



---

# **Setup DB for lambda**

**Mobile App Programming**

---

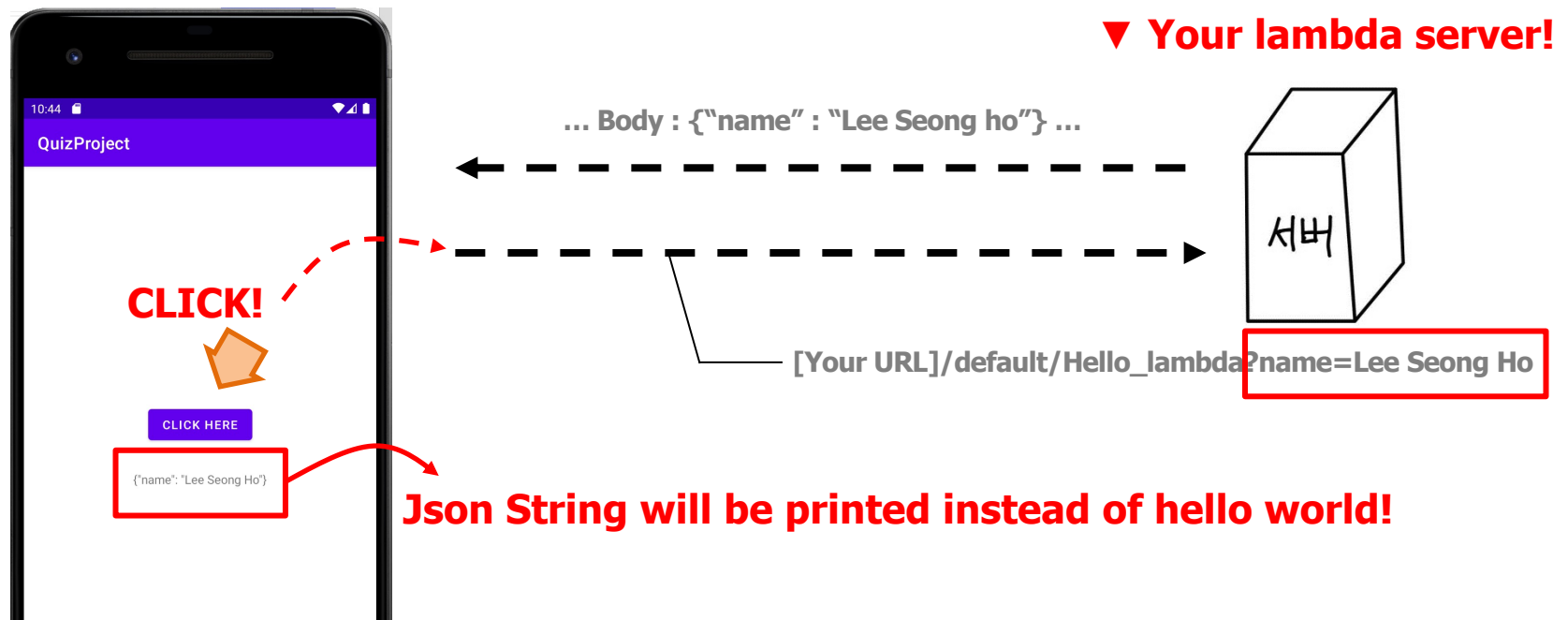


# What we learn today?

- **Let's connect AWS RDS PostgreSQL to flask.**
  - Implement a web server with PostgreSQL using SQLAlchemy.
  - Simply upload web server to AWS Lambda.
  - Make application contains login function.

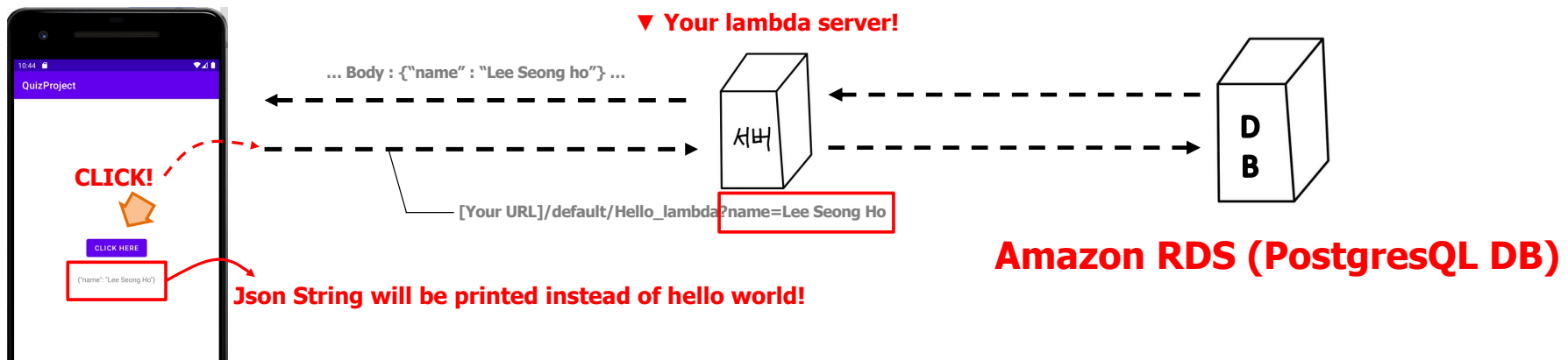
# Remind

- **We upload Flask server to AWS Lambda using Zappa!**
  - We implemented a simple server using python Flask
  - We upload our flask server to AWS lambda using Zappa.



