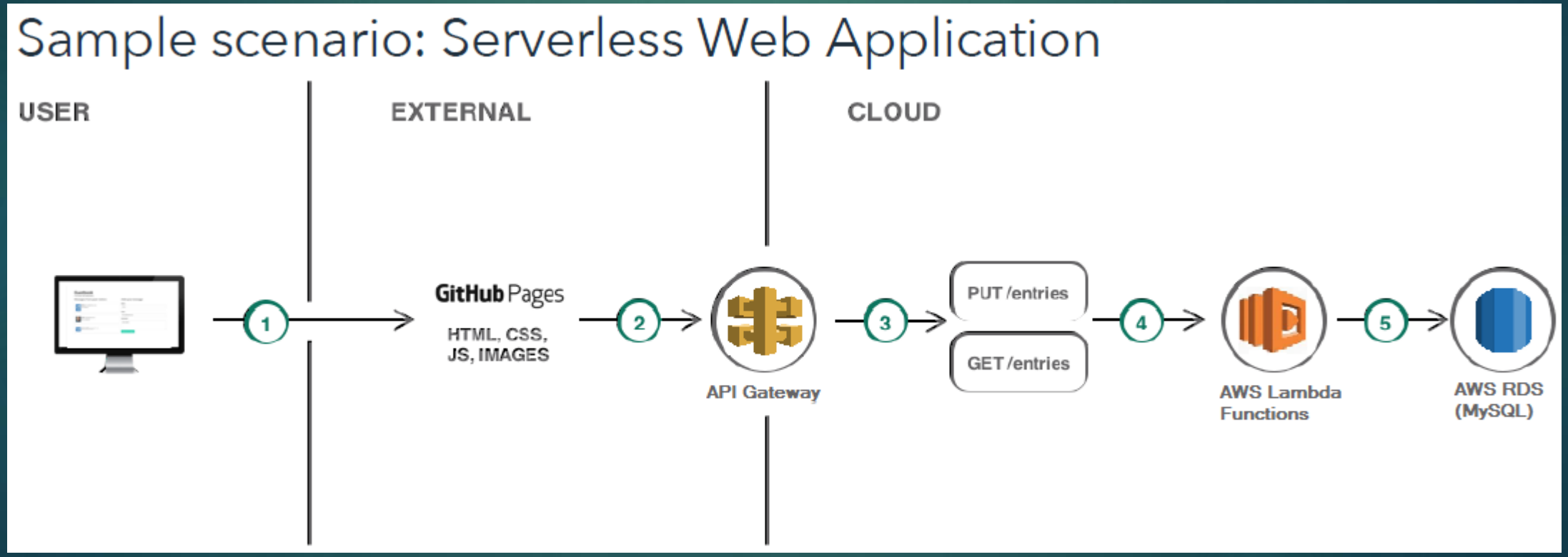
Building A 101 Web App

THE SERVERLESS WAY

Runcy Oommen
https://runcy.me

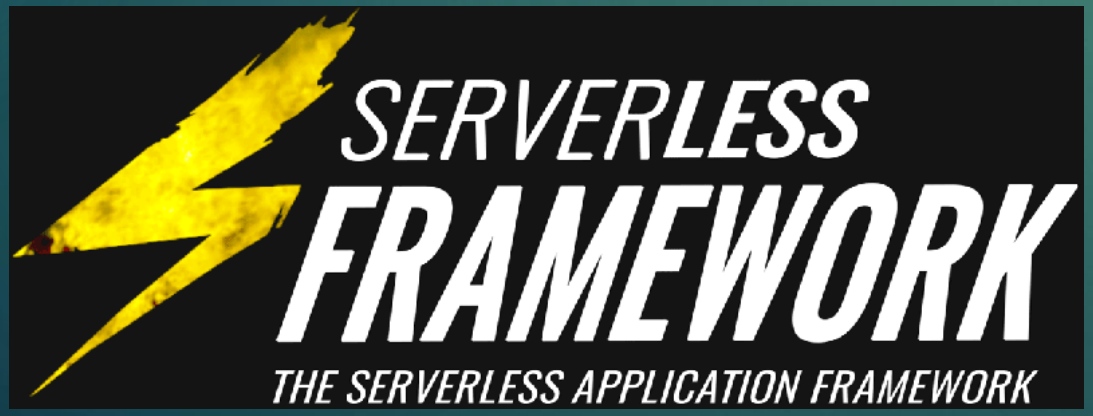
What is Serverless?

"Serverless computing is a cloud computing execution model in which the cloud provider dynamically manages the allocation of machine resources. Pricing is based on the actual amount of resources consumed by an application." (via Wikipedia)



What is “Serverless”?

- A way to host and run your code without having to *think* about servers.
- The ability to expose a single function, run in on a shared server, and pay only for the milliseconds it takes to execute.
- A command line tool that simplifies building and deploying serverless functions (serverless framework).



github.com/serverless/serverless

A Few Good Resources

- **AWS Info page on serverless**

  https://aws.amazon.com/serverless/

- **Serverless Architectures**

  https://martinfowler.com/articles/serverless.html

- **Lambda + Serverless**

  https://www.youtube.com/watch?v=71cd5XerKss


SHUT UP
SHOW ME DEMO

Pre-requisites

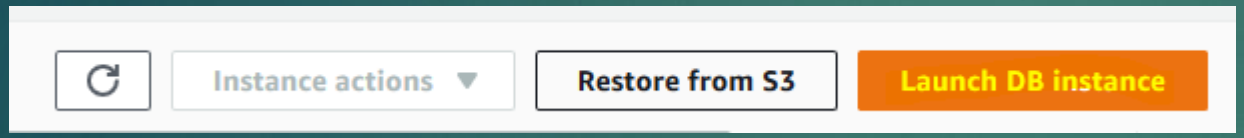
- AWS Free Tier
- Source Code For Cloning

  https://github.com/roommen/serverless101

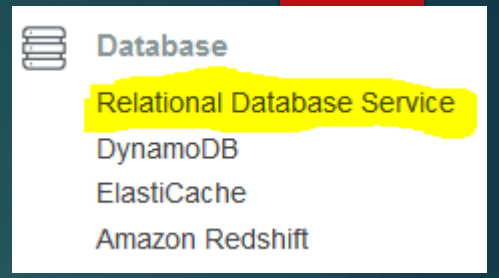
- Basic knowledge of Python, HTML, JS, CSS
- A good IDE like Visual Studio Code

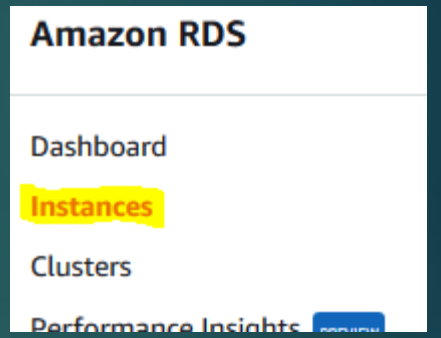
Let's start up our DB

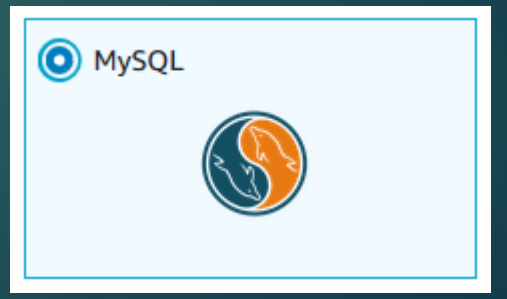
- Login to AWS Console
- Select "Relational Database Service" from Database
- Click "Instances" from the left-menu
- Click on "Launch DB Instance"



- Select "MySQL" as the engine and click "Next"

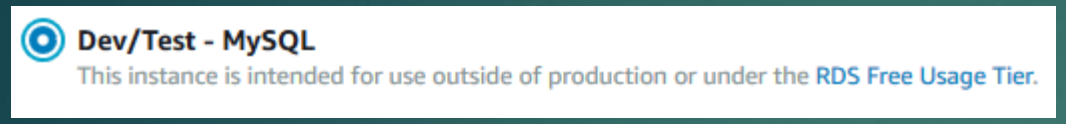




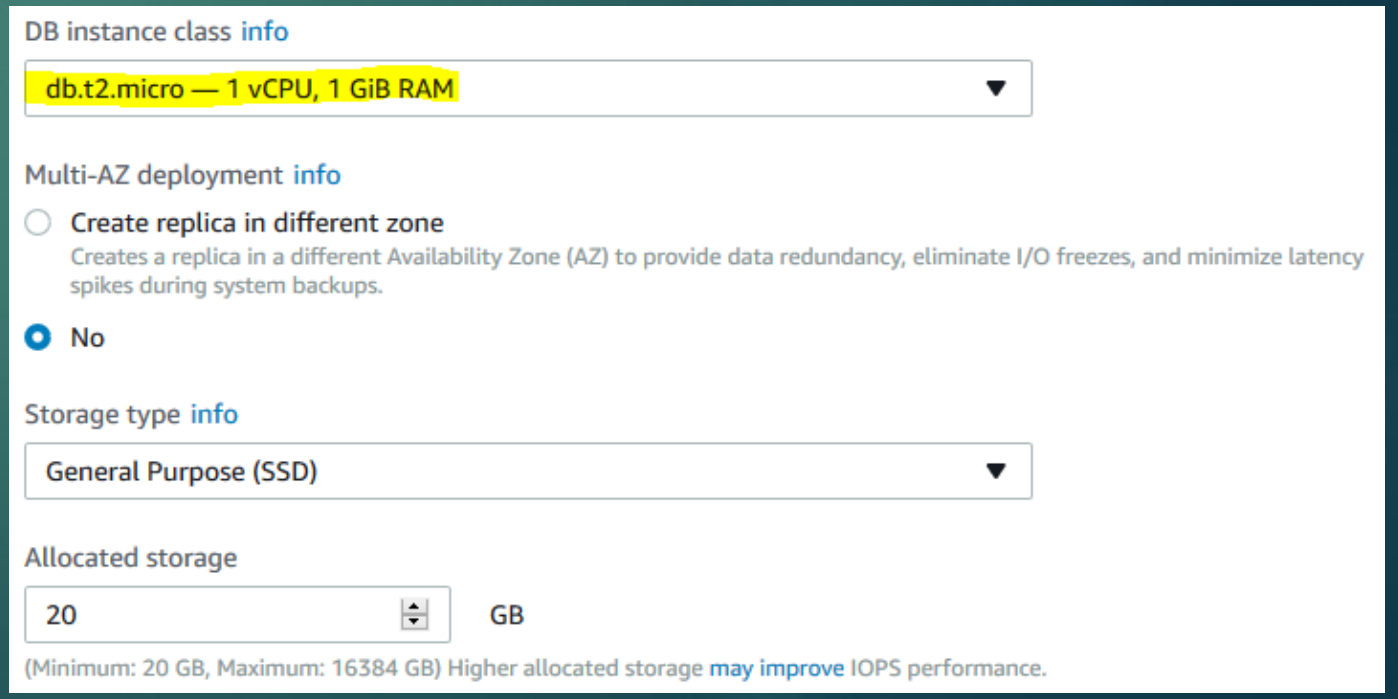


MySQL Setup

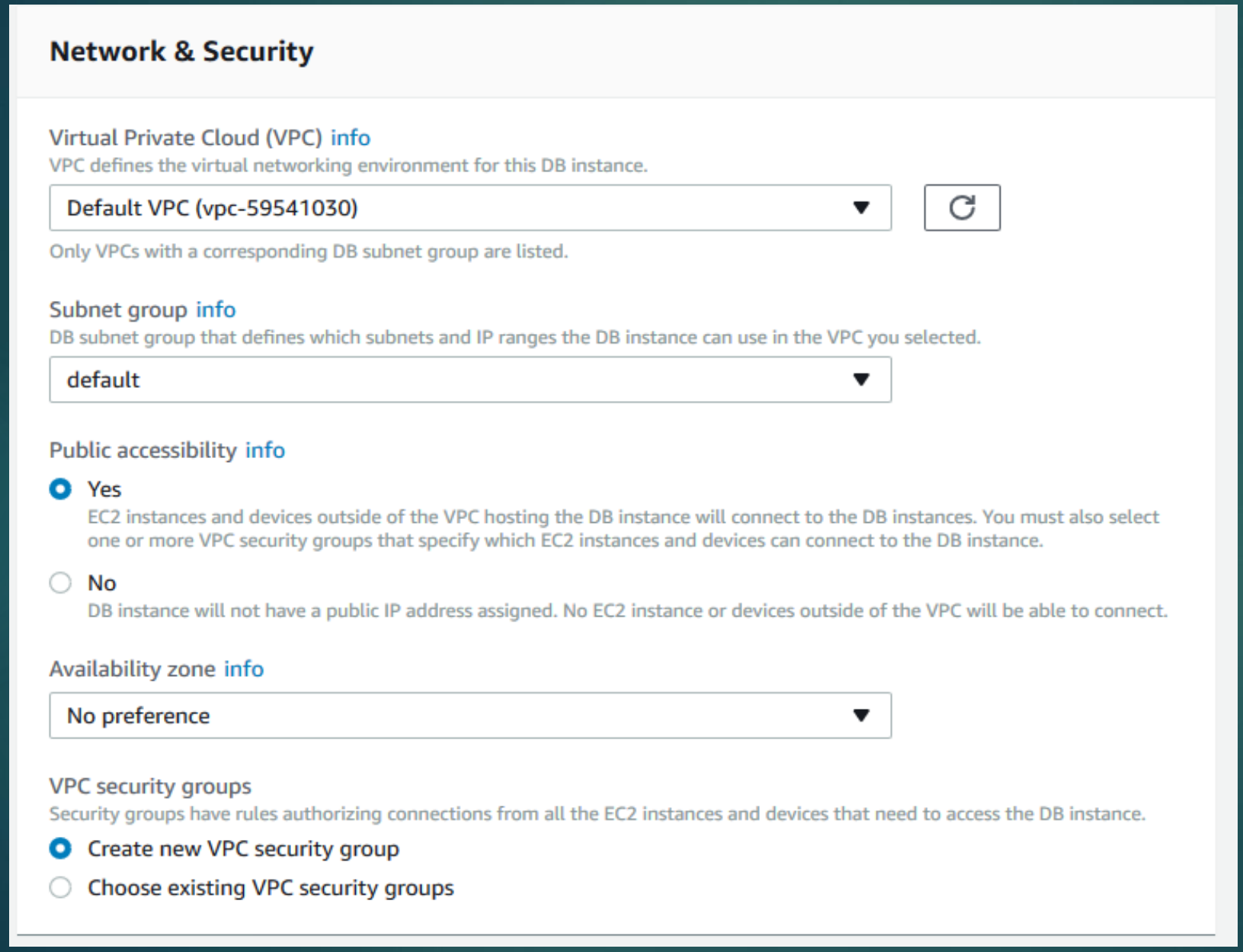
- In next screen, choose “Dev/Test – MySQL”



