Author: Madhurima Rawat

Cloud Databases and Data Management

This experiment covers setting up and managing cloud-based relational databases using PostgreSQL, Docker, and LocalStack. It provides hands-on experience with database management in a simulated cloud environment. The setup replicates real-world cloud database operations using containerized solutions.

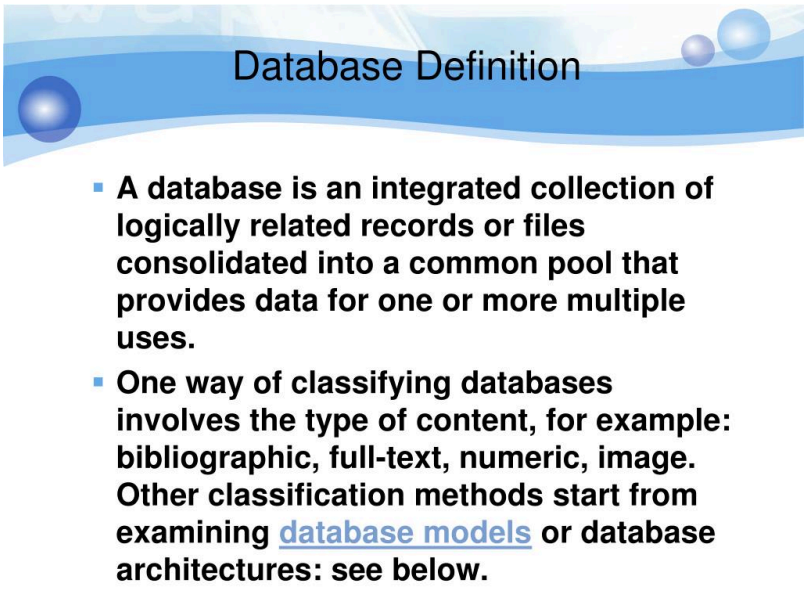
Cloud Databases and Data Management

Overview

This experiment covers setting up and managing cloud-based relational databases using **PostgreSQL, Docker, and LocalStack**. It provides hands-on experience with database management in a **simulated cloud environment**, replicating real-world cloud database operations using containerized solutions.

What are Databases?

Databases store, organize, and manage data efficiently. They are crucial in applications ranging from websites to large-scale enterprise systems.

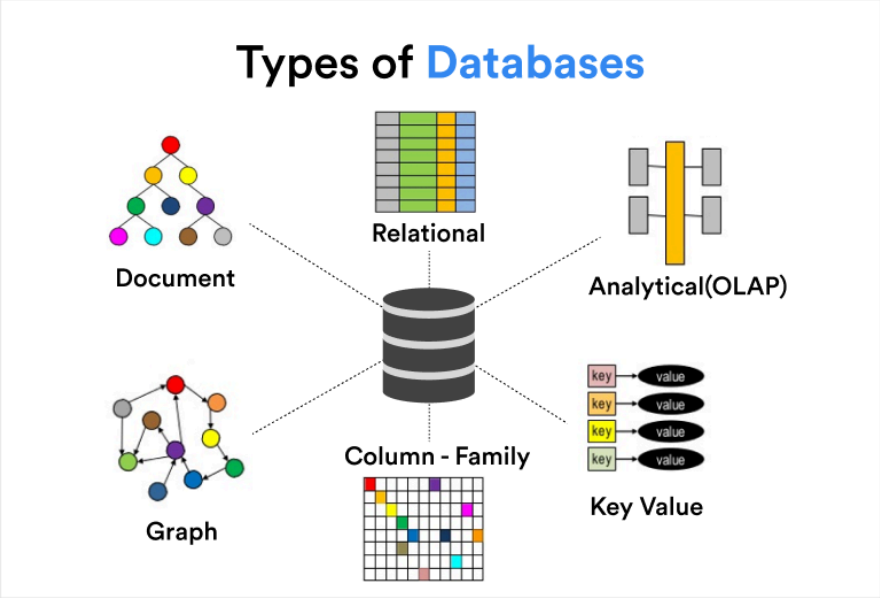


Types of Databases

Databases are broadly categorized into:

- **Relational Databases (SQL-based)** – Structured data stored in tables (e.g., PostgreSQL, MySQL).
- **NoSQL Databases** – Flexible schema for handling unstructured data (e.g., MongoDB, Cassandra).