# Today ...

- **We connect our lambda flask with postgresQL RDS instance.**
  - We create RDS instance (PostgresQL).
  - We connect our flask server to RDS instance!





# What we learn today?

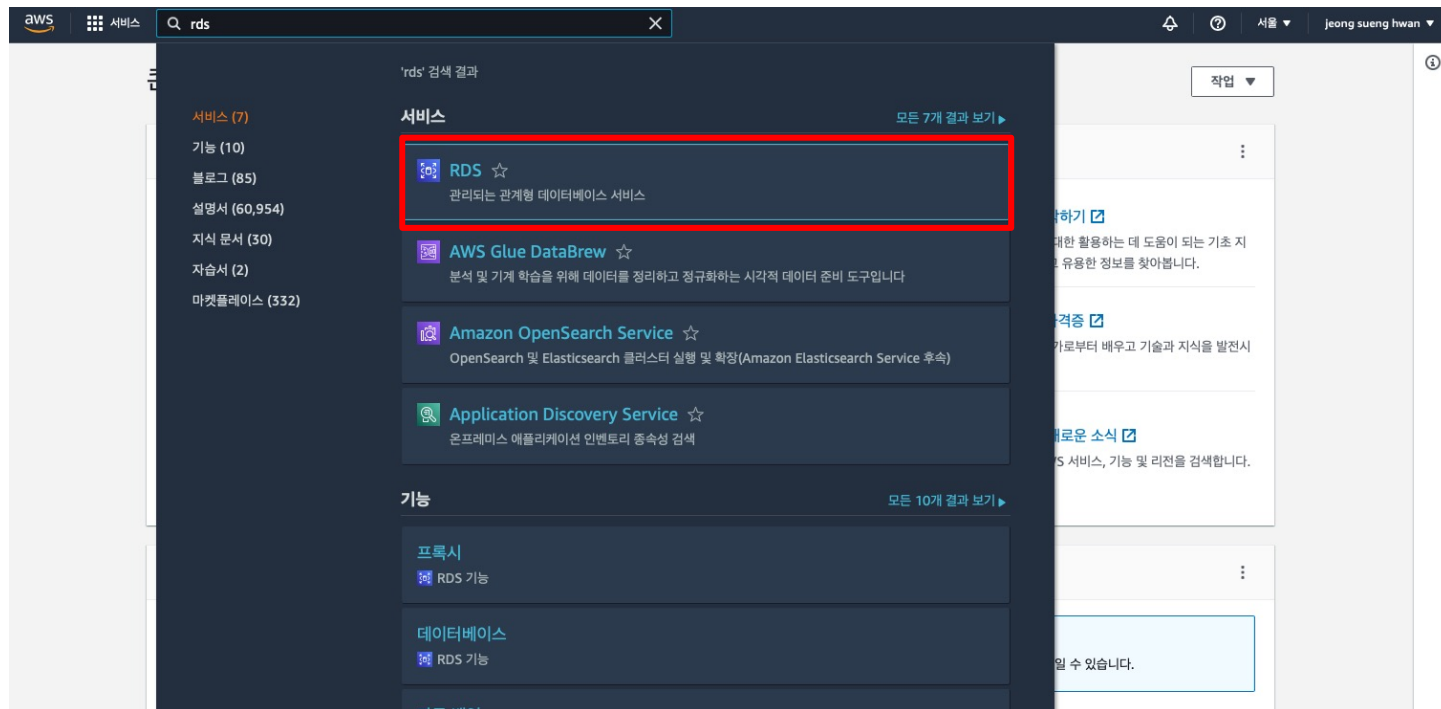
- **Process**

- 1) Make a new RDS instance
- 2) Setting VPC network
- 3) Install SQLAlchemy and psycopg2-binary
- 4) Create Engine, Session, Base
- 5) Send add transaction and query transaction

