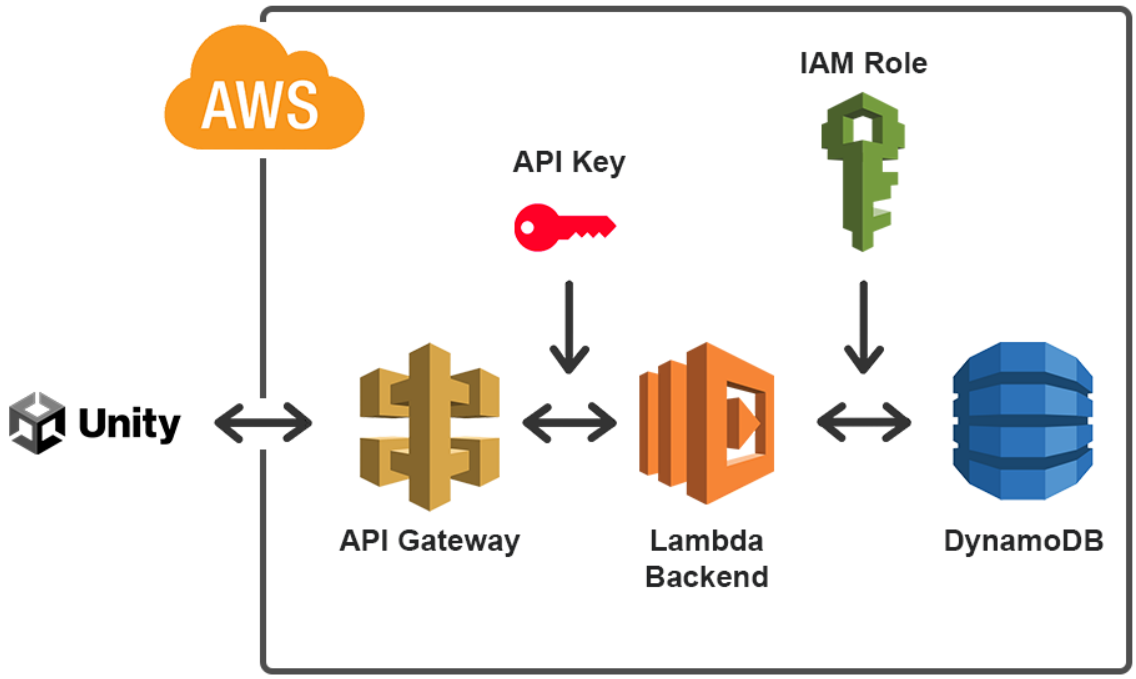
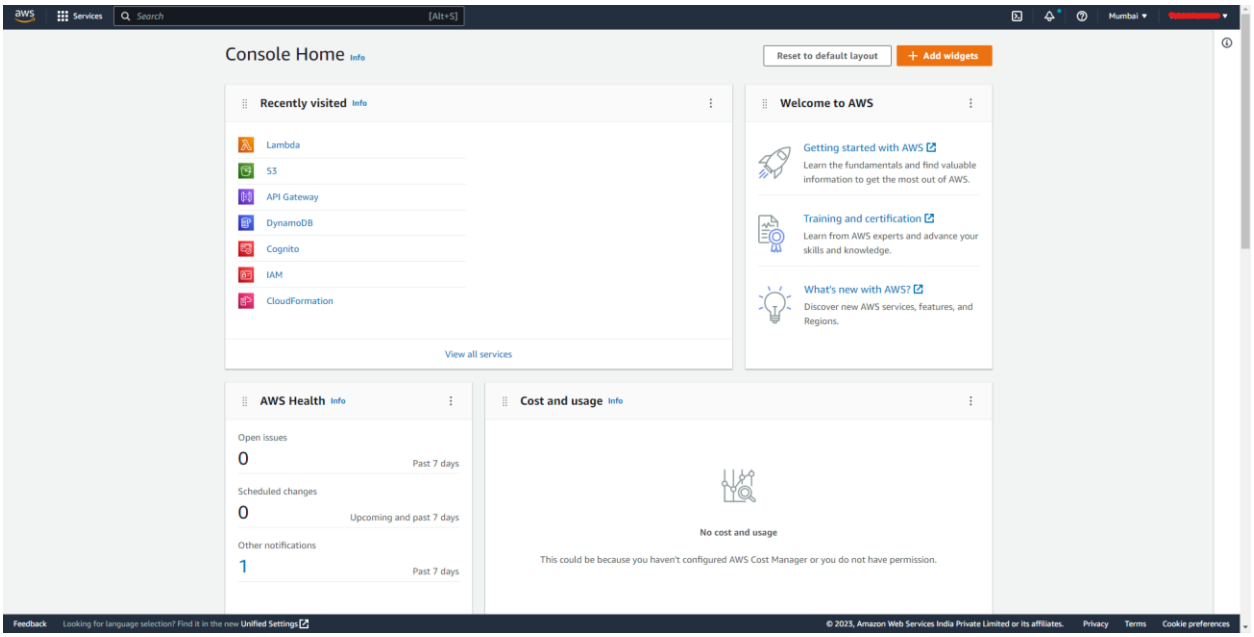
AWS DynamoDB and Server-less Backend Integration with Unity



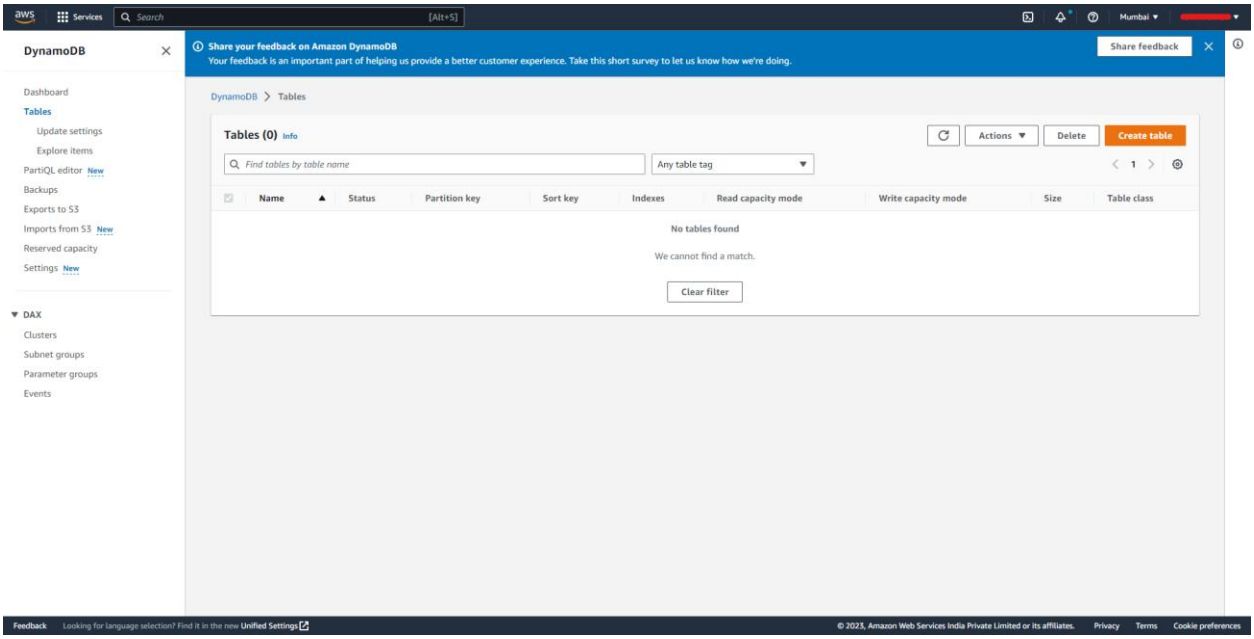
Process Blueprint

1. Unity sends web request to one of the endpoints in API Gateway
2. The web request must have the API Key as one of the headers
3. API Gateway then directs the request to Lambda Backend
4. Lambda Backend needs a Default Execution Role to function
5. Lambda then does the necessary operation (register, login, verify etc.) by accessing the DynamoDB
6. The data is retrieved / updated and the status is returned as JSON object
7. JSON object is parsed in Unity