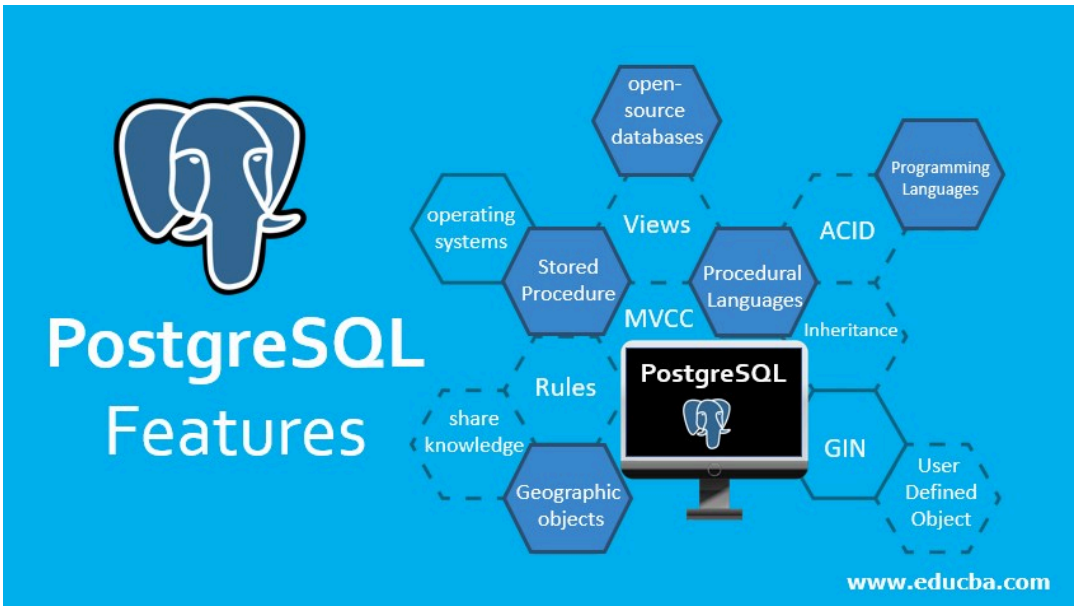
- **Cloud Databases** – Managed services with scalable storage (e.g., AWS RDS, Google Cloud Firestore).



PostgreSQL Features

PostgreSQL is a powerful, open-source relational database with features like:

- **ACID Compliance** – Ensures data integrity.
- **Extensibility** – Supports custom functions and data types.
- **Scalability** – Handles large volumes of data.



Real-World Application & Case Study

Use Case: Financial Data Management
A leading bank implemented PostgreSQL on the cloud to handle transaction processing, fraud detection, and real-time analytics. With **Dockerized deployments**, they achieved high availability, ensuring **99.9% uptime** and enhanced security.

This experiment provides insights into deploying such systems using **Docker and LocalStack** to simulate real-world cloud database management.

Database Operations with Postgres

1. Creating an RDS Instance Using LocalStack

Command:

```
aws rds create-db-instance --db-instance-identifier mydb \
  --db-instance-class db.t3.micro \
  --engine mysql \
  --master-username admin \
  --master-user-password password \
  --allocated-storage 20 \
  --endpoint-url=http://localhost:4566
```

Error Output:

```
Could not connect to the endpoint URL: "http://localhost:4566/"
```

```
An error occurred (InternalFailure) when calling the CreateDBInstance operation:
API for service 'rds' not yet implemented or pro feature - please check
https://docs.localstack.cloud/references/coverage/ for further information
```

Explanation:

- The command attempts to create an **RDS instance** in **LocalStack**.
- `--endpoint-url=http://localhost:4566` → Uses LocalStack instead of AWS.
- **Errors indicate**:
  - LocalStack is either not running or misconfigured.
  - RDS API might not be fully implemented in the **free** version of LocalStack.

Output Breakdown:

- **`Could not connect to the endpoint URL`** → LocalStack might not be running or accessible.
- **`InternalFailure` error** → The RDS API might require LocalStack Pro for full functionality.
- **Possible Fixes**:
  - Ensure LocalStack is running:

```
docker run --rm -d --name localstack_main -p 4566:4566 localstack/localstack
```

- Check service coverage:
  LocalStack RDS Coverage

2. Starting a PostgreSQL Container

Command:

```
docker start my-postgres
```

```
docker start postgres
```

Error Output:

```
Error response from daemon: No such container: my-postgres
Error: failed to start containers: my-postgres
```

```
Error response from daemon: No such container: postgres
Error: failed to start containers: postgres
```

Explanation:

- The **containers do not exist** under the specified names.
- Verify running containers with:

  ```
  docker ps -a
  ```

- If needed, create a new container:

  ```
  docker run --name my-postgres -e POSTGRES_USER=admin \
    -e POSTGRES_PASSWORD=password -e POSTGRES_DB=mydb \
    -p 5432:5432 -d postgres:15
  ```

Output Breakdown:

- **`No such container: my-postgres`** → The container was never created or was removed.
- **`failed to start containers`** → The container name does not match any existing instances.
- **Possible Fixes**:
  - Check existing containers: `docker ps -a`
  - Create and start a new PostgreSQL container using `docker run` (above).

