Pipeline E2E AWS per analizzare BTC e XMR

Introduzione

CryptoData Insights è un'azienda innovativa specializzata nell'analisi dei dati sulle criptovalute. Questo progetto implementa una pipeline End-to-End (E2E) su AWS per analizzare Bitcoin (BTC) e Monero (XMR). La pipeline automatizza il processo di acquisizione, pulizia e trasformazione dei dati, rendendoli pronti per l'esplorazione e la visualizzazione.

Architettura della Soluzione

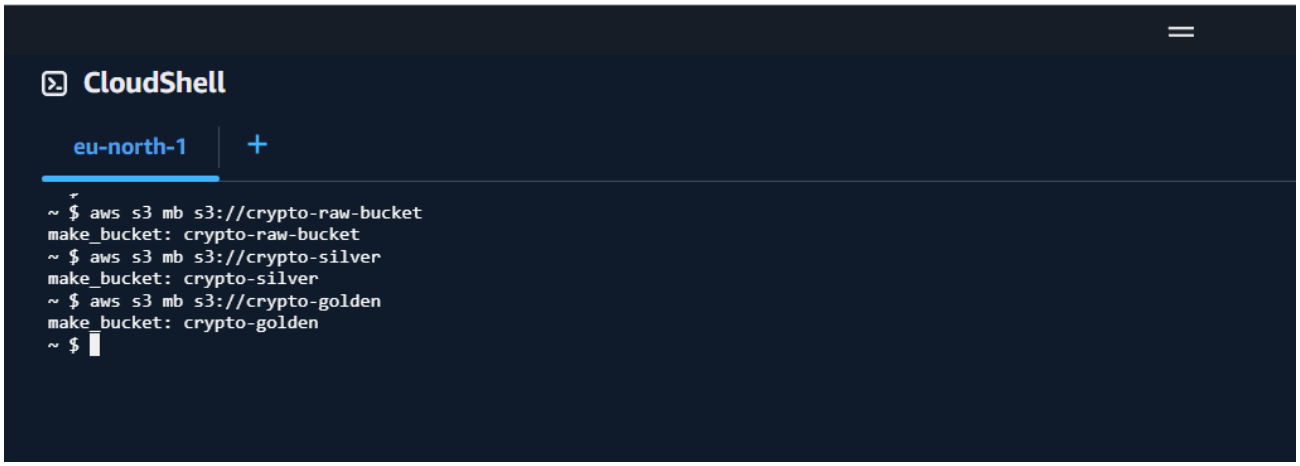
La pipeline si articola in diverse fasi:

1. Caricamento dati grezzi su S3 .
2. Pulizia e trasformazione dei dati da Raw a Silver.
3. Trasformazione dei dati da Silver a Golden.
4. Creazione dei Glue Jobs
5. Creazione gruppo di lavoro Redshift
6. Caricamento su Redshift da Golden.
7. Orchestrazione con Step Functions per coordinare tutte le fasi.

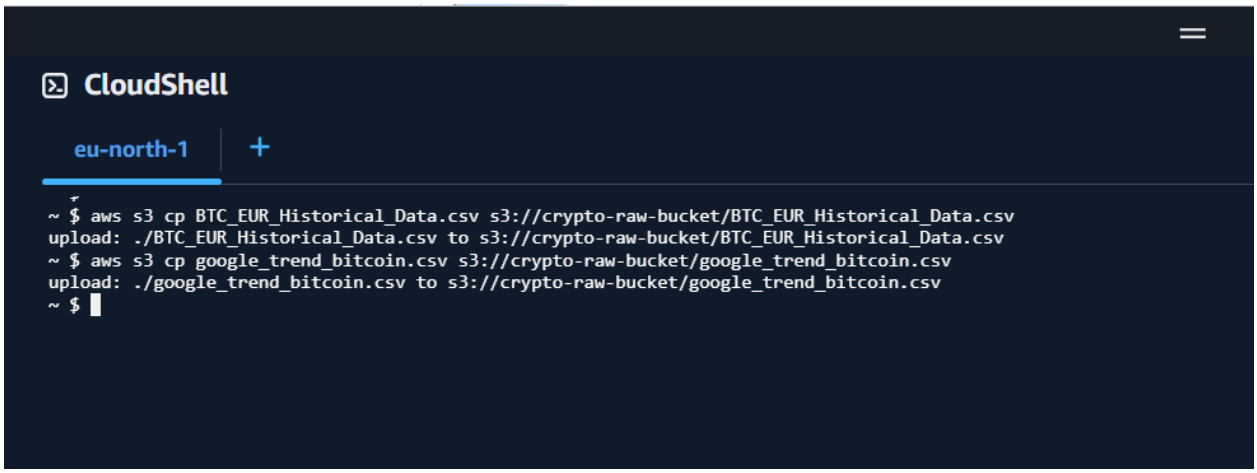
Di seguito sono descritti tutti i passaggi implementati.

