

Programming of RDS using Lambda

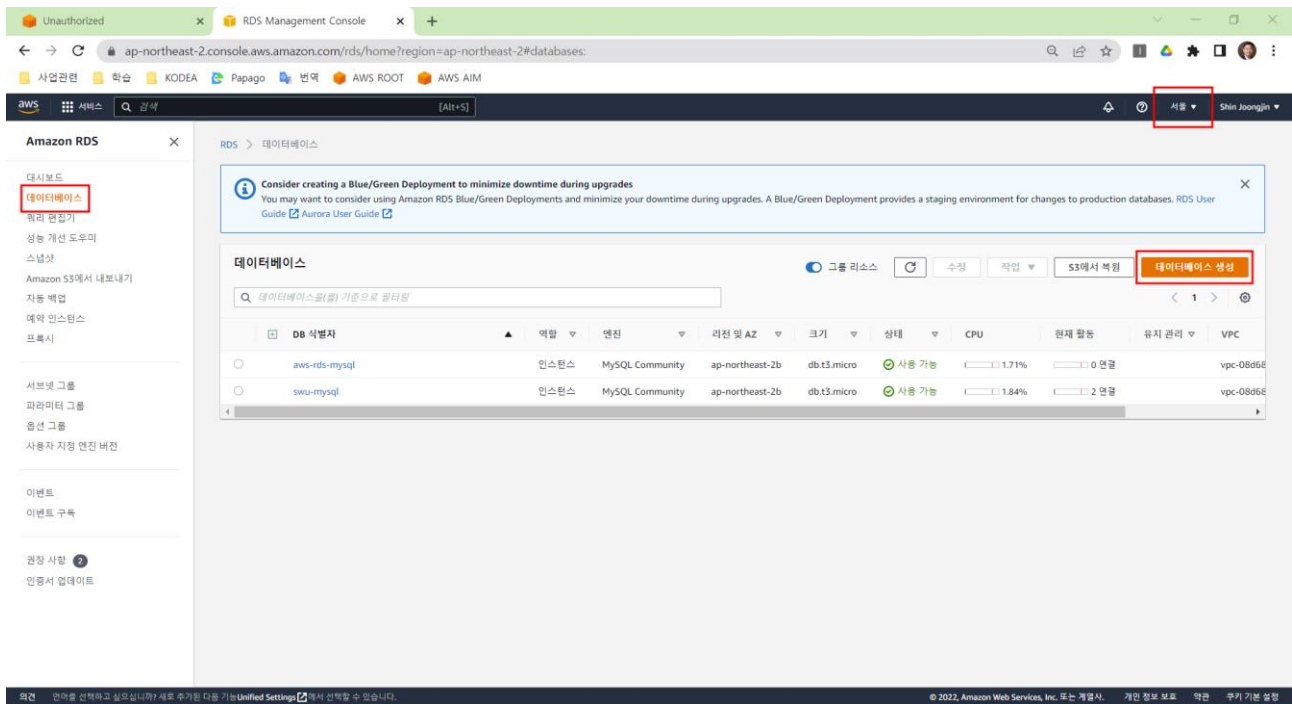
1. AWS RDS-MySQL 생성 및 MySQL Workbench로 작업하기
2. Python용 Pymysql 패키지 Layer 추가
3. mysql에 연결하기
4. SELECT문으로 데이터 가져오기
5. insert / update / delete문 사용하기

1. AWS RDS-MySQL 생성 및 MySQL Workbench로 작업하기

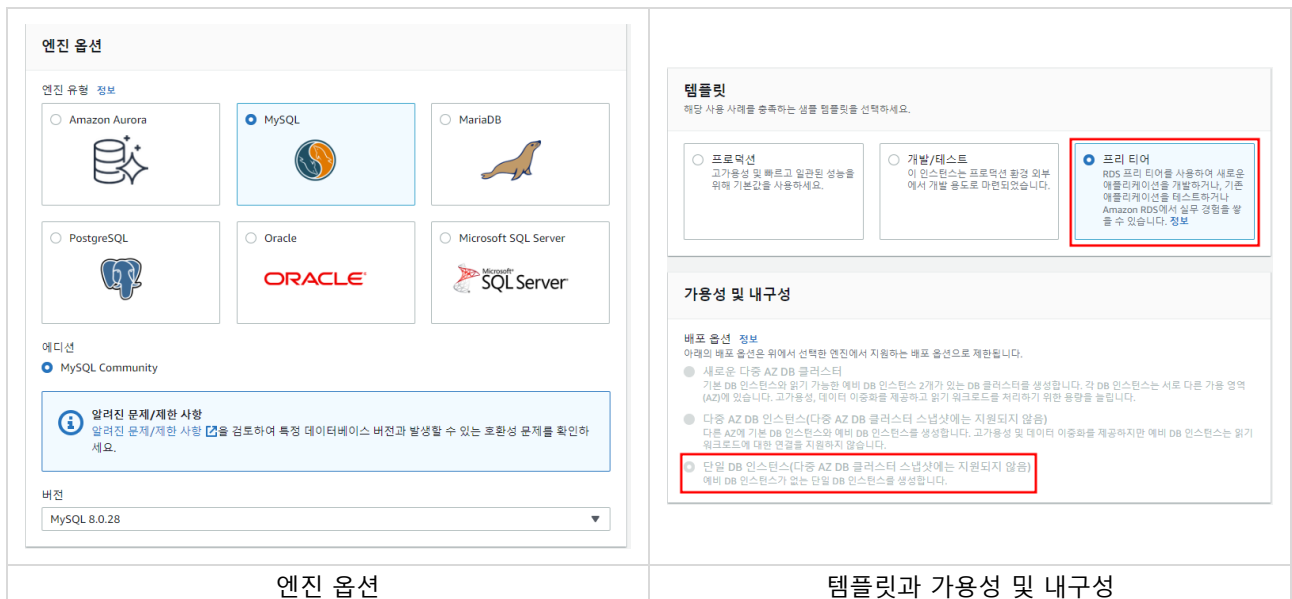
1) AWS RDS Service – MySQL 생성하기

① AWS RDS → 데이터베이스 → “데이터베이스 생성” 버튼 클릭

단, 이때 주의 해야할 점은 간혹 리전이 변경되는 경우가 있으니 “서울”리전이 맞는지 확인 후 “데이터베이스 생성” 버튼을 클릭해야 함.