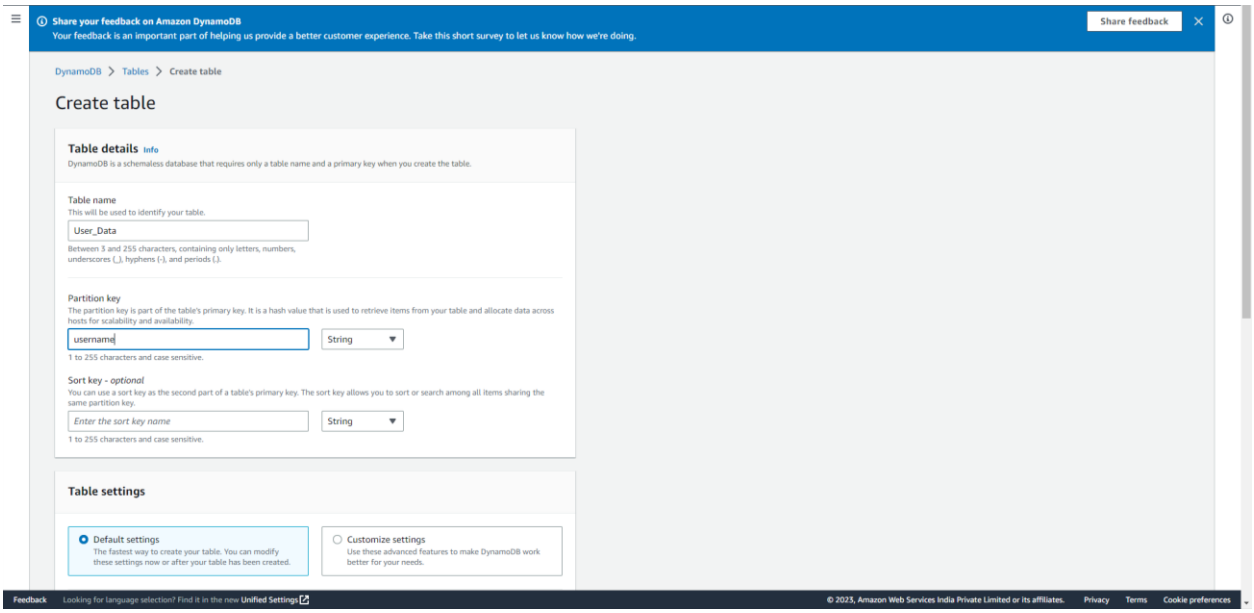
Login into AWS Console



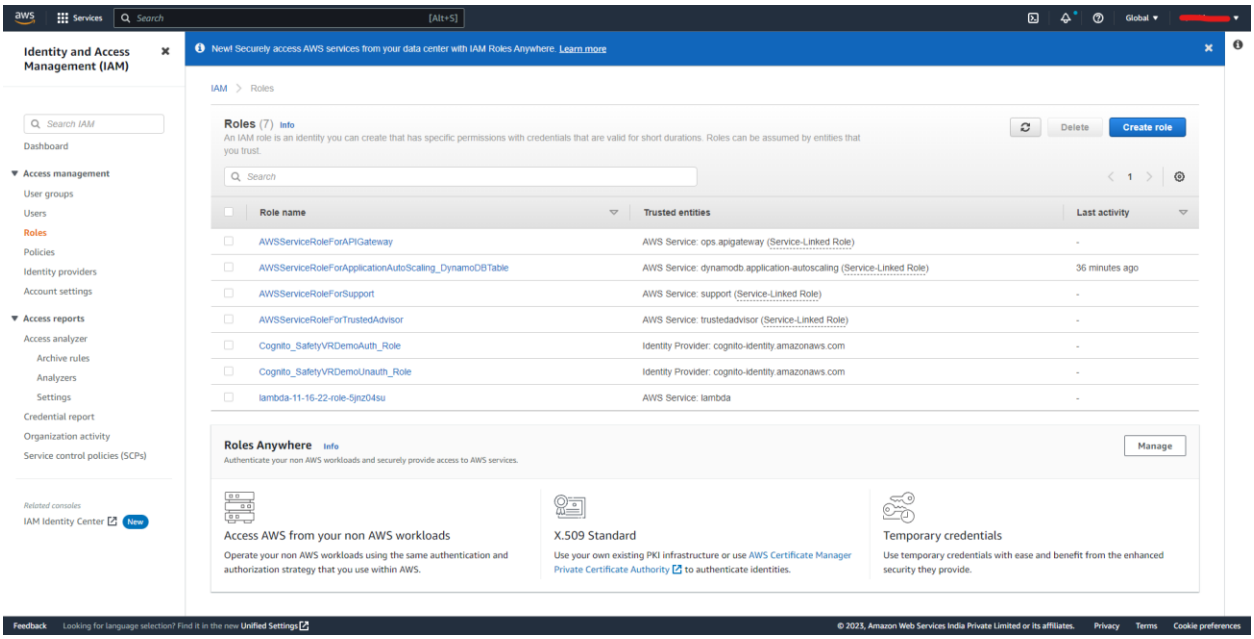
Creating DynamoDB Table

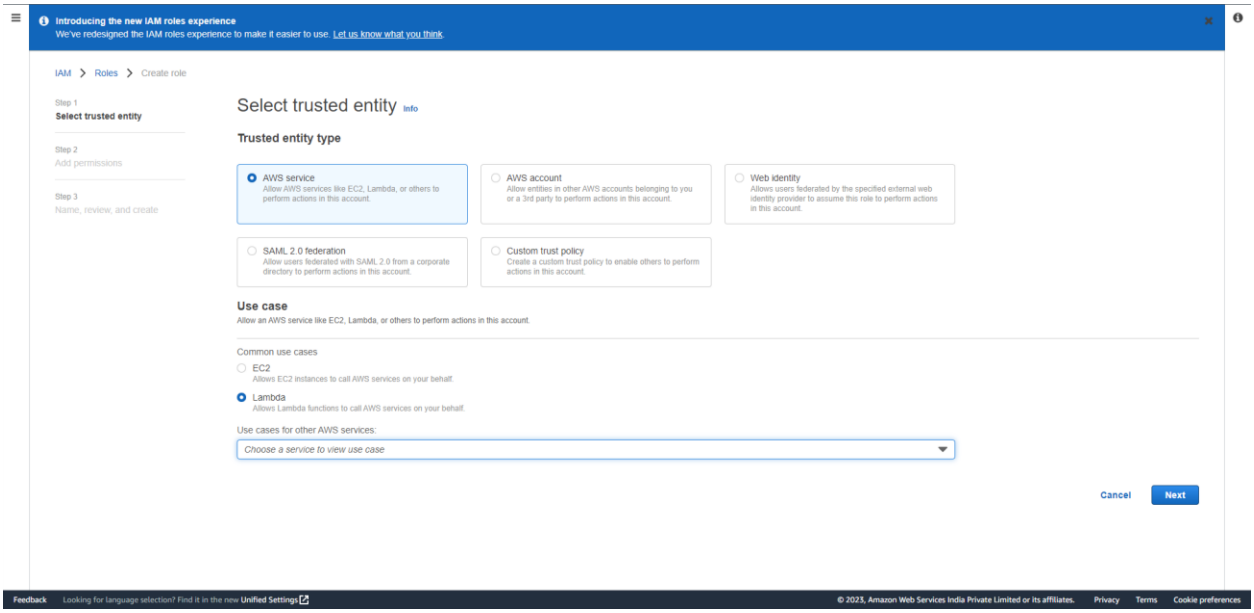


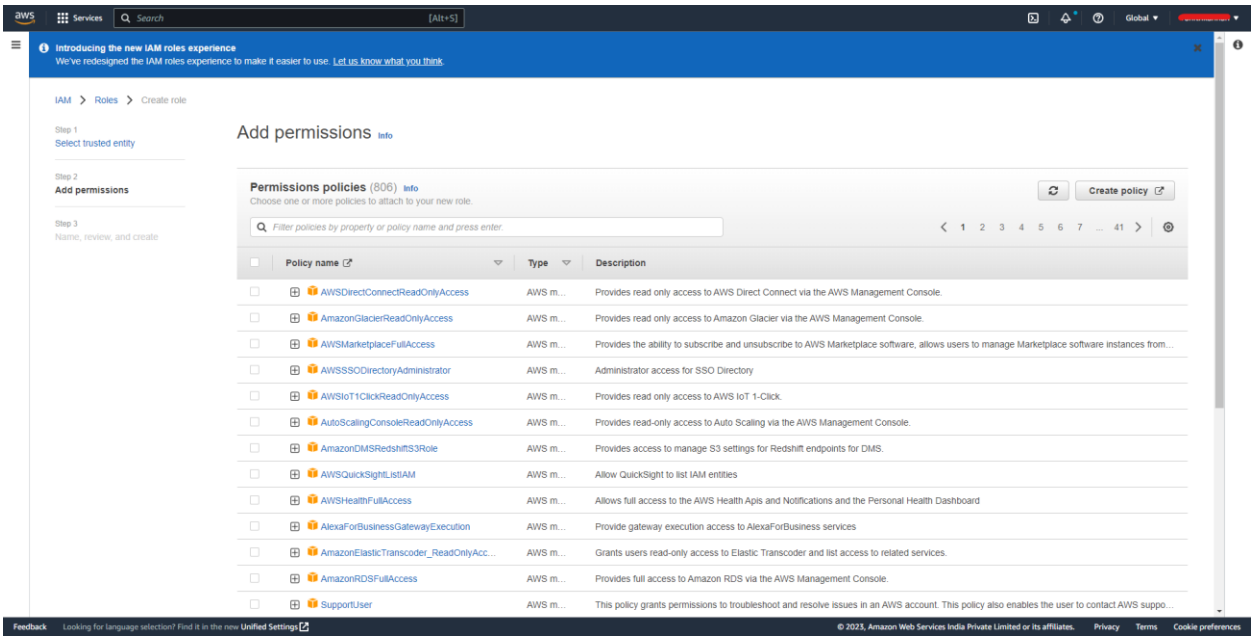
Go to DynamoDB service > Tables and then Create Table



Creating IAM Role

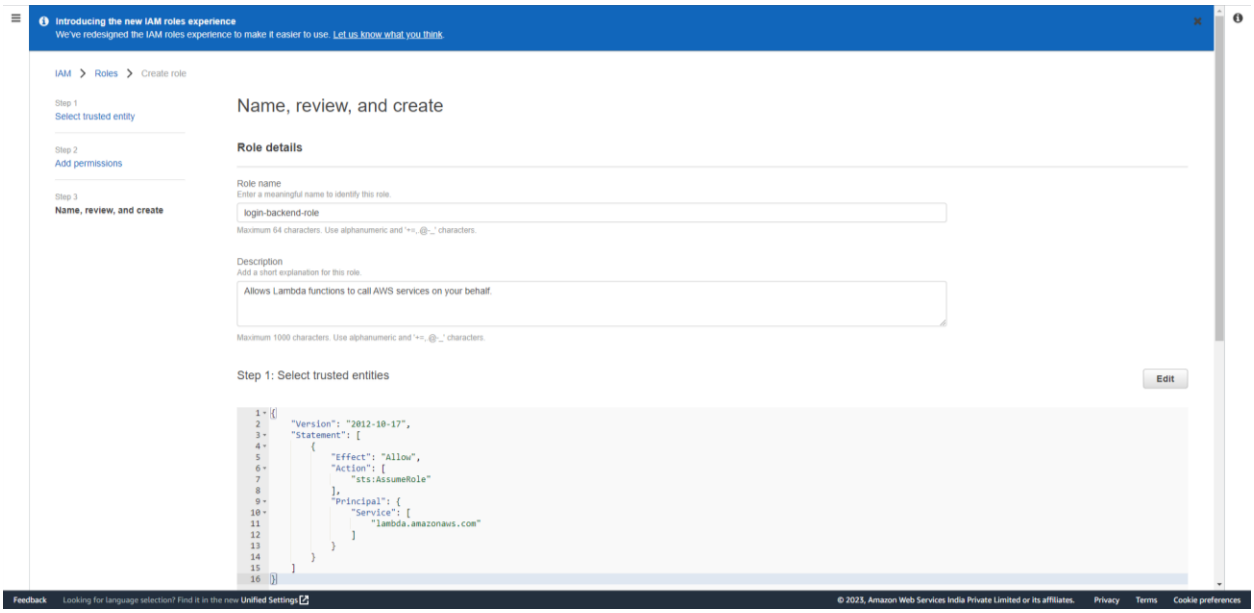






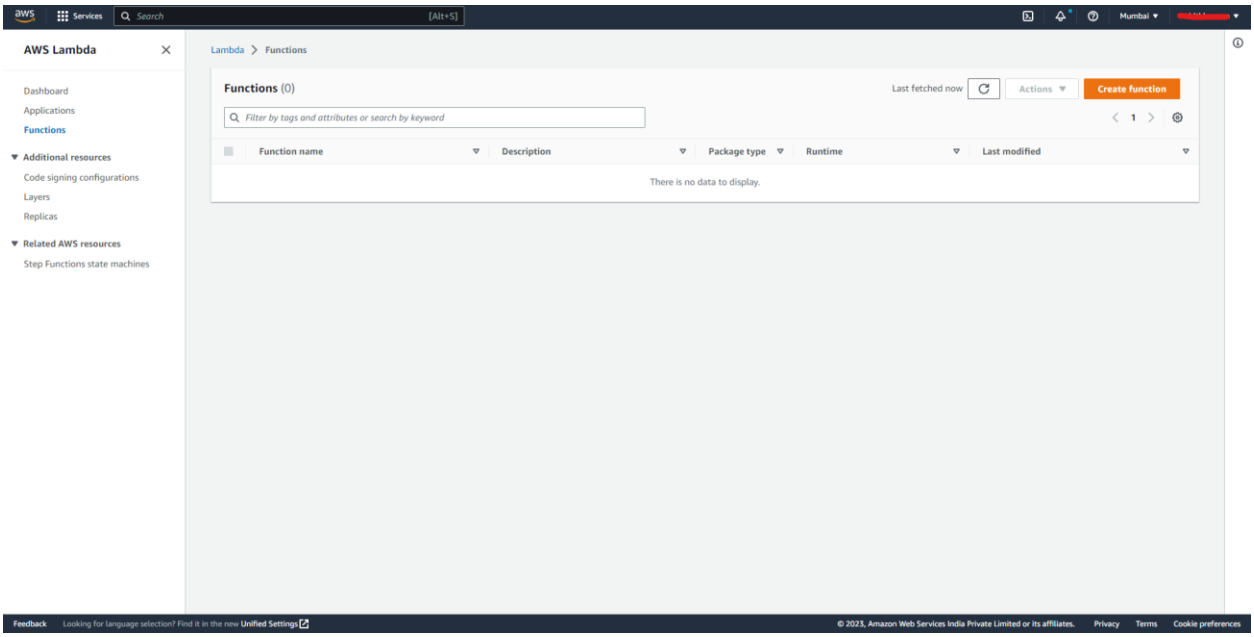
Add Policies

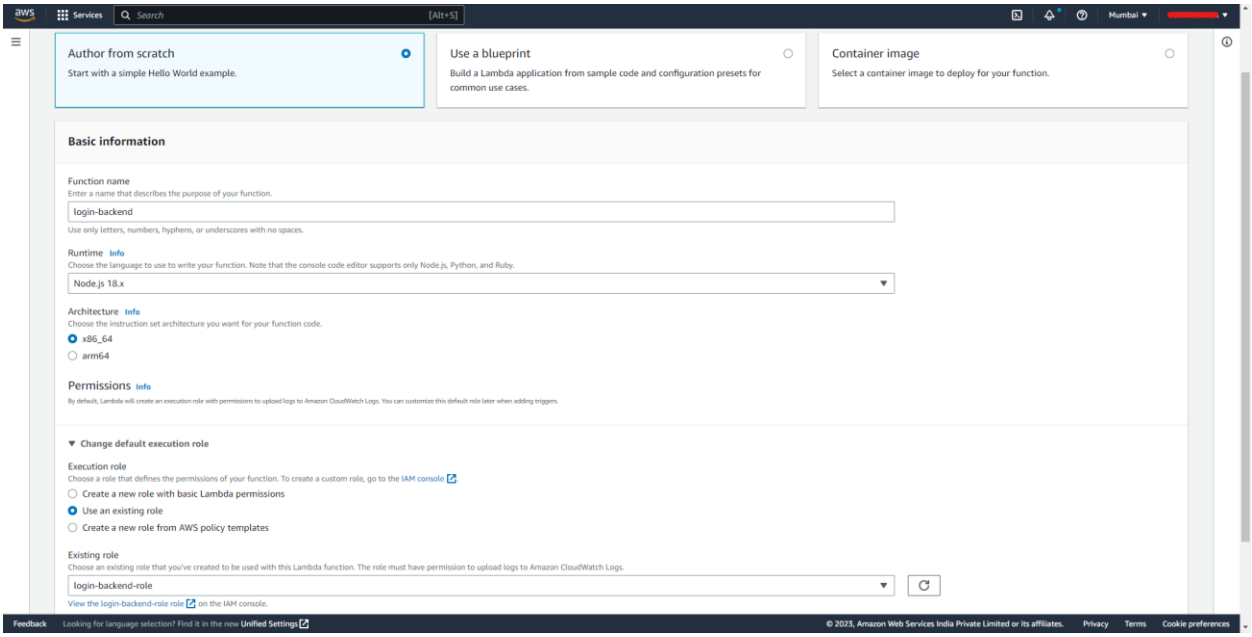
1. *CloudWatchLogsFullAccess*
2. *AmazonDynamoDBFullAccess*

