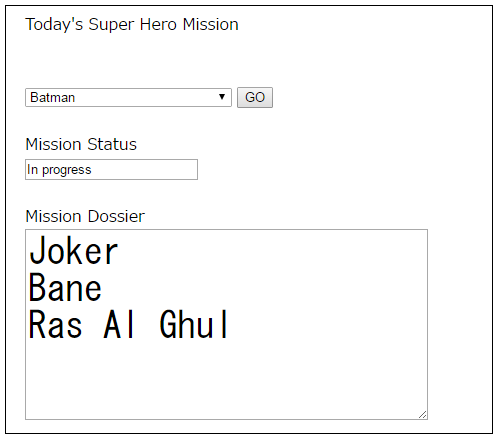
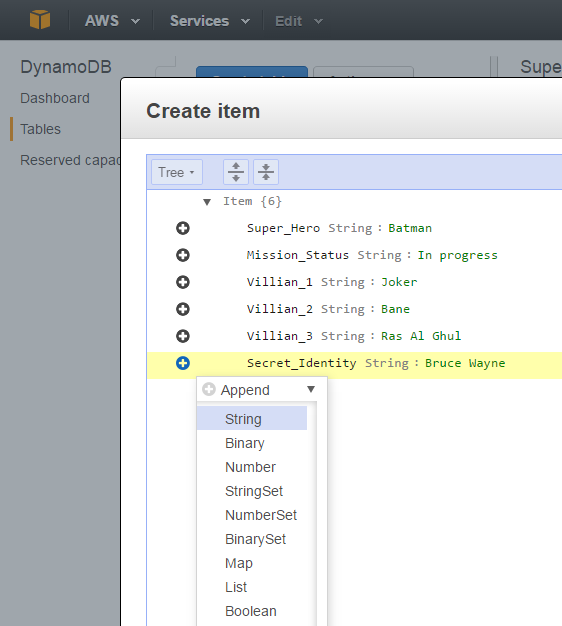
Building Serverless Web Applications using Amazon DynamoDB

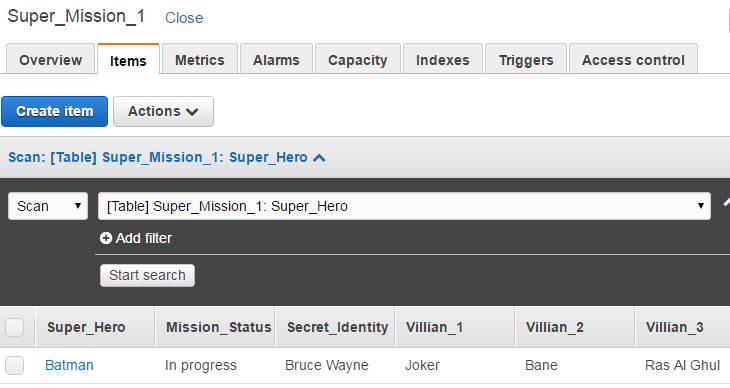
1. [INFO] The web application uses a simple HTML/javascript web interface. It will ask you for your Super Hero name that you can select from a drop-down list, and show you the Mission Dossier.



1. Log on to the aws.amazon.com console.
2. On the top right hand corner, select the region drop-down, and select “US East (N Virginia)”. This is the AWS region where all your AWS resources will be hosted. Note that that “US East (N Virginia)” is also known as “us-east-1”. So we will be using these two names interchangeably.
3. On the Top Left corner, select Services. Select DynamoDB. This is the DynamoDB console. You are now ready to start creating a table.
4. From the left side navigation bar, choose “Tables”. On the right hand navigation pane, select “Create Table”. For “Table Name” enter “Super\_Mission”. For “Primary Key” enter “Super\_Hero” and ensure data type drop down is set to “String”. Review the details for “Table Setting”, choose the “Default settings”. Click Create. You have now created the DynamoDB table that will be used in this demo.
5. Now we will populate the “Super\_Mission” table with sample data. On the right side navigation pane, select the tab named “Items”. Click “Create Item”. Ensure “Tree” is selected at the top left side, and enter data in the text box which displays “value”. All the data entered in this lab will use the “String data type”. Use the “+”, “Append”, “String” selections to insert additional attributes (columns). Look at the screenshot below to see what data will be entered. Ensure you are using the same attribute names including the case. When you have entered all the values, click “Save”.



