AWS Orders Pipeline — Complete Playbook (Final Working Steps Only)

Covers every step we actually used, with names, configs, verification, fixes, and code.

Architecture (Final)

```
S3 (omsai-traya-bucket) → Lambda (orders_pipeline_fn) → RDS MySQL (ordersdb)
                                   ■
                                    EC2 (orders-analytics-instance) → S3 analytics/
Extras: SNS alert via VPC Interface Endpoint (com.amazonaws.us-east-1.sns)
```

Global Settings

```
Region: us-east-1 (N. Virginia)
IAM: LabRole / LabInstanceProfile only (no edits allowed)
Cost mode: Minimal — no NAT, stop EC2 + stop RDS when idle, release Elastic IP
File naming: analytics/summary_YYYYMMDD.json
```

Step 1 — S3 Bucket

```
Service: S3
Bucket: omsai-traya-bucket
Folders created: raw/ , processed/ , analytics/
Lifecycle: Move all objects older than 15 days → Glacier Flexible Retrieval
Trigger target later: Lambda on prefix raw/ (ObjectCreated:Put)
Verify: folders visible; lifecycle rule listed under Management → Lifecycle rules
```

Step 2 — VPC & Subnets (Wizard)

```
Service: VPC → Create VPC (VPC and more)
Name: orders-pipeline-vpc-vpc
Wizard output: 2 public subnets + 2 private subnets
Routing: public subnets route to IGW; private subnets have no IGW
Use: EC2 in public; Lambda + RDS in private
Verify: Subnets list and Route Tables reflect above
```

Step 3 — Security Groups (Final Rules Used)

```
A) lambda-rds-sg  (attached to Lambda and to RDS)
   Inbound:
     - MySQL/Aurora 3306 from lambda-rds-sg (self-reference)   # allows Lambda→RDS
     - MySQL/Aurora 3306 from ec2-analytics-sg                  # allows EC2→RDS
   Outbound: allow all (default)

B) ec2-analytics-sg  (attached to EC2)
   Inbound:
     - SSH 22 from 0.0.0.0/0  (lab convenience)
   Outbound: allow all (default)
Verify: RDS shows lambda-rds-sg; EC2 shows ec2-analytics-sg; Lambda VPC config uses lambda-rds-sg
```

Step 4 — RDS MySQL (Private)

```
Service: RDS → Create database
Engine: MySQL 8.x  | Class: db.t3.micro (Free tier)
Public access: NO
VPC/Subnets: Private subnets in orders-pipeline-vpc-vpc
Security group: lambda-rds-sg
Endpoint: ordersdb.c5bgvvjgyoqm.us-east-1.rds.amazonaws.com
DB user: admin
Schema name: ordersdb
Table created via EC2:
  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
  );
Verify: RDS status = Available; from EC2, mysql client connects successfully
```

Step 5 — Lambda Function (orders_pipeline_fn)

```
Author from scratch; Runtime: Python 3.12; Role: LabRole
VPC: orders-pipeline-vpc-vpc → select the 2 PRIVATE subnets
Security group: lambda-rds-sg
Layers:
  - AWS provided layer: AWSSDKPandas-Python312
Extra libs: PyMySQL vendored in ZIP via CloudShell:
  mkdir orders_lambda && cd orders_lambda
  pip install PyMySQL -t .
  (add lambda_function.py later) → zip -r9 ../orders_pipeline.zip .
  Upload ZIP to Lambda (then edit code in console if needed)
Environment variables (final):
  DB_HOST=ordersdb.c5bgvvjgyoqm.us-east-1.rds.amazonaws.com
  DB_USER=admin
  DB_PASSWORD=<set by you>
  DB_NAME=ordersdb
  BUCKET=omsai-traya-bucket
  SNS_TOPIC_ARN=<VPC Endpoint ARN for SNS>
  SALES_ALERT_THRESHOLD=50000
Verify: Test by uploading raw/orders_detailed.csv, see CloudWatch logs
```

Lambda Code — Final Working Version

```
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", "")
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 6 — S3 Trigger

```
Bucket: omsai-traya-bucket
Event: All object create events
Prefix filter: raw/
Verify: Upload raw/orders_detailed.csv → CloudWatch shows function invoked
```

Step 7 — SNS + VPC Interface Endpoint

```
SNS Topic: created; email subscription confirmed
VPC Endpoint: com.amazonaws.us-east-1.sns (Interface)
  Subnets: 2 private subnets
  Security group: lambda-rds-sg
  Private DNS names enabled: YES
Lambda env var SNS_TOPIC_ARN set to THIS VPC endpoint ARN (Academy restriction bypass)
Verify: After upload with total_sales > threshold, email alert is received
```

Step 8 — EC2 (with Elastic IP during work, then release)

```
Launch: Amazon Linux 2023 t3.micro in PUBLIC subnet
Security group: ec2-analytics-sg
Initial public IP: obtained; then Elastic IP allocated & associated for stable access
  EC2 → Elastic IPs → Allocate → Associate to orders-analytics-instance
After finishing work: RELEASE Elastic IP to avoid charges
MobaXterm SSH: ec2-user@<Elastic-IP> using .pem key
```

```
Packages installed:
  sudo yum update -y
  sudo yum install python3-pip -y
  pip3 install pymysql boto3 --user
Verify: 'python3 --version' and 'pip3 --version' show installed
```

Step 9 — EC2 Analytics Script (pipeline.py)

```
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 10 — Cron (Every 12 hours)

```
Install & start cron on Amazon Linux 2023:
  sudo dnf install cronie -y
  sudo systemctl enable --now crond
Avoid editor lag — load crontab from file:
  echo "0 */12 * * * /usr/bin/python3 /home/ec2-user/pipeline.py >> /home/ec2-user/pipeline.log 2>&1" > mycr
  crontab mycron
  crontab -l  # verify
Verify: pipeline.log updated on schedule
```

Verification Checklist (What we observed)

```
• CloudWatch showed: Raw shape (90,10) → Clean shape (88,11)
• S3 processed: processed/cleaned_orders_detailed.csv present
• Lambda log: Rows inserted: 88, total sales: 171411.14
• S3 analytics: analytics/summary_YYYYMMDD.json written by Lambda
• EC2 run: printed uploaded: summary_YYYYMMDD.json (also in analytics/)
• SNS: email alert after VPC endpoint and env var = endpoint ARN
```

Troubleshooting We Hit & Fixes Applied

```
• MySQL Unknown database 'ordersdb' → Fix: create DB+table from EC2
• NaN cannot be used with MySQL → Fix: df = df.fillna(0) before insert
• SNS connect timeout → Fix: Create SNS VPC Interface Endpoint + set env SNS_TOPIC_ARN to endpoint ARN
• EC2 NoCredentialsError → Fix: attach LabInstanceProfile to EC2
• Cron missing → Fix: sudo dnf install cronie; load crontab from file
• Editor lag / ^Z → Fix: used 'crontab mycron' method
```

Safe Shutdown / Sleep Checklist (Cost Control)

```
• Release Elastic IP (EC2 → Elastic IPs → Release)
• Stop EC2 instance (NOT Terminate)
• Stop RDS temporarily (compute stops; tiny storage cost remains)
• Leave S3/Lambda/SNS/VPC endpoint as-is (negligible cost)
```