Passaggi Implementati

Creazione dei Bucket S3



Caricamento dei File CSV su S3



Creazione degli Script per Trasformazioni ETL

---

raw-silver-btc.py

```
import sys
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.dynamicframe import DynamicFrame
from pyspark.sql import functions as F
from pyspark.sql.functions import col, to_date, regexp_replace, round
from pyspark.sql.window import Window

# Parametri
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

# --- Path configurati ---
RAW_BUCKET = "s3://crypto-raw-bucket"
SILVER_BUCKET = "s3://crypto-silver"

BTC_RAW_PATH = f"{RAW_BUCKET}/BTC_EUR_Historical_Data.csv"
GT_BTC_RAW_PATH = f"{RAW_BUCKET}/google_trend_bitcoin.csv"

BTC_SILVER_PATH = f"{SILVER_BUCKET}/btc"
GT_BTC_SILVER_PATH = f"{SILVER_BUCKET}/gt_bitcoin"

# --- Funzioni ETL ---
```

```
def load_csv(path: str):
    dynf = glueContext.create_dynamic_frame.from_options(
        connection_type="s3",
        connection_options={"paths": [path]},
        format="csv",
        format_options={"withHeader": True}
    )
    return dynf.toDF()

def clean_price_df(df, date_col="Date", price_col="Price"):
    df = df.withColumn(price_col, regexp_replace(col(price_col), ",", "").cast("float"))
    df = df.withColumn(date_col, to_date(col(date_col), "MM/dd/yyyy"))

    window = Window.orderBy(date_col)
    df = df.withColumn(
        price_col,
        F.when((col(price_col).isNull()) | (col(price_col) == -1),
            (F.lag(price_col).over(window) + F.lead(price_col).over(window)) / 2)
        .otherwise(col(price_col))
    )

    df = df.withColumn(price_col, round(col(price_col), 3))
    return df

def clean_trend_df(df, date_col="Settimana", value_col="interesse bitcoin"):
    return df.withColumn(date_col, col(date_col).cast("date")) \
            .withColumn(value_col, col(value_col).cast("float"))

def save_parquet(df, output_path):
    dynf = DynamicFrame.fromDF(df, glueContext, "parquet_out")
    glueContext.write_dynamic_frame.from_options(
        frame=dynf,
        connection_type="s3",
        connection_options={"path": output_path},
        format="parquet"
    )

# --- Esecuzione ETL ---
btc_raw_df = load_csv(BTC_RAW_PATH)
gt_btc_raw_df = load_csv(GT_BTC_RAW_PATH)

btc_clean_df = clean_price_df(btc_raw_df)
gt_btc_clean_df = clean_trend_df(gt_btc_raw_df)

save_parquet(btc_clean_df, BTC_SILVER_PATH)
save_parquet(gt_btc_clean_df, GT_BTC_SILVER_PATH)
```

