

# Building A Serverless Web App



Runcy Oommen

<https://runcy.me>

# Session Objectives

1. Bring up DBand associated tables with RDS (MySQL)
2. Create and deploy serverless functions with Lambda (Python 3.x)
3. Integration and deployment of these functions with API Gateway
4. Static hosting of web files with S3 bucket
5. Enabling the DNS redirection with Route 53



# Let's run some serverless



# What is serverless?

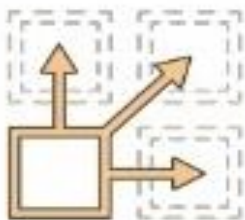
Build and run applications without thinking about servers





“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)

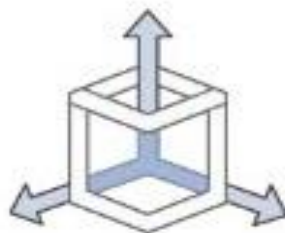
Removes the need for...



Provisioning  
and Utilization



Operations  
and Management



Scaling



Availability and  
Fault Tolerance

Provides these...



Abstraction  
of servers

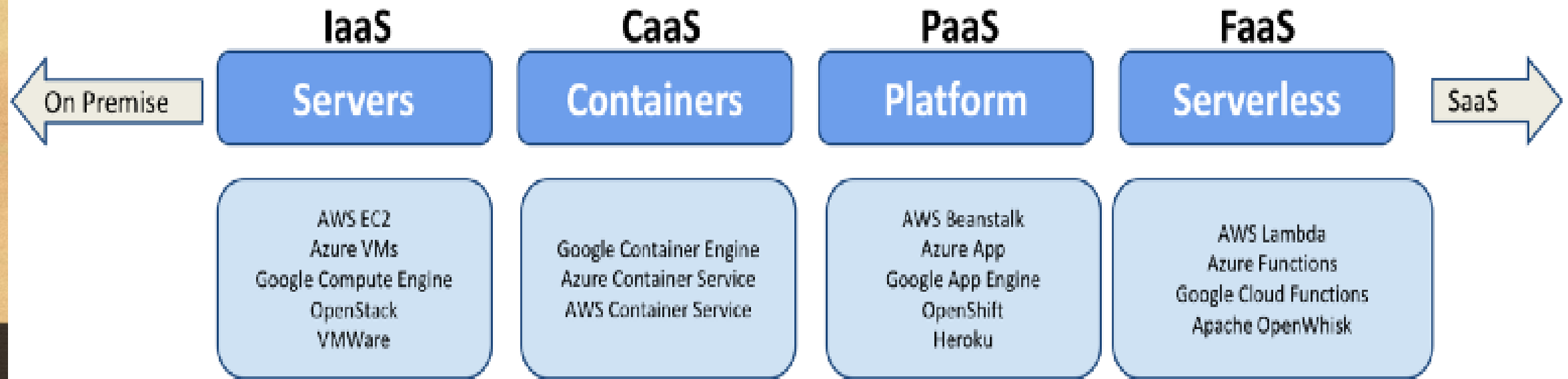


Event-  
driven/  
instant scale



Sub-second  
billing





More control  
Management overhead  
Highly customizable  
Low Velocity  
Low Abstraction

Less control  
No Management  
No customization  
High Velocity  
High Abstraction



SERVERLESS

FAAS



# THiNK Faas

## Programming Model

- Event Driven
- Shares Nothing
- Stateless

## Operational Model

- Zero Ops
- Managed Security
- Auto Scaling

## Billing Model

- Pay for usage
- Cost scales to zero

# 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>*

JUMP OUT...

THINK SERVERLESS!





**What are we building today?**



## Serverless 101 - Login Registration

\*Full Name :

\*Email Address :

\*Password :

\*Location :

\*Comments :

Register

1

Login registration



## Serverless 101 - Login

Email :

Password :

Login

Not a registered user? [Click here to register](#)

2

User login page



India  
CLOUD  
SUMMIT2019

Active Users

User▼

Active Users

Blocked Users

#	Full Name	Email Address	Location	Comments	Action
1	Peter Parker	spidey@marvel.com	New York	Friendly neighborhood guy	🗑️
2	Clark Kent	superman@krypton.com	Smallville	My name is Kal-El	🗑️

### 3 Show active users

India  
CLOUD  
SUMMIT2019

Blocked Users

User▼

Blocked Users

Active Users

#	Full Name	Email Address	Location	Comments	Action
1	Bruce Wayne	batman@dc.com	Gotham City	Kill all jokers	✅
2	Arthur Curry	aquaman@orin.com	Atlantis	King of seven seas	✅

### 4 Show blocked users

**SHUT UP**

**SHOW ME DEMO**



## **Pre-requisites**

- **AWS Free Tier**

- **Source Code**

<https://github.com/roommen/serverless101>

- **Basic knowledge of Python, HTML, JS, CSS**

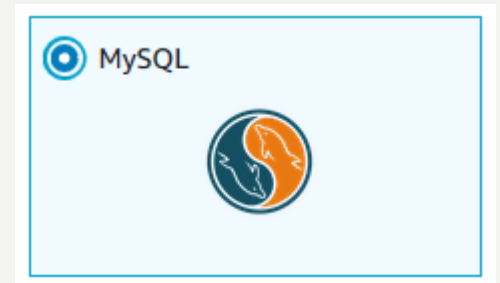
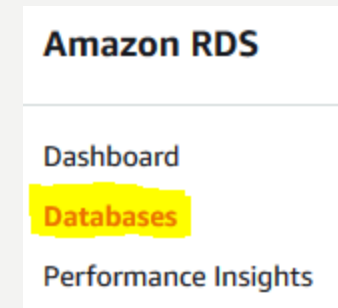
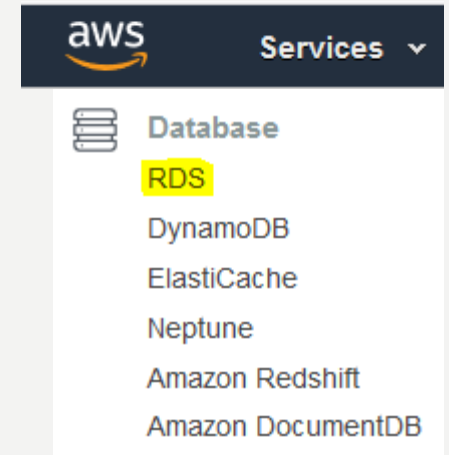
- **A good IDE like Visual Studio Code**

# Let's fire up the DB

- Login to AWS Console
- Select “RDS” from Database category
- Click “Databases” from the left-menu
- Click on “Create database”



- Select “MySQL” as the engine and click “Next”



# MySQL setup

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



- Instance Specifications

DB engine

MySQL Community Edition

License model [Info](#)

general-public-license ▼

DB engine version [Info](#)

MySQL 5.6.40 ▼

DB instance class [info](#)

db.t2.micro — 1 vCPU, 1 GiB RAM ▼

Multi-AZ deployment [info](#)

☐ Create replica in different zone

Creates a replica in a different Availability Zone (AZ) to provide data redundancy, eliminate I/O freezes, and minimize latency spikes during system backups.

☒ No

Storage type [info](#)

General Purpose (SSD) ▼

Allocated storage

20



GB

(Minimum: 20 GB, Maximum: 16384 GB) Higher allocated storage [may improve](#) IOPS performance.



# MySQL DB settings

## Settings

### DB instance identifier [Info](#)

Specify a name that is unique for all DB instances owned by your AWS account in the current region.

DB instance identifier is case insensitive, but stored as all lower-case, as in "mydbinstance". Must contain from 1 to 63 alphanumeric characters or hyphens (1 to 15 for SQL Server). First character must be a letter. Cannot end with a hyphen or contain two consecutive hyphens.

### Master username [Info](#)

Specify an alphanumeric string that defines the login ID for the master user.

