Building A Serverless Web App

Runcy Oommen
https://runcy.me

Session Objectives

1. Bring up DB and 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

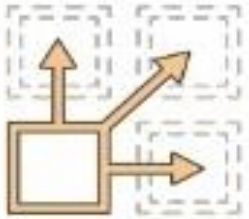
Let's Start Serverless

What is serverless?

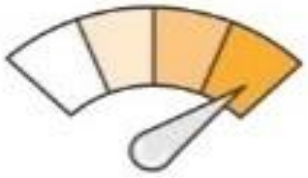
Build and run applications
without thinking about servers

“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)

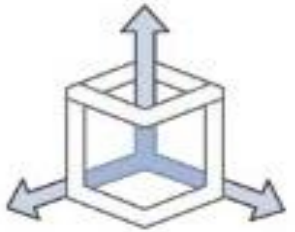
Removes the need for...



Provisioning and Utilization



Operations and Management



Scaling

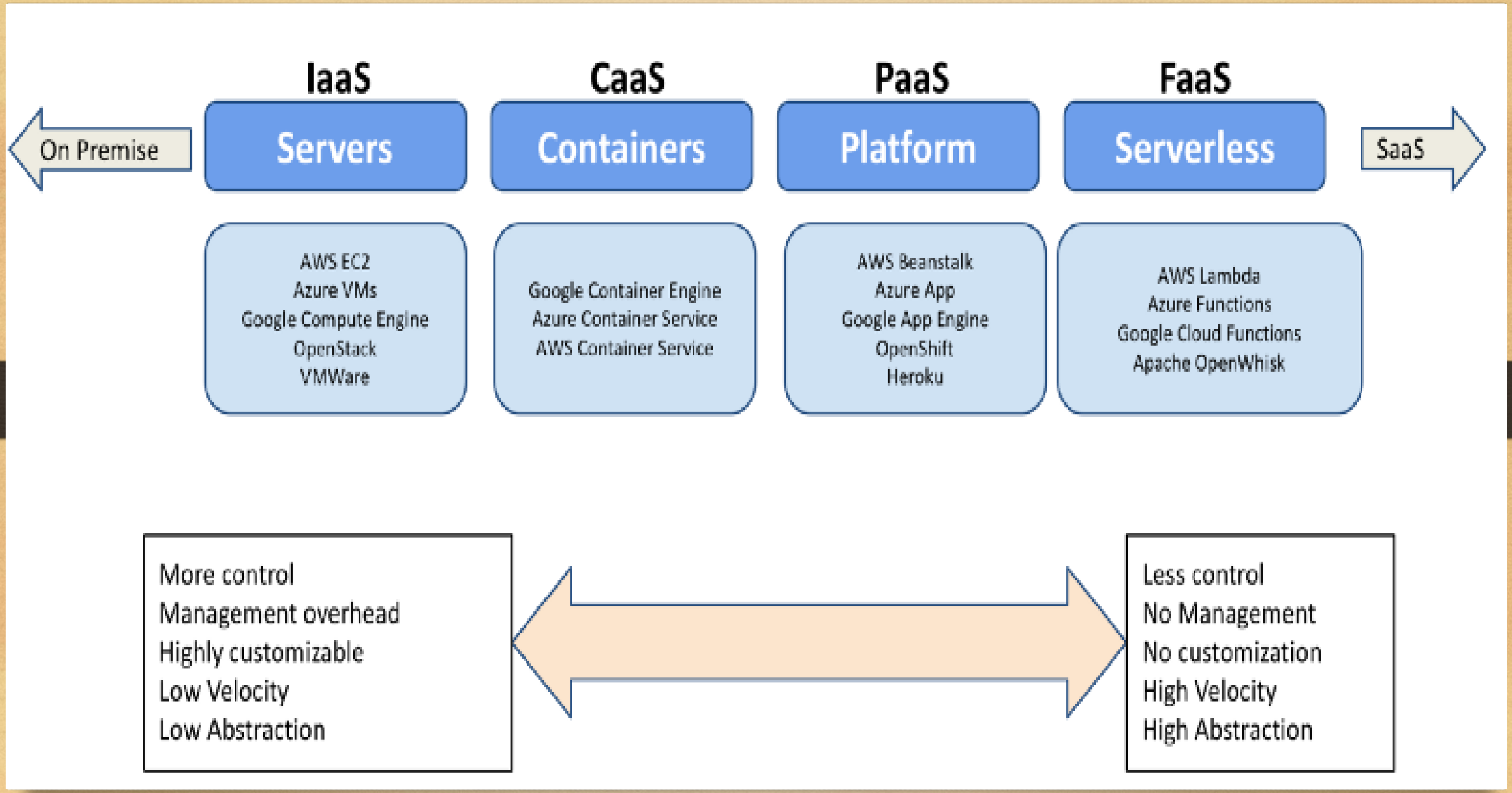
Availability and Fault Tolerance

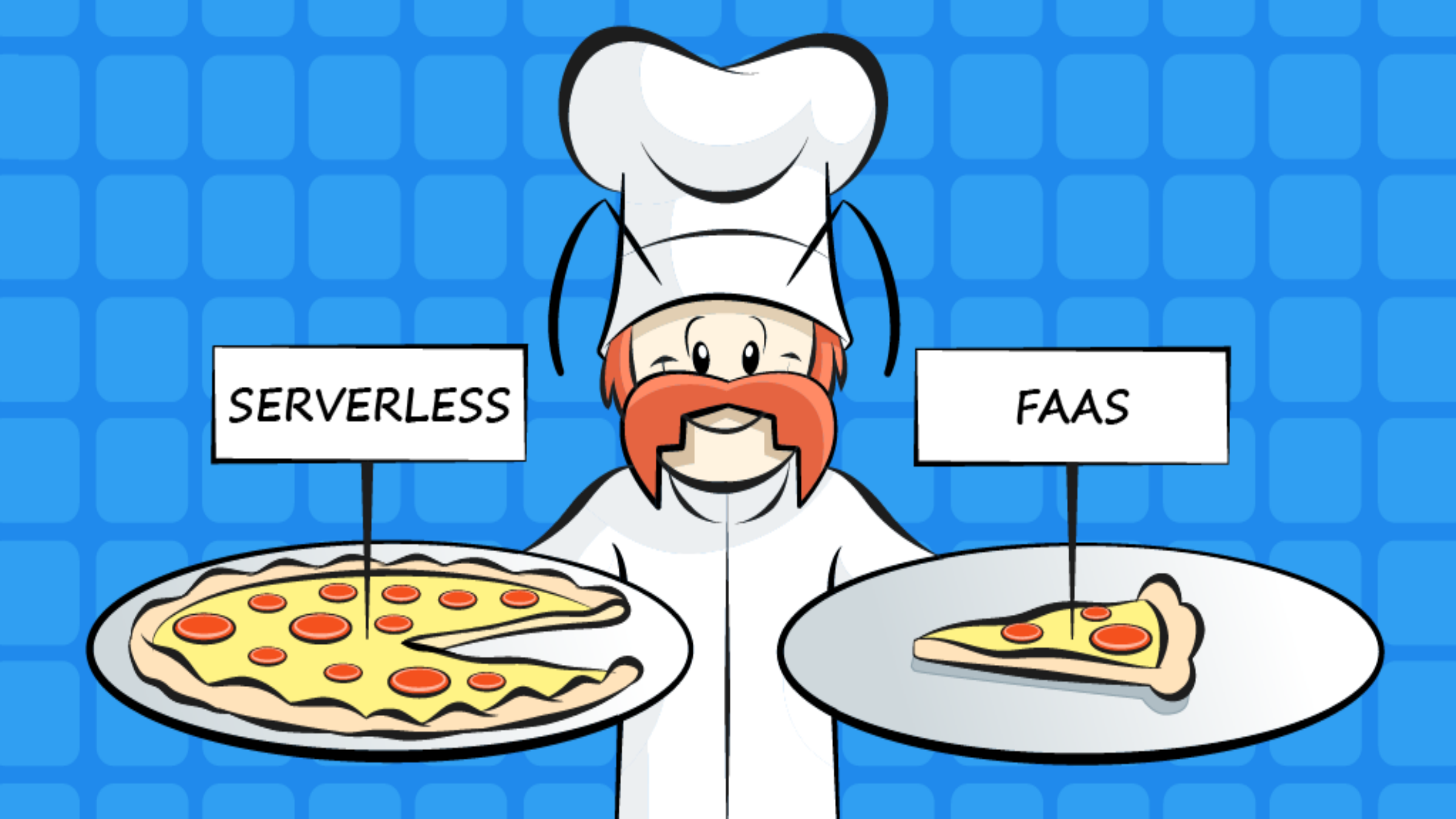
Provides these...

Abstraction of servers

Event-driven/ instant scale

Sub-second billing


IaaS
CaaS
PaaS
FaaS
On Premise
Servers
Containers
Platform
Serverless
SaaS
AWS EC2
Azure VMs
Google Compute Engine
OpenStack
VMWare
Google Container Engine
Azure Container Service
AWS Container Service
AWS Beanstalk
Azure App
Google App Engine
OpenShift
Heroku
AWS Lambda
Azure Functions
Google Cloud Functions
Apache OpenWhisk
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	Operational Model	Billing Model
• Event Driven • Shares Nothing • Stateless	• Zero Ops • Managed Security • Auto Scaling	• 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

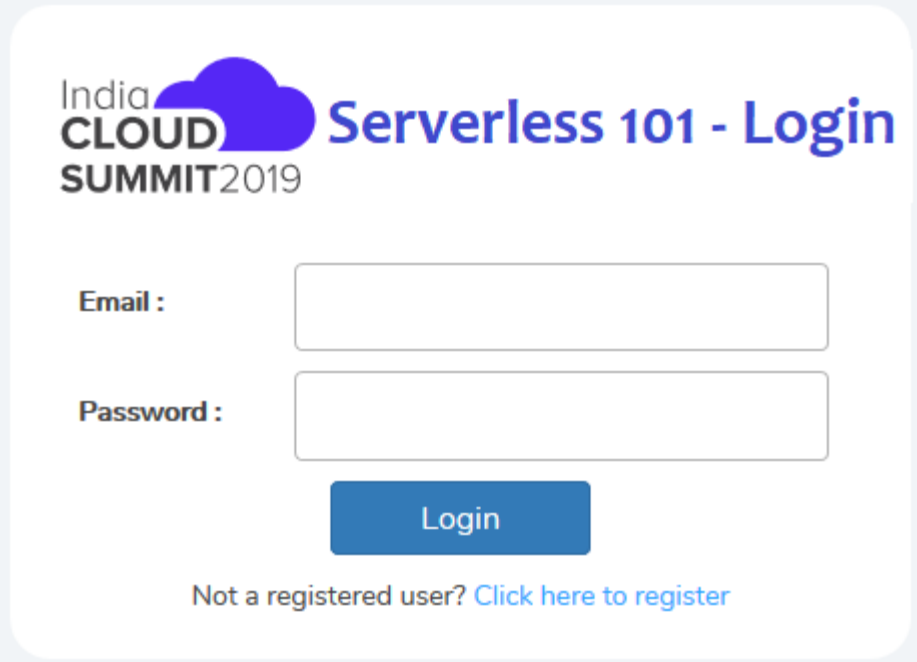
- **Lambda + Serverless**

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

JUMP OUT...

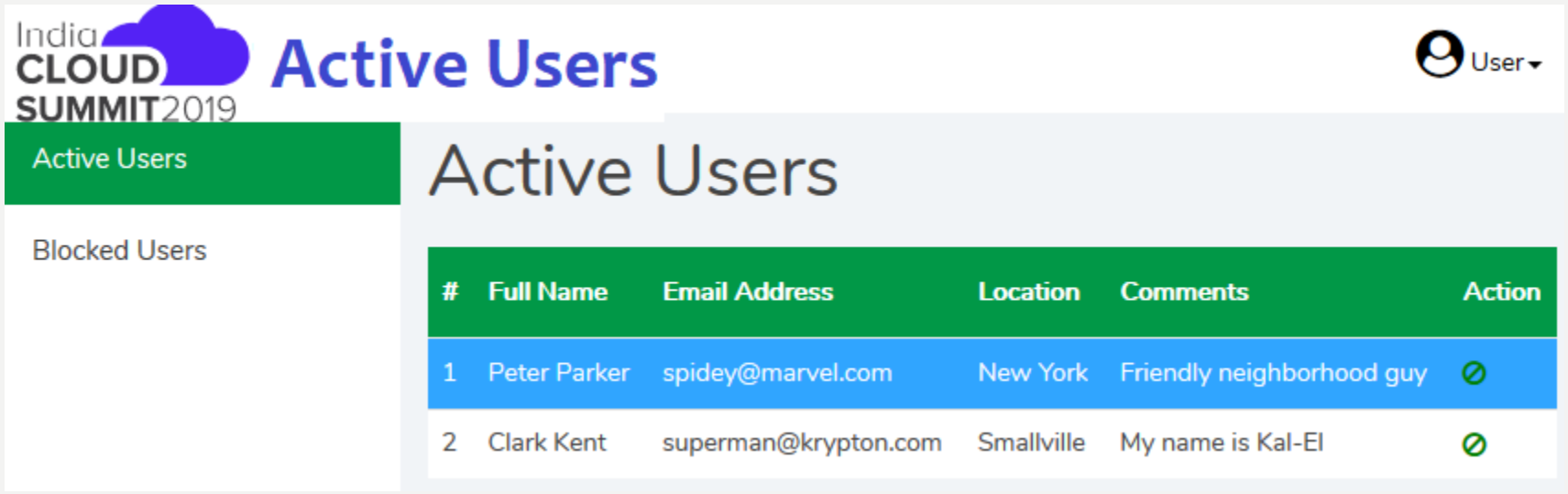
THINK SERVERLESS!

What are we building today?