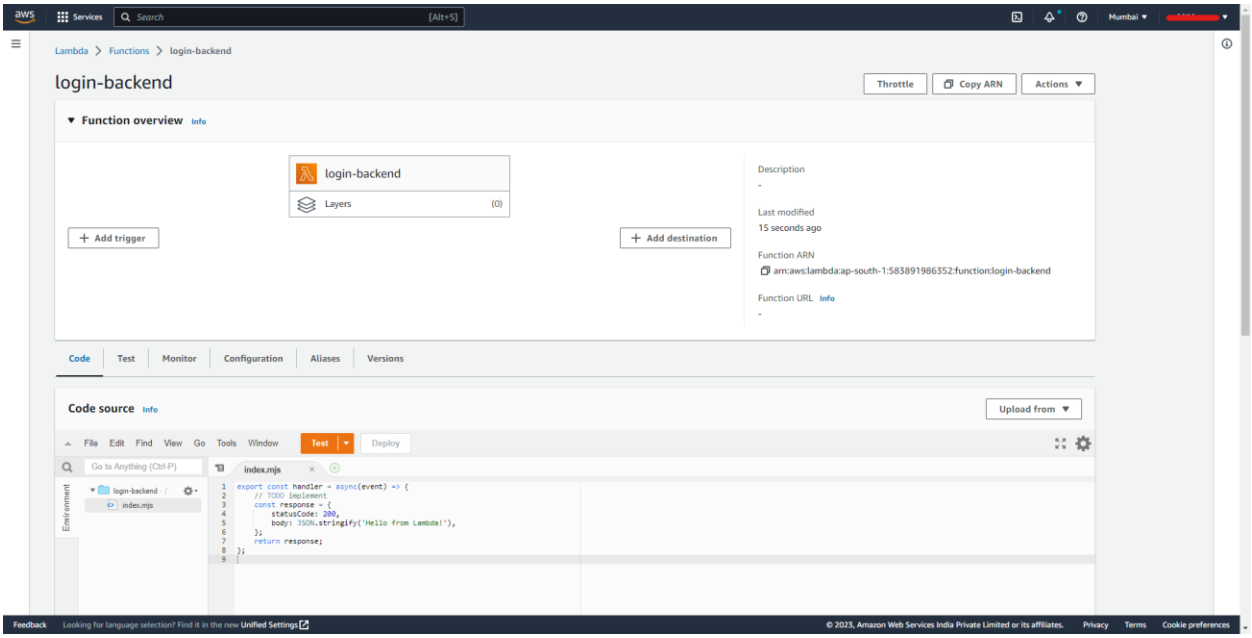


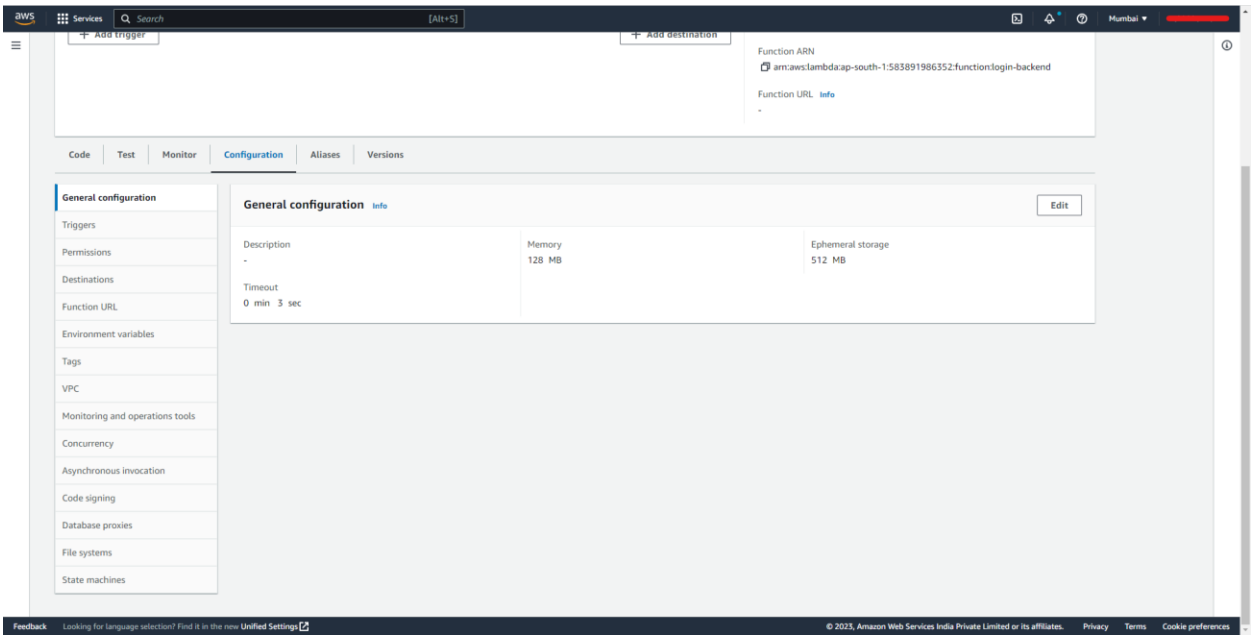
Add Role Name and then create role

Creating Lambda function




Author from scratch
Start with a simple Hello World example.
Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.
Container image
Select a container image to deploy for your function.
Basic information
Function name
Enter a name that describes the purpose of your function.
login-backend
Use only letters, numbers, hyphens, or underscores with no spaces.
Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Node.js 18.x
Architecture Info
Choose the instruction set architecture you want for your function code.
x86_64
arm64
Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.
Change default execution role
Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.
Create a new role with basic Lambda permissions
Use an existing role
Create a new role from AWS policy templates
Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
login-backend-role
View the login-backend-role role on the IAM console.
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2023, Amazon Web Services India Private Limited or its affiliates.
Privacy
Terms
Cookie preferences

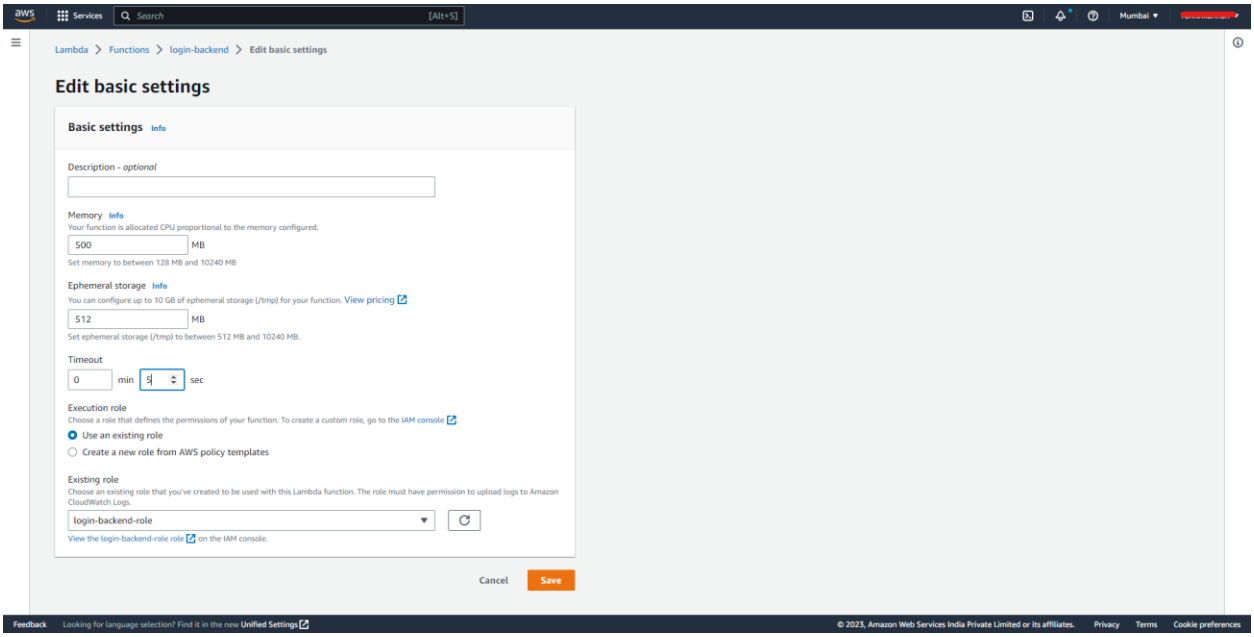

Lambda > Functions > login-backend
login-backend
Throttle
Copy ARN
Actions
Function overview Info
login-backend
Layers (0)
Add trigger
Add destination
Description
-
Last modified
15 seconds ago
Function ARN
arn:aws:lambda:ap-south-1:583891986352:function:login-backend
Function URL Info
-
Code
Test
Monitor
Configuration
Aliases
Versions
Code source Info
Upload from
File Edit Find View Go Tools Window
Test
Deploy
Go to Anything (Ctrl-P)
index.mjs
export const handler = async(event) => {
// TODO implement
const response = {
statusCode: 200,
body: JSON.stringify('Hello from Lambda!'),
};
return response;
};
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2023, Amazon Web Services India Private Limited or its affiliates.
Privacy
Terms
Cookie preferences



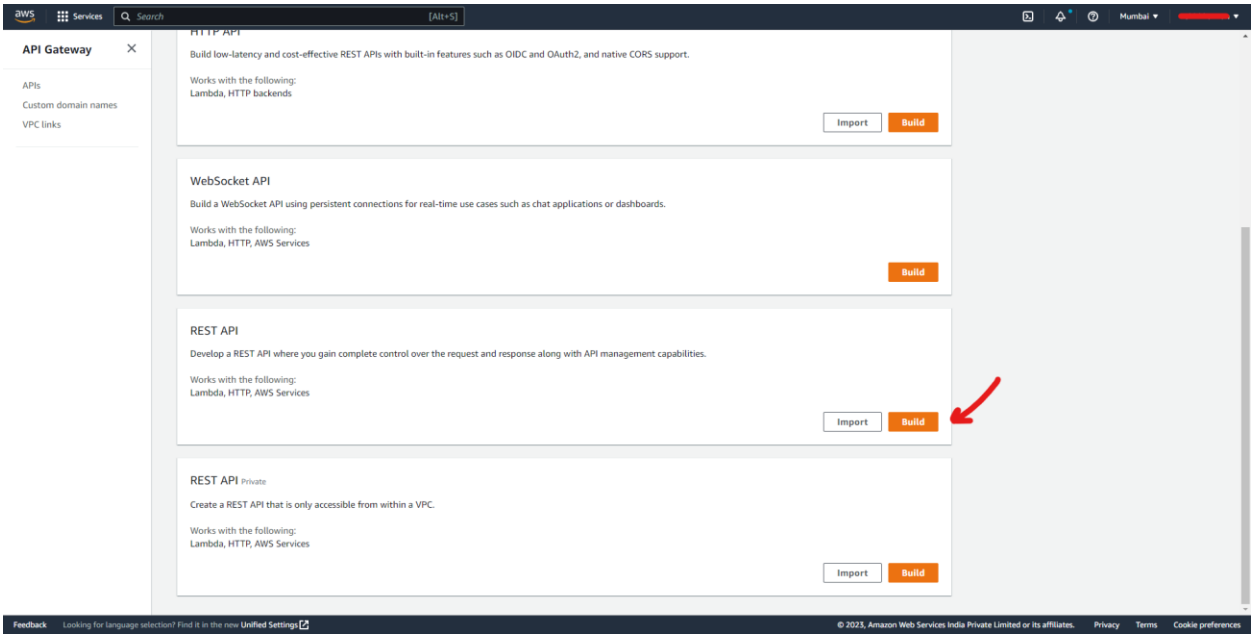
Edit General Configuration (Optional)