② 엔진 옵션, 템플릿, 가용성 및 내구성



③ 설정 및 인스턴스 구성

설정	인스턴스 구성
<p>설정</p> <p>DB 인스턴스 식별자 정보 DB 인스턴스 이름을 입력하세요. 이름은 현재 AWS 리전에서 AWS 계정이 소유하는 모든 DB 인스턴스에 대해 고유해야 합니다.</p> <p>swu-mysql-2</p> <p>DB 인스턴스 식별자는 대소문자를 구분하지 않지만 'mydbinstance'와 같이 모두 소문자로 저장됩니다. 제약: 1~60자의 영숫자 또는 하이픈으로 구성되어야 합니다. 첫 번째 문자는 글자여야 합니다. 하이픈 2개가 연속될 수 없습니다. 하이픈으로 끝날 수 없습니다.</p> <p>▼ 자격 증명 설정</p> <p>마스터 사용자 이름 정보 DB 인스턴스의 마스터 사용자에게 로그인 ID를 입력하세요.</p> <p>admin</p> <p>1~16자의 영숫자, 첫 번째 문자는 글자여야 합니다.</p> <p><input type="checkbox"/> 암호 자동 생성 Amazon RDS에서 사용자를 대신하여 암호를 생성하거나 사용자가 직접 암호를 지정할 수 있습니다.</p> <p>마스터 암호 정보</p> <p>*****</p> <p>대략 32-64자 이상의 인쇄 가능한 ASCII 문자. 다음은 포함될 수 없습니다: / (슬래시), (여백바꿈표), (큰따옴표) 및 @ (앳 기호).</p> <p>암호 확인 정보</p> <p>*****</p>	<p>인스턴스 구성 아래의 DB 인스턴스 구성 옵션은 위에서 선택한 엔진에서 지원하는 옵션으로 제한됩니다.</p> <p>DB 인스턴스 클래스 정보</p> <ul style="list-style-type: none"> ● 스탠다드 클래스(m 클래스 포함) ● 메모리 최적화 클래스(r 및 x 클래스 포함) ● 버스팅 클래스(t 클래스 포함) <p>db.t3.micro 2 vCPUs 1 GiB RAM 네트워크: 2,085Mbps</p> <p><input type="radio"/> 이전 세대 클래스 포함</p>
설정	인스턴스 구성

④ 스토리지

스토리지

스토리지 유형 정보

범용 SSD(gp2)
볼륨 크기에 따라 기준 성능 결정

할당된 스토리지

200 GiB

최솟값은 20GiB이고, 최댓값은 6,144GiB입니다.

스토리지 자동 조정 정보
애플리케이션의 필요에 따라 데이터베이스 스토리지의 동적 조정 지원을 제공합니다.

☒ **스토리지 자동 조정 활성화**
이 기능을 활성화하면 지정한 임계값 초과 후 스토리지를 늘릴 수 있습니다.

최대 스토리지 임계값 정보
데이터베이스를 지정한 임계값으로 자동 조정하면 요금이 부과됩니다.

1000 GiB

최솟값은 220GiB이고, 최댓값은 6,144GiB입니다.

⑤ 연결

연결 정보

컴퓨팅 리소스
이 데이터베이스의 컴퓨팅 리소스에 대한 연결을 설정할지 여부를 선택합니다. 연결을 설정하면 컴퓨팅 리소스가 이 데이터베이스에 연결할 수 있도록 연결 설정이 자동으로 변경됩니다.

☒ **EC2 컴퓨팅 리소스에 연결 안 함**
이 데이터베이스의 컴퓨팅 리소스에 대한 연결을 설정하지 않습니다. 나중에 컴퓨팅 리소스에 대한 연결을 수동으로 설정할 수 있습니다.

☐ **EC2 컴퓨팅 리소스에 연결**
이 데이터베이스의 EC2 컴퓨팅 리소스에 대한 연결을 설정합니다.

Virtual Private Cloud(VPC) 정보
VPC를 선택합니다. VPC는 이 DB 인스턴스의 가상 네트워크 환경을 정의합니다.

Default VPC (vpc-08d6828002eb3fa3a)

해당 DB 서브넷 그룹이 있는 VPC만 나열됩니다.

☒ **데이터베이스를 생성한 후에는 VPC를 변경할 수 없습니다.**

DB 서브넷 그룹 정보
DB 서브넷 그룹을 선택합니다. DB 서브넷 그룹은 선택한 VPC에서 DB 인스턴스가 어떤 서브넷과 IP 범위를 사용할 수 있는지를 정의합니다.

default-vpc-08d6828002eb3fa3a

퍼블릭 액세스 정보

☒ **예**
RDS는 데이터베이스에 퍼블릭 IP 주소를 할당합니다. VPC 외부의 Amazon EC2 인스턴스 및 다른 리소스가 데이터베이스에 연결할 수 있습니다. VPC 내부의 리소스도 데이터베이스에 연결할 수 있습니다. 데이터베이스에 연결할 수 있는 리소스를 지정하는 VPC 보안 그룹을 하나 이상 선택합니다.