- Instance Specifications

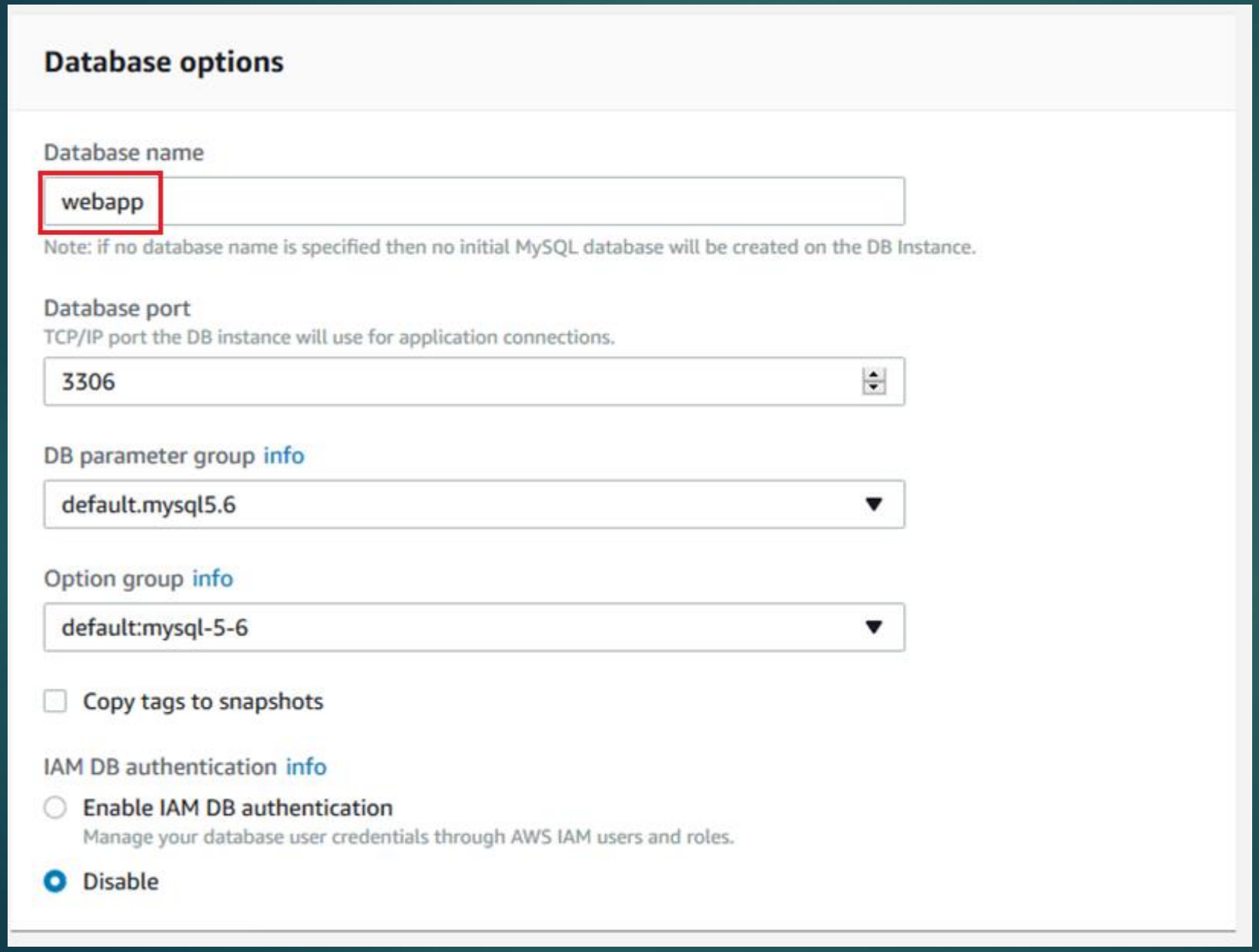


MySQL Advanced Settings - Network & Security



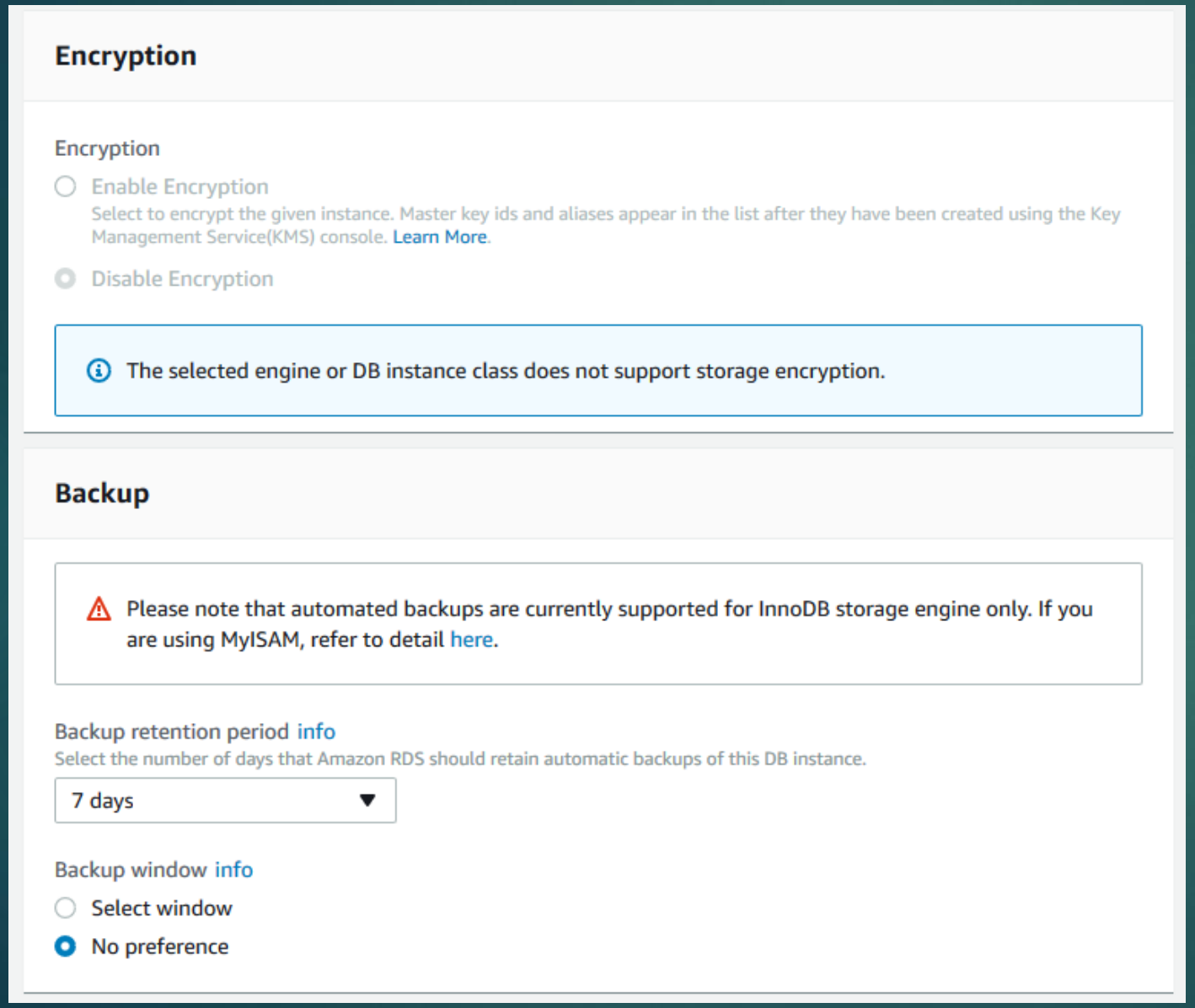
Keep everything as the default setting

MySQL Advanced Settings – Database options



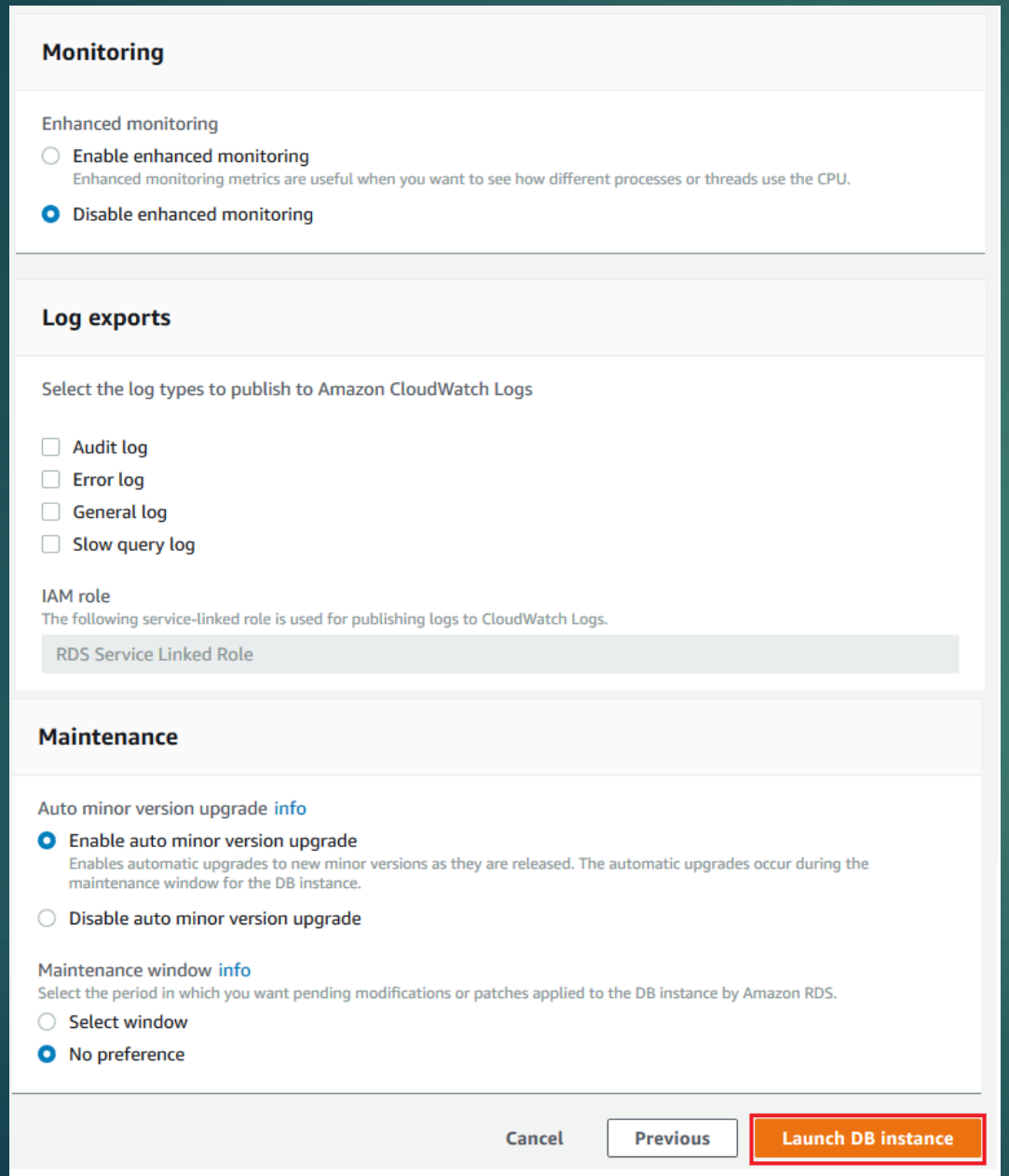
Provide an appropriate DB name

MySQL Advanced Settings – Encryption & Backup



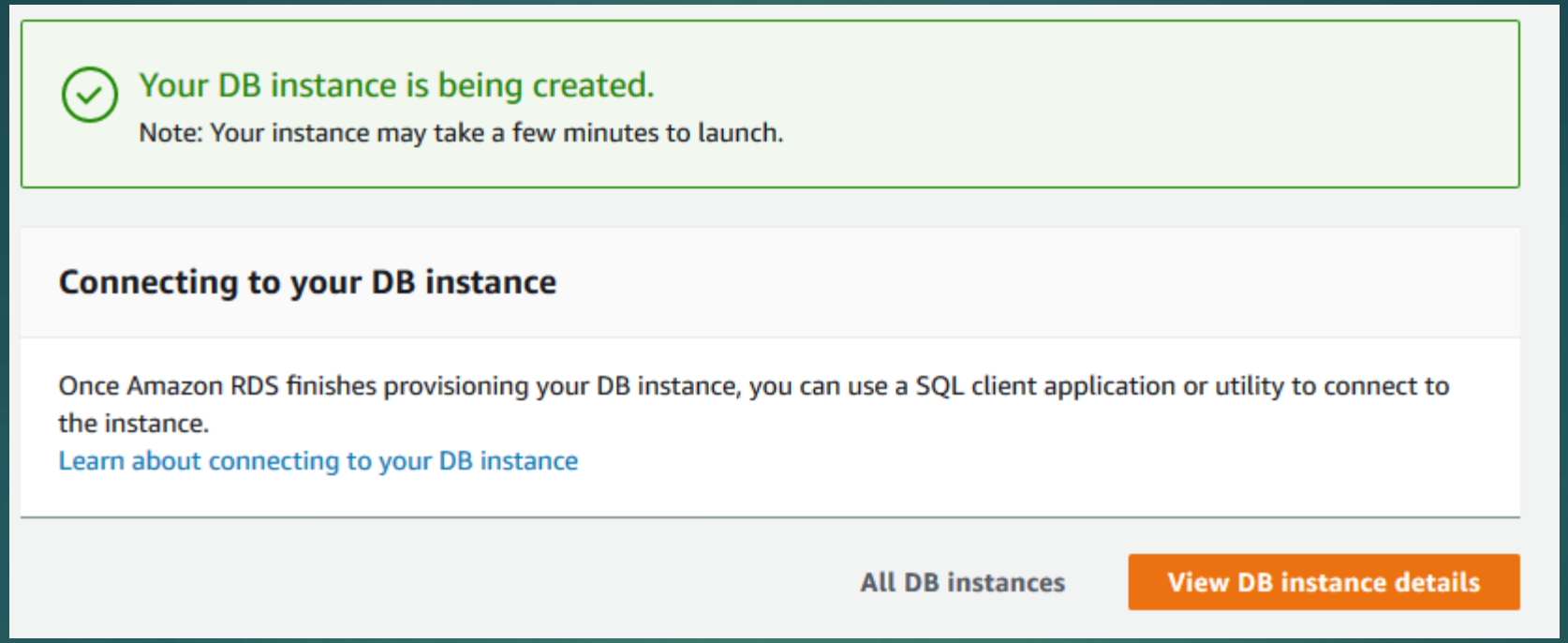
Leave everything as default

MySQL Advanced Settings – Monitoring, Log, Maintenance



- Leave everything as default
- Click “Launch DB Instance”