**Exercise - Make an application with login function**

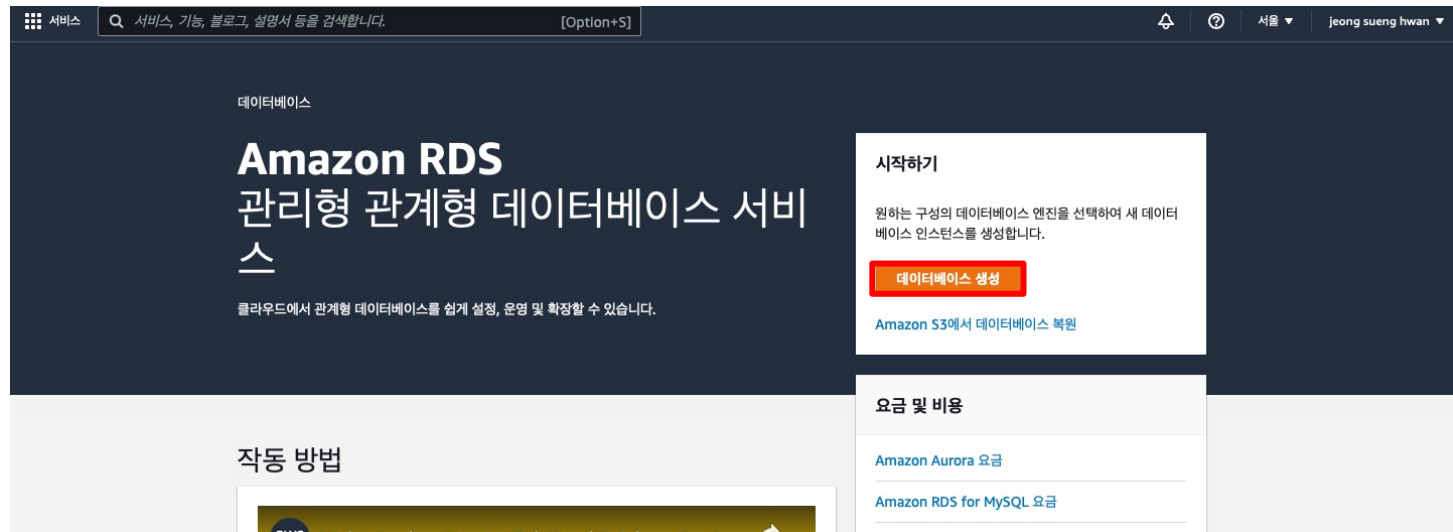
# 1) Make a new RDS instance

- Go to AWS console, search RDS and select it



# 1) Make a new RDS instance

- Click “Create new database”
- **Please be careful to setting your RDS.**



# 1) Make a new RDS instance

## • Setting a new RDS database

RDS > 데이터베이스 생성

### 데이터베이스 생성

데이터베이스 생성 방식 선택 정보

☒ 표준 생성  
가용성, 보안, 백업 및 유지 관리에 대한 옵션을 포함하여 모든 구성 옵션을 설정합니다.

☐ 손쉬운 생성  
관장 모델 사례 구성을 사용합니다. 일부 구성 옵션은 데이터베이스를 생성한 후 변경할 수 있습니다.

엔진 옵션

엔진 유형 정보

☐ Amazon Aurora

☐ MySQL

☐ MariaDB

☒ PostgreSQL

☐ Oracle

☐ Microsoft SQL Server

버전

PostgreSQL 13.4-R1

템플릿

해당 사용 사례를 충족하는 샘플 템플릿을 선택하세요.

☐ 프로덕션  
고가용성 및 빠르고 일관된 성능을 위해 기본값을 사용하세요.

☐ 개발/테스트  
이 인스턴스는 프로덕션 환경 외부에서 개발 용도로 마련되었습니다.

☒ 프리 티어  
RDS 프리 티어를 사용하여 새로운 애플리케이션을 개발하거나, 기존 애플리케이션을 테스트하거나 Amazon RDS에서 실무 경험을 쌓을 수 있습니다. [정보](#)

**Must select free-tier!!!**

### 설정

DB 인스턴스 식별자 정보

DB 인스턴스 이름을 입력하세요. 이름은 현재 AWS 리전에서 AWS 계정이 소유하는 모든 DB 인스턴스에 대해 고유해야 합니다.

map-database

DB 인스턴스의 역할자는 소문자로 구분하지 않지만 'mydbinstance'와 같이 모두 소문자로 저장됩니다. 제약: 1자~60자의 영숫자 또는 하이픈으로 구성되어야 합니다. 첫 번째 문자는 글자이어야 합니다. 하이픈 2개가 연속될 수 없습니다. 끝에 하이픈이 올 수 없습니다.

▼ 자격 증명 설정

마스터 사용자 이름 정보

DB 인스턴스의 마스터 사용자에 로그인 ID를 입력하세요.

postgres

1~16자의 영숫자, 첫 번째 문자는 글자이어야 합니다.

☐ 암호 자동 생성  
Amazon RDS에서 사용자를 대신하여 암호를 생성하거나 사용자가 직접 암호를 지정할 수 있습니다.

마스터 암호 정보

\*\*\*\*\*

제약 조건: 8자 이상의 인쇄 가능한 ASCII 문자. 다음을 포함할 수 없습니다. /(슬래시), '(작은따옴표)', '(큰따옴표) 및 @ (앳 기호).

암호 확인 정보

\*\*\*\*\*

**Please don't forget master username and password**

### 인스턴스 구성

아래의 DB 인스턴스 구성 옵션은 위에서 선택한 엔진에서 지원하는 옵션으로 제한됩니다.

DB 인스턴스 클래스 정보

☐ 스탠다드 클래스(m 클래스 포함)

☐ 메모리 최적화 클래스(r 및 x 클래스 포함)

☒ 버스터블 클래스(t 클래스 포함)

db.t3.micro

2 vCPUs 1 GiB RAM db.t3.micro 35Mbps

☐ 이전 세대 클래스 포함



# 1) Make a new RDS instance

- Setting a new RDS database

### 스토리지

#### 스토리지 유형 정보

범용 SSD(gp2)

볼륨 크기에 따라 기본 성능 결정

할당된 스토리지

20

GIB

(최소: 20GiB, 최대: 16,384GiB) 할당된 크로리저가 필요한 IOPS 성능이 **unlimited**.

#### 스토리지 자동 조정 정보

애플리케이션의 필요에 따라 데이터베이스 스토리지의 동적 조정 지원을 제공합니다.

☐ 스토리지 자동 조정 활성화  
이 기능을 활성화하면 지정된 임계값 초과 시 스토리지를 늘릴 수 있습니다.

### 가용성 및 내구성

#### 다중 AZ 배포 정보

☒ 대기 인스턴스 생성(생산 사용량에 권장)  
데이터 중복을 제공하고, I/O 중지를 없애고, 시스템 백업 중에 지연 시간 스파이크를 최소화하기 위해 다른 가용 영역(AZ)에 대기 인스턴스를 생성합니다.