☐ **아니오**
RDS는 퍼블릭 IP 주소를 데이터베이스에 할당하지 않습니다. VPC 내부의 Amazon EC2 인스턴스 및 다른 리소스만 데이터베이스에 연결할 수 있습니다. 데이터베이스에 연결할 수 있는 리소스를 지정하는 VPC 보안 그룹을 하나 이상 선택합니다.

VPC 보안 그룹(보안 그룹) 정보
데이터베이스에 대한 액세스를 허용할 VPC 보안 그룹을 하나 이상 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
Choose one or more options

default

가용 영역 정보
기본 설정 없음

RDS 프로кси
RDS 프로시는 애플리케이션 확장성, 복원력 및 보안을 개선하는 완전관리형 고가용성 데이터베이스 프로시입니다.

☐ **RDS 프로кси 생성 정보**
RDS는 프로시에 대한 IAM 역할과 Secrets Manager 보안 암호를 자동으로 생성합니다. RDS 프로시에 대한 추가 비용이 있습니다. 자세한 내용은 다음을 참조하세요. [Amazon RDS 프로시 요금](#)

▶ 추가 구성

⑥ 기타 : 이후 변경할 사항은 없으며 “데이터베이스 생성” 버튼 클릭을 통해 마무리 한다.

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 허용하는 디렉터리를 선택합니다.

모니터링

모니터링

☐ **Enhanced 모니터링 활성화**
Enhanced 모니터링 지표를 활성화하면 다른 프로세스 또는 스레드에서 CPU를 사용하는 방법을 확인하려는 경우에 유용합니다.

▶ 추가 구성

데이터베이스 옵션, 암호화 커밋, 백업 커밋, 복구 커밋, 유지 관리, CloudWatch Logs, 삭제 방지 커밋.

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, db.t3.micro 또는 db.t4g.micro 인스턴스에서 Amazon RDS를 750시간 사용.
- 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 중형 서비스 요금이 적용됩니다.

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

취소

데이터베이스 생성

데이터베이스 인증과 모니터링
데이터베이스 생성

⑦ 생성 중 화면 : 아직 완료가 되지 않아 사용할 수 없음

swu-mysql-2 데이터베이스 생성 중

데이터베이스를 시작하는 데 몇 분 정도 걸릴 수 있습니다.

[자격 증명 세부 정보 보기](#)

RDS > 데이터베이스

Consider creating a Blue/Green Deployment to minimize downtime during upgrades

You may want to consider using Amazon RDS Blue/Green Deployments and minimize your downtime during upgrades. A Blue/Green Deployment provides a staging environment for changes to production databases. [RDS User Guide](#)

[Guide](#) [Aurora User Guide](#)

데이터베이스

☒ 그룹 리소스

수정

작업

S3에서 복원

데이터베이스 생성

데이터베이스를(들) 기준으로 필터링

	DB 식별자	역할	엔진	리전 및 AZ	크기	상태	CPU	현재 활동	유지 관리	VPC
<input type="radio"/>	aws-rds-mysql	인스턴스	MySQL Community	ap-northeast-2b	db.t3.micro	사용 가능	1.78%	0 연결		vpc-08d68
<input type="radio"/>	swu-mysql	인스턴스	MySQL Community	ap-northeast-2b	db.t3.micro	사용 가능	2.26%	2 연결		vpc-08d68
<input type="radio"/>	swu-mysql-2	인스턴스	MySQL Community	-	db.t3.micro	생성 중	-			vpc-08d68

⑧ 생성 완료 상태 : 상태값이 "사용 가능"으로 표현되어야 이후부터 사용이 가능

swu-mysql-2 데이터베이스 생성에 성공했습니다. 연결 세부 정보 보기

RDS > 데이터베이스

Consider creating a Blue/Green Deployment to minimize downtime during upgrades
You may want to consider using Amazon RDS Blue/Green Deployments and minimize your downtime during upgrades. A Blue/Green Deployment provides a staging environment for changes to production databases. [RDS User Guide](#) [Aurora User Guide](#)

데이터베이스 그룹 리소스 수정 작업 S3에서 복원 데이터베이스 생성

데이터베이스(들) 기준으로 필터링

DB 식별자	역할	엔진	리전 및 AZ	크기	상태	CPU	현재 활동	유지 관리	VPC
aws-rds-mysql	인스턴스	MySQL Community	ap-northeast-2b	db.t3.micro	사용 가능	2.02%	0 연결		vpc-08d68
swu-mysql	인스턴스	MySQL Community	ap-northeast-2b	db.t3.micro	삭제 중	1.73%	2 연결		vpc-08d68
swu-mysql-2	인스턴스	MySQL Community	ap-northeast-2c	db.t3.micro	사용 가능	4.78%	0 연결		vpc-08d68

2) MySQL Workbench로 작업하기

① DB 식별자(이름) 클릭

데이터베이스 그룹 리소스 수정 작업 S3에서 복원 데이터베이스 생성

데이터베이스(들) 기준으로 필터링

DB 식별자	역할	엔진	리전 및 AZ	크기	상태	CPU	현재 활동	유지 관리	VPC
aws-rds-mysql	인스턴스	MySQL Community	ap-northeast-2b	db.t3.micro	사용 가능	1.98%	0 연결		vpc-08d68
swu-mysql-2	인스턴스	MySQL Community	ap-northeast-2c	db.t3.micro	사용 가능	2.41%	0 연결		vpc-08d68