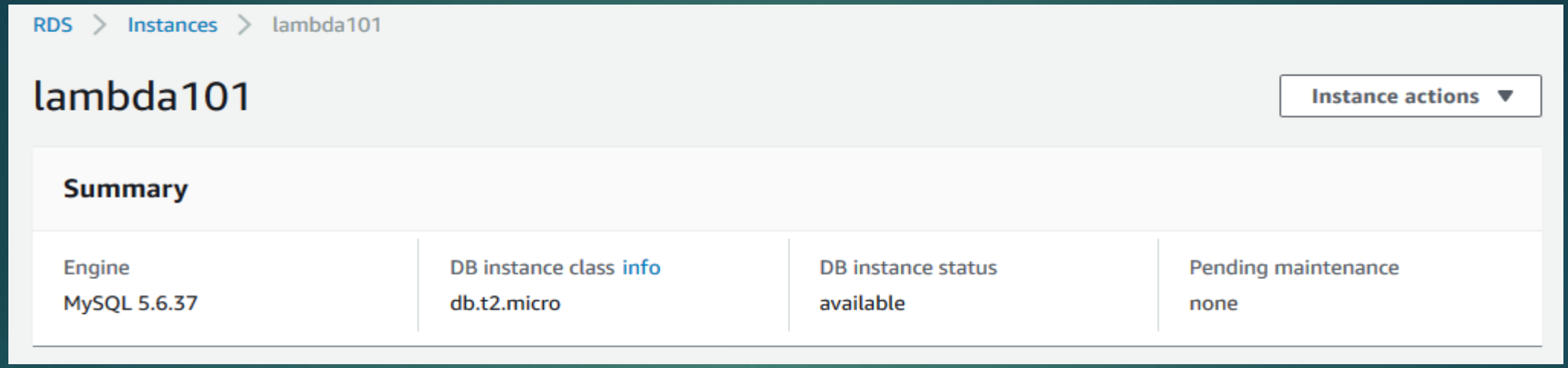
MySQL Getting Initialized

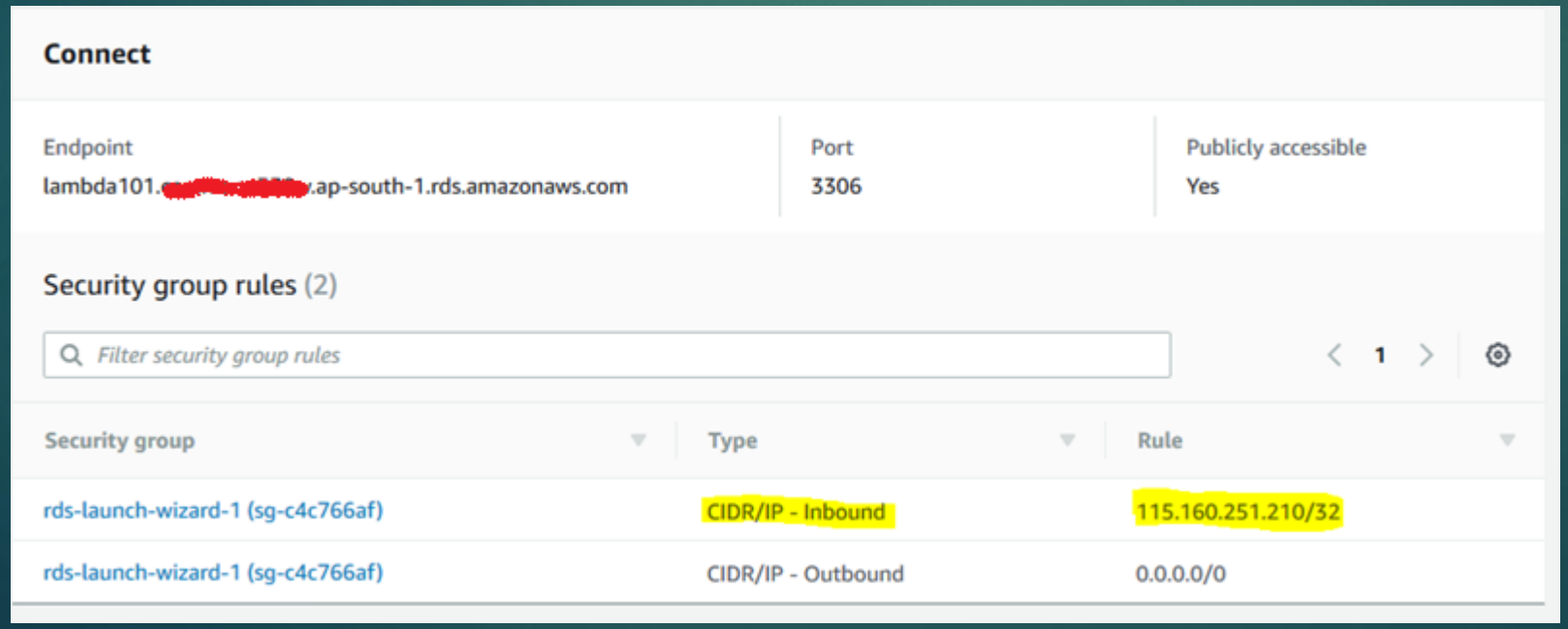


- It may take sometime for DB to be initialized and available depending on your AZ/Region

MySQL Endpoint

- Once the DB creation is successful, you should have something like this:

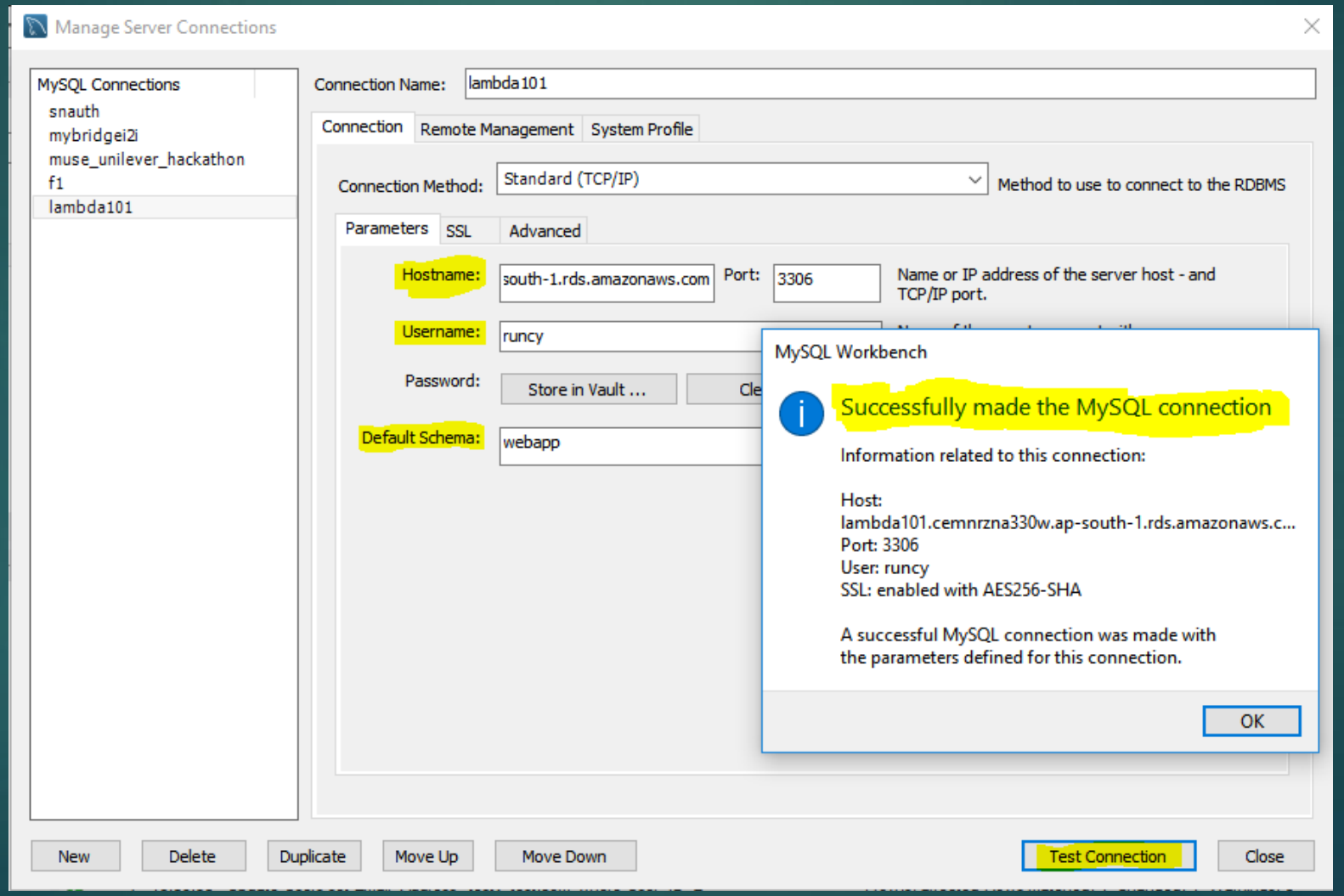




Make sure you've the right inbound and outbound rules associated with the security group

Test the connection

Use a software like MySQL Workbench to test connection, view table details, run queries etc..



Creating Users table

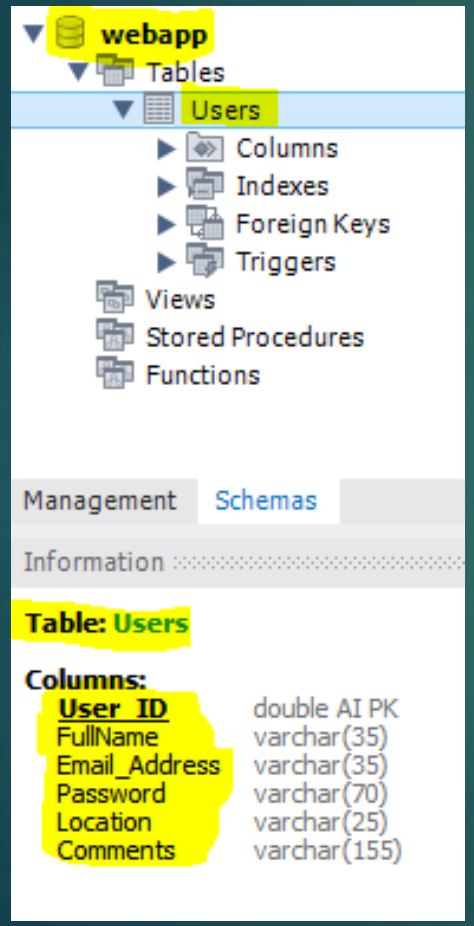
- Go to the cloned “serverless101” repository
- Navigate to the “db” folder
- Edit the ‘Create_Users.py’ file with the DB info you created earlier

Edit the details in Create_Users.py

```
def create_users():
    connection, cursor = None, None
    try:
        #Database Connection Parameters -- Replace this with your DB endpoint
        lambda101_cnx_str = {'host': 'lambda101.[redacted].ap-south-1.rds.amazonaws.com',
            'username': 'runcy',
            'password': '[redacted]',
            'db': 'webapp'}
        connection = mysql.connector.connect(host=lambda101_cnx_str['host'], user=lambda101_cnx_str['username']
                                             password=lambda101_cnx_str['password'], database=lambda101_cnx_str
        cursor = connection.cursor()
        cursor.execute('CREATE TABLE Users('
                       'User_ID DOUBLE NOT NULL AUTO_INCREMENT PRIMARY KEY,'
                       'FullName VARCHAR(35) NOT NULL,'
                       'Email_Address VARCHAR(35) NOT NULL,'
                       'Password VARCHAR(70) NOT NULL,'
                       'Location VARCHAR(25) NOT NULL,'
                       'Comments VARCHAR(155) NOT NULL)'
                       ';')
        print("Table Users created successfully.")
    except mysql.connector.Error as err:
        print(err)
    finally:
        if connection:
            connection.close()
        if cursor:
            cursor.close()

if __name__ == '__main__':
    create_users()
```

