**Microservices without the Servers**

**What is Serverless Computing?**

Serverless computing allows you to build and run applications and services without thinking about servers. Serverless applications don't require you to provision, scale, and manage any servers. You can build them for virtually any type of application or backend service, and everything required to run and scale your application with high availability is handled for you.

Building Serverless applications means that your developers can focus on their core product instead of worrying about managing and operating servers or runtimes, either in the cloud or on-premises. This reduced overhead lets developers reclaim time and energy that can be spent on developing great products which scale and that are reliable.

**Why Serverless Computing is useful?**

1. **Decreased Time to Market:** Serverless approaches allow developers to create new apps in hours and days instead of weeks and months. Examples abound in new apps that rely on third-party APIs for services like: authentication (OAuth), social (Twitter), maps (Mapbox), artificial intelligence (IBM’s Watson), and more.
2. **Enhanced Scalability:** Everyone wants his app to be the next Facebook, but if that happens, can it handle the load? Provisioning infrastructure just in case is a big risk — but so is being unprepared when success strikes. Serverless architecture means you don’t need to make that choice. One online training program scaled to 40,000 users in six months without a single server.
3. **Lower Cost:** In terms of both computing power and human resources, serverless saves. Why pay to reinvent the wheel for authorization, presence detection, and image processing? Or to manage infrastructure? And if there’s no need for always-on servers, operational costs plummet. The days of spending hundreds of thousands of dollars for servers are gone.
4. **Improved Latency:** An app’s ability to scale depends on three things: its number of users, those users’ locations, and network latency. Today’s apps have global audiences, which can create latencies that diminish experiences. With serverless, providers have points of presence near every user, and apps perform equally well for everyone.

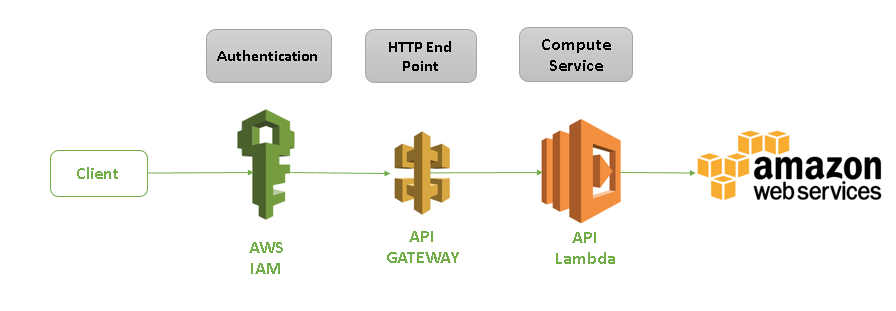
**How to implement Serverless?**

Let’s create a simple Microservice without spinning any server. The following architecture can be scaled up and down based on load automatically. Here we have used three main AWS services:

IAM: https://aws.amazon.com/iam/

API Gateway: https://aws.amazon.com/api-gateway/

Lambda: https://aws.amazon.com/lambda/

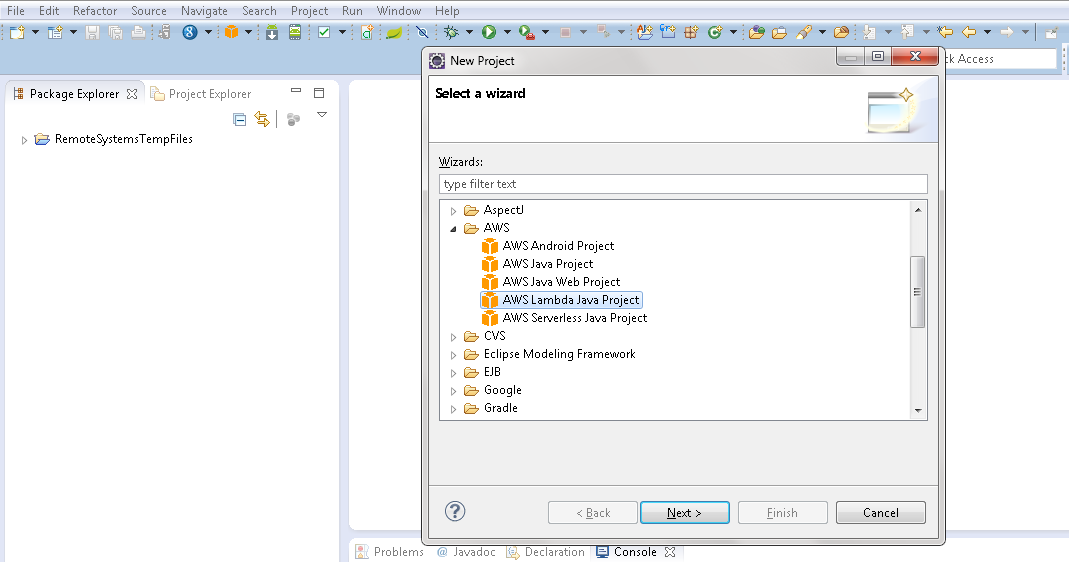


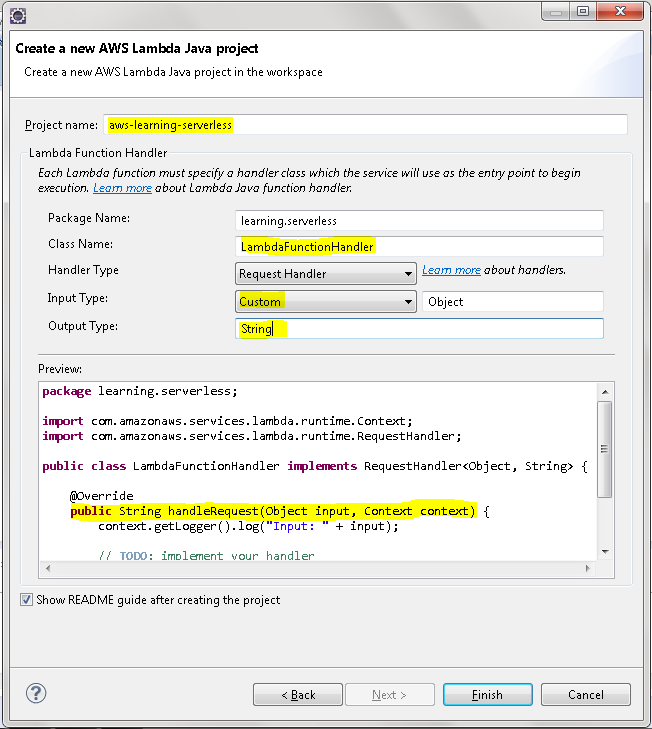
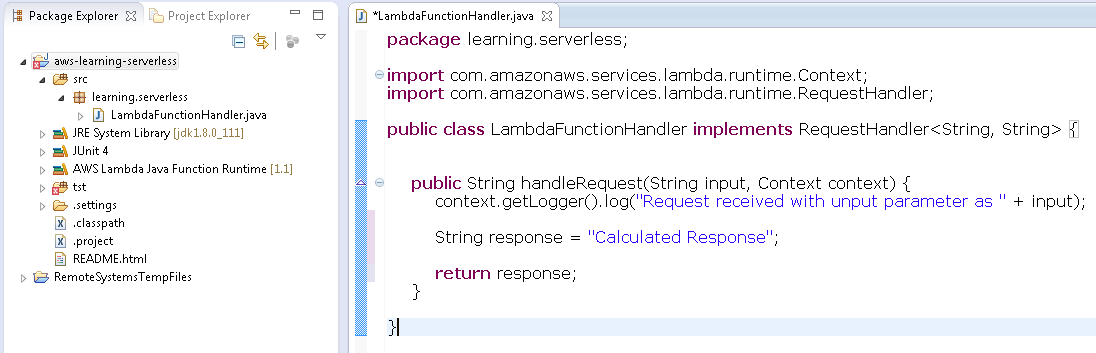
Below are the steps to configure simple Serverless Microservice.

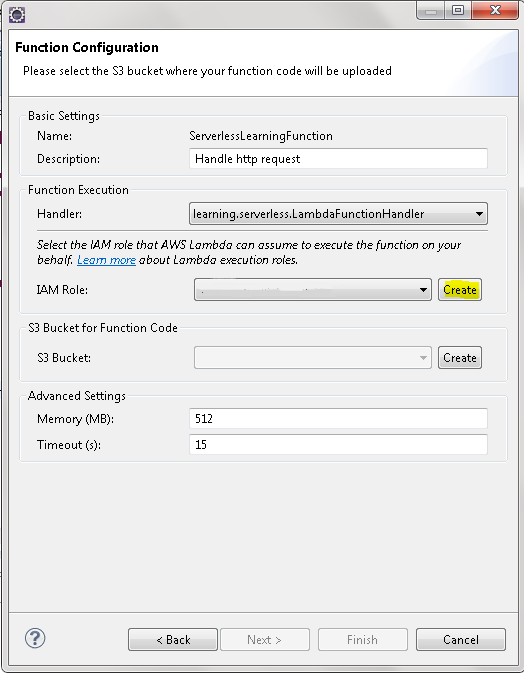
1. Install eclipse. Follow below steps to setup AWS plugin in eclipse.