Master Username must start with a letter. Must contain 1 to 16 alphanumeric characters.

### Master password [Info](#)

Master Password must be at least eight characters long, as in "mypassword". Can be any printable ASCII character except "/", "", or "@".

### Confirm password [Info](#)

Provide DB instance name and credentials

# MySQL advanced settings - Network & Security

## Network & Security

### Virtual Private Cloud (VPC) [info](#)

VPC defines the virtual networking environment for this DB instance.

Default VPC (vpc-59541030) ▼



Only VPCs with a corresponding DB subnet group are listed.

### Subnet group [info](#)

DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selected.

default ▼

### Public accessibility [info](#)

☒ Yes

EC2 instances and devices outside of the VPC hosting the DB instance will connect to the DB instances. You must also select one or more VPC security groups that specify which EC2 instances and devices can connect to the DB instance.

☐ No

DB instance will not have a public IP address assigned. No EC2 instance or devices outside of the VPC will be able to connect.

### Availability zone [info](#)

No preference ▼

### VPC security groups

Security groups have rules authorizing connections from all the EC2 instances and devices that need to access the DB instance.

☒ Create new VPC security group

☐ Choose existing VPC security groups

Keep everything as  
the default setting

# MySQL advanced settings – Database options

## Database options

Database name

Info

serverless101

Note: if no database name is specified then no initial MySQL database will be created on the DB Instance.

Port

Info

TCP/IP port the DB instance will use for application connections.

3306

DB parameter group

Info

default.mysql5.6

Option group

Info

default:mysql-5-6

IAM DB authentication

Info

☐ Enable IAM DB authentication

Manage your database user credentials through AWS IAM users and roles.

☒ Disable


Provide appropriate  
DB name

# MySQL advanced settings – Encryption & Backup


## Encryption

### Encryption

- ☐ Enable Encryption  
Select to encrypt the given instance. Master key ids and aliases appear in the list after they have been created using the Key Management Service(KMS) console. [Learn More](#).
- ☒ Disable Encryption

 The selected engine or DB instance class does not support storage encryption.

## Backup

 Please note that automated backups are currently supported for InnoDB storage engine only. If you are using MyISAM, refer to detail [here](#).

### Backup retention period [info](#)

Select the number of days that Amazon RDS should retain automatic backups of this DB instance.

7 days ▼

### Backup window [info](#)

- ☐ Select window
- ☒ No preference

Leave everything as default

# MySQL advanced settings – Monitoring, Log, Maintenance

### Monitoring

Enhanced monitoring

☐ Enable enhanced monitoring  
Enhanced monitoring metrics are useful when you want to see how different processes or threads use the CPU.

☒ Disable enhanced monitoring

### Log exports

Select the log types to publish to Amazon CloudWatch Logs

☐ Audit log  
☐ Error log  
☐ General log  
☐ Slow query log

IAM role  
The following service-linked role is used for publishing logs to CloudWatch Logs.

RDS Service Linked Role

### Maintenance

Auto minor version upgrade [info](#)

☒ Enable auto minor version upgrade  
Enables automatic upgrades to new minor versions as they are released. The automatic upgrades occur during the maintenance window for the DB instance.

☐ Disable auto minor version upgrade

Maintenance window [info](#)  
Select the period in which you want pending modifications or patches applied to the DB instance by Amazon RDS.

☐ Select window  
☒ No preference

Cancel

Previous

Create database

- Leave everything as default
- Click on “Create database”



# MySQL getting initialized



Your DB instance is being created.

Note: Your instance may take a few minutes to launch.

## Connecting to your DB instance

Once Amazon RDS finishes provisioning your DB instance, you can use a SQL client application or utility to connect to the instance.

[Learn about connecting to your DB instance](#)

[All DB instances](#)

[View DB instance details](#)

- It may take sometime for DB to be initialized and available depending on region

# MySQL Endpoint

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

RDS > Databases > serverless101

serverless101

Modify

Actions ▼

## Summary

DB Name

serverless101

CPU

1.48%

Info

Available

Class

db.t2.micro

Role

Instance

Current activity

2 Connections

Engine

MySQL

Region & AZ

ap-south-1b

## Endpoint & port

Endpoint

serverless101.cemnrzna330w.ap-south-1.rds.amazonaws.com

Port

3306

## Security group rules (2)

Filter security group rules

< 1 > ⚙

Security group

Type

Rule

rds-launch-wizard-1 (sg-c4c766af)

CIDR/IP - Inbound

115.160.251.210/32

rds-launch-wizard-1 (sg-c4c766af)

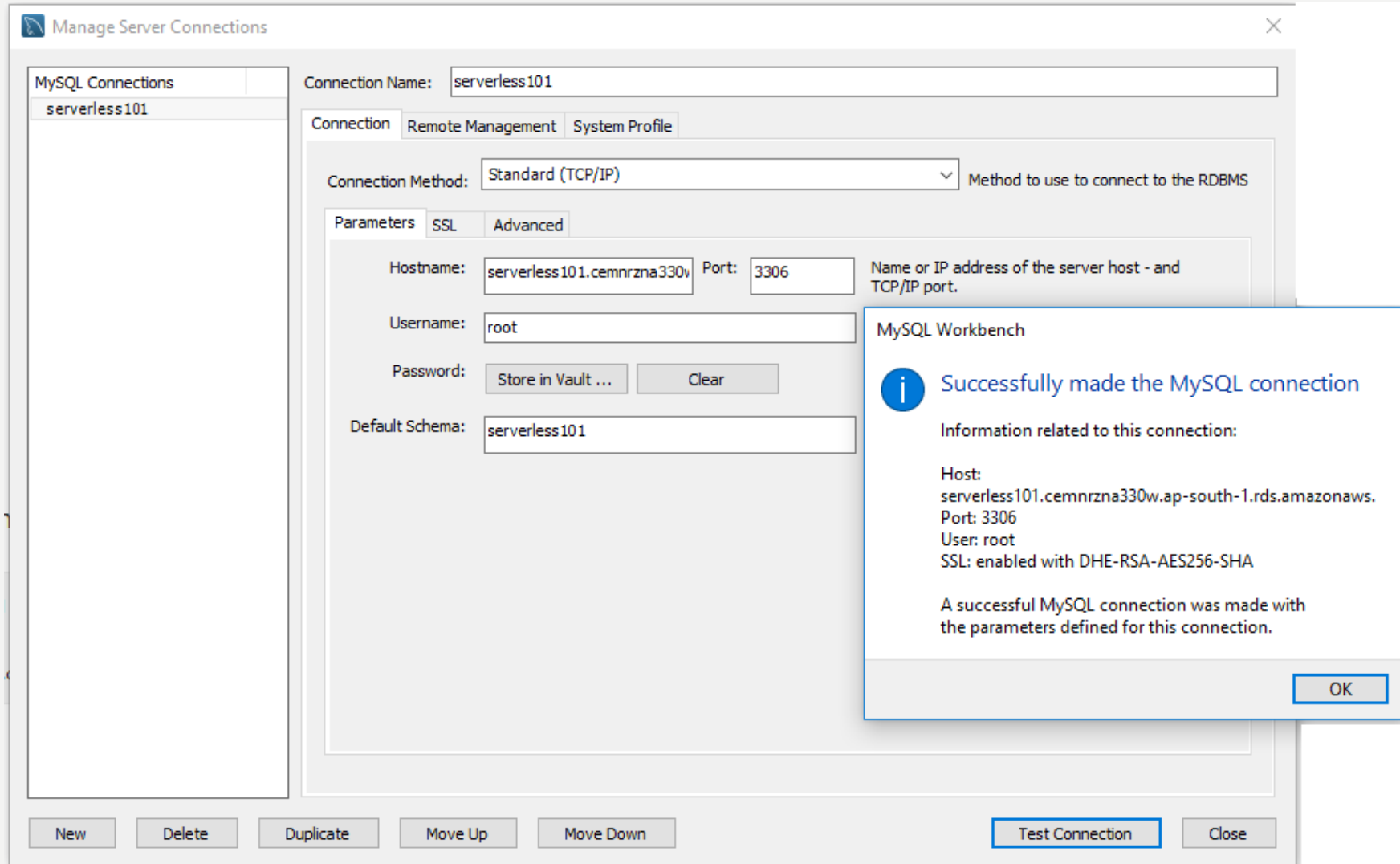
CIDR/IP - Outbound

0.0.0.0/0

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..