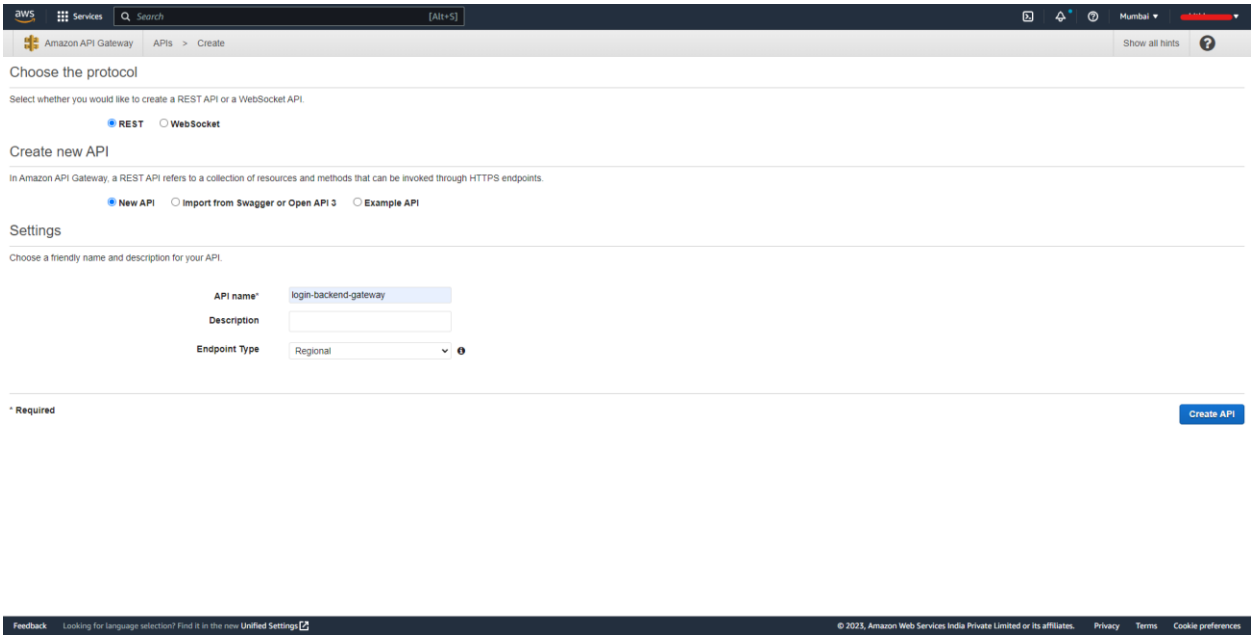
500 MB Memory

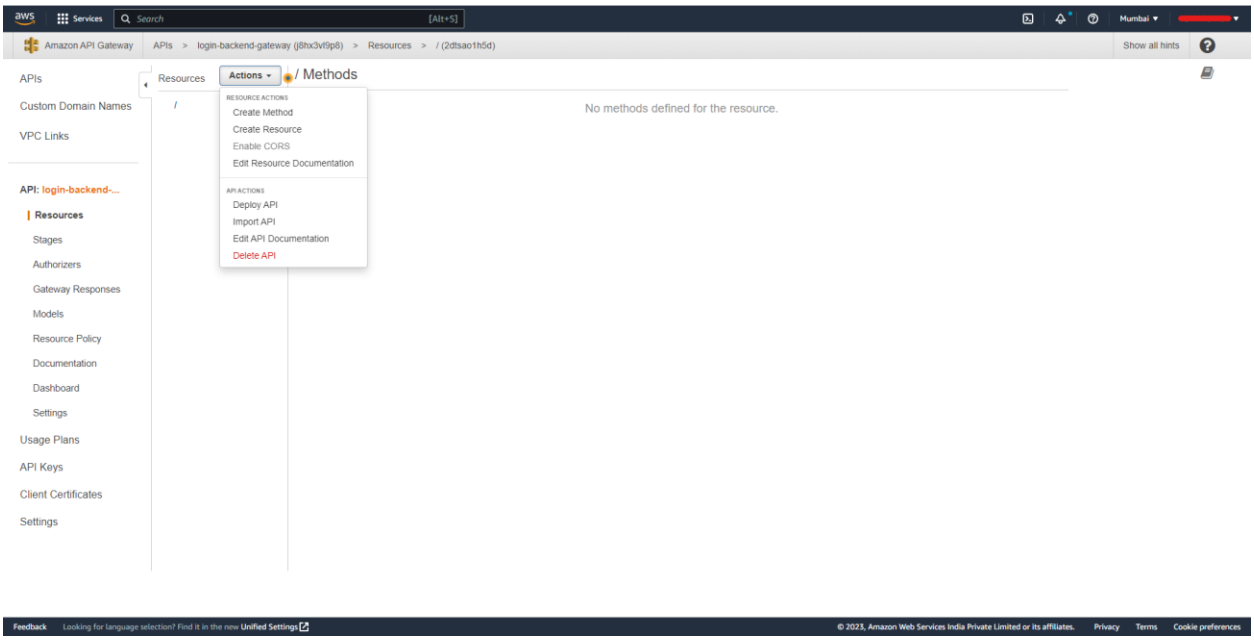
5 Sec Timeout

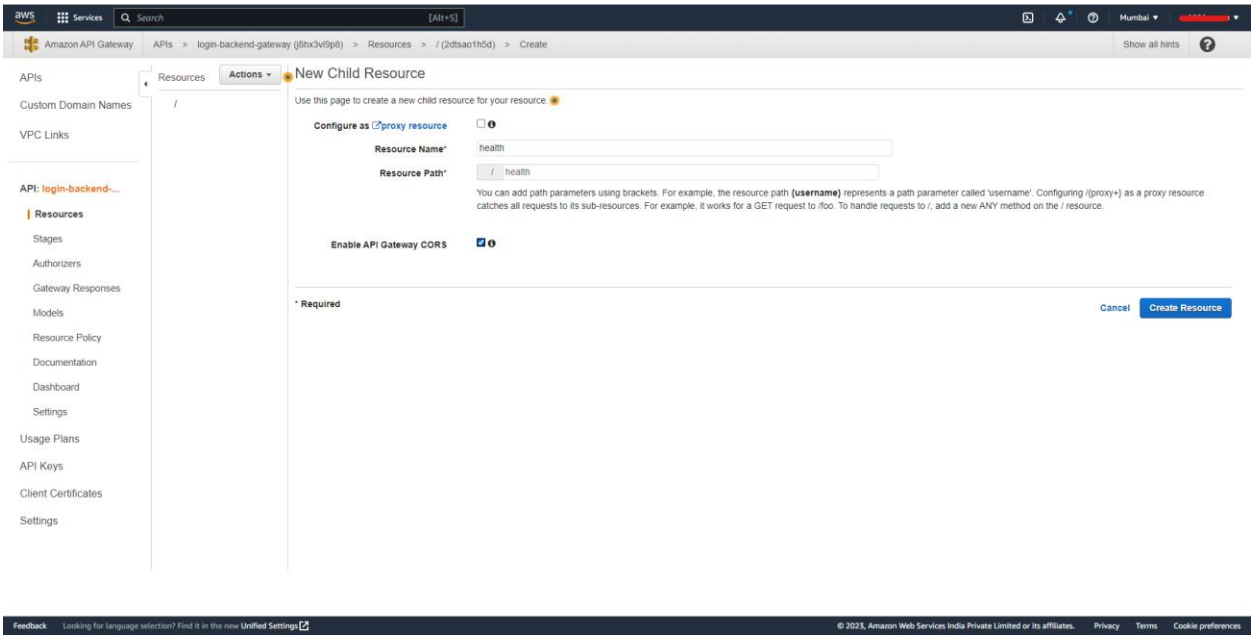


Creating API Gateway



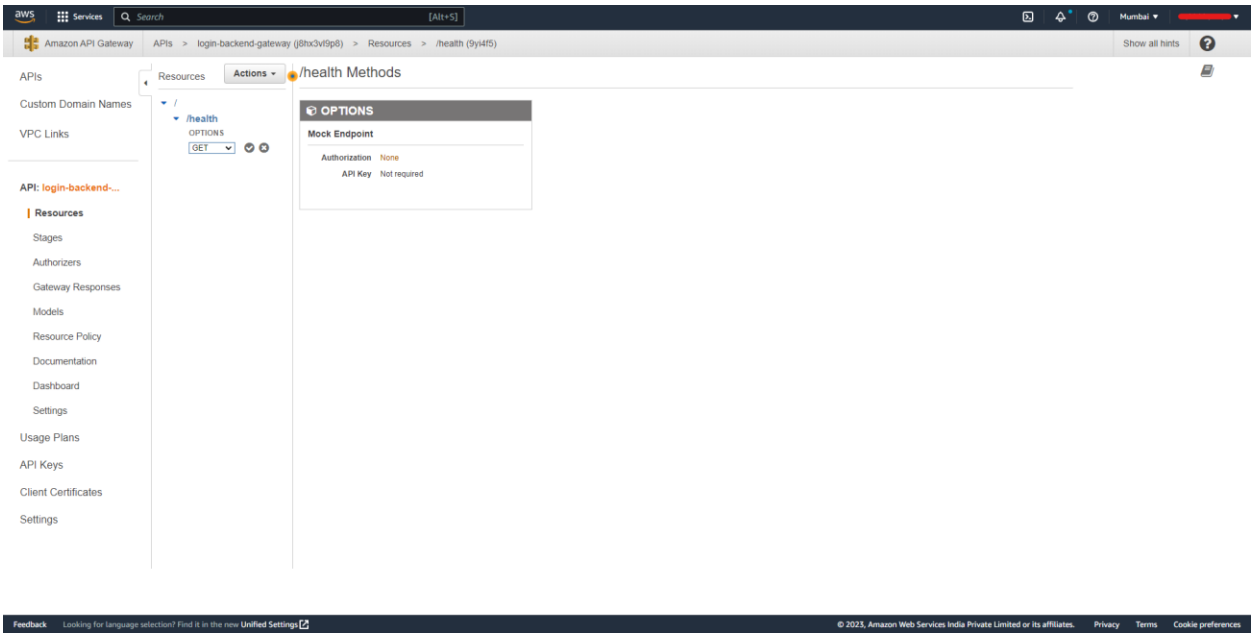






*Create New Resource "**health**"*

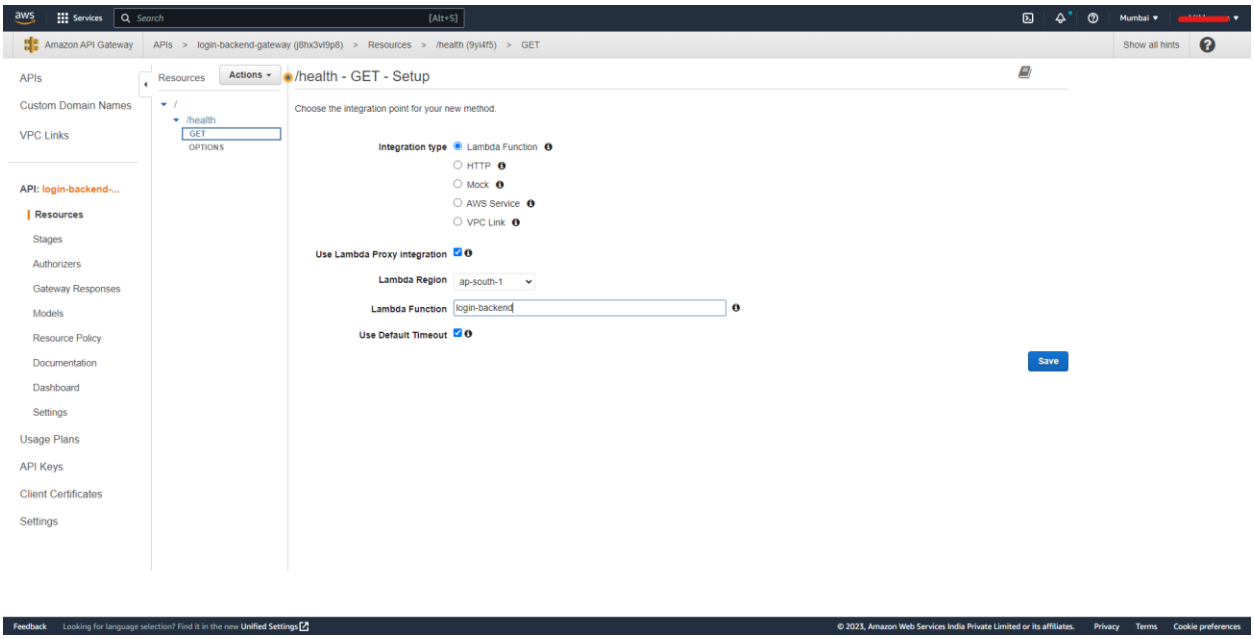
Enable API Gateway CORS



*Select the "**health**" resource*

Actions > Create Method

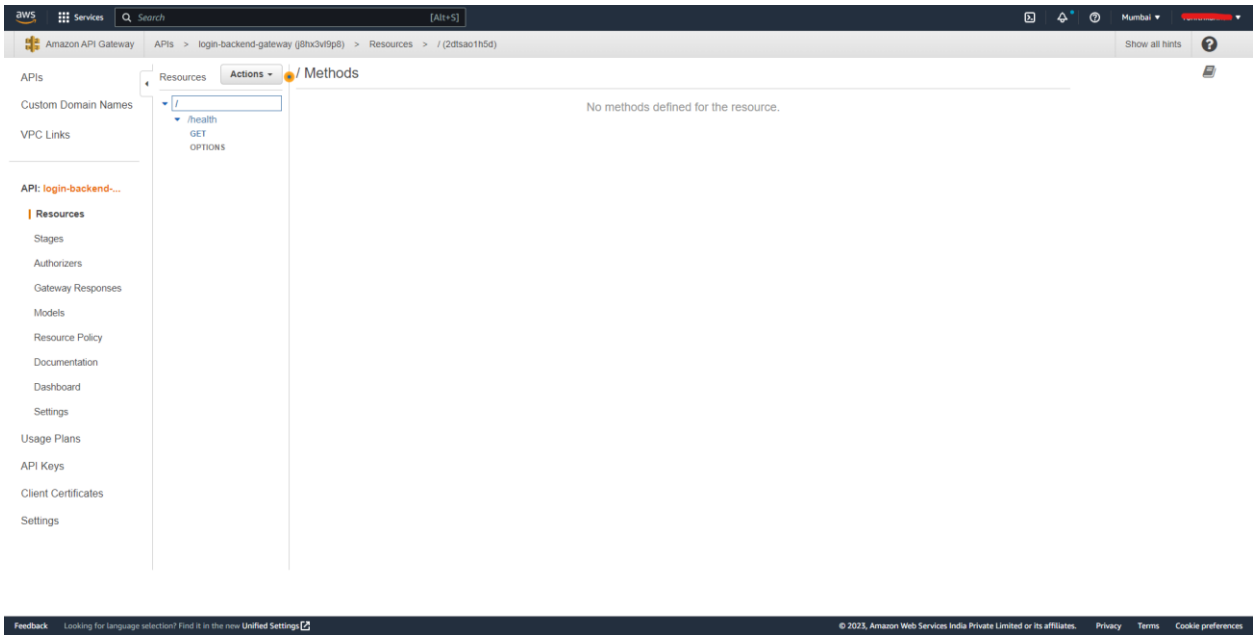
*Set Method to "**GET**"*

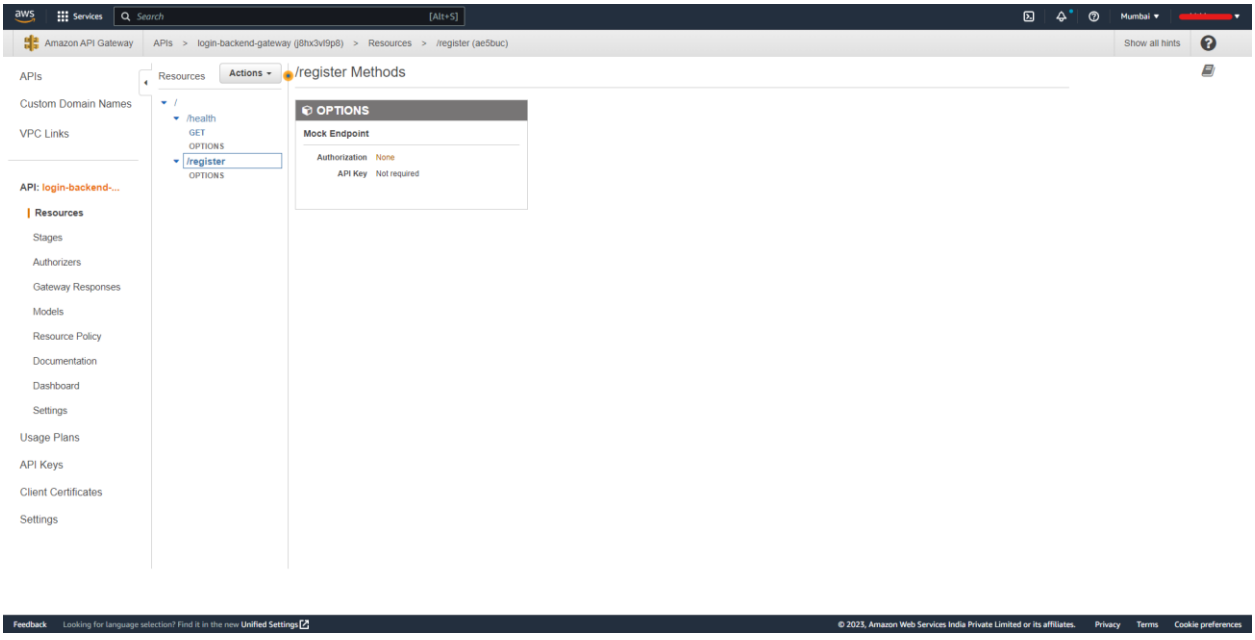


*Now, repeat the above steps for creating Resources "**register**", "**login**", "**verify**"*