Run the Create_Users.py file

```
runcy@RUNCYOOMMEN-PC:/mnt/f/serverless101/db$ python3 Create_Users.py
Table Users created successfully.
```



- Go to MySQL Workbench
- Verify the Users table got created successfully

AWS Lambda with Python - Steps

- In this web app example, we have:
  - User Registration – handled by serverless/register_login.py
  - User Login – handled by serverless/login.py
  - User Display – handled by serverless/users.py

- Edit each of these *.py* files with DB connection parameters as created earlier

- For Python to be enabled as AWS Lambda function, we need to zip all our source code and dependencies – we use *mysql.connector* as a dependency in each of these files

AWS Lambda with Python – Extract dependencies

- Create a temp folder called *register* and copy *register_login.py* to it

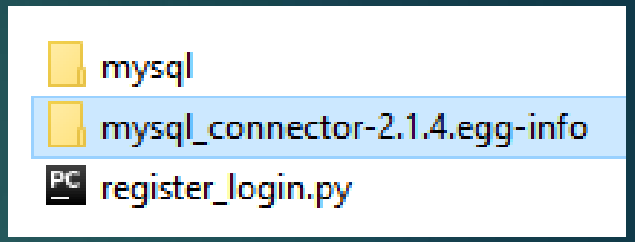
- Do a pip install of the *mysql-connector* under that folder

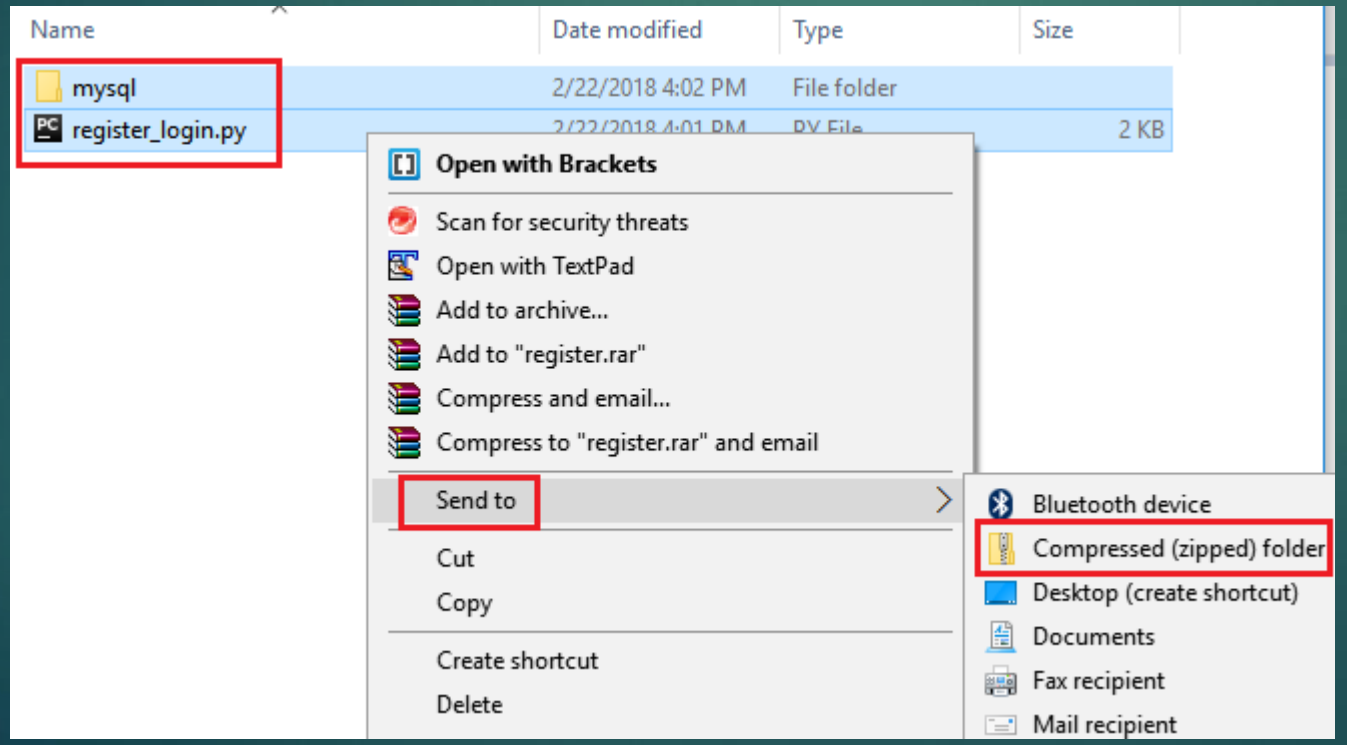
(Use specific version *2.1.4* – I was getting an error for the latest one)

```
runcy@RUNCYOOMMEN-PC:/mnt/f/serverless101/serverless$ mkdir register
runcy@RUNCYOOMMEN-PC:/mnt/f/serverless101/serverless$ cp register_login.py register
runcy@RUNCYOOMMEN-PC:/mnt/f/serverless101/serverless$ cd register/
runcy@RUNCYOOMMEN-PC:/mnt/f/serverless101/serverless/register$ pip3 install mysql-connector==2.1.4 --target .
Downloading/unpacking mysql-connector==2.1.4
  Downloading mysql-connector-2.1.4.zip (355kB): 355kB downloaded
  Running setup.py (path:/tmp/pip_build_runcy/mysql-connector/setup.py) egg_info for package mysql-connector

    warning: no files found matching 'README.txt'
Installing collected packages: mysql-connector
  Running setup.py install for mysql-connector
    Not Installing C Extension

    warning: no files found matching 'README.txt'
Successfully installed mysql-connector
Cleaning up...
```

AWS Lambda with Python – Zip 'em up

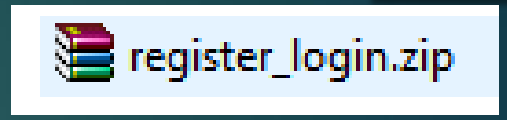
- Under the register folder, You might see a folder *mysql_connector-2.1.4.egg-info* which can be deleted if you want to
- Select the rest (*register_login.py* **file** and the *mysql* **folder**) and extract it to a zip file by right-clicking on it





AWS Lambda with Python – Zip details

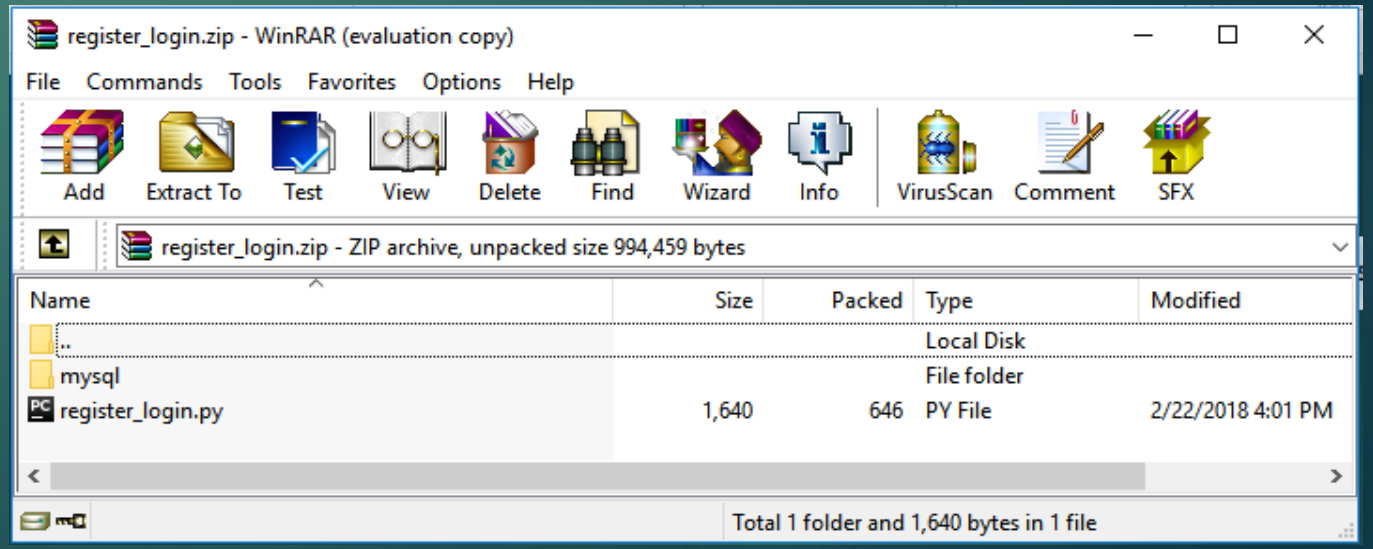
- You should now have a register_login.zip file created



- Verify the contents of this zip file and ensure that the contents look identical to screenshot below

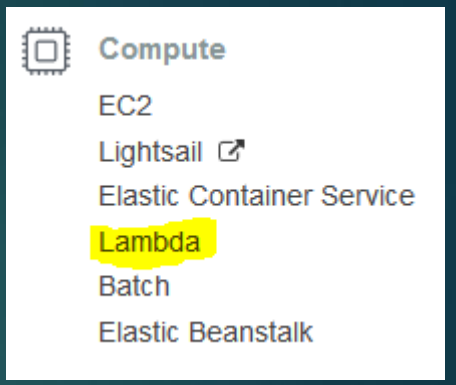
NB: The *register_login.py* file and *mysql* folder should be visible as is and not under another folder inside the zip file. Otherwise there will be problems while creating the lambda functions (later steps)

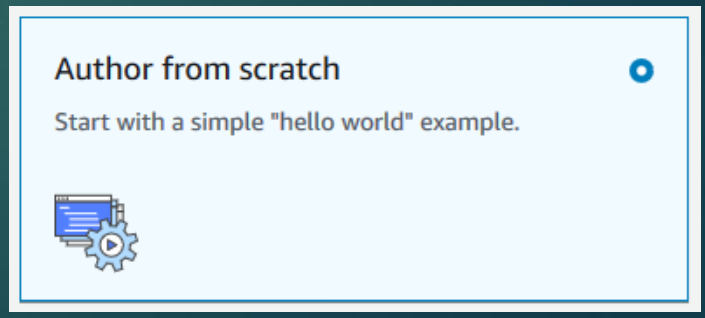
Repeat this process for *login.py* and *users.py to create login.zip* and *users.zip*



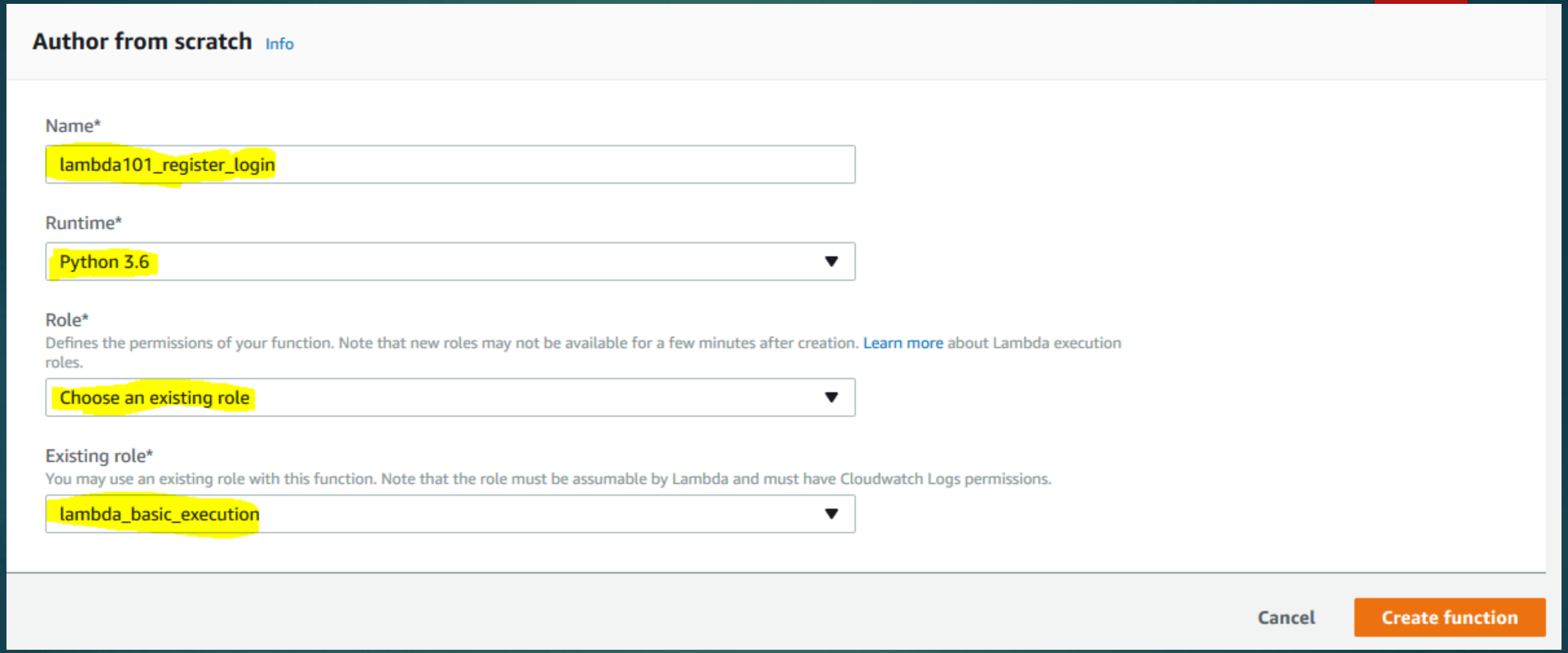
Let's create the Lambda functions

- Login to AWS Console
- Select "Lambda" from Compute
- Click "Create function"
- Select "Author from scratch"