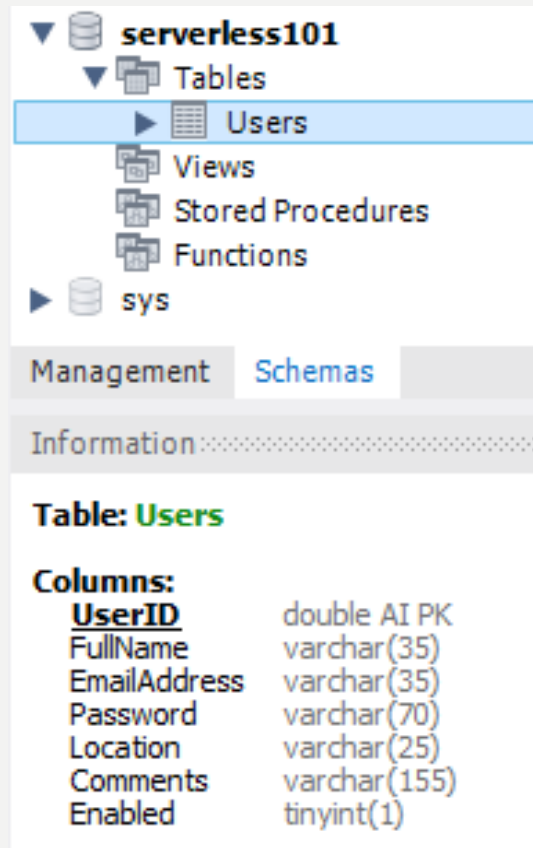
# Creating Users table

- Go to the cloned `serverless101` repository location
- Navigate to the “dbscripts” folder
- Edit the ‘CreateTableUsers.py’ file with the DB info you created earlier

```
1  import mysql.connector
2
3  def create_users():
4      connection, cursor = None, None
5      try:
6          # Database connection parameters -- replace this with your DB endpoint
7          serverless101cnxstr = {'host': 'serverless101.cemnrzna330w.ap-south-1.rds.amazonaws.com', 'user': 'root', '\
8          'password': 'password', 'database': 'serverless101'}
9          connection = mysql.connector.connect(host=serverless101cnxstr['host'], user=serverless101cnxstr['user'], \
10         password=serverless101cnxstr['password'], database=serverless101cnxstr['database'])
11         cursor = connection.cursor()
12         cursor.execute('CREATE TABLE Users (UserID DOUBLE NOT NULL AUTO_INCREMENT PRIMARY KEY, \
13         FullName VARCHAR(35) NOT NULL, EmailAddress VARCHAR(35) NOT NULL, Password VARCHAR(70) NOT NULL, \
14         Location VARCHAR(25) NOT NULL, Comments VARCHAR(155) NOT NULL, Enabled BOOLEAN NOT NULL);')
15         print("Table Users created successfully.")
16     except mysql.connector.Error as err:
17         print(err)
18     finally:
19         if connection:
20             connection.close()
21         if cursor:
22             cursor.close()
23
24
25 if __name__ == '__main__':
26     create_users()
```

