01 Amazon RDS Intro

Amazon RDS, Aurora, and Integrating MySQL RDS with Flask

1. What is Amazon RDS?

Amazon Relational Database Service (RDS) is a **managed database service** provided by AWS that simplifies the setup, operation, scaling, and maintenance of relational databases in the cloud.

Key Features of RDS

- **Managed Service**: AWS handles backups, patching, monitoring, and failure recovery.
- **Supported Engines**:
  - MySQL
  - PostgreSQL
  - MariaDB
  - Oracle
  - Microsoft SQL Server
  - **Amazon Aurora** (MySQL/PostgreSQL-compatible)
- **High Availability**: Multi-AZ deployments for failover.
- **Automated Backups**: Point-in-time recovery, snapshots.
- **Scaling**:
  - Vertical: Change instance size
  - Horizontal: Read replicas
- **Security**: IAM integration, VPC security groups, encryption at rest/transit.

---

2. Use Cases & Real-World Examples

Use Case	Example
Web & Mobile Applications	E-commerce platform using MySQL RDS to store user data, orders, inventory.
Business Intelligence (BI)	Data warehouse with PostgreSQL RDS + read replicas for reporting dashboards.
Dev/Test Environments	Spin up temporary MySQL RDS instances for CI/CD pipelines.
SaaS Applications	Multi-tenant app using Aurora Serverless for auto-scaling per tenant load.
Enterprise Applications	Migrating on-prem Oracle database to RDS Oracle with minimal downtime.

> **Example**: A startup builds a Flask blog app. They use **MySQL RDS** to store posts, comments, and user profiles. During traffic spikes (e.g., viral post), read replicas offload traffic.

---

3. What is Amazon Aurora?

Aurora is a **MySQL and PostgreSQL-compatible** relational database built for the cloud by AWS.

Aurora vs Standard RDS (MySQL/PostgreSQL)

Feature	Standard RDS (MySQL)	Aurora
Performance	Good	Up to **5x faster** than MySQL, **3x** than PostgreSQL
Storage	EBS-based	Distributed, auto-scaling (10 GB → 128 TB)
Replication	Read replicas (up to 15)	Up to **15 read replicas**, sub-10ms latency
High Availability	Multi-AZ (1 standby)	**6 copies across 3 AZs**, continuous backup
Serverless Option	No	**Aurora Serverless v2** – auto-scale to zero
Cost	Pay per instance hour + storage	Higher compute, but lower I/O cost

Aurora Architecture

- **Storage Layer**: Decoupled from compute. Data replicated 6 ways across 3 AZs.
- **Compute Layer**: Instances only run SQL engine; storage is shared.
- **Failover**: <30 seconds (vs ~2 mins in standard RDS).

> **Use Aurora when**: You need high performance, scalability, and availability with MySQL/PostgreSQL compatibility.

---

4. Integrating MySQL RDS with a Flask Application

Step-by-Step Setup

Step 1: Create MySQL RDS Instance (via AWS Console)

1. Go to **RDS > Create database**
2. Choose:
   - Engine: **MySQL**
   - Edition: Latest (e.g., 8.0)
   - Template: **Production** (Multi-AZ) or **Dev/Test**
3. Settings:
   - DB instance identifier: `flask-blog-db`
   - Master username: `admin`
   - Password: [secure password]
4. Instance configuration: `db.t3.micro` (free tier eligible)
5. Storage: 20 GB, enable auto-scaling
6. **Connectivity**:
   - VPC: Default or custom
   - **Public access**: **Yes** (for dev; **No** in production)

- Subnet group: Multi-AZ

7. **Security**:
   - Create new security group: `rds-flask-sg`
   - **Inbound rules** → Add rule:

```
Type: MySQL/Aurora (port 3306)
Source: Your IP (or EC2 security group)
```

1. Launch DB → Wait for status: **Available**

> **Note**: Get the **Endpoint** (e.g., `flask-blog-db.xxxx.us-east-1.rds.amazonaws.com`)

Step 2: Configure Security Group (Inbound Rules)

Rule	Purpose
Type: MySQL (3306)	Allow DB connections
Source: `sg-xxxxx` (EC2 instance SG)	**Best Practice** – Allow only from app server
OR Source: `0.0.0.0/0`	Only for testing; **never in production**

> **Production Best Practice**:
>
> - Place Flask app in **EC2 or ECS** in same VPC
> - Allow RDS inbound **only from app's security group**
> - Use **IAM DB Authentication** (optional, passwordless)

Step 3: Flask Application Code

PYTHON

```python
# app.py
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
import os

app = Flask(__name__)

# RDS Configuration
app.config['SQLALCHEMY_DATABASE_URI'] = (
    'mysql+pymysql://admin:yourpassword@'
    'flask-blog-db.xxxx.us-east-1.rds.amazonaws.com:3306/blogdb'
)
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

# Model
class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    content = db.Column(db.Text, nullable=False)

# Create tables
with app.app_context():
    db.create_all()

@app.route('/')
def home():
    posts = Post.query.all()
    return f"<h1>Posts: {len(posts)}</h1>"

if __name__ == '__main__':
    app.run(debug=True)
```

Step 4: Secure Connection (Best Practices)

1. **Use Environment Variables** (never hardcode credentials):

```python
# Use python-decouple or dotenv
from decouple import config

app.config['SQLALCHEMY_DATABASE_URI'] = (
    f"mysql+pymysql://{config('DB_USER')}:{config('DB_PASS')}@"
    f"{config('DB_HOST')}:{config('DB_PORT')}/{config('DB_NAME')}"
)
```

`.env` file:

```
DB_USER=admin
DB_PASS=SuperSecret123!
DB_HOST=flask-blog-db.xxxx.us-east-1.rds.amazonaws.com
DB_PORT=3306
DB_NAME=blogdb
```

2. **Enable SSL/TLS** (RDS enforces it by default in some regions):

```python
app.config['SQLALCHEMY_DATABASE_URI'] += '?ssl_ca=rds-combined-ca-bundle.pem'
```

Download certificate:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.SSL.html

3. **Connection Pooling** (for production):

```python
app.config['SQLALCHEMY_ENGINE_OPTIONS'] = {
    'pool_size': 5,
    'max_overflow': 10,
    'pool_timeout': 30,
}
```

Step 5: Testing the Connection

```shell
$ python app.py
```

Visit: `http://localhost:5000` → Should show number of posts.

Summary

Topic	Key Takeaway
RDS	Managed relational DB with automation
Aurora	High-performance, scalable, cloud-native MySQL/PG
Use Cases	Web apps, BI, SaaS, enterprise migration
Flask + RDS	Use `SQLAlchemy`, env vars, security groups, SSL
Security	Restrict inbound to app SG, use IAM, encrypt

Homework / Practice

1. Deploy a **Flask + MySQL RDS** app on EC2.
2. Add a **read replica** and route `SELECT` queries to it.
3. Migrate the DB to **Aurora MySQL** and compare performance.
4. Enable **[RDS Proxy]** for connection pooling.