---

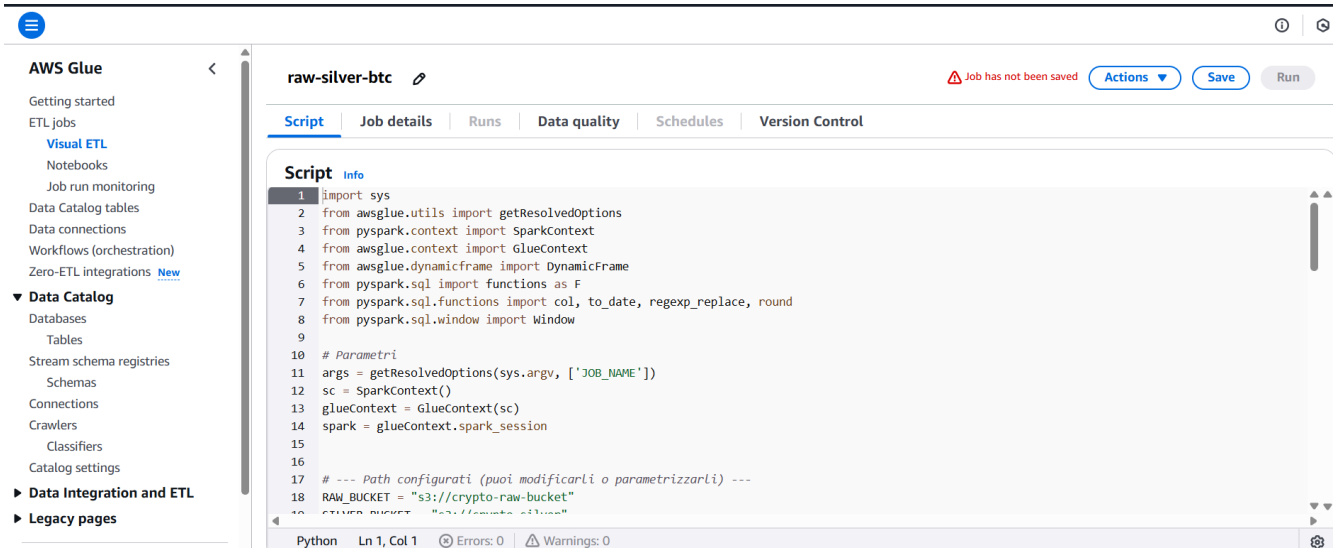
silver-gold-btc.py

```
import sys
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.dynamicframe import DynamicFrame
from pyspark.sql.functions import col, avg
from pyspark.sql.window import Window

# Parametri
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

# --- Percorsi Silver e Gold ---
SILVER_BUCKET = "s3://crypto-silver"
GOLD_BUCKET = "s3://crypto-golden"

BTC_SILVER_PATH = f"{SILVER_BUCKET}/btc"
GT_BTC_SILVER_PATH = f"{SILVER_BUCKET}/gt_bitcoin"
BTC_GOLD_PATH = f"{GOLD_BUCKET}/btc_with_trend"

# --- Funzioni di supporto ---
def load_parquet(path):
    dynf = glueContext.create_dynamic_frame.from_options(
        connection_type="s3",
        connection_options={"paths": [path]},
        format="parquet"
    )
    return dynf.toDF()

def save_to_gold(df, path):
    dynf = DynamicFrame.fromDF(df, glueContext, "gold_output")
    glueContext.write_dynamic_frame.from_options(
        frame=dynf,
        connection_type="s3",
        connection_options={"path": path},
        format="parquet"
    )

# --- ETL: Caricamento dati Silver ---
btc_df = load_parquet(BTC_SILVER_PATH)
gt_df = load_parquet(GT_BTC_SILVER_PATH)

# --- Media mobile a 10 giorni sul prezzo BTC ---
window_spec = Window.orderBy("Date").rowsBetween(-9, 0)  # 10 giorni (incluso il corrente)
btc_df = btc_df.withColumn("Price", avg("Price").over(window_spec))
```