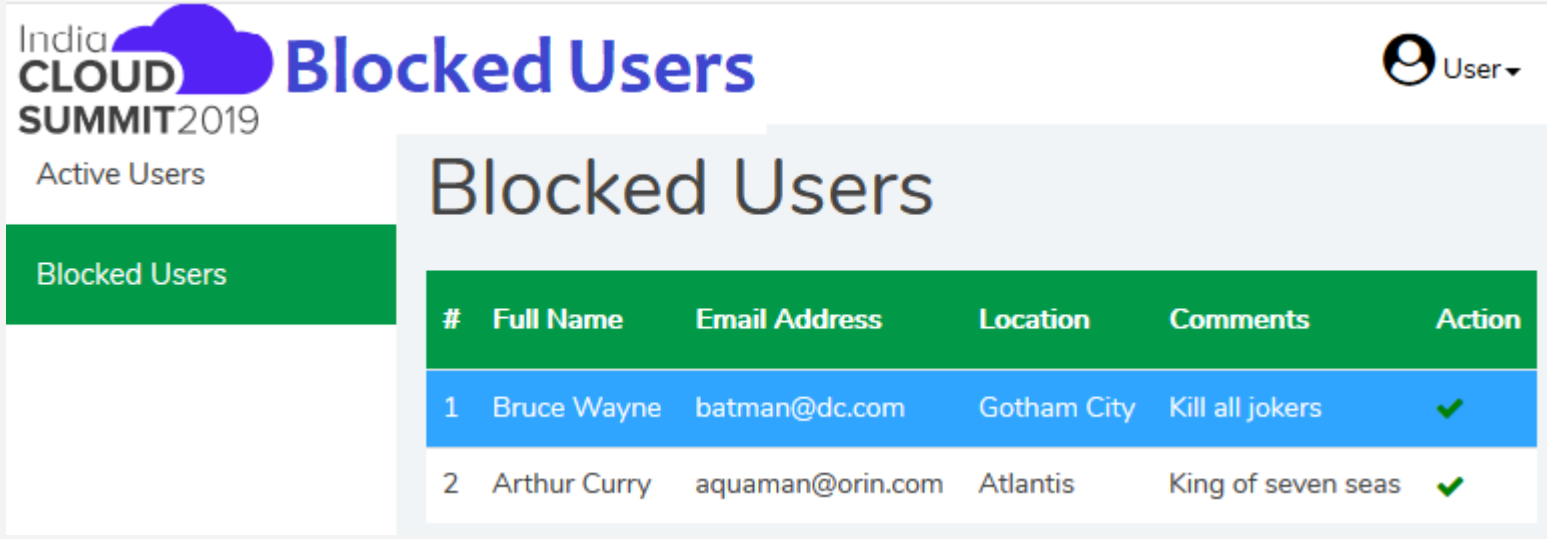


1 Login registration

2 User login page



3 **Show active users**



4 **Show blocked users**


SHUT UP
SHOW ME DEMO

Pre-requisites

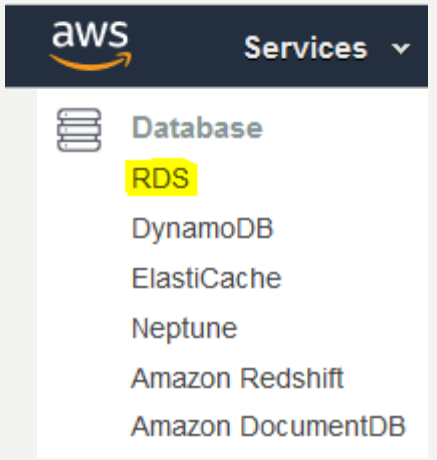
- **AWS Free Tier**
- **Source Code**

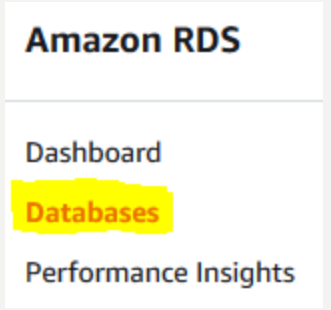
  https://github.com/roommen/serverless101

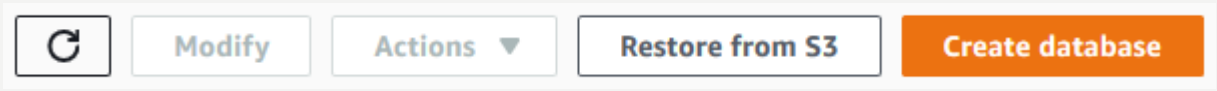
- **Basic knowledge of Python, HTML, JS, CSS**
- **A good IDE like Visual Studio Code**

Let's fire up the DB

- Login to AWS Console
- Select "RDS" from Database under Services
- Click "Databases" from the left-menu
- Click on "Create database"
- Select "MySQL" as the engine and click "Next"

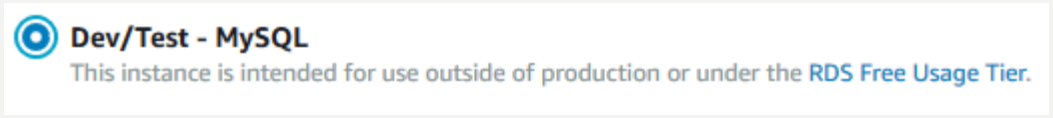




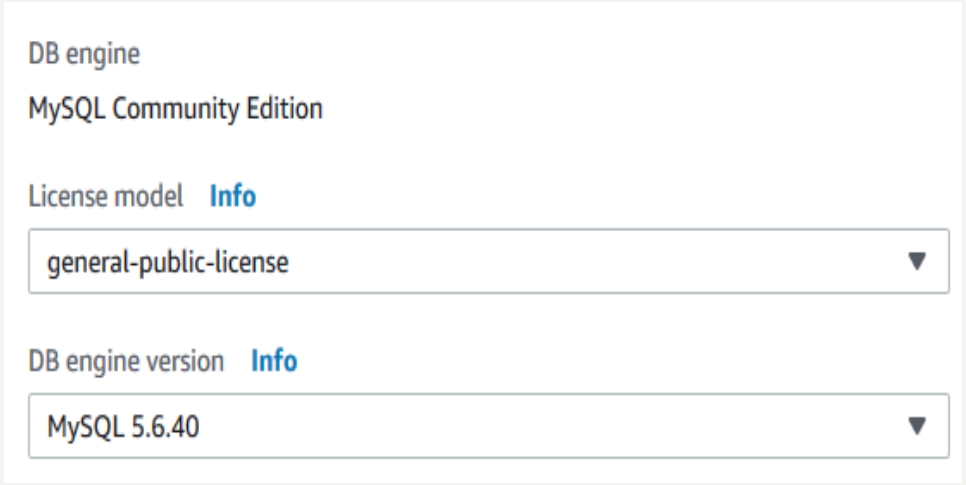


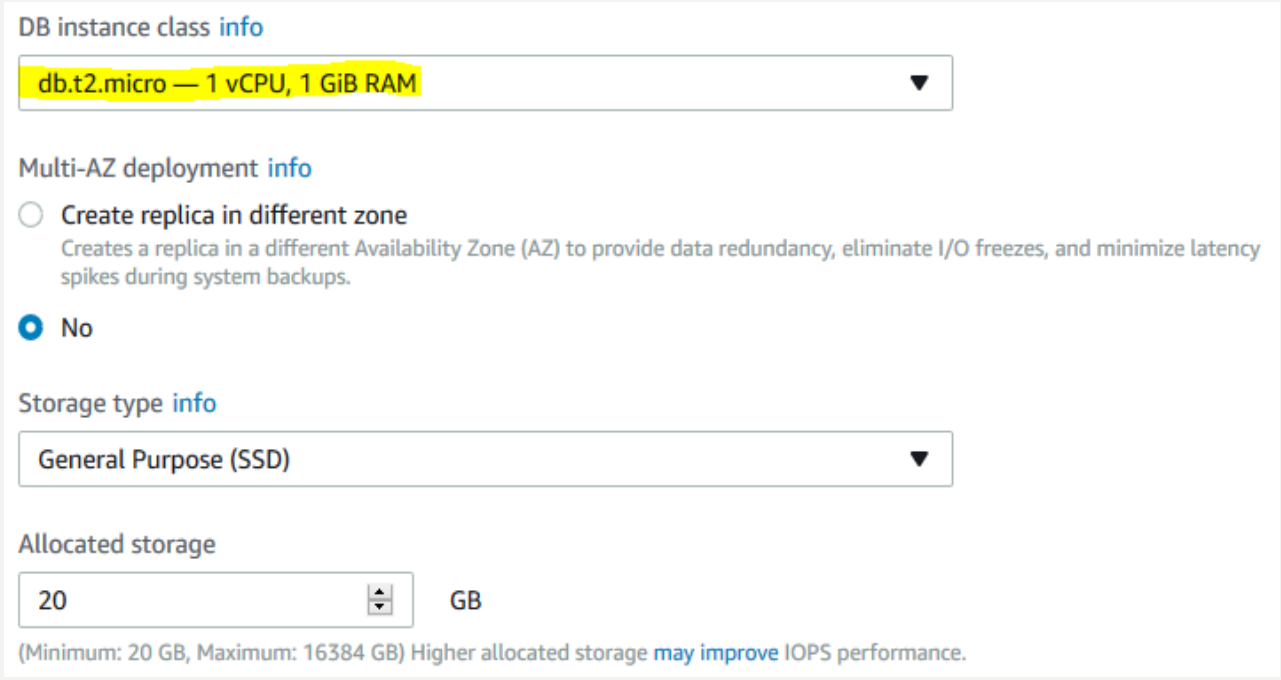
MySQL setup

- In next screen, choose "Dev/Test – MySQL"