☐ 대기 인스턴스를 생성하지 마세요.

Turn off auto-scaling option!!

### 연결

## Keep going with default option!

#### Virtual Private Cloud(VPC) 정보

이 DB 인스턴스의 가상 네트워킹 환경을 정의하는 VPC.

Default VPC ( )

해당 DB 서버넷 그룹이 있는 VPC만 나열됩니다.

① 데이터베이스를 생성한 후에는 VPC를 변경할 수 없습니다.

#### 서브넷 그룹 정보

선택한 VPC에서 DB 인스턴스가 어떤 서브넷과 IP 범위를 사용할 수 있는지를 정의하는 DB 서버넷 그룹.

default ( )

#### 퍼블릭 액세스 정보

☒ 예  
VPC 외부의 Amazon EC2 인스턴스 및 디바이스는 데이터베이스에 연결할 수 있습니다. 데이터베이스에 연결할 수 있는 VPC 내부의 EC2 인스턴스 및 디바이스를 지정하는 하나 이상의 VPC 보안 그룹을 선택하십시오.

☐ 아니요  
RDS는 데이터베이스에 퍼블릭 IP 주소를 할당하지 않습니다. VPC 내부의 Amazon EC2 인스턴스 및 디바이스만 데이터베이스에 연결할 수 있습니다.

#### VPC 보안 그룹

데이터베이스에 대한 액세스를 허용할 VPC 보안 그룹을 선택합니다. 보안 그룹 규칙이 적절한 수신 트래픽을 허용하는지 확인합니다.

☒ 기존 항목 선택  
기존 VPC 보안 그룹 선택

☐ 새로 생성  
새 VPC 보안 그룹 생성

#### 기존 VPC 보안 그룹

VPC 보안 그룹 선택

default X

#### 가용 영역 정보

기본 설정 없음

▶ 추가 구성

Check public access to "Yes"

# 1) Make a new RDS instance

## • Setting a new RDS database

### 데이터베이스 인증

데이터베이스 인증 옵션 [정보](#)

☒ 암호 인증  
데이터베이스 암호를 사용하여 인증합니다.

☐ 암호 및 IAM 데이터베이스 인증  
AWS IAM 사용자 및 역할을 통해 데이터베이스 암호와 사용자 자격 증명을 사용하여 인증합니다.

☐ 암호 및 Kerberos 인증  
권한이 부여된 사용자가 Kerberos 인증을 사용하여 이 DB 인스턴스에서 인증하도록 허용하려는 디렉터리를 선택합니다.

### ▶ 추가 구성

데이터베이스 옵션, 암호화 활성화, 백업 활성화, 역추적 비활성화, 성능 개선 도우미 활성화, 향상된 모니터링 비활성화, 유지 관리, CloudWatch Logs, 식재 보호 비활성화.

### 월별 추정 요금

Amazon RDS 프리 티어는 12개월 동안 사용할 수 있습니다. 매월 프리 티어를 통해 아래 나열된 Amazon RDS 리소스를 무료로 사용할 수 있습니다.

- 단일 AZ db.t2.micro 인스턴스에서 Amazon RDS의 750시간.
- 20GB의 범용 스토리지(SSD).
- 20GB의 자동 백업 스토리지 및 사용자가 시작한 모든 DB 스냅샷.

[AWS 무료 티어에 대해 자세히 알아보세요.](#)

무료 사용이 만료되었거나 애플리케이션에서 프리 티어 사용량을 초과한 경우 [Amazon RDS 요금 페이지](#)에서 설명한 대로, 표준 종량 서비스 요금이 적용됩니다.

❗ 귀하는 AWS 서비스와 함께 사용하는 타사 제품 또는 서비스 일체에 대해 필요한 모든 권리를 보유할 책임이 있습니다.

취소 **데이터베이스 생성**

Create a PostgreSQL Database!

