AWS Glue vs Spark on EMR for 2024

Your Name

2025-01-22

Table of contents


```
import pandas as pd
import plotly.express as px

# 1) Load ETL notifications from Parquet
df = pd.read_parquet("scripts/etl_history_2024.parquet")

# 2) Convert timestamp (milliseconds) to datetime
df["datetime"] = pd.to_datetime(df["timestamp"], unit="ms")
df["date"] = df["datetime"].dt.date

# 3) Hypothetical cost model
#    No runtime data here, so we'll assume cost is per-row:
#      cost_glue = rows * $0.00002
#      cost_emr  = rows * $0.000015
#    Adjust these multipliers as needed.
df["cost_glue"] = df["rows"] * 0.00002
df["cost_emr"]  = df["rows"] * 0.000015

# 4) Aggregate by date
daily_costs = (
    df.groupby("date", as_index=False)
      .agg({
          "cost_glue": "sum",
          "cost_emr": "sum",
          "rows": "sum"
      })
```

```
)

# 5) Compare daily costs (Glue vs EMR) in a line chart
fig_costs = px.line(
    daily_costs,
    x="date",
    y=["cost_glue", "cost_emr"],
    labels={"value": "Daily Cost (USD)", "date": "Date"},
    title="Daily Hypothetical Costs: AWS Glue vs EMR (2024)"
)
fig_costs.update_layout(hovermode="x unified")

# Display the Plotly figure
fig_costs.show()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

```
# 6) Visualize daily rows processed in a bar chart
fig_rows = px.bar(
    daily_costs,
    x="date",
    y="rows",
    labels={"rows": "Rows Processed", "date": "Date"},
    title="Total Rows Processed per Day"
)
fig_rows.show()
```

```
Unable to display output for mime type(s): text/html
```

0.0.1 Customizing the Document

1. **Cost Model**
    - Adjust your cost multipliers based on real usage (e.g., DPUs for Glue, instance-hour costs for EMR, cluster overhead, etc.).

- If you have actual **runtimes** or **DPUs** used, incorporate those into your calculation instead of a simple "per-row" approach.

2. **Chart Types**
   - Plotly supports many chart types (scatter, area, box, etc.). Choose what best displays your data.
   - You can also add interactive tooltips, facet the data by `destination` or `table`, etc.
3. **Grouping & Granularity**
   - If you have thousands of daily data points, consider grouping by **week** or **month** to simplify the charts:

     ```
     df["year_week"] = df["datetime"].dt.strftime('%Y-%U')
     ```

   - Then group by `year_week` instead of `date`.
4. **Styling and Themes**
   - Quarto + Plotly uses the default Plotly theme by default. You can customize layout, colors, etc. using Plotly's theming options.

With this Quarto document, you'll have an end-to-end workflow showing how much you *would* have spent each day on AWS Glue vs. EMR, plus a breakdown of how many rows were processed overall. Adjust the details to match your actual cost structures and data fields.