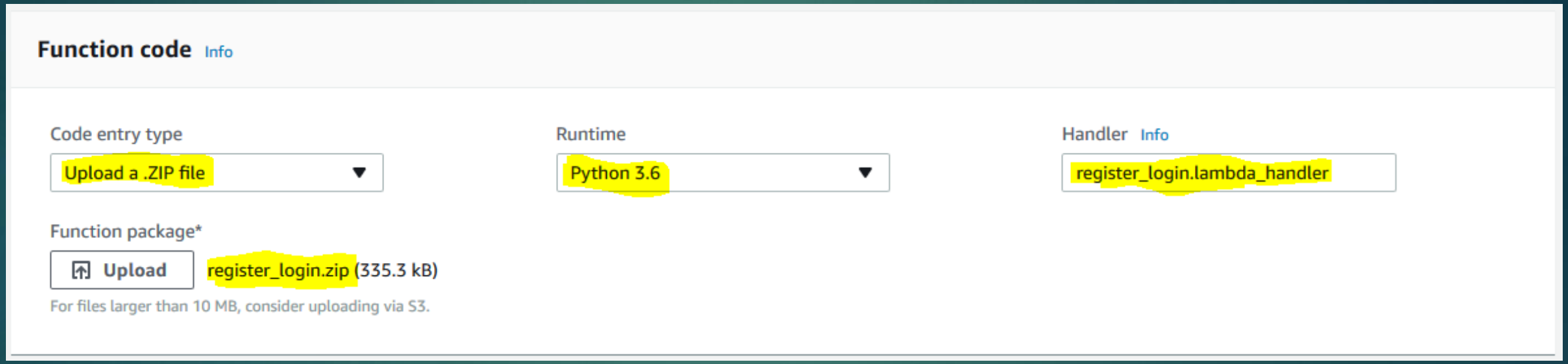


User Registration - Lambda function creation



User Registration - Lambda function code

- In the next screen, upload the zip file created earlier (*register_login.zip*) and change the Handler info to *register_login.lambda_handler*
- The format of the Handler should be *<python_filename>.lambda_handler*

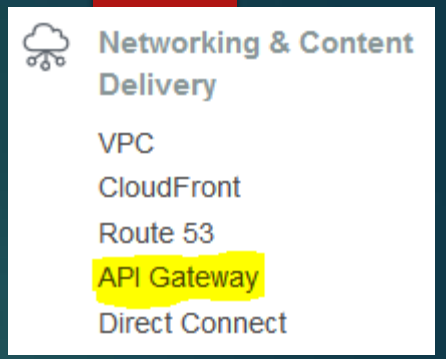


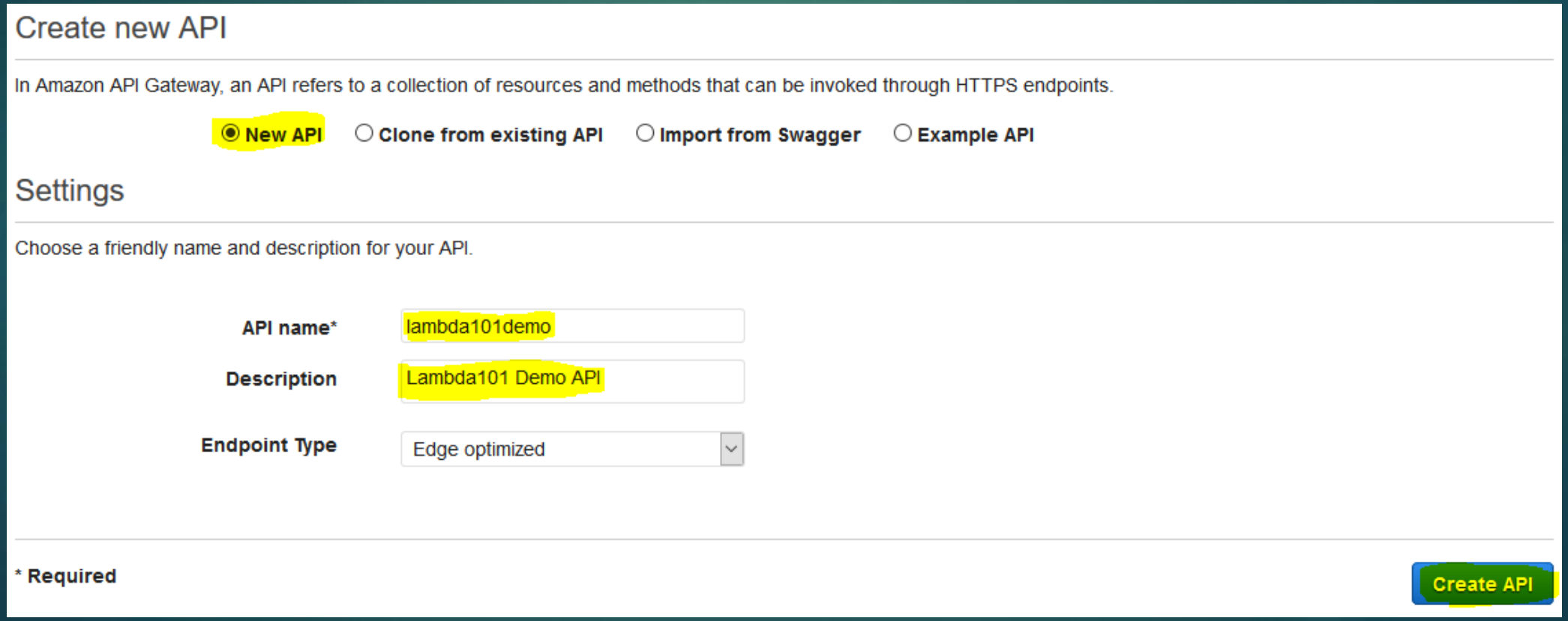
- Once done, click "Save"

- Do this for each of the remaining zip files to create three lambda functions for user registration, login and view details

Integration with API Gateway

- Login to AWS Console
- Select "API Gateway" from Networking & Content Delivery
- Click "Create API"
- Choose "New API", provide name and other details

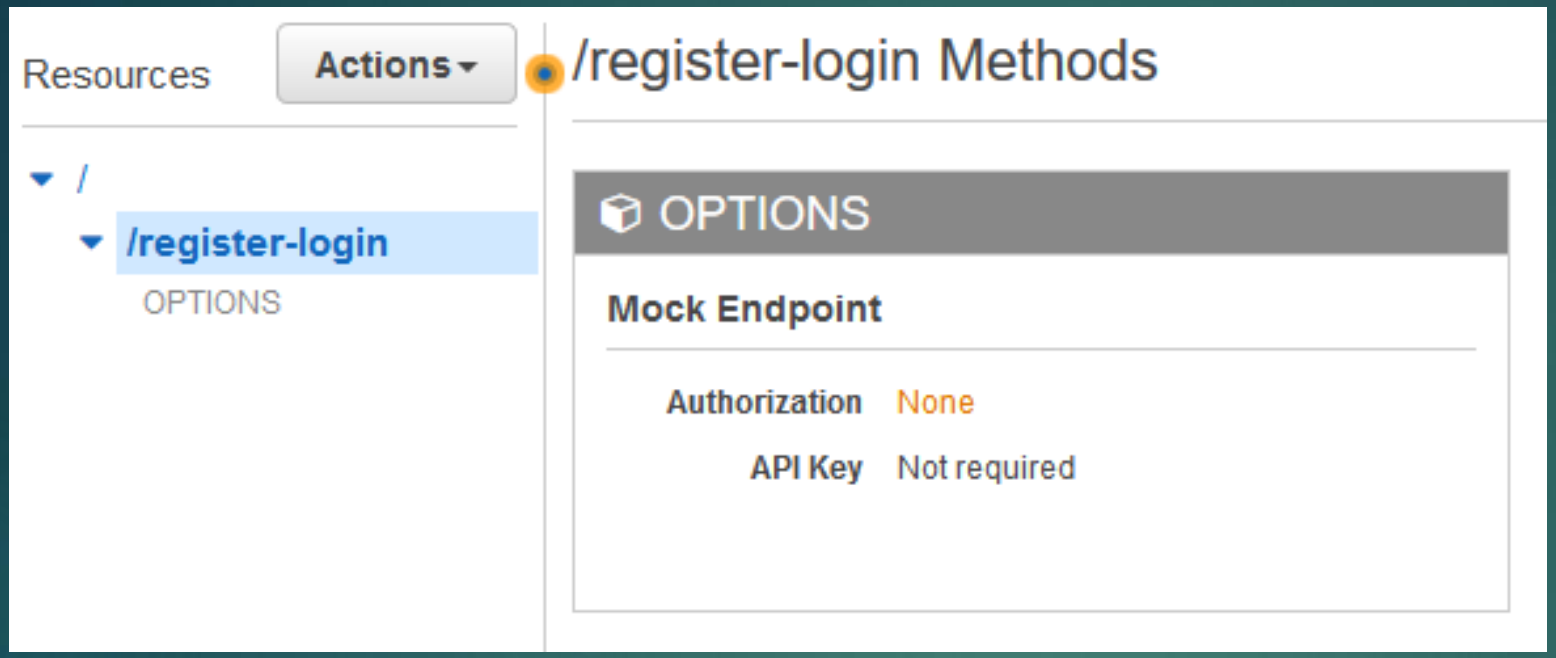




API Gateway – Create Resource (register-login)

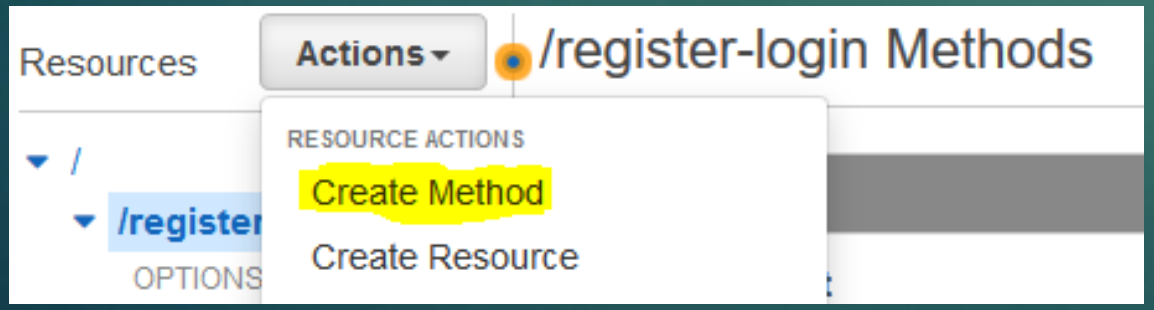
- In the next screen, choose "Create Resource" from Actions and provide appropriate details

API Gateway – Resource created (register-login)



You should see a screen similar to this after the resource is created

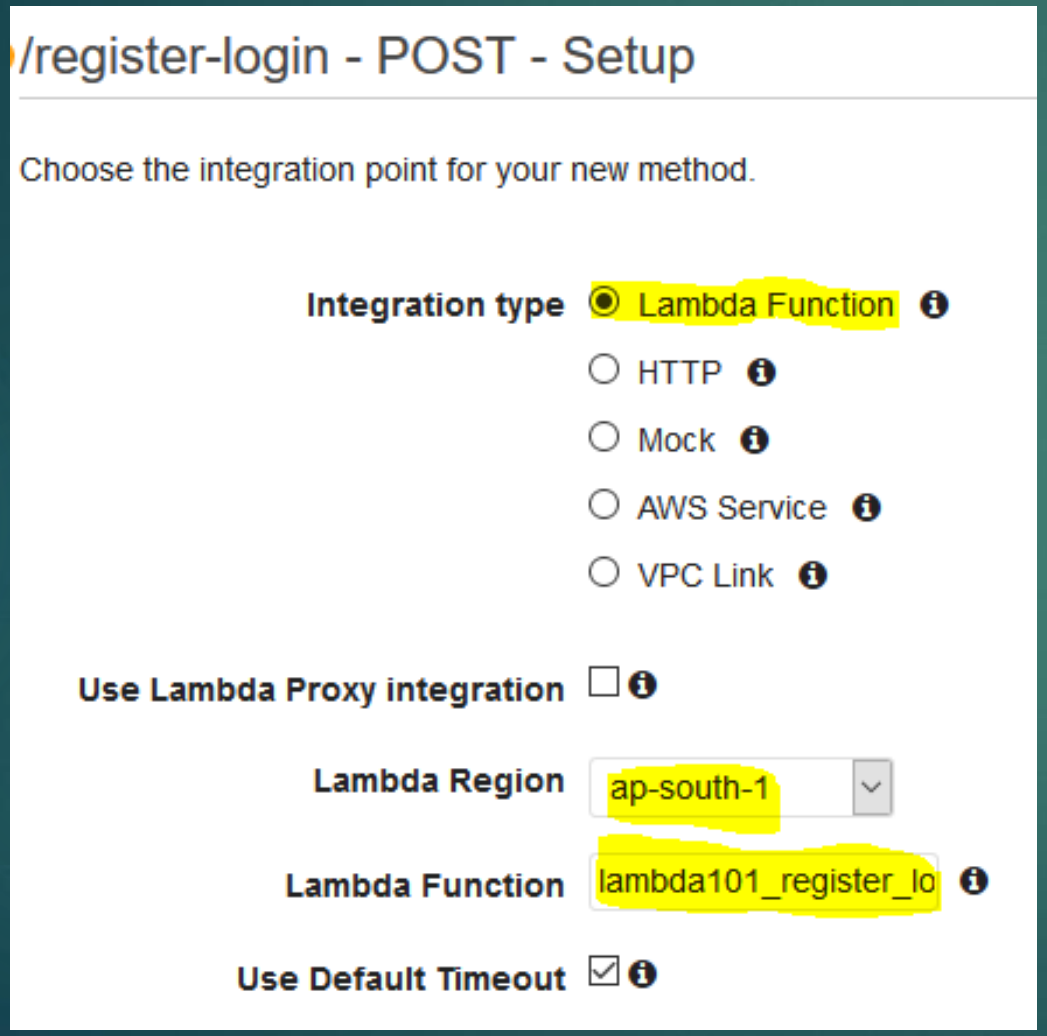
API Gateway – Create Method



- Select the resource and now click "Create Method"
- Choose "POST"