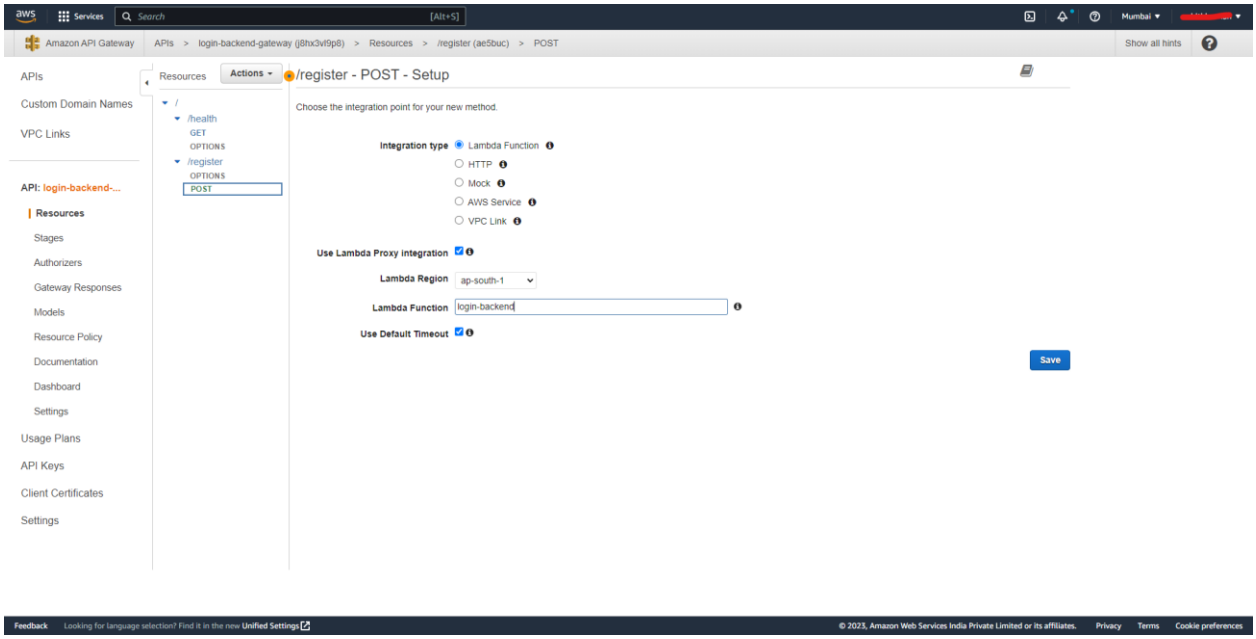
*With method as "**POST**"*

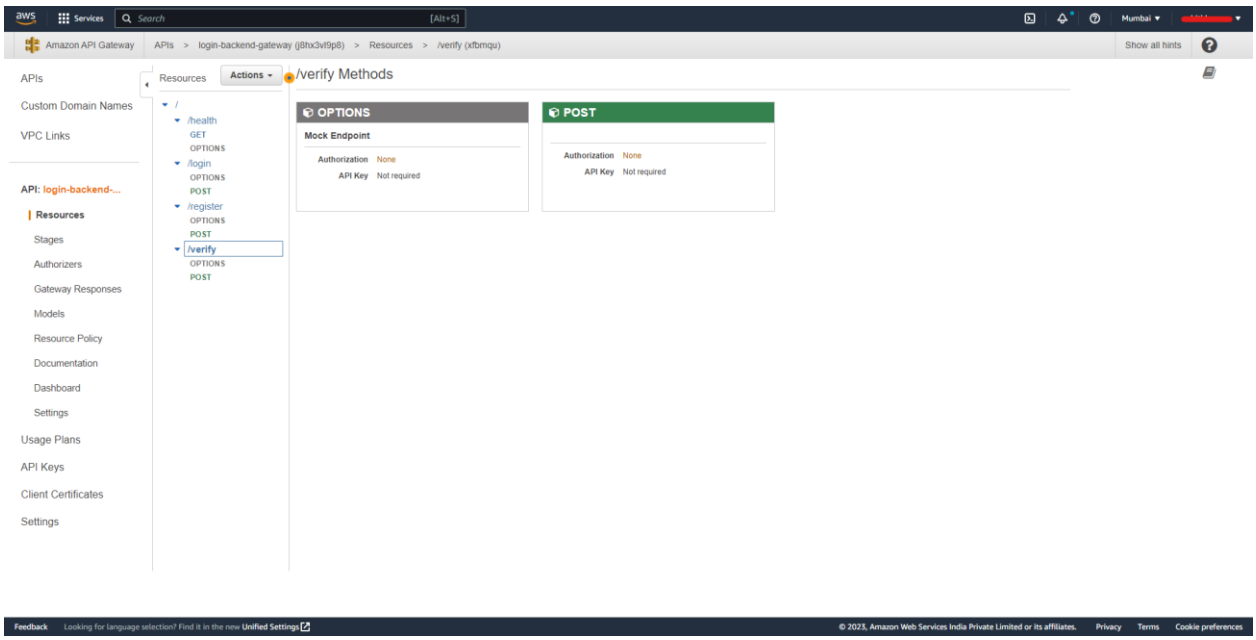
*Select **/** (root) and then Actions > Create Resource*





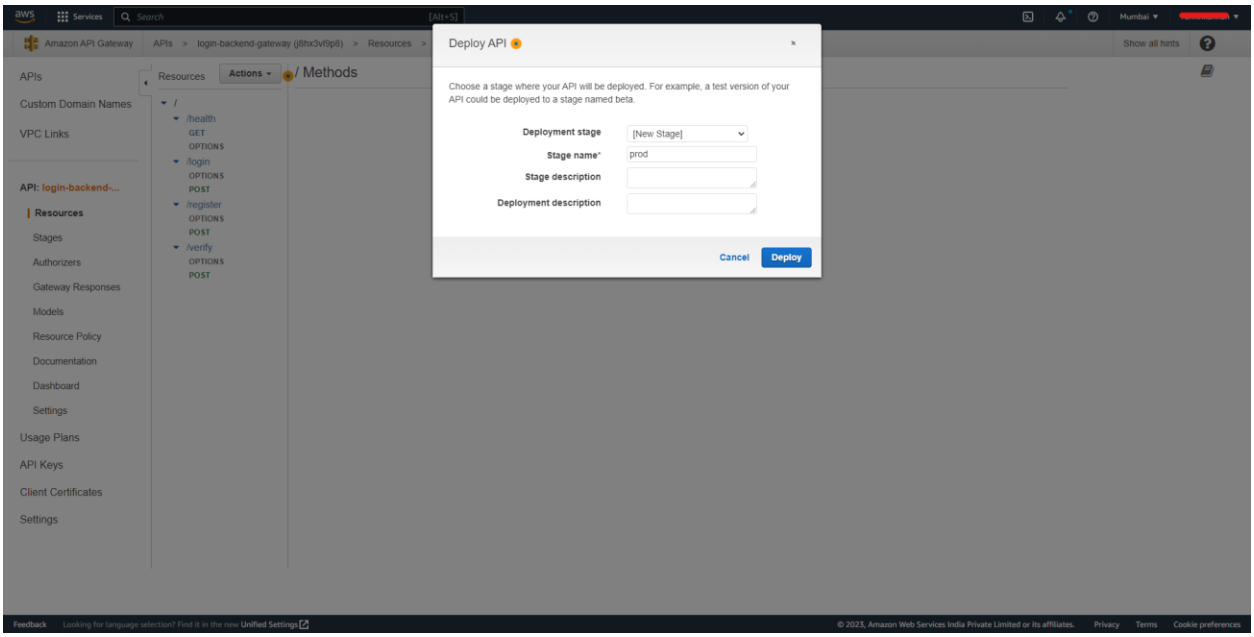
*Select **/register** and then Actions > Create Resource*



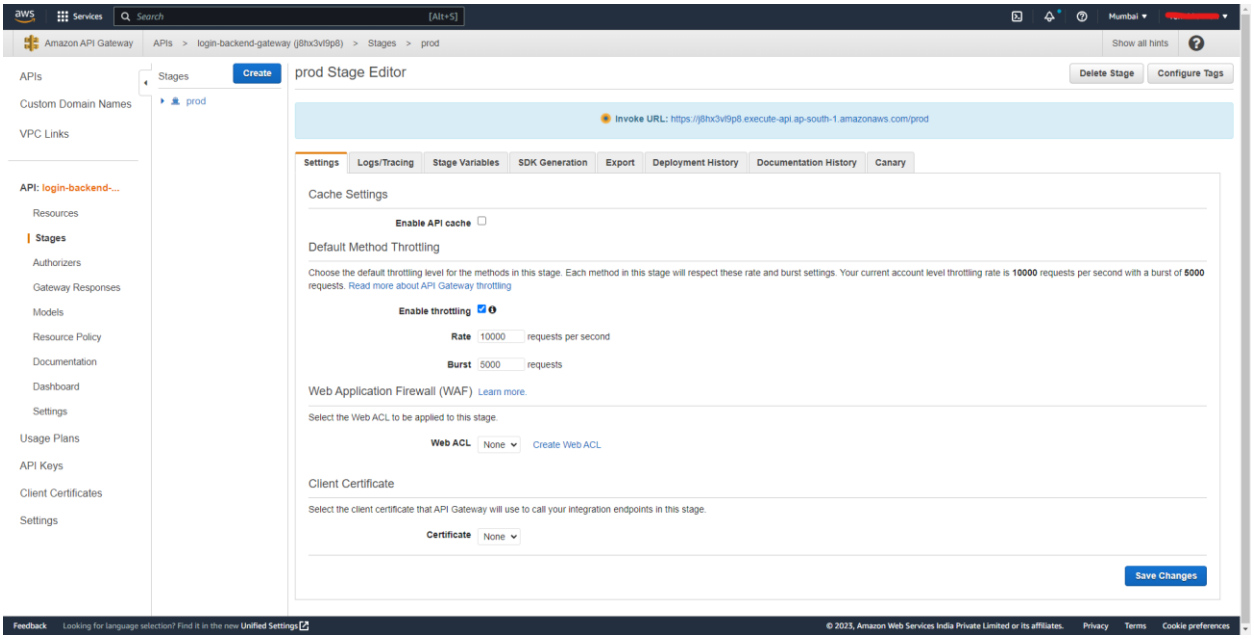


All four resources with methods have been created

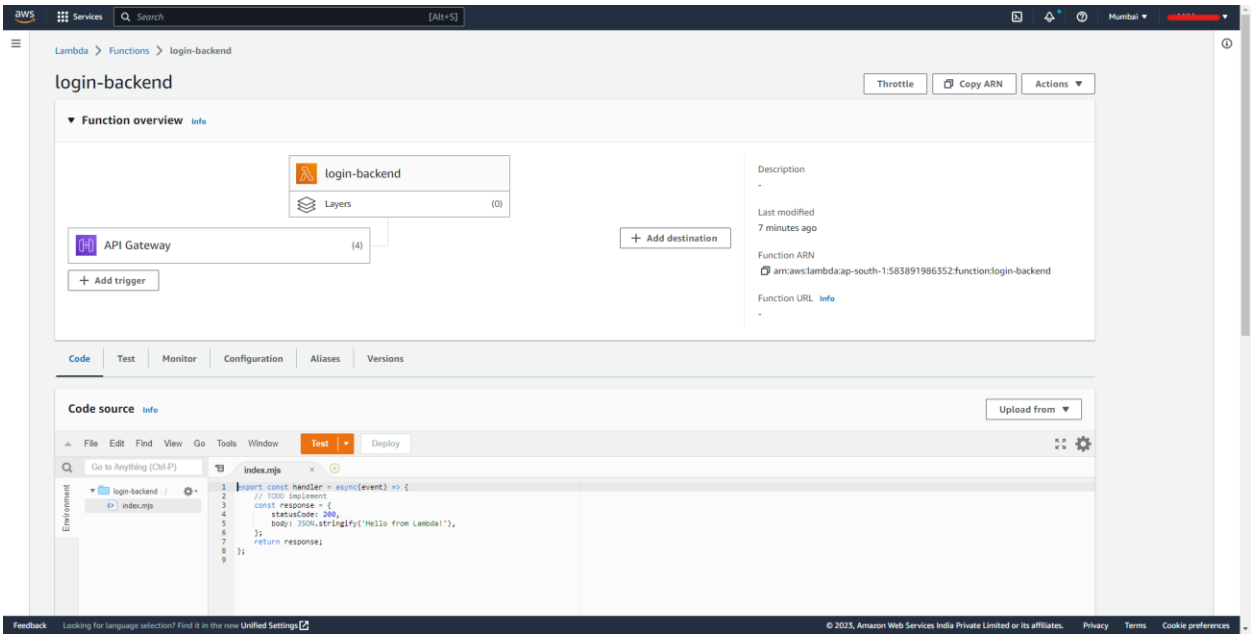
Select “ / ” (root) and then Actions > Deploy API



