

AWS Data Engineering Project - Global Crypto Market Pulse

Pramod Pawar (<https://github.com/Pramod-Data-Eng>)

Goal: Build an end-to-end pipeline that ingests real-time crypto prices, processes them with Spark, and transforms them into an analytics layer.

Architecture Breakdown

Layer	Service(s)	Role
Ingestion	Lambda + Kinesis	Python script fetches price data from a public API (like CoinGecko) and streams it.
Storage	S3	The "Raw" (Bronze) zone for stream logs.
Processing	Glue (PySpark)	A Spark job cleans the JSON logs and saves them as Parquet (Silver layer).
Governance	Lake Formation	Manage permissions for the Glue Data Catalog.
Orchestration	Step Functions	A visual workflow to trigger pipeline, prepare gold layer in Athena.

Phase 1: Ingestion (Lambda → Kinesis → S3)

Create your S3 Data Lake

First, create an S3 bucket with a unique name (e.g., crypto-analytics-lake-2026). Inside this bucket, create two folders:

- raw/: This is your **bronze** layer (raw JSON data).
- processed/: This is your **silver** layer (cleaned Parquet data).

Set up Kinesis Data Firehose

1. Go to the **Kinesis Console** -> Select **Amazon Data Firehose** - new Formerly **Kinesis Data Firehose** → **Create Firehose Stream**.
2. **Source:** Direct PUT (this allows Lambda to send data directly).
3. **Destination:** Amazon S3.
4. **S3 Bucket:** Choose the bucket you just created and set the prefix to raw/.
5. **Buffer Hints:** Set to 1 MiB or 60 seconds (this ensures data shows up quickly for your testing).

Note: - During setup of Kinesis Data Firehose; AWS itself creates required Role. Sometime we receive an error saying that “role does not exist”. If we press the create button once again; it successfully creates firehose.

The Lambda "Producer" (Python)

This script will act as your data generator. It fetches the current price of Bitcoin and Ethereum and pushes it to Kinesis.

Code (`lambda_function.py`):

https://github.com/Pramod-Data-Eng/Real-Time-Crypto-Analytics/blob/main/lambda_function.py

Note: To run this in Lambda, you'll need to add a **Layer** for the requests library or bundle it in a ZIP. You can also use `urllib.request` (built-in) to avoid the extra step.

```
mkdir python
```

```
pip install requests -t python/
```

```
zip -r requests_layer.zip python
```

- Go to AWS Lambda → Layers → Create Layer.

- Upload `requests_layer.zip`.

- Attach the layer to your function.

- Keep your function zip containing only `lambda_function.py`.

Attach role that will have default lambda execution role as well as Firehose S3 Put Object Role.

Phase 2: Cataloguing & Spark Processing

The Glue Crawler (Schema Discovery)

Before Spark can process the data efficiently, AWS needs to "understand" its structure.

1. Go to **AWS Glue Console** -> **Data Catalog** → **Crawlers** -> **Create crawler**.
2. **Name:** crypto_raw_crawler.
3. **Data source:** Add your S3 path (`s3://crypto-analytics-lake-2026/raw/`).
4. **IAM Role:** Create a new role (Glue will suggest this). Ensure it has access to your S3 bucket.
5. **Output Database:** Create a new database called crypto_db.
6. **Run it:** Once created, click **Run crawler**.

- Wait 1-2 minutes. When it finishes, go to **Tables** in Glue. You should see a table (e.g., **raw**) with the schema inferred from your JSON.

#	Column name	Data type	Partition key	Comment
1	bitcoin	struct	-	-
2	ethereum	struct	-	-
3	timestamp	string	-	-
4	partition_0	string	Partition (0)	-
5	partition_1	string	Partition (1)	-
6	partition_2	string	Partition (2)	-
7	partition_3	string	Partition (3)	-

The Spark Transformation (Glue ETL Job)

Now we write the **PySpark** script to flatten the JSON and save it as Parquet in your processed/
(Silver) folder.

1. Go to **Glue ETL jobs** -> **Visual ETL** (or Script editor if you prefer coding directly).
2. Select **Python Shell** or **Spark script**. Let's use Spark for your GitHub portfolio.

The PySpark Logic (`glue_spark_transform.py`):

https://github.com/Pramod-Data-Eng/Real-Time-Crypto-Analytics/blob/main/glue_spark_transform.py

<https://github.com/Pramod-Data-Eng>

The screenshot shows the AWS Glue Studio interface. The top navigation bar includes tabs for 'Script', 'Job details', 'Runs' (which is selected), 'Data quality', 'Schedules', and 'Version Control'. Below the navigation is a table titled 'Job runs (1/3) Info' with columns for 'Run status', 'Retries', 'Start time (Local)', 'End time (Local)', 'Duration', 'Capacity (DPUs)', 'Worker type', and 'Glue version'. The table shows three entries: one 'Succeeded' run and two 'Failed' runs. The 'Run details' section below the table provides specific details for the successful run, including the job name 'glue_spark_transform', start time '02/15/2026 08:23:34', glue version '5.0', and worker type 'G.1X'. The bottom of the page includes links for 'CloudShell', 'Feedback', and 'Console Mobile App', along with standard AWS footer links for 'Privacy', 'Terms', and 'Cookie preferences'.

Note: - Before execution of AWS Glue Job; please ensure that the job is having appropriate S3 Put object permission in order to write output parquet file in S3 processed folder

Phase 3: The Athena Validation (Sanity Check)

Before moving to Redshift, we need to ensure the Glue Catalog correctly maps to your Parquet files.