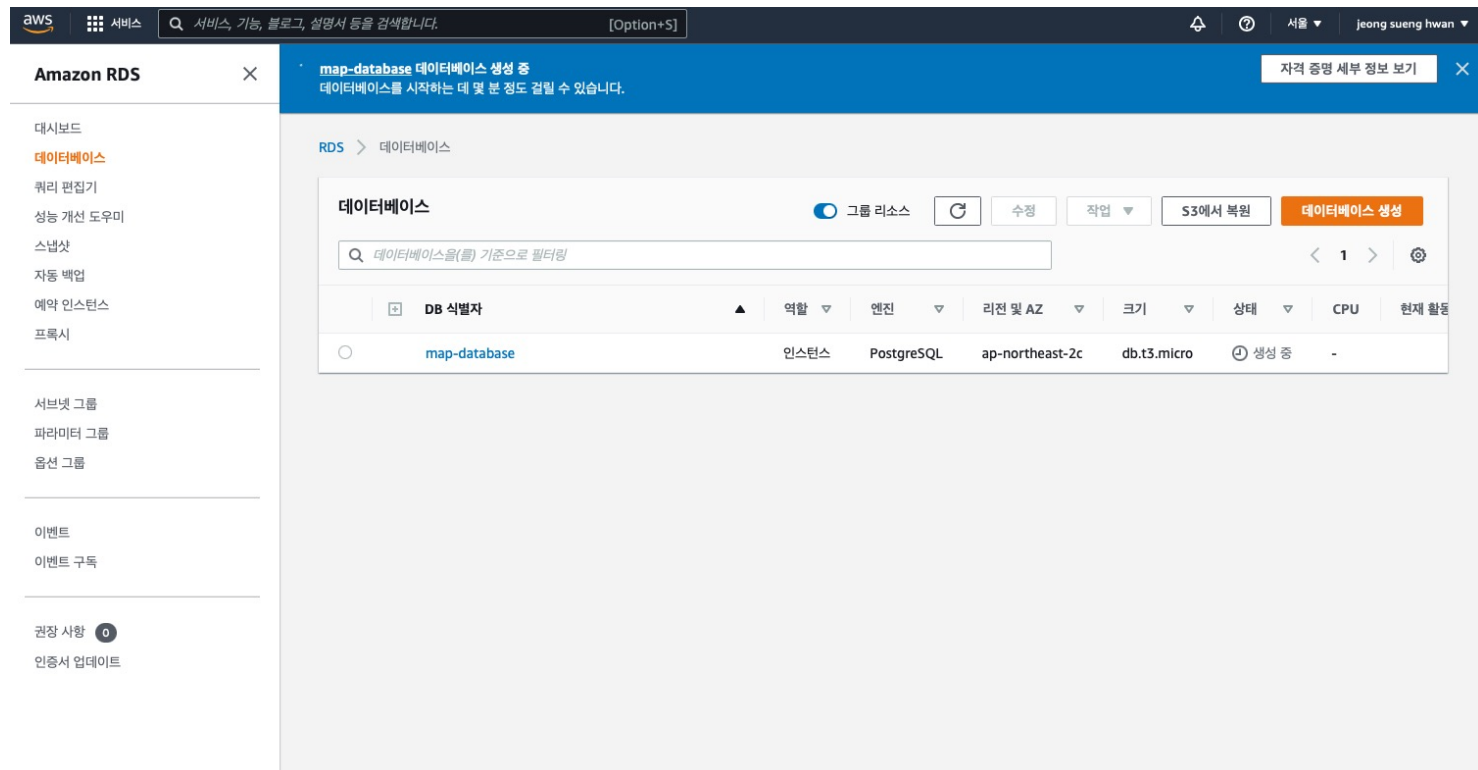
# 1) Make a new RDS instance

- **Be careful!!!!**

- Free-tier RDS instance is only when **db.t3.micro** class with **20GB** storage.
- In addition, if you use RDS instances more than **750 hours/month**, you must pay for overuse.
  - Ex) 1 RDS instance during 1 month  
 $\Rightarrow 24 \text{ hours} * 31 = 744 \text{ hours/month} \rightarrow \text{free!}$
  - Ex) 2 RDS instance during 20 days  
 $\Rightarrow 24 \text{ hours} * 20 * 2 = 960 \text{ hours/month} \rightarrow \text{pay for 210 hours/month!}$
- Therefore, you must keep only **1 RDS free-tier instance!!**
- When you do your project, remove other one.

# 1) Make a new RDS instance

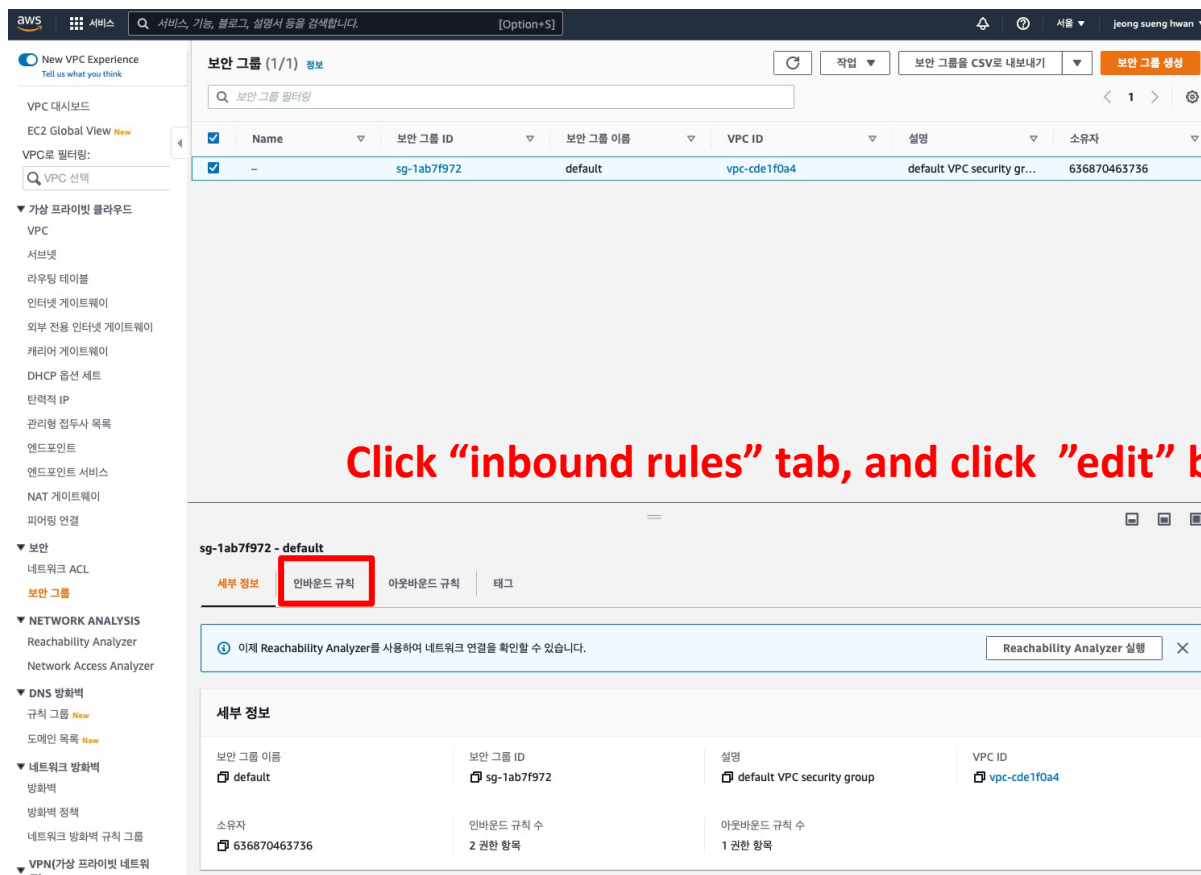
- New database is created successfully!