② 데이터베이스 End Point 주소 복사

swu-mysql-2 수정 작업

요약

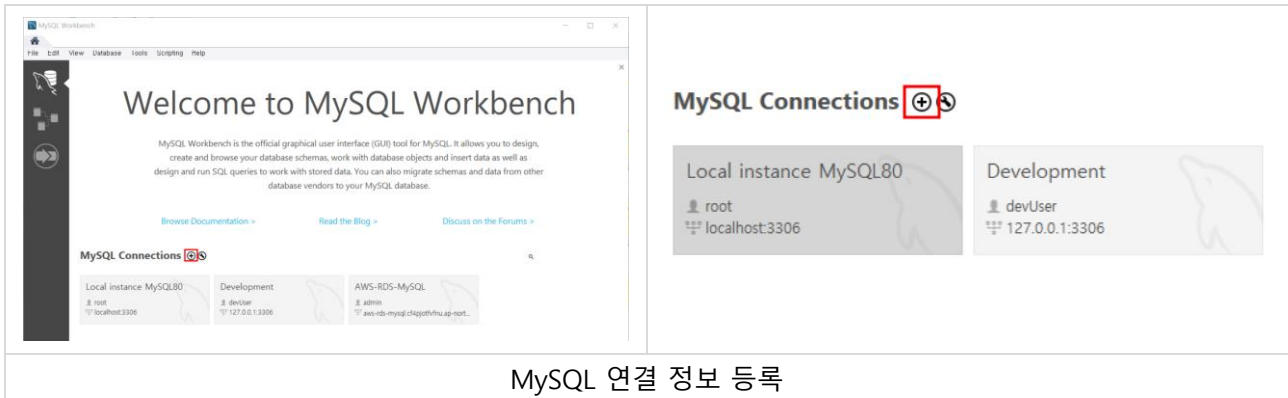
DB 식별자 swu-mysql-2	CPU 1.82%	상태 사용 가능	클래스 db.t3.micro
역할 인스턴스	현재 활동 0 연결	엔진 MySQL Community	리전 및 AZ ap-northeast-2c

연결 & 보안 | 모니터링 | 로그 및 이벤트 | 구성 | 유지 관리 및 백업 | 태그

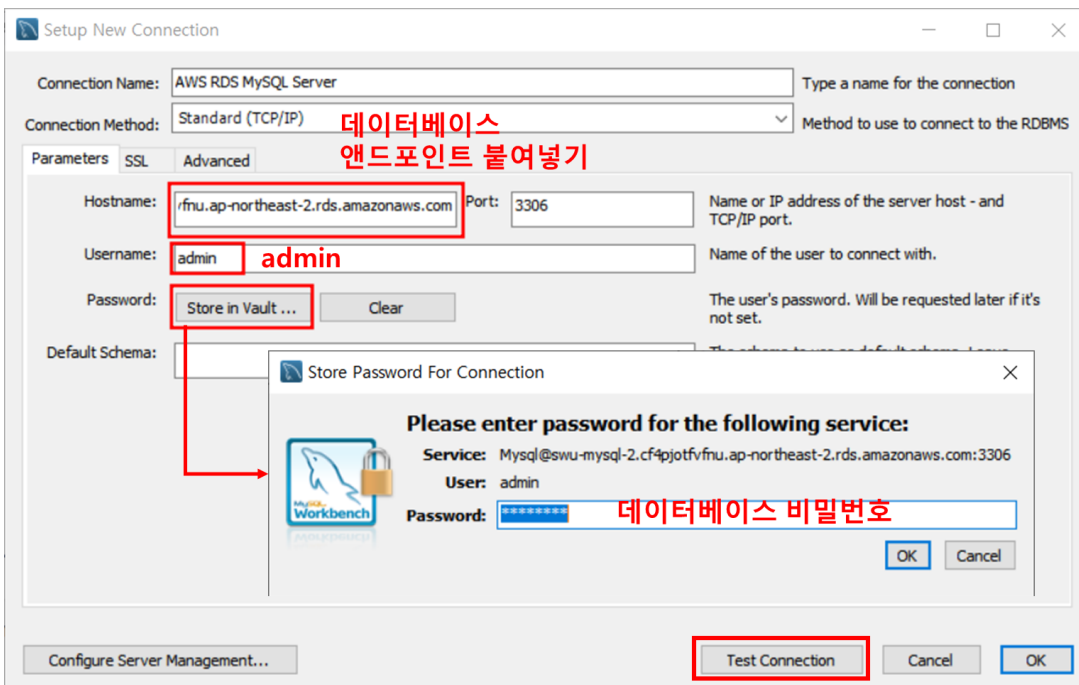
연결 & 보안

엔드포인트 및 포트	네트워킹	보안
엔드포인트 swu-mysql-2.cf4pjotvfmw.ap-northeast-2.rds.amazonaws.com	가용 영역 ap-northeast-2c	VPC 보안 그룹 default (sg-0cc08c7f09a18380)
포트 3306	VPC vpc-08d6828002eb3fa3a	퍼블릭 액세스 가능 예
	서브넷 그룹 default-vpc-08d6828002eb3fa3a	인증 기관 정보

