Exercise: Secure Amazon DynamoDB using the AWS Software Development Kit (AWS SDK)

Overview

In this exercise, you will learn how to *develop* with Amazon DynamoDB by using the AWS Software Development Kit (AWS SDK). Following the scenario provided, you will learn how to apply the **least privilege principle** to access DynamoDB and communicate with DynamoDB without going over the internet. This exercise gives you hands-on experience with AWS Identity and Access Management (IAM) and Amazon Virtual Private Cloud (VPC) Gateway Endpoints.

Objectives

After completing this exercise, you will be able to:

- Create Fine-Grained Access Control IAM Policy Conditions for Amazon DynamoDB
- Move a Lambda function inside a VPC
- Create a VPC Gateway Endpoint for DynamoDB

Story continued

Steve has just notified you that sensitive data will be stored in the *dragon_stats* DynamoDB table. That sensitive data shouldn't be readable by the Lambda function, as he's afraid that a malicious developer could change the code to read that sensitive data. He reviewed some of the IAM Policies that are applied to the IAM Role, *call-dynamodb-role*, used by the Lambda function that runs your code and found out that they are too permissive.

He wants you to follow the least privilege principle mandated by the Security Team that only allows:

- specific actions executed by the application
- querying for specific attributes required by the application

As you know which action types and attributes are used in your application, he thought it would make more sense to leave this task to you. So you will need to read the code to find that data and create the most Fine-Grained IAM Policy you can without breaking the application.

He also wants you to move the Lambda function inside of a Virtual Private Cloud (VPC) that doesn't permit any access to the Internet. He mentioned that the reason for doing that is because another developer is working on a feature that will only be available inside the VPC. As you know, DynamoDB is available over the internet and that's what you have been using so far. So he asked you to find a

way for the Lambda function to communicate to DynamoDB from within the VPC without going over the internet. That's when you remembered about VPC Gateway Endpoints. It's a good time to implement this while the Lambda function isn't in production.

Prepare the exercise

Before you can start this exercise, you need to import some files and install some tools in the AWS Cloud9 environment that you have created.

1. From the AWS Management Console, go to the **Services** menu and choose **Cloud9**.
2. Choose **Open IDE** to open the AWS Cloud9 environment. Close down any tabs that you are not using and collapse any inactive folders.
3. Ensure you are in the base path in your AWS Cloud9 terminal by using this command:

`cd /home/ec2-user/environment`

1. Install the **jq** package by running the following command in the AWS Cloud9 **bash terminal**:

```
sudo yum install jq
# if it says "is this ok?", select y (for yes)
```

2. To get the files that will be used for this exercise, go to the Cloud9 **bash terminal** and run the following `wget` command:

```
wget \
https://s3.amazonaws.com/awsu-hosting/edx_dynamo/c9/dynamo-secure/lab6.zip \
-P /home/ec2-user/environment
```

   You should also see that a root folder called **dynamolab** with a `lab6.zip` file has been downloaded and added to your AWS Cloud9 filesystem (on the top left).

3. To unzip the lab6.zip file, run the following command:

```
cd ~/environment
unzip lab6.zip
```

   In your Cloud9 filesystem, you should see a lab6 folder.

4. To keep things clean, run the following commands to remove the zip:

```
rm lab6.zip
cd lab6
echo "done"
```

Step 1: Create a new IAM Policy and Role

The current IAM Role associated to the **DragonSearch** Lambda function is used by more than one function. To ensure that the other functions continue to work, you will create a specific IAM Role just for this Lambda function as it's a best practice to do so.

You will start with creating your own IAM Policy using generic statements using the Visual Policy Editor and look at the JSON code.

You will then create the new Role and associate your IAM Policy and the **AWSLambdaVPCAccessExecutionRole** AWS managed Policy. The *AWSLambdaVPCAccessExecutionRole* Policy will be used later in this exercise to allow Lambda to run inside your VPC.

Finally, you will modify the **DragonSearch** Lambda function to use your new Role and test it before moving to the next step.

Step 1A): Create new Policy

1. From **Cloud9**, click the **AWS Cloud9** button next to File.
2. Select **Go To Your Dashboard**.
3. Click on the **Services** menu and choose **IAM**.
4. Click on the **Policies** link from the left side menu.
5. Click on the **Create policy** button.
6. Click on the **Choose a service** link to expand it.
7. In the **Find a service** bar, input `DynamoDB`.
8. Click on the **DynamoDB** link to select it.
9. In the **Actions** section, click the checkbox next to **All DynamoDB actions**.
10. Click on the **Resources** section to expand it.
11. Select the radio button next to **All resources**.
12. Click on the **Review policy** button.
13. In the **Name** field, input `edx-ddb-dragonsearch-policy`.
14. Click on the **Create policy** button.

