|  |
| --- |
| **PROJECT** |
| **APPLICATION ENVIRONMENT** |
|  |
|  |
|  |
|  |
|  |
|  |

**Application Environment**

**Aim**

This project aims at creating a highly available, fault tolerant application environment with specific points.

**PREREQUISITES**

**Working with s3**

**Creating a LAMBDA FUNCTION**

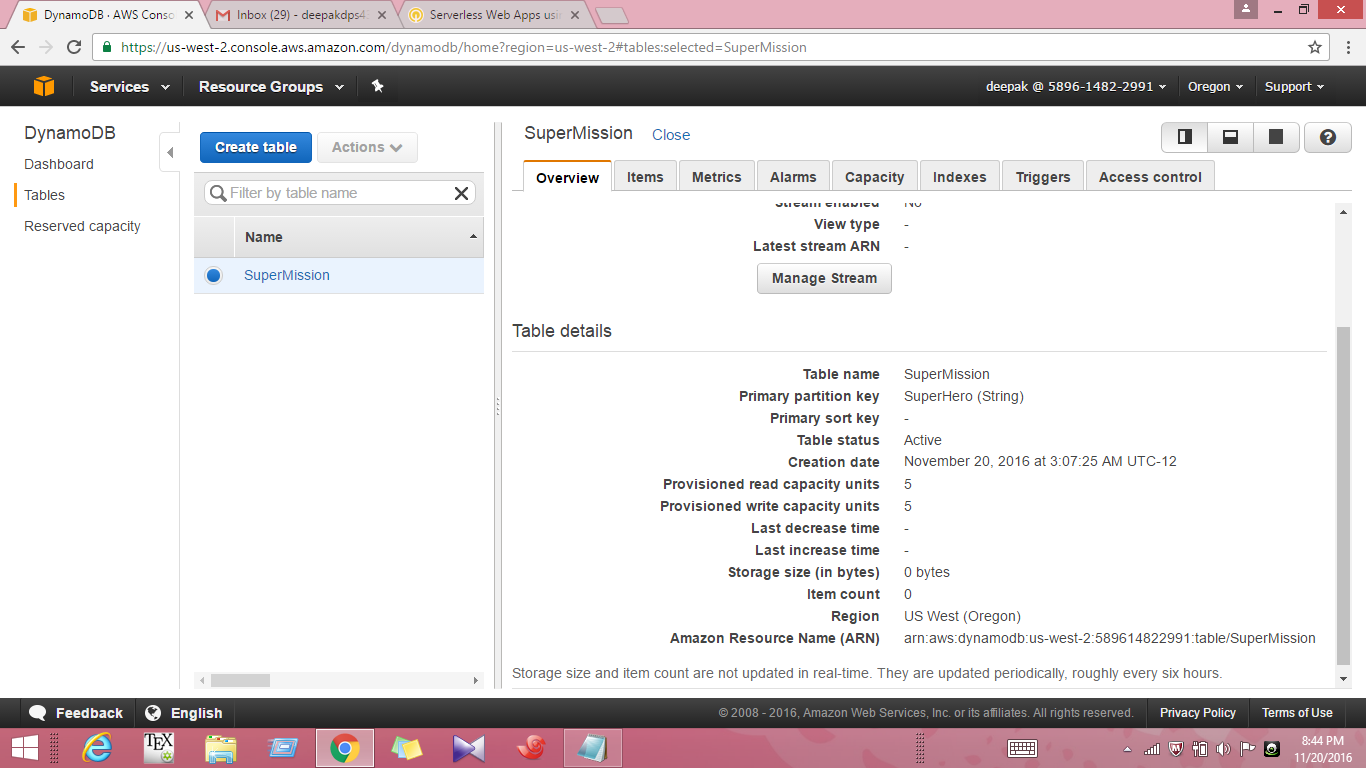
**Working with API gateway**

**Creating a Dynamodb table**

**STEPS**

**CREATE DYNAMODB TABLE ALSO CREATE IAM ROLES AND POLICIES**

**In the aws management console got to services > Dynamo DB>table name:** **SuperMission>primary key:** **SuperHero>select string**

****

**1. Again, click Create Item.**

**2. Click Text at the top left corner.**

**3. You will see a text editor. Remove all the placeholder code.**

**4. Copy the code below.**

**{**

**"SuperHero": "Superman",**

**"Villain1": "Doomsday",**

**"Villain2": "General Zod",**

**"Villain3": "Lex Luthor",**

**"MissionStatus": "In progress",**

**"SecretIdentity": "Clark Kent"**

**}**

**5. Paste the code the into the editor.**

**6. Click Save.**

**7. Repeat the above step to enter the third and fourth items.**

**{**

**"SuperHero": "The Winchester Brothers",**

**"Villain1": "Vampires",**

**"Villain2": "Ghosts",**

**"Villain3": "Werewolves",**

**"MissionStatus": "Complete",**

**"SecretIdentity": "Sam and Dean"**

**}**

**{**

**"SuperHero": "Iron Man",**

**"Villain1": "Apocalypse",**

**"Villain2": "Doctor Doom",**

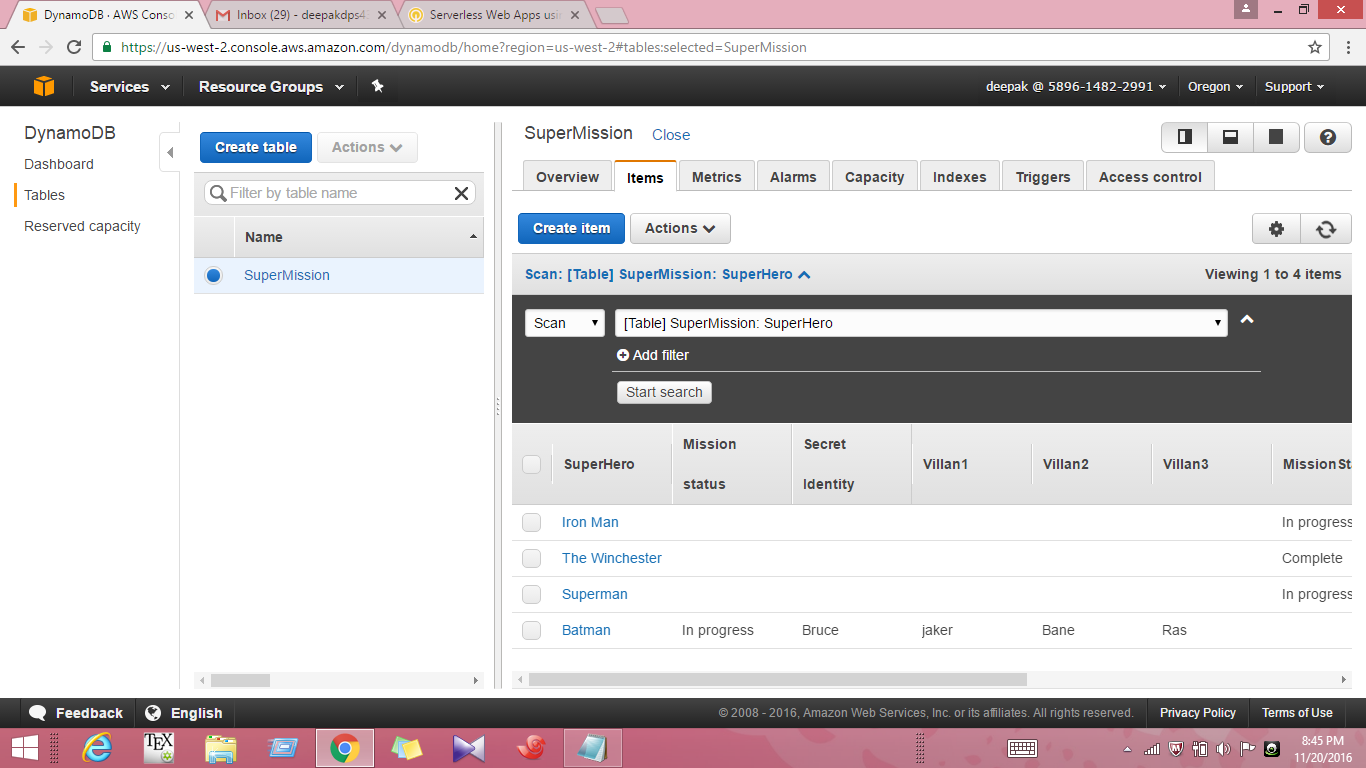
**"Villain3": "Loki",**

**"MissionStatus": "In progress",**

**"SecretIdentity": "Tony Stark"**

**}**

**8. Click the Overview tab.**

****

**9. Scroll down and copy the Amazon Resource Name (ARN). It will look something like this: am.aws:dynamodb:us-east-**

**something like this: am:aws:dynamodb:us-east-1:682455246427•table/SuperMission. Paste the ARN into your local text editor. You will need this information later.**