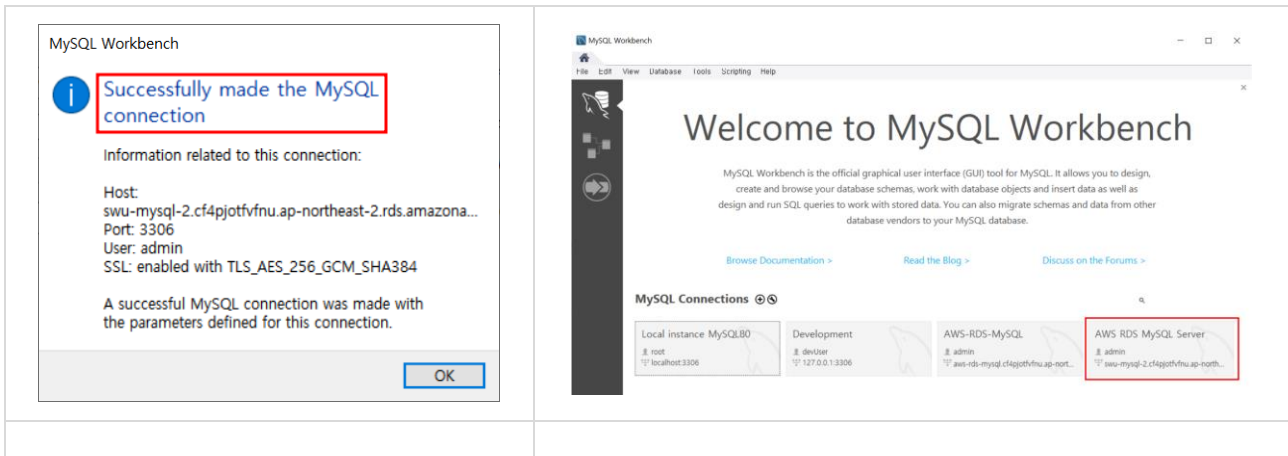
③ Workbench 실행 → MySQL 연결 정보 등록



④ 데이터베이스 연결 설정



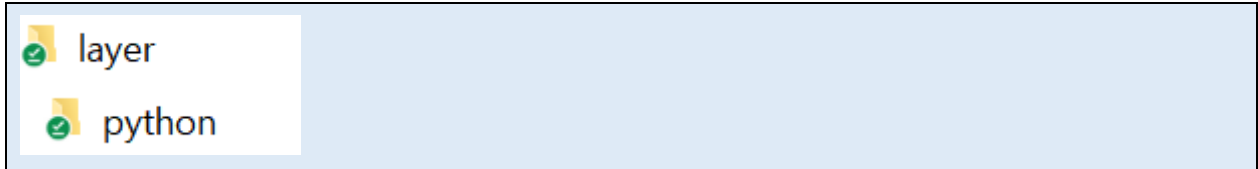
⑤ Test Connection 버튼을 통한 연결 확인 → "OK"버튼 클릭



2. pymysql 패키지 레이어에 추가하기

1) pymysql 다운로드 및 설치

① 아래와 같은 구조로 폴더를 생성한다.



② 관리자 권한으로 "명령 프롬프트"를 실행합니다.

③ CD 명령어를 사용하여 "python"폴더로 이동합니다.

```

관리자: 명령 프롬프트
Microsoft Windows [Version 10.0.19044.2364]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd #
C:\>cd googledrive
C:\GoogleDrive>cd 01_study
C:\GoogleDrive\01_STUDY>cd 04_cloud
C:\GoogleDrive\01_STUDY\04_Cloud>cd 0402_developing on aws
C:\GoogleDrive\01_STUDY\04_Cloud\0402_Developing On AWS>cd layer
C:\GoogleDrive\01_STUDY\04_Cloud\0402_Developing On AWS\layer>cd python
C:\GoogleDrive\01_STUDY\04_Cloud\0402_Developing On AWS\layer\python>
  
```

③ 파이썬용 pymysql 설치 및 다운로드.

```
C:\W...W...Wpython>pip install pymysql
```

④ 설치가 잘 되었는지 확인

```
C:\W...W...Wpython>pip list
```

⑤ 설치 및 다운로드를 통해 디렉토리(폴더)가 생성되었는지 확인

```
C:\W...W...Wpython>dir/w
```

```

c:\#GoogleDrive#01_STUDY#04_Cloud#0402_Developing On AWS#layer#python>pip install pymysql
Collecting pymysql
  Using cached PyMySQL-1.0.2-py3-none-any.whl (43 kB)
Installing collected packages: pymysql
Successfully installed pymysql-1.0.2

c:\#GoogleDrive#01_STUDY#04_Cloud#0402_Developing On AWS#layer#python>pip list
Package Version
-----
pip      22.3.1
PyMySQL 1.0.2
setuptools 65.5.0

c:\#GoogleDrive#01_STUDY#04_Cloud#0402_Developing On AWS#layer#python>dir/w
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 082D-12E3

c:\#GoogleDrive#01_STUDY#04_Cloud#0402_Developing On AWS#layer#python 디렉터리
[.]
                0개 파일                0 바이트 [pymysql] [PyMySQL-1.0.2.dist-info]
                4개 디렉터리 132,327,088,128 바이트 남음

c:\#GoogleDrive#01_STUDY#04_Cloud#0402_Developing On AWS#layer#python>
  