# Run the CreateTableUsers.py file

```
runcy@runcyoommen-PC:/mnt/f/serverless101/dbscripts$ python3 CreateTableUsers.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:
  - Login Registration – handled by *serverless/loginregister.py*
  - User Login – handled by *serverless/login.py*
  - Active Users – handled by *serverless/activeusers.py*
  - Blocked Users – handled by *serverless/blockedusers.py*
  - Allow User – handled by *serverless/allowuser.py*
  - Block User – handled by *serverless/blockuser.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 have *mysql.connector* as a dependency in each of these files

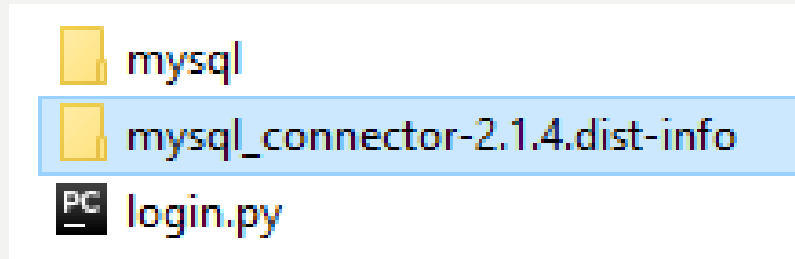
# AWS Lambda with Python – Extract dependencies

- Create a temp folder called *login* and copy *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)

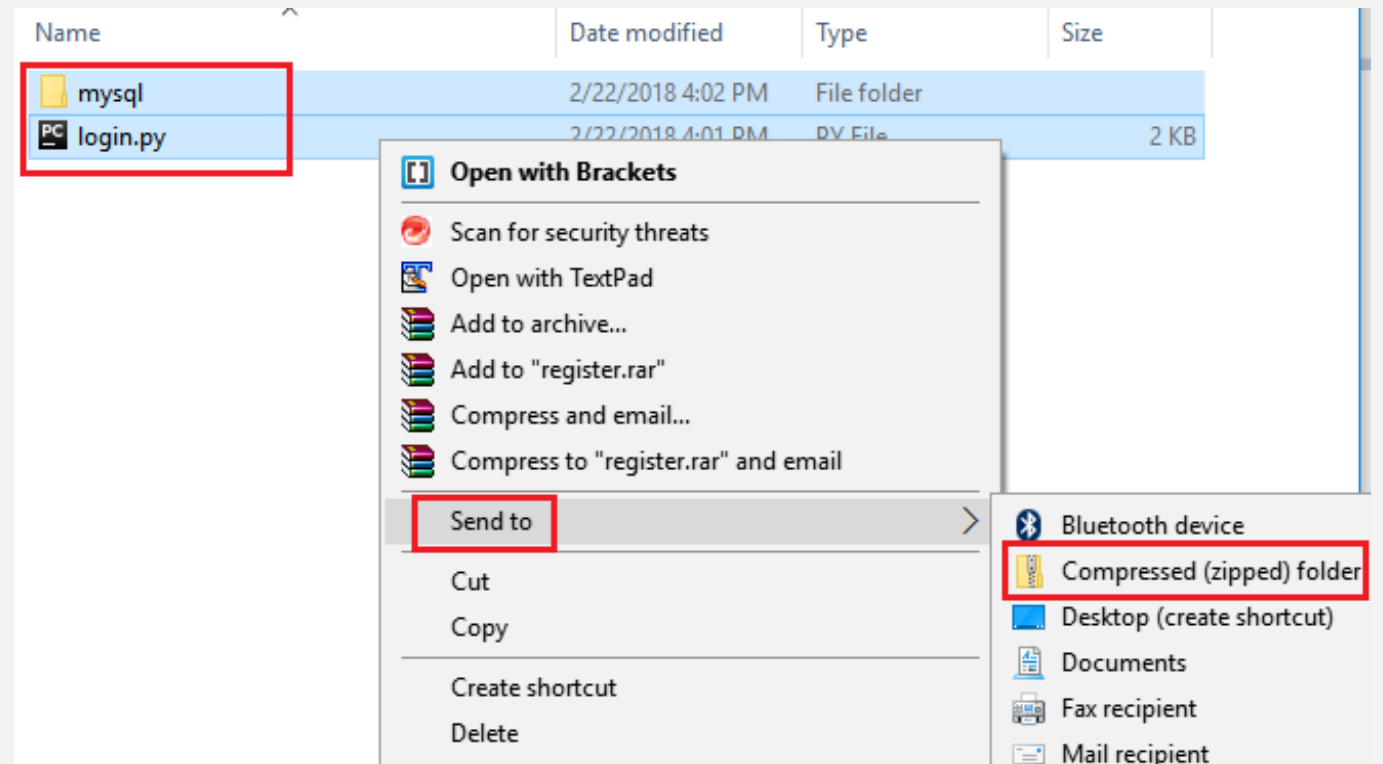
```
runcy@runcyoommen-PC:/mnt/f/serverless101/serverless$ mkdir login
runcy@runcyoommen-PC:/mnt/f/serverless101/serverless$ cd login/
runcy@runcyoommen-PC:/mnt/f/serverless101/serverless/login$ cp ../login.py .
runcy@runcyoommen-PC:/mnt/f/serverless101/serverless/login$ pip3 install mysql-connector==2.1.4 --target .
Collecting mysql-connector==2.1.4
  Downloading https://files.pythonhosted.org/packages/53/61/26b0bc2655ad64d550565252baf83611fe9db7d98a3c571950b2b829ffa7/mysql-connector-2.1.4.zip (355kB)
    100% |████████████████████████████████████████| 358kB 520kB/s
Building wheels for collected packages: mysql-connector
  Running setup.py bdist_wheel for mysql-connector ... done
  Stored in directory: /home/runcy/.cache/pip/wheels/1a/db/29/c7d096eaa31cc71ac259b9183a91ddf62bfd293cf52cfd3dc
Successfully built mysql-connector
```

# AWS Lambda with Python – Zip ‘em up

- Under the *login* folder, you might see a folder *mysql\_connector-2.1.4.dist-info* which can be deleted if you want to



- Select the rest (*login.py* **file** and the *mysql* **folder**) and extract it to a zip file by right-clicking on it

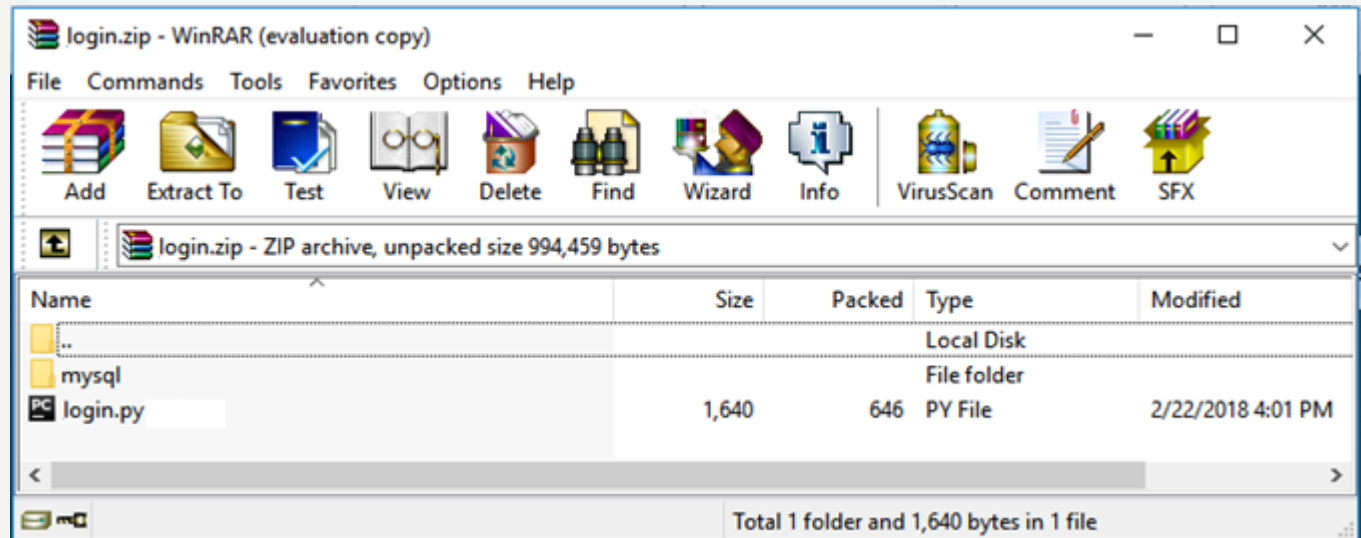


# AWS Lambda with Python – Zip file details



- You should now have a login.zip file created
- Verify the contents of this zip file and ensure that the contents look identical to screenshot below

PS: The *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 the remaining files:

1. *loginregister.py* 2. *activeusers.py* 3. *blockedusers.py* 4. *allowuser.py* 5. *blockuser.py*



# Fret not, automation to the rescue!

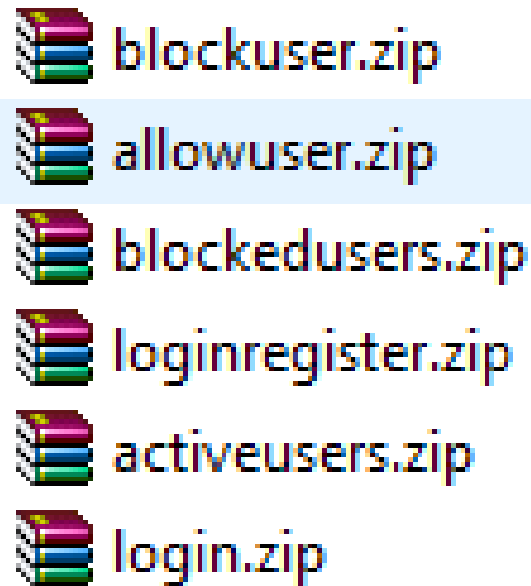
- Go to the cloned *serverless/01* repository location
- Navigate to the “serverless” folder
- Edit the ‘create\_serverless.py’ file with the required filenames

```
1  import os
2
3  files = ["login.py", "activeusers.py", "blockedusers.py", "loginregister.py", \
4  |...|... "allowuser.py", "blockuser.py"]
5
6  try:
7  |...|# remove mysql unzipped folder if exist
8  |...|os.system("rm -rf mysql")
9  |...|# unzip mysql
10 |...|os.system("unzip mysql")
11
12 |...|for file in files:
13 |...|... temp = file.split(".")[0]
14 |...|...|# remove root zipped folder if exist
15 |...|...|os.system("rm -rf " + temp)
16 |...|...|# remove existing folders
17 |...|...|os.system("rm -rf " + temp + "; rm -rf " + temp + ".zip")
18 |...|...|# create root folder
19 |...|...|os.makedirs(temp)
20 |...|...|# copy mysql folders to root
21 |...|...|os.system('cp -a mysql ' + temp + '; cp ' + file + ' ' + temp)
22 |...|...|# move to root folder and zip contents
23 |...|...|os.system('cd ' + temp + '; zip -r ' + temp + '.zip *; mv ' + temp + '.zip ../')
24 |...|...|# remove root folder
25 |...|...|os.system("rm -rf " + temp)
26 |...|...|# remove mysql unzipped folder if exist
27 |...|...|os.system("rm -rf mysql")
28 except Exception as e:
29 |...|print(e)
30
```

## Run the create\_serverless.py file