**Congratulations! You learned to enter data to DynamoDB table using the AWS console, using two different methods.**

**Create IAM Policies and Roles**

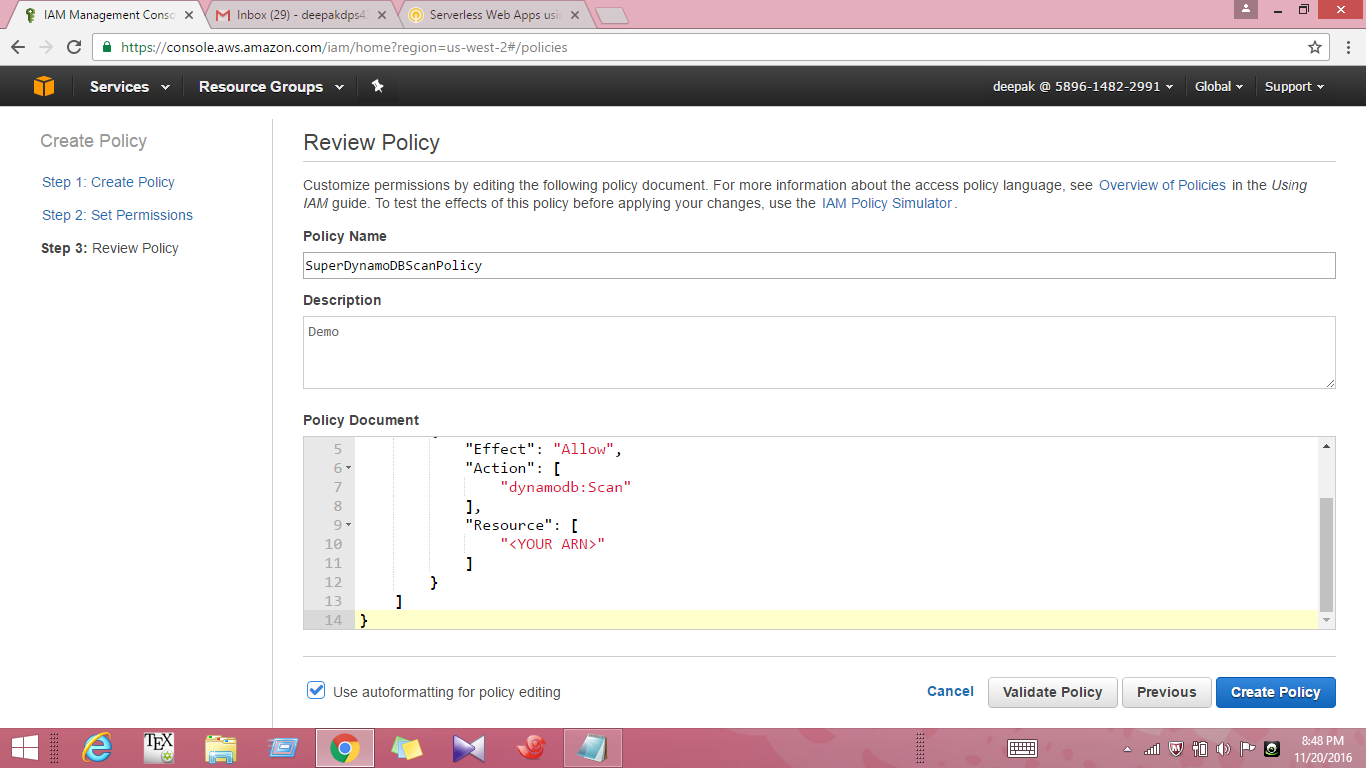
**10. On the Services menu, click IAM.**

**11. In the navigation pane, click Policies.**

**12. Click Get Started, then click Create Policy.**

**13. Locate the option to Create Your Own Policy and click Select. Note A list of standard policies are pre-built for you, but in this lab you will create your own.**

**14. For Name, type SuperDynamoDBScanPolicy**

****

**15. For Description, type Demc 16**

**16. Copy the code below and paste it into the editor. Note Remove any default text from the editor first.**

**{**

**"Version": "2012-10-17",**

**"Statement": [**

**{**

**"Effect": "Allow",**

**"Action": [**

**"dynamodb:Scan"**

**],**

**"Resource": [**

**"<YOUR ARN>"**

**]**

**}**

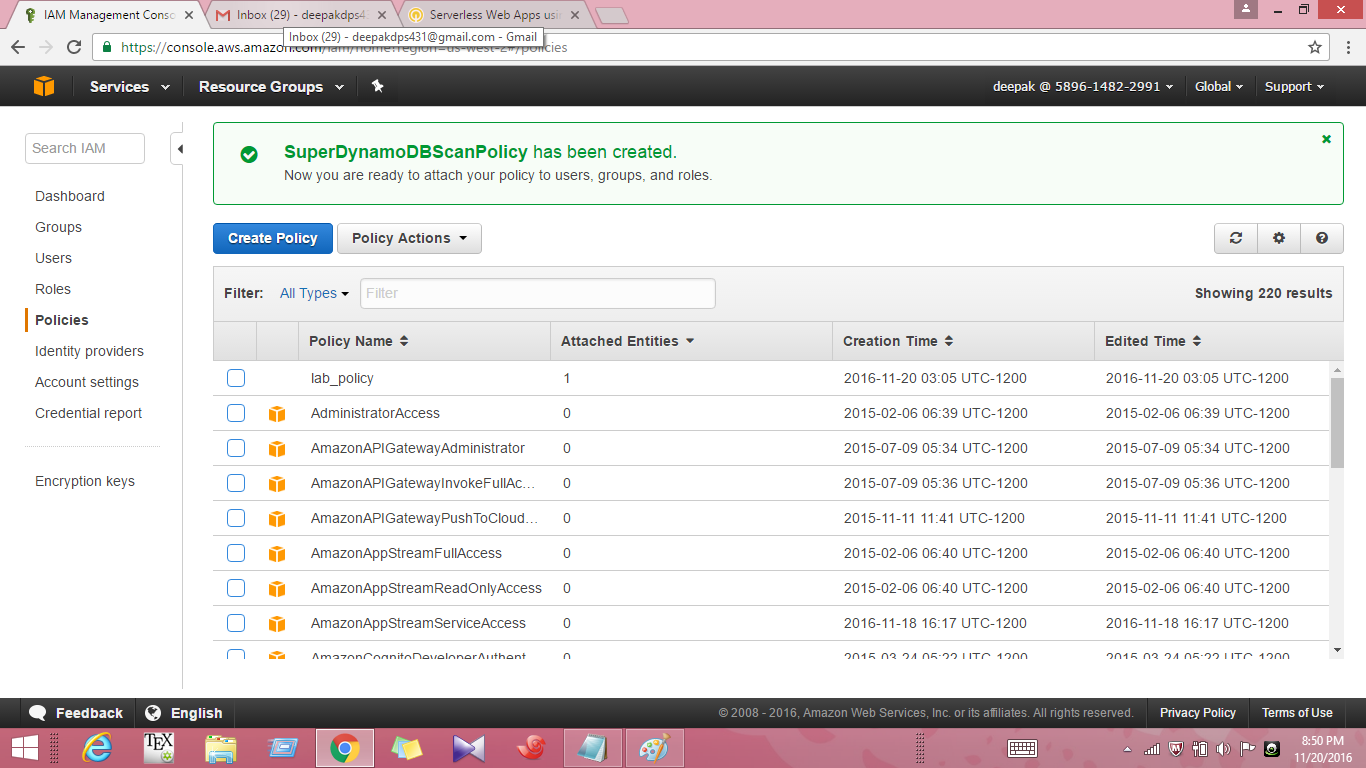
**]**

**}**

**17. Replace <YOUR ARN> with the DynamoDB ARN you copied earlier.**

**18. Click Validate Policy. The system will check the policy you created and verify there are no errors.**

**19. Click Create Policy.**

****

**20. Next, you will create the second policy you will need. Click Create Policy.**

**21. Locate the option to Create Your Own Policy and click Select.**

**22. For Policy Name, type SuperDynamoDBQueryPolicy**

**23. For Description, type Demc 16**

**24. Copy the code below and paste it into the editor. Note Remove any default text.**

**{**

**"Version": "2012-10-17",**

**"Statement": [**

**{**

**"Effect": "Allow",**

**"Action": [**

**"dynamodb:Query"**

**],**

**"Resource": [**

**"<YOUR ARN>"**

**],**

**"Condition": {**

**"ForAllValues:StringEquals": {**

**"dynamodb:Attributes": [**

**"SuperHero",**

**"MissionStatus",**

**"Villain1",**

**"Villain2",**

**"Villain3"**

**]**

**},**

**"StringEquals": {**

**"dynamodb:Select": "SPECIFIC\_ATTRIBUTES"**

**}**

**}**

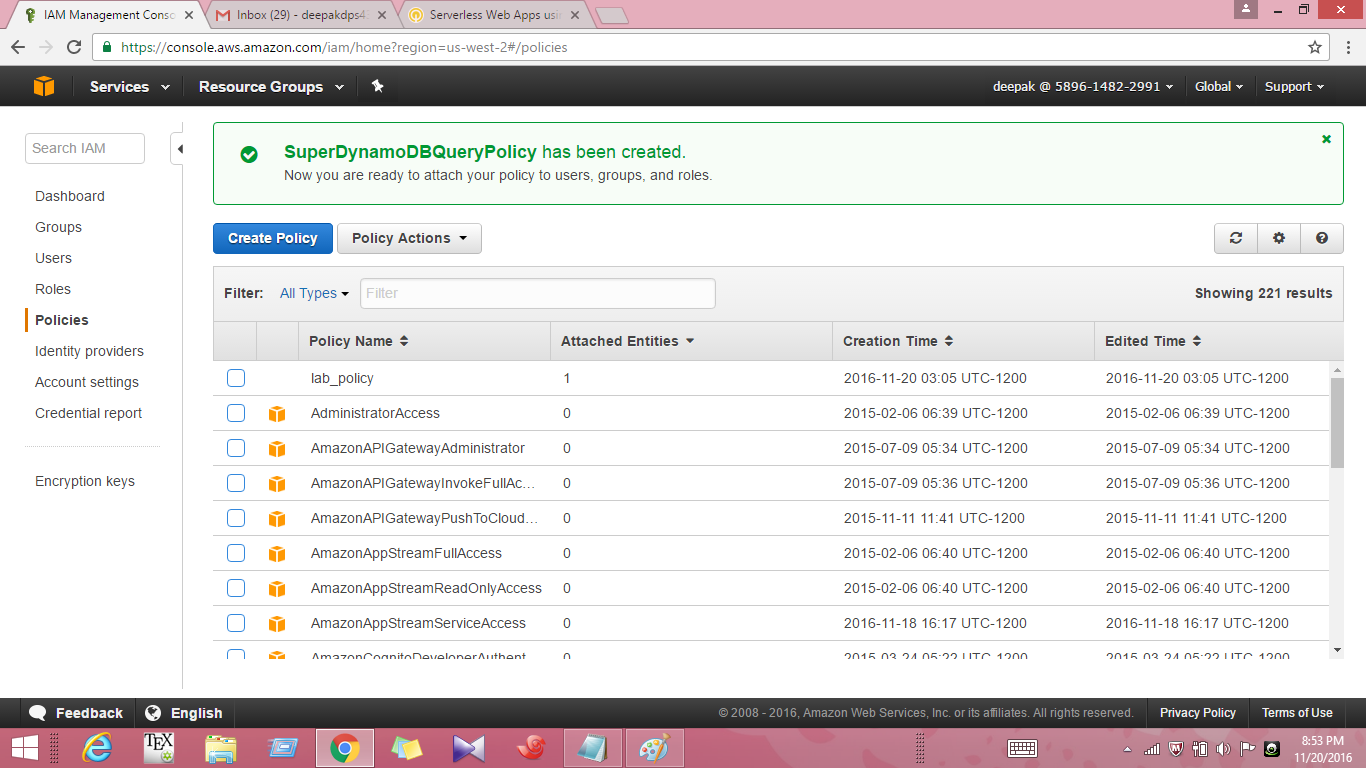
**}**

**]**

**}**

**25. Replace the place holder <YOUR ARN> with the DynamoDB MN you copied earlier. Note Make sure to remove the < > brackets when you paste in your ARN.**

**26. Click Validate Policy. The system will check the policy you created and verify there are no errors.**

****

**27 Click Create Policy. Next, you will create two new roles.**

**28.In the navigation pane, click Roles.**

**29. Click Create New Role.**

**30. For Role Name, type SuperDynamoDBScanRole 113**

**31. Click Next Step.**

**32. For Role Type, make sure AWS Service Roles is selected (this should be the default selection).**

**33. Locate AWS Lambda and click Select.**

**34. Use the filter to search for SuperDynamoDBScanPolicy 111**

**35. Select the policy, then click Next Step.**

**36. Click Create Role.**

**37 Now you will create the second role. Click Create New Role.**

**38. For Role Name type SuperDynamoDBQueryRole wj**

**39. Click Next Step.**

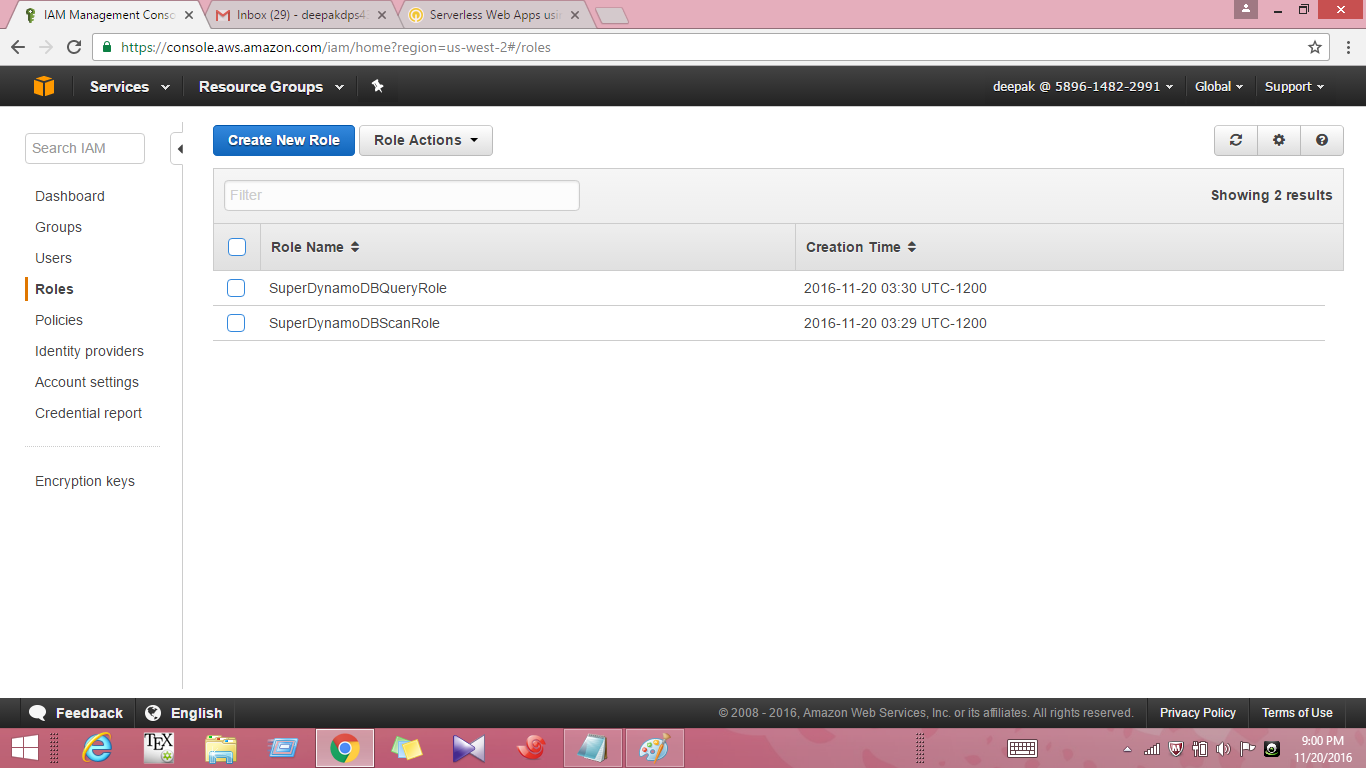
**40. Again, for Role Type, AWS Service Roles should be selected by default.**

**41. Locate AWS Lambda and click Select.**

**42. Use the filter to search for SuperDynamoDBQueryPolicy**

**43. Select the policy, then click Next Step.**

**44. Click Create Role.**

****

**CREATE AND TEST LAMBDA FUNCTION**

**CREATE LAMBDA FUNCTIONS**

**The first Lambda function will retrieve a list of super heroes that are stored in the SuperMission DynamoDB table.**

**The second will retrieve the mission details from the same DynamoDB table.**

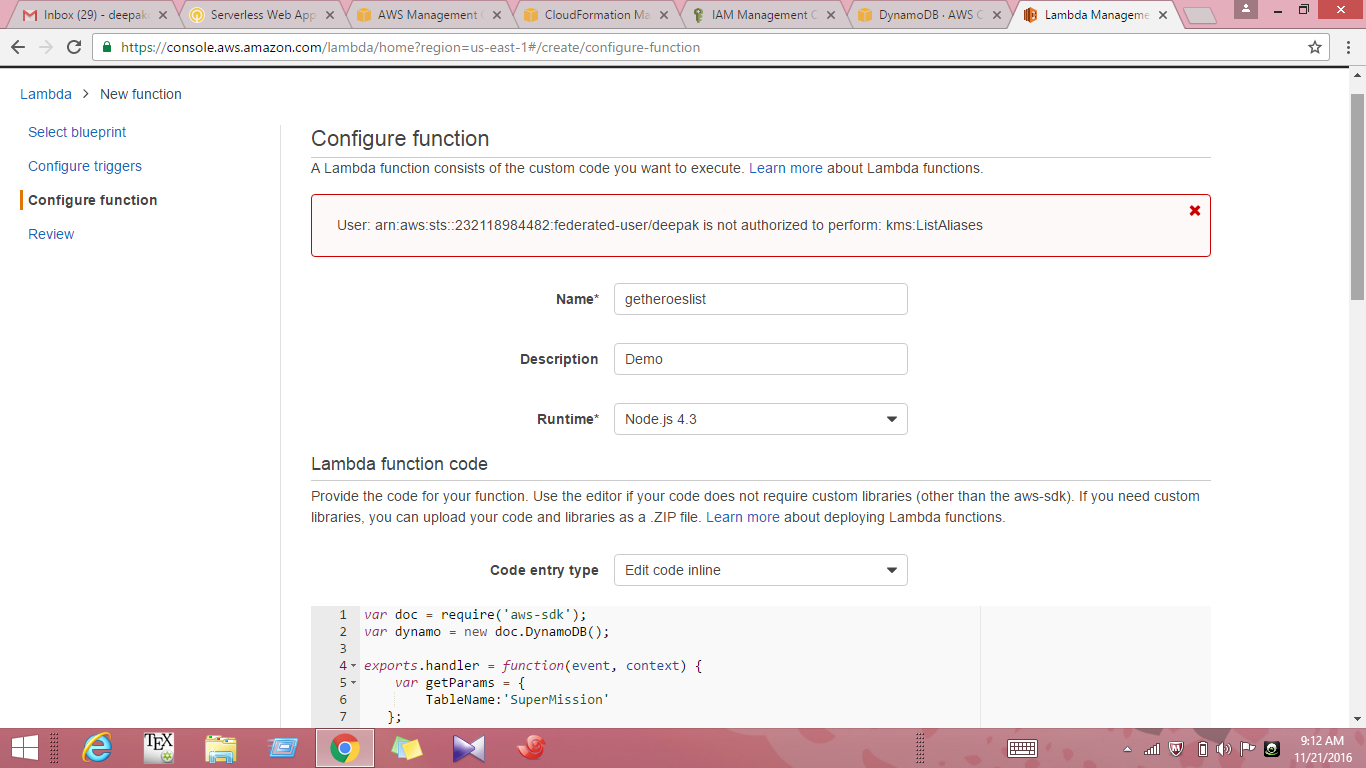
**1. On the Services menu, click Lambda.**

**2. Click Create a Lambda Function.**

**3. On the Select blueprint page, click Blank Function.**

**4. On the Configure triggers page, click Next.**

**Now you will configure your function.**

****

**5. Configure your function by typing Name ( getheroeslist ) and Description.**

**6. Select Node.js 4.3 for Runtime.**

**7. Select Edit code inline for Code entry type.**

**8. In the editor, remove all of the code, and provide the following code:-**

**/// This Lambda function retrieves a list of super heroes stored in the SuperMission DynamoDB table.**

**var doc = require('aws-sdk');**

**var dynamo = new doc.DynamoDB();**

**exports.handler = function(event, context) {**

**var getParams = {**

**TableName:'SuperMission'**

**};**

**dynamo.scan(getParams, function(err, data){**

**if (err) console.log(err, err.stack); // an error occurred**

**else {**

**context.succeed(data);**

**}**

**});**

**};**

**9. Select index.handler for Handler.**

**10. Choose an existing role for Role.**

**11. Select the Scan role for DynamoDB Table.**

**12. Leave defaults in the Advanced Settings section.**

**13. Click Next and Review your Lambda function.**

**14. Then click Create function.**

**15. Click Configure test event from the Actions menu in the Lambda console.**

**16. Select 'Hello World' for Sample event template. Click save and test.**

**17. If the test is success you can see the message intimating that 'Execution Result: Succeeded'**

**( We can see a text box displaying the contents of the SuperMission DynamoDB table)**

**18. Follow same procedure to create another Lambda function to retrieve the mission details from the SuperMission DynamoDB table.**

**19. Name the second Lambda function as getmissiondetails.**

**20. In the editor provide following code for the lambda function which retrieves mission details from DynamoDB table.**

**/// Provide Query Role for DynamoDb when creating the this Lambda function.**

**var doc = require('aws-sdk');**

**var dynamo = new doc.DynamoDB();**

**exports.handler = function(event, context) {**

**condition = {};**

**condition["SuperHero"] = {**

**ComparisonOperator: 'EQ',**

**AttributeValueList:[{S: event.superhero}]**

**}**

**var getParams = {**

**TableName:'SuperMission',**

**ProjectionExpression:"SuperHero, MissionStatus, Villain1, Villain2, Villain3",**

**KeyConditions: condition**

**};**

**dynamo.query(getParams, function(err, data){**

**if (err) console.log(err, err.stack); // an error occurred**

**else {**

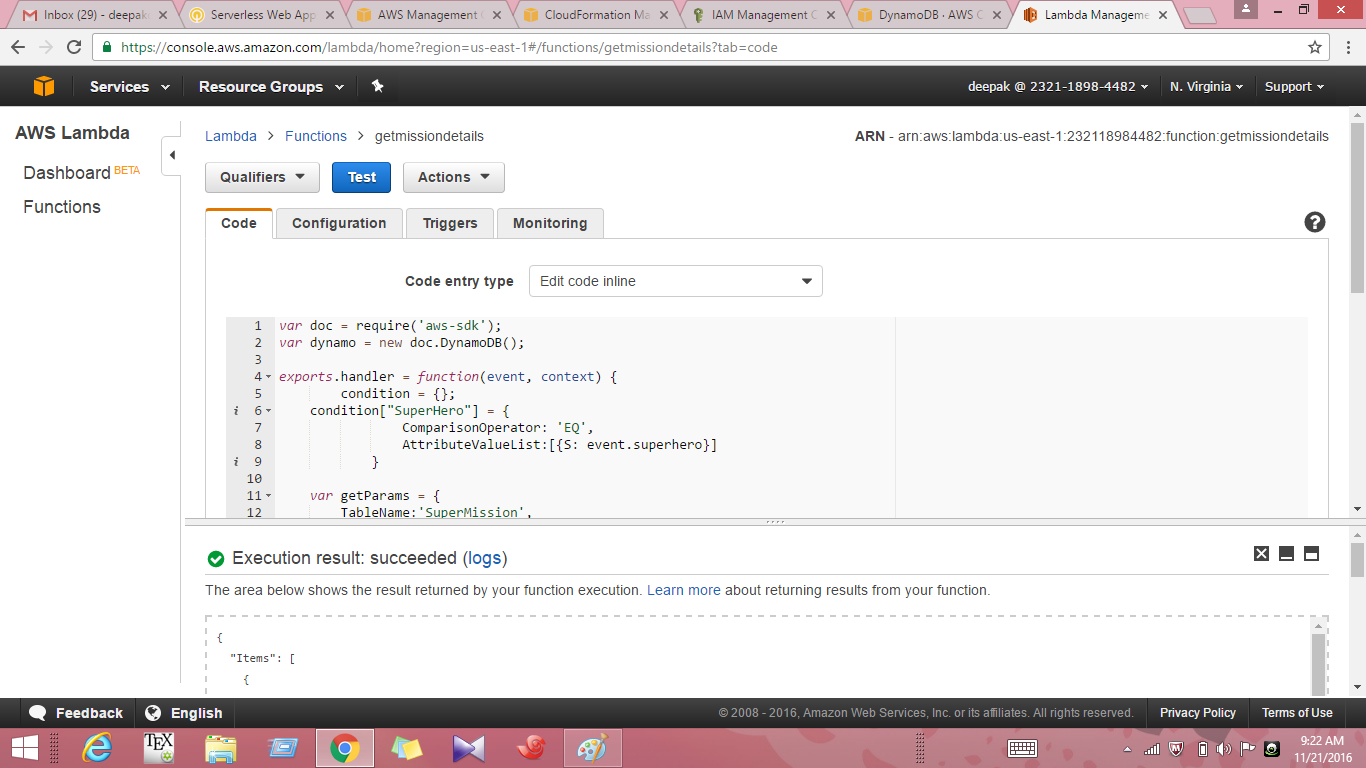
**context.succeed(data);**

**}**

**});**

**};**

**21. Test the function by giving the following code in the editor.**

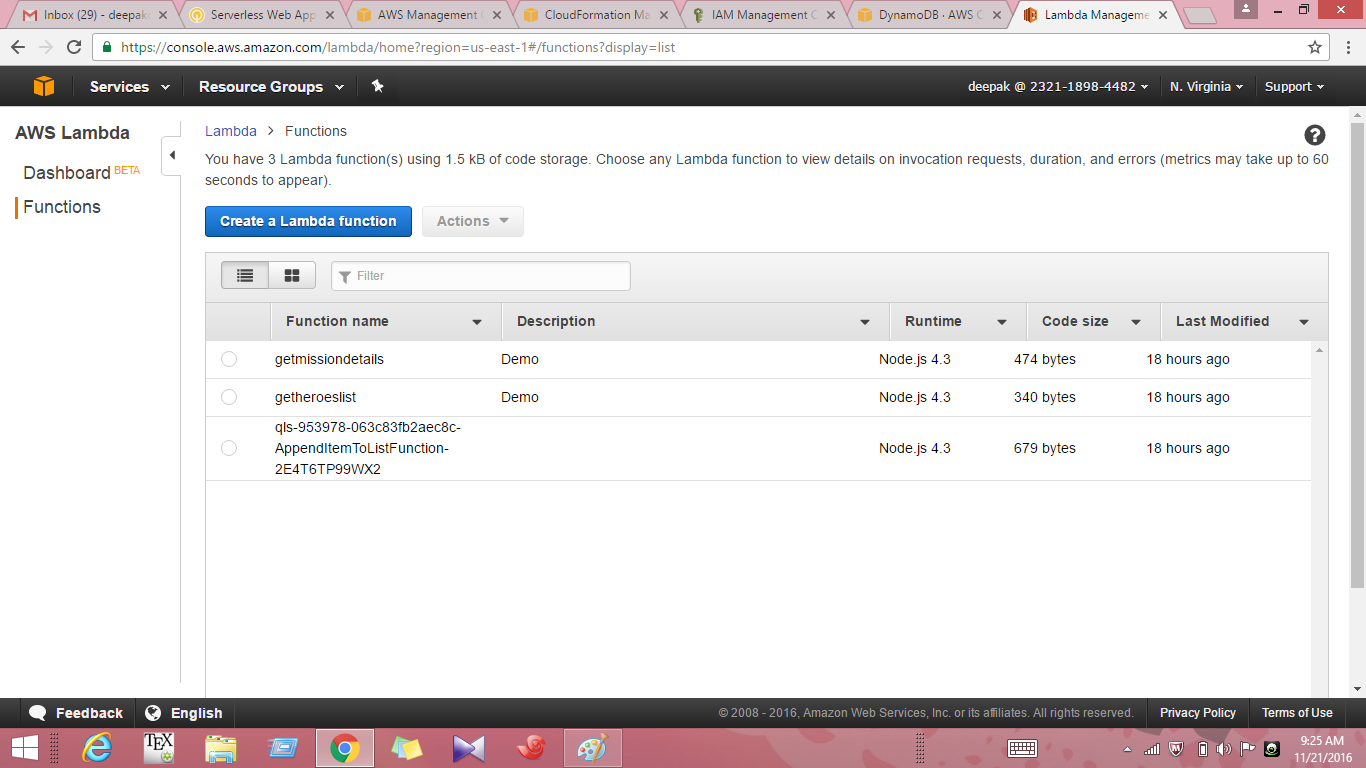
****

**/// If the test is success, the details of Batman's mission will be returned.**

**{**

**"superhero": "Batman"**

**}**

****

**CREATE AN API GENERATE SDK FOR THE API ALSO PUBLISH THE CONTENT USING S3**

**Create and Deploy an API**

**1. On the Services menu, click API Gateway.**

**2. Click Get Started.**

**3. On the Create Example API screen, click OK. This example API is prepopulated for you by AWS.**

**4. To start, click the New API radio button.**

**5. For API Name, type SuperheroesMi ssi on ri**

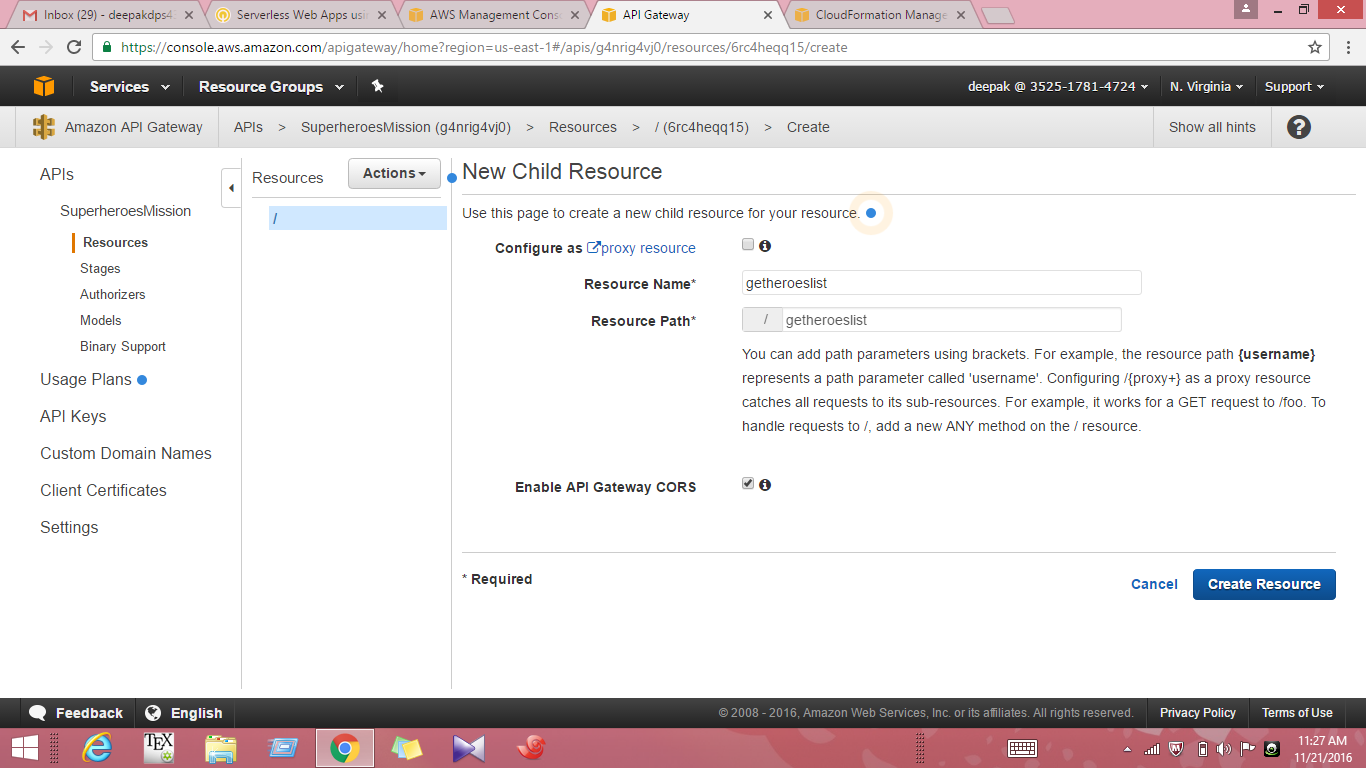
**6. (Optional) For Description, type Demc It**

**7. Click Create API.**

**8. Click / (the root of the folder you're looking at).**

**9. Click Actions, then click Create Resource.**

**10. For Resource Name, type getheroesli**

****

**11. For Resource Path, type getheroesli st**

**Notice that this path is automatically appended to the root folder. The path may auto-populate for you.**

**12. Click Create Resource. Now you will create another child resource.**

**13. Click / (again, the root of the folder).**

**14. Click Actions, then click Create Resource.**

**15. For Resource Name, type getmissiondetails**

**16. For Resource Path, type getmissiondetails k5 (again, this path may auto-populate for you).**

**17. Click Create Resource.**

**18. Click the getheroeslist resource.**

**19. With the getheroeslist resource selected, click Actions, then click Create Method.**

**20. You will see a dropdown menu. Click Post. Then click the check mark.**

**21. Now you will set up your method. For Integration type, click Lambda Function.**

**22. For Lambda Region, click us-east-1.**

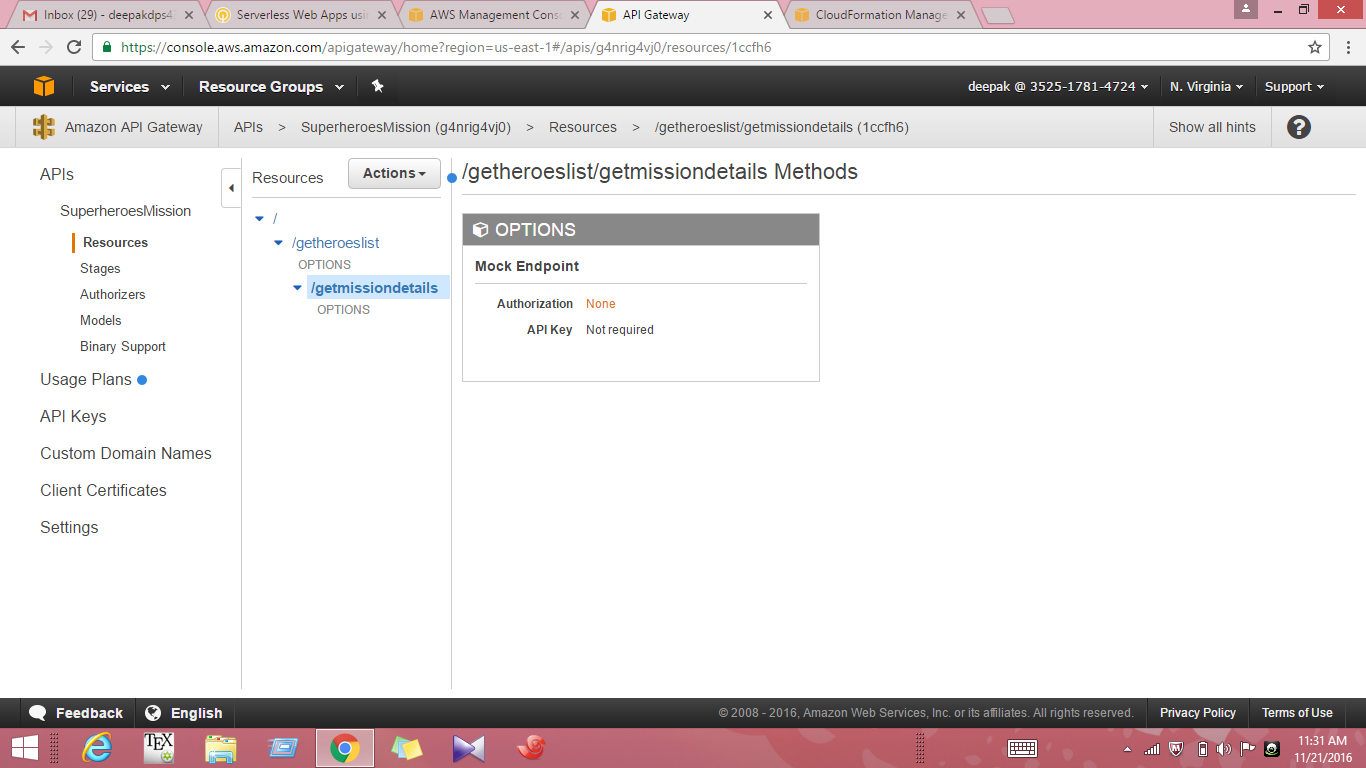
**23. For Lambda Function, select qlstack2-labinstance-XXXXXX-getheroeslistFunction-XXXXXXXXX (the exact function name is unique to your lab) and click save**

**24. You will see a warning that you are about to give the API Gateway service permission to invoke your Lambda function. Click OK.**

**25. Click the getmissiondetails resource.**

**26. With the getmissiondetails resource selected, click Actions, then click Create Method.**

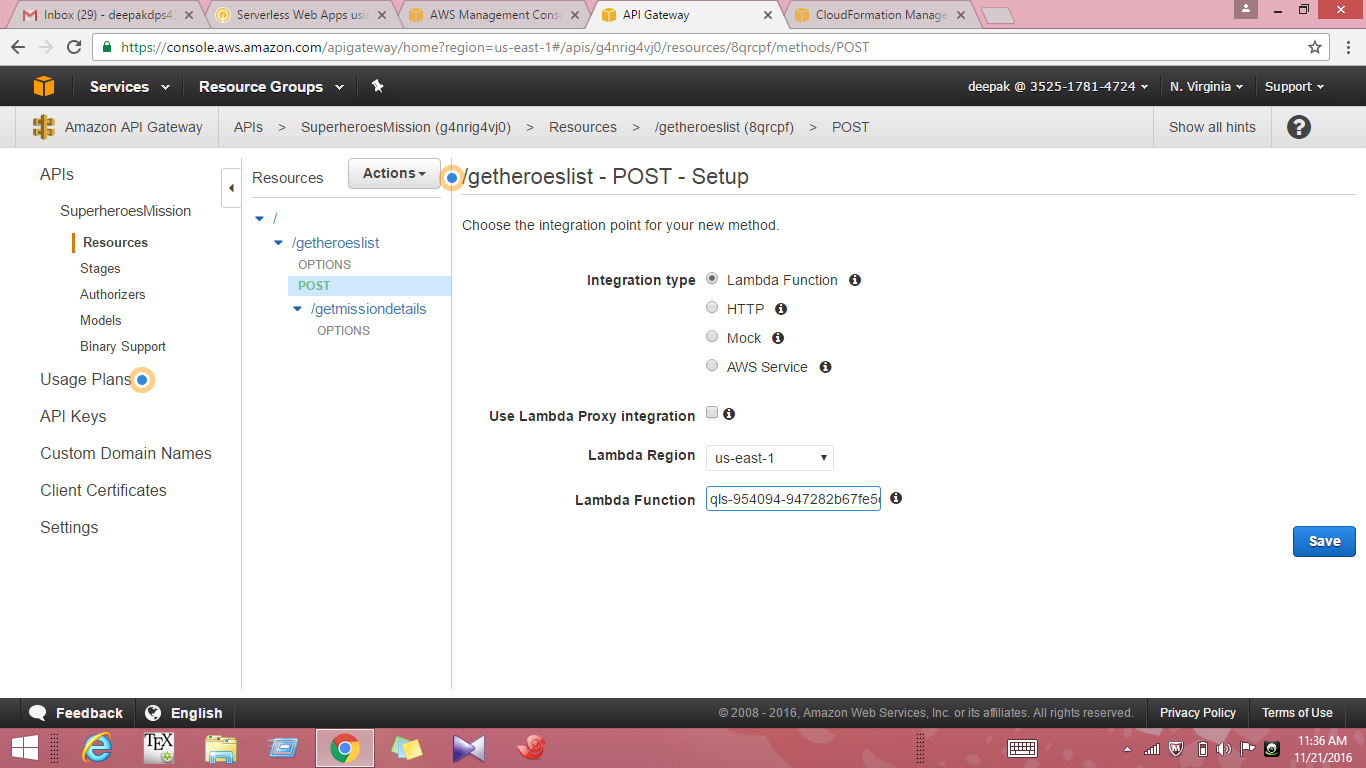
**27. You will see a dropdown menu. Click Post. Then click the check mark.**

****

**28. Now you will set up your method. For Integration type, click Lambda Function.**

**29. For Lambda Region, click us-east-1.**

**30. For Lambda Function, select qlstack2-labinstance-XXXX-getmissiondetailsFunctio-XXXXXXXXX.**

****

**31. Click Save.**

**32. You will see the same warning. Click OK.**

**Next, you will enable CORS for the methods you just created in API Gateway.**

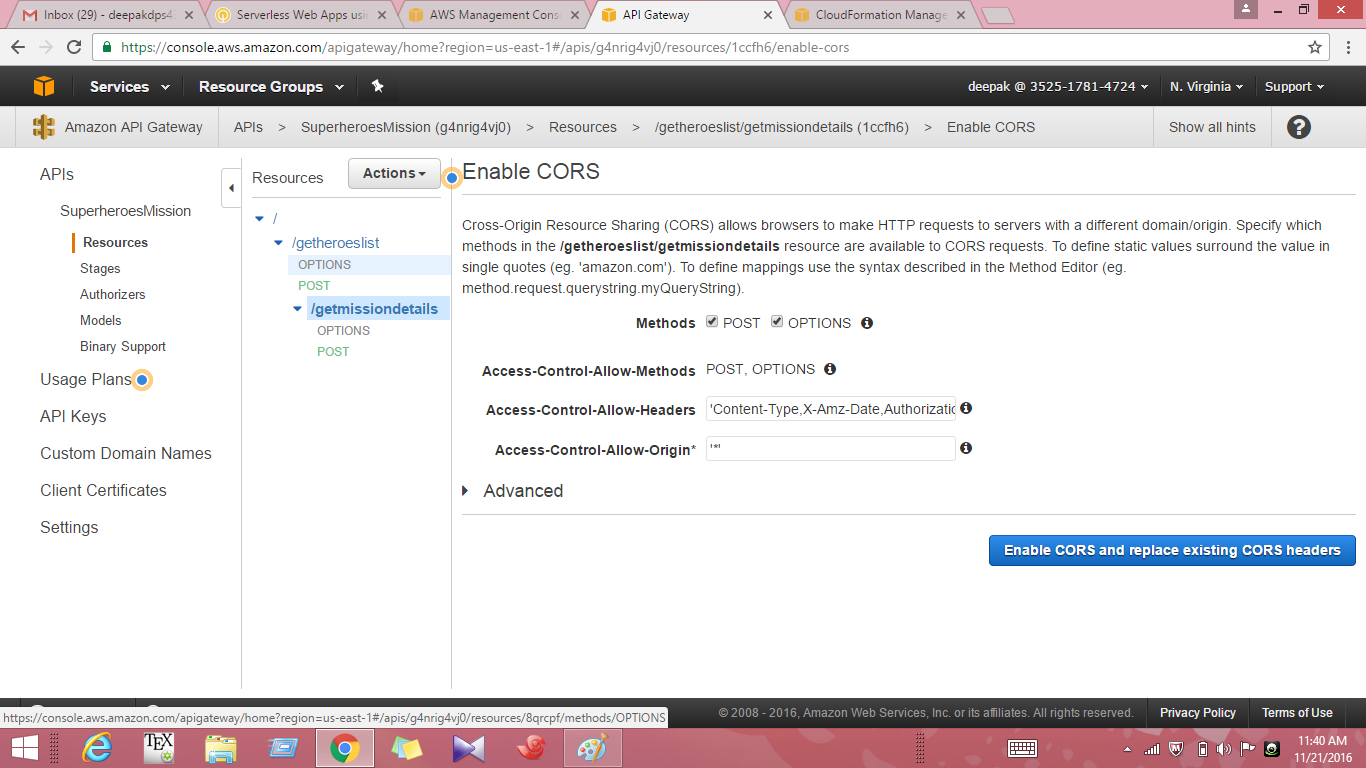
**Cross-Origin Resource Sharing (CORS) allows browsers to make FITTP requests to servers with a different domain/origin.**

**33. Click the getheroeslist resource.**

**34. With the getheroeslist resource selected, click Actions, then click Enable CORS.**

**35. Make sure both methods, Post and Options, are selected.**

**36. Click Enable CORS and replace existing CORS headers.**

****

**37. Confirm that you want to Yes, replace existing values.**

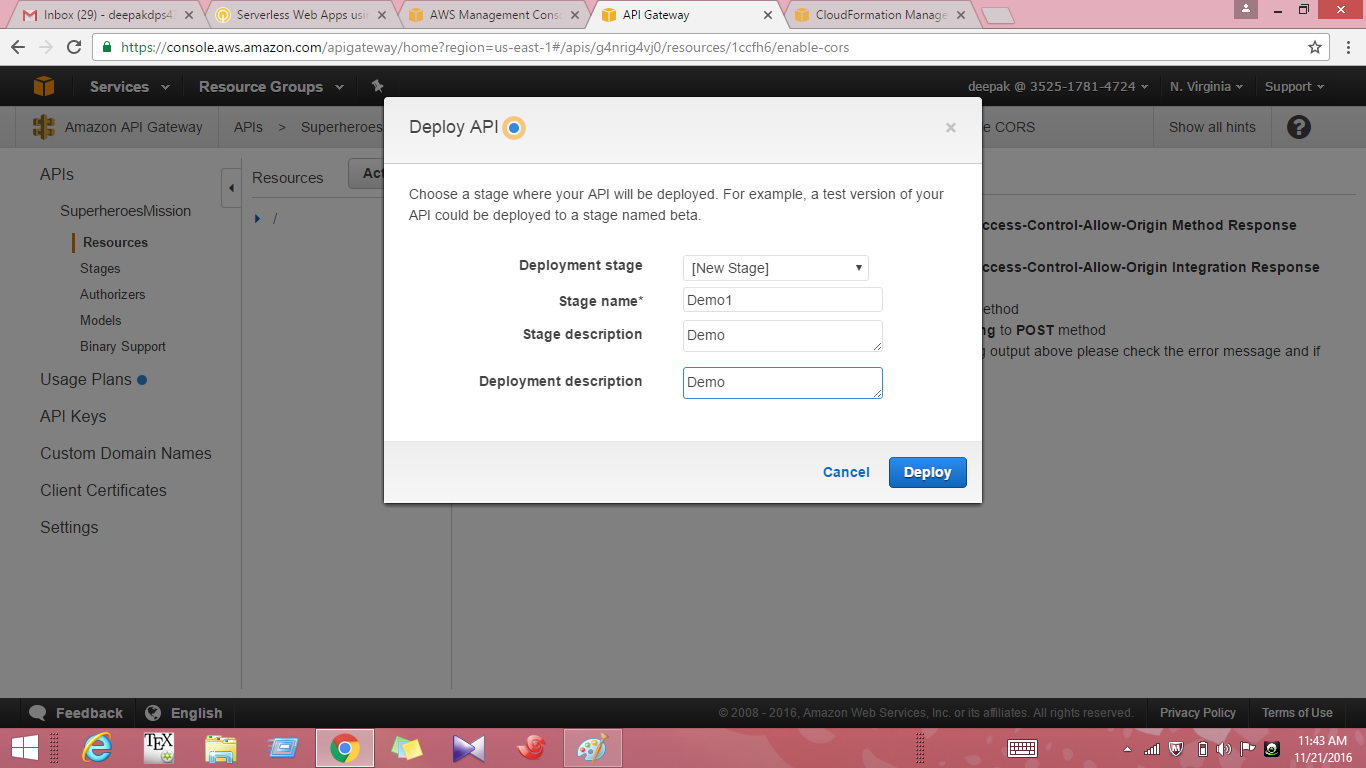
**38. Use the same process to enable CORS, using exactly the same settings, for the getmissiondetails resource.**

