

AWS Lambda and RDS connection

Creating RDS is straightforward. But the jargon around security groups, VPC, IAM permissions that are required to set up Lambda connection can be overwhelming. I'm writing this after spending close to 8 hours on understanding them.

Security Groups, VPC, IAM and Public Access

Select suitable options, remember the password of the DB, you can only see it once you create it. Choose a suitable VPC. VPC is virtual private cloud. Imagine your RDS instance as a big castle with lots of rooms where you keep all your important stuff. You want to make sure that only people you trust can enter the castle and access your stuff. So, you have a gatekeeper, called a security group, who checks everyone who wants to come in. When you open the castle to the public, you allow anyone to come in and look around. But, when you set the castle to private, only people who you have explicitly given permission to can come in. Note that even if we keep the castle open to public, the gatekeeper must allow them.

Put public access to 'yes' so that you can access it from local. When creating the lambda and RDS ensure you set both of them to same VPC and security group.

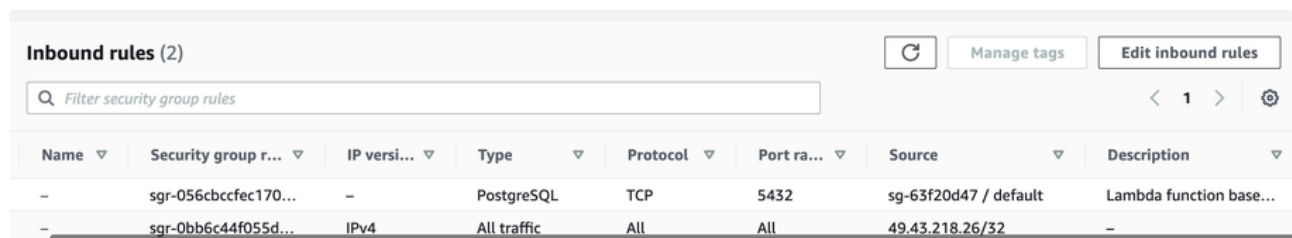
To ensure lambda can access the RDS instance connection (postgres for example), you have to add an inbound rule to permit lambda access (even if it is in same VPC).

To allow your Lambda function to communicate with your RDS instance, you will need to modify the inbound rules of the security group associated with your RDS instance to allow traffic from the security group associated with your Lambda function.

Here are the steps to modify the inbound rules of your RDS instance security group:

1. Go to the Amazon RDS console.
2. Select your RDS instance.
3. Go to the "Details" tab and note the security group associated with the instance.
4. Go to the "Security Groups" tab and select the security group associated with the instance.
5. Click the "Edit" button to modify the inbound rules.
6. Add a new rule for inbound traffic with the following settings:
 - Type: PostgreSQL (assuming you are using PostgreSQL for your RDS instance)
 - Protocol: TCP
 - Port Range: 5432 (assuming the default port is being used)
 - Source: Choose "Custom" and enter the security group ID associated with your Lambda function.
7. Save the changes.

By adding this inbound rule, you are allowing traffic from the security group associated with your Lambda function to communicate with your RDS instance over the PostgreSQL default port. If you want to connect your local system to DB, you need to add another inbound rule, check the below figure.



Inbound rules (2)							
<input type="text" value="Filter security group rules"/>							
<div>Manage tags Edit inbound rules</div>							
<div>< 1 > ⚙</div>							
Name	Security group r...	IP versi...	Type	Protocol	Port ra...	Source	Description
-	sg-056cbccfec170...	-	PostgreSQL	TCP	5432	sg-63f20d47 / default	Lambda function base...
-	sg-0bb6c44f055d...	IPv4	All traffic	All	All	49.43.218.26/32	-

Using same security group for both lambda and RDS instance saves time.