- **Search “VPC” and select it. Click “Security Group”.**

## 2) Setting VPC network

- Search "VPC" and select it. Click "Security Group".



The screenshot shows the AWS Management Console interface for the '보안 그룹 (1/1) 정보' (Security Groups) page. The left sidebar contains navigation links for various AWS services, including '보안' (Security) and '네트워크' (Network). The main content area displays a table of security groups. The first row is selected, showing details for the security group 'sg-1ab7f972'. The '보안 그룹 ID' (Security Group ID) is highlighted in red. Below the table, the '세부 정보' (Details) section shows the '인바운드 규칙' (Inbound rules) tab selected. The 'edit' button is visible next to the '인바운드 규칙' tab. A red text overlay reads: 'Click "inbound rules" tab, and click "edit" button'.

Name	보안 그룹 ID	보안 그룹 이름	VPC ID	설명	소유자
-	sg-1ab7f972	default	vpc-cde1f0a4	default VPC security gr...	636870463736

**세부 정보**

보안 그룹 이름	보안 그룹 ID	설명	VPC ID
default	sg-1ab7f972	default VPC security group	vpc-cde1f0a4

소유자	인바운드 규칙 수	아웃바운드 규칙 수
636870463736	2 권한 항목	1 권한 항목

## 2) Setting VPC network

- Add new rule
  - Type : PostgreSQL, Source : “사용자 지정” – 0.0.0.0/0

VPC > 보안 그룹 > sg-1ab7f972 - default > 인바운드 규칙 편집

### 인바운드 규칙 편집 정보

인바운드 규칙은 인스턴스에 도달하도록 허용된 수신 트래픽을 제어합니다.

#### 인바운드 규칙 정보

보안 그룹 규칙 ID	유형 <small>정보</small>	프로토콜 <small>정보</small>	포트 범위 <small>정보</small>	소스 <small>정보</small>	설명 - 선택 사항 <small>정보</small>	
sgr-03a8f2d5992c8b386	모든 트래픽 ▼	전체	전체	사용자 ... ▼	<input type="text"/>	<input type="button" value="삭제"/>
sgr-09e53d88b6113d8ac	PostgreSQL ▼	TCP	5432	사용자 ... ▼	<input type="text" value="0.0.0.0/0"/> <input type="button" value="X"/> <input type="text" value="0.0.0.0/0"/> <input type="button" value="X"/>	<input type="button" value="삭제"/>

Save it!

### 3) Install SQLAlchemy and psycopg2-binary

- **Activate Virtual environment and install belows.**

#### **In Windows**

```
$ .\[가상환경이름]\Scripts\activate.bat  
$ pip install SQLAlchemy  
$ pip install psycopg2-binary
```

#### **In Mac**

```
$ source [가상환경이름]/bin/activate  
$ pip install SQLAlchemy  
$ pip install psycopg2-binary
```

#### **In Linux ( include WSL)**

```
$ source [가상환경이름]/bin/activate  
$ pip install SQLAlchemy  
$ pip install psycopg2-binary
```



### 3) Install SQLAlchemy and postgresQL

- Make URI based on your RDS setting!

**설정**

**DB 인스턴스 식별자** 필수  
DB 인스턴스 이름을 입력하세요. 이름은 현재 AWS 리전에서 AWS 계정에 소유하는 모든 DB 인스턴스에 대해 고유해야 합니다.  
map-database

**자격 증명 설정**  
마스터 사용자 이름 필수  
DB 인스턴스의 마스터 사용자에 로그인 ID를 입력하세요.  
postgres

☐ 암호 자동 생성  
Amazon RDS에서 사용자에 대신하여 암호를 생성하거나 사용자가 직접 암호를 지정할 수 있습니다.

