccount.com \*

*--role roles/pubsub.editor*

*$ gcloud projects add-iam-policy-binding data-lab \*

*--member serviceAccount:svc-cloudrun-api@data-lab.iam.gserviceaccount.com \*

*--role roles/storage.admin*

*$ gcloud projects add-iam-policy-binding data-lab \*

*--member serviceAccount:svc-cloudrun-api@data-lab.iam.gserviceaccount.com \*

*--role roles/run.admin*

Step-3: Build & Deploy Cloud Run App

Cloud Run is fully managed compute platform used for deploying and scaling containerized applications quickly and securely. The application code depicted below illustrates multiple steps involving — packaging application code into a container image, upload the container image to Container Registry, and then deploy the container image to Cloud Run.

The source code is written in node.js language but it can be written in multiple languages like Go, Python, Java, Ruby.

|  |
| --- |
| require("dotenv").config(); |
| const express = require('express'); |
| const app = express(); |
| const pubSubMessageController = require('./controllers/pubSubMessageController'); |
|  |
| app.use(bodyParser.json()); |
| app.post('/', (req, res) => { |
| pubSubMessageController.index(req, res); |
| }); |
|  |
| const PORT = process.env.PORT || 8080; |
| app.listen(PORT, () => { |
| console.log(`App listening on port ${PORT}`); |
| }); |
|  |
| module.exports = app; |

Application Code that Handles Incoming Request

|  |
| --- |
| const axios = require("axios");  const { Storage } = require('@google-cloud/storage'); |
| const storage = new Storage(); |
| const { BigQuery } = require('@google-cloud/bigquery'); |
| const bigquery = new BigQuery(); |
| const path = require('path'); |
| const cwd = path.join(\_\_dirname, '..'); |
| const fs = require('fs') |
|  |
| async function readStorageBucket(pubsubMsg) { |
| const bucketName = pubsubMsg.bucket; |
| const blobName = pubsubMsg.name; |
| const destFileName = path.join(cwd, 'download.json'); |
| const options = { destination: destFileName }; |
|  |
| await storage.bucket(bucketName).file(blobName).download(options); |
| console.log(`gs://${bucketName}/${blobName} downloaded to ${destFileName}.`); |
|  |
| const jsonData = fs.readFileSync('download.json', 'utf-8'); |
| return jsonData; |
| } |
|  |
| async function insertRowsAsStream(messages) { |
| var rowsList = []; |
|  |
| try { |
| var rows = messages.split("\n"); |
| for (var i = 0; i < rows.length; i++) { |
| rowsList.push(JSON.parse(rows[i])); |
| } |
| } catch{ |
| rowsList = messages; |
| } |
|  |
| await bigquery.dataset("bq\_poc").table("txn\_msg").insert(rowsList); |
| const response = `Record Inserted = ${rowsList.length}`; |
| return response; |
| } |
|  |
| exports.index = async function (req, res) { |
| const pubSubMessage = req.body.message; |
|  |
| try { |
| const pubsubMsg = JSON.parse(Buffer.from(pubSubMessage.data, 'base64').toString().trim()); |
| const response = await readStorageBucket(pubsubMsg); |
| const result = await insertRowsAsStream(response); |
| res.status(200).send(result); |
| } |
| catch (err) { |
| console.error(err.message); |
| res.status(500).send(err.message); |
| } |
| }; |

Pub/Sub Message Controller

*# Build and Upload App to Container Registry*

*$ gcloud builds submit --tag gcr.io/data-lab/stream-analytics*

Containerizing App and Upload to Container Registry

*# Download App Image from Container Registry and Deploy*

*$ gcloud run deploy --image gcr.io/data-lab/stream-analytics \*

*--service-account svc-cloudrun-api@data-lab.iam.gserviceaccount.com \*

*--platform managed*

Deploy Cloud Run App

Step-4: Create Pub/Sub Topic and Subscription

Pub/Sub subscription with the service account is created to push messages to the endpoint of Cloud Run service. Each message contains file name and file location in Cloud Storage bucket.

*# Create PubSub Topic*

*$ gcloud pubsub topics create pubsub-topic*

*# Create PubSub Subscription with Service Account*

*$ gcloud pubsub subscriptions create pubsub-subscription \*

*--topic pubsub-topic \*

*--push-endpoint=https://stream-analytics-la7ldrpbna-ue.a.run.app/ \*

*--push-auth-service-account=svc-cloudrun-api@data-lab.iam.gserviceaccount.com*

Step-5: Create Pub/Sub Notifications for Cloud Storage

Pub/Sub Notifications for Cloud Storage created with OBJECT\_FINALIZE event type to ensure that the notifications will trigger only when a new object is successfully created in the bucket.

*$ gsutil notification create \*

*-f json \*

*-t projects/data-lab/topics/pubsub-topic \*

*-e OBJECT\_FINALIZE gs://data-ingestion*

Step-6: Push Messages into Cloud Storage

Messages can be pushed to Cloud Storage bucket by enabling streaming pipeline or it can also be achieved by writing JSON formatted message files directly into cloud storage bucket.