Please note that security groups and IAMs are different concepts that work together to control access to resources like RDS instances. Asking Chatgpt:

Question:

How is security group related to IAM role of lambda function. I see there are permissions like AmazonRDSFullAccess, AWSLambdaBasicExecutionRole, AWSLambdaVPCAccessExecutionRole. But by default if lambda has same VPC as compared to RDS instance then it will automatically have permission right? Please explain normally as well as with the ELI5 example (preferably continuing the castle example)

Answer:

In simple terms, IAM roles define the set of permissions that a Lambda function has to interact with AWS services. For example, the AmazonRDSFullAccess policy provides the necessary permissions to access an RDS instance. When you create a Lambda function, you can assign it an IAM role that has the appropriate permissions to access your RDS instance.

On the other hand, a security group is like a virtual firewall that controls inbound and outbound traffic to and from a resource like an RDS instance. When you create an RDS instance, you can assign it a security group that defines the inbound and outbound rules for the instance. For example, you can create a security group that allows inbound traffic from your Lambda function and denies inbound traffic from all other sources.

When you configure a Lambda function to connect to an RDS instance in a VPC, you need to ensure that the security groups associated with the Lambda function and the RDS instance allow the necessary traffic between them. By default, if a Lambda function is configured to use the same VPC as the RDS instance, it will automatically be able to access the RDS instance if the security groups allow the necessary traffic.

For example, let's continue with the castle example. Imagine that the castle has a moat around it, which is like a security group that controls access to the castle. The drawbridge over the moat is like the IAM role that allows certain people to enter the castle. If a person has the right permissions (IAM role) and the drawbridge is down (security group allows traffic), they can enter the castle and access its resources (RDS instance).

Question:

And by default if both have same VPC, then it means automatically there is a moat, correct? So in that case I don't need this permission AmazonRDSFullAccess, correct?

Answer:

Correct, if both the Lambda function and RDS instance are in the same VPC and security group, then they are already behind the same moat and can communicate with each other by default. In this case, you don't need to explicitly grant the IAM permission "AmazonRDSFullAccess" to the Lambda function's IAM role, as the default "AWSLambdaVPCAccessExecutionRole" will be enough for the Lambda function to access the RDS instance.

However, if you need to perform specific actions on the RDS instance or its resources, you may need to grant additional permissions to the Lambda function's IAM role using IAM policies. For example, if you need to create, modify, or delete database tables, you would need to add the appropriate IAM permission to the Lambda function's role.

Installing python packages into Lambda

Lambda comes with almost no preinstalled python libraries. There is a specified version of python though that the function is using. So you have to send the library files and code as a `package` and upload them as a zip file [📦 Deploy Python Lambda functions with .zip file archive](#)

`s - AWS Lambda`

Packages without any C dependency can be easily uploaded but issues can arise for others. Any C dependency needs a install from source. It is better to check online for a particular library if import fails while testing the lambda function. Sadly testing the lambda function can only be done after it's deployed (afaik). Take the case of SQLAlchemy and psycopg2 I faced.

Using both the two methods I was able to install (`pip install <package>`) but when I deployed them into Lambda there was import error.

Various online answers were tried but later it was realised that I'm downloading in MacOS and the lambda function executes in Linux, causing discrepancy. So I had to install this way:

```
1 python -m pip install --platform=manylinux1_x86_64 --only-binary=:all: pandas numpy pymysql psycopg2-binary SQLAlchemy
2 zip -r ../my-deployment-package.zip .
3 cd .. zip
4 my-deployment-package.zip lambda_function.py
5 aws lambda update-function-code --function-name crud_postgres --zip-file fileb://my-deployment-package.zip
6 EXAMPLE OUTPUT OF ABOVE COMMAND. IF THE 'CodeSha' gets updated in Console then it
```

```
7 means function is updated. There will be small drop in memory due to compression.
8 {
9   "FunctionName": "mylambdafunction",
10  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:mylambdafunction",
11  "Runtime": "python3.9",
12  "Role": "arn:aws:iam::123456789012:role/lambda-role",
13  "Handler": "lambda_function.lambda_handler",
14  "CodeSize": 5912988,
15  "CodeSha256": "A2P0NUWq1J+LtSbkuP8tm9uNYqs1TAa3M76ptmZCw5g=",
16  "Version": "$LATEST",
17  "RevisionId": "5afdc7dc-2fcb-4ca8-8f24-947939ca707f",
18  ...
19 }
```