AWS API Gateway How-To

10/28/2016

Yammering to Start

- The goal of this is to create a functioning API with AWS API Gateway
- More involved than previous session, touching on more of AWS
- AWS Components:
  1) IAM
  2) DynamoDB
  3) Lambda
  4) API Gateway
- Note: names are important

Part I: IAM

- Navigate to IAM > Roles
- Create a role called **`lambda_basic_execution`**
- Role Type (trusted service) is API Gateway, we'll add Lambda later
- Add **`lambda.amazonaws.com`** to trusted services
- Attach manages policies;
  `AmazonDynamoDBFullAccess` and **`AWSLambdaExecute`**
- Note this is not an IAM demo!

Part II: DynamoDB

- Navigate to DynamoDB press **`Create Table`**
- Call the table **`urly`**
- Make the Primary key **`url_name`**
- Smash **`Create`** button

Part III: Lambda

- Navigate to Lambda
- Get the code for shorty.py in your cut-n-paste buffer: http://tinyurl.com/shortypy
- Create a new blank Lambda function with no triggers
  - Name: shorty
  - Runtime: Python 2.7
  - Dump the code in your buffer into the code box
  - Handler: lambda_function.request_handler
  - Existing role: lambda_basic_execution
  - Next > Create Function

Part IV [0]: API Gateway

- Now the fun begins! If I go too fast launch something at me!
- Navigate to API Gateway and Get Started
- Create a new API called **`urly`**
- Create a resource called **`url`** and Enable API Gateway CORS
- Now select the OPTIONS method and we can look at the life cycle of a request before we continue.
- Request life cycle: Method Request > Integration Request > Work(our lambda) > Integration Response > Method Response

Part IV [1]: API Gateway

- Adding the **`url`** resource
- Now click on the **`url`** resource and use Actions to create a method (POST) hint: check the check mark
- Integration type: Lambda Function
- Lambda Region: e.g. us-west-2
- Lambda Function: shorty
- Strike Save and let it add the permission to allow execution

Part IV [2]: API Gateway

- Creating the model for a url
- Click models > Create
- Load the JSON schema from http://tinyurl.com/zdfmpwu into your cut-n-paste buffer
- Drop the schema into the box
- Model name: urlthing
- Content type: application/json
- Punch Create Model

Part IV [3]: API Gateway

- Configuring the Method Request
- This one is simple just leave everything as it is

Part IV [4]: API Gateway

- Configuring the Integration Request
- Activate “When there are no templates defined (recommended)”
- Add a mapping template application/json
- Save some time by generating a template from urlthing
- Set action to POST
- Set url_name to $inputRoot.url_name
- Set url to $inputRoot.url
- Strike Save
- If you care the template engine used above is http://velocity.apache.org

Part IV [5]: API Gateway

- Configuring the Method Response
- We need to indicate we are going to send back the HTTP header Access-Control-Allow-Origin, remember to check the check mark
- For Response Model we will just send back the JSON the Lambda generates

Part IV [6]: API Gateway

- Configuring the Integration Response
- We need to map a value we are going to send back in the Access-Control-Allow-Origin header: '*' again checking the check mark
- We leave the Body Mapping Templates alone

Part IV [7]: API Gateway

- Deploy the API
- Pull down Action > Deploy API
- Deployment Stage: create a new one or select one. I call mine v0
- Now the API is out in the magical cloud

Part IV [8]: API Gateway

- Testing the API
- In the Stages page click around until you find a url to which we can POST something
- Copy that url into your cut-n-paste buffer
- Visit http://static.mknote.us/url and paste your url into the API endpoint, populate URL and URL Name
- Press Record URL
- Message good is good, bad is bad and no message is terrible
- That maybe enough for today or we can continue ...

Part V [0]: Another Method

- Create a child resource of url called {url_name}
  - This is a path parameter
  - UI is busted so fix the Resource Path to be {url_name}
- No need to do the CORS thing since the only verb we will use is GET

Part V [1]: Another Method

- Implement GET on our {**`url_name`**} resource
- Create a GET method on the new child resource
- Integration type: Lambda
- Lambda Region: e.g. us-west-2
- Lambda Function: shorty (note shorty is happily serving more than one end-point)
- Depress the Save button and give API Gateway the permission to execute shorty, again

Part V [2]: Another Method

- Examine the Method Request on our resource
- Note that we have a new item on the Method Request page, Request Paths
- As before we don't need to change anything here

Part V [3]: Another Method

- Implement the Integration Request on our resource
- Activate “When there are no templates defined (recommended)”
- Add mapping template for Content-Type application/json
- Generate from the urlthing model
- Set action to GET
- Set url_name to $input.params('url_name')
- Remove the url element since that is what we are GETing
- Mash on the Save button

Part V [4]: Another Method

- Implement the Method Response on our resource
- Remember when I complain about AWS console being terrible? Behold what I am talking about in this step.
- Delete the 200 response
- Add 302
- Indicate we are sending back a Location header

Part V [5]: Another Method

- Implement the Integration Response on our resource
- Delete the 200 response with the (x)
- Add 302
- Add a Header Mapping for Location, integration.response.body.location
- We can look back at Lambda shorty and note that the read_entry() function returns a dict with the location element
- Note when we back out of the Integration Response the Method Response doesn't know about 302 yet, just refresh the page

Part V [6]: Another Method

- Deploy and test
- Deploy the API with the stage created/used earlier
- Take note of the GET url, copy it into a browser and add your url_name from the first test
- You should be redirected to the URL that was associated with the url_name
- Questions?