Extra Credit - Deployment

Assume you are asked to get your code running in the cloud using AWS. What tools and AWS services would you use to deploy the API, database, and a scheduled version of your data ingestion code? Write up a description of your approach.

1. Database Deployment

Service Used: Amazon RDS (Relational Database Service)

Steps:

- **Create RDS Instance:**
  - Use Amazon RDS to deploy a PostgreSQL database instance. Choose the appropriate instance type based on your workload requirements (e.g., t3.medium for moderate usage).
  - Configure the database instance details such as instance size, storage type (SSD-backed), allocated storage size, and availability zone.
  - Set up the database engine version (PostgreSQL) and optionally enable automatic minor version upgrades.
  - Configure security settings:
    - **Security Group:** Create or select a security group that allows inbound traffic on port 5432 (PostgreSQL default port) from your Elastic Beanstalk environment and Lambda function.
    - **Subnet Group:** Place the database instance in a custom subnet group within your Virtual Private Cloud (VPC) for network isolation.
- **Integrate with AWS Secrets Manager:**
  - Store database credentials securely in AWS Secrets Manager. Create a secret named weather_db_credentials with key-value pairs for username and password.
  - Ensure IAM policies are configured to grant necessary permissions for accessing the secret.
- **Configure Application Environment:**
  - Modify your application's configuration to fetch database credentials securely from AWS Secrets Manager at runtime.
  - Update SQLALCHEMY_DATABASE_URI in your application's environment variables to reference the secret stored in AWS Secrets Manager.

2. API Deployment

Service Used: AWS Elastic Beanstalk

Steps:

- **Create Elastic Beanstalk Application:**
  - Set up an Elastic Beanstalk environment for deploying and managing your Flask-based API application.
  - Choose the appropriate platform (Python) and environment type (Web server environment).
  - Configure environment properties:
    - **Environment Variables:** Define environment variables such as SQLALCHEMY_DATABASE_URI pointing to the RDS instance endpoint, database name, username, and password.
    - **Instance Configuration:** Configure instance types, scaling options (e.g., load balancing, auto-scaling), and EC2 instance security settings.
    - **Network Configuration:** Associate the environment with your VPC and subnet configuration. Ensure security groups allow inbound HTTP traffic (port 80) from the internet.
- **Deploy Flask Application:**
  - Package your Flask application into a ZIP file containing app.py, requirements.txt listing dependencies (including Flask, SQLAlchemy, psycopg2), and other necessary files.
  - Upload the ZIP file to Elastic Beanstalk using the AWS Management Console or AWS CLI.
  - Monitor deployment status through Elastic Beanstalk console and view logs for troubleshooting (eb logs command).
- **Access AWS Secrets Manager in Code:**
  - Use AWS SDKs (boto3 for Python) within your Flask application (app.py) to fetch database credentials securely from AWS Secrets Manager.
  - Retrieve the database credentials (username and password) and construct SQLALCHEMY_DATABASE_URI dynamically based on the fetched values.
- **Configure Auto-scaling and Load Balancing:**
  - Elastic Beanstalk automatically provisions an Application Load Balancer (ALB) to distribute incoming traffic across multiple EC2 instances running your Flask application.
  - Configure auto-scaling policies based on CPU utilization or request metrics to automatically add or remove EC2 instances to handle varying traffic loads.

3. Scheduled Data Ingestion

Service Used: AWS Lambda with EventBridge (formerly CloudWatch Events)

Steps:

- **Create AWS Lambda Function:**
  - Develop a Python function (data_ingestion_lambda.py) using the boto3 library to interact with AWS services and process data ingestion tasks.
  - Ensure the Lambda function has appropriate permissions:
    - **Execution Role:** Create an IAM role granting permissions to access AWS services like RDS (for database operations) and CloudWatch Logs (for logging).
    - **Environment Variables:** Set environment variables such as SQLALCHEMY_DATABASE_URI to securely connect to the RDS instance.
- **Use AWS Secrets Manager in Lambda Function:**
  - Grant Lambda function permissions (secretsmanager:GetSecretValue) to access the weather_db_credentials secret stored in AWS Secrets Manager.
  - Update the Lambda function code (data_ingestion_lambda.py) to retrieve database credentials dynamically from AWS Secrets Manager.
- **Upload Code to Lambda:**
  - Package your Lambda function code (data_ingestion_lambda.py) along with any additional dependencies (psycopg2, sqlalchemy) into a ZIP file.
  - Upload the ZIP file to Lambda using the AWS Management Console, AWS CLI, or AWS SDKs.
- **Schedule Execution using EventBridge:**
  - Use AWS EventBridge to schedule the Lambda function to execute at predefined intervals (e.g., daily at 2 AM UTC).
  - Configure the schedule using cron expressions (cron(0 2 * * ? *)) to trigger the Lambda function based on UTC time.
- **Error Handling and Logging:**
  - Implement error handling within the Lambda function to capture exceptions and errors encountered during data ingestion.
  - Log detailed execution logs to AWS CloudWatch Logs for monitoring and troubleshooting (print statements or logging module).

4. Additional Considerations

- **Security:**
    - **IAM Roles and Policies:** Apply the principle of least privilege by creating IAM roles with specific permissions for each service (RDS, Lambda, Elastic Beanstalk).
    - **Encryption:** Enable encryption at rest for the RDS instance using AWS KMS (Key Management Service) and in-transit using SSL/TLS.
    - **Network Isolation:** Use VPC Endpoints for accessing AWS services (e.g., S3 for data storage) privately without exposing them to the internet.
- **Monitoring and Logging:**
    - **CloudWatch Metrics and Alarms:** Set up CloudWatch alarms to monitor API latency, error rates, and Lambda function invocations. Create custom metrics for monitoring specific application-level metrics.
    - **Dashboards:** Build custom CloudWatch dashboards to visualize key performance indicators (KPIs) and operational metrics for the API and data ingestion processes.
- **Scaling:**
    - **Elastic Beanstalk Auto-scaling:** Configure Elastic Beanstalk environment to scale automatically based on CPU utilization or request metrics.
    - **Lambda Concurrency:** Adjust Lambda function concurrency limits to handle concurrent executions during peak data ingestion periods.
- **Backup and Recovery:**
    - **RDS Automated Backups:** Enable automated backups for the RDS instance to retain backups for point-in-time recovery.
    - **Snapshot Management:** Create manual snapshots of the RDS instance before making significant changes to the database schema or configuration.