1. Run a Glue Crawler on your processed/ folder.

This will create table processed in Glue

The screenshot shows the AWS Glue Console. The left sidebar has sections for 'AWS Glue', 'Data Catalog tables', 'Data Catalog', 'Tables', and 'Data Integration and ETL'. The main area is titled 'processed' and shows the 'Table overview' tab. It displays details such as Name ('processed'), Database ('crypto_db'), Location ('s3://crypto-analytics-lake-2026/processed/'), and Connection ('-'). The 'Schema' tab is selected, showing columns like 'Name', 'Type', and 'Comment'. The bottom of the page includes links for 'CloudShell', 'Feedback', and 'Console Mobile App', along with standard AWS footer links for 'Privacy', 'Terms', and 'Cookie preferences'.

2. Once finished, go to the **Athena Console**.

<https://github.com/Pramod-Data-Eng>

3. Select your crypto_db database.

4. Run this SQL:

SQL

```
select * from processed limit 10;
```

The screenshot shows the AWS Athena Query Editor interface. On the left, the sidebar displays the database 'crypto_db' and two tables: 'processed' and 'raw'. The 'processed' table is selected. The main area shows the SQL query 'select * from processed limit 10;'. Below the query, the 'Query results' tab is active, showing a table with two rows of data. The columns are timestamp, price_usd, volume_24h, symbol, and partition_0. The data is as follows:

#	timestamp	price_usd	volume_24h	symbol	partition_0
1	2026-02-14T14:27:28.614877	2069.69	2.0747344147481026E10	ETH	crypto_prices
2	2026-02-14T14:27:28.614877	69403.0	4.5936131877974556E10	BTC	crypto_prices

Phase 4: Orchestration with AWS Step Functions

We will create a State Machine that acts as the "conductor" for your entire pipeline. Instead of running things manually, this will handle the sequence and errors.

1. The Workflow Logic

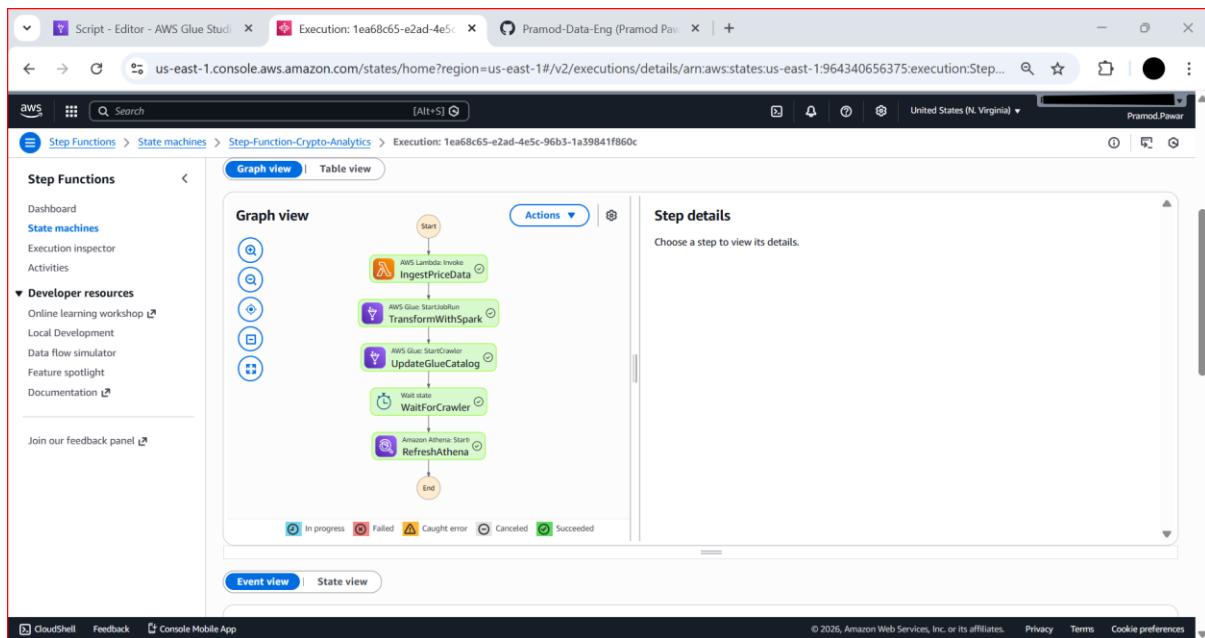
- Step 1: Ingest – Trigger the Lambda to fetch the latest prices.
- Step 2: Transform – Run the Glue Spark job once the data is in S3.
- Step 3: Catalog – Run the Crawler to update the schema.
- Step 4: Load – Use Athena as Gold layer

2. The Step Function Code (ASL)

Go to Step Functions > Create state machine > Blank > Code tab. Paste this (replace placeholders with your actual ARNs):

JSON

```
https://github.com/Pramod-Data-Eng/Real-Time-Crypto-Analytics/blob/main/Step-Function-Crypto-Analytics.json
```



Final Output

The screenshot shows the Amazon Athena Query editor. On the left, the 'Data' sidebar is open, showing 'Data source: AwsDataCatalog', 'Catalog: None', and 'Database: crypto_db'. Below it, 'Tables and views' are listed: 'hourly_summary', 'processed', and 'raw'. Under 'Tables', 'hourly_summary' is selected. The main area shows a query named 'Query 1' with the SQL command:

```
1 Select * from crypto_db.hourly_summary;
```

Below the query, the 'SQL' tab shows the execution details: 'Run again', 'Explain', 'Cancel', 'Clear', and 'Create'. The 'Query results' tab shows the results of the query, which has completed successfully. The results table has two columns: '# symbol' and 'avg_price'. The results section includes 'Copy' and 'Download results CSV' buttons. The footer of the page includes links for 'CloudShell', 'Feedback', and 'Console Mobile App'.