API Gateway – Configure POST (register-login)

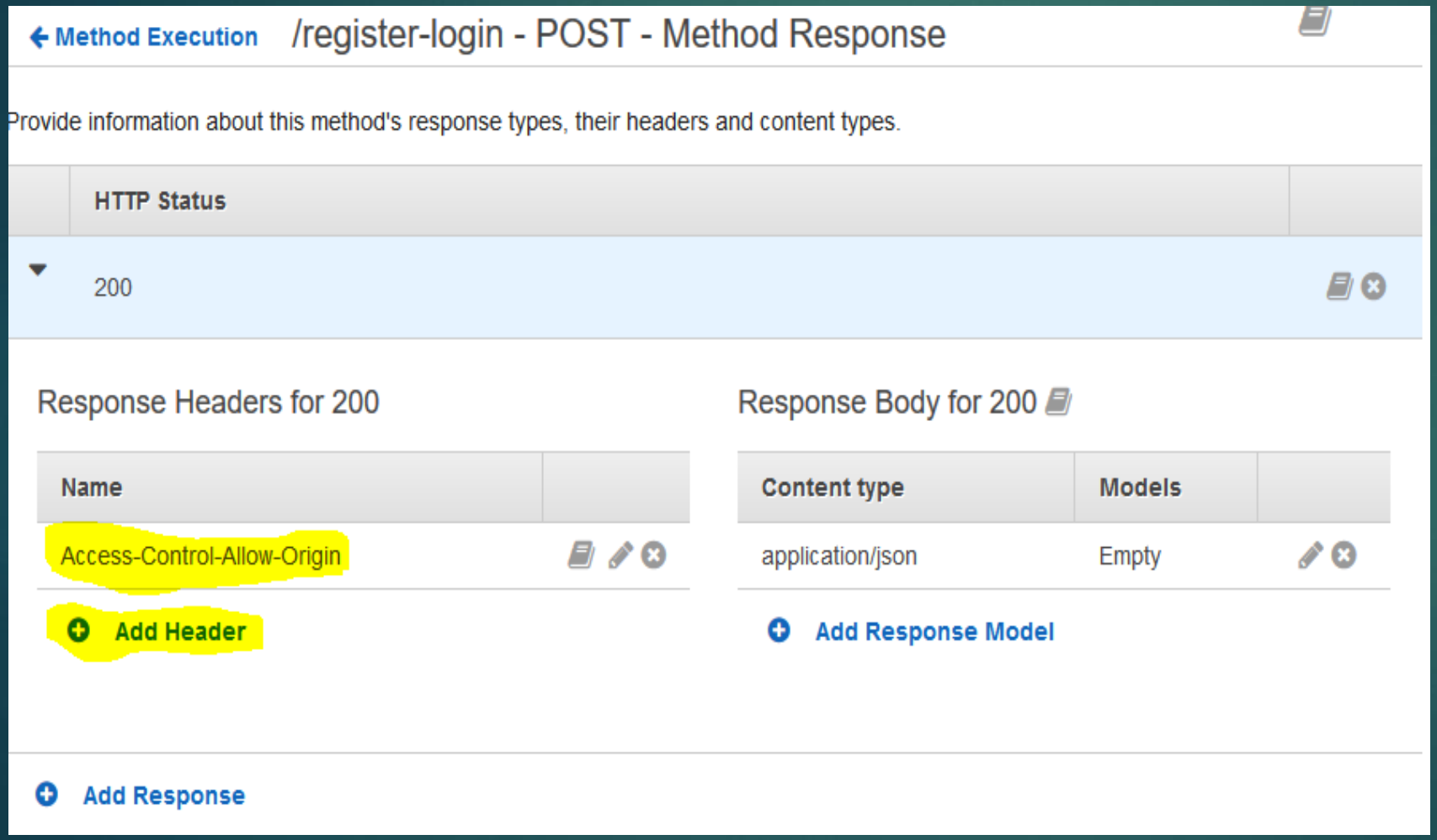
Click on the "POST" method and enter the configuration as below



- Select the appropriate region to choose the lambda function which we had created earlier

API Gateway – POST - Method Response (register-login)

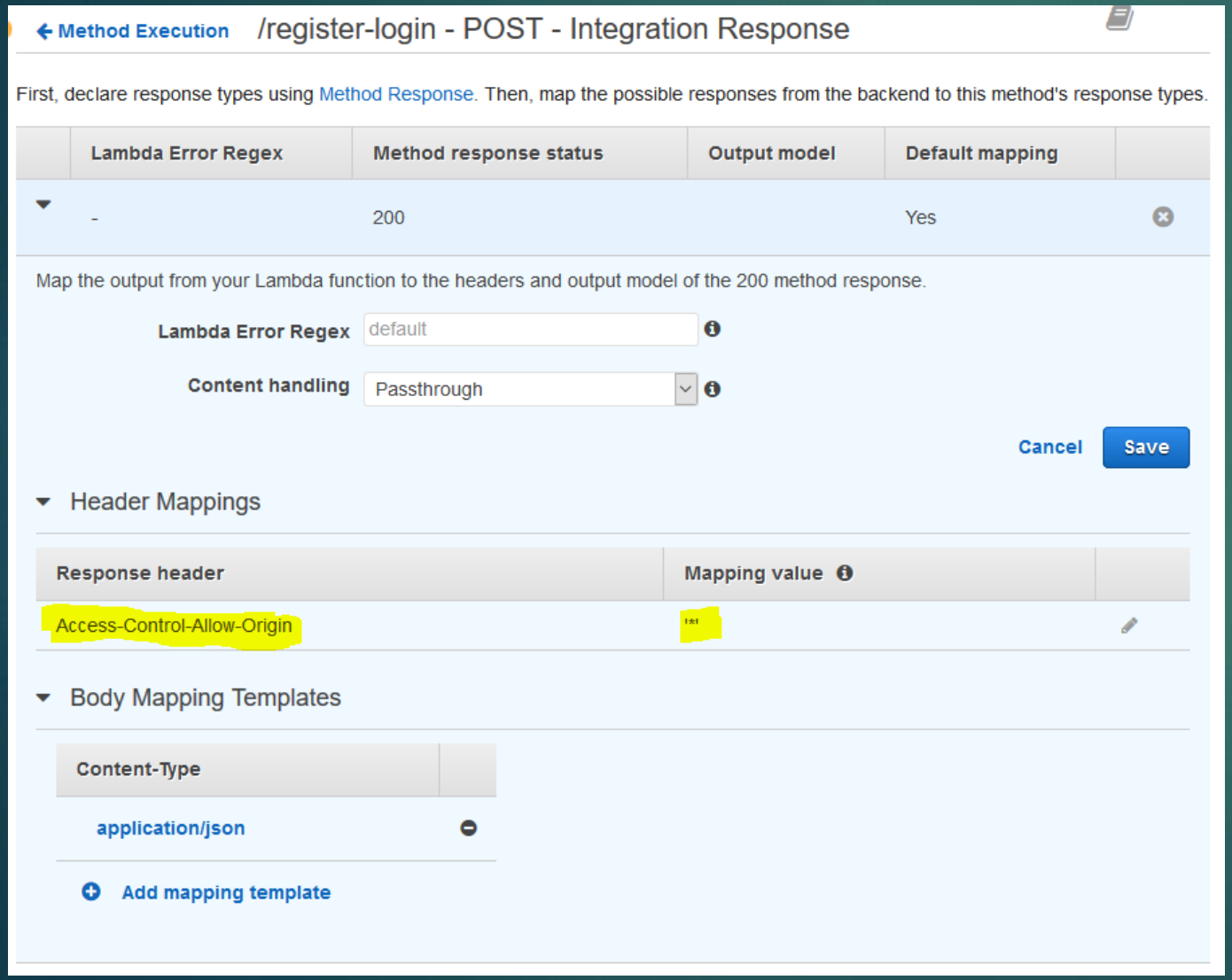
Click on the "POST" method created and choose "Method Response"

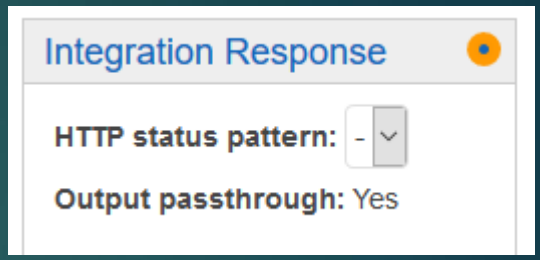


Click on "Add Header" and provide value as *Access-Control-Allow-Origin*

API Gateway – POST - Integration Response (register-login)

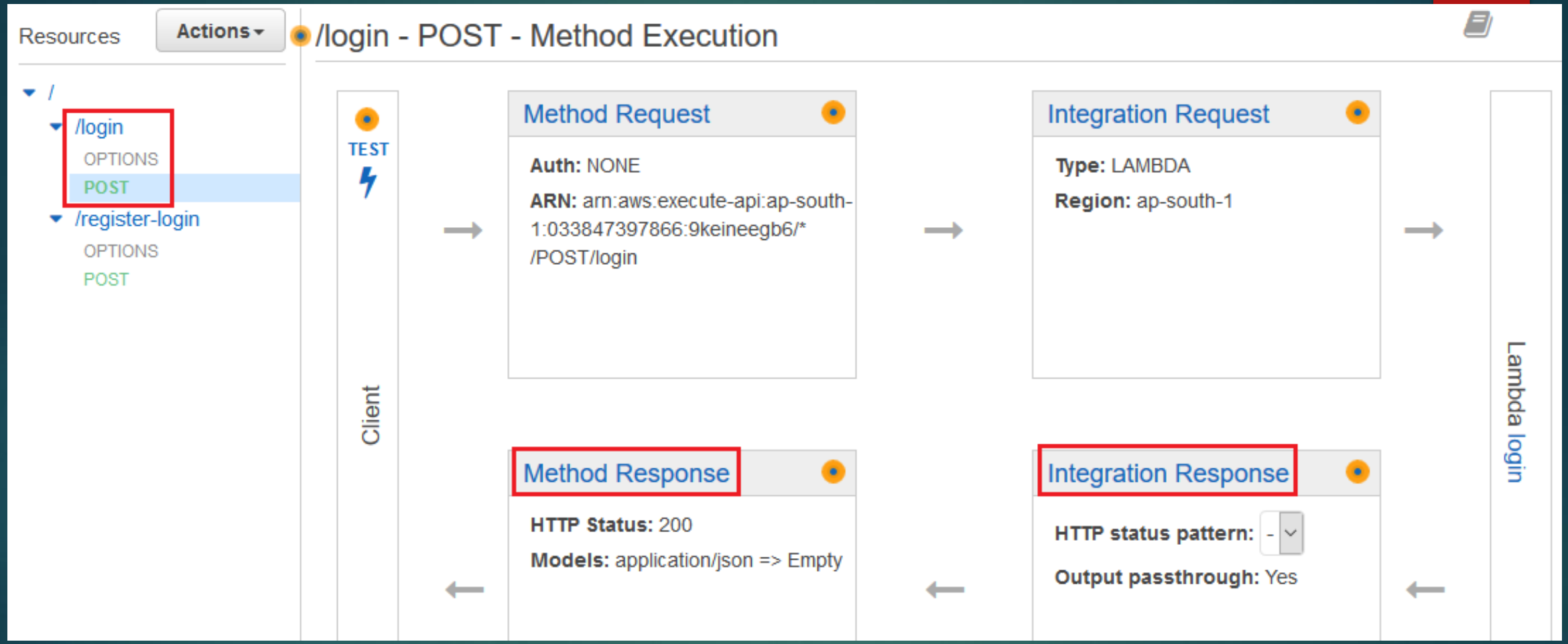
Click on the "POST" method created and choose "Integration Response"





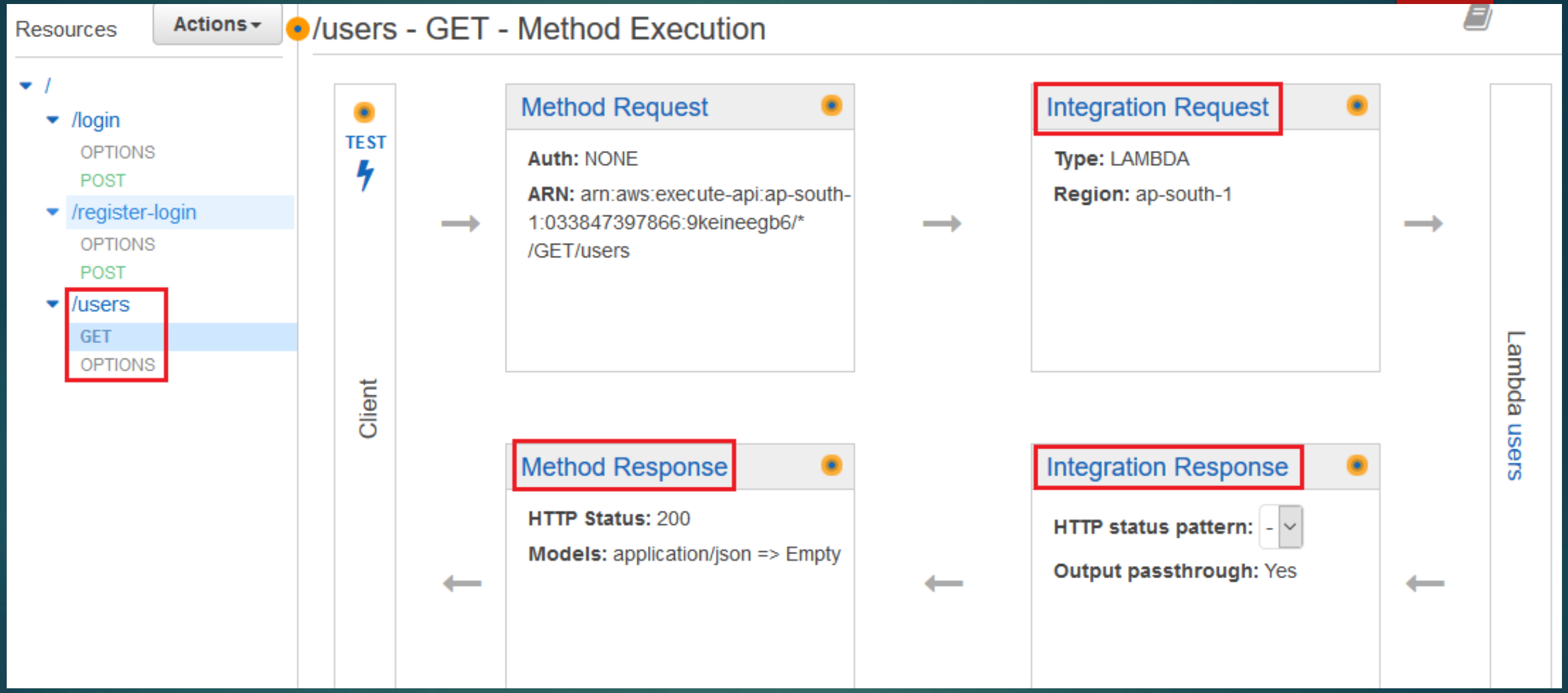
Click on the *pencil* icon next to *Access-Control-Allow-Origin* and add the value **'*'**