```

2) 라이브러리 압축

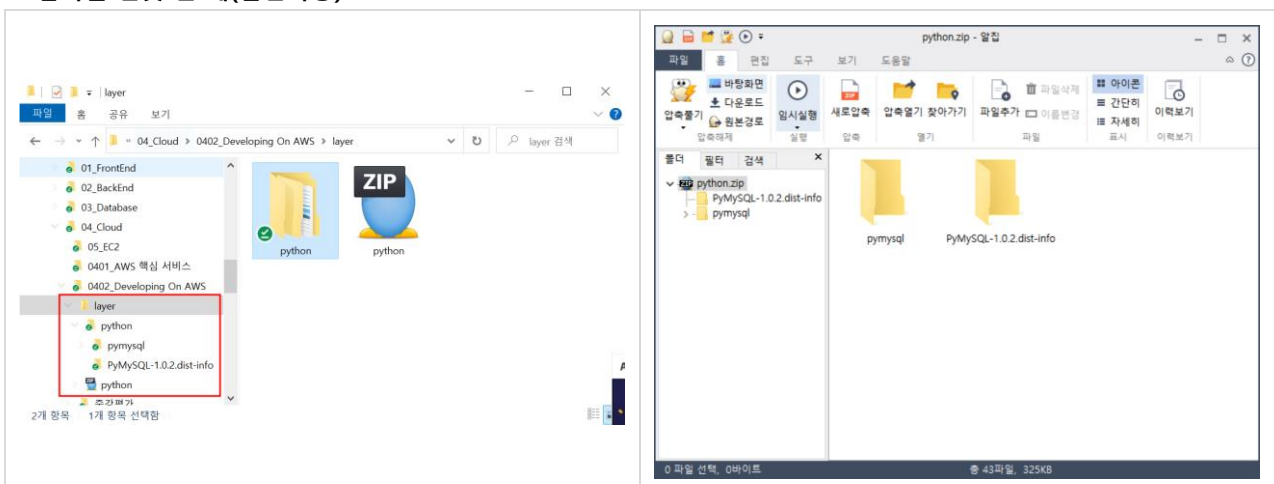
이 부분은 아래 이미지를 보고 동일한 형태로 압축이 되었는지 반드시 확인이 필요하다. 반드시 압축파일의 구조가 아래와 같이 생성이 되어있어야 한다.

• 디렉터리(폴더) 구조



① 압축 유틸리티를 사용하여 압축하는 방법

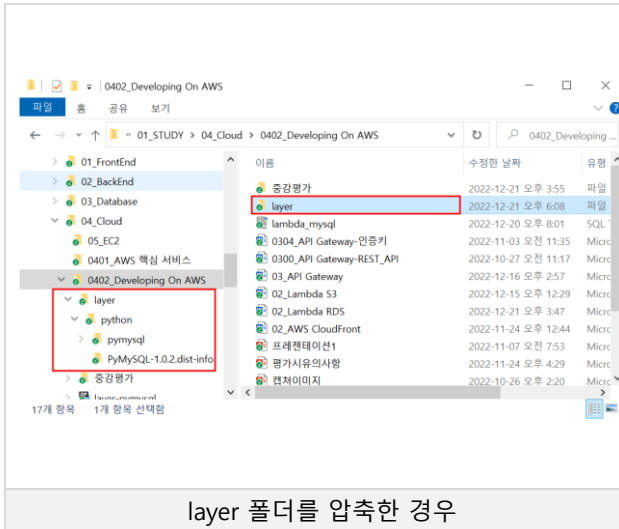
• 압축을 잘못 한 예(알집사용)



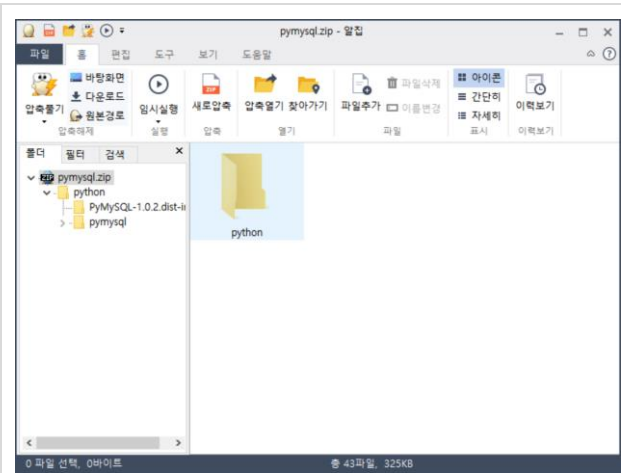
Python 폴더를 압축한 경우

압축파일의 내부구조에 python이 없음

• 올바른 압축 예(알집사용)



layer 폴더를 압축한 경우




압축파일의 내부구조에 python이 있음

② 명령 프롬프트를 사용하여 압축하는 방법

- 아래 사이트를 통해 zip, unzip 파일을 다운받는다.

<http://stahlworks.com/dev/?tool=zipunzip>

StahlWorks Technologies. Software Tools For Professionals.



- 다음 경로에 다운로드 받은 zip.exe 파일과 unzip.exe파일을 복사한다.

C:\windows\system32

- 명령 프롬프트를 실행하고 "cd"명령을 이용하여 python폴더의 부모 폴더로 이동한다. "dir/w"명령을 이용하여 "python" 폴더가 아래 그림과 같이 확인이 되어야 한다.

```

C:\WINDOWS\system32\cmd.exe

c:\GoogleDrive\01_STUDY\04_Cloud\0402_Developing On AWS\layer>cd #
c:\>cd c:\googledrive
c:\GoogleDrive>cd 01_study
c:\GoogleDrive\01_STUDY>cd 04_cloud
c:\GoogleDrive\01_STUDY\04_Cloud>cd 0402_Developing on aws
c:\GoogleDrive\01_STUDY\04_Cloud\0402_Developing On AWS>cd layer
c:\GoogleDrive\01_STUDY\04_Cloud\0402_Developing On AWS\layer>dir/w
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 082D-12E3

c:\GoogleDrive\01_STUDY\04_Cloud\0402_Developing On AWS\layer 디렉터리
.          ..          [python]
0개 파일              0 바이트
3개 디렉터리 132,248,911,872 바이트 남음

c:\GoogleDrive\01_STUDY\04_Cloud\0402_Developing On AWS\layer>
  
```

- zip명령을 이용하여 "python"폴더를 압축한다.

zip [option] [압축결과파일명] [압축대상]