API Deployed



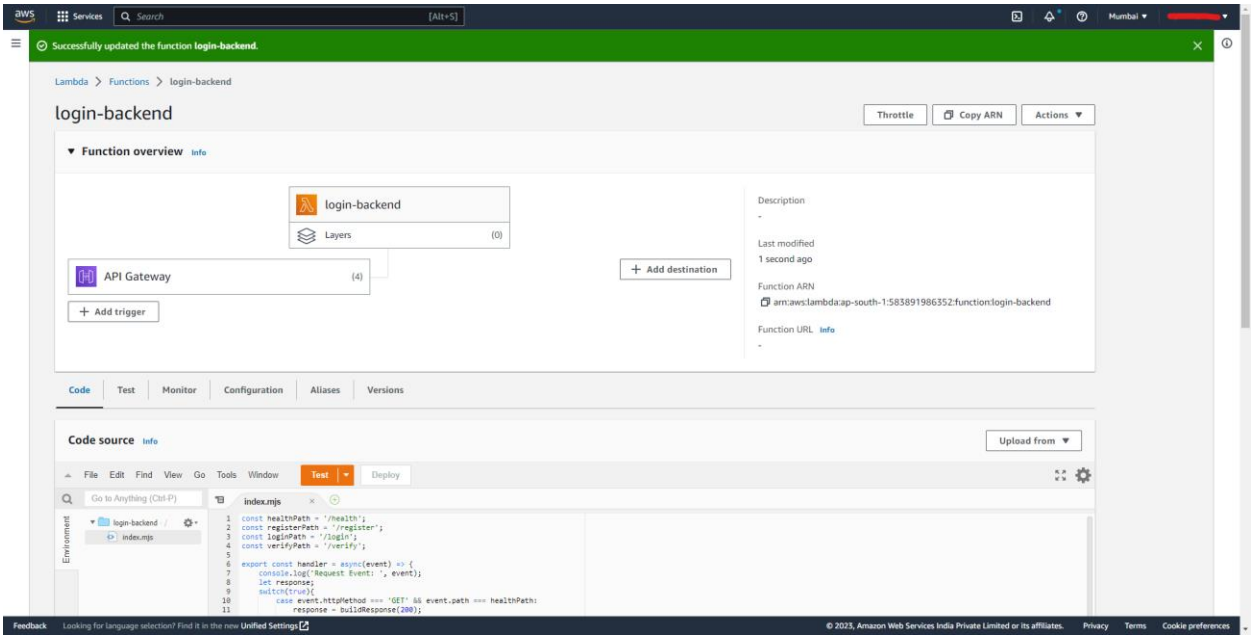
Testing the API



Modify index.js with the below program

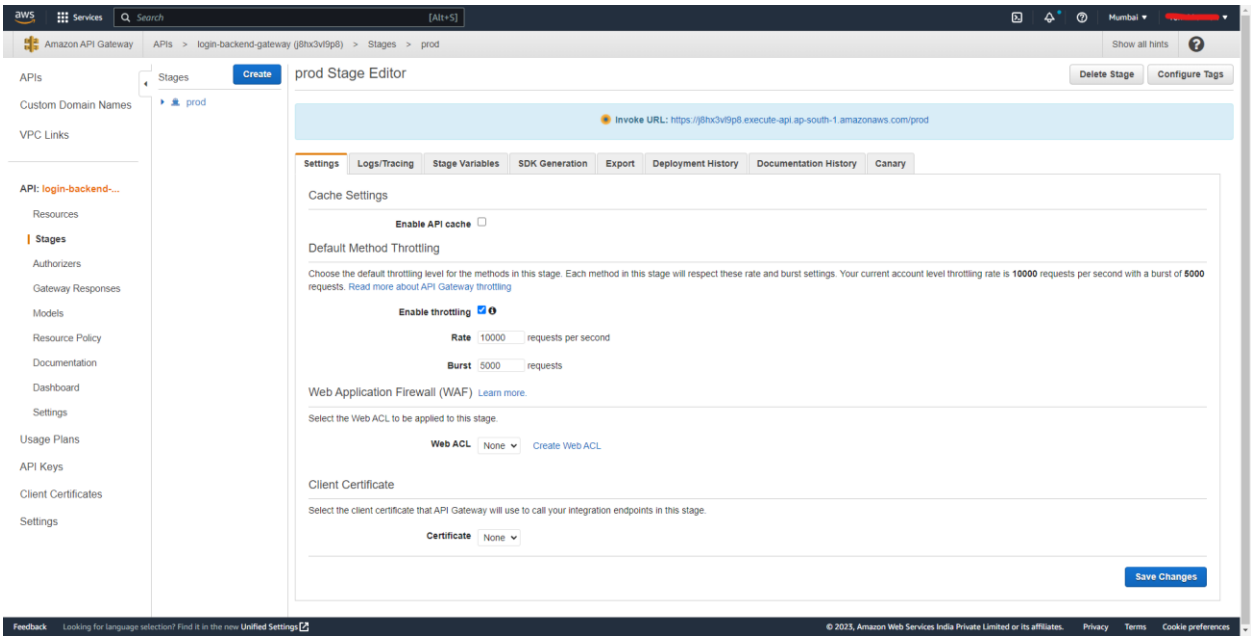
```
const healthPath = '/health';
const registerPath = '/register';
const loginPath = '/login';
const verifyPath = '/verify';

export const handler = async(event) => {
    console.log('Request Event: ', event);
    let response;
    switch(true){
        case event.httpMethod === 'GET' && event.path === healthPath:
            response = buildResponse(200);
            break;
        case event.httpMethod === 'POST' && event.path === registerPath:
            response = buildResponse(200);
            break;
        case event.httpMethod === 'POST' && event.path === loginPath:
            response = buildResponse(200);
            break;
        case event.httpMethod === 'POST' && event.path === verifyPath:
            response = buildResponse(200);
            break;
        default:
            response = buildResponse(404, '404 Not Found');
    }
    return response;
};

function buildResponse(statusCode, body) {
    return {
        statusCode : statusCode,
        headers : {
            'Access-Confirm-Allow-Origin' : '*',
            'Content-Type' : 'application/json',
        },
        body : JSON.stringify(body)
    }
}
```

Deploy changes

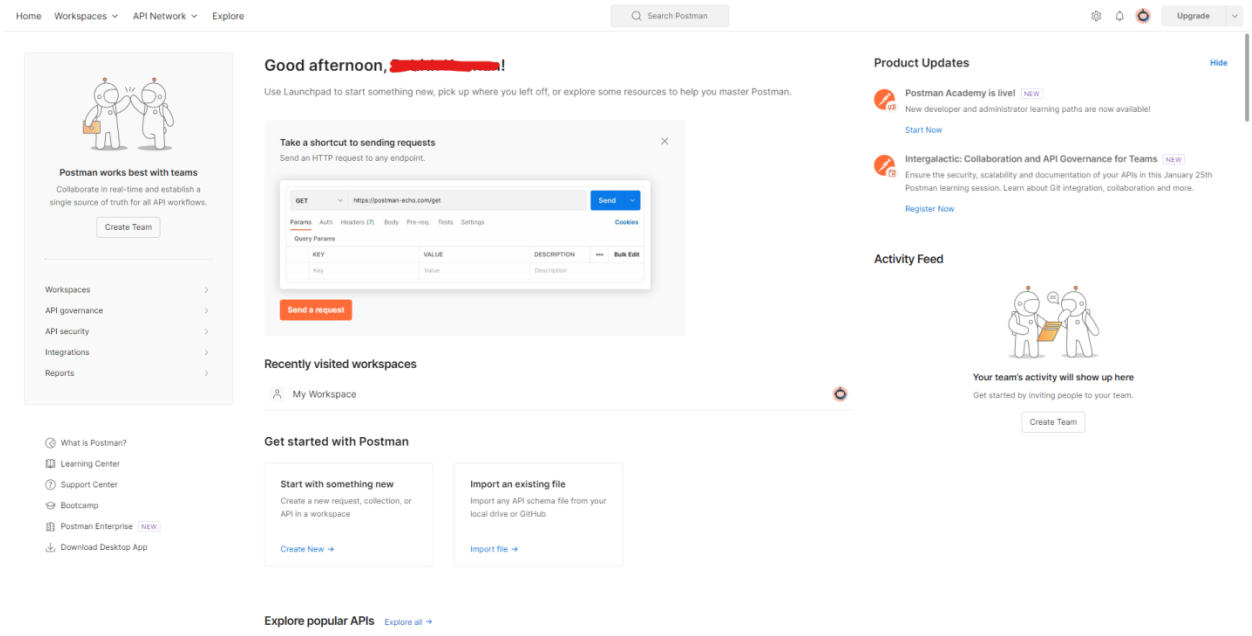


Go back to API Gateway

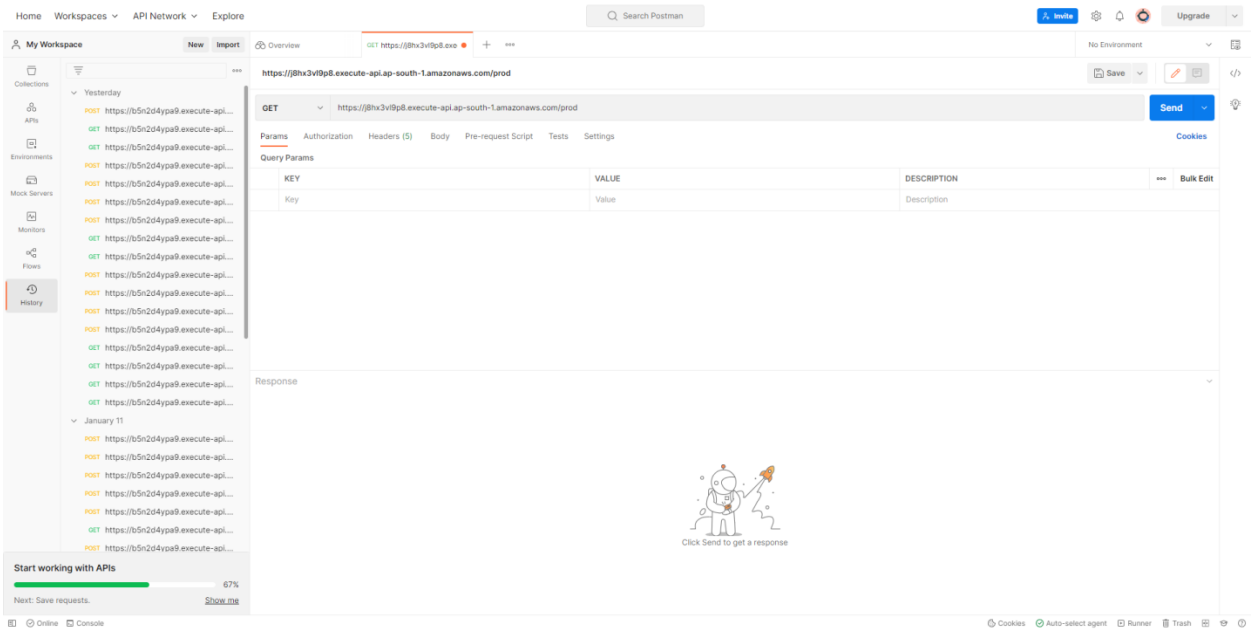
Copy the API invoke URL

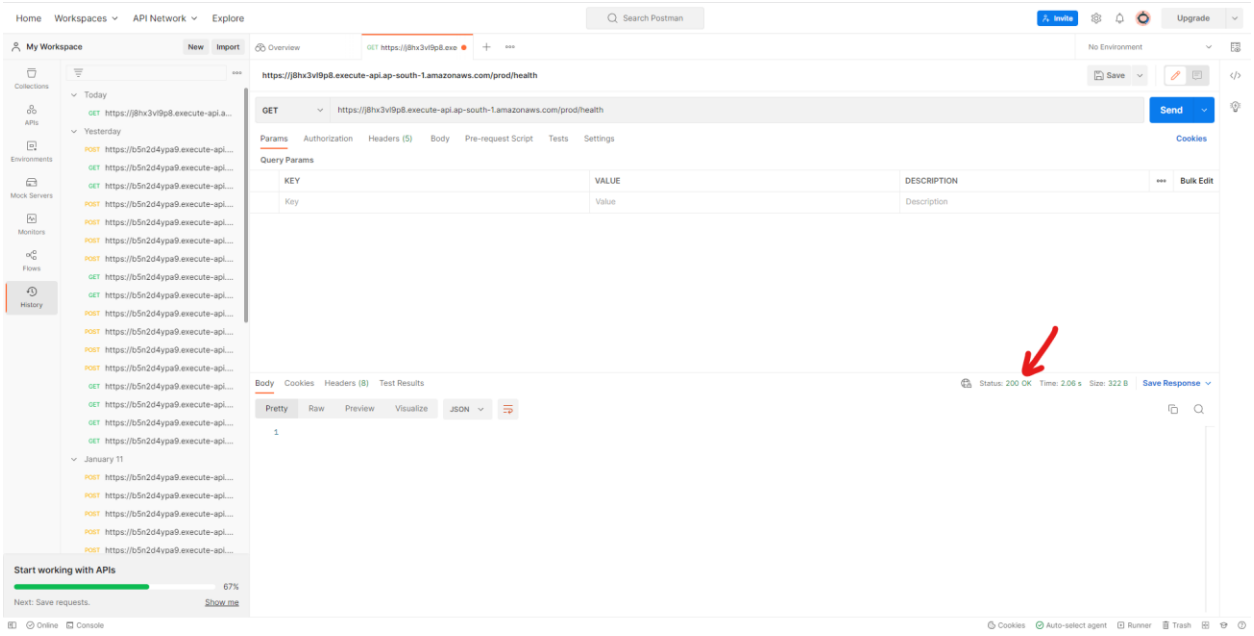


Open Postman and click on Send a request



Paste the invoke url





*Add "**/health**" to the end of the url and click Send*

Response received with Status 200 Ok

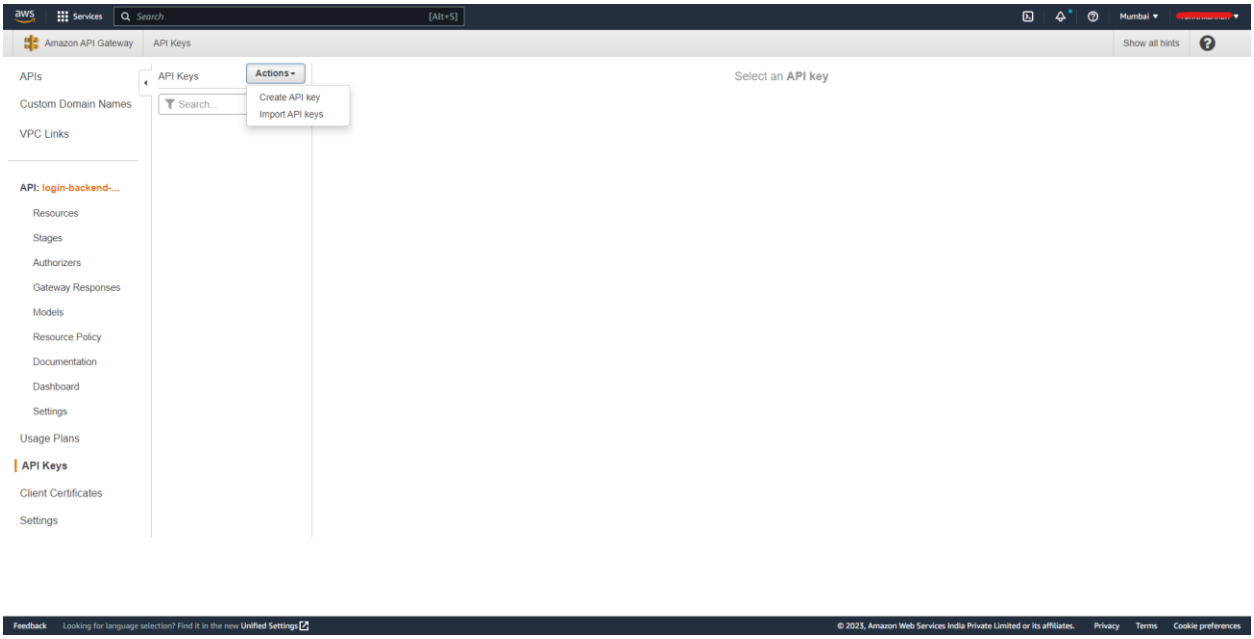
*Try the same for all endpoints "**/register**" "**/login**" "**/verify**"*

