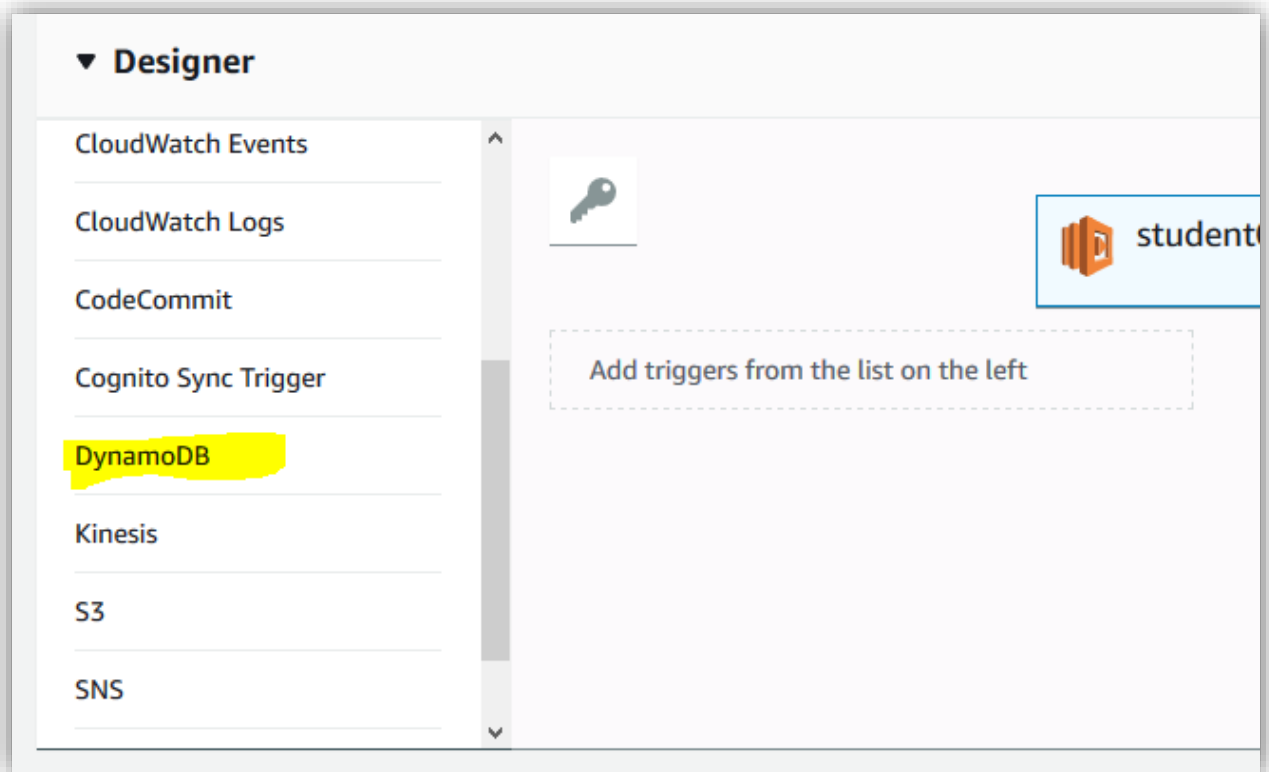
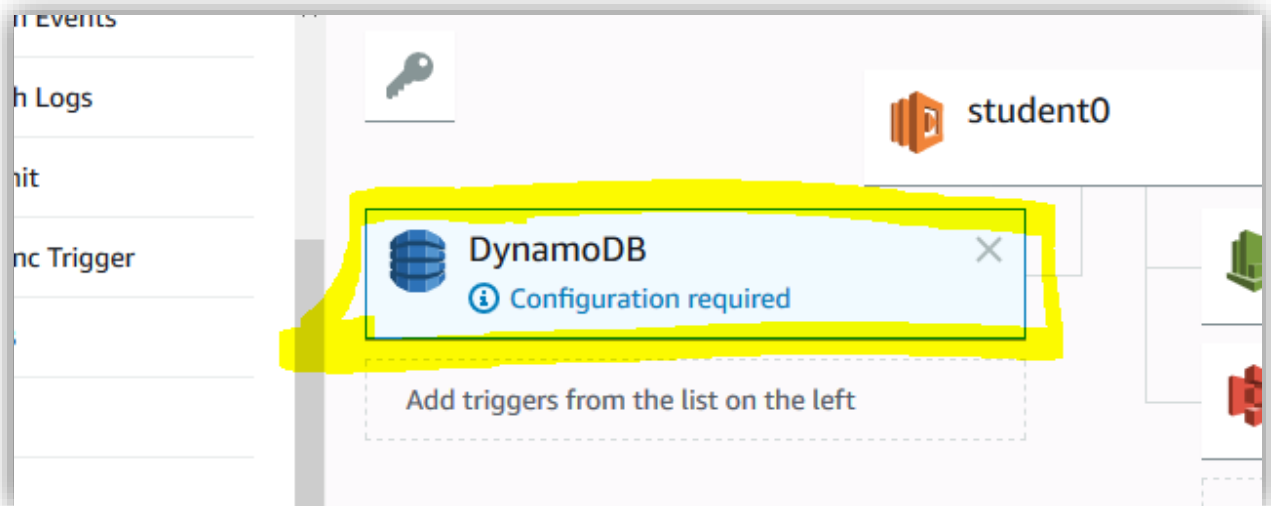


Lab 2: Connect your Lambda function to DynamoDB

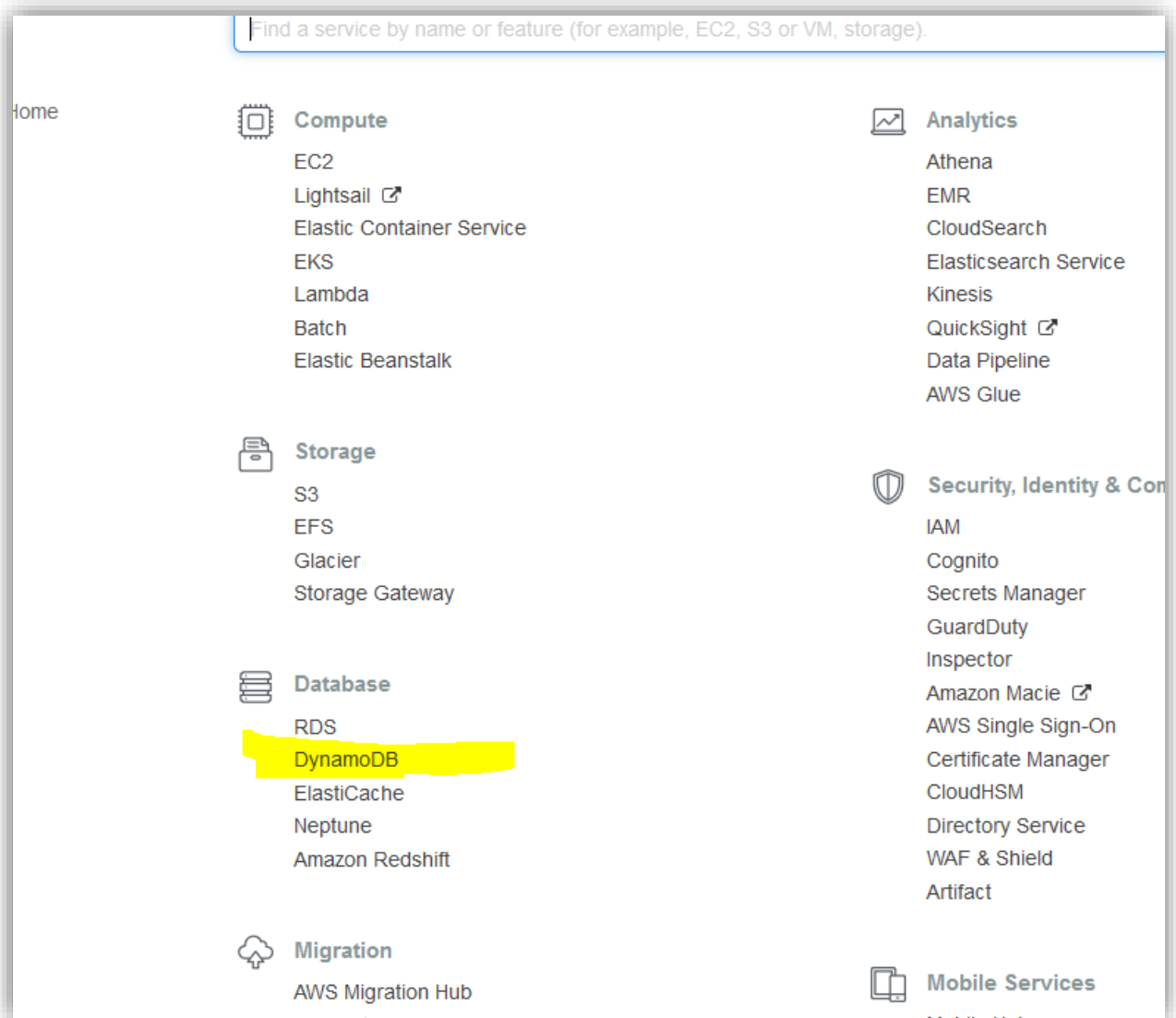
On the left hand side of the Lambda function designer scroll down and select the DyanmoDB trigger:



You will be prompted to configure DyanmoDB. Since this is a shared account tables exist already, however we want to have a table just for you:



Select DynamoDB from the services menu in the top left:



From the DyanmoDB pane, create a new table:

a fast and flexible NoSQL database service for all applica
latency at any scale. Its flexible data model and reliable p
r mobile, web, gaming, ad-tech, IoT, and many other appl

Create table

[Getting started guide](#)

Name your table student# and set your primary key to be "id" and create the table:

Create DynamoDB table

Tutorial ?

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name* ⓘ

Primary key* Partition key

String ⓘ

☐ Add sort key

Table settings

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

☒ Use default settings

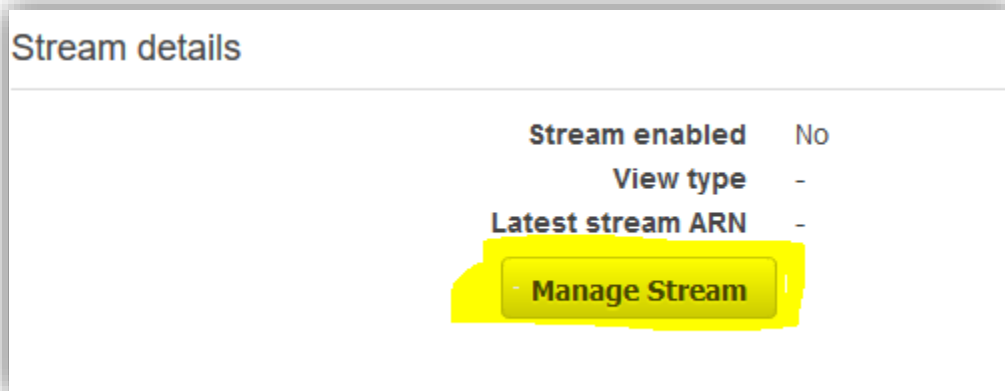
- No secondary indexes.
- Provisioned capacity set to 5 reads and 5 writes.
- Basic alarms with 80% upper threshold using SNS topic "dynamodb".
- On-Demand Backup and Restore Enabled **NEW!**

ⓘ You do not have the required role to enable Auto Scaling by default.
Please refer to [documentation](#).

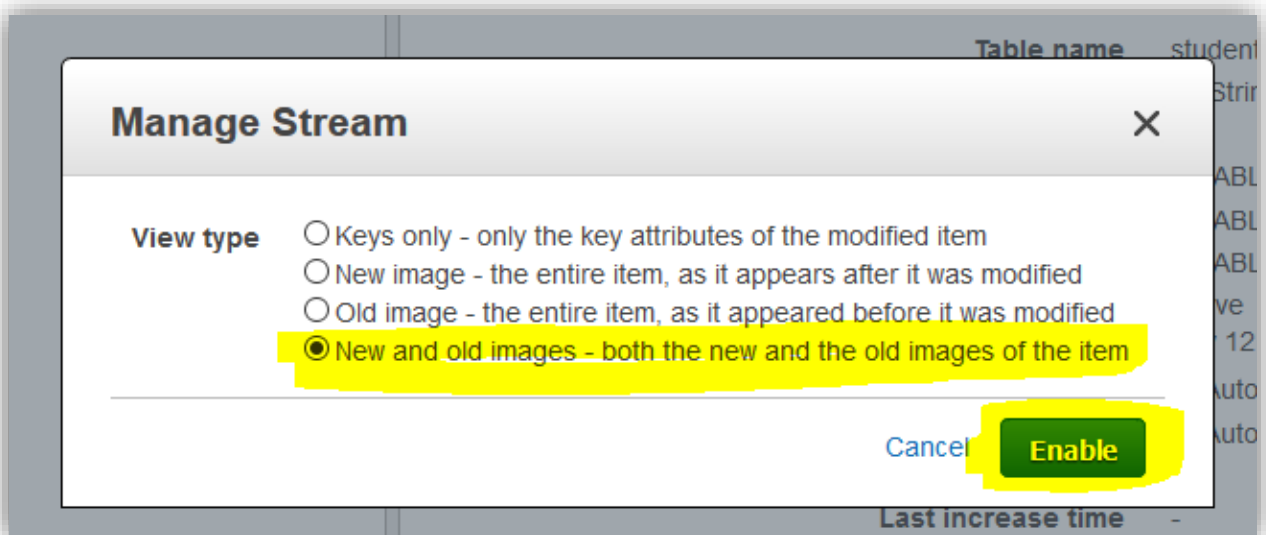
Additional charges may apply if you exceed the AWS Free Tier levels for CloudWatch or Simple Notification Service. Advanced alarm settings are available in the CloudWatch management console.

Cancel **Create**

Once you have created the table, you should have the option to configure a stream. Click on the manage stream button to configure the stream that will be used as the trigger from our Lambda function:



Be sure to select new and old images, this will send the old data and the new data in all stream triggers and then click Enable. This prevents you from having to make a query to determine anything about the data. This saves you execution time, and requests:



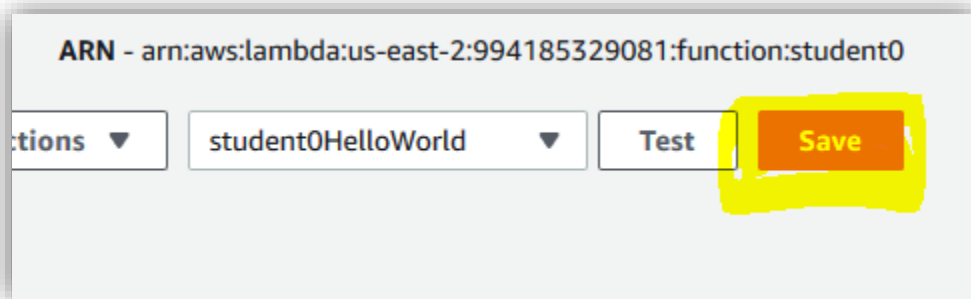
Back in the Lambda function designer, select your function and, and select the DyanmoDB trigger integration again. This time select the table matching your student#. Leave the rest of the defaults and click Add:

The screenshot shows the 'Configure triggers' dialog in the AWS Lambda console. On the left, a sidebar lists available triggers: Cognito Sync Trigger, DynamoDB, Kinesis, S3, SNS, and SQS. The 'DynamoDB' trigger is selected and highlighted in blue, with a 'Configuration required' icon. Below the sidebar, a dashed box prompts to 'Add triggers from the list on the left'. On the right, a list of resources includes 'Amazon CloudWatch Logs' and 'Amazon S3', with a dashed box below stating 'Resources the function's role has access to will be shown here'. The main 'Configure triggers' section contains the following fields:

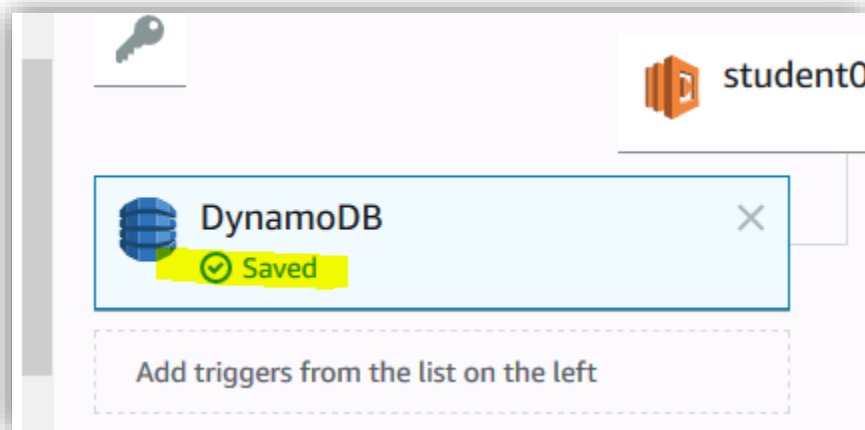
- DynamoDB table:** A dropdown menu with 'student0' selected.
- Batch size:** A numeric input field with '100' entered.
- Starting position:** A dropdown menu with 'Latest' selected.

Below these fields, a note states: 'In order to read from the DynamoDB trigger, your execution role must have proper permissions.' A checkbox labeled 'Enable trigger' is checked. Below the checkbox, a note reads: 'Enable the trigger now, or create it in a disabled state for testing (recommended).' At the bottom right, there are 'Cancel' and 'Add' buttons, with the 'Add' button highlighted in yellow.

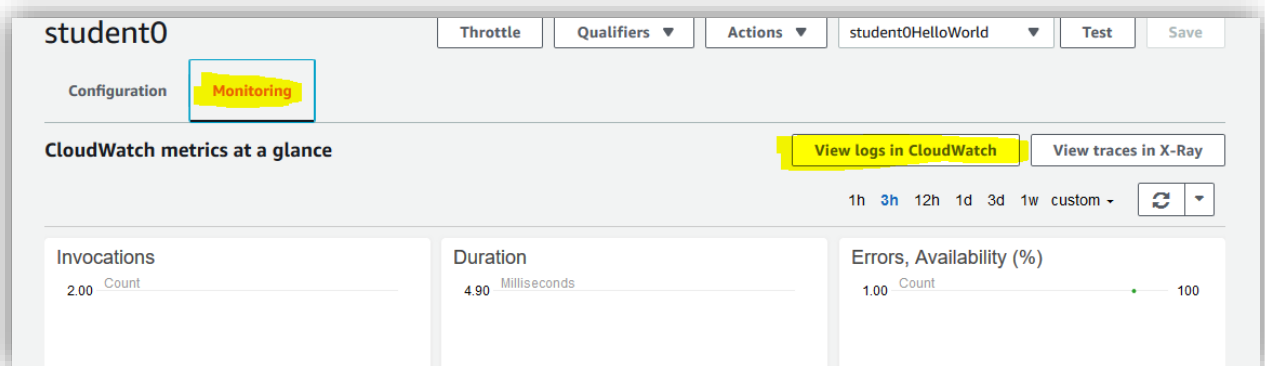
Your changes aren't saved yet, you will see below the DynamoDB trigger it says unsaved changes. Click on the Save button in the upper right hand corner to save the changes and enable the trigger:



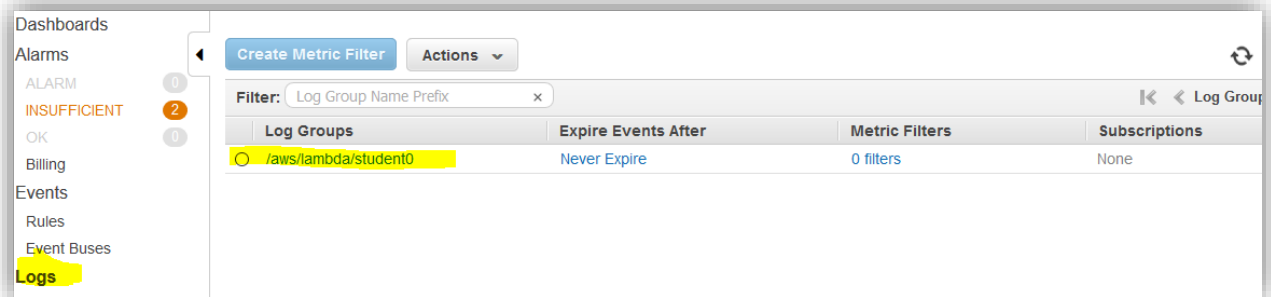
You should now see that your changes are saved:



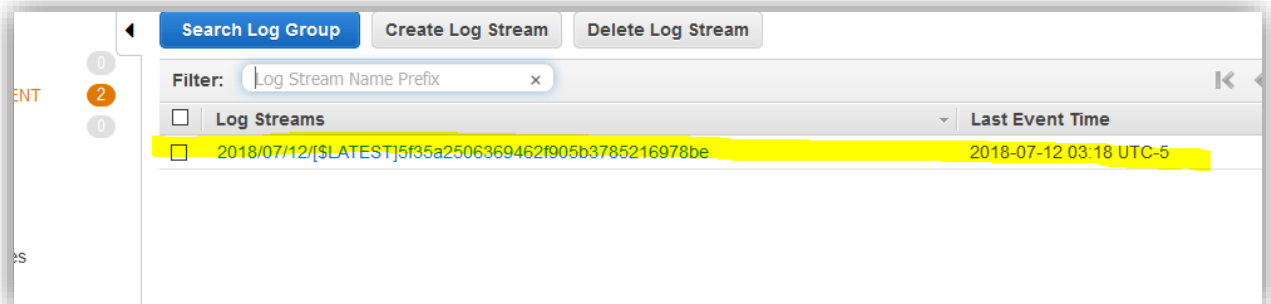
Since triggers have no viewable output we will need to use the CloudWatch logs to see what our function is actually doing. Click on the monitoring tab, and then select view logs in CloudWatch:



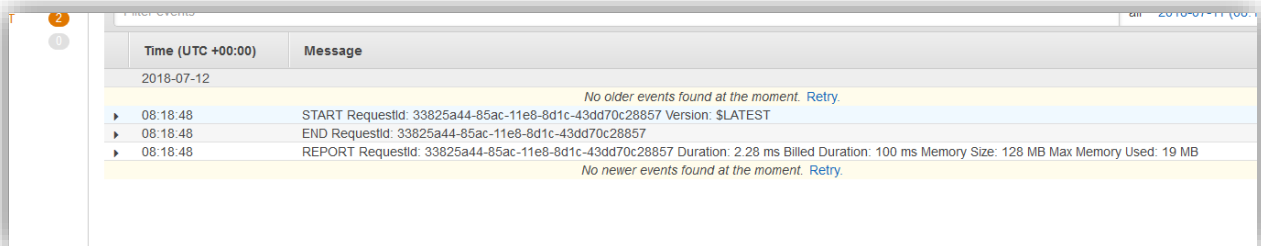
When we created the function, it automatically made a log group for the function. Select logs from the left hand side, and then click on the log group that matches your Lambda function name:



You will see a log stream for every execution, these are automatically time stamped and saved. Click on the latest stream you see:



This is the log stream from when we tested our service when we set it up, the hello world function doesn't have any logging so we are shown the basic data that all logs will have. Execution time and billing info:




Time (UTC +00:00)	Message
2018-07-12	No older events found at the moment. Retry
08:18:48	START RequestId: 33825a44-85ac-11e8-8d1c-43dd70c28857 Version: \$LATEST
08:18:48	END RequestId: 33825a44-85ac-11e8-8d1c-43dd70c28857
08:18:48	REPORT RequestId: 33825a44-85ac-11e8-8d1c-43dd70c28857 Duration: 2.28 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 19 MB
	No newer events found at the moment. Retry

Our Hello World function doesn't know about our new DynamoDB stream so let's replace it with something that simply reads the stream and outputs it to the logs so we can see them in CloudWatch. Go back to the Lambda function designer and replace the code for index.js with the following:

```
'use strict';
exports.handler = (event, context, callback) => {

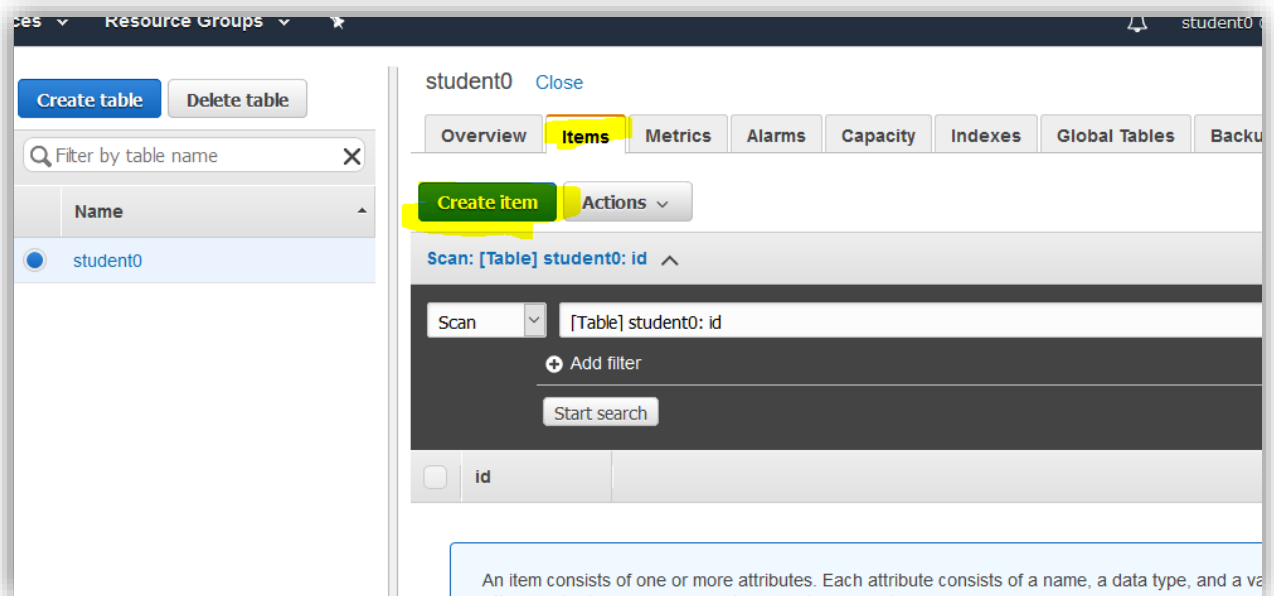
  event.Records.forEach((record) => {
    console.log('Stream record: ', JSON.stringify(record, null, 2));
  });
  callback(null, 'Successfully processed ${event.Records.length} records.');
```

Be careful, because this is a PDF you may have to fix quotes and such. Now save the function, and head back over to the DynamoDB services. Select tables, and select the table that matches your student#:

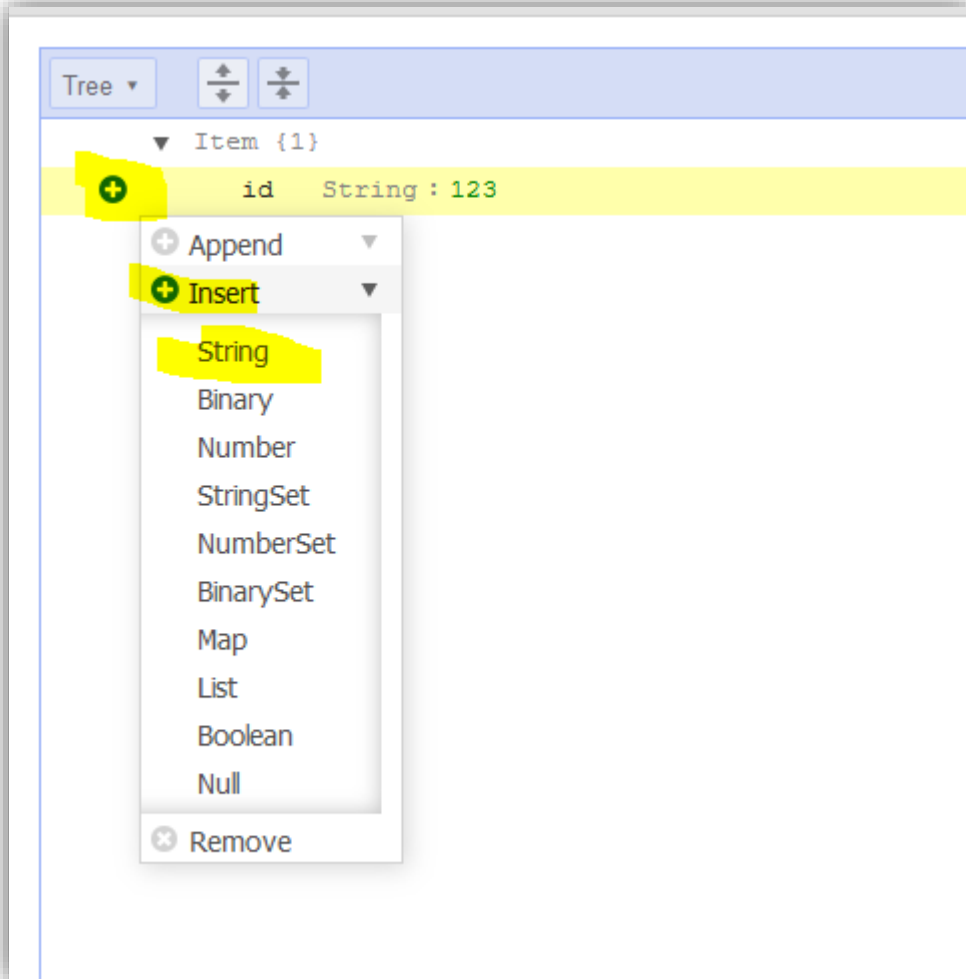


Name	Status	Partition key	Sort key	Indexes	Total read capacity	Total write capacity
student0	Active	id (String)	-	0	5	5

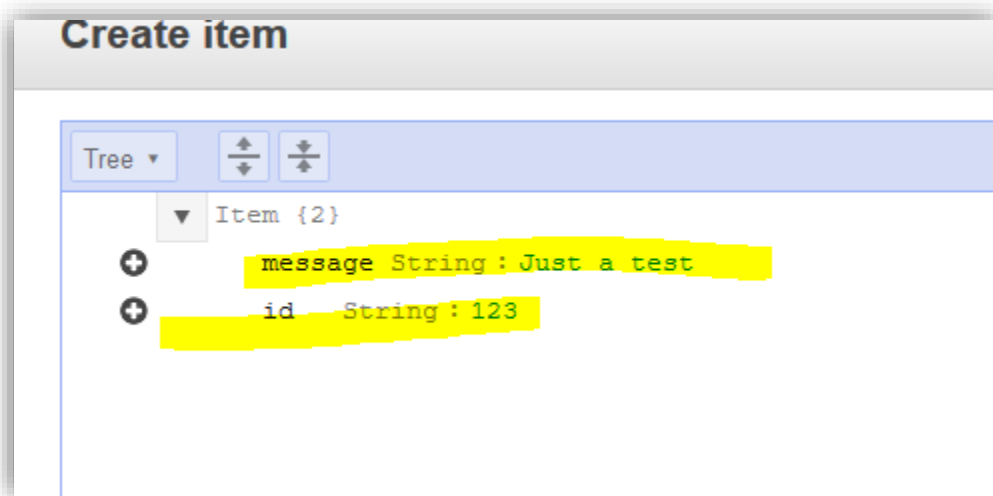
Create a new item in the table:



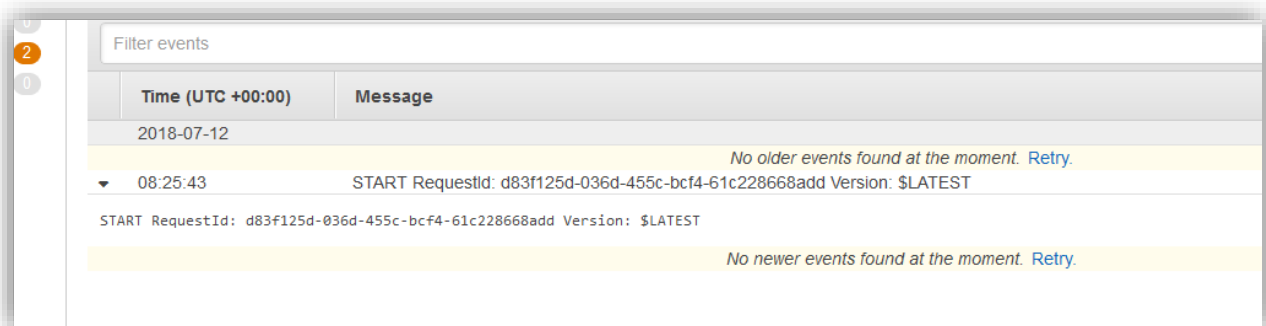
ID will be populated already, put any value you would like in there, and then insert a new string value:



You can name it whatever you want, and give it some data:



Click on create item and go look at your log stream in CloudWatch – if you were quick the container won't have finished running yet:



Refresh after a moment and you should see the full execution details:

Filter events		all 2018-07-11 (08:25:43) ▾
Time (UTC +00:00)	Message	
2018-07-12		
No older events found at the moment. Retry .		
▶ 08:25:43	START RequestId: d83f125d-036d-455c-bcf4-61c228668add Version: \$LATEST	
▼ 08:25:43	2018-07-12T08:25:43.221Z d83f125d-036d-455c-bcf4-61c228668add Stream record: { "eventID": "a2a39b13414e6fe7d183a802a1e00d7a", "eventName": "INSERT", "eventVers	
2018-07-12T08:25:43.221Z d83f125d-036d-455c-bcf4-61c228668add Stream record:		
{		
"eventID": "a2a39b13414e6fe7d183a802a1e00d7a",		
"eventName": "INSERT",		
"eventVersion": "1.1",		
"eventSource": "aws:dynamodb",		
"awsRegion": "us-east-2",		
"dynamodb": {		
"ApproximateCreationDateTime": 1531383900,		
"Keys": {		
"id": {		
"S": "123"		
}		
},		
"NewImage": {		
"id": {		
"S": "123"		
},		
"message": {		
"S": "Just a test"		
}		
},		
"SequenceNumber": "360000000006439269290",		
"SizeBytes": 28,		
"StreamViewType": "NEW_AND_OLD_IMAGES"		
},		
"eventSourceARN": "arn:aws:dynamodb:us-east-2:994185329081:table/student0/stream/2018-07-12T07:57:04.688"		
}		
▶ 08:25:43	END RequestId: d83f125d-036d-455c-bcf4-61c228668add	
▶ 08:25:43	REPORT RequestId: d83f125d-036d-455c-bcf4-61c228668add Duration: 38.00 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 19 MB	
No newer events found at the moment. Retry .		

Congratulations, you now have a Lambda function that is automatically executed any time the values in a DynamoDB table are changed.