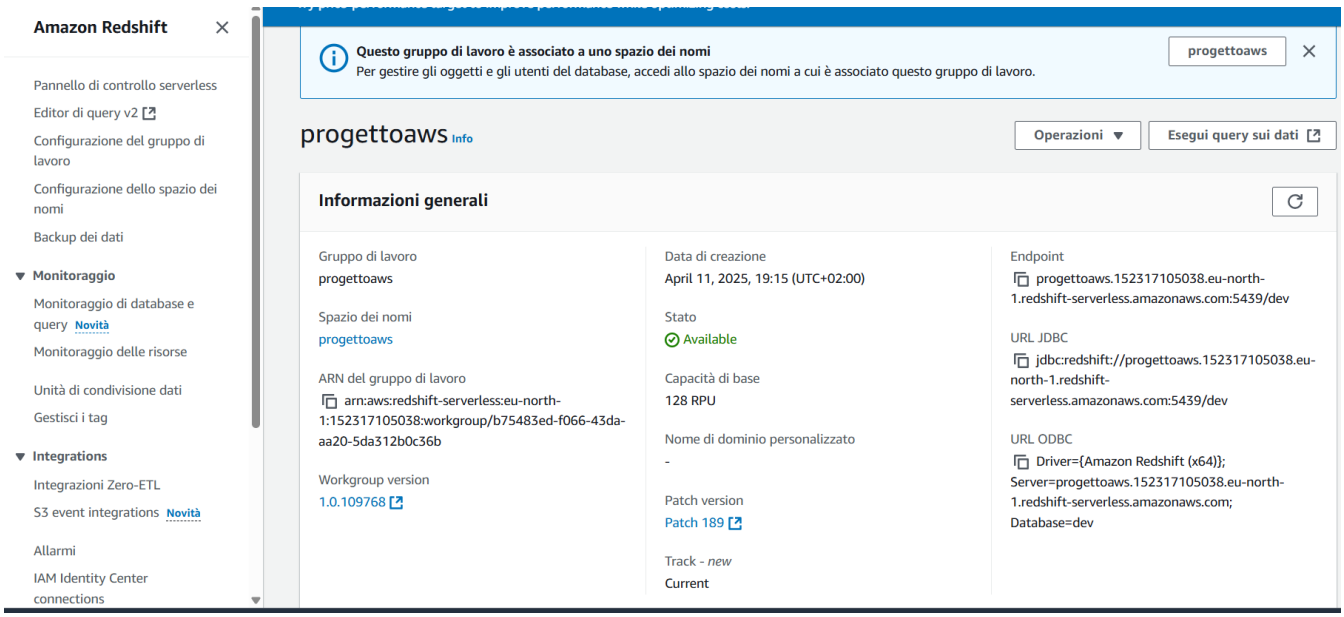
```
# --- Join tra BTC e Google Trends ---
joined_df = btc_df.join(gt_df, btc_df.Date == gt_df.Settimana, "inner") \
            .select(btc_df.Date, btc_df.Price, gt_df["interesse bitcoin"].alias("GoogleTrend"))

# --- Salvataggio su Bucket Gold ---
save_to_gold(joined_df, BTC_GOLD_PATH)
```

Gli script per Monero (**raw-silver-xmr.py** e **silver-gold-xmr.py**) sono analoghi.

