1. Why use NodeJS to develop an API ?  
     
   Considering that Node.js can be run almost out-of-the-box on [Amazon's EC2](http://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud) instances.

Node.js

* is a command-line tool that can be run as a regular web server and lets one run JavaScript programs
* utilizes the great [V8 JavaScript engine](http://en.wikipedia.org/wiki/V8_%28JavaScript_engine%29)
* is very good when you need to do several things at the same time
* is event-based so all the wonderful [Ajax](http://en.wikipedia.org/wiki/Ajax_%28programming%29)-like stuff can be done on the server side
* let’s us share code between the browser and the backend
* let’s us work with noSQL

Node.js is especially suited for applications where like to maintain a persistent connection from the browser back to the server. Using a technique known as ["long-polling"](http://en.wikipedia.org/wiki/Push_technology#Long_polling), we can write an application that sends updates to the user in real time. Doing long polling on many of the web's giants, like java, would create immense load on the server, because each active client eats up one server process. When we use something like Node.js, the server has no need of maintaining separate threads for each open connection.

This means you we create a backend microservices in Node.js that takes almost no system resources to serve a great many clients. Any time we want to do this sort of long-polling, Node.js is the great option.

JavaScript is exceptionally well situated to a callback-based concurrency model, and it excels. Also, being able to serialize and deserialize with JSON native to both the client and the server

It's worth pointing out that Node.js is also great for situations in which you'll be reusing a lot of code across the client/server gap.

So we can build a REST API in Node.js with AWS Lambda, API Gateway, DynamoDB, and Serverless Framework

Serverless means different things depending on the context. It mean an event-driven architecture style.  
AWS Lambda is the third compute service from Amazon. It's very different from the existing two compute services EC2 (Elastic Compute Cloud) and ECS (Elastic Container Service). AWS Lambda is an event-driven, Serverless computing platform that executes the code in response to events. It manages the underlying infrastructure scaling it up or down to meet the event rate. We're only charged for the time that the code is executed. AWS Lambda currently supports Java, Python, and Node.js language runtimes.  
The Serverless Framework makes it easy to build applications using AWS Lambda. It is multi-provider framework, which means you can use it to build Serverless applications using other providers as well. For AWS, Serverless relies on CloudFormation to do the provisioning. It also scaffolds the project structure and takes care of deploying functions.

Once we are able to make HTTP requests to the API we need that data can be saved to DynamoDB.  
  
We could use MongoDB instead DynamoDB not because it has more features, it doesn't. Mongo has a better query language, we can index within a structure, there's lots of little things. The superiority of Dynamo is because it is easy we don't have to take care of any servers. When we start to set up a Mongo sharded solution, it gets complicated. We can go to one of the hosting companies, but that's not cheap either. With Dynamo, if we need more throughput, we just click a button. We can write scripts to scale automatically. When it's time to upgrade Dynamo, it's done for us. That is all a lot of precious stress and time not spent. If don't have dedicated ops people, Dynamo is excellent.

So we are now going on Dynamo by default. Mongo maybe, if the data structure is complicated enough to warrant it, but then we'd probably go back to a SQL database. Dynamo is obtuse, we really need to think about how you're going to build it, and likely we use Redis in ElastCcache to make it work for complex stuff. But it sure is nice to not have to take care of it.

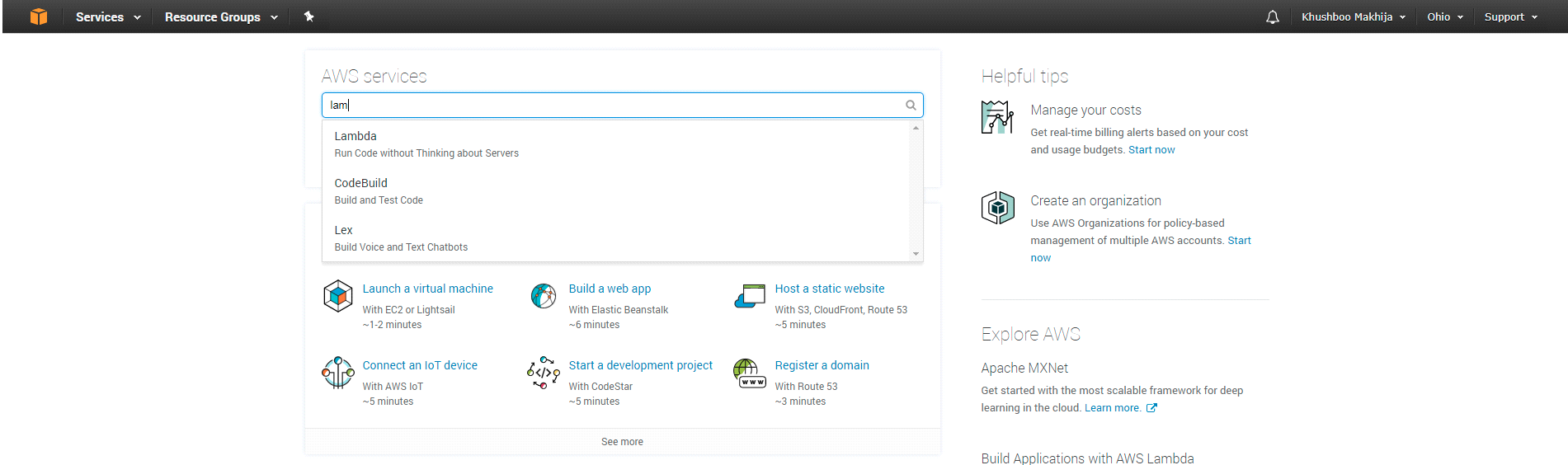
With AWS API Gateway, we can setup a simple http proxy to your application. Based on our needs, we can use features like auth, throttling, API keys management, client certificates, and use of wrapping the Serverless lambda microservices.

For managed and pulling software we can use of the The AWS Command Line Interface (CLI) that is a unified tool to manage the AWS services. With just one tool to download and configure, we can control multiple AWS services from the command line and automate them through scripts.

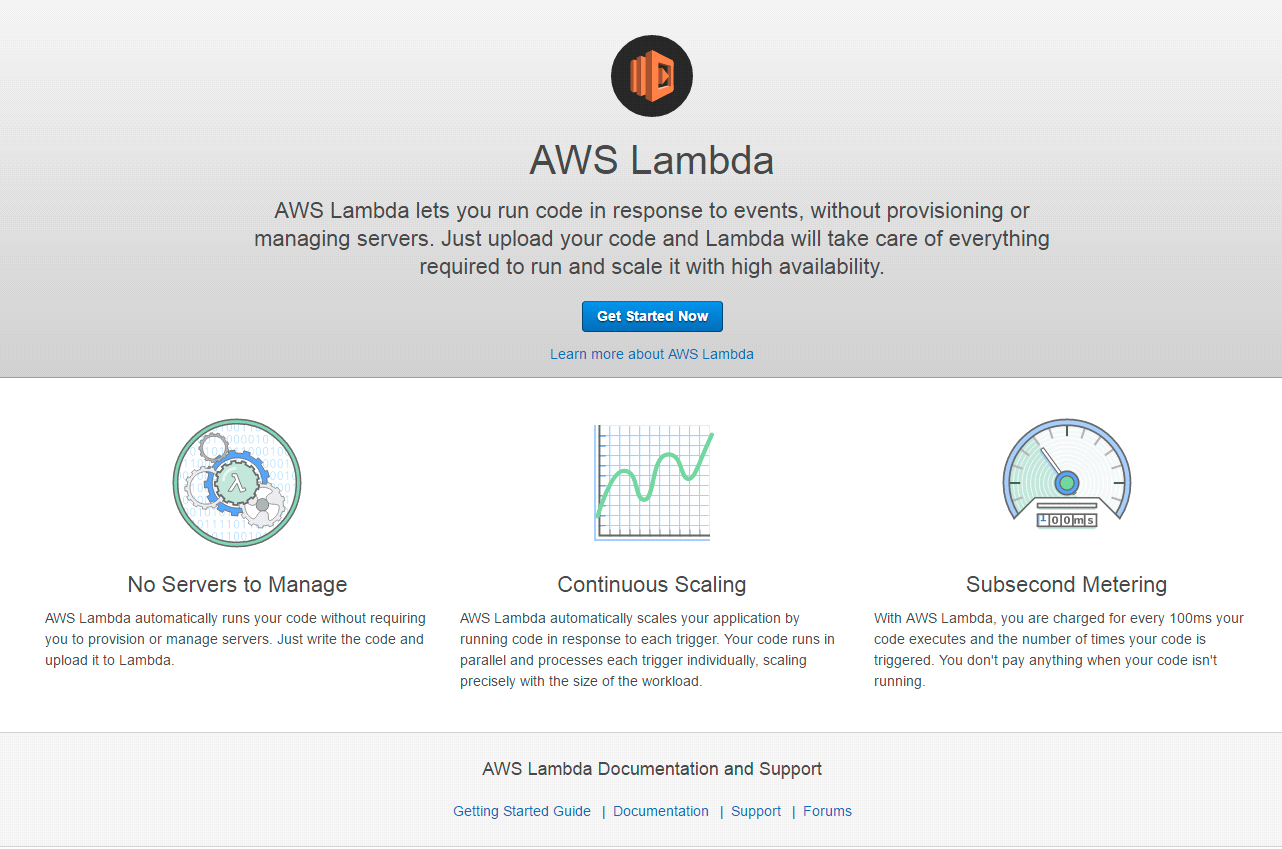
Anyway we always can use the AWS Web console <https://aws.amazon.com/console> to perform the operations mentioned

**Steps for Creating AWS Lambda Function:**

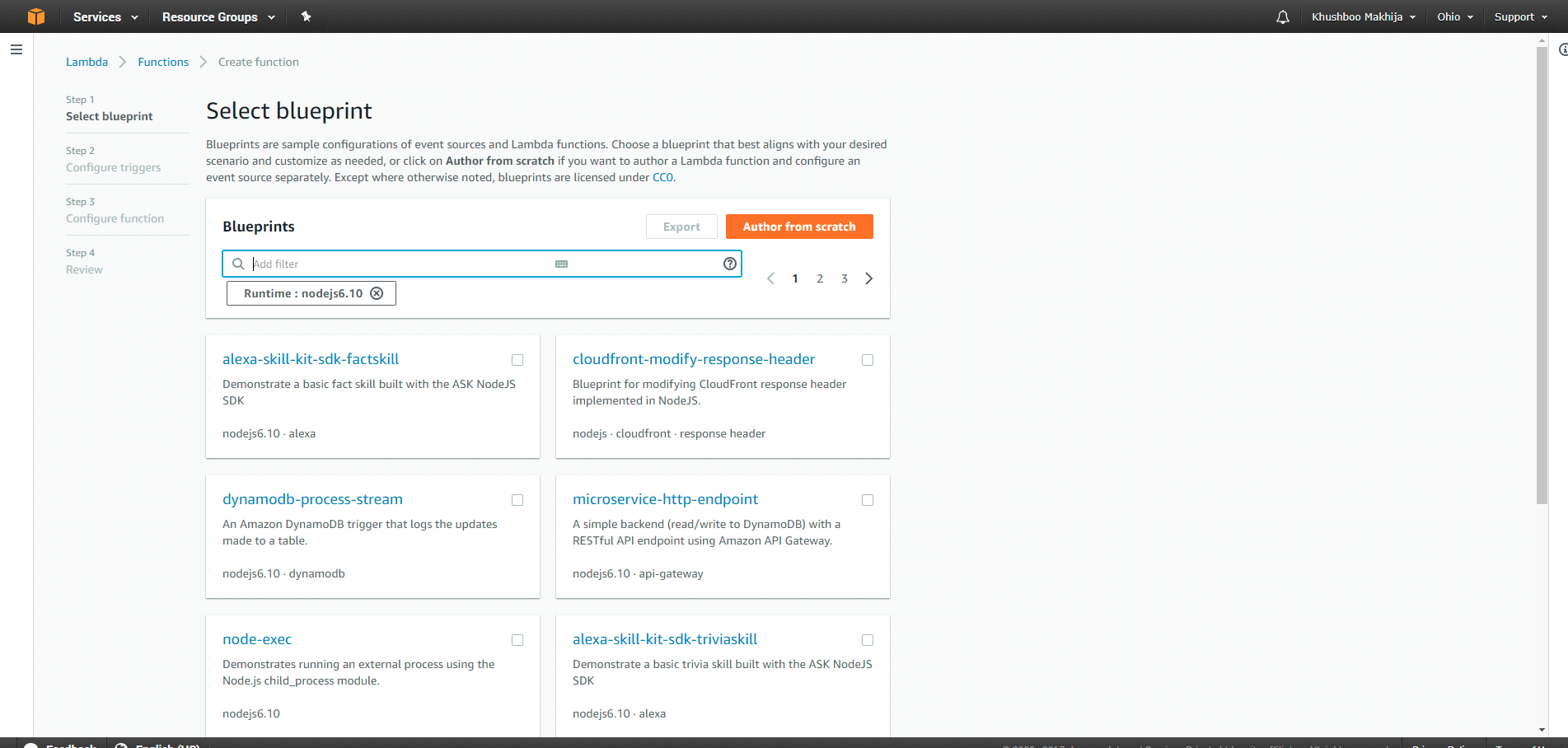
1. After Sign in, go in Services tab and search for Lambda and then select Lambda.



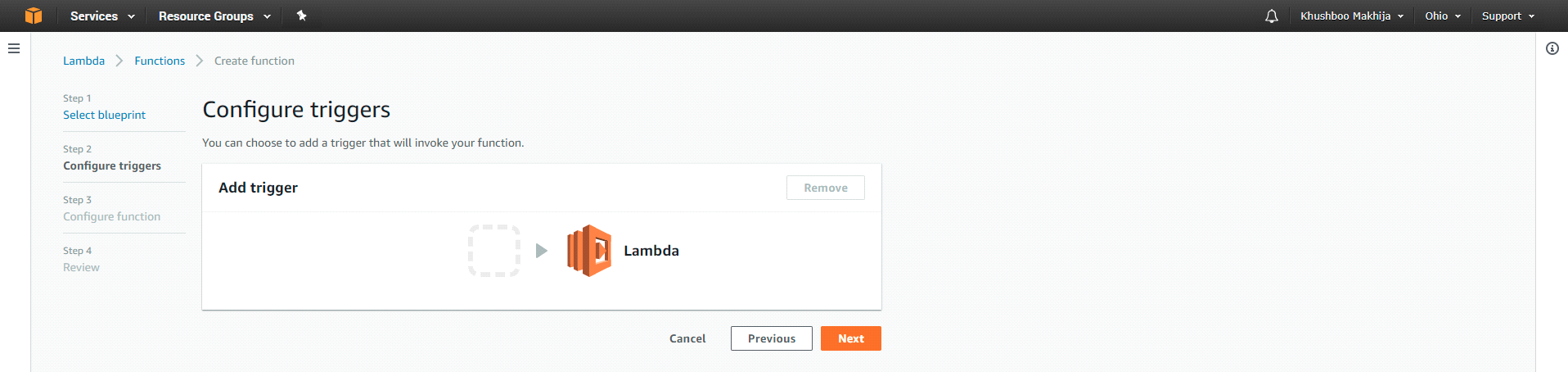
1. Click on Get started Now.



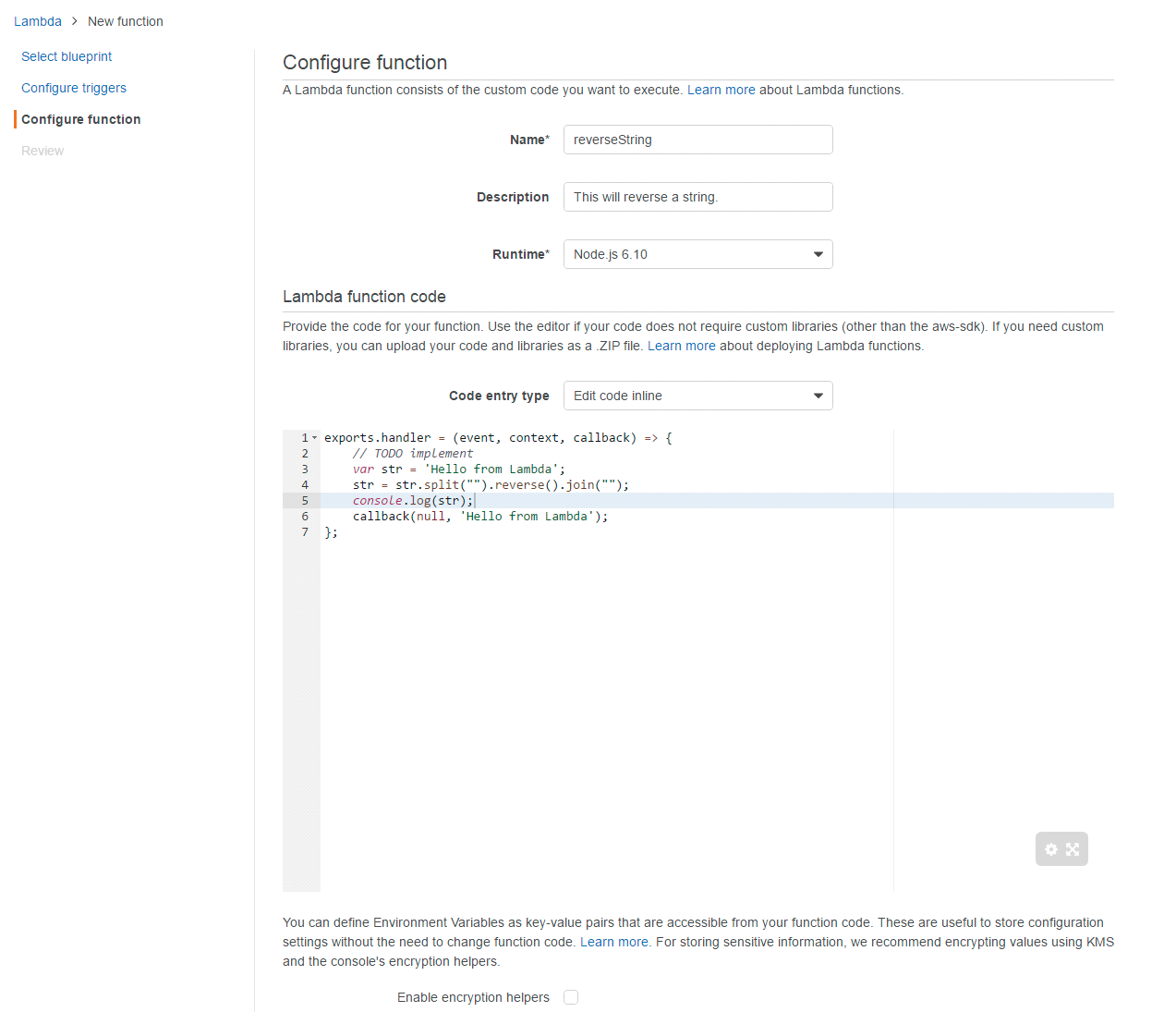
1. In Select blueprint page, select runtime from dropdown and select your blueprint accordingly. Now We’re selecting Node.js 6.10 and Blank Function as the blueprint.



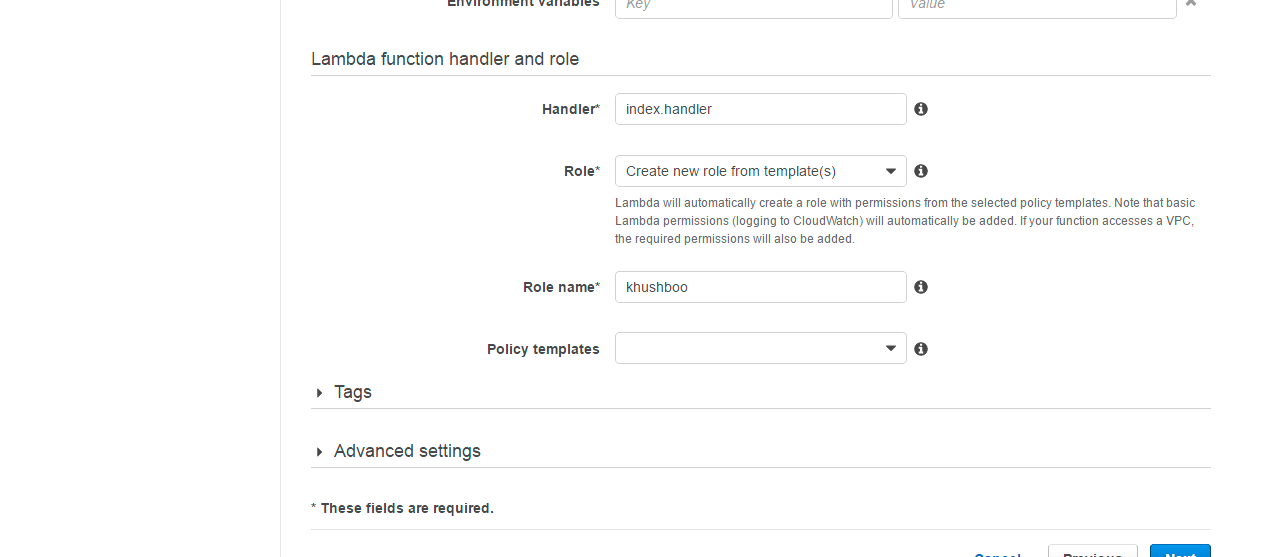
1. After that, we need to select how you will call Lambda function by clicking on blank square box or we can skip this by clicking on next. We’re skipping this step by clicking on next.



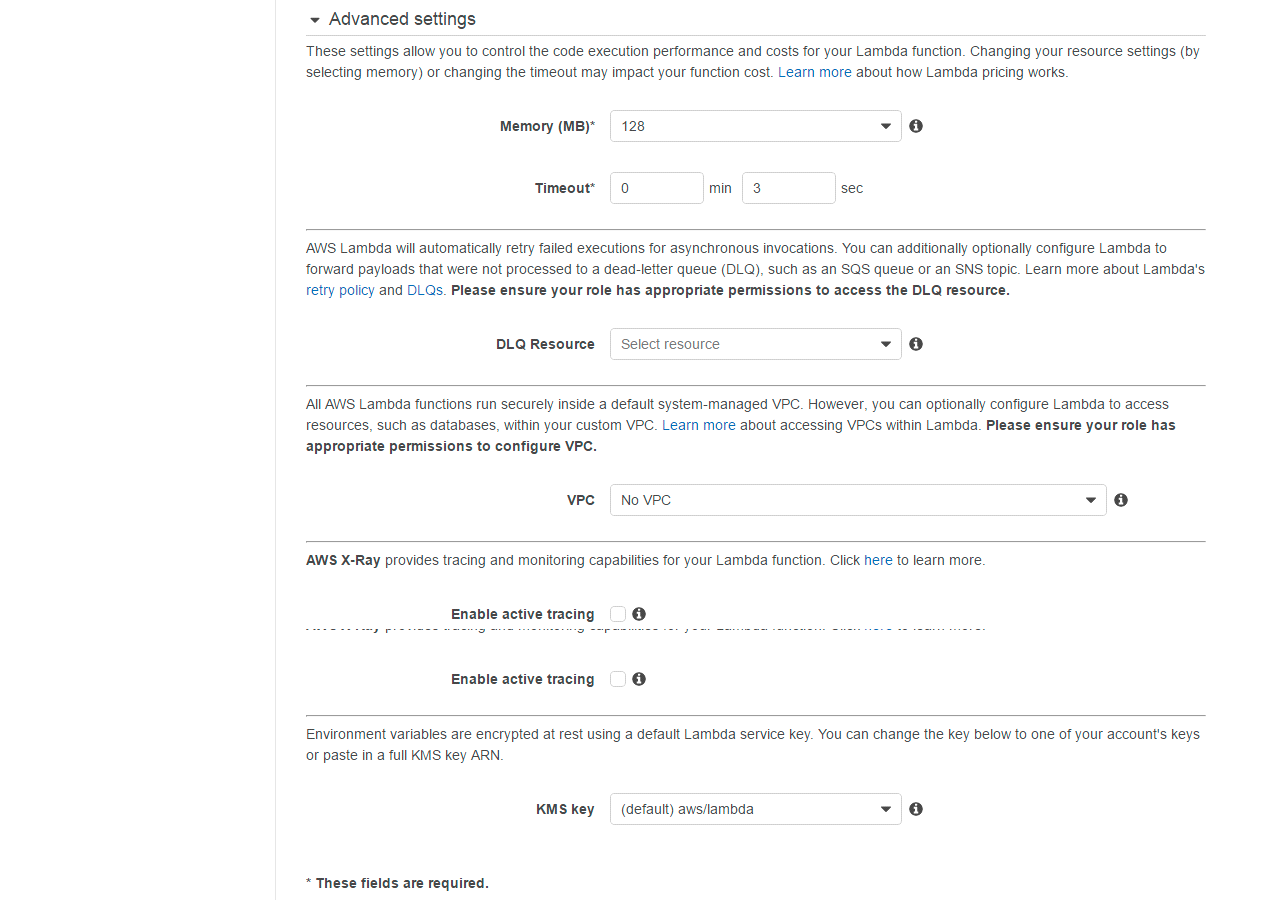
1. In configure function tab, we have to enter function name, description and environment in which we want to run and can change function code in Lambda function code tab. We have created a Lambda function which will reverse a string and written logic for same in Lambda function code tab.



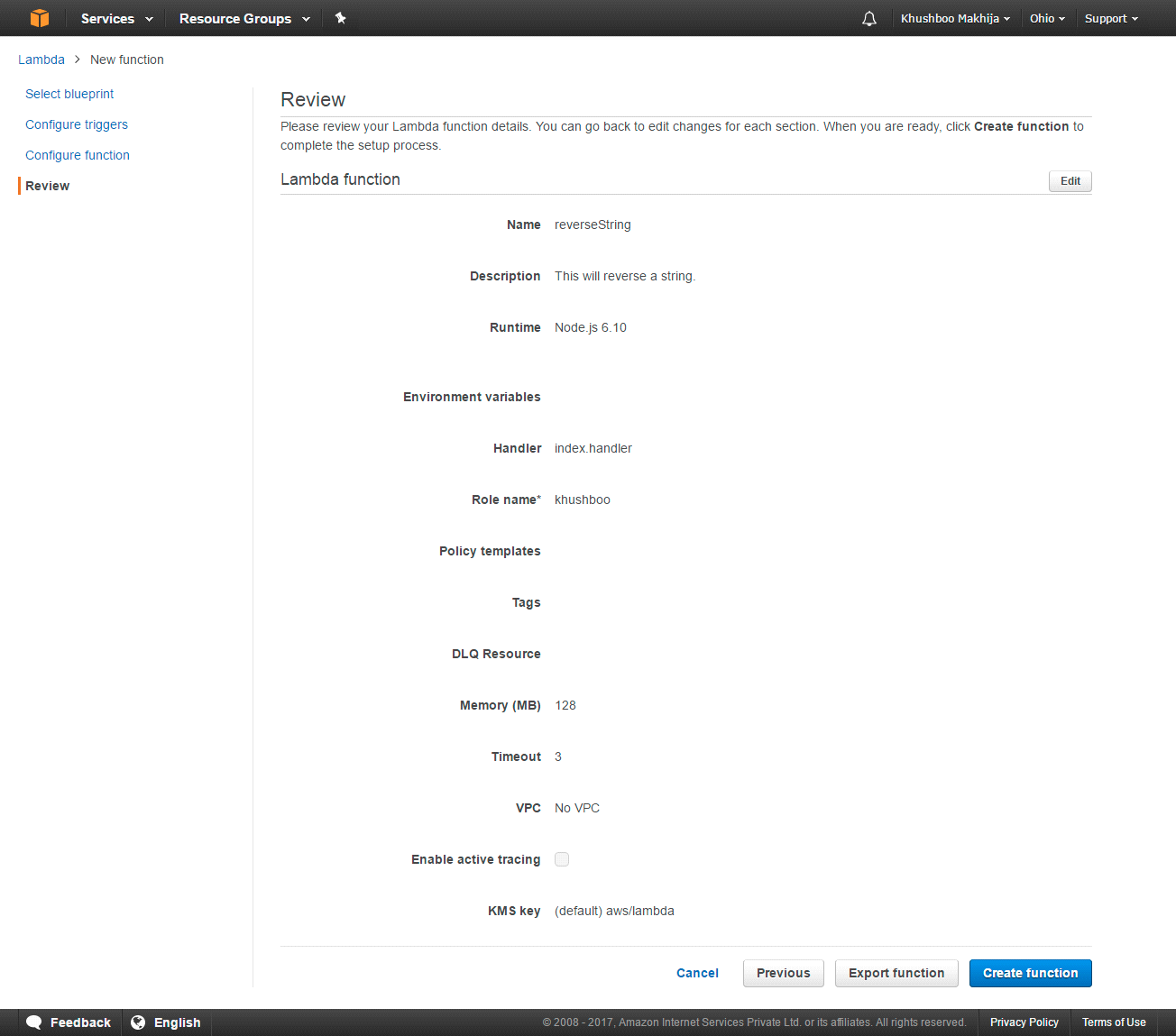
Scroll Down and set Role for the function. Select Create New Role from template(s) and enter role name.



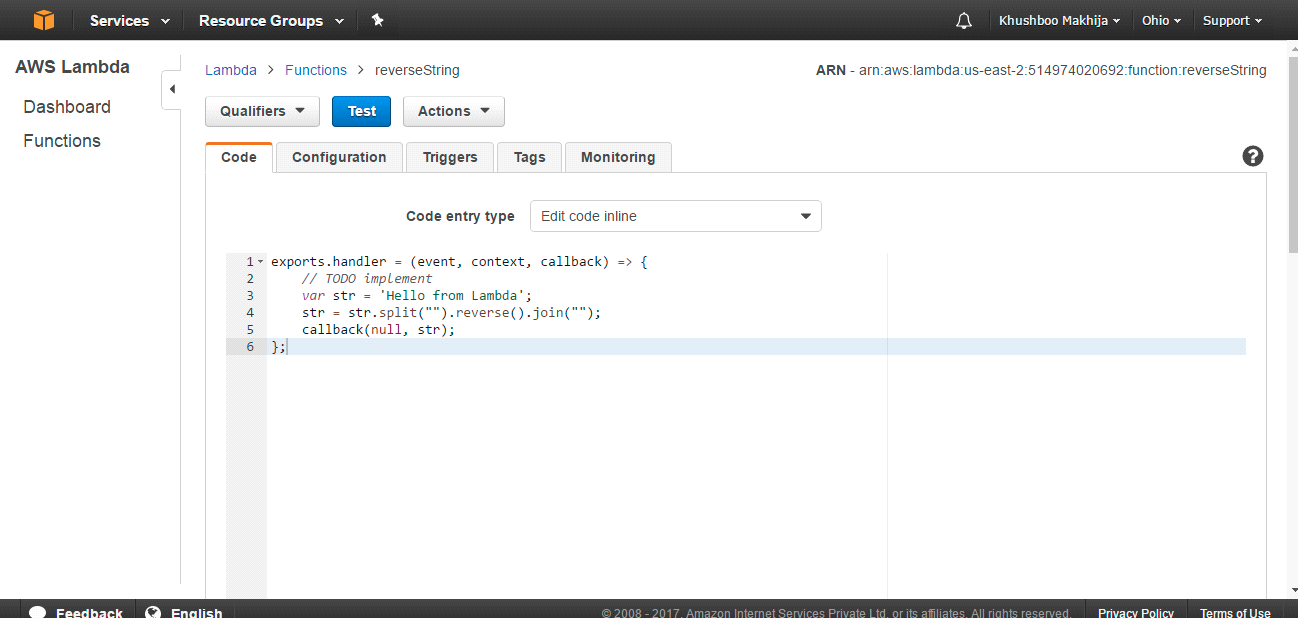
Click on Advance Tab and here memory for function is set for 128mb and timeout for running function is set for 3s ( the amount depends on the scalability).



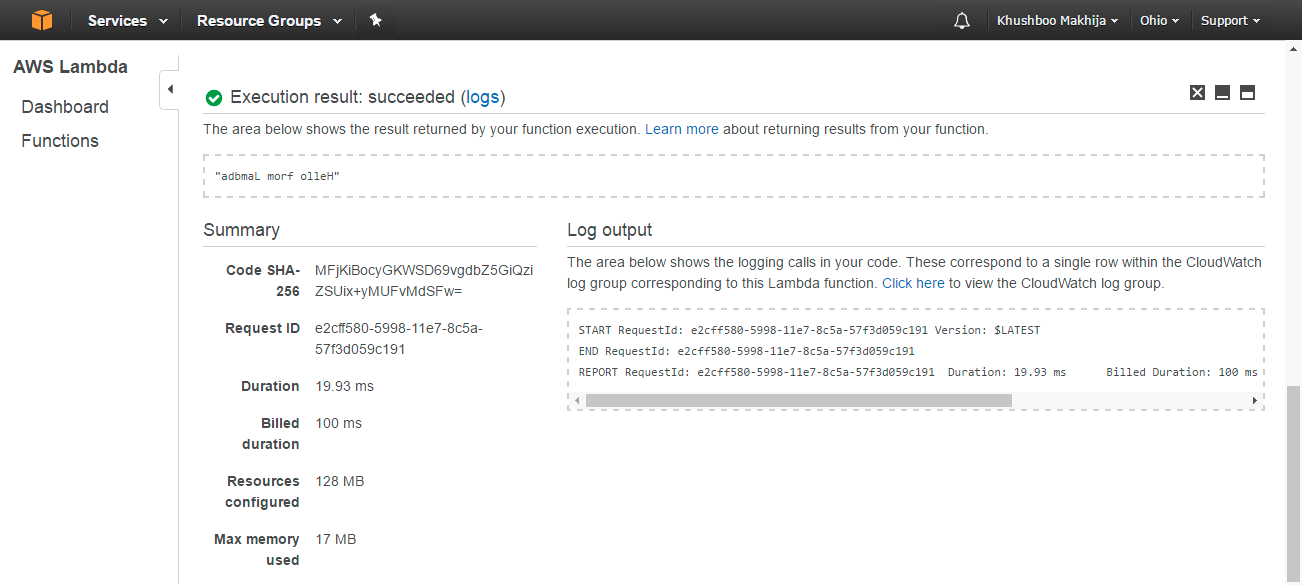
1. After that review of function will come and then click on Create function.



We can test your Lambda function by clicking on Test.

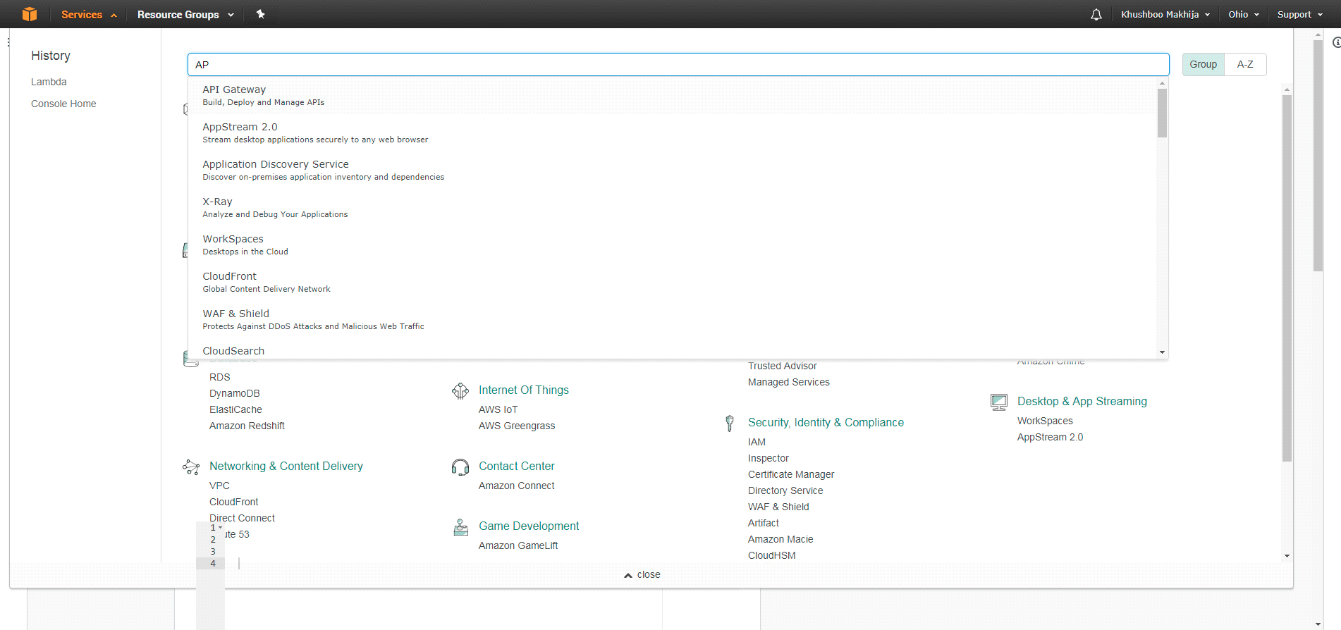


We can see our function output in execution result. We have created function of reverse string of “Hello from Lambda” so we can see the result.

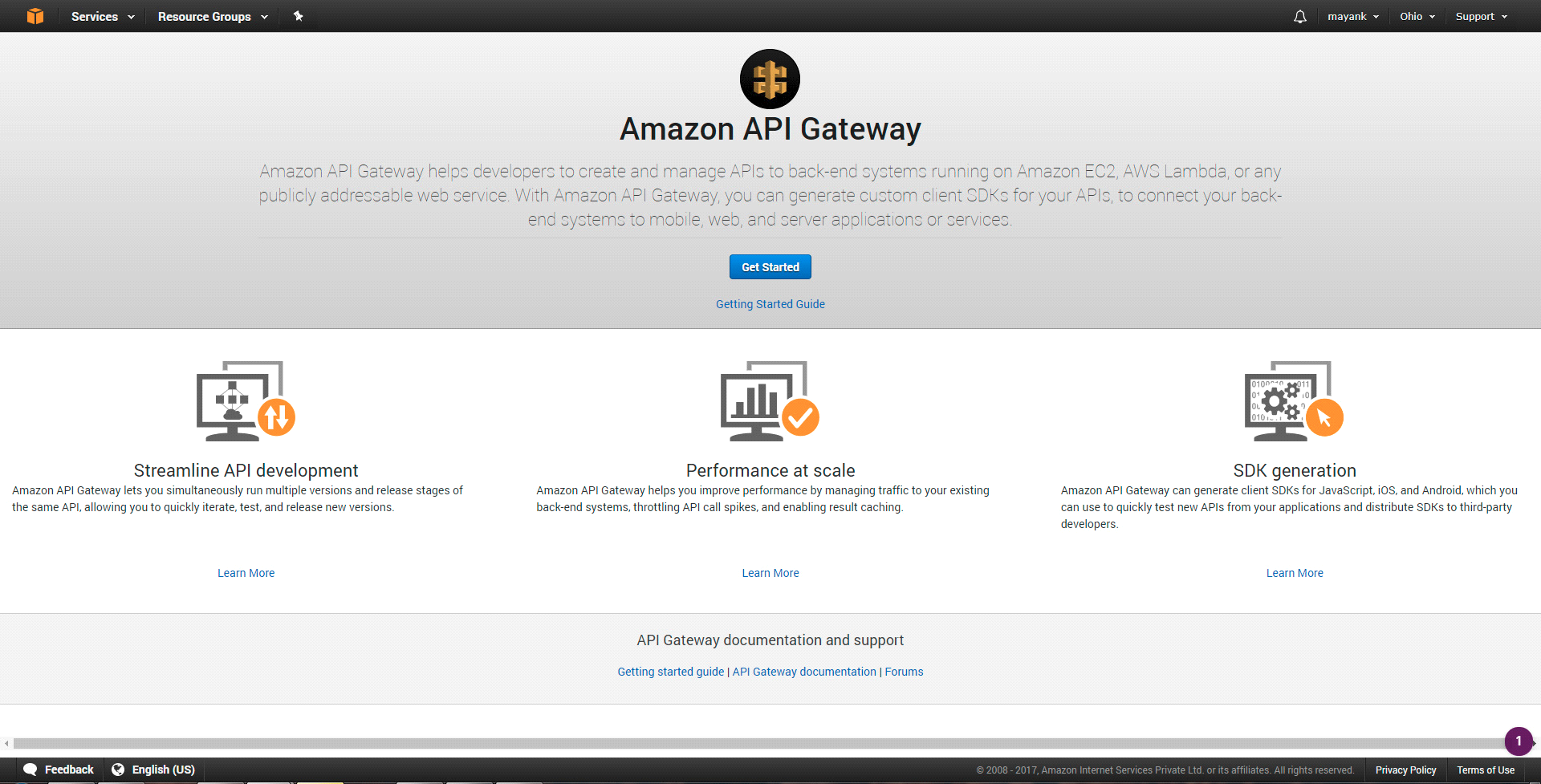


# Steps for Creating API for Lambda Function

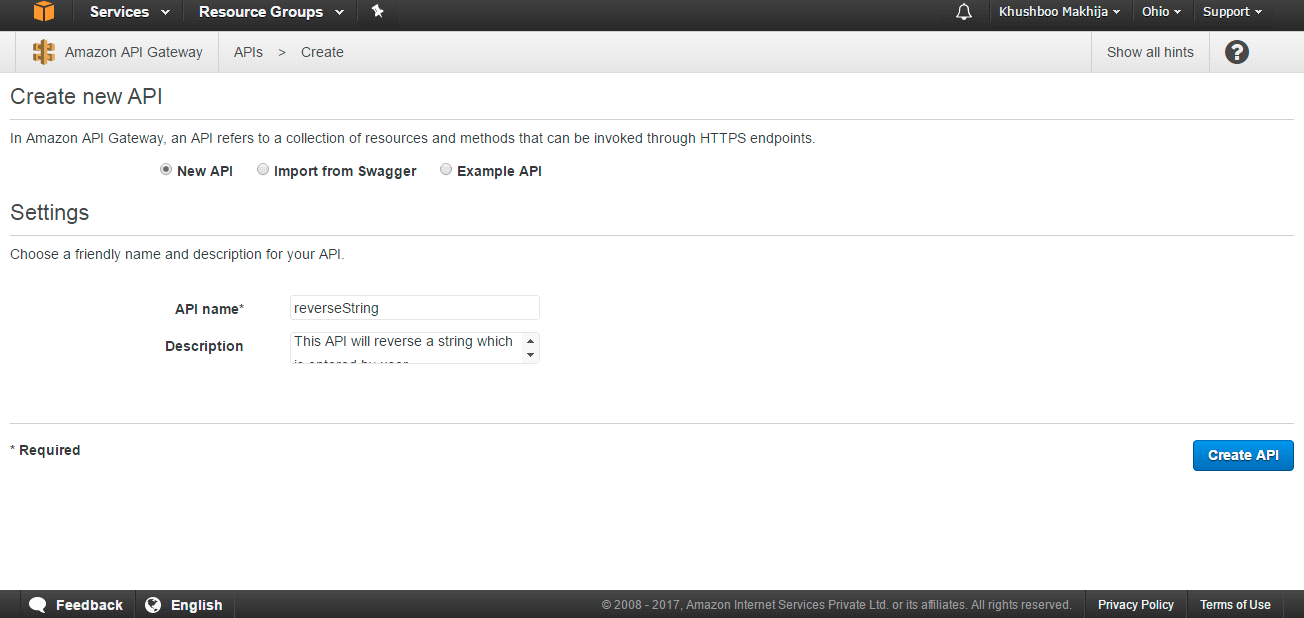
1. Go in service Tab and search for API Gateway and click on API Gateway.



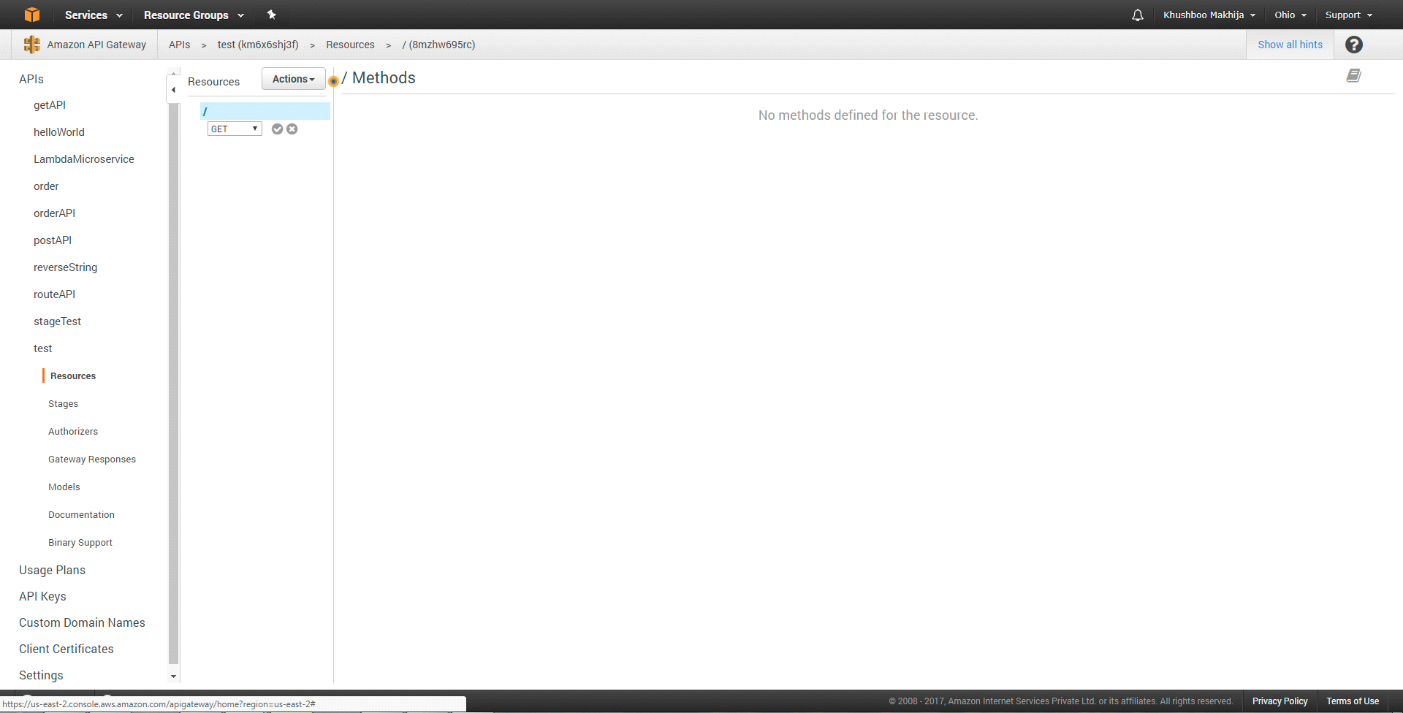
1. Click on Get Started.



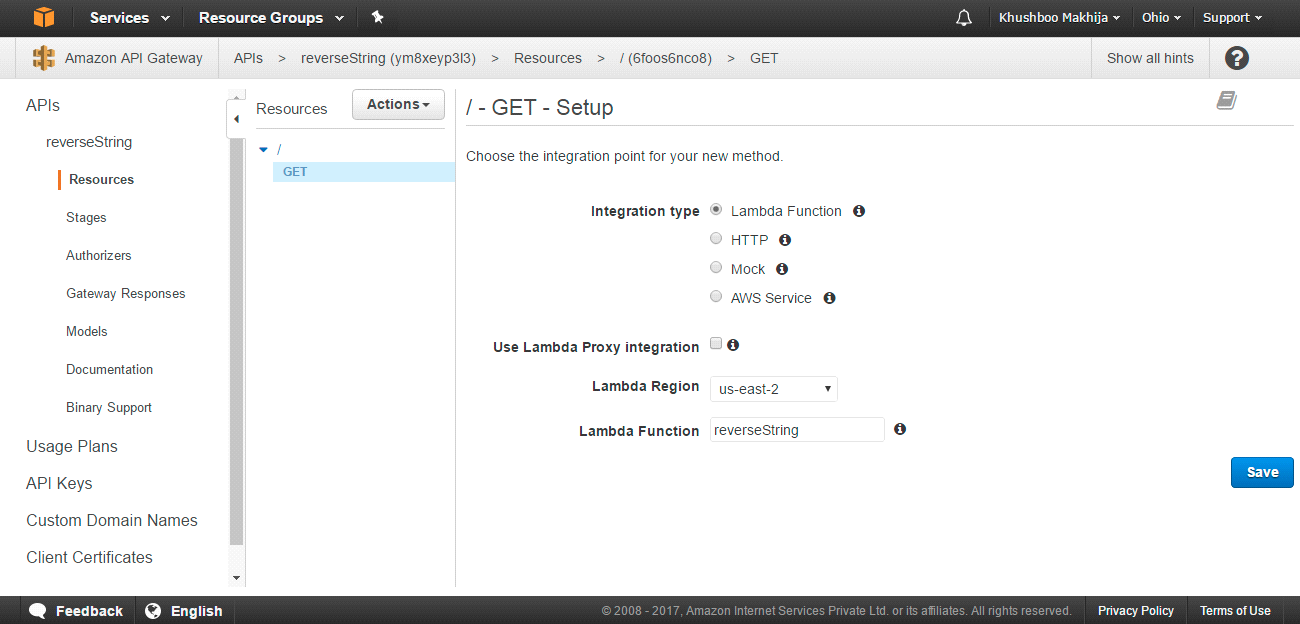
1. Click on New API and enter API name, description and then click on Create API.



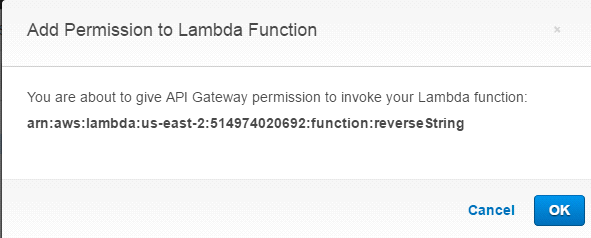
1. Here we creating API whose request type is Get so click on Actions and then click on Create Method in which we have to select GET as method.



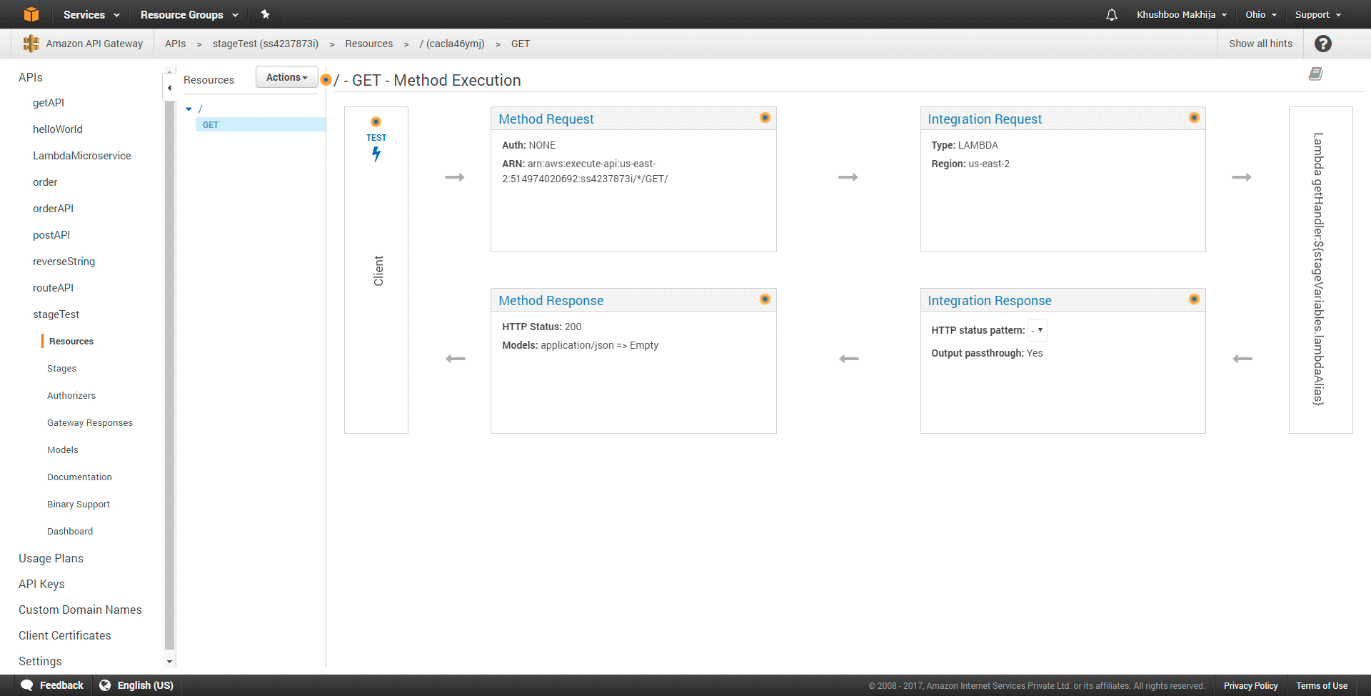
1. Click on right button below Get method. Then select Lambda Region, enter your Lambda function name and then click on Save.



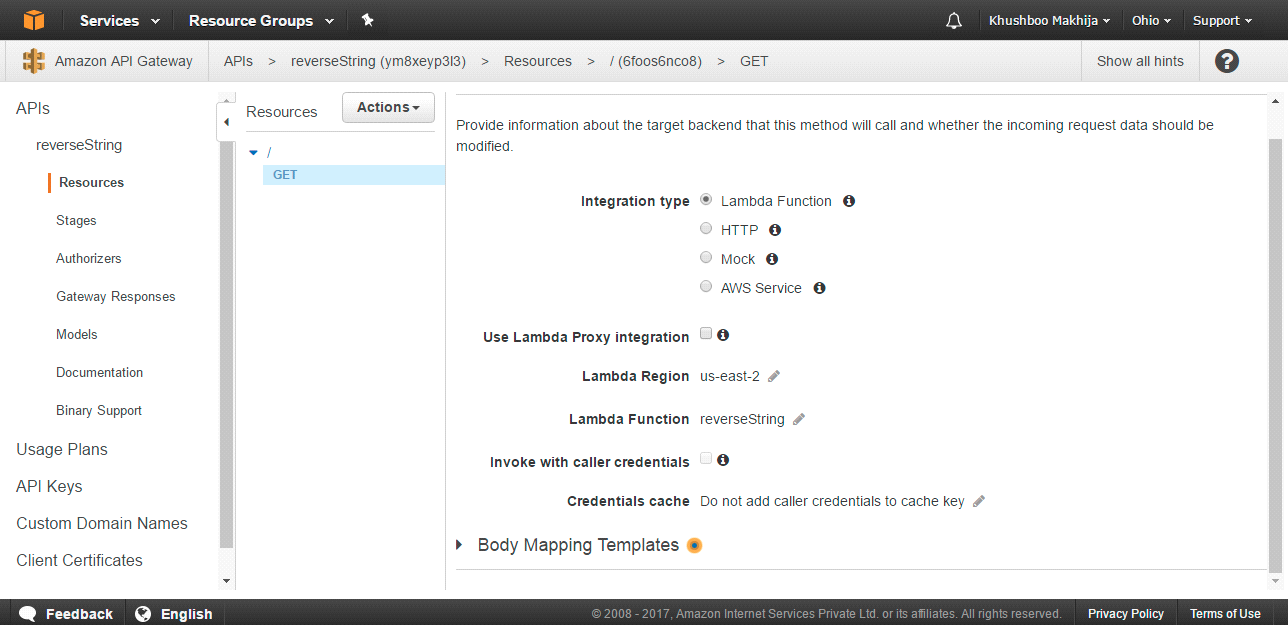
For creating API, you need to add permission so click on OK.



1. In Get Method Execution Screen, click on Integration Request.

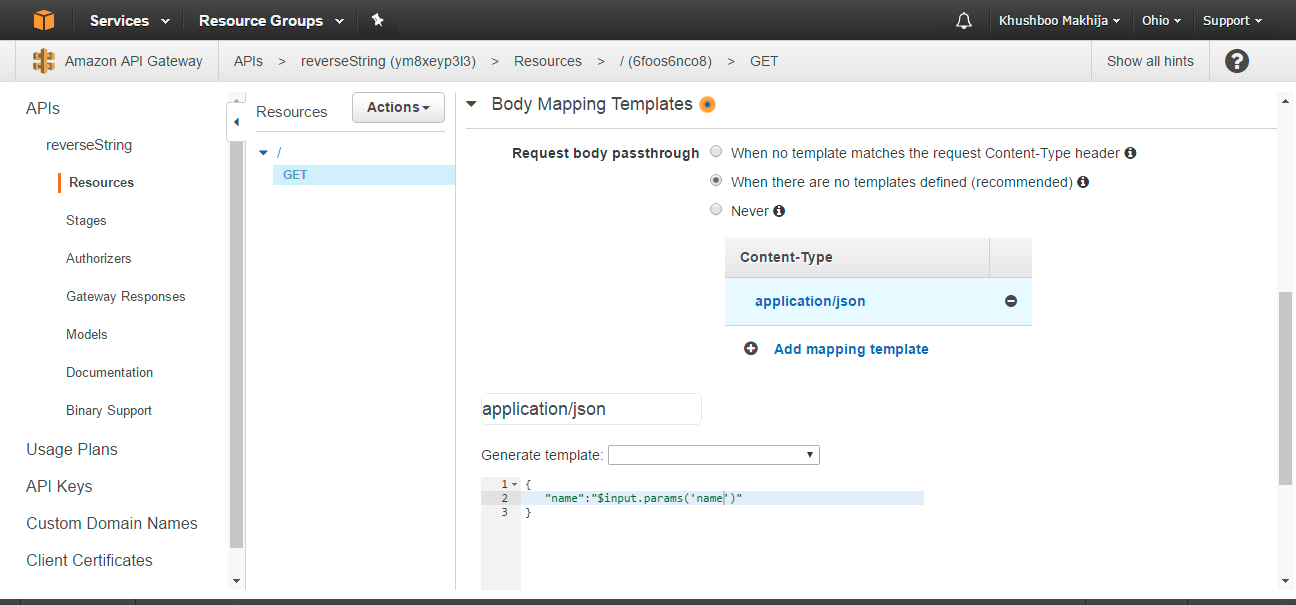


We can check whether your Lambda function is selected or not.



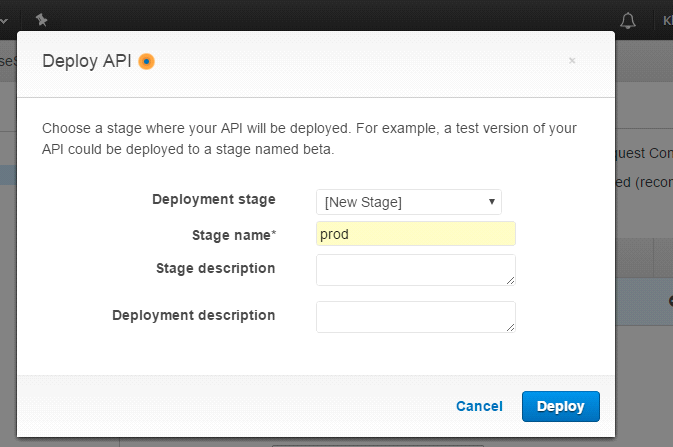
Expand Body Mapping Templates tab and select “When there are no templates defined (recommended)” and enter Content Type as “application/json” by clicking on Add mapping template and then click on right button.

Enter your JSON object in General template which helps you to pass input in headers and then click on save button.

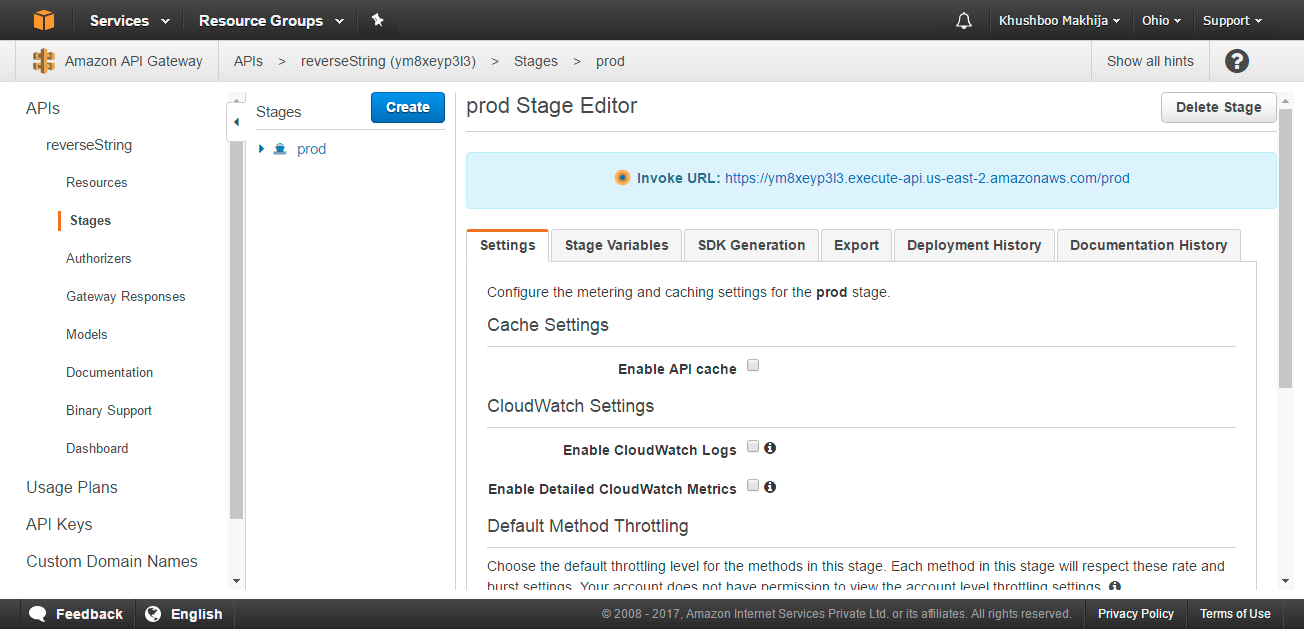


Now API is ready, to Deploy API click on Actions menu and then click on Deploy API.

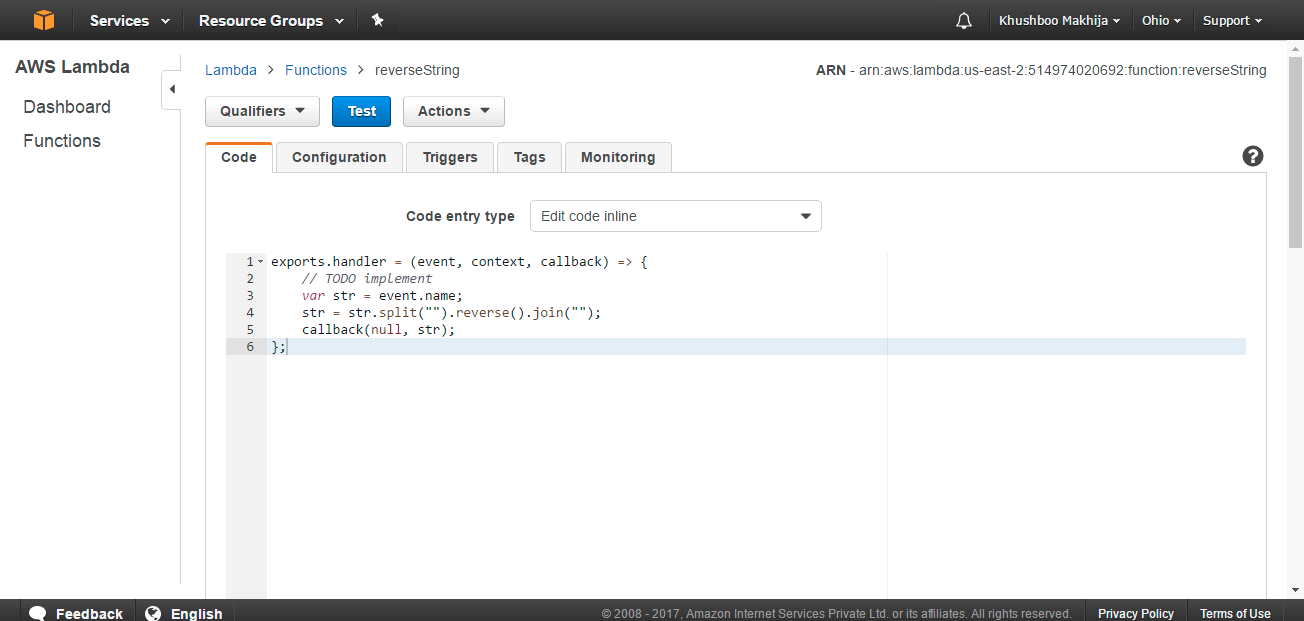
Select New Stage for Deployment Stage and enter Stage name. We can also enter production description, deployment description and then click on Deploy.



We can call API by clicking on Invoke URL.



To use input of API in our Lambda function we can get value from event object. We’re here modifying our Lambda function by going in Lambda tab and select your function name which we want to edit. Once we edited the code click on Save and Test,



Now we can API by URL followed by parameters in query string.