```
runcy@runcyoommen-PC:/mnt/f/serverless101/serverless$ python3 create_serverless.py
```

- All the respective .zip files with all dependencies will now be created at one shot!

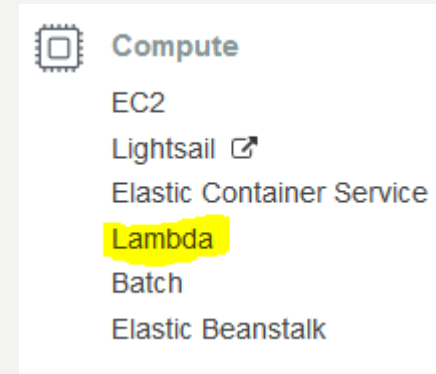


- blockuser.zip
- allowuser.zip
- blockedusers.zip
- loginregister.zip
- activeusers.zip
- login.zip

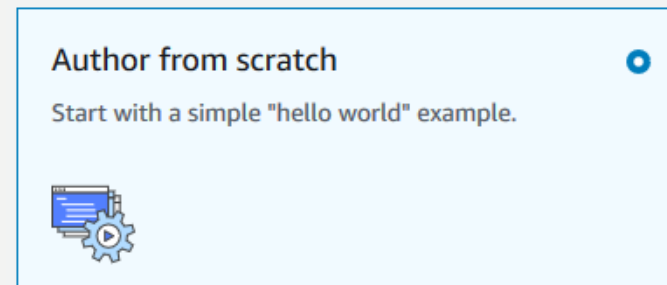


# Let's create the Lambda functions

- Select “Lambda” from Compute category
- Click “Create function”
- Select “Author from scratch”



**Create function**



# Login Registration - Lambda function creation

## Author from scratch [Info](#)

Name

serverless101-loginregister

Runtime

You can select a supported AWS Lambda runtime or provide your own runtime as part of the function deployment package or Lambda layer after creating the function.

Python 3.6

Role

Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. [Learn more](#) about Lambda execution roles.

Choose an existing role

Existing role

You can use an existing role with this function. Lambda must be able to assume this role, and the role must have Amazon CloudWatch Logs permissions.

lambda\_basic\_execution

# Login Registration - Lambda function code


- In the next screen, upload the zip file created earlier (*loginregister.zip*) and change the Handler info to *loginregister.lambda\_handler*
- The format of the Handler should be *<python\_filename>.lambda\_handler*

**Function code** [Info](#)

Code entry type Runtime Handler [Info](#)

Upload a .zip file Python 3.6 loginregister.lambda\_handler

Function package

 Upload loginregister.zip (331.1 kB)

For files larger than 10 MB, consider uploading using Amazon S3.

Save

- Once done, click “Save”
- Do this for each of the remaining zip files to create lambda functions for *login*, *activeusers*, *blockedusers*, *allowuser* and *blockuser* functionality

# Lambda functions - Created

	Function name ▼	Description	Runtime ▼	Code size ▼	Last modified
<input type="radio"/>	<a href="#">serverless101-blockuser</a>	Serverless 101 - Block User	Python 3.6	330.9 kB	2 days ago
<input type="radio"/>	<a href="#">serverless101-allowuser</a>	Serverless 101 - Allow User	Python 3.6	330.9 kB	2 days ago
<input type="radio"/>	<a href="#">serverless101-login</a>	Serverless 101 - Login	Python 3.6	326.7 kB	2 days ago
<input type="radio"/>	<a href="#">serverless101-loginregister</a>	Serverless 101 - Login Register	Python 3.6	326.8 kB	2 days ago
<input type="radio"/>	<a href="#">serverless101-blockedusers</a>	Serverless 101 - Blocked Users	Python 3.6	326.7 kB	4 days ago
<input type="radio"/>	<a href="#">serverless101-activeusers</a>	Serverless 101 - Active Users	Python 3.6	326.7 kB	4 days ago

Once done, you should have six lambda functions created for the app

# Integration with API Gateway

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



Networking & Content  
Delivery

VPC

CloudFront

Route 53

API Gateway

Direct Connect

## Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

☒ REST ☐ WebSocket

## Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

☒ New API ☐ Clone from existing API ☐ Import from Swagger or Open API 3 ☐ Example API

## Settings

Choose a friendly name and description for your API.

API name\*

serverless101

Description

Serverless101 Web App

Endpoint Type

Regional



# API Gateway – Create Resource (loginregister)

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

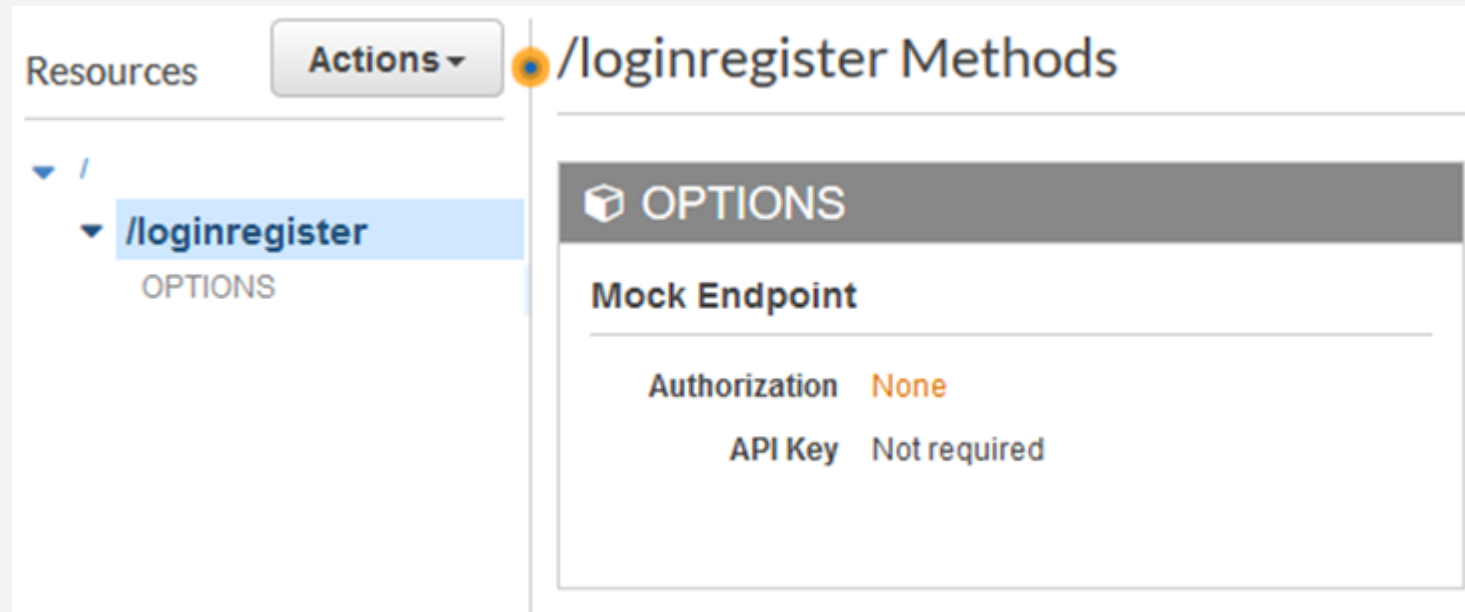
The screenshot shows the AWS API Gateway console interface. On the left, a sidebar lists navigation options: APIs, Resources, Stages, Authorizers, Gateway Responses, Models, Resource Policy, Documentation, Dashboard, and Settings. The 'APIs' section is active, showing a list of APIs with 'serverless101' highlighted. The 'Resources' tab is selected for 'serverless101', showing a tree view with a root resource '/'. The 'Actions' dropdown menu is open, and 'Create Resource' is highlighted. The main panel is titled 'New Child Resource' and contains the following fields and options:

- Resource Name\***: A text input field containing 'loginregister'.
- Resource Path\***: A text input field containing '/ loginregister'.
- Enable API Gateway CORS**: A checkbox that is checked.
- proxy resource**: A checkbox that is unchecked.

Below the fields, there is a text block explaining path parameters: "You can add path parameters using brackets. For example, the resource path {username} represents a path parameter called 'username'. Configuring /{proxy+} as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource."

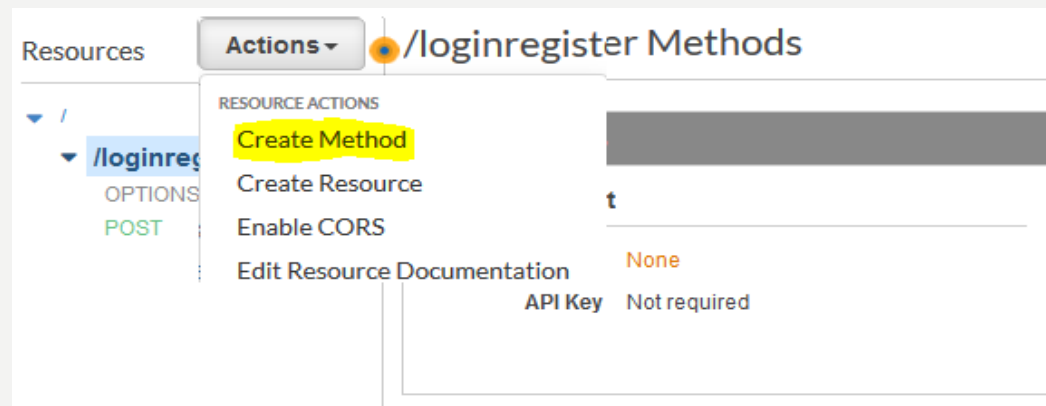
At the bottom of the form, there is a legend indicating that an asterisk (\*) denotes a required field. On the right side, there are two buttons: 'Cancel' and 'Create Resource'.

## API Gateway – Resource created (loginregister)



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

## API Gateway – Create Method

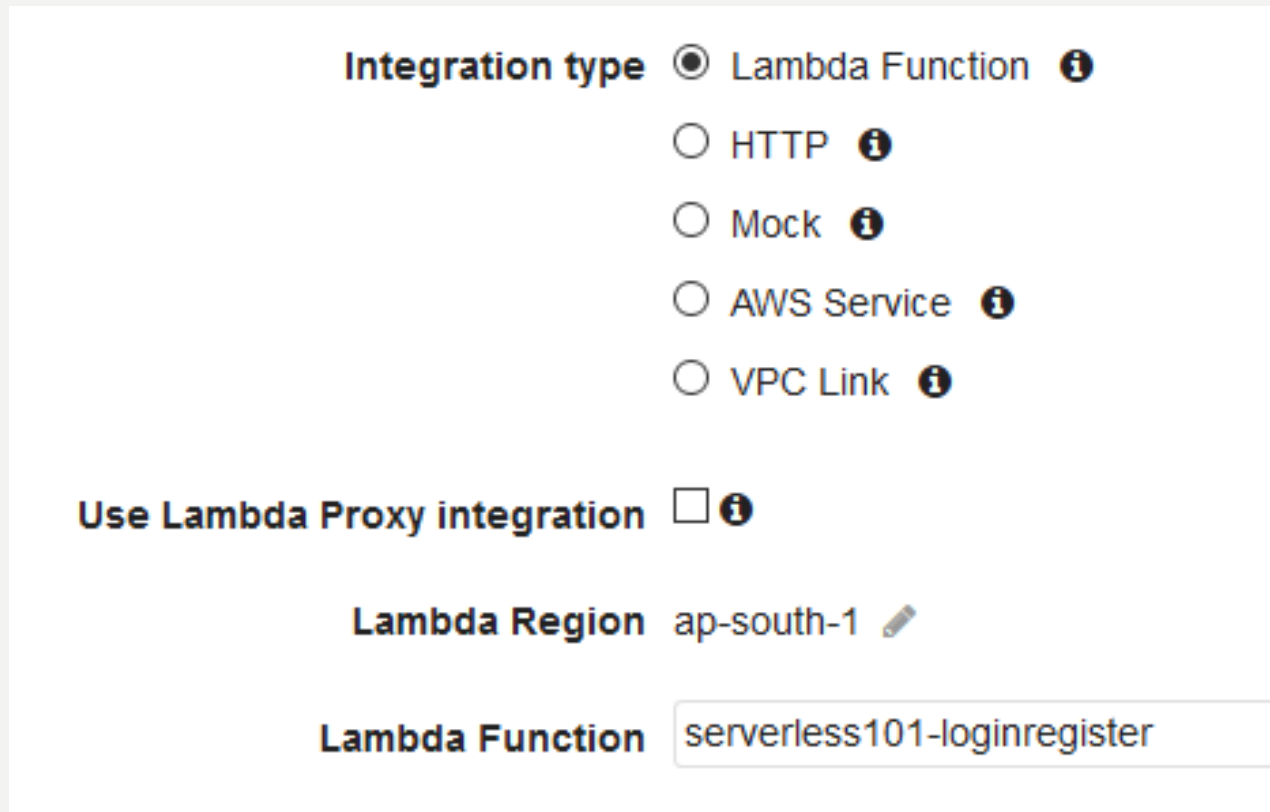


- Select the resource and now click “Create Method”
- Choose “POST”



# API Gateway – Configure POST (loginregister)

Click on the “POST” method and enter the configuration as below



The screenshot shows the configuration interface for a POST method in AWS API Gateway. It includes radio buttons for integration types, a checkbox for Lambda Proxy integration, a dropdown for the Lambda region, and a text input for the Lambda function name.

**Integration type** ☒ Lambda Function ⓘ  
☐ HTTP ⓘ  
☐ Mock ⓘ  
☐ AWS Service ⓘ  
☐ VPC Link ⓘ

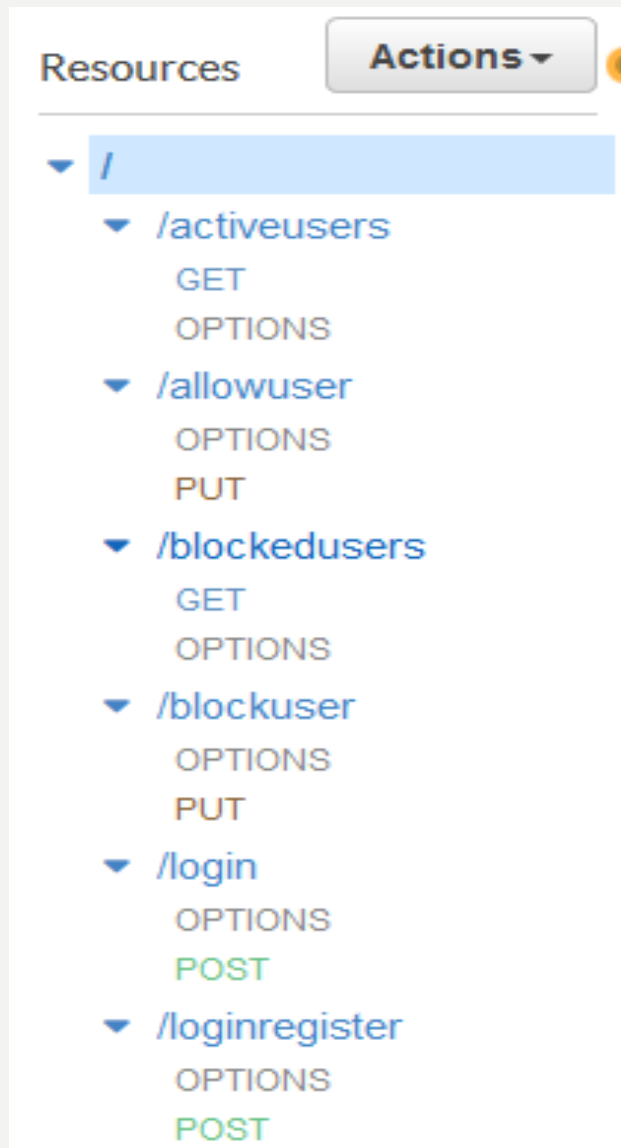
**Use Lambda Proxy integration** ☐ ⓘ

**Lambda Region** ap-south-1 ✎

**Lambda Function**

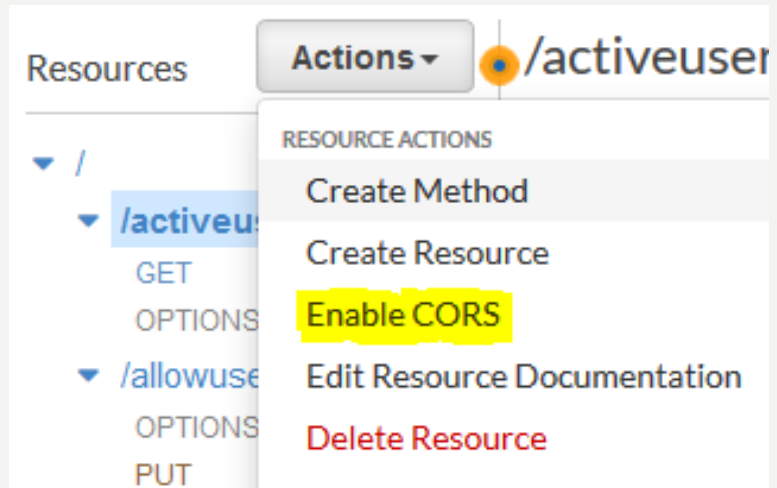
Select the appropriate region to choose the lambda function which we had created earlier

# API Gateway – Create remaining resources & methods



- Create *login* resource; associate **POST** method
- Create *activeusers* and *blockedusers* resources; associate **GET** method
- Create *allowuser* and *blockuser* resources; associate **PUT** method
- Follow identical steps as the previous *loginregister* for lambda configuration and integration

# Enable CORS



- Select a method and click “Enable CORS”
- On the next screen, leave everything as is and click “Enable CORS and replace existing headers”

## Enable CORS

**Gateway Responses for** *serverless101 API* ☐ DEFAULT 4XX ☐ DEFAULT 5XX ⓘ

**Methods** ☒ GET ☒ OPTIONS ⓘ

**Access-Control-Allow-Methods** GET, OPTIONS ⓘ


**Access-Control-Allow-Headers** 'Content-Type,X-Amz-Date,Authorizatio ⓘ

**Access-Control-Allow-Origin\*** ⓘ

► Advanced

**Enable CORS and replace existing CORS headers**


# It's time to deploy!

**Actions**  [← Method Exe](#)

**METHOD ACTIONS**  
[Edit Method Documentation](#)  
[Delete Method](#)


**RESOURCE ACTIONS**  
[Create Method](#)  
[Create Resource](#)  
[Enable CORS](#)  
[Edit Resource Documentation](#)  
[Delete Resource](#)

**API ACTIONS**  
**Deploy API**  
[Import API](#)  
[Edit API Documentation](#)  
[Delete API](#)

Deploy API 

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.


Deployment stage

[New Stage] 


Stage name\*

serverless101

Stage description

Serverless 101 web app 

Deployment description



[Cancel](#) [Deploy](#)

Choose *[New Stage]* and provide appropriate values

# Get the deployed API endpoints

The screenshot displays the AWS API Gateway console. On the left, the 'Stages' sidebar shows a tree view with 'serverless101' expanded, and '/loginregister' selected, with its 'POST' method highlighted. The main panel shows the configuration for 'serverless101 - POST - /loginregister'. A light blue box contains the 'Invoke URL: https://jthp9bhj27.execute-api.ap-south-1.amazonaws.com/serverless101/loginregister'. Below this, a message states: 'Use this page to override the serverless101 stage settings for the POST to /loginregister method.' The 'Settings' section has two radio buttons: 'Inherit from stage' (selected) and 'Override for this method'. A 'Save Changes' button is located at the bottom right of the main panel.

- 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*

# loginregister.html, loginregister() - serverless101.js, loginregister.py

```
<div class="row">
→ <div class="col-sm-4"></div>
→ <div class="col-sm-4" style="margin-left: 10px;>
→   <button type="button" class="btn btn-primary" style="width: 100%;">
→   login
→   onclick="loginregister({email: email.value, location: location.value,
→   comments: comments.value})"
→ </div>
→ <div class="col-sm-4"></div>
</div>
```

loginregister.html

**SERVERLESS  
TRINITY**

```
/* Login Register */
function loginregister(loginregister) {
→ if((loginregister.email) && (loginregister.password)) {
→ {
→     passwordValue = sha256(loginregister.password);
→     //API Endpoint -- Replace this with endpoint.yourid
→     loginregisterurl = 'https://jthp9bhj27.execute-api.us-east-1.amazonaws.com';
→     var obj = new Object();
→     obj.fullname = loginregister.fullname;
→     obj.email = loginregister.email;
→     obj.password = passwordValue;
→     obj.location = loginregister.location;
→     obj.comments = loginregister.comments;
→     axios.post(loginregisterurl, obj)
→     .then(function(response) {
→         console.log(response);
→     })
→     .catch(function(error) {
→         console.log(error);
→     });
→ }
→ }
}
```

loginregister() – serverless101.js

```
def loginregister(fullname, email, password):
→ try:
→     # Database connection parameters
→     serverless101cnxstr = {'host': 'localhost', 'port': 3306, 'user': 'root', 'password': 'root'}
→     connection = mysql.connector.connect(**serverless101cnxstr)
→     # Check if email already exists
→     sql = "SELECT UserID FROM User WHERE Email = %s"
→     cursor = connection.cursor()
→     cursor.execute(sql)
→     userid = cursor.fetchall()
→     if userid:
→         return {"result": False}
→     else:
→         # Insert new user
→         sql = "INSERT INTO User (Fullname, Email, Password, Location, Comments) VALUES (%s, %s, %s, %s, %s)"
→         cursor.execute(sql, (fullname, email, password, location, comments))
→         connection.commit()
→         return {"result": True}
→ except Exception as e:
→     print(e)
```

loginregister.py

# Enable the APIs – Edit the JS functions

```
/*-Login-Register-*/
function loginregister(loginregister) {
  if((loginregister.email) && (loginregister.password) && (loginregister.fullname) && (loginregister.comments) && (1
  {
    passwordValue = sha256(loginregister.password)
    //API-Endpoint -- Replace this with endpoint you created
    loginregisterurl = 'https://jthp9bhj27.execute-api.ap-south-1.amazonaws.com/serverless101/loginregister';
    var obj = new Object();
    obj.fullname = loginregister.fullname;
    obj.email = loginregister.email;
    obj.password = passwordValue;
    obj.location = loginregister.location;
    obj.comments = loginregister.comments;

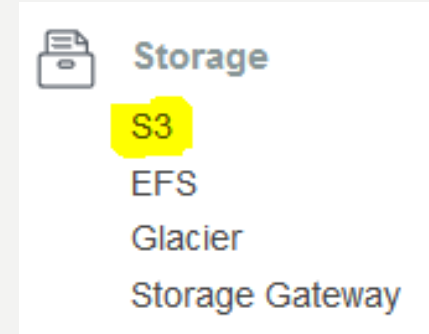
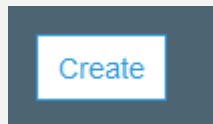
    var jsonObj = JSON.stringify(obj);
    $.ajax({
      url: loginregisterurl,
      headers: {"Content-Type": "application/json"},
      type: 'POST',
      data: jsonObj,
      dataType: 'json',
      success: function(resp)
      {
        loginregistersuccess = resp['result'];
        if(loginregistersuccess === true){
```

- Integrate each of these APIs with the relevant functions defined in serverless101.js to have them eventually invoked



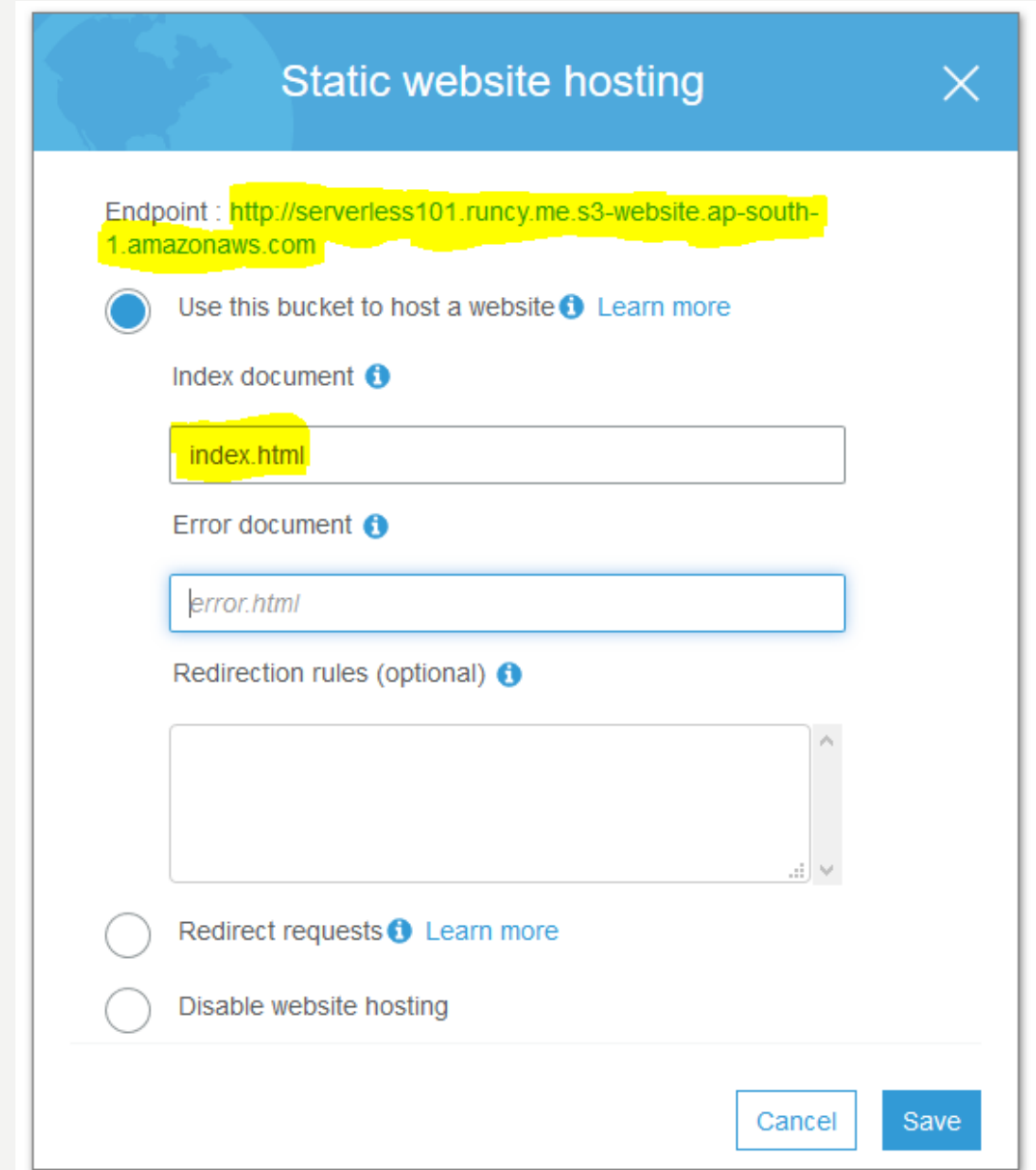
# Let's host the web files

- Select “S3” from Storage category
- Click “Create bucket”
- Provide appropriate name (a subdomain or domain that you own for host hosting the site)
- Click “Create”

A screenshot of the "Name and region" step in the AWS S3 bucket creation wizard. The form has a blue header with a circled "1" and the text "Name and region". Below the header, there are two sections: "Name and region" and "Bucket name". The "Bucket name" section has a text input field containing "serverless101.runcy.me". Below that, there is a "Region" section with a dropdown menu showing "Asia Pacific (Mumbai)".

# 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



The screenshot shows the 'Static website hosting' configuration window in the AWS console. The window has a blue header with the title 'Static website hosting' and a close button. The main content area is white. At the top, the 'Endpoint' is displayed as 'http://serverless101.runcy.me.s3-website.ap-south-1.amazonaws.com'. Below this, there is a radio button selected for 'Use this bucket to host a website', with a 'Learn more' link. Underneath, the 'Index document' field is set to 'index.html' and the 'Error document' field is set to 'error.html'. There is a section for 'Redirection rules (optional)' which is currently empty. At the bottom, there are two radio buttons: 'Redirect requests' (selected) and 'Disable website hosting', both with 'Learn more' links. The window concludes with 'Cancel' and 'Save' buttons.

Static website hosting

Endpoint : `http://serverless101.runcy.me.s3-website.ap-south-1.amazonaws.com`

☒ Use this bucket to host a website [Learn more](#)

Index document [i](#)

`index.html`

Error document [i](#)

`error.html`

Redirection rules (optional) [i](#)

☐ Redirect requests [Learn more](#)

☐ Disable website hosting

Cancel Save

# Enable appropriate Bucket Policy

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

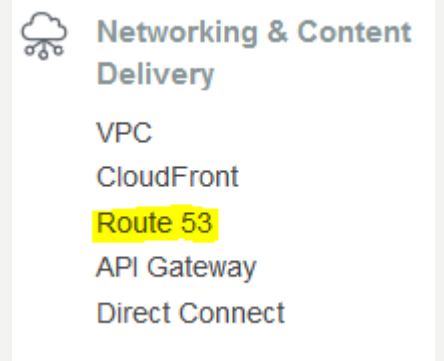
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::serverless101.runcy.me/*"
    }
  ]
}
```

The screenshot shows the AWS IAM console's 'Bucket Policy editor' for the bucket 'arn:aws:s3:::serverless101.runcy.me'. At the top, there are four tabs: 'Public access settings', 'Access Control List', 'Bucket Policy' (which is selected and highlighted in blue), and 'CORS configuration'. Below the tabs, the title 'Bucket policy editor' is followed by the bucket ARN. A text prompt says 'Type to add a new policy or edit an existing policy in the text area below.' The main area contains a JSON policy document with line numbers 1 through 11 on the left. The policy is identical to the one shown in the previous block, with the resource 'arn:aws:s3:::serverless101.runcy.me/\*' highlighted in yellow in the original image.

```
1 "Version": "2012-10-17",      "Statement":
2   [
3     {
4       "Sid": "PublicReadGetObject",
5       "Effect": "Allow",
6       "Principal": "*",
7       "Action": "s3:GetObject",
8       "Resource": "arn:aws:s3:::serverless101.runcy.me/*"
9     }
10  ]
11
```

# Let's setup DNS

- Select “Route 53” from Networking category
- Select your *Hosted Zone* for the website\*
- Click “Create Record Set”



\* Assuming you have a website that is managed with Route 53. Settings 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 sometime for records to propagate (usually 3-4 mins)

**Create Record Set**

**Name:**

**Type:**

**Alias:** ☒ Yes ☐ No

**Alias Target:**

**Alias Hosted:**   
You can also type  
- CloudFront distribution:   
- Elastic Beanstalk environment CNAME: example.elasticbeanstalk.com  
- ELB load balancer DNS name: example-1.us-east-1.elb.amazonaws.com  
- S3 website endpoint: s3-website.us-east-2.amazonaws.com  
- Resource record set in this hosted zone: www.example.com  
[Learn More](#)

**Routing Policy:**

Route 53 responds to queries based only on the values in this record.  
[Learn More](#)

**Evaluate Target Health:** ☐ Yes ☒ No

**Create**

Your web app is \*now\* LIVE!