Creazione dei Glue Jobs



Creazione del Gruppo di Lavoro su Redshift

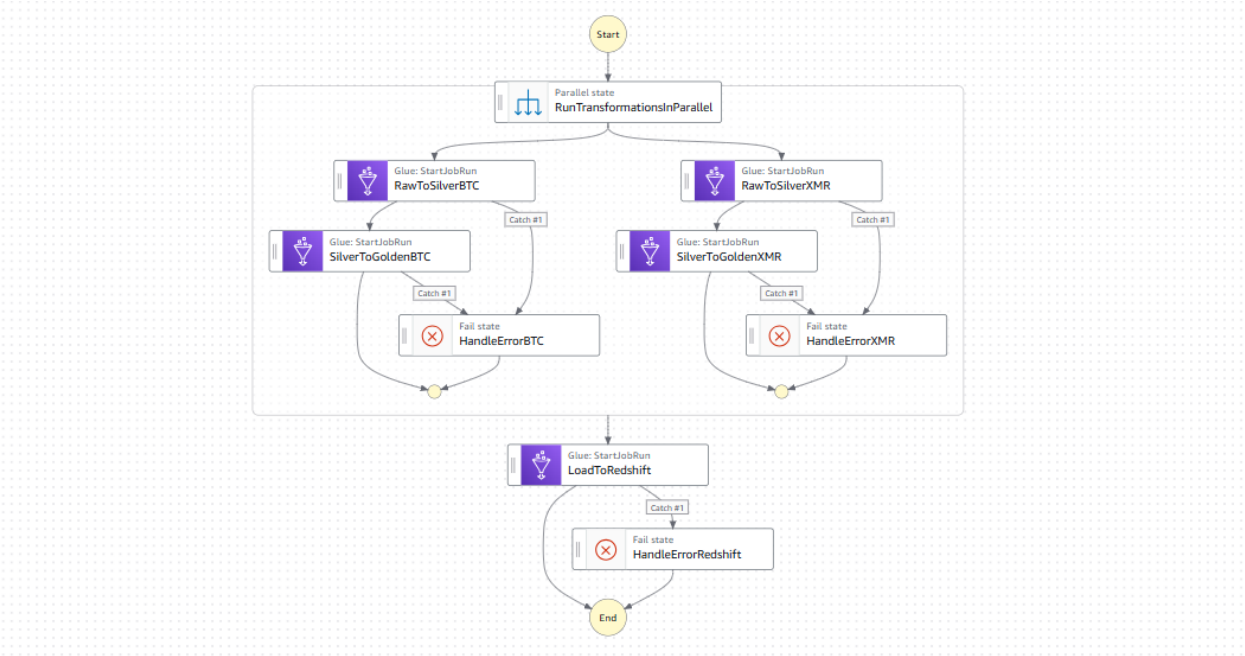


Script per il Caricamento dei Dati su Redshift

```
import psycopg2
import pandas as pd
from pyspark.sql import SparkSession

# Parametri di connessione
JDBC_URL = "jdbc:redshift://progettoaws.152317105038.eu-north-1.redshift-
serverless.amazonaws.com:5439/dev"
USER = "admin"
PASSWORD = "JIMFDlnbt792"
PORT = "5439"
DATABASE = "dev"

# Funzione per connettersi a Redshift e caricare i dati
def connect_to_redshift():
    conn = psycopg2.connect(
        dbname=DATABASE,
        user=USER,
        password=PASSWORD,
        host="progettoaws.152317105038.eu-north-1.redshift-serverless.amazonaws.com",
        port=PORT
    )
    return conn

# Caricamento dei dati da S3 (per Bitcoin e Monero)
def load_data_from_s3_to_redshift():
    # Inizializza una sessione Spark per caricare i dati da Parquet su S3
    spark = SparkSession.builder.appName("S3-to-Redshift").getOrCreate()

    # Carica i dati da S3 per Bitcoin e Monero (nel bucket crypto-golden)
    btc_df = spark.read.parquet("s3://crypto-golden/btc_with_trend")
    xmr_df = spark.read.parquet("s3://crypto-golden/xmr_with_trend")

    # Converte i DataFrame Spark in Pandas DataFrame per inviarli a Redshift
    btc_pd = btc_df.toPandas()
    xmr_pd = xmr_df.toPandas()

    # Connessione a Redshift
    conn = connect_to_redshift()
    cursor = conn.cursor()

    # Creazione delle tabelle (se non esistono già)
    create_btc_table_query = """
    CREATE TABLE IF NOT EXISTS btc_with_trend (
        Date DATE,
        Price FLOAT,
        GoogleTrend FLOAT
```

```
    );
    """
    cursor.execute(create_btc_table_query)

    create_xmr_table_query = """
    CREATE TABLE IF NOT EXISTS xmr_with_trend (
        Date DATE,
        Price FLOAT,
        GoogleTrend FLOAT
    );
    """
    cursor.execute(create_xmr_table_query)

    conn.commit()

    # Inserimento dei dati da Pandas DataFrame a Redshift per Bitcoin
    for index, row in btc_pd.iterrows():
        insert_btc_query = """
        INSERT INTO btc_with_trend (Date, Price, GoogleTrend)
        VALUES (%s, %s, %s)
        """
        cursor.execute(insert_btc_query, (row['Date'], row['Price'], row['GoogleTrend']))

    # Inserimento dei dati da Pandas DataFrame a Redshift per Monero
    for index, row in xmr_pd.iterrows():
        insert_xmr_query = """
        INSERT INTO xmr_with_trend (Date, Price, GoogleTrend)
        VALUES (%s, %s, %s)
        """
        cursor.execute(insert_xmr_query, (row['Date'], row['Price'], row['GoogleTrend']))

    conn.commit()

    # Chiude la connessione
    cursor.close()
    conn.close()

# Esegui il caricamento
load_data_from_s3_to_redshift()
```