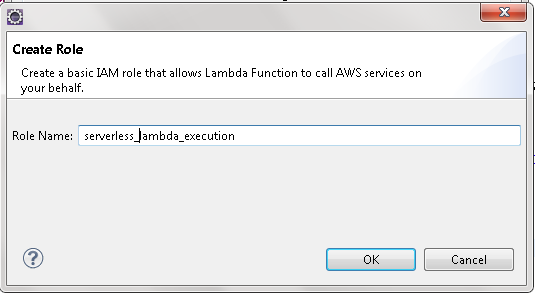
<http://docs.aws.amazon.com/toolkit-for-eclipse/v1/user-guide/setup-install.html>

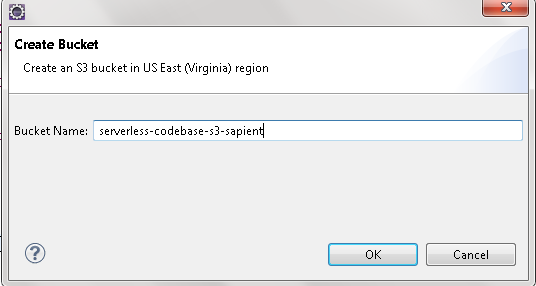
1. Create a new AWS Lambda project. To create new project, select File > New Project and select below option.

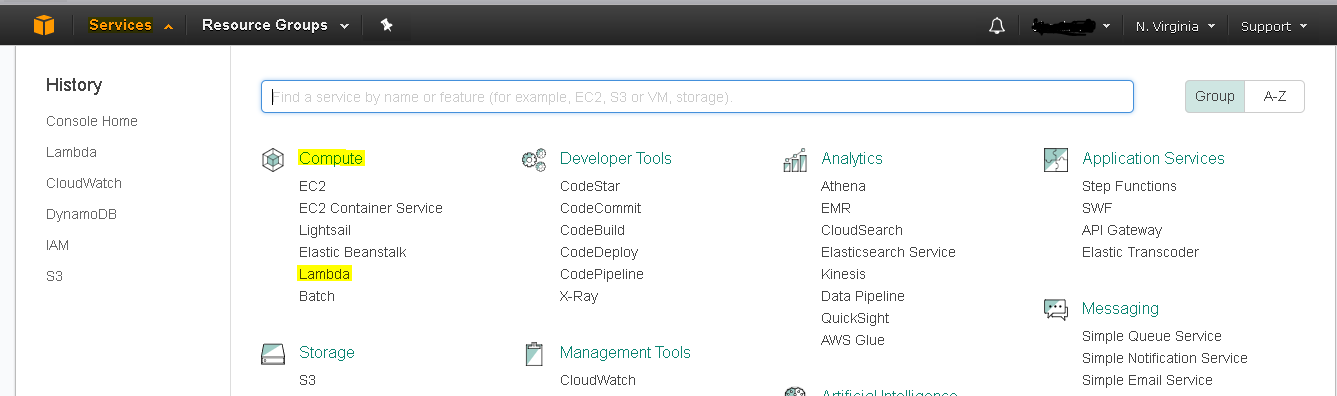


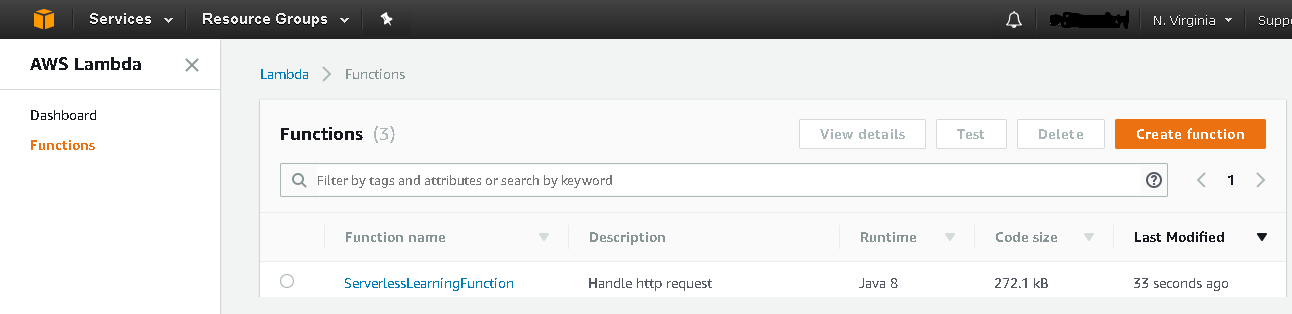
1. Provide project name and Lambda Handler details as below. Whenever lambda is triggered, this “handleRequest” method is called:
2. In the handler method, execution logic can be defined. For our example, we are returning simple string as a response of this request.
3. Now we can deploy our Lambda function on cloud server. To deploy, select project and right click. In the context menu, select **Amazon Web Service > Upload function to AWS Lambda**