API Gateway – Create Resource (login) & POST Method



- Create *login* resource and associate **POST** method
- Configure *Integration Response* & *Method Response*
- Follow identical steps as the previous *register-login* for configuration

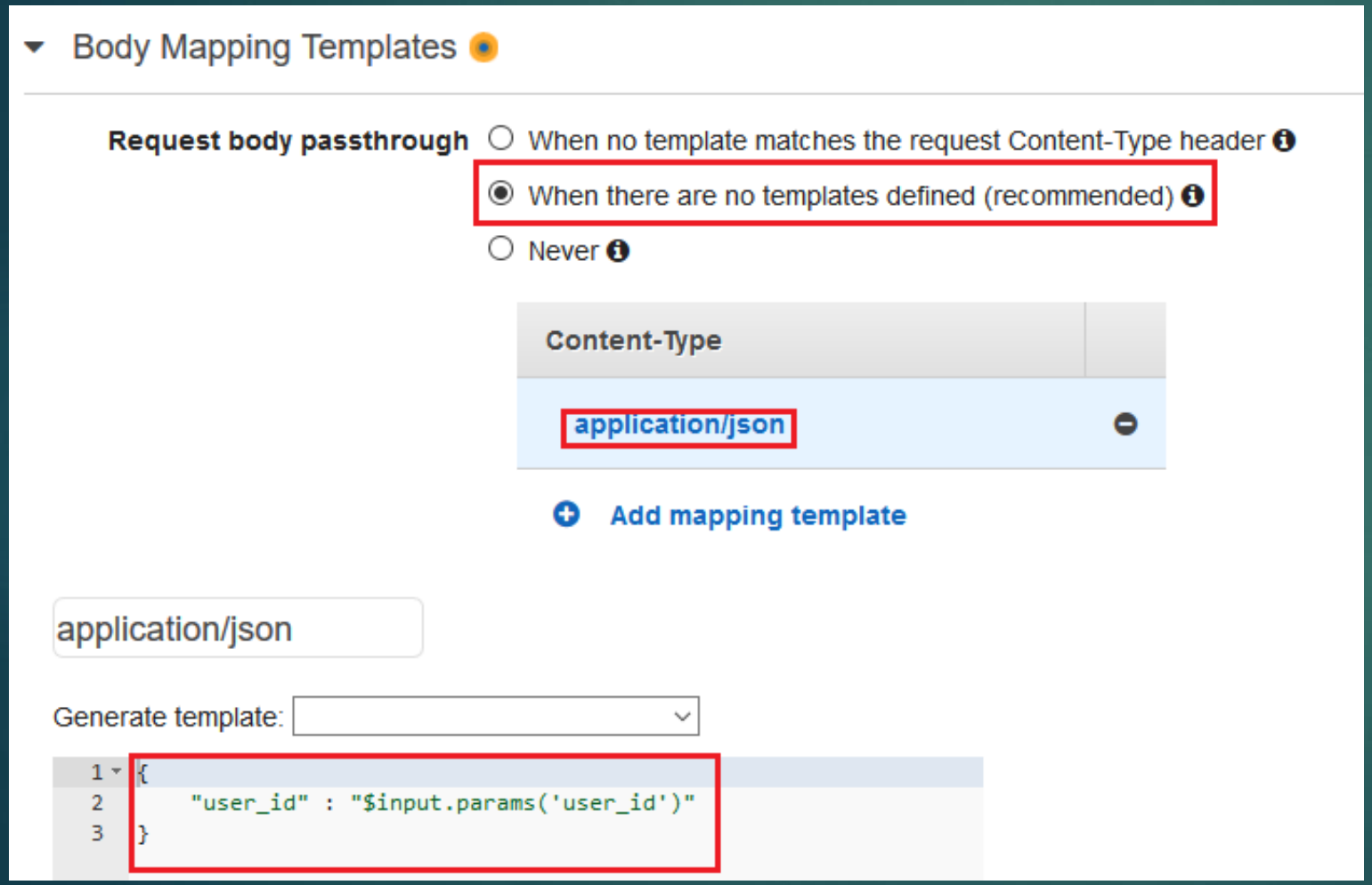
API Gateway – Create Resource (users) & GET Method



- Create *login* resource and associate **POST** method
- Configure *Integration Response* & *Method Response as done previously*

API Gateway – GET – Integration Request (users)

Click on the "GET" method created and choose "Integration Request"



Configure the *Body Mapping Templates* with *Content-Type* as *application/json*

{
"user_id" : "$input.params('user_id')"
}

users.html, showUserInfo() - lambda101.js & users.py (Lambda)

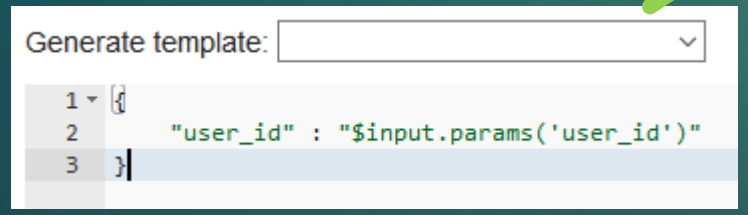
```
<script>
    $(document).ready(function(){
        showUserInfo();
    });
</script>
```

users.html

THE HOLY TRINITY

```
$.ajax({
    url: info_url,
    type: 'GET',
    data : {"user_id" : user_id},
    dataType: 'html',
    async: false,
    success: function(data)
    {
        var result = $.parseJSON(data);
```

showUserInfo() – lambda101.js

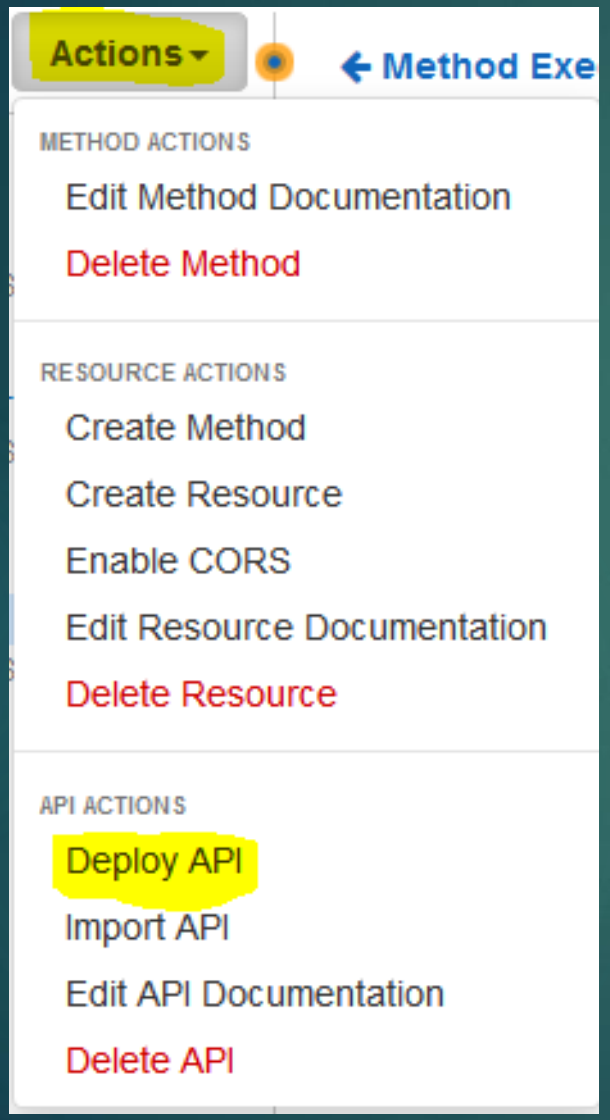


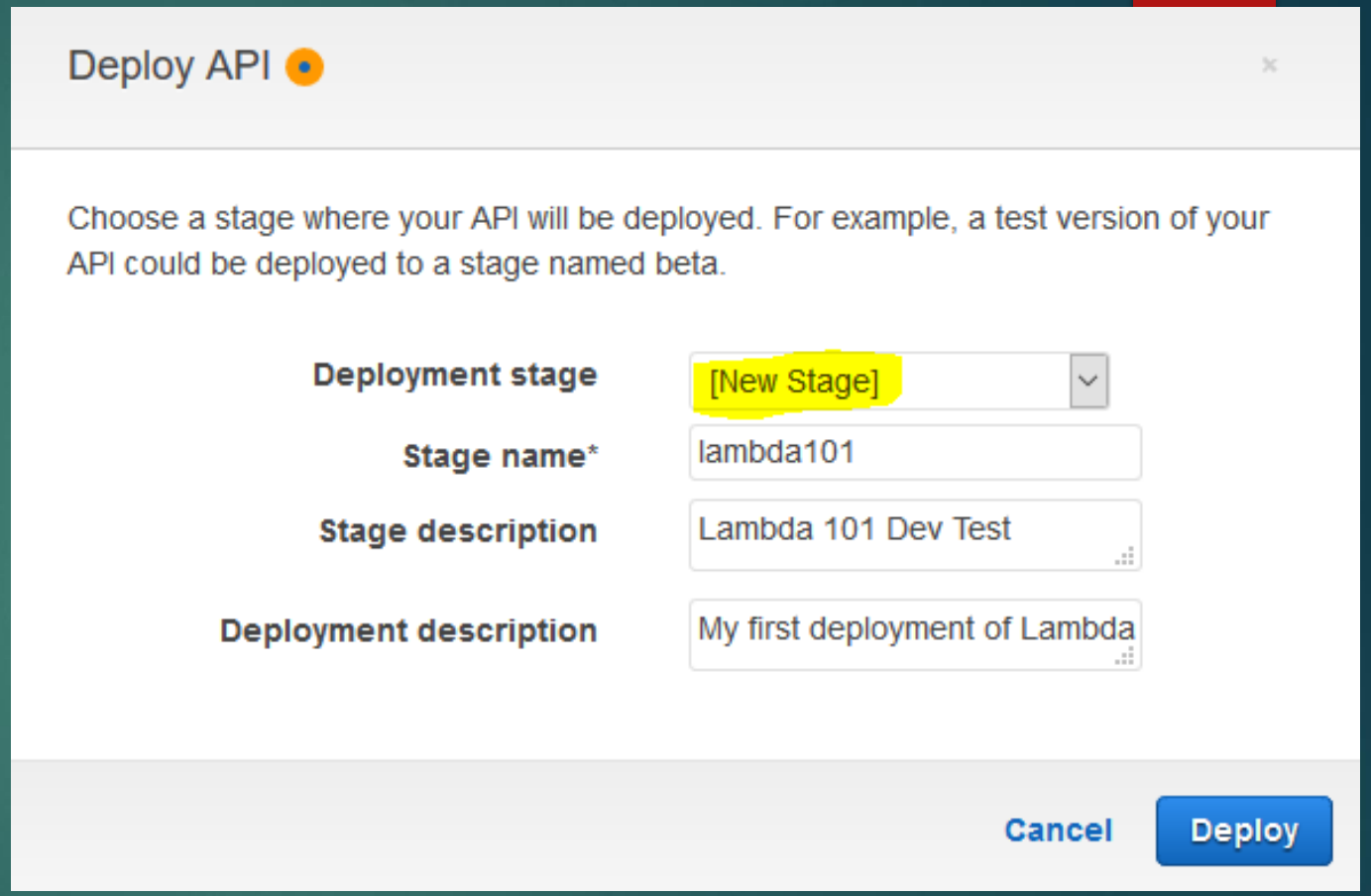
AWS Lambda config
(Integration Request)

```
29          if cursor:
30              cursor.close()
31
32  def lambda_handler(event, context):
33      user_id = event['user_id']
34      return users(user_id)
35
```

users.py

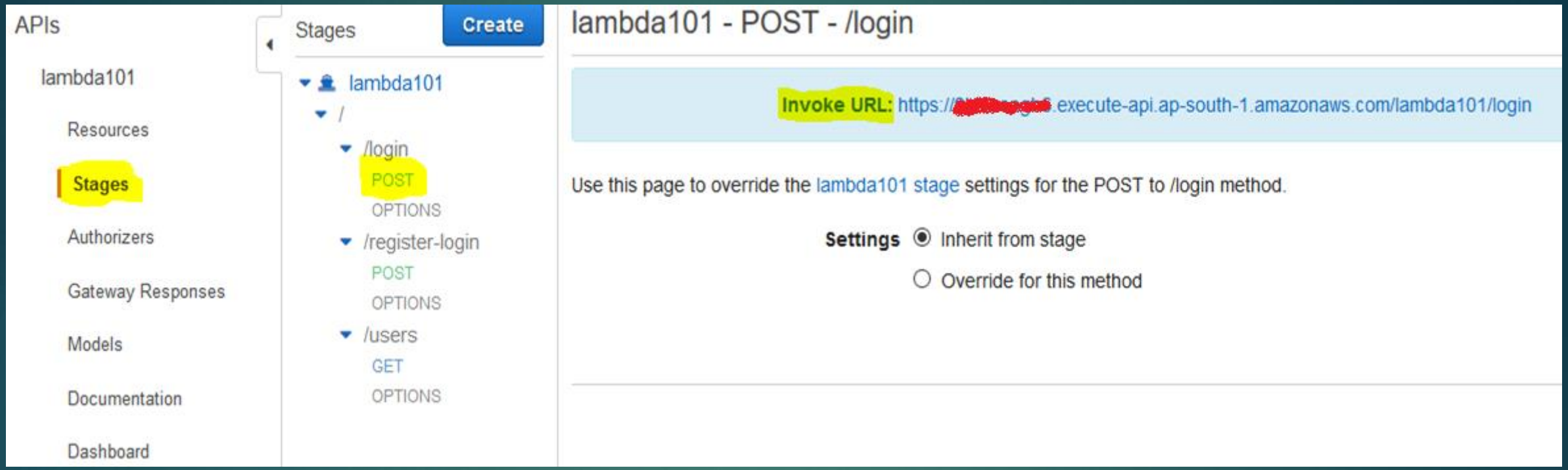
It's time to deploy!