[option]

압축률 : 1~9, 높을수록 압축률이 좋음

v : 압축되는 진행상황을 보여줌

m : 압축 대상 파일들을 압축파일로 이동(거의 사용 안됨)

r : 하위 디렉토리(폴더)의 파일들까지 모두 묶어 압축

Zip -9vr pymysql.zip ./python

```

C:\WINDOWS\system32\cmd.exe

c:\GoogleDrive\01_STUDY\04_Cloud\0402_Developing On AWS\layer>zip -9vr pymysql.zip ./python
adding: python/ (192 bytes security) (in=0) (out=0) (stored 0%)
adding: python/pymysql/ (164 bytes security) (in=0) (out=0) (stored 0%)
adding: python/pymysql/charset.py (164 bytes security) (in=10293) (out=1778) (deflated 83%)
adding: python/pymysql/connections.py (164 bytes security) (in=51251) (out=12512) (deflated 76%)
adding: python/pymysql/constants/ (164 bytes security) (in=0) (out=0) (stored 0%)
adding: python/pymysql/constants/CLIENT.py (164 bytes security) (in=878) (out=430) (deflated 51%)
adding: python/pymysql/constants/COMMAND.py (164 bytes security) (in=679) (out=292) (deflated 57%)
adding: python/pymysql/constants/CR.py (164 bytes security) (in=1927) (out=695) (deflated 64%)
adding: python/pymysql/constants/ER.py (164 bytes security) (in=12296) (out=4595) (deflated 63%)
adding: python/pymysql/constants/FIELD_TYPE.py (164 bytes security) (in=370) (out=204) (deflated 45%)
adding: python/pymysql/constants/FLAG.py (164 bytes security) (in=214) (out=147) (deflated 31%)
adding: python/pymysql/constants/SERVER_STATUS.py (164 bytes security) (in=333) (out=165) (deflated 50%)
adding: python/pymysql/constants/_init_.py (172 bytes security) (in=0) (out=0) (stored 0%)
adding: python/pymysql/constants/_pycache_/ (164 bytes security) (in=0) (out=0) (stored 0%)
adding: python/pymysql/constants/_pycache_/CLIENT.cpython-311.pyc (164 bytes security) (in=1041) (out=764) (deflated 27%)
adding: python/pymysql/constants/_pycache_/COMMAND.cpython-311.pyc (164 bytes security) (in=1136) (out=708) (deflated 38%)
adding: python/pymysql/constants/_pycache_/CR.cpython-311.pyc (164 bytes security) (in=2646) (out=1365) (deflated 48%)
adding: python/pymysql/constants/_pycache_/ER.cpython-311.pyc (164 bytes security) (in=17369) (out=7899) (deflated 55%)
adding: python/pymysql/constants/_pycache_/FIELD_TYPE.cpython-311.pyc (164 bytes security) (in=850) (out=596) (deflated 30%)
adding: python/pymysql/constants/_pycache_/FLAG.cpython-311.pyc (164 bytes security) (in=542) (out=434) (deflated 20%)
adding: python/pymysql/constants/_pycache_/SERVER_STATUS.cpython-311.pyc (164 bytes security) (in=642) (out=447) (deflated 30%)
adding: python/pymysql/constants/_pycache_/__init__.cpython-311.pyc (164 bytes security) (in=203) (out=167) (deflated 18%)
  
```

3) 계층 생성

AWS 콘솔 로그인 → AWS Lambda → 추가 리소스 → 계층 → 계층 생성

계층 구성

이름

설명 - 선택 사항

☒ .zip 파일 업로드

☐ Amazon S3에서 파일 업로드

pymysql.zip (130.8kB)

10MB가 넘는 파일의 경우, Amazon S3를 사용한 업로드를 고려하십시오.

호환 아키텍처 - 선택 사항 정보

계층에 대해 호환 명령 세트 아키텍처를 선택합니다.

☒ x86_64

☐ arm64

호환 런타임 - 선택 사항 정보

최대 15개의 런타임을 선택합니다.

실행 시간

Python 3.9 X

라이선스 - 선택 사항 정보

취소

생성

4) Lambda 함수에 계층 추가

AWS 콘솔 로그인 → AWS Lambda → 함수 → 함수 선택 → 페이지 하단 계층 → [Add a layer]

계층 추가

함수 런타임 설정

런타임
Python 3.9

아키텍처
x86_64

계층 선택

계층 소스 정보

호환 런타임 및 명령 세트 아키텍처가 있는 계층에서 선택하거나 계층 버전의 Amazon 리소스 이름(ARN)을 지정합니다. 새로운 계층을 생성할 수도 있습니다.

☐ AWS 계층

AWS에서 제공하는 계층 목록에서 계층을 선택합니다.

☒ 사용자 지정 계층

AWS 계정 또는 조직에서 생성한 계층 목록에서 계층을 선택합니다.

☐ ARN 지정

ARN을 제공하여 계층을 지정합니다.