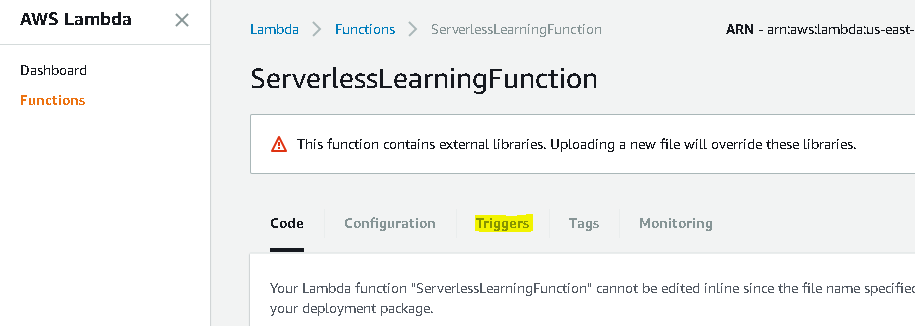
1. Create new role to execute lambda function as well as new bucket to store codebase into cloud.

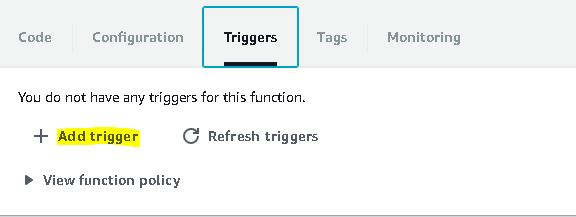


1. We have completed Lambda project. Now, we will configure its trigger. To configure, first login into the AWS console. Then select **Services > Lambda**