Choose *[New Stage]* and provide appropriate values

Get the deployed API endpoints



- After deployment, the APIs would be available at Stages
- For example, click on **POST** method created for */login* and see the URL
- Similar ones would exist for the **POST** of */register-login* and **GET** of */users*

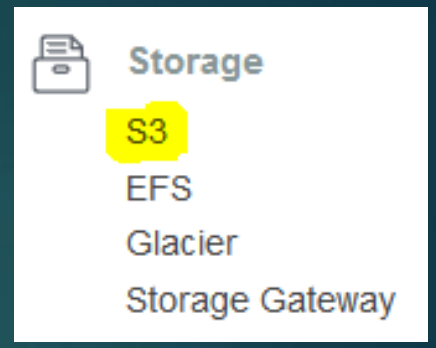
Enable the APIs – Edit the JS functions

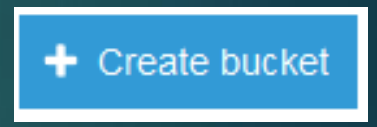
```
/* Login */
function login(auth_details)
{
    var result = null;
    if((auth_details.email) && (auth_details.password))
    {
        $("#error").css('visibility', 'hidden');
        passwordValue = SHA256(auth_details.password)
        login_url = 'https://[redacted].execute-api.ap-south-1.amazonaws.com/lambda101/login';
        var obj = new Object();
        obj.email = auth_details.email;
        obj.password = passwordValue;

        var jsonObj = JSON.stringify(obj);

        $.ajax({
            url: login_url,
            type: 'POST',
            data: jsonObj,
            dataType: 'json',
            success: function(result)
            {
```

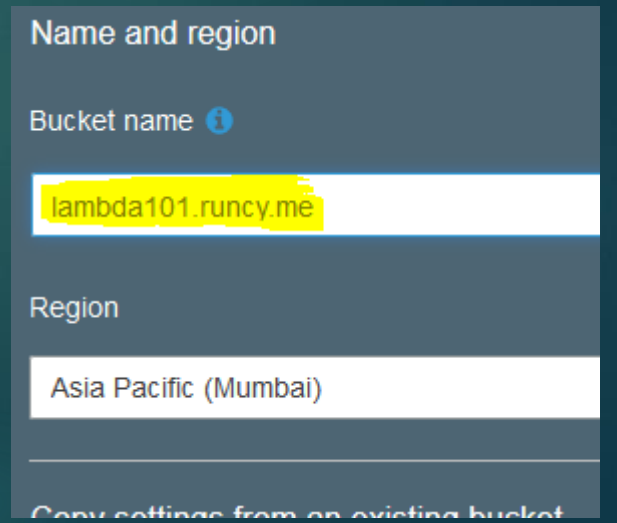
- Integrate each of these APIs with the relevant functions defined in lambda101.js to have them eventually invoked

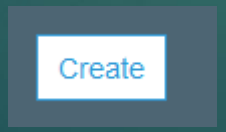
Let's host the web files

- Login to AWS Console
- Select "S3" from Storage
- Click "Create bucket"
- Provide appropriate name (a subdomain or domain that you own host hosting the site)
- Click "Create"



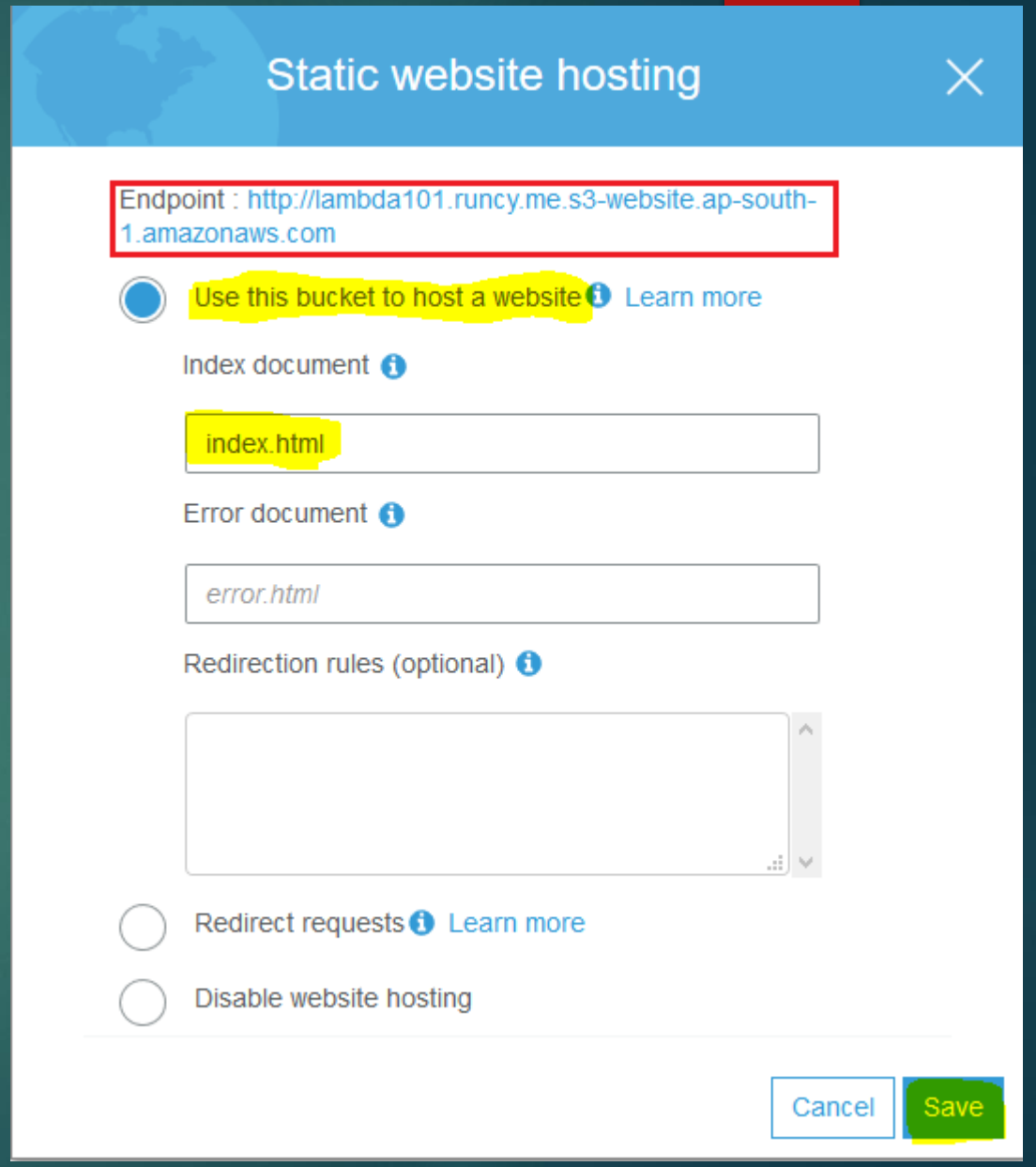






Enable Static Website Hosting

- Select the bucket that you created earlier
- From the "*Properties*" tab select *Static website hosting*
- Provide appropriate *Index document* and hit *Save*
- *You will now see an endpoint available which will serve you the website contents*



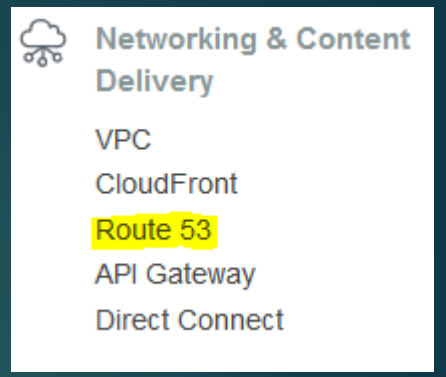
Enable appropriate Bucket Policy

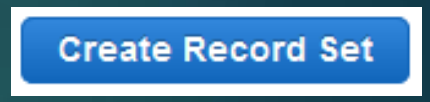
- Click on the "*Permissions*" tab
- Select "Bucket Policy" sub-tab
- *Enter the below policy to make it world readable*

```
{
    "Version": "2012-10-17",        "Statement":
    [
        {
        "Sid": "PublicReadGetObject",
        "Effect": "Allow",
        "Principal": "*",
        "Action": "s3:GetObject",
        "Resource":"arn:aws:s3:::lambda101.runcy.me/*"
        }
    ]
}
```

Let's setup DNS

- Login to AWS Console
- Select "Route 53" from Networking
- Select your *Hosted Zone* for the website*
- Click Create Record Set

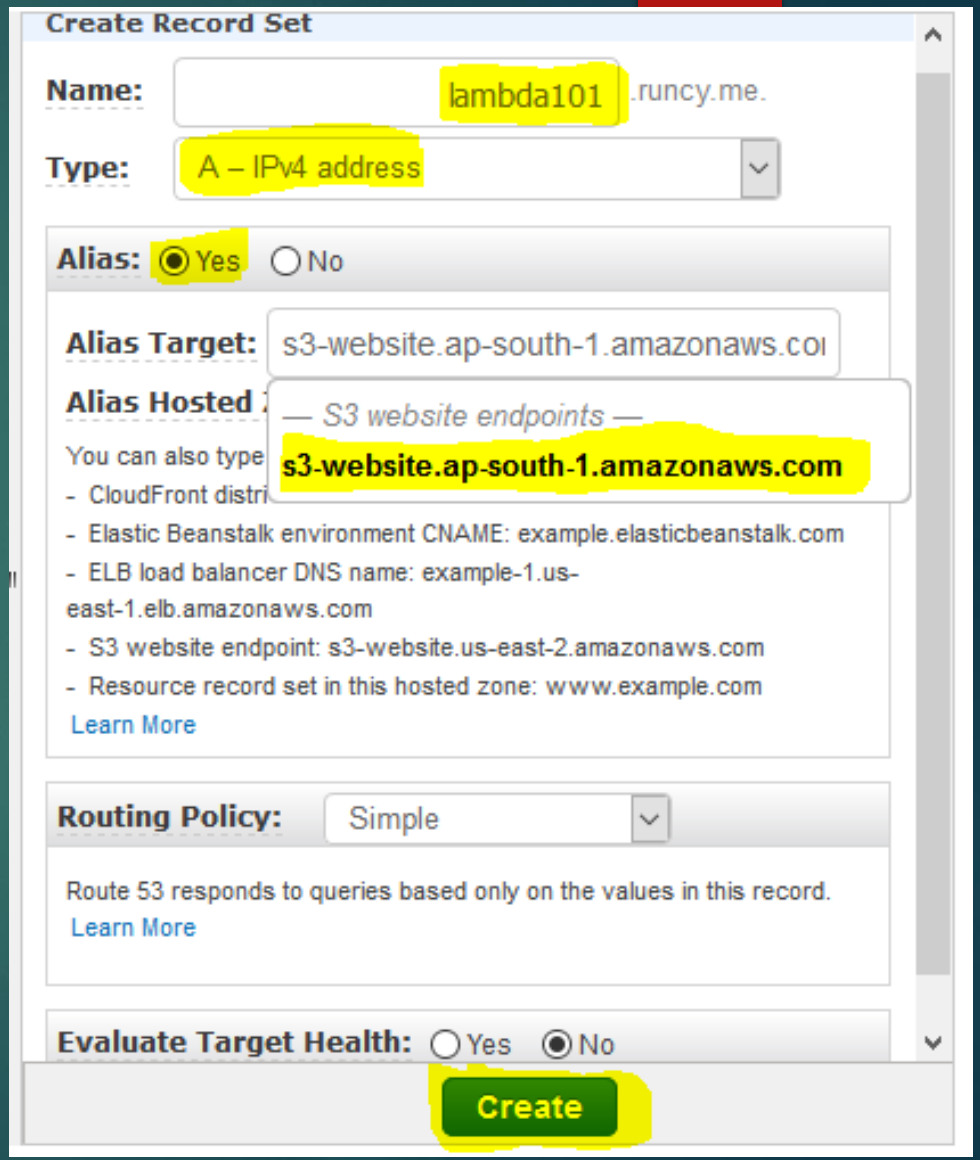




* Assuming you have a website that is managed with Route 53. Setting will vary from provider to provider if using anything else like GoDaddy, Big Rock etc…

Create Record Set

- Provide the subdomain name on which you want the site to be available
- Select Type as “A” record which is an alias to the S3 bucket that was created earlier
- Click Create button
- Wait for sometime for records to propagate (usually 3-4 mins)



Your site is *now* LIVE!!!

Thank You!

Q & A

Runcy Oommen
https://runcy.me