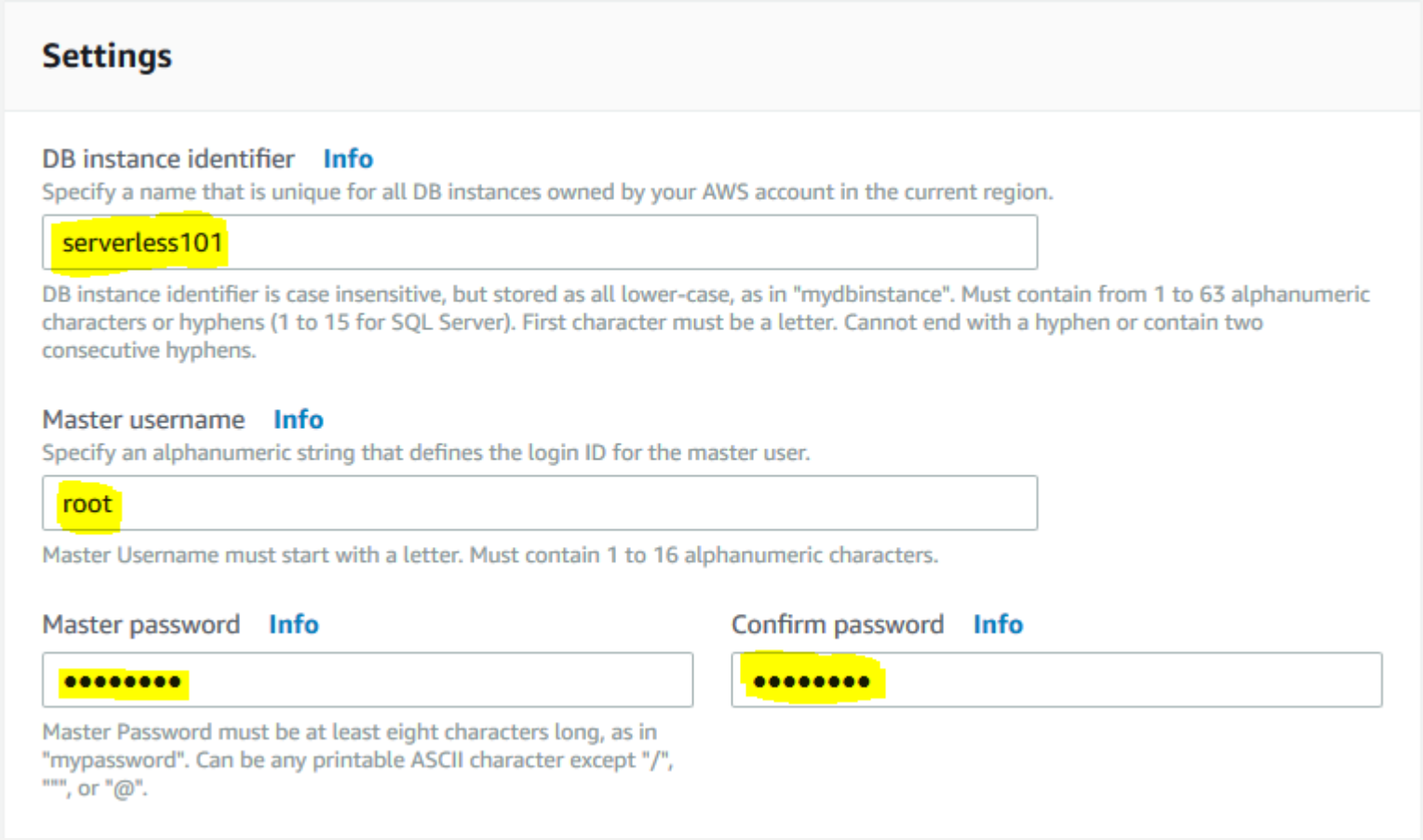


- Instance Specifications



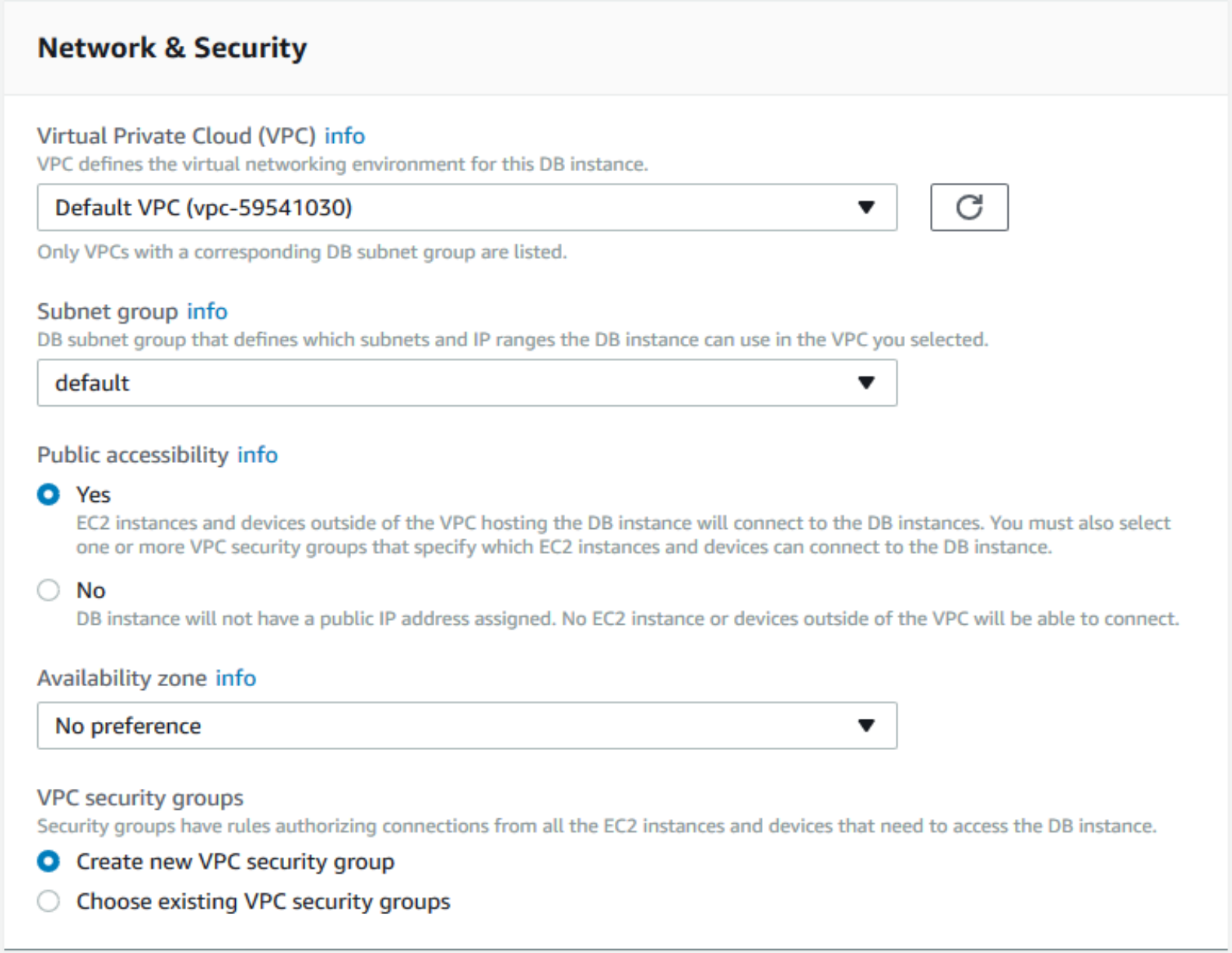


MySQL DB settings



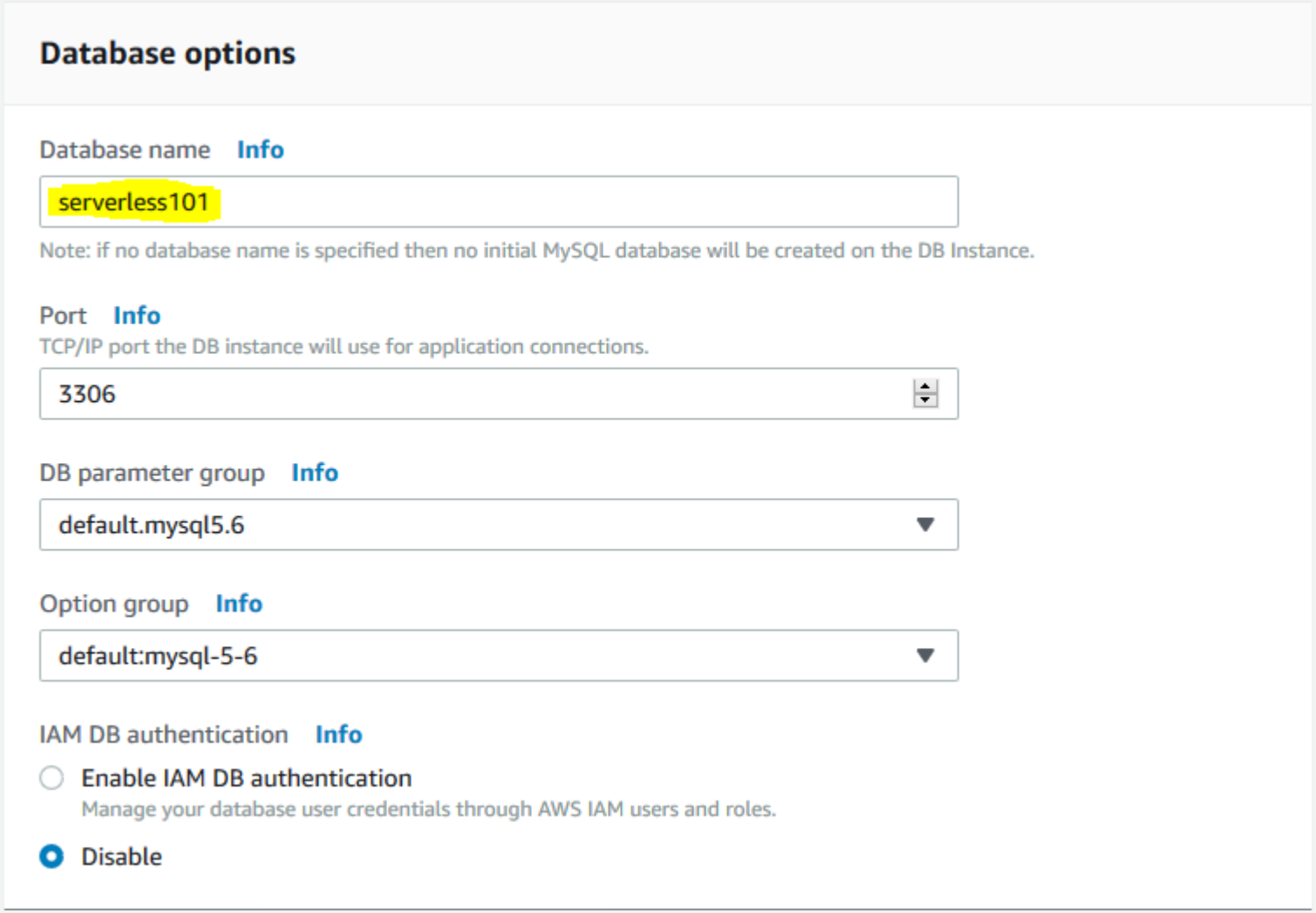
Provide DB instance name and credentials

MySQL advanced settings - Network & Security



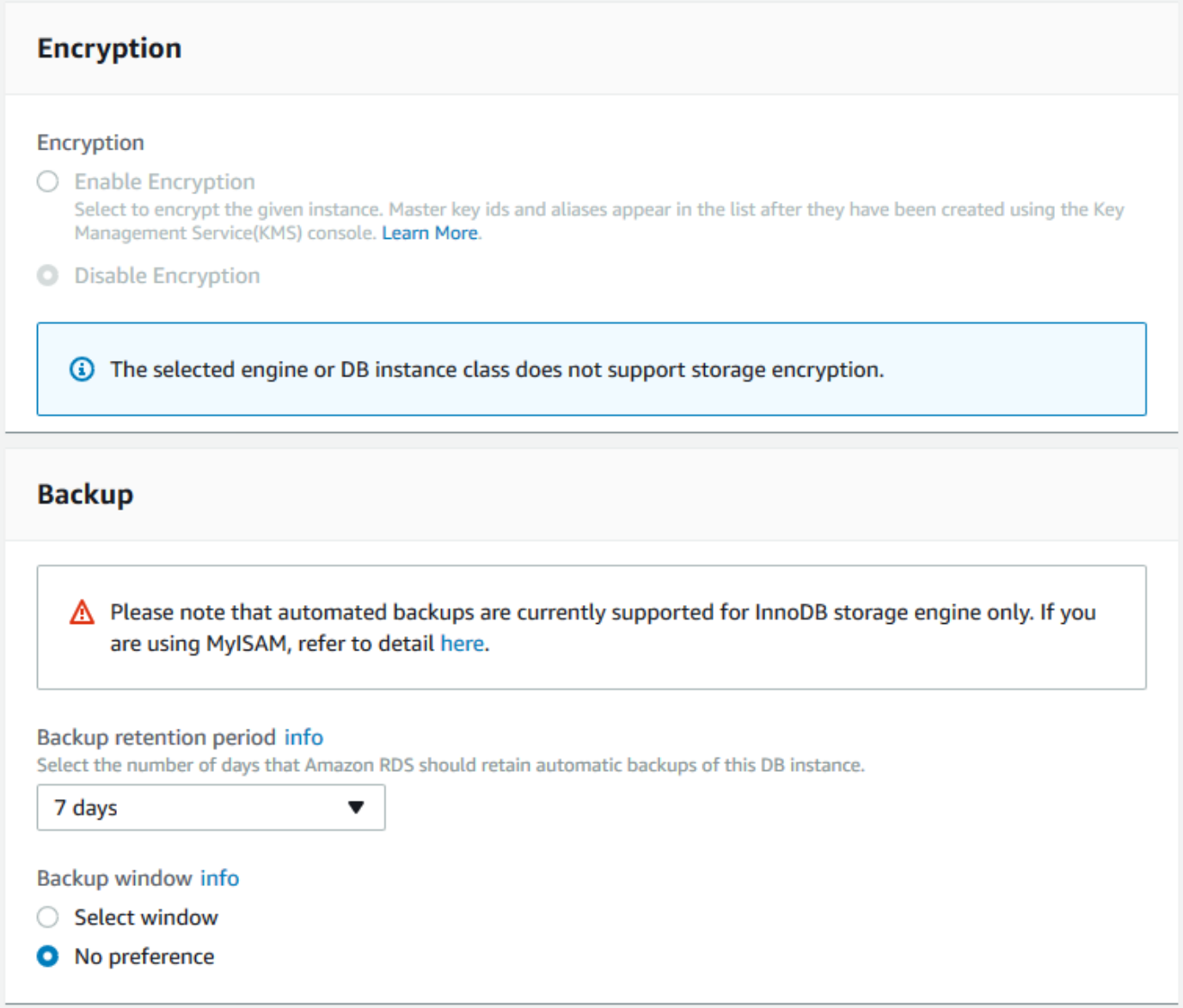
Keep everything as the default setting

MySQL advanced settings – Database options



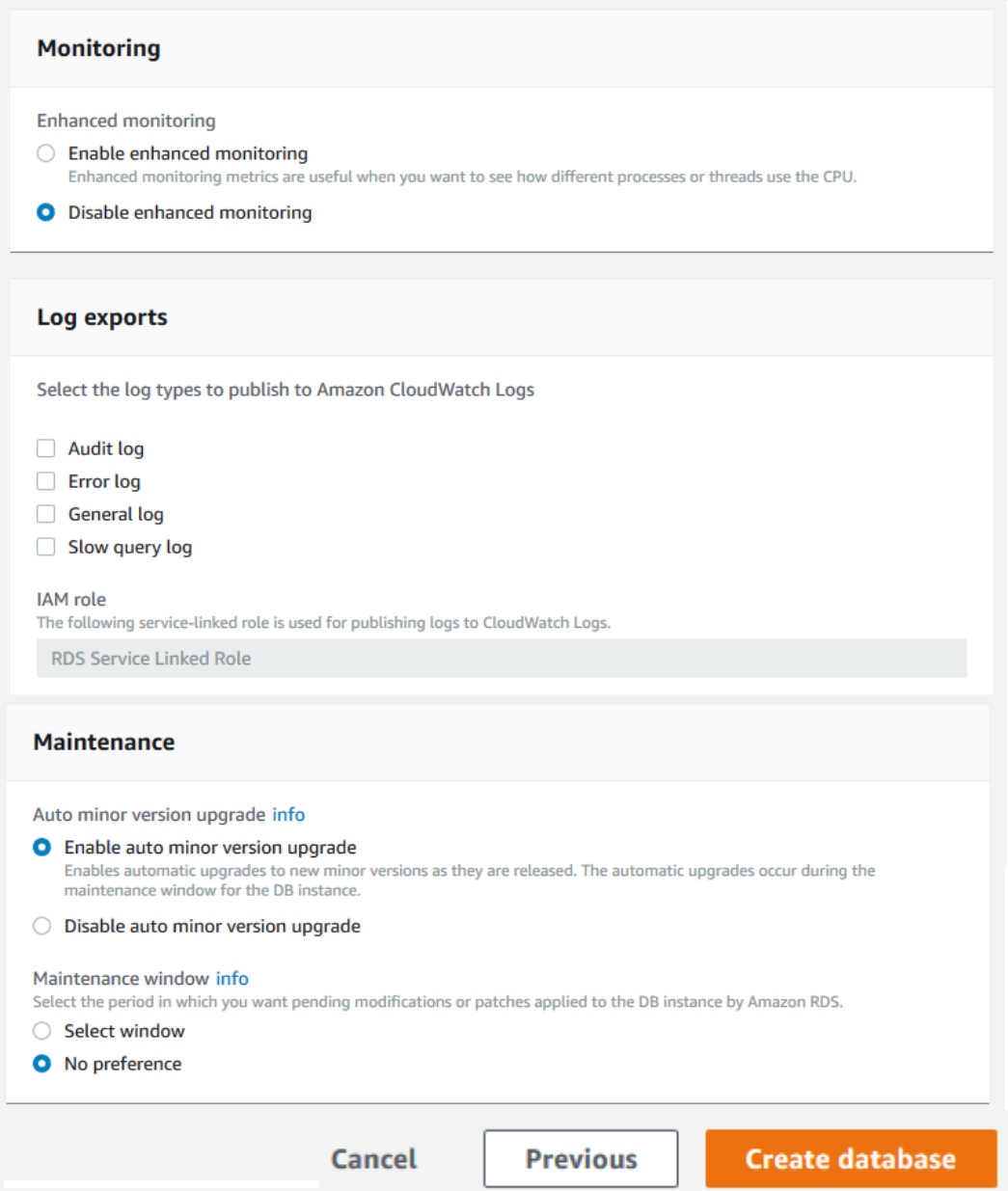
Provide appropriate DB name

MySQL advanced settings – Encryption & Backup



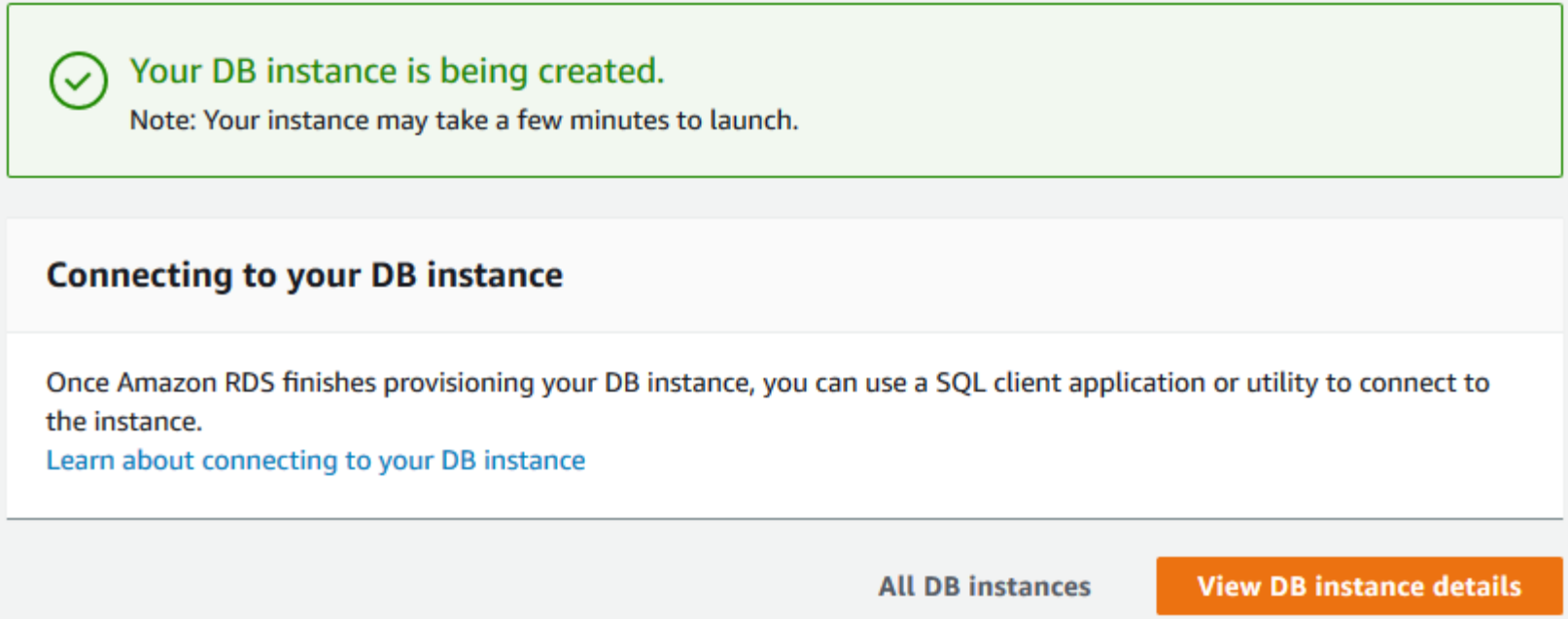
Leave everything as default

MySQL advanced settings – Monitoring, Log, Maintenance



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

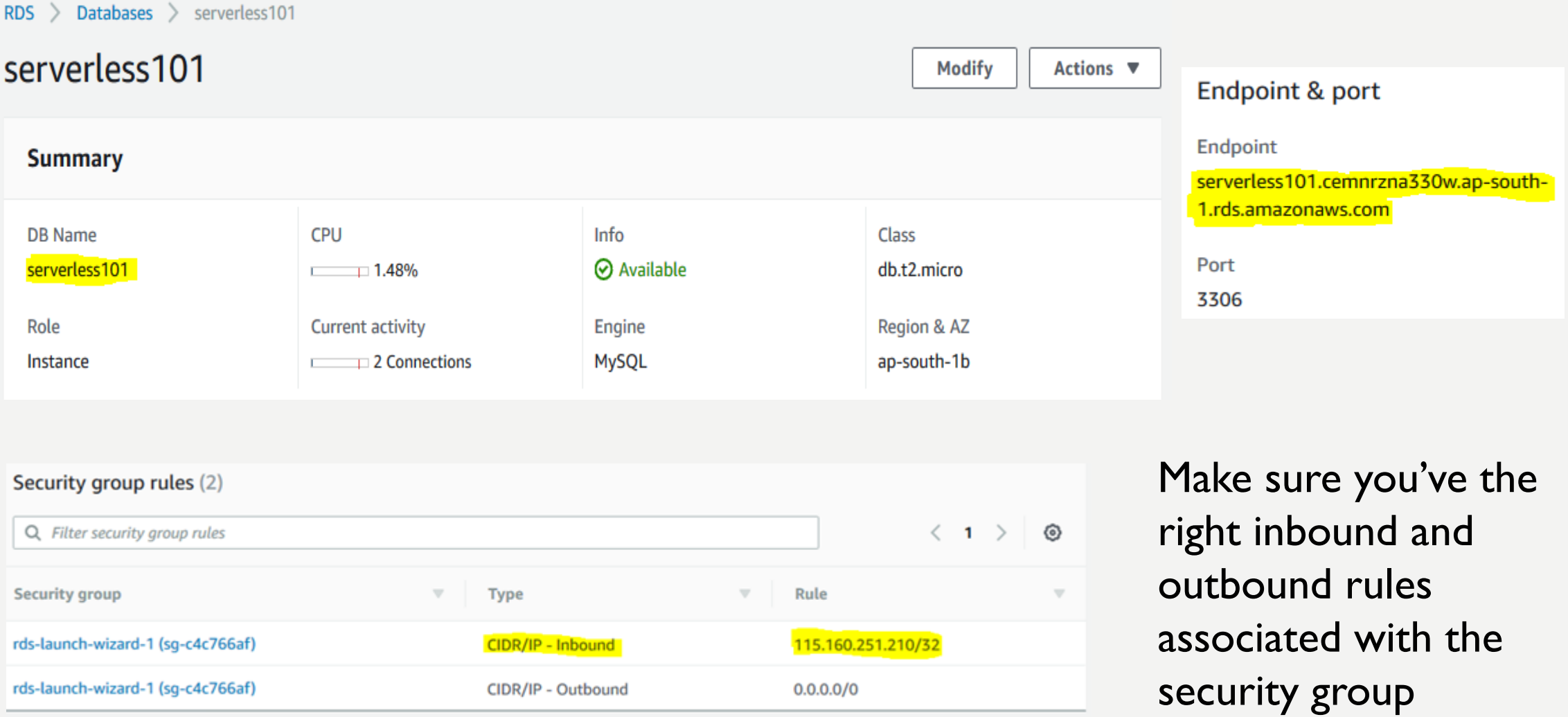
MySQL getting initialized



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

MySQL Endpoint

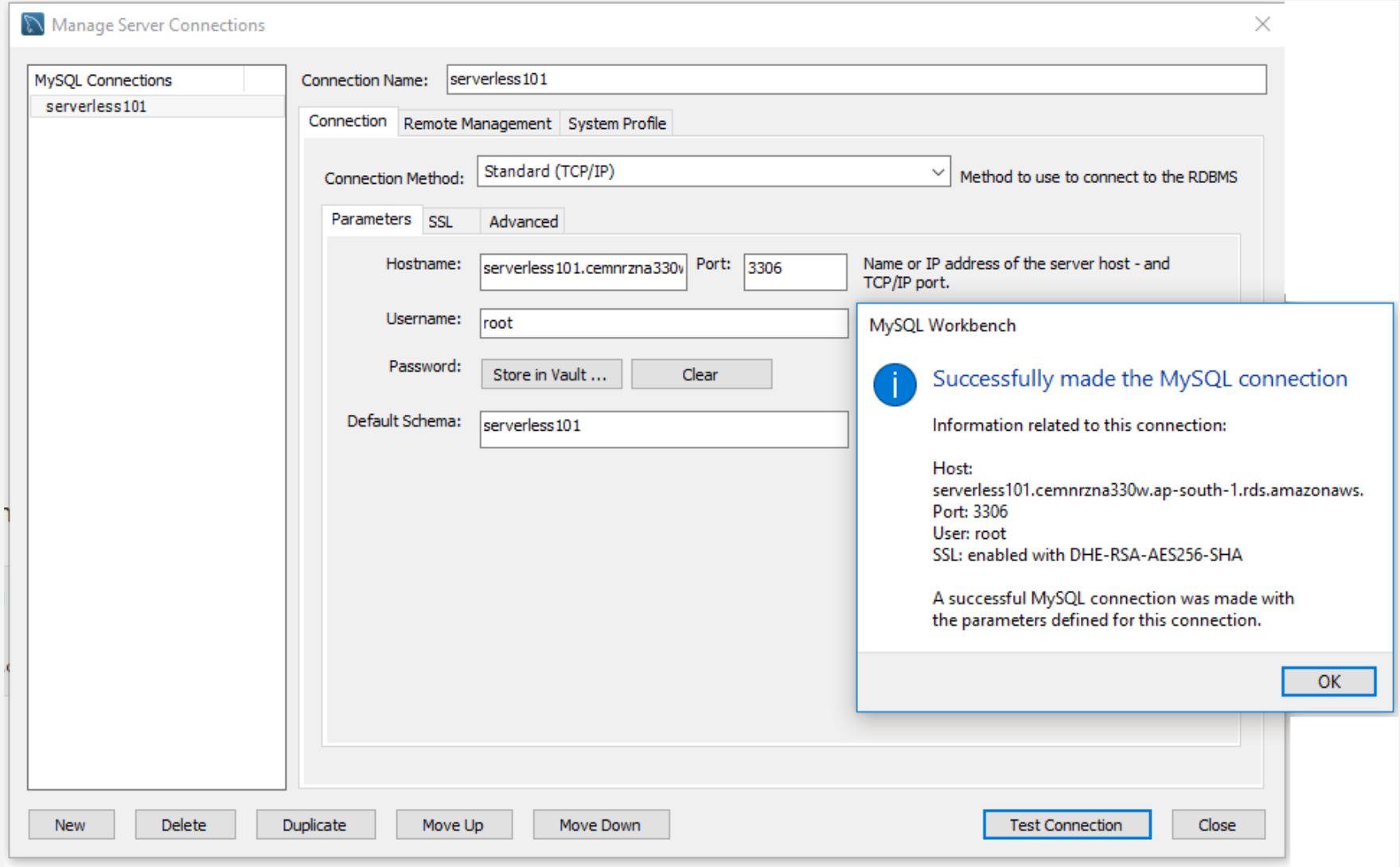
Once the DB creation is successful, you should have something like this:



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..



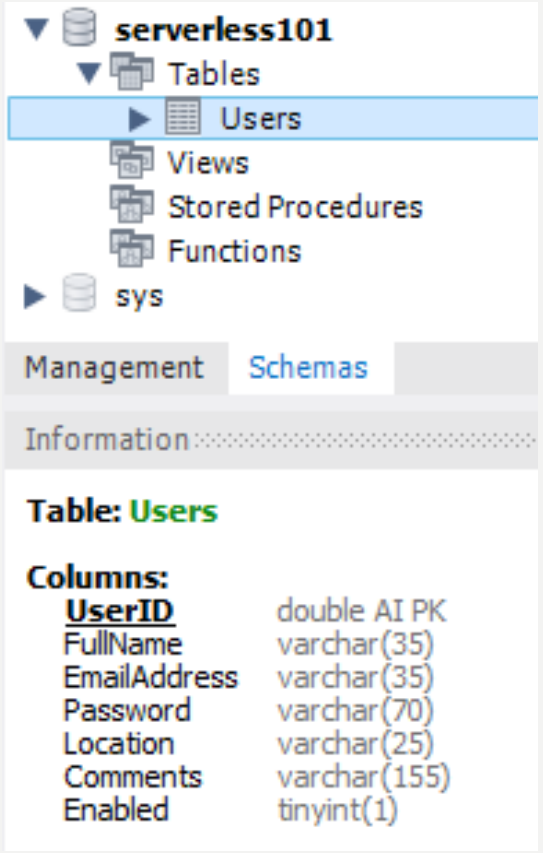
Creating Users table

- Go to the cloned *serverless101* repository location
- Navigate to the "dbscripts" folder
- Edit the 'CreateTableUsers.py' file with the DB info you created earlier

```python
import mysql.connector

def create_users():
    connection, cursor = None, None
    try:
        # Database connection parameters - replace this with your DB endpoint
        serverless101cnxstr = {'host': 'serverless101.cemnrzna330w.ap-south-1.rds.amazonaws.com', 'user': 'root', \
        'password': 'password', 'database': 'serverless101'}
        connection = mysql.connector.connect(host=serverless101cnxstr['host'], user=serverless101cnxstr['user'], \
        password=serverless101cnxstr['password'], database=serverless101cnxstr['database'])
        cursor = connection.cursor()
        cursor.execute('CREATE TABLE Users (UserID DOUBLE NOT NULL AUTO_INCREMENT PRIMARY KEY, \
        FullName VARCHAR(35) NOT NULL, EmailAddress VARCHAR(35) NOT NULL, Password VARCHAR(70) NOT NULL, \
        Location VARCHAR(25) NOT NULL, Comments VARCHAR(155) NOT NULL, Enabled BOOLEAN 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 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

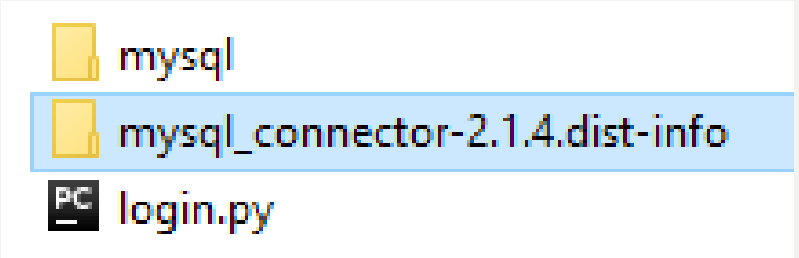
(Use specific version 2.1.4 – I was getting an error for the latest one)

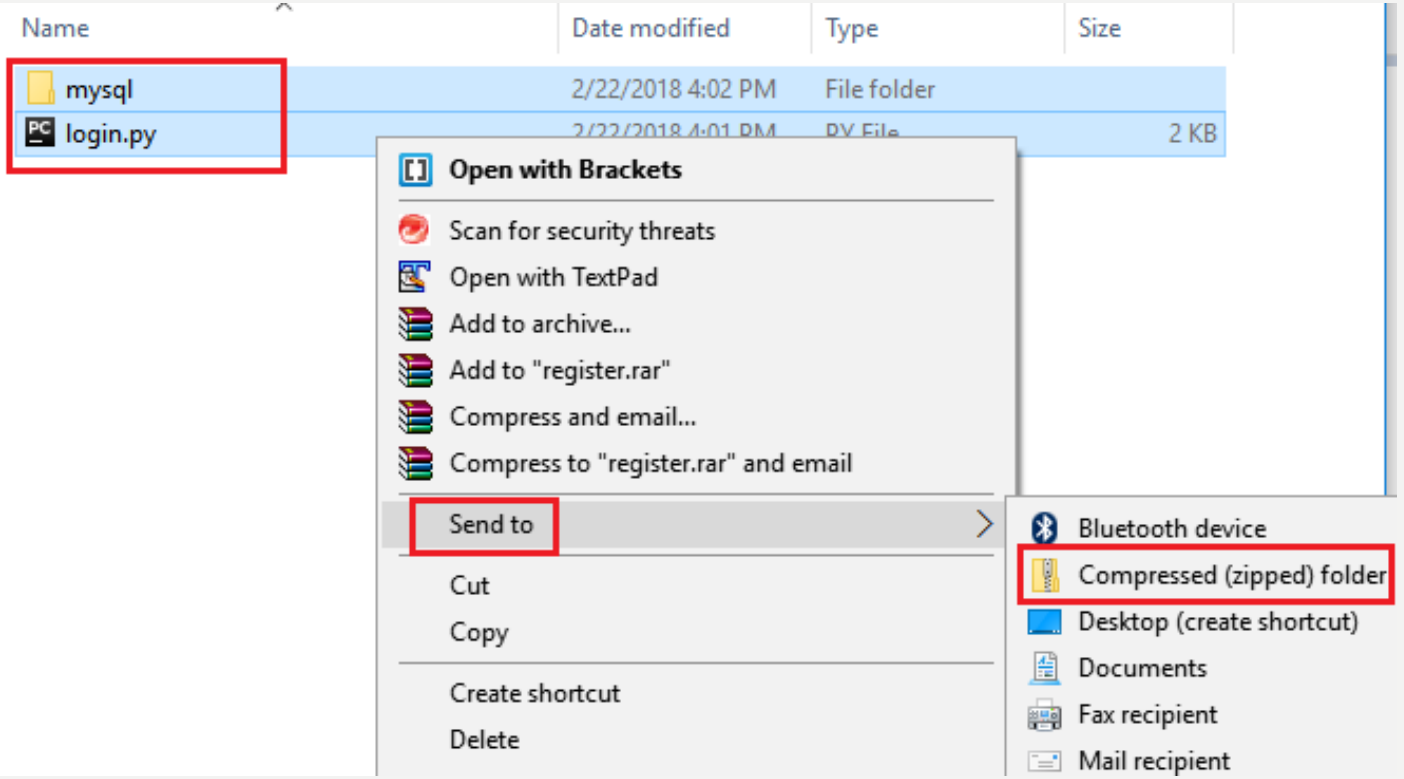
```
runcy@runcyoommen-PC:/mnt/f/serverless101/serverless$ mkdir login
runcy@runcyoommen-PC:/mnt/f/serverless101/serverless$ cd login/
runcy@runcyoommen-PC:/mnt/f/serverless101/serverless/login$ cp ../login.py .
runcy@runcyoommen-PC:/mnt/f/serverless101/serverless/login$ pip3 install mysql-connector==2.1.4 --targ
et .
Collecting mysql-connector==2.1.4
  Downloading https://files.pythonhosted.org/packages/53/61/26b0bc2655ad64d550565252baf83611fe9db7d98a
3c571950b2b829ffa7/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/c7d096eaa31cc71ac259b9183a91ddf62bfd293c
f52cfdf3dc
Successfully built mysql-connector
```

AWS Lambda with Python – Zip 'em up

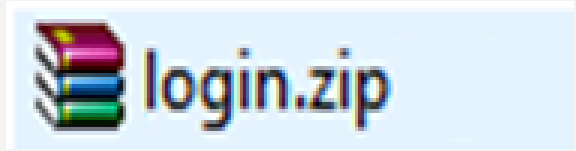
- Under the register folder, You might see a folder *mysql_connector-2.1.4.dist-info* which can be deleted if you want to

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



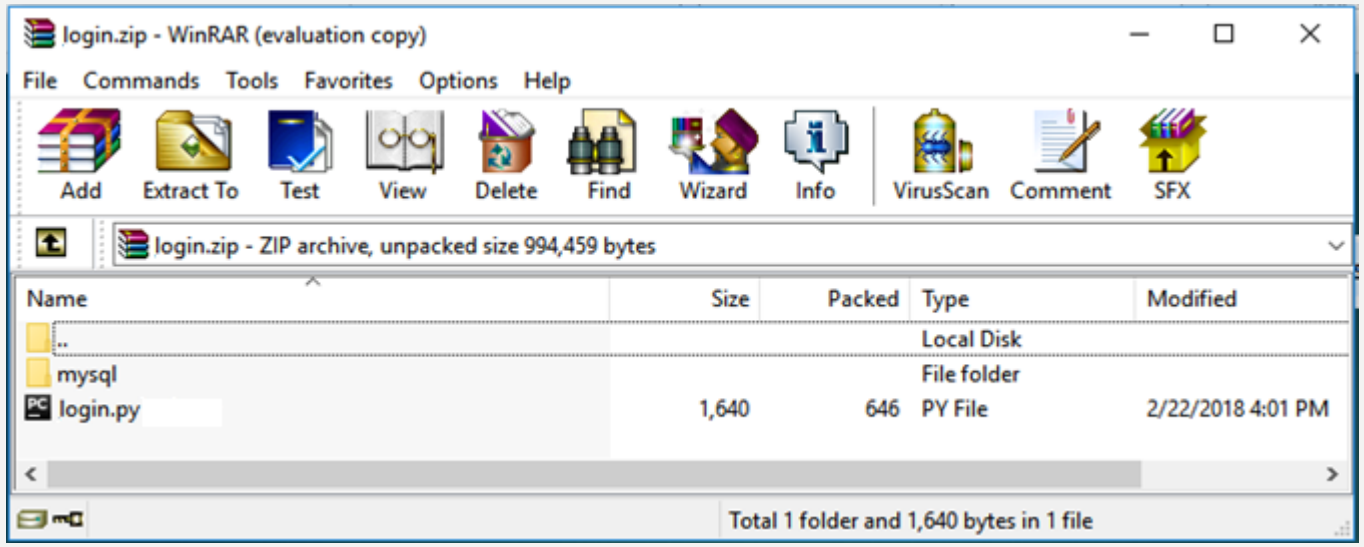


AWS Lambda with Python – Zip file details



- 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!

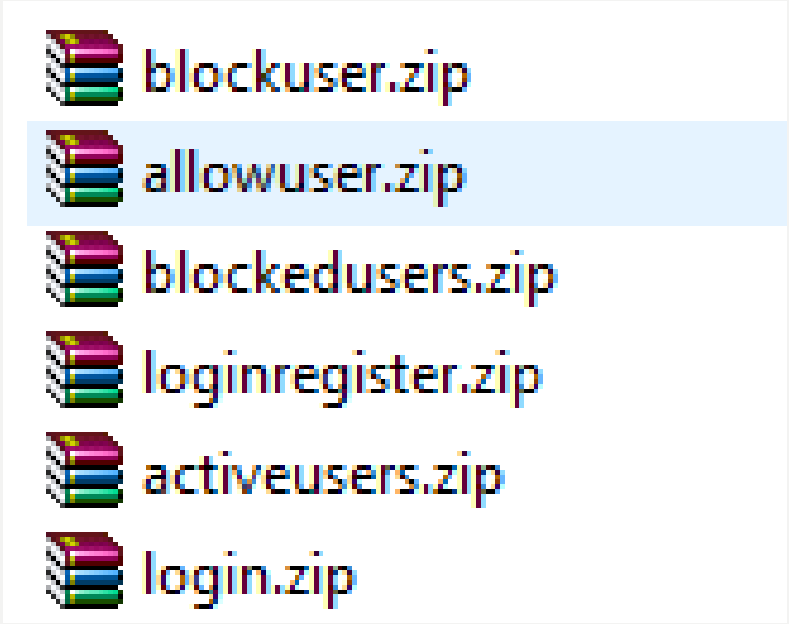
- Go to the cloned *serverless101* repository location
- Navigate to the "serverless" folder
- Edit the 'create_serverless.py' file with the required filenames

```python
import os

files = ["login.py", "activeusers.py", "blockedusers.py", "loginregister.py",\
        "allowuser.py", "blockuser.py"]

try:
    # remove mysql unzipped folder if exist
    os.system("rm -rf mysql")
    # unzip mysql
    os.system("unzip mysql")

    for file in files:
        temp = file.split(".")[0]
        # remove root zipped folder if exist
        os.system("rm -rf " + temp)
        # remove existing folders
        os.system("rm -rf " + temp + "; rm -rf " + temp + ".zip")
        # create root folder
        os.makedirs(temp)
        # copy mysql folders to root
        os.system('cp -a mysql ' + temp + '; cp ' + file + ' ' + temp)
        # move to root folder and zip contents
        os.system('cd ' + temp + '; zip -r ' + temp + '.zip *; mv ' + temp + '.zip ../'
        # remove root folder
        os.system("rm -rf " + temp)
    # remove mysql unzipped folder if exist
    os.system("rm -rf mysql")
except Exception as e:
    print(e)

```

Run the create_serverless.py file

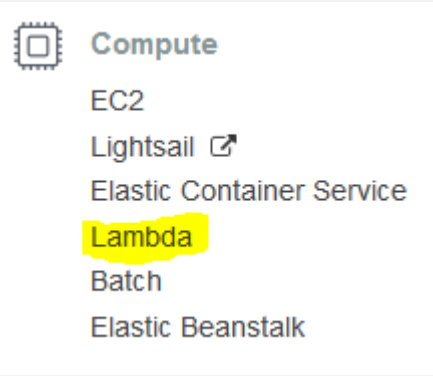
```
runcy@runcyoommen-PC:/mnt/f/serverless101/serverless$ python3 create_serverless.py
```

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



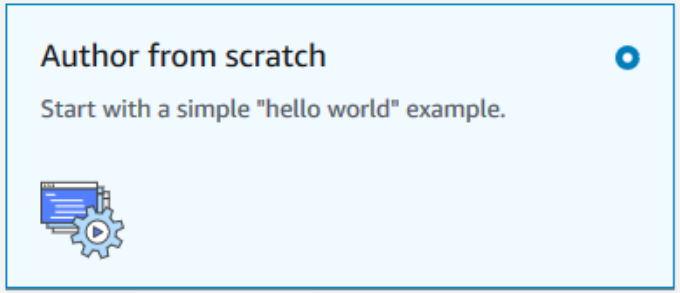
Let's create the Lambda functions

- Select "Lambda" from Compute category



- Click "Create function"

- Select "Author from scratch"



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

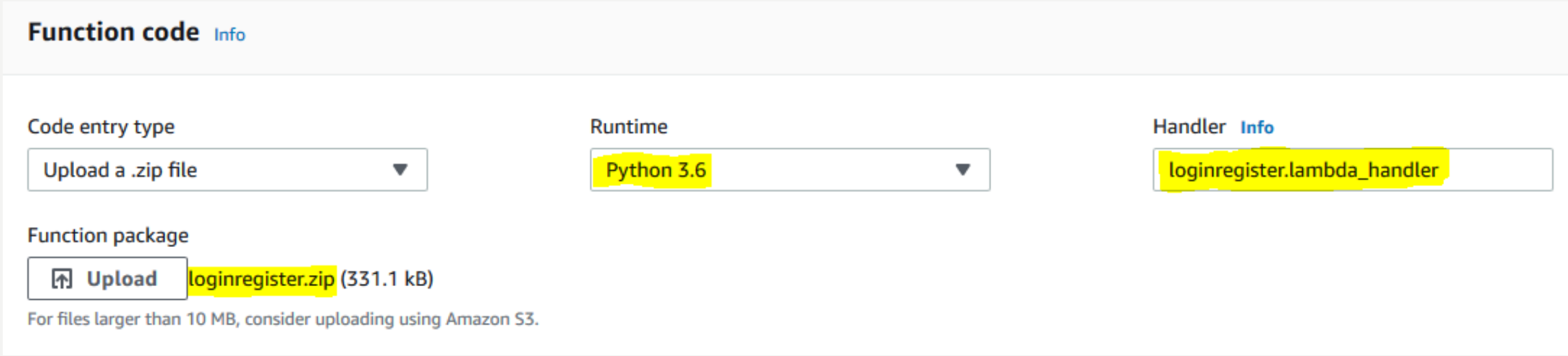
Existing role

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*



- 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

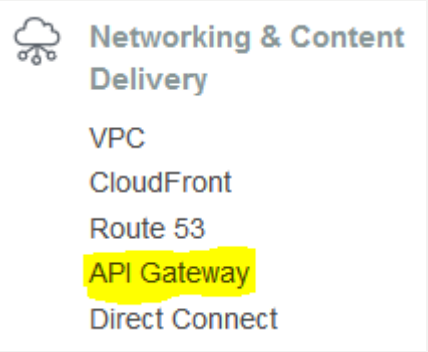
Lambda functions - Created

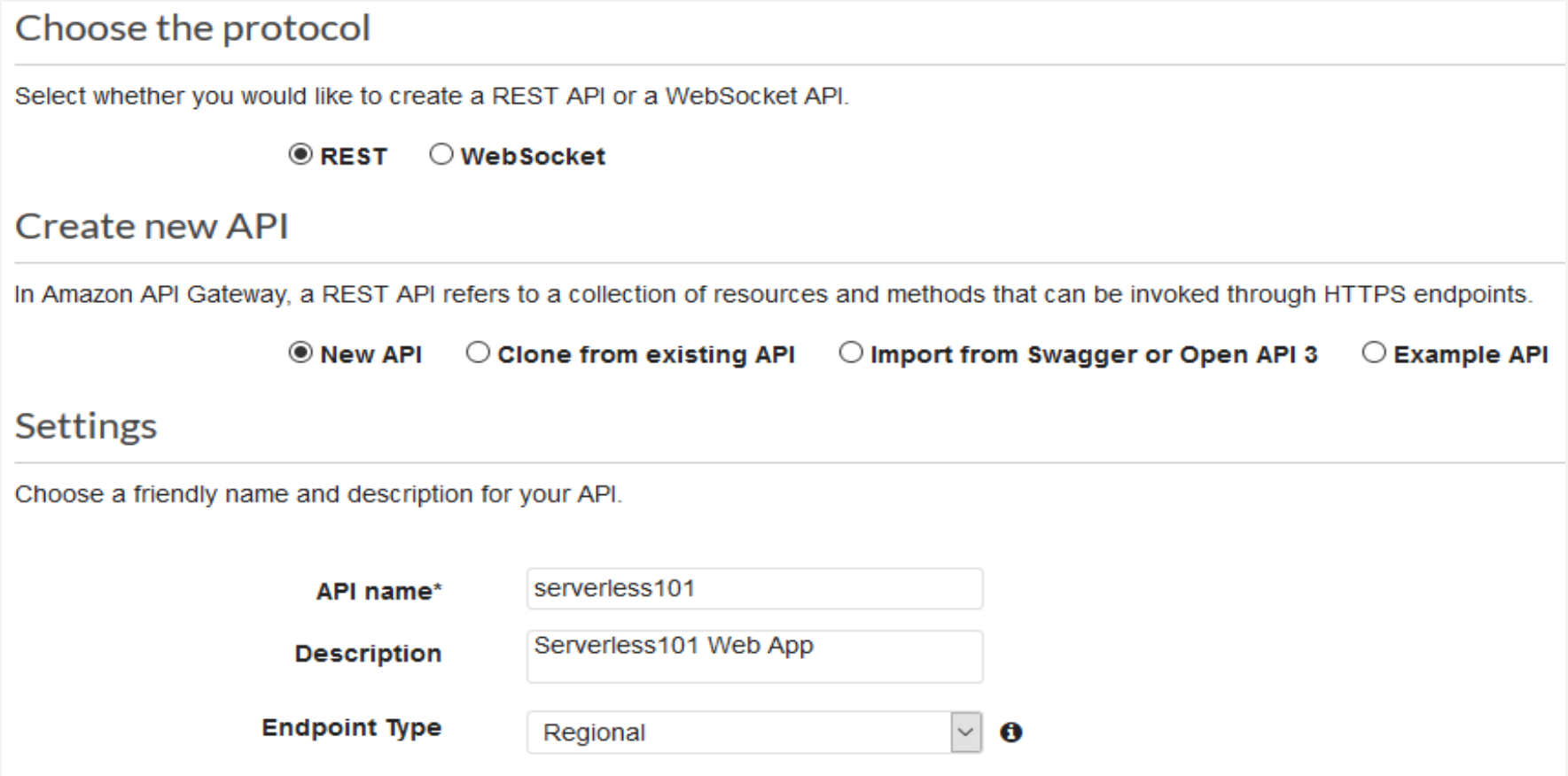
Function name	Description	Runtime	Code size	Last modified
serverless101-blockuser	Serverless 101 - Block User	Python 3.6	330.9 kB	2 days ago
serverless101-allowuser	Serverless 101 - Allow User	Python 3.6	330.9 kB	2 days ago
serverless101-login	Serverless 101 - Login	Python 3.6	326.7 kB	2 days ago
serverless101-loginregister	Serverless 101 - Login Register	Python 3.6	326.8 kB	2 days ago
serverless101-blockedusers	Serverless 101 - Blocked Users	Python 3.6	326.7 kB	4 days ago
serverless101-activeusers	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

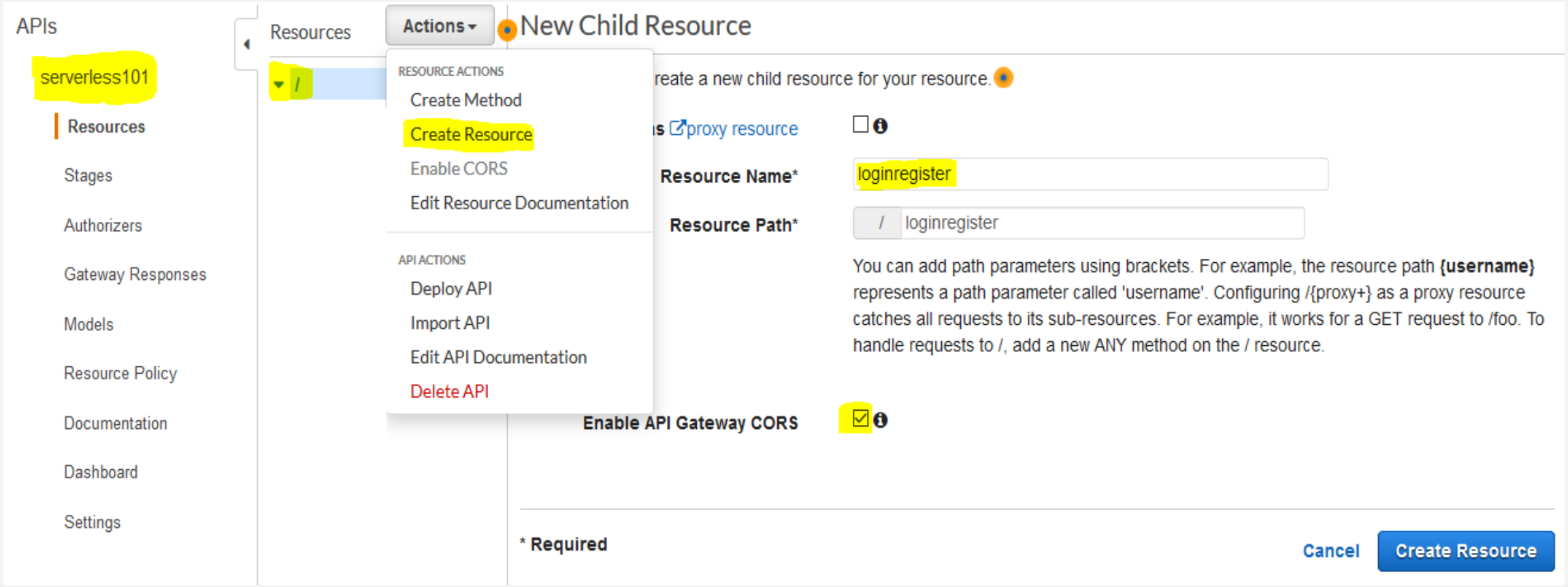
- Login to AWS Console
- Select "API Gateway" from **Networking & Content Delivery**
- Click "Create API"
- Choose "REST", "New API", API name and other details



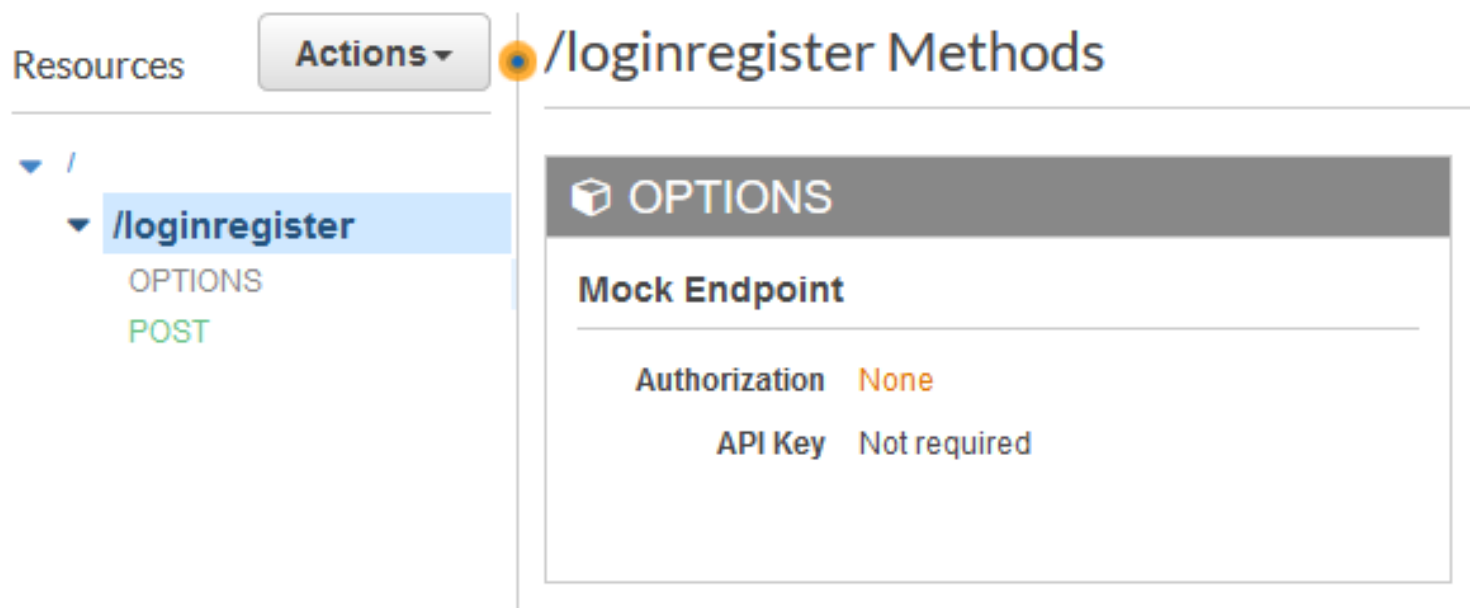


API Gateway – Create Resource (loginregister)

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

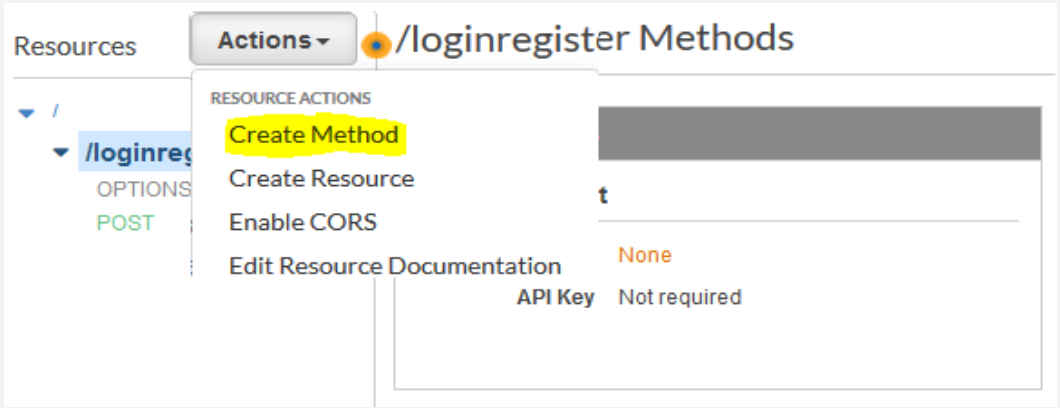


API Gateway – Resource created (loginregister)



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

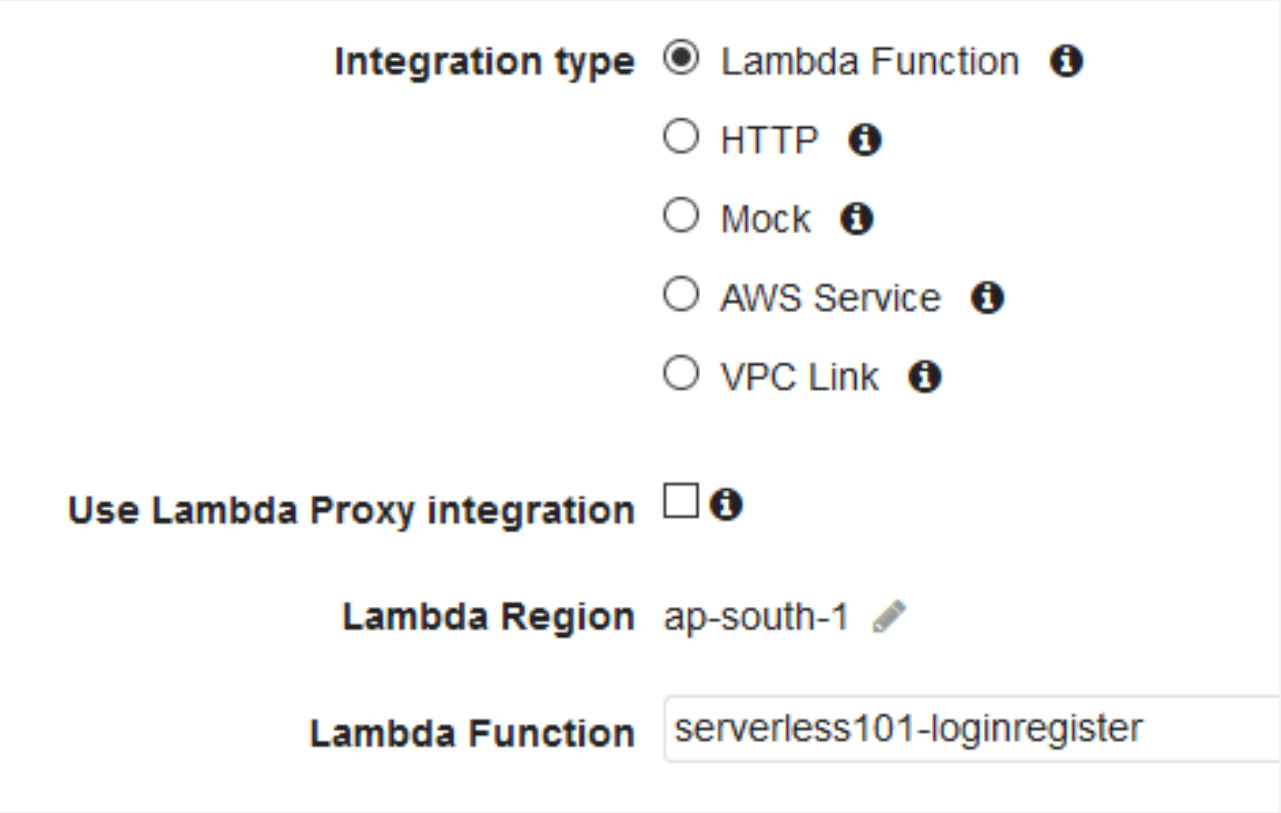
API Gateway – Create Method



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

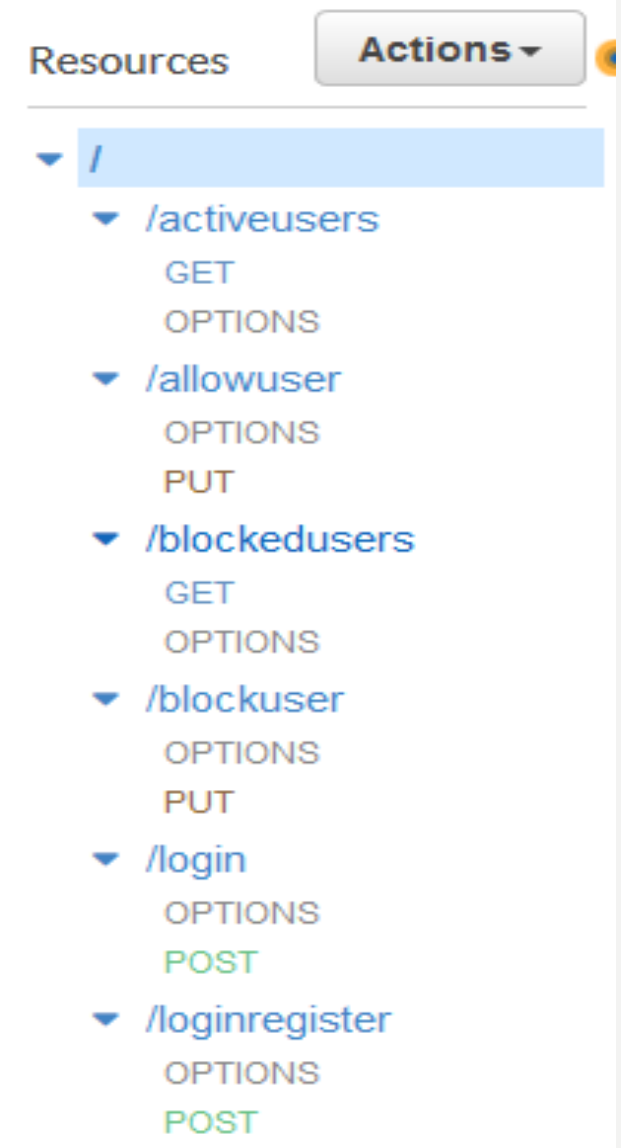
API Gateway – Configure POST (loginregister)

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



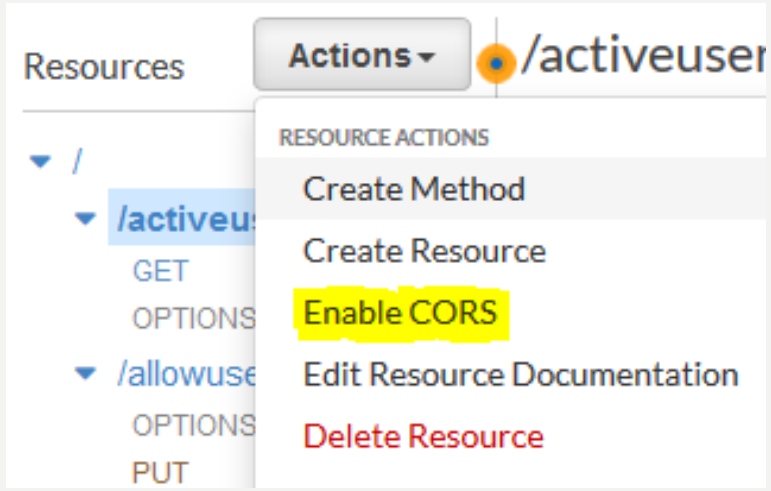
Select the appropriate region to choose the lambda function which we had created earlier

API Gateway – Create remaining resources & methods

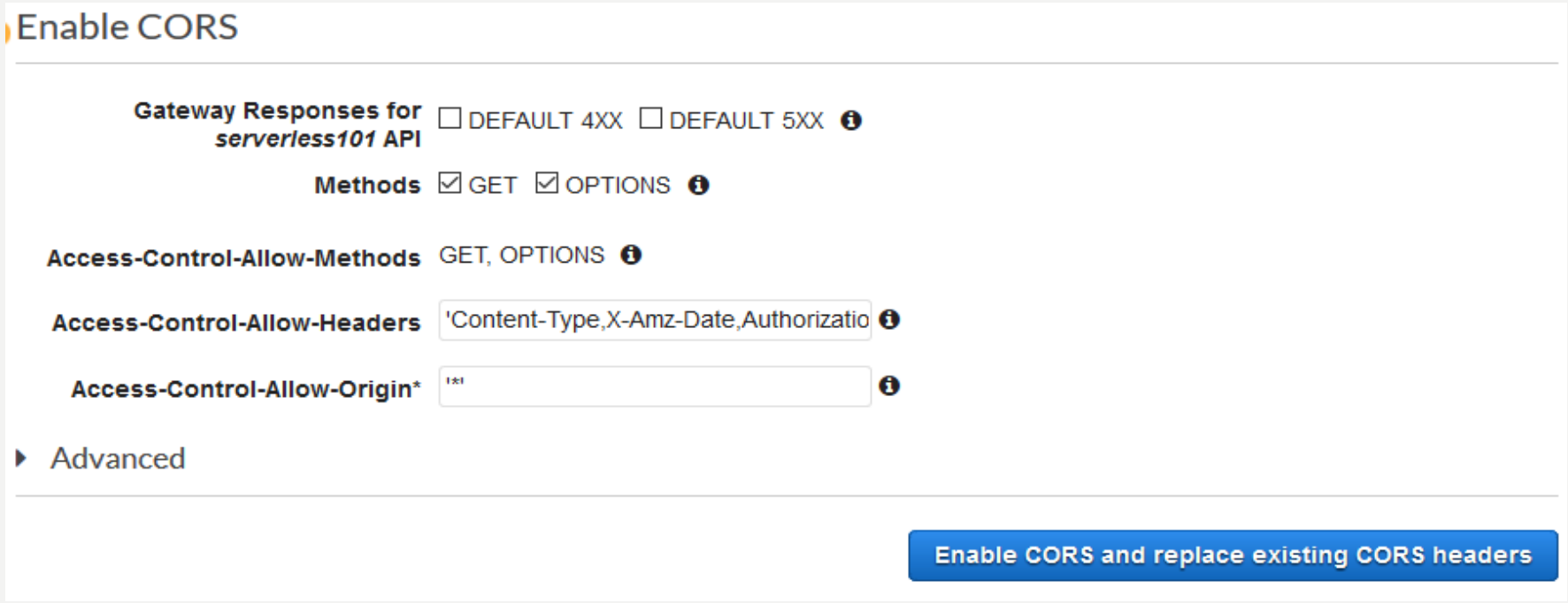


- Create *login* resource; associate **POST** method
- Create *activeusers* and *blockedusers* resources; associate **GET** method
- Create *allowuser* and *blockuser* resources; associate **PUT** method
- Follow identical steps as the previous *loginregister* for lambda configuration and integration

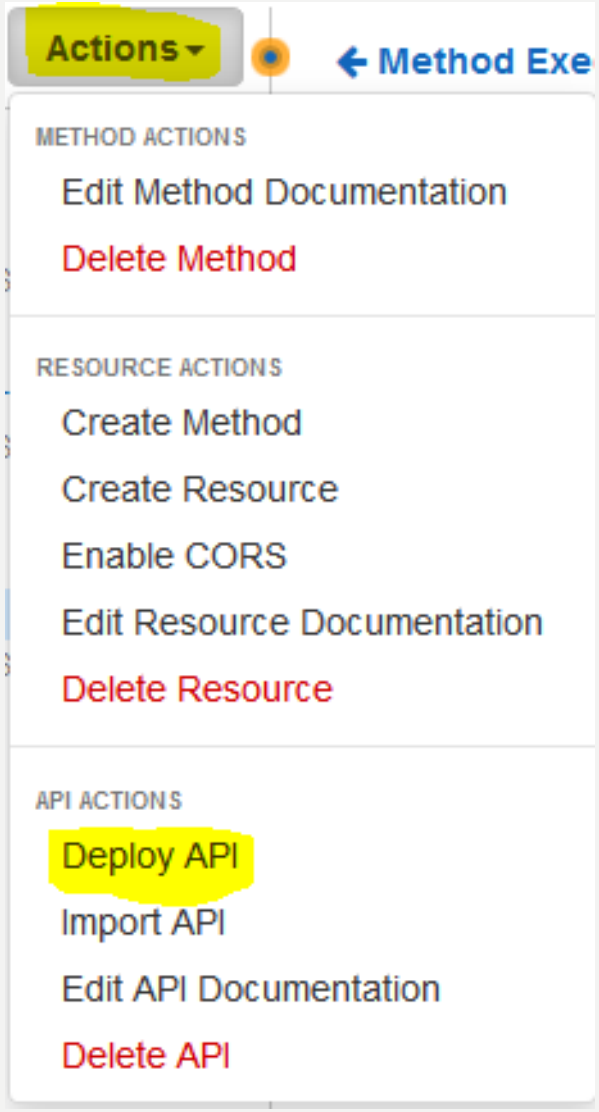
Enable CORS

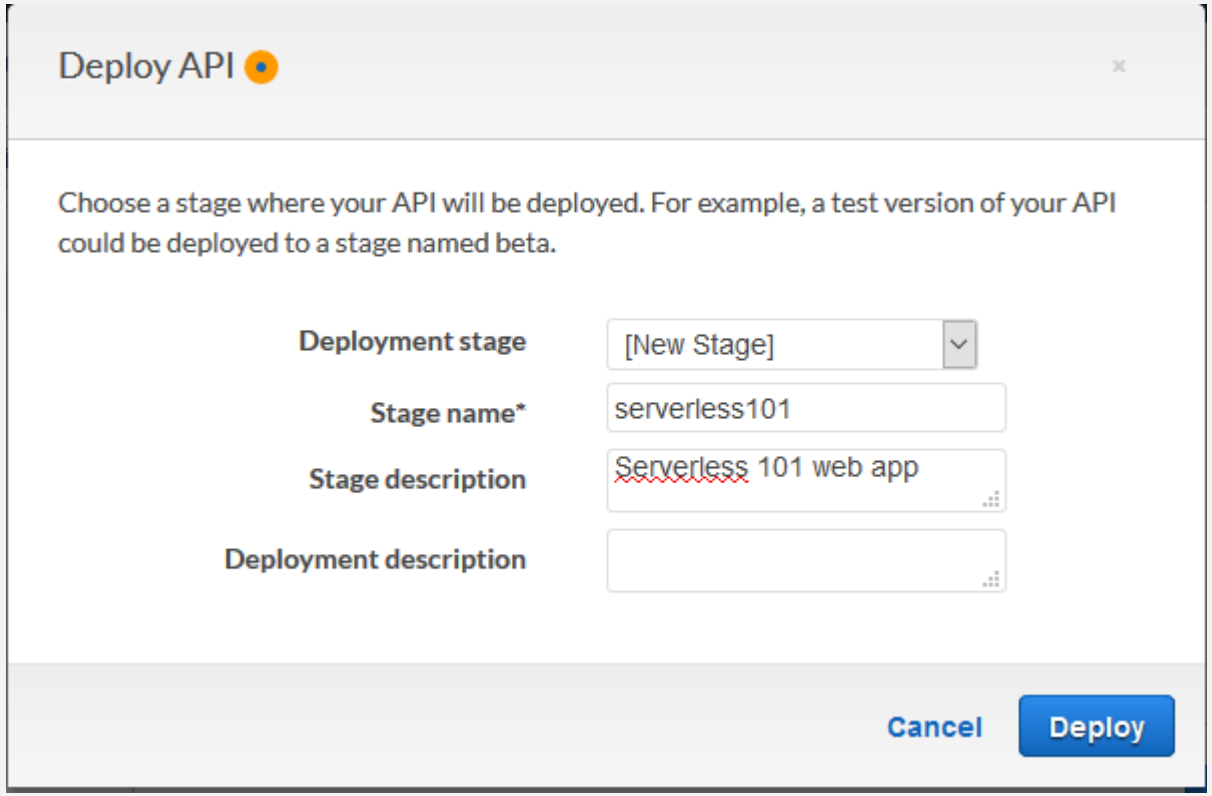


- Select a method and click "Enable CORS"
- On the next screen, leave everything as is and click "Enable CORS and replace existing headers"



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*

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

```
<div class="row">
    <div class="col-sm-4"></div>
    <div class="col-sm-4" style="margi
        <button type="button" class="b
        onclick="loginregister({email
            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.passwo
    {
        passwordValue = sha256(loginregister.password
        //API Endpoint - Replace this with endpoint y
        loginregisterurl = 'https://jthp9bhj27.execut
        var obj = new Object();
        obj.fullname = loginregister.fullname;
        obj.email = loginregister.email;
        obj.password = passwordValue;
        obj.location = loginregister.location;
        obj.comments = loginregister.comments;
```

loginregister() – serverless101.js

```
def loginregister(fullname, email, pa
    try:
        # Database connection paramet
        serverless101cnxstr = {'host'
        connection = mysql.connector.
        # Check if email already exis
        sql = "SELECT UserID FROM Use
        cursor = connection.cursor()
        cursor.execute(sql)
        userid = cursor.fetchall()
        if userid:
            return {"result" : False}
```

loginregister.py

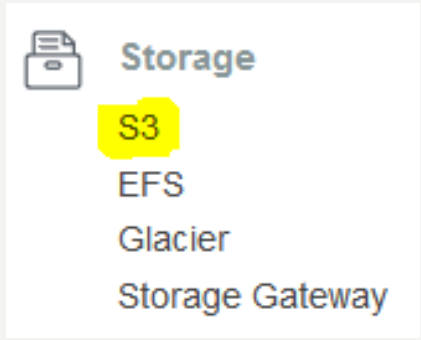
Enable the APIs – Edit the JS functions

```
/* Login Register */
function loginregister(loginregister) {
    if((loginregister.email) && (loginregister.password) && (loginregister.fullname) && (loginregister.comments) && (l
    {
        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){
```

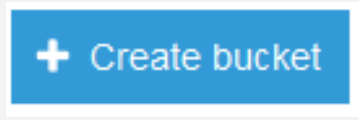
- Integrate each of these APIs with the relevant functions defined in serverless101.js to have them eventually invoked

Let's host the web files

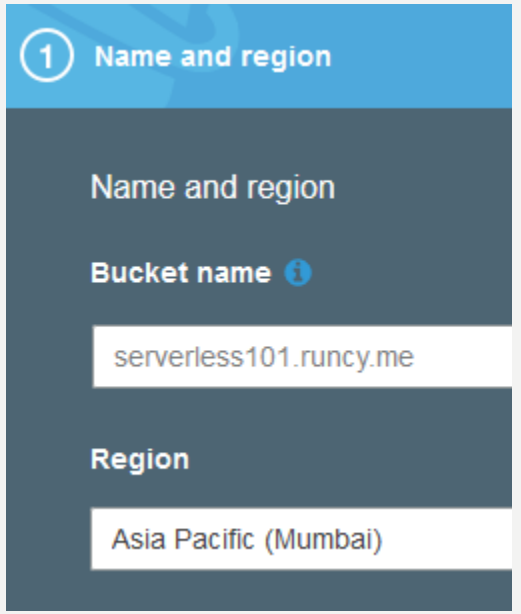
- Select "S3" from Storage category



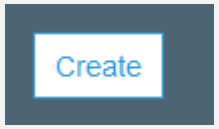
- Click "Create bucket"



- Provide appropriate name (a subdomain or domain that you own for host hosting the site)

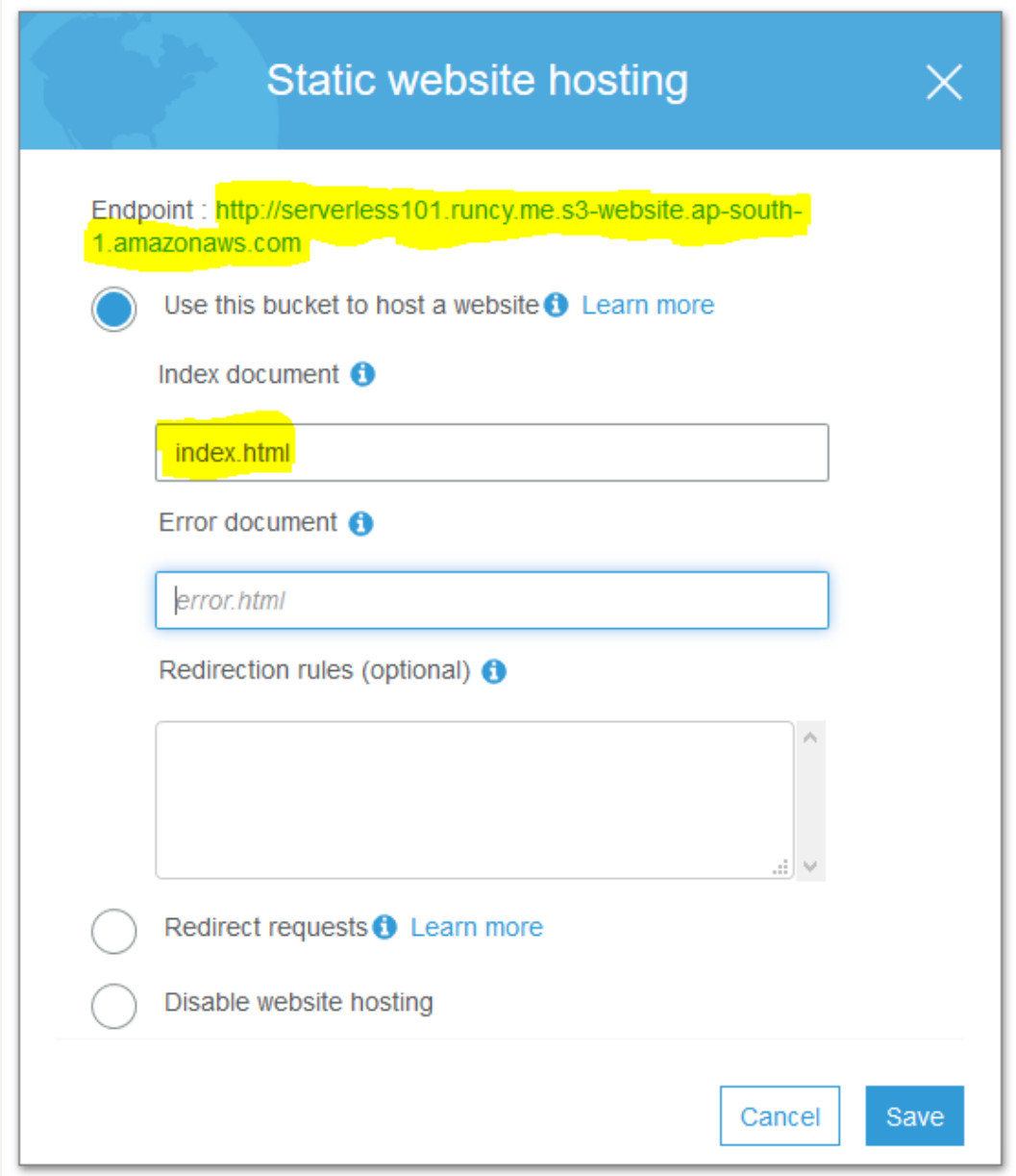


- Click "Create"



Enable Static Website Hosting

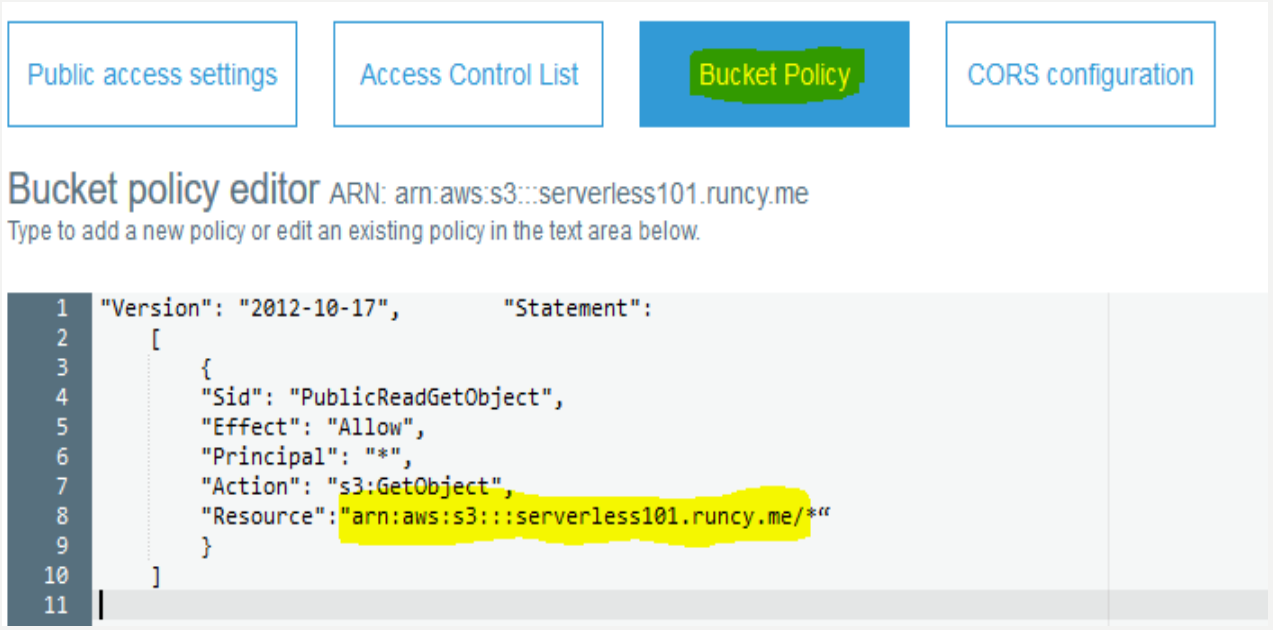
- Select the bucket that you created earlier
- From the "*Properties*" tab select ***Static website hosting***
- Provide appropriate *Index document* and hit *Save*
- *You will now see an endpoint available which will serve you the website contents*



Enable appropriate Bucket Policy

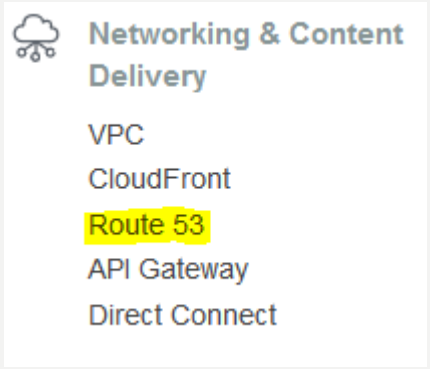
- Click on the "*Permissions*" tab
- Select "Bucket Policy" sub-tab
- *Enter the below policy to make it world readable*

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



Let's setup DNS

- Select "Route 53" from Networking category
- Select your *Hosted Zone* for the website*

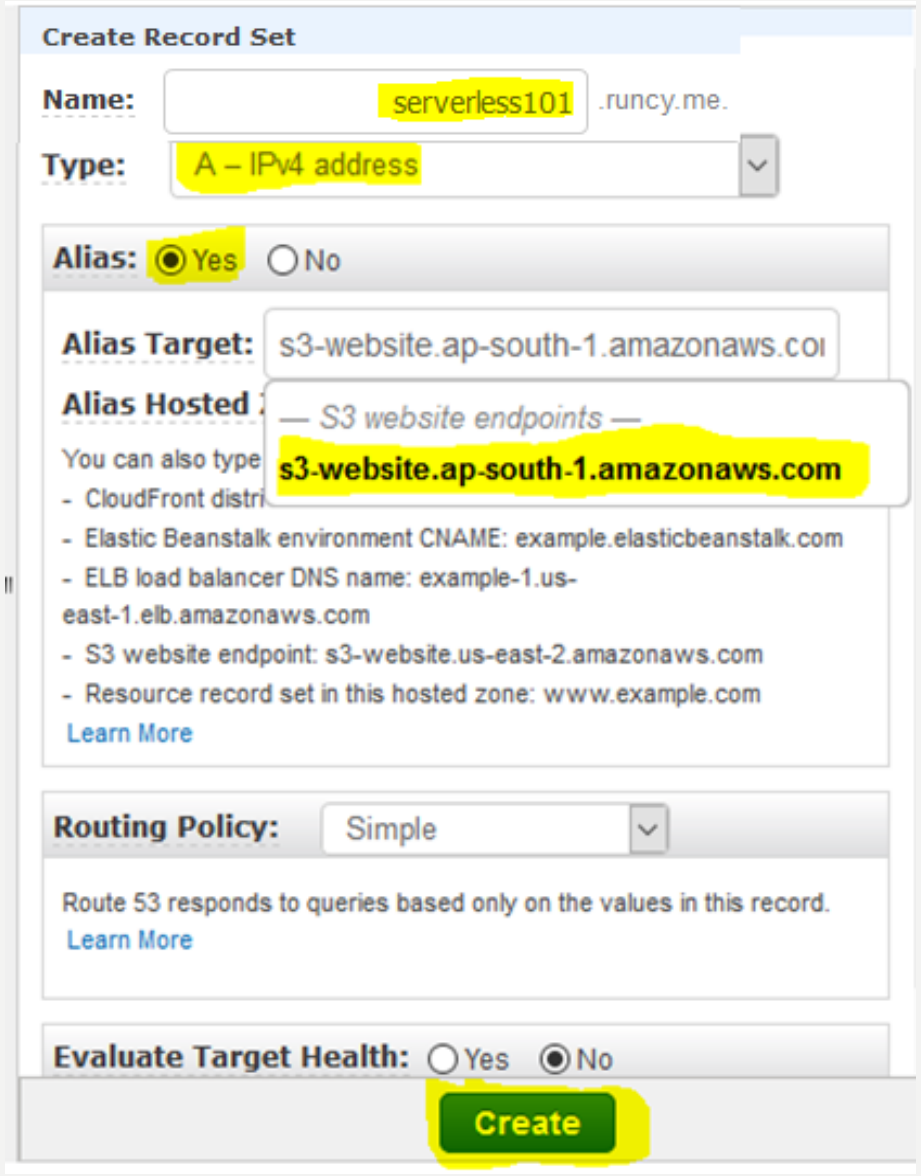


- Click "Create Record Set"

* 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)



Your web app is *now* LIVE!