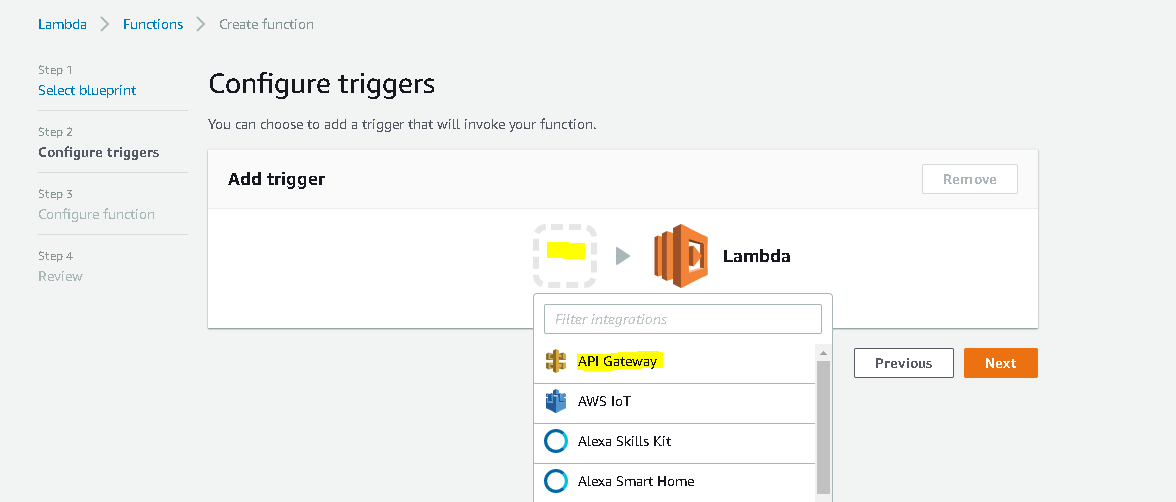


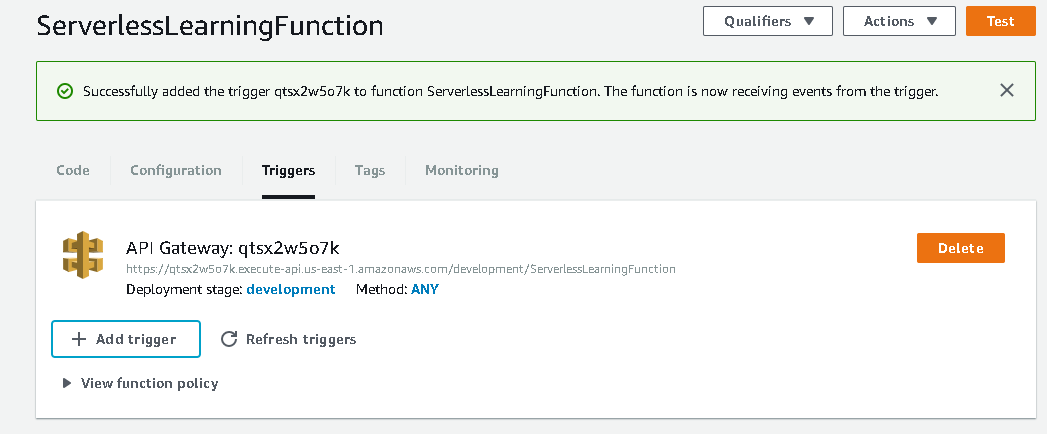
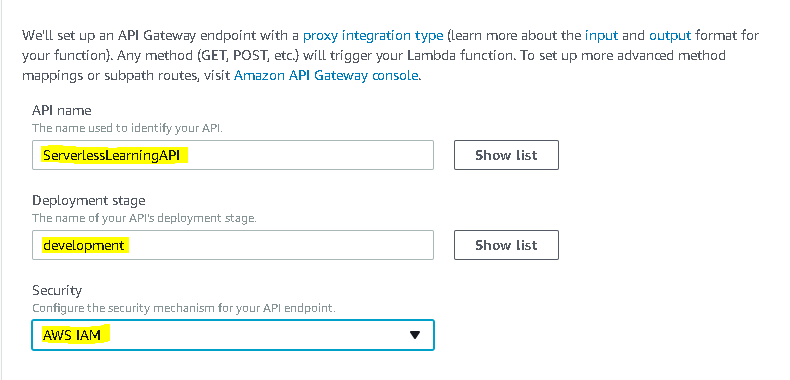
1. Now, select Function and Select Triggers. In the Trigger page, select “Add trigger” option.



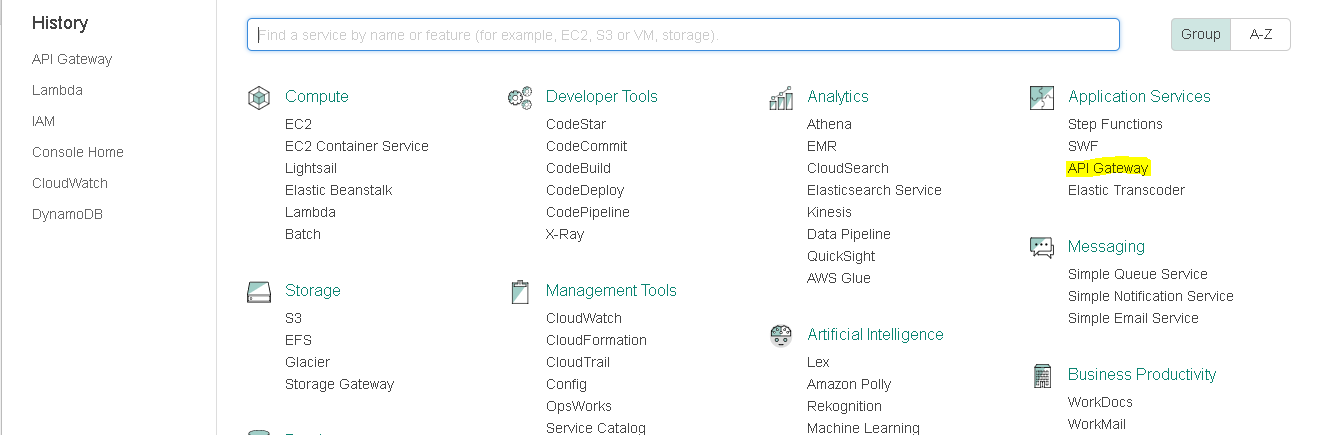


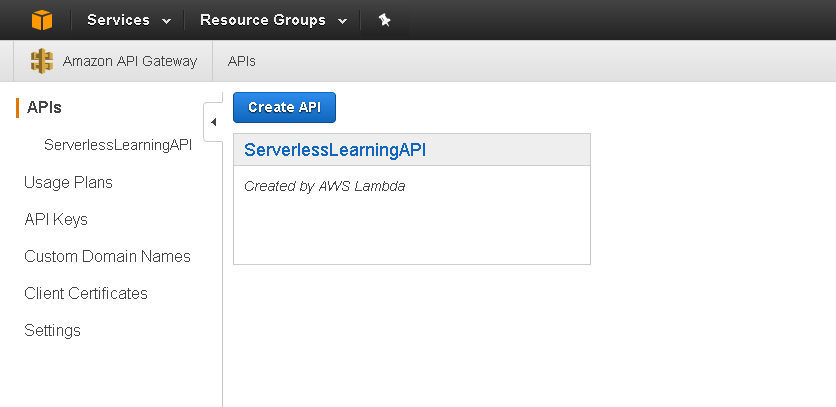
1. To configure API Gateway, we need to select **API Gateway** as trigger and provide API and deployment stage information:



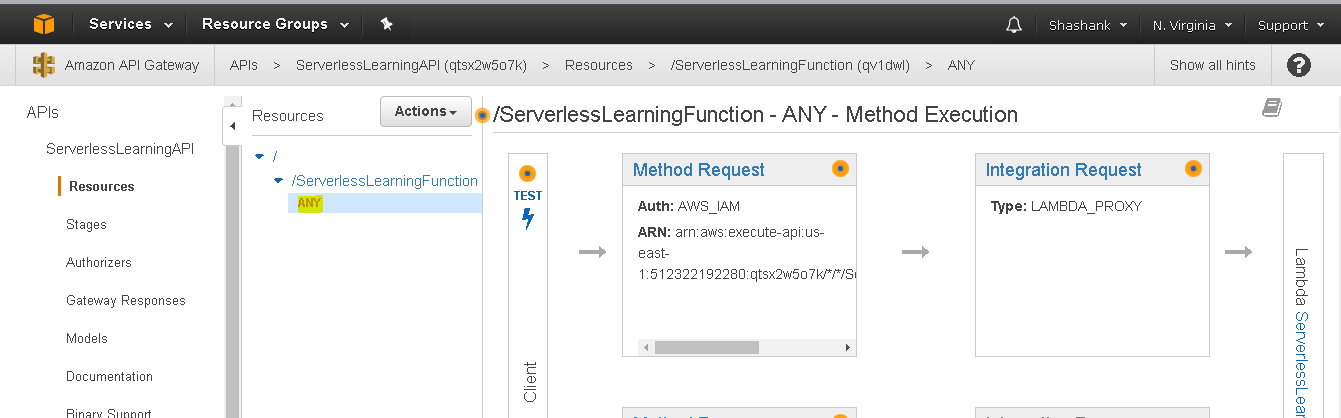


1. Now, submit the request. Till now, we have created a Lambda function to process our request and API gateway which will redirect HTTP request to Lambda function. But, currently, API Gateway will direct all HTTP methods to Lambda. But if we want to direct only GET request. We need to modify API Gateway. To open API Gateway, select **Services > API Gateway**

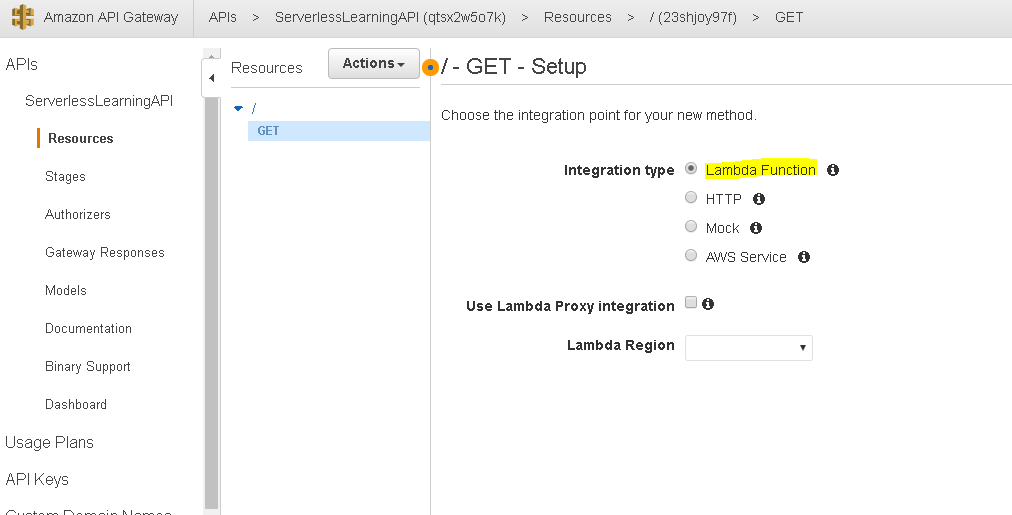
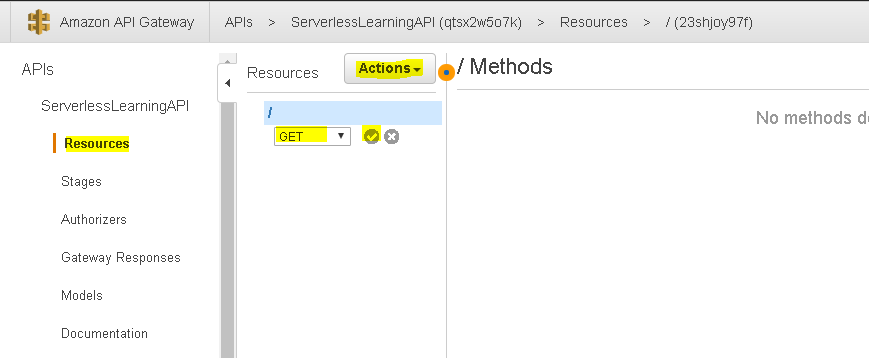