Definizione delle Step Functions




```
{
  "Comment": "Pipeline E2E per BTC e XMR",
  "StartAt": "RunTransformationsInParallel",
  "States": {
   "RunTransformationsInParallel": {
    "Type": "Parallel",
    "Branches": [
     {
      "StartAt": "RawToSilverBTC",
      "States": {
       "RawToSilverBTC": {
        "Type": "Task",
        "Resource": "arn:aws:states:::glue:startJobRun.sync",
        "Parameters": {
         "JobName": "raw-silver-btc"
        },
        "Next": "SilverToGoldenBTC",
        "Catch": [
         {
          "ErrorEquals": [
           "States.ALL"
          ],
          "Next": "HandleErrorBTC"
         }
        ]
       },
       "SilverToGoldenBTC": {
        "Type": "Task",
        "Resource": "arn:aws:states:::glue:startJobRun.sync",
```

```
      "Parameters": {
        "JobName": "silver-gold-btc"
      },
      "End": true,
      "Catch": [
        {
          "ErrorEquals": [
            "States.ALL"
          ],
          "Next": "HandleErrorBTC"
        }
      ]
    },
    "HandleErrorBTC": {
      "Type": "Fail",
      "Error": "BTCPipelineFailed",
      "Cause": "Errore nella pipeline BTC"
    }
  }
},
{
  "StartAt": "RawToSilverXMR",
  "States": {
    "RawToSilverXMR": {
      "Type": "Task",
      "Resource": "arn:aws:states:::glue:startJobRun.sync",
      "Parameters": {
        "JobName": "raw-silver-xmr"
      },
      "Next": "SilverToGoldenXMR",
      "Catch": [
        {
          "ErrorEquals": [
            "States.ALL"
          ],
          "Next": "HandleErrorXMR"
        }
      ]
    },
    "SilverToGoldenXMR": {
      "Type": "Task",
      "Resource": "arn:aws:states:::glue:startJobRun.sync",
      "Parameters": {
        "JobName": "silver-gold-xmr"
      },
      "End": true,
      "Catch": [
        {
```

```
            "ErrorEquals": [
              "States.ALL"
            ],
            "Next": "HandleErrorXMR"
          }
        ]
      },
      "HandleErrorXMR": {
        "Type": "Fail",
        "Error": "XMRPipelineFailed",
        "Cause": "Errore nella pipeline XMR"
      }
    }
  ],
  "Next": "LoadToRedshift"
},
"LoadToRedshift": {
  "Type": "Task",
  "Resource": "arn:aws:states:::glue:startJobRun.sync",
  "Parameters": {
    "JobName": "load_redshift"
  },
  "End": true,
  "Catch": [
    {
      "ErrorEquals": [
        "States.ALL"
      ],
      "Next": "HandleErrorRedshift"
    }
  ]
},
"HandleErrorRedshift": {
  "Type": "Fail",
  "Error": "RedshiftLoadFailed",
  "Cause": "Errore durante il caricamento in Redshift"
}
}
}
```

Risultati Attesi

Al termine della pipeline:

- I dati grezzi vengono puliti e trasformati.
- I dati finali (prezzo e trend unificati) vengono caricati su Redshift.
- Le tabelle **btc_with_trend** e **xmr_with_trend** sono pronte per l'analisi SQL o la visualizzazione con QuickSight.

Author: Michele Colangelo