**마스터 암호** 필수  
마스터 암호는 8자 이상이어야 하며, 다음을 포함할 수 없습니다. / (슬래시), (특수문자), (문자) 및 @ (앳 기호).  
\*\*\*\*\*

**암호 확인** 필수  
\*\*\*\*\*

**인스턴스 구성**  
아래의 DB 인스턴스 구성 옵션은 위에서 선택한 엔진에서 지원하는 옵션으로 제한됩니다.

**DB 인스턴스 클래스** 필수  
● 스탠다드 클래스(m 클래스 포함)  
● 메모리 최적화 클래스(r 및 x 클래스 포함)  
● 버스터블 클래스(t 클래스 포함)  
db.t3.micro

2 vCPU, 1 GiB RAM, db.t3.micro

☐ 이전 세대 클래스 포함

**Amazon RDS**

**DB 인스턴스 database-17가 성공적으로 삭제되었습니다.**

**RDS > 데이터베이스 > map-database**

**map-database**

**요약**

DB 식별자 map-database	CPU 6.15%	상태 사용 가능	클래스 db.t3.micro
역할 인스턴스	현재 활동 0.00 sessions	엔진 PostgreSQL	리전 및 AZ ap-northeast-2c

**연결 & 보안**

**엔드포인트 및 포트**

엔드포인트 map-database.cqpvfmj4sdrap-northeast-2-rds.amazonaws.com	네트워킹 가용 영역 ap-northeast-2c VPC vpc-cde1f0a4 서브넷 그룹 default-vpc-cde1f0a4 서브넷 subnet-3563815d subnet-255a6779 subnet-8ff2ab2c subnet-9a21e7e1 네트워크 유형 IPv4	보안 VPC 보안 그룹 default (sg-1ab7f972) 퍼블릭 액세스 가능 예 인증 기관 rds-ca-2019 인증 기관 날짜 August 23, 2024, 02:08 (UTC+2:08)
---	---	--

URI =>

postgresql://{USER}:{PASSWORD}@{ENDPOINT URL}:{PORT}/{DB\_NAME}

Default DB\_NAME is "postgres"

### 3) Install SQLAlchemy and postgresQL

- **Connect to your RDS instance using below code!**

```
from sqlalchemy import create_engine
from sqlalchemy.orm import scoped_session, sessionmaker
from sqlalchemy.ext.declarative import declarative_base

USER = "postgres"
PW = " "
URL = "map-database :ap-northeast-2.rds.amazonaws.com"
PORT = "5432"
DB = "postgres"

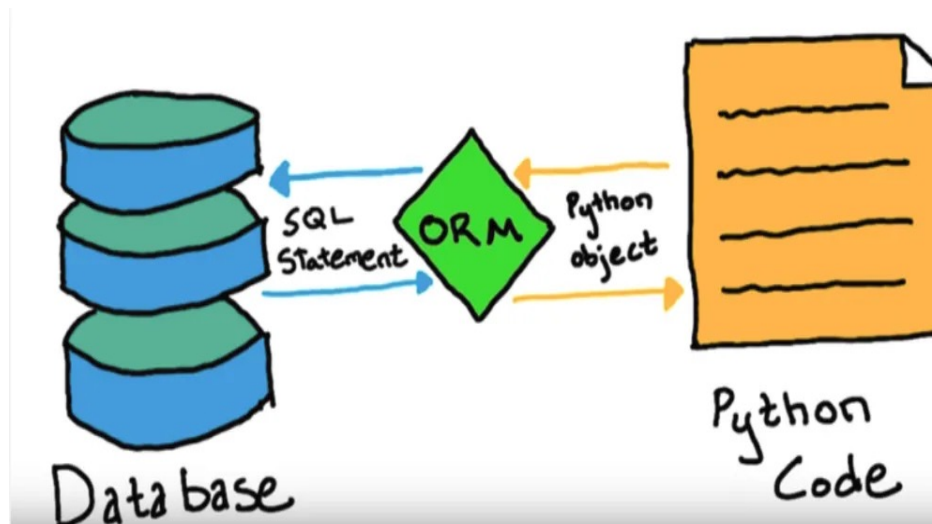
engine = create_engine("postgresql://{user}:{pw}@{url}:{port}/{db}".format(USER, PW, URL, PORT, DB))
db_session = scoped_session(sessionmaker(autocommit=False, autoflush=False, bind=engine))

Base = declarative_base()
Base.query = db_session.query_property()

Base.metadata.create_all(bind=engine)
```

## 4) Create Engine, Session, Base

- **SQLAlchemy is ORM(Object Relation Mapping) library.**
  - SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.
  - It provides a full suite of well-known enterprise-level persistence patterns, designed for efficient and high-performing database access.



## 4) Create Engine, Session, Base

### server.py (RDS part)

```
from sqlalchemy import create_engine
from sqlalchemy.orm import scoped_session, sessionmaker
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String → Added!
```

```
USER = "postgres"
PW =  
URL = "map-database- ap-northeast-2.rds.amazonaws.com"
PORT = "5432"
DB = "postgres"
```

```
engine = create_engine("postgresql://{}:{}_@{}/{}".format(USER, PW, URL, PORT, DB))
db_session = scoped_session(sessionmaker(autocommit=False, autoflush=False, bind=engine))
```

```
Base = declarative_base()
Base.query = db_session.query_property()
```

```
class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String(50), unique=True)
    passwd = Column(String(120), unique=False)
```

```
def __init__(self, name=None, passwd=None):
    self.name = name
    self.passwd = passwd
```

```
def __repr__(self):
    return f'<User {self.name!r}>'
```

```
# Base.metadata.drop_all(bind=engine)
Base.metadata.create_all(bind=engine)
```