**39. Once you have enabled CORS for all methods, click the root folder, /.**

**40. Click Actions, then click Deploy API.**

**41. For Deployment stage, select New Stage.**

**42. For Stage Name, type Demo] ri**

****

**43. (Optional) For Stage Description, type Demc MI**

**44. (Optional) For Deployment Description, type Demc**

**45. Click Deploy. Congratulations, you have dep oyed the API!**

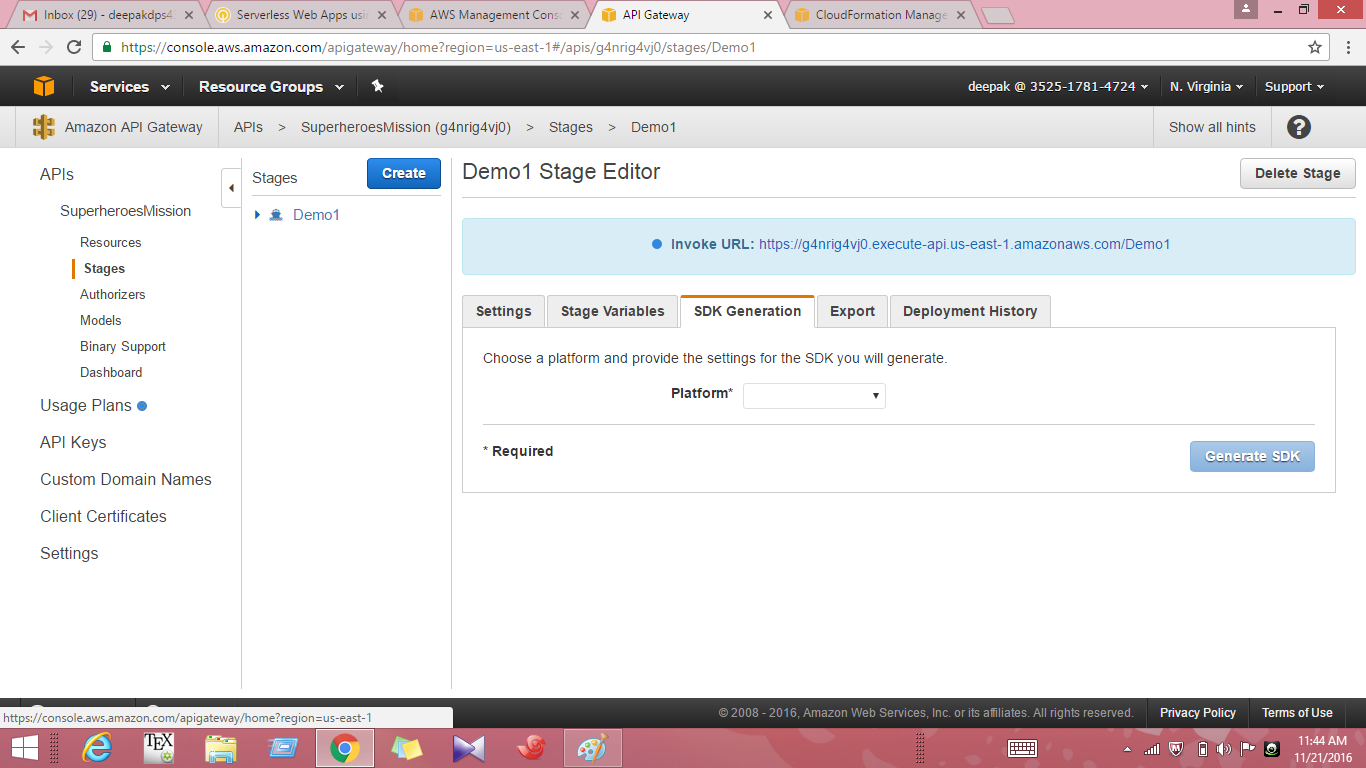
**Generate the SDK for your API**

**In the next part of the lab, you will generate the SDK for your API.**

**46. After deploying the API, you will be redirected to the Demol Stage Editor.**

**47. On the stage editor, click the SDK Generation tab.**

**48. For Platform, click Javascript.**

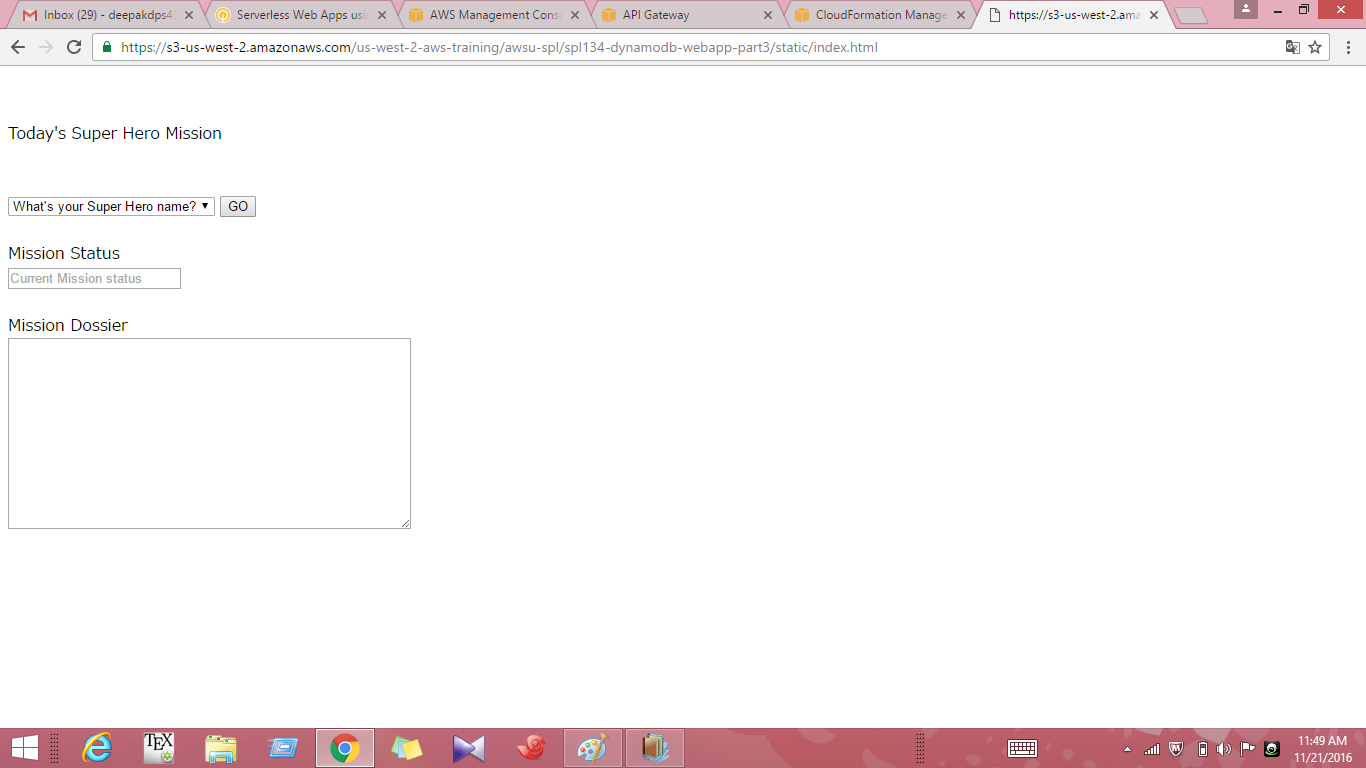
****

**49. Click Generate SDK. You will automatically download a ZIP file.**

**50. Open the directory (likely your Downloads folder, depending on your browser settings) and extract the contents of the ZIP file to your local computer.**

**51. Next, you will retrieve an HTML page that you will use to test your API. Go to https://s3-us-west-2.amazonaws.com/us-west-2-aws-training/awsu-spl/spl 34-dynamodb-webapp-part3/static/index.html**

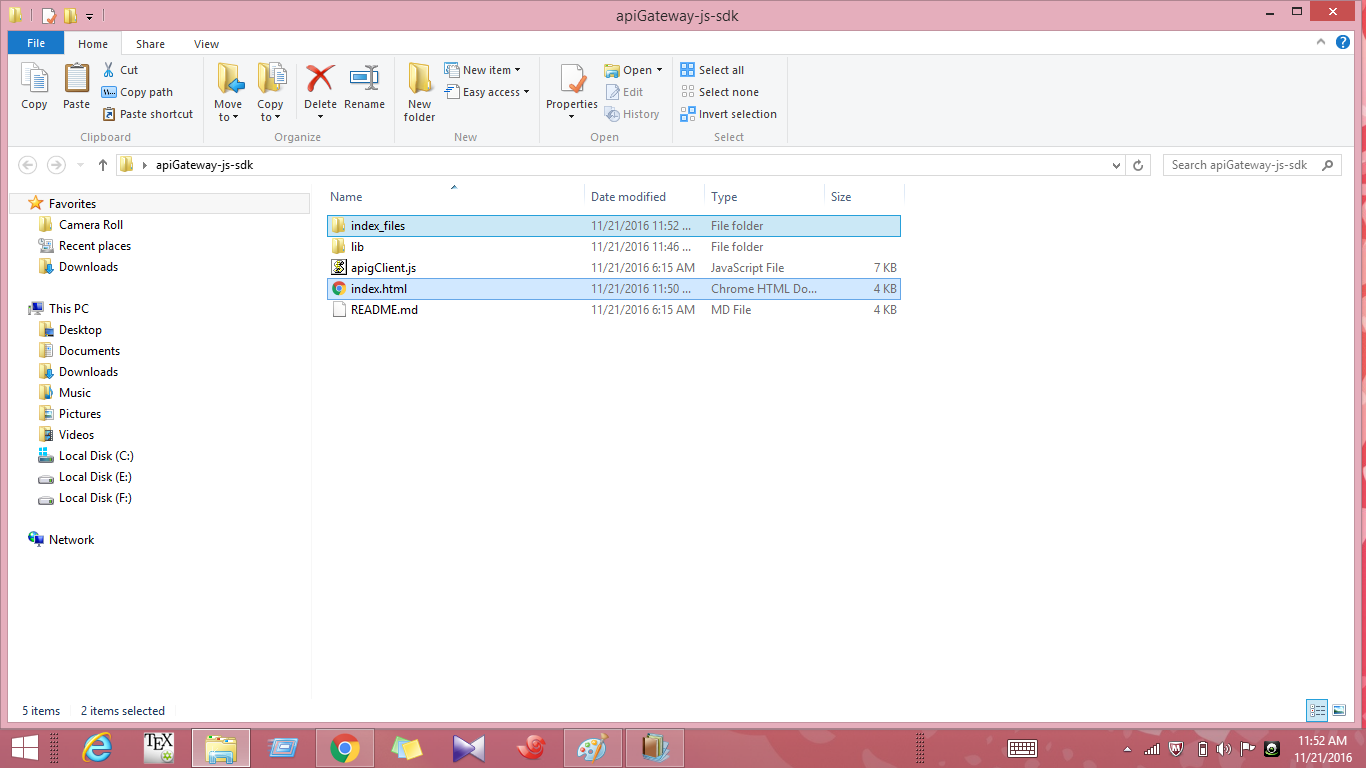
**52. In your browser, type Ctrl + S on your keyboard (Windows) or Command + S (Mac) to save the web page as HTML.**

****

**53. Save the HTML file as index.html.**

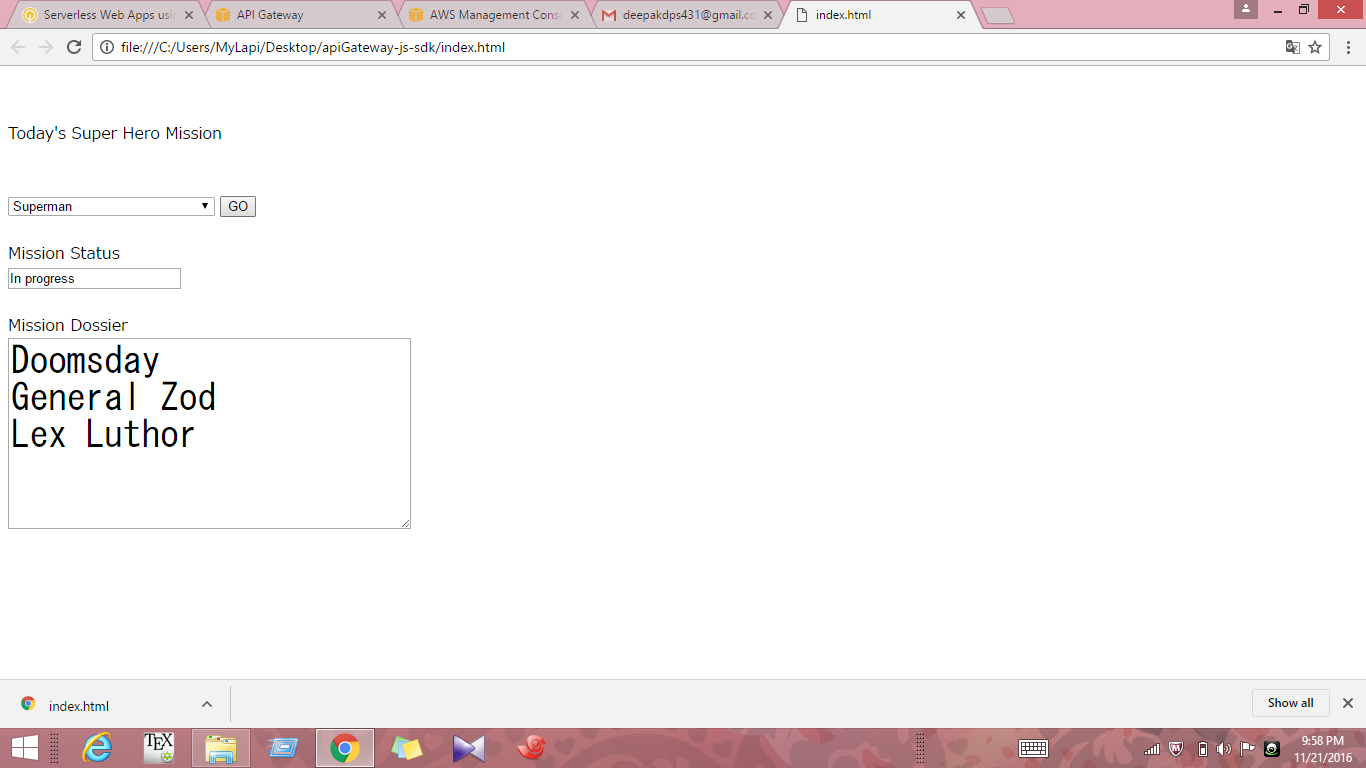
**Important When you save the index.html file, make sure you save it as a web page, "HTML only". You only need the HTML of this page.**