1. Let’s review the data in the DynamoDB table, which looks like the screenshot below. There is one item in this table. The attribute “Super Hero” is the data that will be displayed in the drop-down on the web application. When you select a value, say “Batman” the backend logic will query the table to retrieve other attributes from the item such as “Mission\_Status”, “Villain\_1”, and so on. Notice the attribute named “Secret\_Identity” which shows Batman’s real name is “Bruce Wayne”. We will develop the set of permissions with “fine grained access controls” that will deny access to this attribute.



1. Let’s add additional rows. To enter the second row, click “Create Item” and on the top left corner select “Text”. Type the following JSON representation to the console and click Save.

{

"Super\_Hero": "Superman",

"Villian\_1": "Doomsday",

"Villian\_2": "General Zod",

"Villian\_3": "Lex Luthor",

"Mission\_Status": "In progress",

"Secret\_Identity": "Clark Kent"

}

1. Repeat the above step to enter the third and the fourth item using the following two documents.

{

"Super\_Hero": "The Winchester Brothers",

"Villian\_1": "Vampires",

"Villian\_2": "Ghosts",

"Villian\_3": "Werewolves",

"Mission\_Status": "Complete",

"Secret\_Identity": "Sam and Dean"

}

{

"Super\_Hero": "Iron Man",

"Villian\_1": "Apocalypse",

"Villian\_2": "Doctor Doom",

"Villian\_3": "LOki",

"Mission\_Status": "In progress",

"Secret\_Identity": "Tony Stark"

}

1. Review the contents in the “Items” tab. These are the four Items you entered in the DynamoDB table. You have now learnt how to enter data to DynamoDB table using the AWS console using two different formats.
2. Go to Overview tab. Under “Table Details” look for the value for “Amazon Resource Name (ARN)”. Save this value somewhere, example in a notepad. This is the ARN for your DynamoDB table. You will use this value in the next few steps.
3. We will now create the AWS policies that are required to access the DynamoDB data. Go to the IAM console. Let’s first create a policy that defines the permissions to scan the underlying DynamoDB table . This policy will be used to get the list of all super heroes from the table to populate the drop-down. On the AWS console, go to services > IAM. On the left side navigation Click Policies. On the right side, click Create New Policy. Select the option “Create your Own Policy”. Type the following in the next page. Policy Name = “Super\_DynamoDB\_Scan\_Policy”. Description = “Demo”. For the Policy Document, paste the following lines. In the place holder where the policy reads “<INSERT YOUR ARN>” paste the DynamoDB ARN you noted down from the earlier steps. Click Validate Policy and verify there are no errors. Click Create Policy.

{

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

"Statement": [

{

"Effect": "Allow",

"Action": [

"dynamodb:Scan"

],

"Resource": [

"<INSERT YOUR ARN>"

]

}

]

}

1. [INFO] Next we will create the IAM roles and attach the policy to the role. An IAM role is an AWS identity that can be assumed by a user or an application to access AWS resources using policy attached to the role.
2. Now we will create the role that will use the policy you created. You are already on the Policies page. Select the “Roles” on the left hand navigation to start creating a role. Select “Create a new role”. For the Role Name, type “Super\_DynamoDB\_Scan\_Role”. Click Next Step. Under “Select Role Type” Choose “AWS Service Roles” and select “AWS Lambda”. Under “Attach Policy” search for the policy named “Super\_DynamoDB\_Scan\_Policy” in the search bar. Select the Policy and click “Next Step”. Click “Create Role” to create the role.
3. We will now create the second Policy and Role using the same steps as above. This policy will be used to query the mission details for each super hero. We will also specify the conditions (fine grained access controls) in the policy that will deny access to query the “Secret\_Identity” attribute. Go to Policies > Create New Policy. Select “Create your own Policy”. Type Policy Name = “Super\_DynamoDB\_Query\_Policy”. Description = Demo. Type the following for the Policy Document. Replace the text <INSERT YOUR ARN> with the DynamoDB ARN you created above. Notice how the “Conditions” does not specify the attribute named “Secret\_Identify” thus denying access to it. Click “Validate Policy”. Click Create Policy.

{

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

"Statement": [

{

"Effect": "Allow",

"Action": [

"dynamodb:Query"

],

"Resource": [

"<INSERT YOUR ARN>"

],

"Condition": {

"ForAllValues:StringEquals": {

"dynamodb:Attributes": [

"Super\_Hero",

"Mission\_Status",

"Villian\_1",

"Villian\_2",

"Villian\_3"

]

},

"StringEquals": {

"dynamodb:Select": "SPECIFIC\_ATTRIBUTES"

}

}

}

]

}

1. Now we will create the role that will use the policy you created. You are already on the Policies page. Select the “Roles” on the left hand navigation to start creating a role. Select “Create a new role”. For the Role Name, type “Super\_DynamoDB\_Query\_Role”. Click Next Step. Under “Select Role Type” Choose “AWS Service Roles” and select “AWS Lambda”. Under “Attach Policy” search for the policy named “Super\_DynamoDB\_Query\_Policy” in the search bar. Select the Policy and click “Next Step”. Click “Create Role” to create the role.
2. We will create two Lambda functions in the next few steps, associate the roles we created above, and validate the results. On the top left corner, select “Services” and choose “Lambda”. Verify that you are in the same region as where you created the DynamoDB table. (US East N.Virginia).
3. The first Lambda function will retrieve a list of Super Heroes that are stored in the “Super\_Mission” table. Choose “Create a Lambda Function”, select “Skip” in the Select Blueprint screen. On the next screen, type Name = “get\_heroes\_list”, Description = “Demo”, Runtime = “Node.js”. For the Lambda function code, type the following code provided below. Select Role = “Super\_DynamoDB\_Scan\_Role”, and Click Next. Click Create function.

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

var dynamo = new doc.DynamoDB();

exports.handler = function(event, context) {

var getParams = {

TableName:'Super\_Mission'

};

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

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

else {

context.succeed(data);

}

});

};

1. To test this lambda function, choose Actions > Configure Test Event. Click Save and Test. You should see a “Execution Result: succeeded”. And below that, there will be a text box displaying all the contents from your DynamoDB table.
2. The second Lambda function will retrieve the mission details for a super hero from the “Super\_Mission” table. Choose “Create a Lambda Function”, select “Skip” in the Select Blueprint screen. On the next screen, type Name = “get\_mission\_details”, Description = “Demo”, Runtime = “Node.js”. For the Lambda function code, type the following code provided below. Select Role = “Super\_DynamoDB\_Query\_Role”, and Click Next. Click Create function.

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

var dynamo = new doc.DynamoDB();

exports.handler = function(event, context) {

condition = {};

condition["Super\_Hero"] = {

ComparisonOperator: 'EQ',

AttributeValueList:[{S: event.superhero}]

}

var getParams = {

TableName:'Super\_Mission\_1',

ProjectionExpression:"Super\_Hero, Mission\_Status, Villian\_1, Villian\_2, Villian\_3",

KeyConditions: condition

};

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

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

else {

context.succeed(data);

}

});

};

1. To test this lambda function, choose Actions > Configure Test Event. Paste the code provided bellow, and Click Save and Test. You should see a “Execution Result: succeeded”. And below that, there will be a text box displaying the mission details for “Batman”.

{

"superhero": "Batman"

}

1. Now, we will create a policy to allow the application to invoke Lambda functions. Go to Services > IAM. Select Policies from the left navigation. Select “Create Policy”. Select “Create Your Own Policy”. Policy Name = “Super\_Lambda\_Policy”. Description = “Demo”. For the Policy Document, type the code from below. Click Validate Policy to verify the policy. Click Create Policy.

{

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

"Statement": [

{

"Effect": "Allow",

"Resource": [

"\*"

],

"Action": [

"lambda:InvokeFunction"

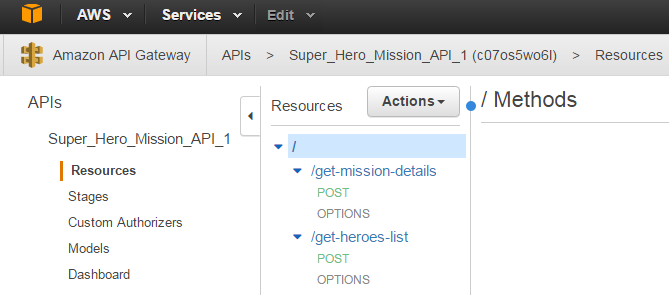
]

}

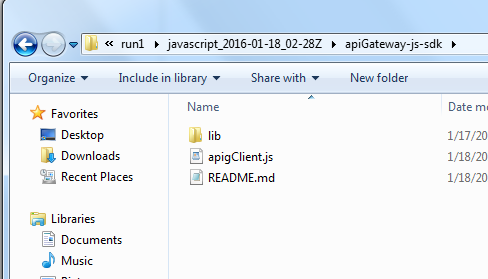
]

}

1. Now we will create Role and attach the policy to it. To create a role, select “Roles” from the left navigation. Select “Create New Role”. Role Name = “Super\_Lambda\_Role”. In “Select Role Type” page, select “AWS Service Roles”, and select “Amazon API Gateway”. Click Next, and click Create Role. Search for the role you created in this step, that is “Super\_Lambda\_Role” and click on it. In the “Permissions” tab, click “Attach Policy”, and in the search bar select the policy named “Super\_Lambda\_Policy”, and select “Attach Policy”. You have now successfully created a role that can be used by API Gateway to invoke Lambda functions.
2. On the Top left corner, select “Services” and select “API Gateway”. For “API Name” type “Superheroes\_Mission”. Description = “Demo”. Click Create API. You will see “Resources” tab on the left. Under the “Resources” tab, you will see the root identified by a “/”.
3. Select the root “/”, choose Actions, and click “Create Resource”. Type Resource Name = “get-heroes-list”, and ensure Resource Path is set to “get-heroes-list”. Click Create Resource. Choose Actions > Click “Create Method”. In the drop down, select POST and choose the “tick” mark next to it. Select Integration Type = “Lambda”, Lambda Region = “us-east-1” and Lambda Function = “get\_heroes\_list”. Click Save. In the pop-up window titled “Add Permission to Lambda Function”. Click OK.
4. Select the root “/”. Repeat the above step to create resource named “get-mission-details” and integrate it with the REST end point of the Lambda function “get\_mission\_details”.
5. The final API resources should look like the screenshot below.



1. Now we will enable CORS (Cross Origin resource sharing) for the methods we created in the API gateway. Click on the resource named “get-heroes-list”. Click “Enable CORS”. On the pop-up Window titled “Confirm Method Changes”, click “Yes”. Ensure the resulting page shows a success green tick mark for each step.
2. Repeat this for the resource “get-mission-details”
3. Now you are ready to deploy the API and download the SDK. On the Resources tab, select the root “/”. Click “Deploy API”. On the pop-up window, select Deployment Stage = “New Stage”, Stage Name = “Demo1”, Stage Description = “Demo”, Deployment Description = “Demo”. Click Deploy. On the Settings tab, don’t make any changes. Go to the “SDK Generation Tab”. Select platform = “JavaScript”. And click “Generate SDK”. This will download a “.zip” file to your local machine’s downloads folder.
4. Move the downloaded Javascript SDK to a convenient location on your laptop, and extract the contents. Verify that the content in your directory look like the screenshot provided below.



1. Copy the index.html file provided to you in this directory. Using a browser, open the index.html page. Now, select a Super\_Hero name, and retrieve the mission details and review the output.
2. Now we will host this website in an S3 bucket so that it can be accessed from anywhere. Go to Services > S3. Click Create Bucket. Note that bucket names have to be unique. Click Create Bucket. Type Bucket Name: “my-unique-bucket-name”, Region = “US Standard”. We will be uploading the newly created website to this bucket.
3. Let’s configure the bucket for website hosting. Click on the bucket you just created. Select “Properties” from the top right side. Under “Permissions”, Click Add bucket Policy. And type the following policy in the editor. Ensure you replace the text < my-unique-bucket-name> with the name of your bucket. Click Close.

{

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

"Statement": [

{

"Sid": "PublicReadForGetBucketObjects",

"Effect": "Allow",

"Principal": "\*",

"Action": "s3:GetObject",

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

}

]

}

1. ON the same page, choose “Static Website Hosting”, select “Enable website hosting” and type Index Document = “index.html”. Click Save.
2. On the left side, Select Upload. Navigate to the location where the Javascript SDK is saved and select the folder named “apiGateway-js-sdk”, and drag and drop it to the console where it says “Drag and drop files and folders to upload here”. Wait for the upload to complete.
3. Open a browser and navigate to the following link to access your website:

http://<my-unique-bucket-name>.s3-website-us-east-1.amazonaws.com/apiGateway-js-sdk/index.html

You have now created an S3 bucket, configured it for website hosting, and successfully uploaded the contents of your website to the S3 bucket.

1. You have now successfully created a website that can be accessed over the internet, and used AWS Lambda backends to query the data from Amazon DynamoDB.