**1) Connect to our DB instance**

**2) Make session to communicate with RDS instance!**

**3) Declare Base class (A base class stores a catalog of classes and mapped tables in the system)**

**4) Declare Data Class!**

**5) Create all tables (drop\_all( ) method delete all tables)**

# 5) Send add transaction and query transaction

## server.py (Flask part)

... Write below of RDS part code ...

```
from flask import Flask
from flask import request
from flask import jsonify
from werkzeug.serving import WSGIRequestHandler
import json
```

```
import json
WSGIRequestHandler.protocol_version = "HTTP/1.1"
```

```
app = Flask(__name__)
```

```
@app.route("/adduser", methods=['POST'])
```

```
def add_user():
```

```
    content = request.get_json(silent=True)
```

```
    name = content["name"]
```

```
    passwd = content["passwd"]
```

```
    if db_session.query(User).filter_by(name=name).first() is None:
```

```
        u = User(name=name, passwd=passwd)
```

```
        db_session.add(u)
```

```
        db_session.commit()
```

```
        return jsonify(success=True)
```

```
    else:
```

```
        return jsonify(success=False)
```

```
if __name__ == "__main__":
```

```
    app.run(host='localhost', port=8888)
```

Send query in **"User"** table,  
Filter with **"name=name"** condition

Make new ROW(data) and send add transaction  
using **{session}.add( )** method

You must call **{session}.commit( )** method  
after send transaction

## 5) Send add transaction and query transaction

### server.py ( Flask part )

... Add new routing method below of before codes ...

```
@app.route("/login", methods=['POST'])
def login():
    content = request.get_json(silent=True)
    name = content["name"]
    passwd = content["passwd"]

    check = False
    result = db_session.query(User).all()
    for i in result:
        if i.name == name and i.passwd == passwd:
            check = True
    return jsonify(success=check)
```

Send query in **"User"** table, to get all ROWs of table.  
all( ) method return list of all satisfied ROWs.

Check data attributes of each ROW.

# 5) Send add transaction and query transaction

If you want to know about updating ROW, deleting ROW, or JOIN operation, Please check below links “SQLAlchemy ORM” tab!

[https://www.tutorialspoint.com/sqlalchemy/sqlalchemy\\_orm\\_declaring\\_mapping.htm](https://www.tutorialspoint.com/sqlalchemy/sqlalchemy_orm_declaring_mapping.htm)

**SQLAlchemy ORM - Declaring Mapping**

Advertisements

Rohde & Schwarz  
**What comes beyond 5G?** [OPEN](#)

Previous Page Next Page

The main objective of the Object Relational Mapper API of SQLAlchemy is to facilitate associating user-defined Python classes with database tables, and objects of those classes with rows in their corresponding tables. Changes in states of objects and rows are synchronously matched with each other. SQLAlchemy enables expressing database queries in terms of user defined classes and their defined relationships.

The ORM is constructed on top of the SQL Expression Language. It is a high level and abstracted pattern of usage. In fact, ORM is an applied usage of the Expression Language.

Although a successful application may be constructed using the Object Relational Mapper exclusively, sometimes an application constructed with the ORM may use the Expression Language directly where specific database interactions are required.

### Declare Mapping

First of all, `create_engine()` function is called to set up an engine object which is subsequently used to perform SQL operations. The function has two arguments, one is the name of database and other is an echo parameter when set to True will generate the activity log. If it doesn't exist, the database will be created. In the following example, a SQLite database is created.

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///sales.db', echo = True)
```

The Engine establishes a real DBAPI connection to the database when a method like `Engine.execute()` or `Engine.connect()` is called. It is then used to emit the SQLORM which does not use the Engine directly; instead, it is used behind the scenes by the ORM.

In case of ORM, the configurational process starts by describing the database tables and then by defining classes which will be mapped to those tables. In SQLAlchemy, these two tasks are performed together. This is done by using Declarative system; the classes created include directives to describe the actual database table they are mapped to.

A base class stores a catalog of classes and mapped tables in the Declarative system. This is called as the declarative base class. There will be usually just one instance of this base in a commonly imported module. The `declarative_base()` function is used to create base class. This function is defined in `sqlalchemy.ext.declarative` module.

```
from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()
```

Once base class is declared, any number of mapped classes can be defined in terms of it. Following code defines a Customer's class. It contains the table to be mapped to, and names and datatypes of

# Exercise – Login Function Application

- Using “zappa deploy dev”, upload your flask server to AWS Lambda!
- Send Screenshot of your RDS instance page and two Screenshots of reqbin webpage with “adduser” and “login” requests.
- Zip your python code with below screenshots and Submit it on ICAMPUS.

The screenshot shows the Amazon RDS console for the 'map-database' instance. The instance is a DB instance named 'map-database' with a DB instance class of 'db.t3.micro'. It is running PostgreSQL and is in the 'available' state. The instance is located in the 'us-east-2' region, 'us-east-2a' Availability Zone, and 'us-east-2' Region. The instance is associated with the 'map-database' DB subnet group and the 'map-database' VPC. The instance is associated with the 'map-database' DB security group and the 'map-database' VPC security group. The instance is associated with the 'map-database' DB parameter group and the 'map-database' VPC parameter group. The instance is associated with the 'map-database' DB snapshot and the 'map-database' VPC snapshot.

The first screenshot shows a reqbin tool interface with a POST request to 'https://api75tstd.execute-api.us-east-2.amazonaws.com/dev/adduser'. The request body is a JSON object: {"name": "shjeong", "password": "test1234"}. The response status is 200 (OK) with a time of 850 ms and a size of 0.02 kb.

The second screenshot shows a reqbin tool interface with a POST request to 'https://api75tstd.execute-api.us-east-2.amazonaws.com/dev/login'. The request body is a JSON object: {"name": "shjeong", "password": "test1234"}. The response status is 200 (OK) with a time of 834 ms and a size of 0.03 kb.