3. Listing Available Docker Images

Command:

```
docker images
```

Output:

```
| REPOSITORY            | TAG    | IMAGE ID     | CREATED      | SIZE   |
| --------------------- | ------ | ------------ | ------------ | ------ |
| my-flask-app          | latest | f5feae0ac7a4 | 6 hours ago  | 139MB  |
| flask-app             | latest | ae4054c49614 | 7 hours ago  | 139MB  |
| hackvortex-backend    | latest | 14e63c26d40b | 21 hours ago | 1.05GB |
| postgres              | 15     | e45d3f5ec589 | 7 days ago   | 430MB  |
| localstack/localstack | latest | b686f3948f42 | 6 weeks ago  | 1.18GB |
| python                | 3.9    | 9f98746e2033 | 3 months ago | 999MB  |
| nginx                 | latest | b52e0b094bc0 | 4 weeks ago  | 192MB  |
```

Explanation:

- Displays **available images** in the local Docker environment.
- PostgreSQL (`postgres:15`) is available.
- LocalStack (`localstack/localstack`) is present but needs verification (`docker ps -a`).

Output Breakdown:

- **`postgres:15` is listed** → The image exists but the container may not be running.
- **`localstack/localstack` exists** → LocalStack is installed but may need to be started.
- **Possible Fixes**:
  - Start PostgreSQL if not running:

    ```
    docker run --name my-postgres -e POSTGRES_USER=admin \
      -e POSTGRES_PASSWORD=password -e POSTGRES_DB=mydb \
      -p 5432:5432 -d postgres:15
    ```

  - Ensure LocalStack is running:

    ```
    docker start localstack_main
    ```

4. Starting a PostgreSQL Container

Command:

```
C:\Users\rawat>docker start postgres
```

Error Output:

```
Error response from daemon: No such container: postgres
Error: failed to start containers: postgres
```

5. Listing All Containers

Command:

```
C:\Users\rawat>docker ps -a
```

Output:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
a10c5a71f625	localstack/localstack	"docker-entrypoint.sh"	2 minutes ago	Up 2 minutes (healthy)	127.0.0.1:4510-4560->4510-4560/tcp, 127.0.0.1:4566->4566/tcp, 5678/tcp
7f0fa023ac4f	3a669f02efff	"python app.py"	7 hours ago	Exited (255) 5 minutes ago	8080/tcp, 0.0.0.0:5002->5000/tcp
9ff472da8892	3a669f02efff	"python app.py"	7 hours ago	Exited (255) 5 minutes ago	8080/tcp, 0.0.0.0:5001->5000/tcp

6. Running a PostgreSQL Container

Command:

```
C:\Users\rawat>docker run --name my-postgres -e
POSTGRES_USER=admin -e POSTGRES_PASSWORD=password -e
POSTGRES_DB=mydb -p 5432:5432 -d postgres:15
```

Error Output:

```
docker: Error response from daemon: driver failed
programming external connectivity on endpoint my-postgres
(feae7f0fb87909bde1853a7ddefa49bb518f11250e54304f75109
68f7a88cca1): Bind for 0.0.0.0:5432 failed: port is already allocated.
```

7. Resolving Port Conflict and Running PostgreSQL on a Different Port

Command:

```
C:\Users\rawat>docker run --name my-new-postgres -e
POSTGRES_USER=admin -e POSTGRES_PASSWORD=password -e
POSTGRES_DB=mydb -p 5433:5432 -d postgres:15
```

Output:

```
b2efdca3c6f0af6cf4154fce236f0b66b5efba0f4f9e14972c94b3e0a5afa9de
```

8. Verifying Running Containers

Command:

```
C:\Users\rawat>docker ps
```

Output:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
b2efdca3c6f0	postgres:15	"docker-entrypoint.s..."	42 seconds ago	Up 41 seconds	0.0.0.0:5433->5432/tcp

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
a10c5a71f625	localstack/localstack	"docker-entrypoint.sh"	3 minutes ago	Up 3 minutes (healthy)	127.0.0.1:4510-4560->4510-4560/tcp, 127.0.0.1:4566->4566/tcp,

9. Connecting to PostgreSQL and Performing SQL Operations

Command:

```
C:\Users\rawat>docker exec -it my-new-postgres psql -U admin -d mydb
```

Output:

```
psql (15.12 (Debian 15.12-1.pgdg120+1))
Type "help" for help.
```

Creating a Table and Inserting Data:

```
CREATE TABLE students (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL
);
INSERT INTO students (name, email) VALUES
('Alice Johnson', 'alice@example.com'),
('Bob Smith', 'bob@example.com'),
('Charlie Brown', 'charlie@example.com');
```

Output:

```
CREATE TABLE
INSERT 0 3
```

10. Performing SQL Queries

Selecting Data:

```
SELECT * FROM students;
```

Output:

id	Name	Email
1	Alice Johnson	alice@example.com
2	Bob Smith	bob@example.com
3	Charlie Brown	charlie@example.com

Updating Data:

```
UPDATE students SET email = 'bob.smith@example.com'
WHERE name = 'Bob Smith';
```

Output:

```
UPDATE 1
```

Deleting Data:

```
DELETE FROM students WHERE name = 'Charlie Brown';
```

Output:

```
DELETE 1
```

Selecting Data with a Condition:

```
SELECT * FROM students WHERE name LIKE 'A%';
```

Output:

id	Name	Email
1	Alice Johnson	alice@example.com

Exiting PostgreSQL:

```
\q
```