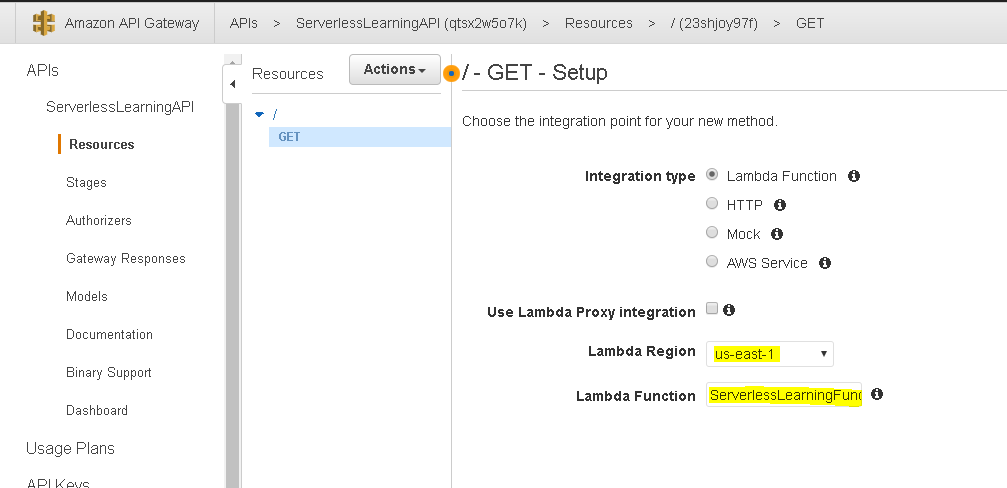


1. Select our API and then select method ANY.

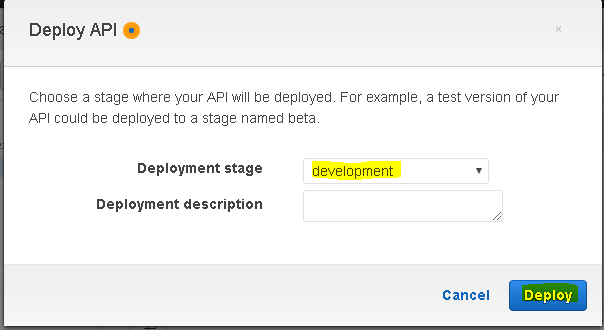


1. Now select **Actions > Delete Method** and to create GET method select **Actions > Create Method**

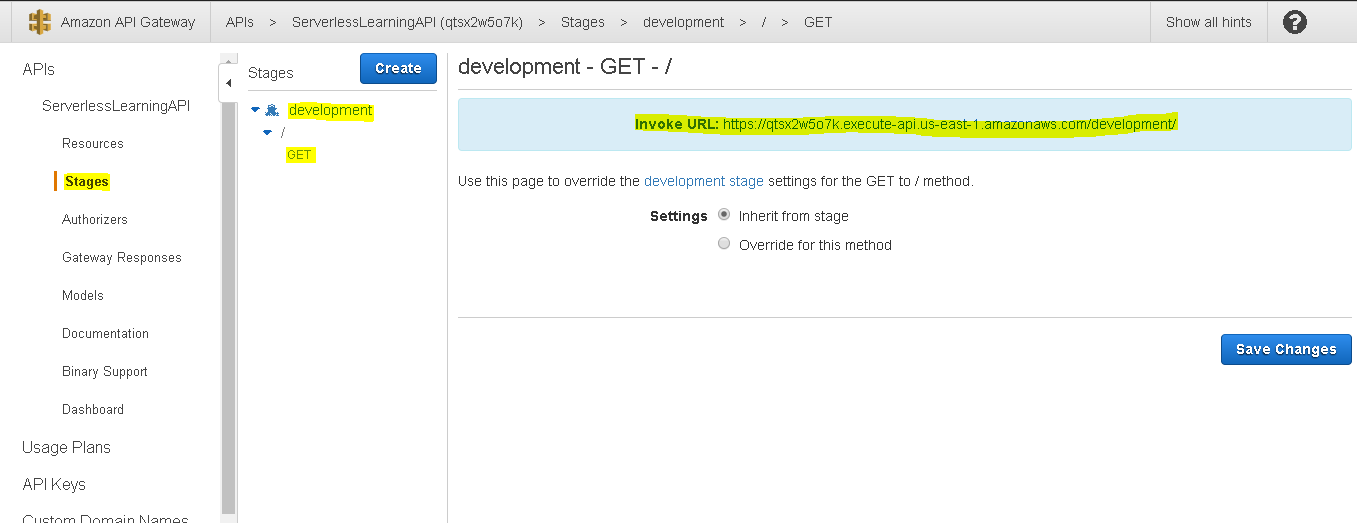




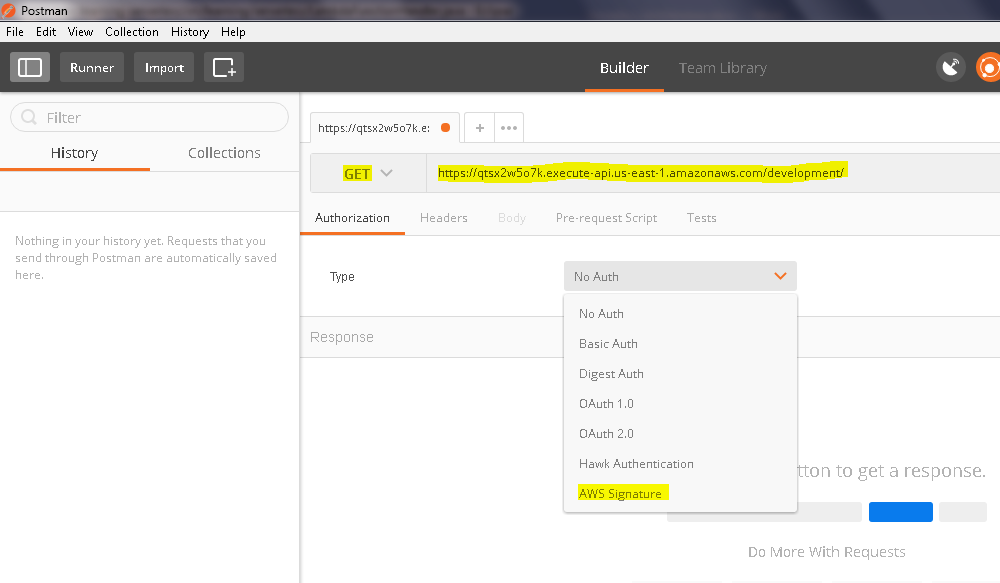
1. Allow access to role and now, to deploy application, select **Action > Deploy API**

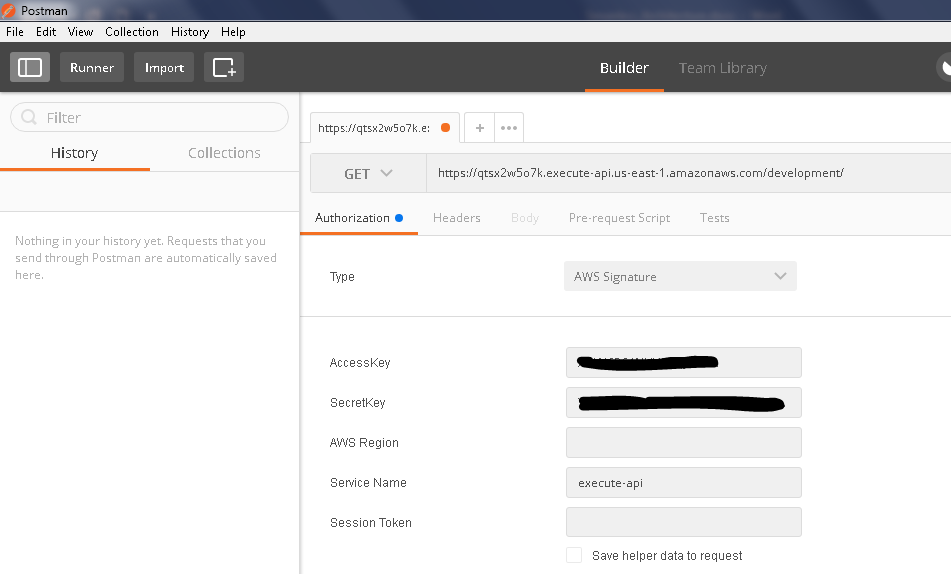


1. You can copy Invoke URL from **Stages**:



1. Now we can test our API using **Postman**. To test our API, we need to provide Invoke URL with GET as a method. choose the Authorization tab.Choose AWS Signature for the authorization Type. Enter your AWS IAM user's access key ID in the AccessKey input field. Enter your IAM user secret key in SecretKey. Specify an appropriate AWS region that matches the region specified in the invocation URL. Enter execute-api in Service Name.





1. Click on **Send** button. Following is the same response which we are returning from Lambda function: