AWS Orders Pipeline – COMPLETE STEP■BY■STEP (Final Working Path)

Region: us-east-1 (N. Virginia) • Budget mode: minimise cost • IAM: LabRole / LabInstanceProfile only

This guide captures ONLY what finally worked for us, in the exact order we executed it. It includes every tiny click, name, command, security group rule, and all the fixes we applied.

Step 0 — Names & Values We Actually Used

```
Bucket: omsai-traya-bucket
VPC: orders-pipeline-vpc-vpc   (created via wizard with 2 public + 2 private subnets)
Security Groups:
  • lambda-rds-sg        (used by Lambda and RDS)
  • ec2-analytics-sg     (used by EC2)
EC2: orders-analytics-instance  (t3.micro, Amazon Linux 2023)
Elastic IP (temporary): 54.146.201.20  (allocated during work, later released)
RDS MySQL endpoint: ordersdb.c5bgvvjgyoqm.us-east-1.rds.amazonaws.com
RDS DB name (schema): ordersdb
RDS user: admin
Lambda function: orders_pipeline_fn (Python 3.12)
Lambda layer: AWSSDKPandas-Python312 (AWS provided)
SNS: Topic created; VPC Endpoint used: com.amazonaws.us-east-1.sns (Interface, private subnets, SG=lambda-rd
S3 object keys used: raw/orders_detailed.csv → processed/cleaned_orders_detailed.csv → analytics/summary_YY
```

Step 1 — Create S3 Bucket & Folders

```
Console → S3 → Create bucket
  • Name: omsai-traya-bucket
  • Region: us-east-1
  • Block Public Access: keep all ON
  • Encryption: SSE-S3 (AES-256)
Create, then inside bucket create folders:
  raw/
  processed/
  analytics/
Lifecycle (cost saver):
  Bucket → Management → Lifecycle rules → Create rule
  • Name: move-15d-to-glacier
  • Scope: entire bucket
  • Transition: Current versions → Glacier Flexible Retrieval after 15 days
```

Step 2 — Create VPC (Wizard)

```
Console → VPC → Create VPC
  • Resources to create: VPC and more (wizard)
  • Name: orders-pipeline-vpc-vpc
  • IPv4 CIDR: 10.0.0.0/16
  • Number of AZs: 2
  • Public subnets: 2
  • Private subnets: 2
  • NAT gateway: None
  • VPC endpoints: None (we will add SNS endpoint later)
Result: VPC with 2 public + 2 private subnets, route tables correct (public via IGW, private without IGW).
```

Step 3 — Security Groups (Final Rules That Worked)

```
Create SG: lambda-rds-sg (VPC = orders-pipeline-vpc-vpc)
  Inbound rules:
    • MySQL/Aurora (3306)  Source = lambda-rds-sg     # allow Lambda↔RDS within same SG
    • MySQL/Aurora (3306)  Source = ec2-analytics-sg  # allow EC2 analytics to RDS
  Outbound: allow all (default)

Create SG: ec2-analytics-sg (VPC = orders-pipeline-vpc-vpc)
  Inbound rules:
    • SSH (22)  Source = 0.0.0.0/0   # for lab convenience
  Outbound: allow all (default)
```

Step 4 — Create RDS MySQL (Private)

```
Console → RDS → Create database (Standard create)
  • Engine: MySQL 8.x
  • Templates: Free tier
  • DB instance class: db.t3.micro
  • DB instance identifier: ordersdb
  • Master username: admin
  • Master password: <keep private>
Connectivity:
  • VPC: orders-pipeline-vpc-vpc
  • Public access: NO
  • DB Subnet group: two private subnets (or create one if asked)
  • VPC security group(s): lambda-rds-sg
Additional cost-saving: disable Monitoring / uncheck auto minor upgrade if allowed.
Create and wait until “Available”.
```

Step 5 — Launch EC2 + (optional) Elastic IP

```
Console → EC2 → Launch instance
  • Name: orders-analytics-instance
  • AMI: Amazon Linux 2023
  • Type: t3.micro (free tier)
  • Key pair: your_lab_key.pem
  • Network: VPC orders-pipeline-vpc-vpc
  • Subnet: PUBLIC subnet
  • Auto-assign public IP: ENABLED
  • Security group: ec2-analytics-sg
Launch.
(Optional) Allocate Elastic IP for stable public IP:
  EC2 → Elastic IPs → Allocate → Associate to orders-analytics-instance
  We used EIP: 54.146.201.20 (later released to avoid charge)
MobaXterm SSH:
  Host: <EC2 Public IP or EIP>
  User: ec2-user
  Key: your .pem
```

Step 6 — Create Database & Table from EC2 (MySQL client)

```
On EC2 shell:
sudo yum install mariadb105 -y

Connect to RDS:
mysql -h ordersdb.c5bgvvjgyoqm.us-east-1.rds.amazonaws.com -u admin -p

Create DB & table:
CREATE DATABASE ordersdb;
USE ordersdb;
CREATE TABLE orders_cleaned (
  order_id INT,
  customer_name VARCHAR(100),
  city VARCHAR(50),
  product VARCHAR(100),
  quantity INT,
  price FLOAT,
  discount_code VARCHAR(20),
  channel VARCHAR(50),
  payment_mode VARCHAR(50),
  order_date DATE,
  total_value FLOAT
);
EXIT;
```

Step 7 — Lambda Function (Code + Packaging + Env)

```
Lambda: orders_pipeline_fn (Python 3.12), Role = LabRole
Layers:
  • Add AWS provided layer: AWSSDKPandas-Python312
PyMySQL packaging using CloudShell (so no custom layer issues):
CloudShell:
  mkdir orders_lambda && cd orders_lambda
  pip install PyMySQL -t .
  cat > lambda_function.py << 'EOF'
  # (we replaced this later with final code from this document)
  print("placeholder")
  def lambda_handler(event, context): return "OK"
  EOF
  zip -r9 ../orders_pipeline.zip .
Upload orders_pipeline.zip to Lambda (Code → Upload from → .zip).

Environment variables (Configuration → Environment variables):
  DB_HOST=ordersdb.c5bgvvjgyoqm.us-east-1.rds.amazonaws.com
  DB_USER=admin
  DB_PASSWORD=<your password>
  DB_NAME=ordersdb
  BUCKET=omsai-traya-bucket
  SNS_TOPIC_ARN=<VPC Endpoint ARN for SNS>
  SALES_ALERT_THRESHOLD=50000
```

Step 8 — Lambda VPC Configuration

```
Lambda → Configuration → VPC → Edit
  • VPC: orders-pipeline-vpc-vpc
  • Subnets: select both PRIVATE subnets
  • Security group: lambda-rds-sg
Save.
This enables private connectivity from Lambda to RDS.
```

Step 9 — S3 Trigger for Lambda

```
Lambda → Configuration → Triggers → Add trigger → S3
  • Bucket: omsai-traya-bucket
  • Event type: PUT (All object create events)
  • Prefix filter: raw/
Add.
Test by uploading orders_detailed.csv to raw/.
```

Step 10 — SNS Topic + VPC Endpoint (Interface) + Manager■style Message

```
Create SNS topic → add Email subscription → confirm by email.
Because Lambda is in private subnets and NAT is avoided, create Interface VPC Endpoint for SNS:
VPC → Endpoints → Create
  • Service: com.amazonaws.us-east-1.sns
  • Type: Interface
  • Subnets: both PRIVATE subnets
  • Security group: lambda-rds-sg
  • Private DNS enabled: Yes
Copy the VPC Endpoint ARN (looks like: arn:aws:ec2:us-east-1:<acct>:vpc-endpoint/vpce-xxxxxxxx)
Set Lambda env var SNS_TOPIC_ARN = that endpoint ARN (lab restriction workaround).

We also changed the email content to manager■friendly:
Subject: Daily Sales Summary
Message example: “Today's processed file raw/orders_detailed.csv generated total sales = 171,411.14 INR. Tar
```

Step 11 — Final Lambda Code (we pasted in editor and Deployed)

```
import os
import json
import math
import boto3
import pymysql
import pandas as pd
from io import StringIO
from datetime import datetime, timezone

DB_HOST = os.environ.get("DB_HOST")
DB_USER = os.environ.get("DB_USER", "admin")
DB_PASSWORD = os.environ.get("DB_PASSWORD")
DB_NAME = os.environ.get("DB_NAME", "ordersdb")
BUCKET = os.environ.get("BUCKET")
SNS_TOPIC_ARN = os.environ.get("SNS_TOPIC_ARN", "")
```

```python
SALES_ALERT_THRESHOLD = float(os.environ.get("SALES_ALERT_THRESHOLD", "50000"))

s3 = boto3.client("s3")
sns = boto3.client("sns")

EXPECTED_COLUMNS = [
    "order_id","customer_name","city","product","quantity","price",
    "discount_code","channel","payment_mode","order_date",
]

def _today_yyyymmdd():
    return datetime.now(timezone.utc).strftime("%Y%m%d")

def _safe_title(x):
    if pd.isna(x):
        return x
    try:
        return str(x).strip().title()
    except Exception:
        return x

def _coerce_numeric(series, default=0):
    return pd.to_numeric(series, errors="coerce").fillna(default)

def _read_csv_from_s3(bucket, key) -> pd.DataFrame:
    resp = s3.get_object(Bucket=bucket, Key=key)
    df = pd.read_csv(resp["Body"])
    return df

def _write_csv_to_s3(df: pd.DataFrame, bucket: str, key: str):
    buf = StringIO()
    df.to_csv(buf, index=False)
    s3.put_object(Bucket=bucket, Key=key, Body=buf.getvalue())

def _write_json_to_s3(obj: dict, bucket: str, key: str):
    s3.put_object(Bucket=bucket, Key=key, Body=json.dumps(obj, indent=2))

def _clean_dataframe(df: pd.DataFrame) -> pd.DataFrame:
    missing = [c for c in EXPECTED_COLUMNS if c not in df.columns]
    if missing:
        raise ValueError(f"Missing required columns in CSV: {missing}")

    df["city"] = df["city"].apply(_safe_title)
    df["customer_name"] = df["customer_name"].astype(str).str.strip()

    df["quantity"] = _coerce_numeric(df["quantity"], default=0).astype(int)
    df["price"] = _coerce_numeric(df["price"], default=0.0).astype(float)

    mask_valid = (df["customer_name"] != "") & (df["price"] > 0)
    df = df.loc[mask_valid].copy()

    df["total_value"] = df["quantity"] * df["price"]

    if not pd.api.types.is_datetime64_any_dtype(df["order_date"]):
        df["order_date"] = pd.to_datetime(df["order_date"], errors="coerce")
    df = df.dropna(subset=["order_date"]).copy()
    df["order_date"] = df["order_date"].dt.date

    df = df.drop_duplicates()

    ordered_cols = [
        "order_id","customer_name","city","product","quantity","price",
        "discount_code","channel","payment_mode","order_date","total_value",
    ]

    df = df.fillna(0)
    return df[ordered_cols]

def _insert_rows_mysql(df: pd.DataFrame):
    conn = pymysql.connect(
        host=DB_HOST, user=DB_USER, password=DB_PASSWORD,
        database=DB_NAME, connect_timeout=10, autocommit=False,
        cursorclass=pymysql.cursors.Cursor,
    )
    try:
        with conn.cursor() as cur:
            sql = """
                INSERT INTO orders_cleaned
                (order_id, customer_name, city, product, quantity, price,
                 discount_code, channel, payment_mode, order_date, total_value)
                VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)
            """
            data = [tuple(row) for _, row in df.iterrows()]
            chunk = 1000
            for i in range(0, len(data), chunk):
                cur.executemany(sql, data[i:i+chunk])
        conn.commit()
    finally:
        conn.close()

def _maybe_publish_alert(file_total: float, source_key: str):
    try:
        if SNS_TOPIC_ARN and file_total > SALES_ALERT_THRESHOLD:
            msg = (
                f"Daily sales alert. Today's processed file {source_key} generated total sales = {file_total
                f" Target threshold was {SALES_ALERT_THRESHOLD:.2f}. Please consider inventory restocking."
            )
            sns.publish(TopicArn=SNS_TOPIC_ARN, Message=msg, Subject="Daily Sales Summary")
    except Exception as e:
        print(f"SNS publish warning: {e}")

def lambda_handler(event, context):
    print(json.dumps(event))
    record = event["Records"][0]
    bucket = record["s3"]["bucket"]["name"]
    key = record["s3"]["object"]["key"]

    if BUCKET and bucket != BUCKET:
        print(f"Note: event bucket '{bucket}' != env BUCKET '{BUCKET}' – continuing anyway")

    print(f"Triggered by s3://{bucket}/{key}")

    df_raw = _read_csv_from_s3(bucket, key)
    print(f"Raw shape: {df_raw.shape}")

    df_clean = _clean_dataframe(df_raw)
    print(f"Clean shape: {df_clean.shape}")

    fname = key.split("/")[-1]
    processed_key = f"processed/cleaned_{fname}"
    _write_csv_to_s3(df_clean, BUCKET or bucket, processed_key)
    print(f"Wrote cleaned CSV to s3://{BUCKET or bucket}/{processed_key}")

    if df_clean.empty:
        rows_inserted = 0
        file_total = 0.0
    else:
        _insert_rows_mysql(df_clean)
        rows_inserted = len(df_clean)
        file_total = float(df_clean["total_value"].sum())

    print(f"Rows inserted: {rows_inserted}, total sales: {file_total:.2f}")

    _maybe_publish_alert(file_total, key)

    date_suffix = _today_yyyymmdd()
    summary_key = f"analytics/summary_{date_suffix}.json"
    summary = {
        "file": key,
        "processed_csv": processed_key,
        "rows_cleaned": rows_inserted,
        "total_sales": round(file_total, 2),
        "generated_at_utc": datetime.now(timezone.utc).isoformat(),
    }
    _write_json_to_s3(summary, BUCKET or bucket, summary_key)
    print(f"Wrote summary to s3://{BUCKET or bucket}/{summary_key}")

    return {
        "status": "ok",
        "rows_inserted": rows_inserted,
        "total_sales": round(file_total, 2),
        "processed_key": processed_key,
        "summary_key": summary_key,
    }
```

Step 12 — Test Run & Expected Logs (from CloudWatch)

```
Upload: raw/orders_detailed.csv
Expected CloudWatch lines:
  Triggered by s3://omsai-traya-bucket/raw/orders_detailed.csv
  Raw shape: (90, 10)
  Clean shape: (88, 11)
  Wrote cleaned CSV to s3://omsai-traya-bucket/processed/cleaned_orders_detailed.csv
  Rows inserted: 88, total sales: 171411.14
  Wrote summary to s3://omsai-traya-bucket/analytics/summary_YYYYMMDD.json
If SNS endpoint connectivity missing, we saw a warning:
  SNS publish warning: Connect timeout on endpoint URL ...
We fixed it by using SNS Interface Endpoint + setting SNS_TOPIC_ARN to the VPC Endpoint ARN.
```

Step 13 — EC2 Analytics Script

```
On EC2 (first time setup):
sudo yum update -y
sudo yum install python3-pip -y
pip3 install pymysql boto3 --user

Create script:
nano pipeline.py
(paste the full code below, replace YOUR_DB_PASSWORD_HERE, save)

Run once:
python3 pipeline.py
Expected: “uploaded: summary_YYYYMMDD.json” in terminal. Check S3 analytics folder.
```

```
import json
import pymysql
import boto3
from datetime import datetime, timezone
from decimal import Decimal

DB_HOST = "ordersdb.c5bgvvjgyoqm.us-east-1.rds.amazonaws.com"
DB_USER = "admin"
DB_PASSWORD = "YOUR_DB_PASSWORD_HERE"
DB_NAME = "ordersdb"
BUCKET = "omsai-traya-bucket"

s3 = boto3.client("s3")

def today_yyyymmdd():
    return datetime.now(timezone.utc).strftime("%Y%m%d")

def to_float(x):
    return float(x) if isinstance(x, Decimal) else x

def main():
    conn = pymysql.connect(host=DB_HOST, user=DB_USER, password=DB_PASSWORD, database=DB_NAME)
    cur = conn.cursor()

    cur.execute("""
        SELECT city, SUM(total_value) as total_sales
        FROM orders_cleaned
        GROUP BY city
        ORDER BY total_sales DESC
        LIMIT 5;
    """)
    top_cities = [{"city": r[0], "total_sales_in_inr": to_float(r[1])} for r in cur.fetchall()]

    cur.execute("""
        SELECT channel, AVG(total_value) as avg_value
        FROM orders_cleaned
        GROUP BY channel;
    """)
    avg_channel = [{"channel": r[0], "avg_order_value_in_inr": to_float(r[1])} for r in cur.fetchall()]
    cur.execute("""
        SELECT product,
               SUM(CASE WHEN discount_code IS NULL OR discount_code='' THEN 0 ELSE 1 END) AS used,
               COUNT(*) AS total
        FROM orders_cleaned
        GROUP BY product;
    """)
    disc_usage = []
    for row in cur.fetchall():
        product, used, total = row
        used = to_float(used); total = to_float(total)
        rate = (used/total)*100 if total>0 else 0
        disc_usage.append({"product": product, "discount_usage_rate_percent": round(rate, 2)})

    conn.close()

    summary = {
        "generated_at_utc": datetime.now(timezone.utc).isoformat(),
        "top_5_cities_sales_in_inr": top_cities,
        "avg_order_value_per_channel_in_inr": avg_channel,
        "discount_usage_rate_per_product_percent": disc_usage
    }

    filename = f"summary_{today_yyyymmdd()}.json"
    with open(filename, "w") as f:
        json.dump(summary, f, indent=2)

    s3.upload_file(filename, BUCKET, f"analytics/{filename}")
    print("uploaded:", filename)

if __name__ == "__main__":
    main()
```

Step 14 — Schedule with Cron (and Editor Troubleshooting)

```
Install & start cron on Amazon Linux 2023:
  sudo dnf install cronie -y
  sudo systemctl enable --now crond
If editor lags or opens vi:
  • For nano: CTRL+O (save), Enter, CTRL+X (exit)
  • For vi: press i (insert), paste; press ESC, type :wq, Enter
If still lagging, bypass editor completely:
  echo "0 */12 * * * /usr/bin/python3 /home/ec2-user/pipeline.py >> /home/ec2-user/pipeline.log 2>&1" > mycr
  crontab mycron
Verify:
  crontab -l
```

Step 15 — Cost Saving Routine (Sleep Mode)

```
Before sleeping:
  • Release Elastic IP: EC2 → Elastic IPs → select 54.146.201.20 → Release
  • Stop EC2: EC2 → Instances → orders-analytics-instance → Instance state → Stop
  • Stop RDS temporarily: RDS → Databases → ordersdb → Actions → Stop
Notes:
  - While RDS is stopped, small storage/IOPS charge remains — this is minimal.
  - Lambda, S3, SNS, and VPC endpoint continue to cost ~zero for our small workload.
Next day:
  • Start RDS, Start EC2, optionally allocate a new Elastic IP and associate to EC2.
  • Cron will continue to run when EC2 is running.
```

This master guide is the exact working recipe we executed today. You can hand it to any beginner and they will be able to reproduce the pipeline exactly.