사용자 지정 계층

함수가 런타임과 호환되는 AWS 계정 또는 조직 생성 계층입니다.

lambda-layer-python39-pymysql

버전

1

취소

추가

▼ 함수 개요 정보



ex-rds-mysql-connect



Layers

(1)

+ 트리거 추가

+ 대상 추가

3. pymysql 주요 객체 및 메서드

➔ 깃허브에 업로드한 "PyMySQL-0.7_Documentation.pdf"참고

4. SELECT문을 이용한 데이터 추출

1) 절차

데이터베이스 연결 → 커서 생성 → 쿼리 실행 → Fetch → 연결 종료

2) 기본 Lambda Function

함수명 : ex-rds-mysql-select

```
import json
import pymysql

def lambda_handler(event, context):
    conn = pymysql.connect(
        host="aws-rds-mysql. xxxxxxxxxxxx.ap-northeast-2.rds.amazonaws.com",
        user="admin", password="XXXX", database=" db_name", port=3306, charset="utf8"
    )
    cursor = conn.cursor(pymysql.cursors.DictCursor)
    cursor.execute("select fid, fpass, fname, femail, fphone, faddr1, faddr2, fbirthday, fgender, fdate
from TMEMBER")
    rows=cursor.fetchall()

    arr_rows = []
    for row in rows:
        json_row = {}
        for key in row:
            json_row[key]=row[key]
        arr_rows.append(json_row)
        print(json.dumps(json_row, default=str, indent=2))

    conn.close()
    return {
        'statusCode': 200,
        "body": json.dumps(arr_rows, default=str, indent=2)
    }
```

3) 코드 주요 내용

- **pymysql.cursors.DictCursor**

컬럼 인덱스가 아닌 컬럼 이름으로 데이터 추출하기 위한 옵션

- **json.dumps(arr_rows, default=str, indent=2)**

Json에 삽입된 데이터 중 날짜 형식의 데이터가 있을 경우 문자열로 변환이 되지 않는다. 이를 허용해주기 위한 옵션으로 default=str, indent=2 를 추가해주면 된다.

5. INSERT문을 이용한 데이터 추가

1) 절차

데이터베이스 연결 → 커서 생성 → 쿼리 실행 → 커밋 → 연결 종료

2) 기본 Lambda Function

함수명 : ex-rds-mysql-insert

```
import json
import pymysql

def lambda_handler(event, context):
    conn = pymysql.connect(
        host="aws-rds-mysql. xxxxxxxxxxxx.ap-northeast-2.rds.amazonaws.com",
        user="admin", password="XXXX", database=" db_name", port=3306, charset="utf8"
    )
    # 쿼리를 실행할 커서 생성
    curs = conn.cursor()
    sql = """INSERT INTO TMEMBER VALUE (%s, %s, %s, %s, %s, %s, %s, %s,%s, now())"""
    result = curs.execute(sql,(
        "abc2222", "1111", "최영", "cy@naver.com", "010-4444-4444",
        "경기도 화성시 와우리", "수원대학교", "1991-01-01", "남"
    ))
    conn.commit()
    conn.close()
    return {
        'statusCode': 200,
        'body': json.dumps(result)
    }
```

3) 응용 Lambda Function

위 함수를 API Gateway를 통해 데이터를 입력 받아 실행 할 수 있도록 수정

6. UPDATE문을 이용한 데이터 수정

1) 절차

데이터베이스 연결 → 커서 생성 → 쿼리 실행 → 커밋 → 연결 종료

2) 기본 Lambda Function

함수명 : ex-rds-mysql-update

```
import json
import pymysql
def lambda_handler(event, context):
    conn = pymysql.connect(
        host="swu-mysql.xxxxxxxx.ap-northeast-2.rds.amazonaws.com", #AWS RDS MySQL Server의 End Point
        user="admin", password="mzc2023!", database="STUDY", port=3306
    )
    curs = conn.cursor()
    uid = event["queryStringParameters"]["uid"]
    passwd = event["queryStringParameters"]["upasswd"]
    uname = event["queryStringParameters"]["uname"]
    email = event["queryStringParameters"]["email"]
    phone = event["queryStringParameters"]["phone"]
    addr1 = event["queryStringParameters"]["addr1"]
    addr2 = event["queryStringParameters"]["addr2"]
    birthday = event["queryStringParameters"]["birthday"]
    gender = event["queryStringParameters"]["gender"]
    sql = "UPDATE TMEMBER SET fname=%s, femail=%s, fphone=%s, faddr1=%s, faddr2=%s, fbirthday=%s, fgender=%s "
    sql = sql + " WHERE fid = %s AND fpass = %s "
    result = curs.execute(sql, (uname, email, phone, addr1, addr2, birthday, gender, uid, passwd))
    conn.commit()
    return {
        'statusCode': 200,
        'body': json.dumps(result)
    }
```

3) 테스트 이벤트의 이벤트 JSON

```
{
  "queryStringParameters": {
    "uid": "ccc-2",
    "upasswd": "111",
    "uname": "장동건",
    "email": "shj@gmail.com",
    "phone": "010-8888-8888",
    "addr1": "제주도 제주시 효행로",
    "addr2": "222동 3333호",
    "birthday": "2004-12-31",
    "gender": "남"
  }
}
```


7. DELETE문을 이용한 행 삭제

1) 절차

데이터베이스 연결 → 커서 생성 → 쿼리 실행 → 커밋 → 연결 종료

2) 기본 Lambda Function

함수명 : ex-rds-mysql-delete

```
import json
import pymysql

def lambda_handler(event, context):
    conn = pymysql.connect(
        host="swu-mysql.xxxxxxxxxx.ap-northeast-2.rds.amazonaws.com", #AWS RDS MySQL Server의 End Point
        user="admin", password="mzc2023!", database="STUDY", port=3306
    )
    curs = conn.cursor()
    uid = event["queryStringParameters"]["uid"]
    passwd = event["queryStringParameters"]["upasswd"]
    sql = "DELETE FROM TMEMBER WHERE fid = %s AND fpass = %s "
    result = curs.execute(sql, (uid, passwd))
    conn.commit()

    return {
        'statusCode': 200,
        'body': json.dumps(result)
    }
```

3) 테스트 이벤트의 이벤트 JSON

```
{
  "queryStringParameters": {
    "uid": "aaa2",
    "upasswd": "1111"
  }
}
```