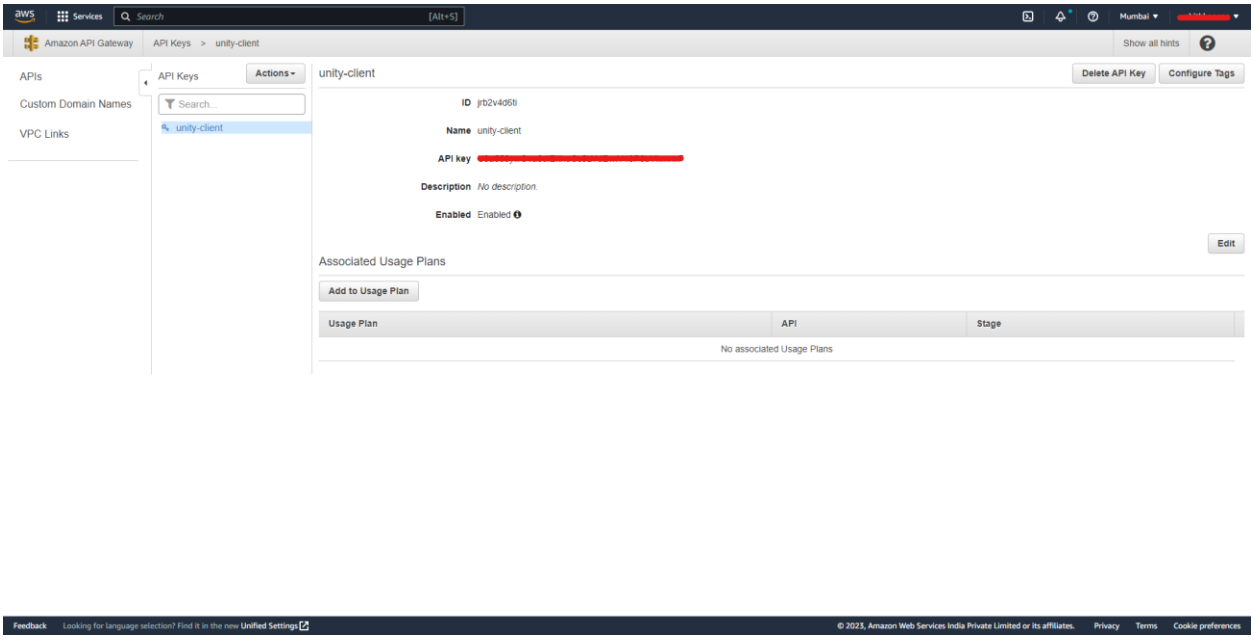
*with respective methods "**POST**"*

This API gateway allows requests by anyone with the url

Create an API Key to send authorized requests

*We'll send the API Key as one of the headers "**x-api-key**"*

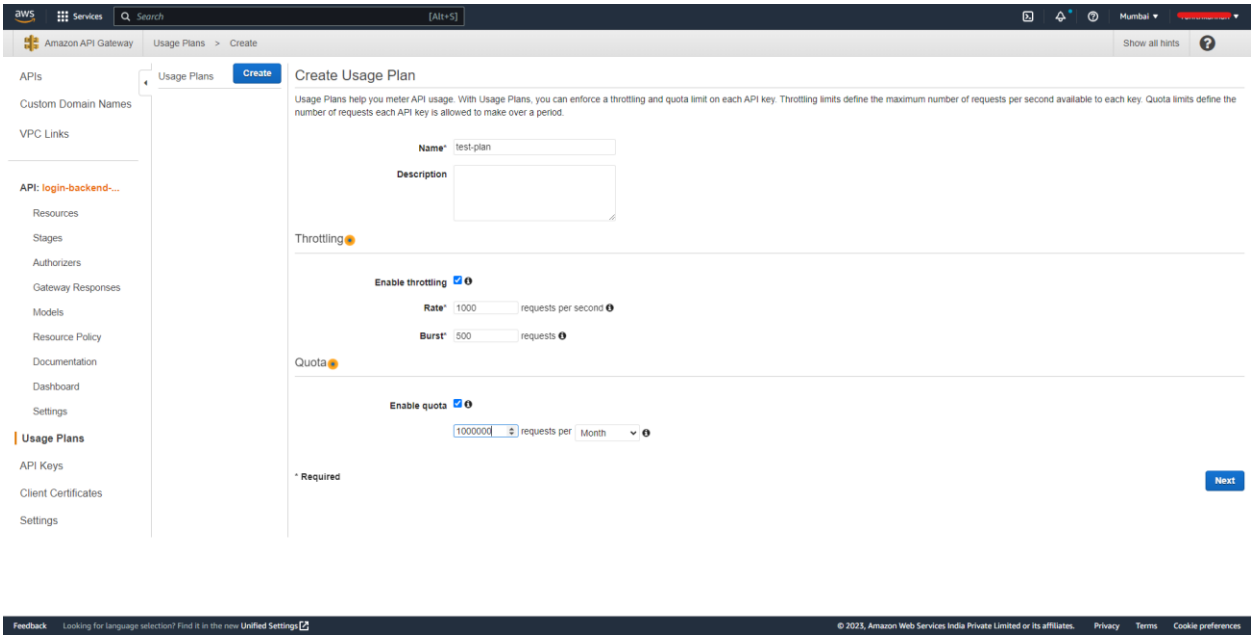




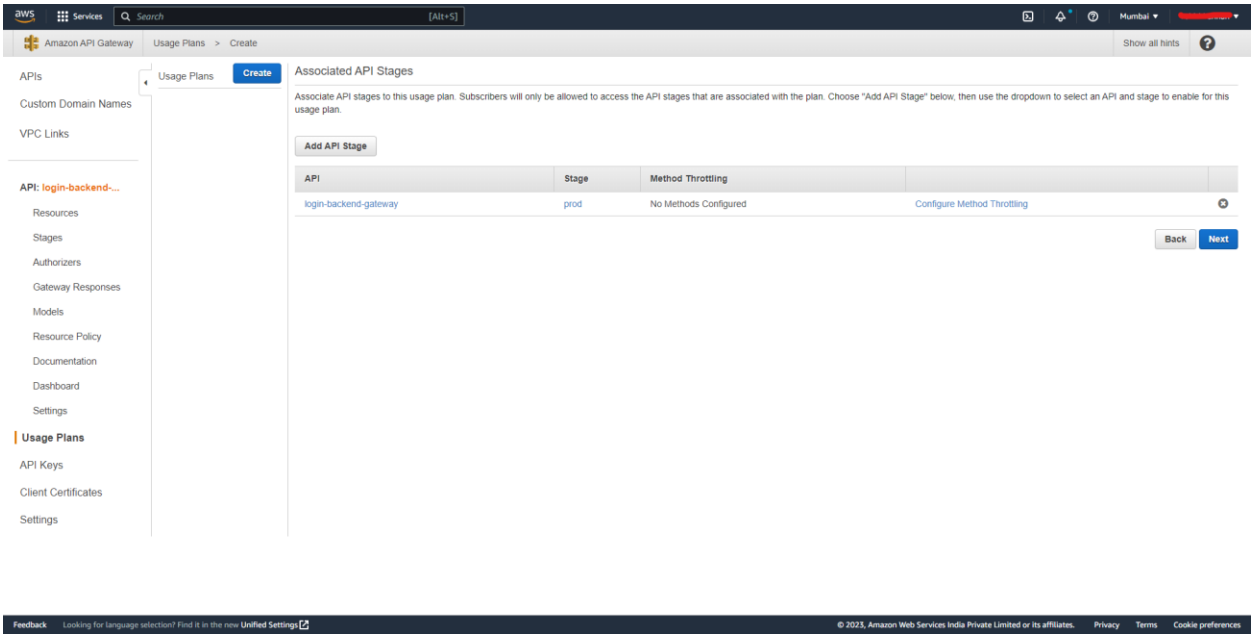
Enter Customer Name and Create API with Auto-Generate on