**54. On your local computer, copy the index.html file and paste it into the extracted apiGateway-js-sdk folder. Recall that you downloaded the ZIP file and extracted the contents to your local computer in previous steps.**

****

**55. Open index.html in your browser.**

**56. Using the index web page, retrieve mission details. Review the output.**

****

**57. On the Services menu, click 53.**

**58. Click Create Bucket.**

**59. Type a bucket name. Note that bucket names must be unique. Make a note of your bucket name, as you will need it later.**

**60. For Region, click US Standard.**

**61. Click Create.**

**62. Click your bucket to open It, tffeTFclick the Properties tab (at the top right).**

**63. In the Permissions section, click Add Bucket Policy.**

**64. Copy the following bucket policy and paste it into the editor:**

**{**

**"Version": "2012-10-17",**

**"Statement": [**

**{"Sid": "PublicReadForGetBucketObjects",**

**"Effect": "Allow",**

**"Principal": "\*",**

**"Action": "s3:GetObject",**

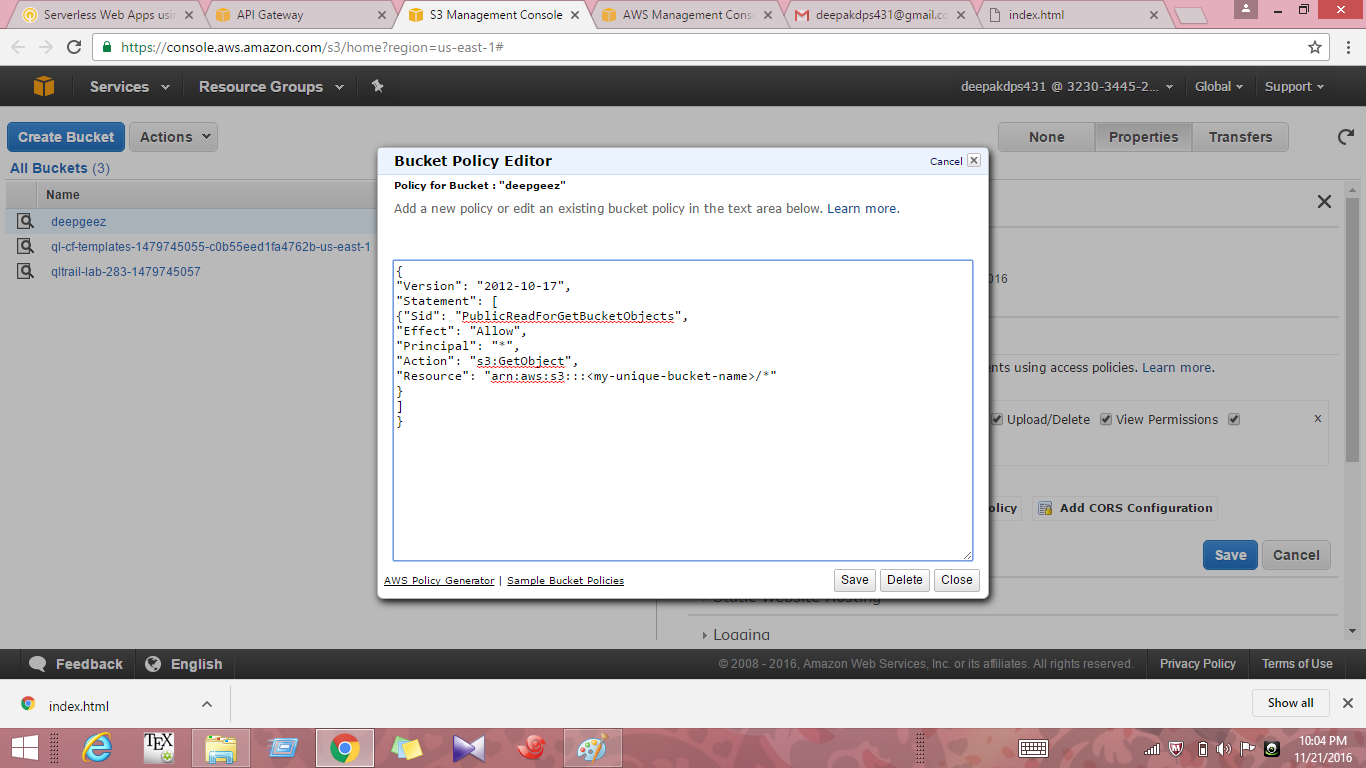
**"Resource": "arn:aws:s3:::<my-unique-bucket-name>/\*"**

**}**

**]**

**}**

**65. Replace <my-unique-bucket-name> with the name of your bucket.**

****

**66. Click Save.**

**67. In the Static Website Hosting section, click Enable Website Hosting.**

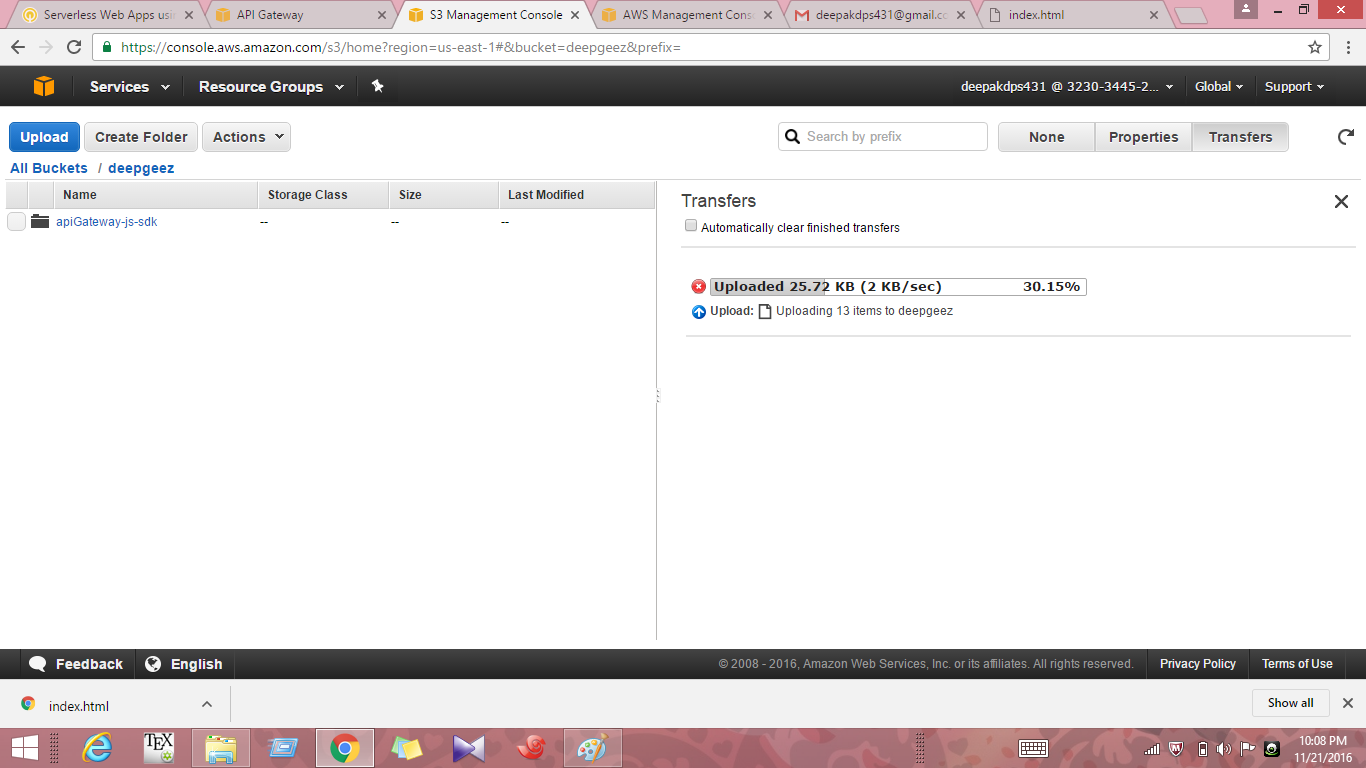
**68. For Index Document, type index.html**

**69. Click Save.**

**70. Click Upload (at the top left).**

**71. Open the folder where you saved the Javascript SDK.**

**72. Select the apiGateway-js-sdk folder.**

****

**73. Drag and drop the whole folder into the Console target area (you will see a section labeled "Drag and drop files and folders to upload here").**

**74. Click Start Upload. Wait until the upload is complete. This may take a minute or two.**

**75. Open a new browser tab/window.**

**76. Navigate to the following URL, replacing <unique-bucket-name> with the name of your S3 bucket.**

**77. You can now look up mission dossier data stored in your DynamoDB database.**

**RESULT**

**Created the API to fetch data from dynamodb table via API gateway**

**Generated a SDK for the created API and used it**

**Published the sample application using s3**

**A highly available and fault toleant application was designed as lambda automatically scales and executes the code when needed also there is no need to provision or manage servers in backend so, the tolerance is enhanced.**