Click on show next to API key to view the key

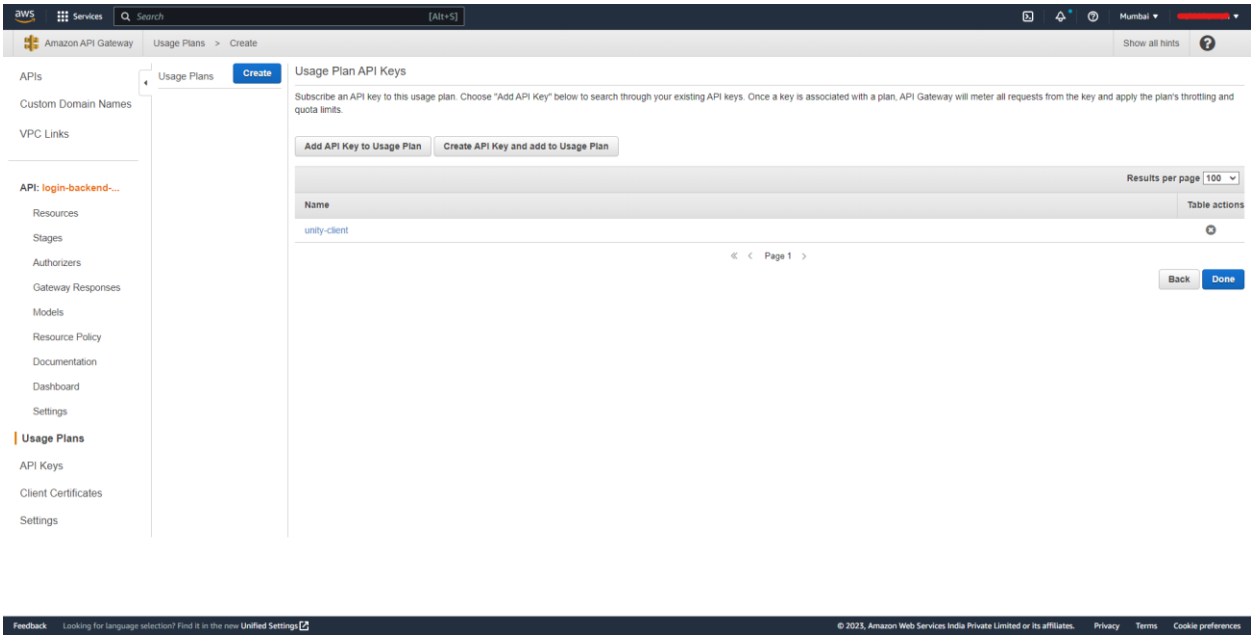
Go back to API Gateway > Usage Plans



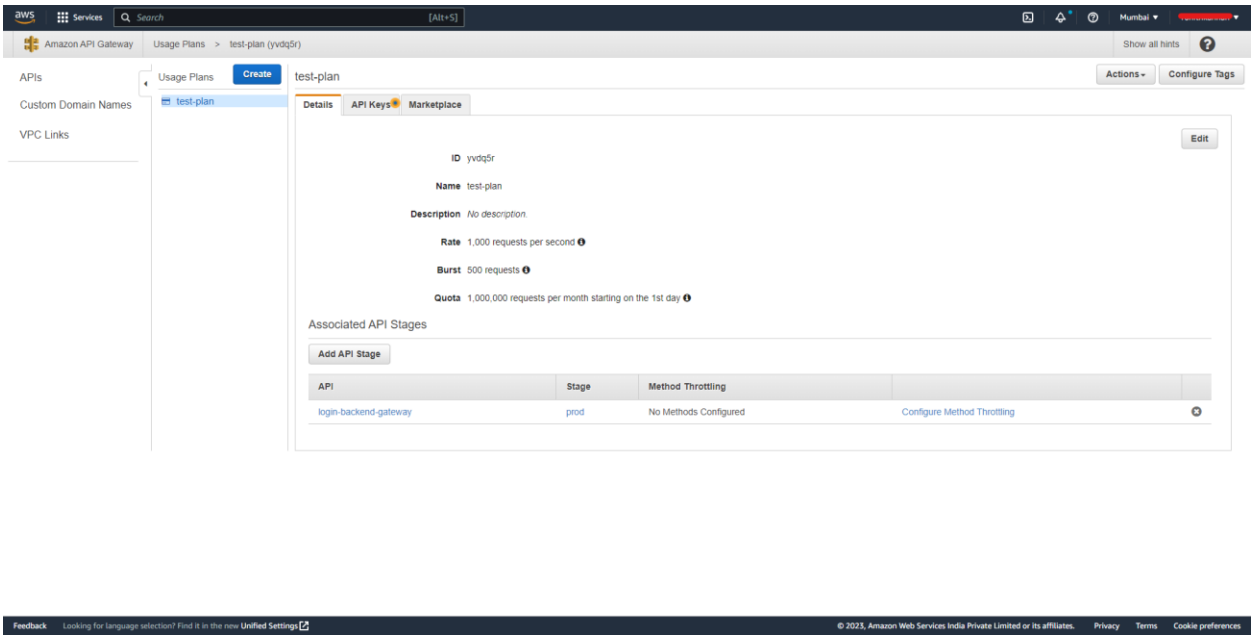
Add API Stage



Add API Key to Usage Plan

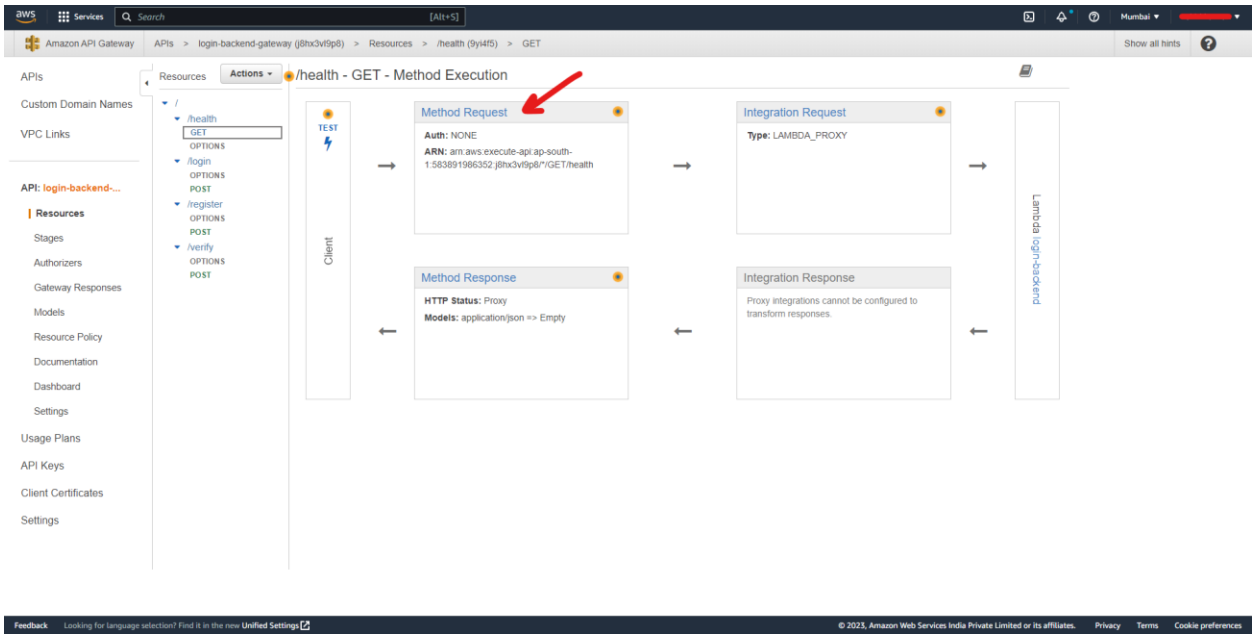


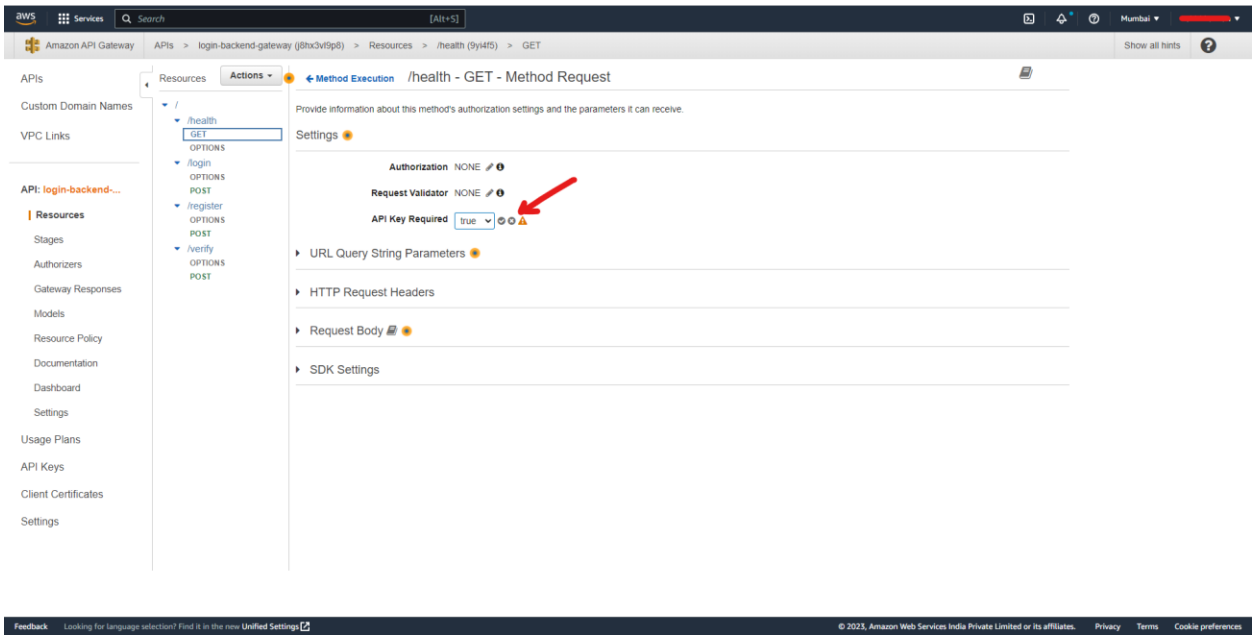
Usage Plan created



Go back to API Gateway

*And for GET method, set API Key required to **True***



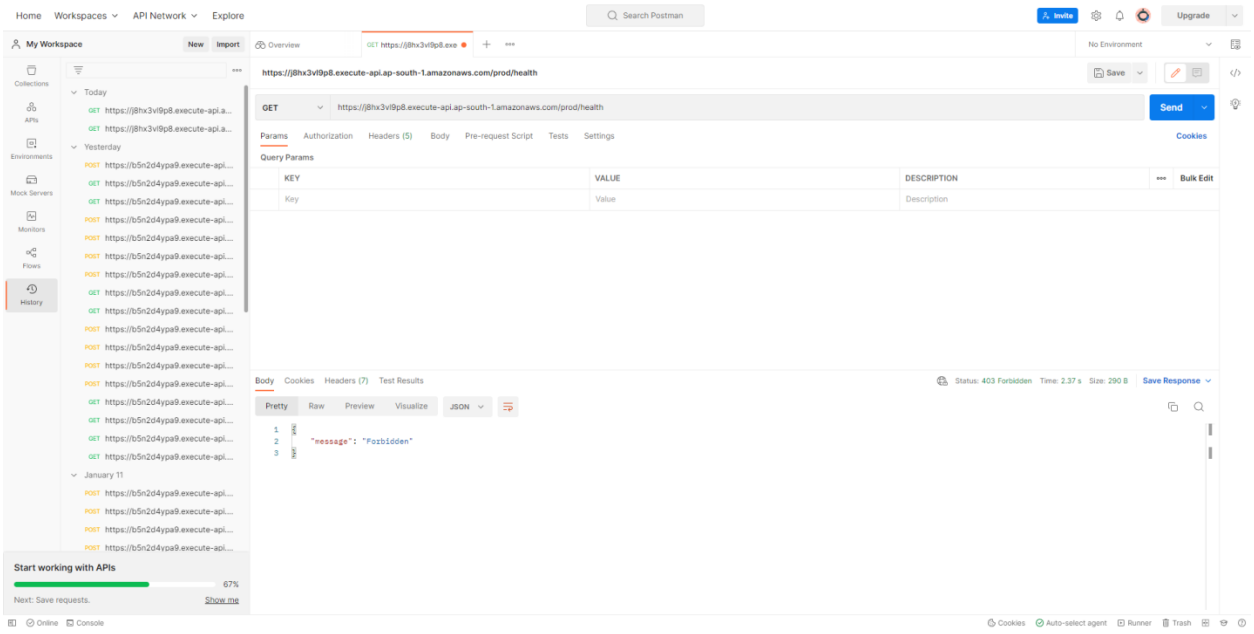


Repeat this for each all the POST methods

*Select “ **/** ” (root) and then Actions > Deploy API*

Go back to Postman

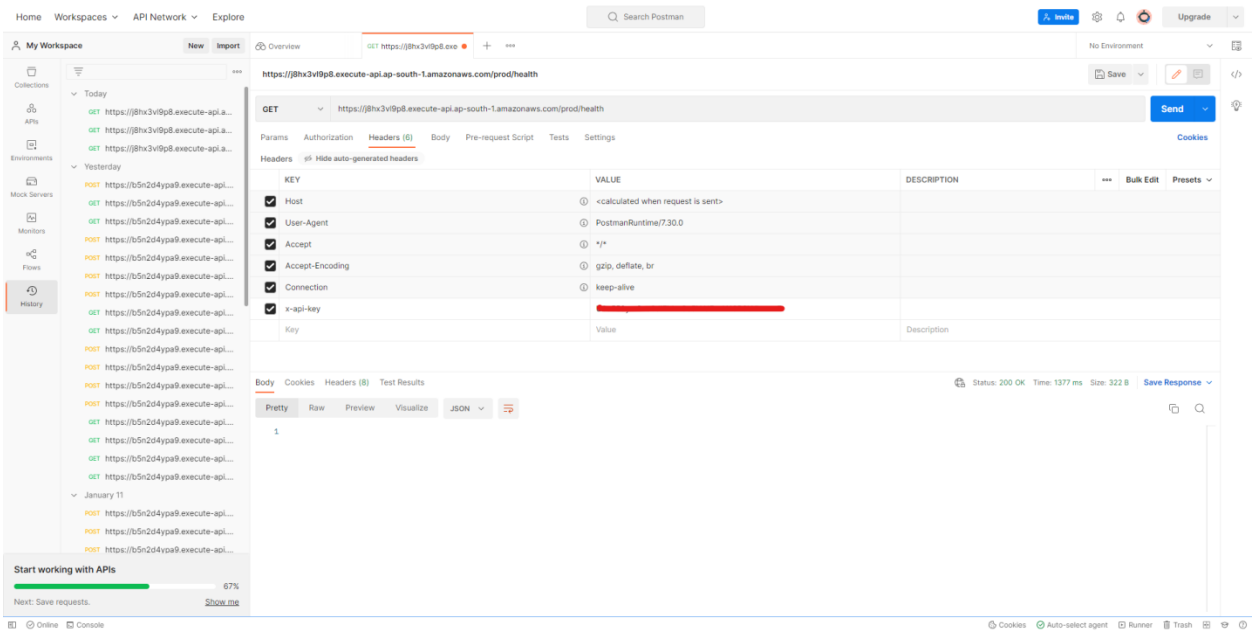
Sending POST and GET requests to the endpoints will return 403 Forbidden



Go to Headers

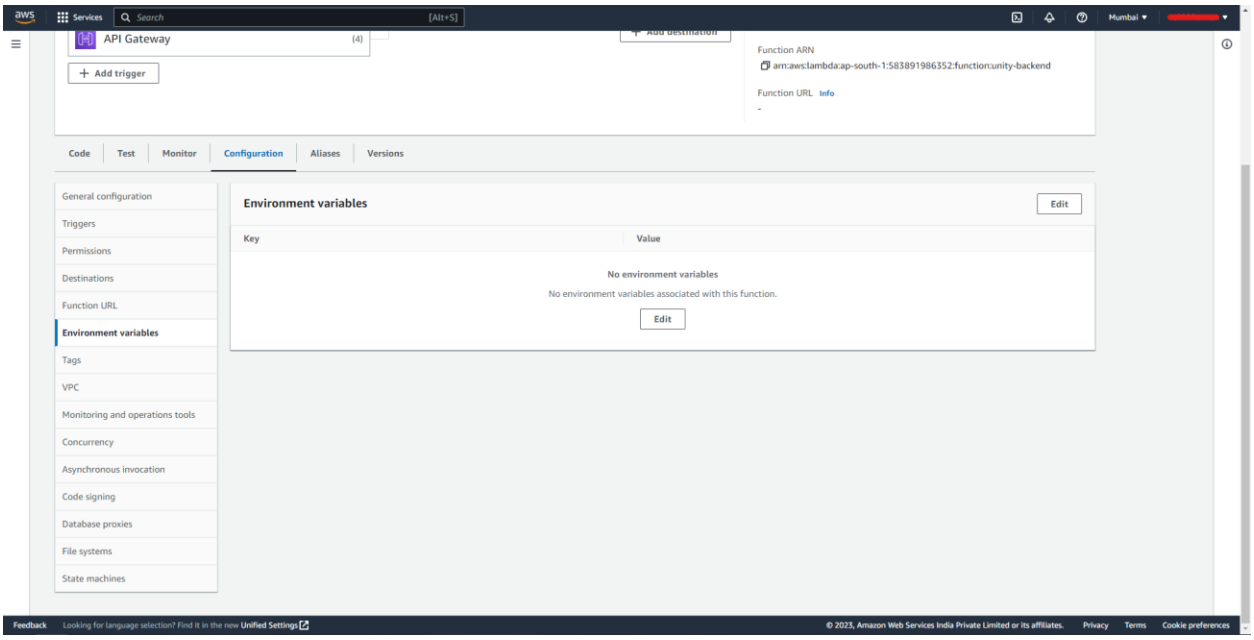
*Create a new header with Key "**x-api-key**" and set Value to the [API key]*

Send request to get response 200 OK



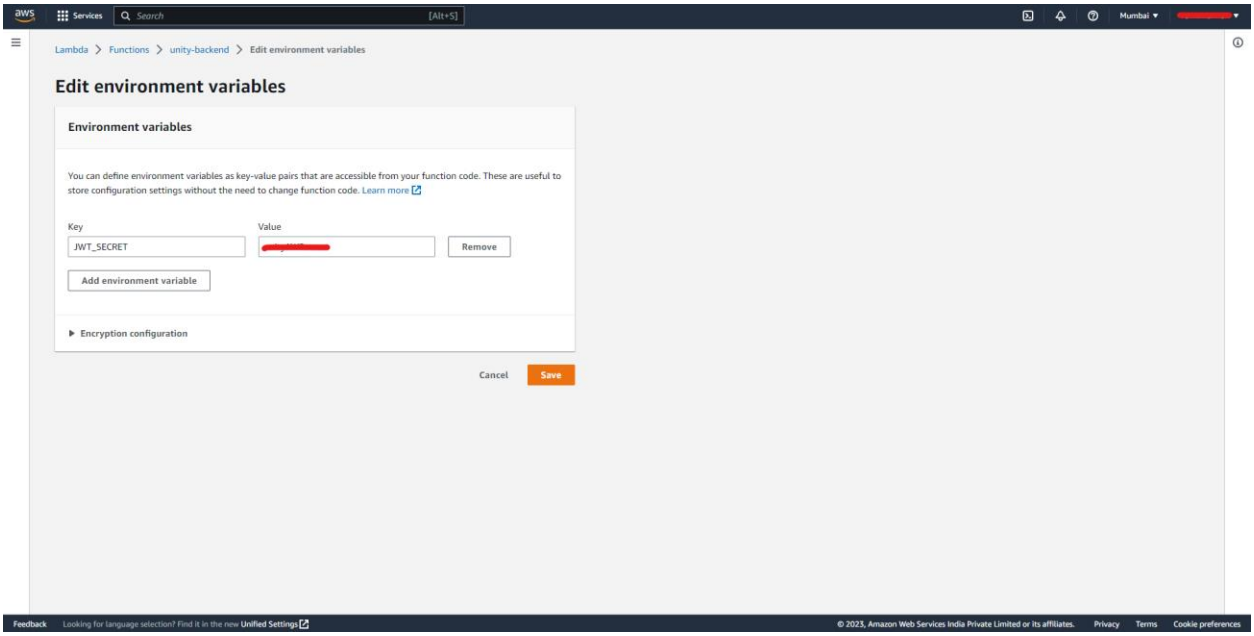
Next, we update the code for Lambda function.
Upload the Backend folder as a zip file to Lambda

Setting Environment Variable



Go to Lambda > Functions > current Lambda Function

Under Configuration > Environment Variables > Edit



Enter a new Environment Variable and its value and then Save.

Test the register, login and verify endpoints on postman by giving the appropriate http body

POST */register*

```
{
     “name” : “somename”,
     “email” : “email@gmail.com”,
     “username” : “someusername”,
     “password” : “somepassword”
}
```

POST */login*

```
{
     “username” : “someusername”,
     “password” : “somepassword”
}
```

POST */verify*

```
{
     “user” : {
          “username” : “someusername”,
          “name” : “somename”
     },
     “token” : “sometoken”
}
```