Step 1B): Create new Role

1. Click on the **Roles** link from the left side menu.
2. Click on the **Create role** button.
3. Click on the **Lambda** link under the **Choose the service that will use this role** section.
4. Click on the **Next: Permissions** button.
5. In the **Search** bar, input `edx-ddb-dragonsearch-policy`.
6. Click the checkbox next to **edx-ddb-dragonsearch-policy**.
7. In the **Search** bar, input `AWSLambdaVPCAccessExecutionRole`.
8. Click the checkbox next to **AWSLambdaVPCAccessExecutionRole**.
9. Click on the **Next: Tags** button.
10. Click on the **Next: Review** button.
11. In the **Role name** field, input `edx-ddb-dragonsearch-role`.
12. Click on the **Create role** button.

Step 1C): Associate the new Role to Lambda and Test

1. Click on the **Services** menu and choose **Lambda**.
2. Click on the **DragonSearch** lambda function link.
3. Scroll down to the **Basic Settings** card and click **Edit**.
4. Click on the **Existing role** dropdown.
5. In the search bar, input `edx-ddb-dragonsearch-role`.
6. Click on the **edx-ddb-dragonsearch-role** entry.
7. Click on the **Save** button.
8. In the dropdown next to the *Test* button, select **JustOneDragon**.
9. Click on the **Test** button.

The details output should be the following:

```
{
    "errorType": "string",
    "errorMessage": "not allowed",
    "trace": []
}
```

This is actually what you should expect, as we locked down access to this API to only allow access to logged in users.

First get a new valid session using the AWS Cloud9 terminal. **NOTE:** You may need to have the lab5 folder still in your Cloud9 IDE for this to work.

```
node ../lab5/solution/login.js test dave@dragoncardgame001.com apple
```

This should give you a new `session_id_str` and expose the `user_name_str`.

```
$2b$10$Q5.0VU2CA5JFnc0r6hSXfeNFpm2XoVXlKMVnniR7pPivIMb7wvoVy apple
Password is correct
{ ConsumedCapacity: { TableName: 'sessions', CapacityUnits: 1 } }
AWAITED 18b3dcf4-06b1-4d2e-98ad-a70f4e0b40ac
null { user_name_str: 'davey65',
  session_id_str: '18b3dcf4-06b1-4d2e-98ad-a70f4e0b40ac' }
```

Now, in the Lambda console for the `DragonSearch` function, press **configure test event** in the test dropdown.

Create a new test event called `withSession`

Replace the content in the <FMI> to use the `session_id_str` you just got.

```
{
  "user_name_str": "davey65",
  "session_id_str": "<FMI>",
  "dragon_name_str": "Dexler"
}
```

Click test and you should see the following:

```
[
  {
    "family": {
      "S": "green"
    },
    "damage": {
      "N": "4"
    },
    "description": {
      "S": "Dexler is a protector of the earth and forests. He is as green as the
earth and burrows into the ground for protection and extra defense."
    },
    "protection": {
      "N": "2"
    },
    "dragon_name": {
      "S": "Dexler"
    }
  }
]
```

Excellent!

You have now created a new policy and a new role and associated it to your Lambda function. By testing it and verifying the output of the test, you can now conclude that your new role and policy works. It's time to apply the least privilege principle to your new policy.

Step 2: Modify the IAM Policy to apply the Least Privilege Principle

Amazon DynamoDB doesn't work the same way as typical databases like the ones under the Amazon Relation Database Service (RDS) where you are still responsible for creating the Users and Permissions inside the database. With Amazon DynamoDB, AWS IAM is still used to control the authentication and authorization, thus liberating the database from this burden. This means that the IAM Policy associated with the authentication entity you are using must follow the least privilege principle. The same idea where you wouldn't use the Root/Master user for your database inside your

application because its permissions are too powerful, you shouldn't grant all access to all of DynamoDB or to your Table if your application doesn't require it.

In this task, you will first review the code of the **DragonSearch** Lambda function to determine what DynamoDB API calls/actions are required. You will also look for all the attributes that are returned so you can lock down the IAM Policy even more.

You will then modify the **edx-ddb-dragonsearch-policy** IAM Policy to apply Fine-Grained Access Controls on your Policy. To do this, IAM Policy Conditions will be used. You should review the documentation to ensure you understand their usage.

Finally, you will test the Lambda function to make sure the `withSession` test still works.

Step 2A): Review the code of DragonSearch

1. Click on the **Services** menu and choose **Lambda**.
2. Click on the **DragonSearch** Lambda function link.
3. Review the code of the **justThisDragon** and **scanTable** functions. Note all of the API calls made to the DynamoDB service. Also, take a note of all the attributes requested by those API calls. Finally, note the name of the DynamoDB table used for each of those calls.

Step 2B): Find the ARN of the DynamoDB table

You need to get the Amazon Resource Name (ARN) of the DynamoDB table as it will be used in the Policy.

1. Click on the **Services** menu and choose **DynamoDB**.
2. Click on the **Tables** link in the left menu.
3. Click on the name of the DynamoDB table that is used in the code. `dragon_stats`
4. At the bottom of the **Table details** section, you will find the **Amazon Resource Name (ARN)** field. Take a note of this ARN which should look like the following: `arn:aws:dynamodb:us-east-1:123456789012:table/dragon_stats`.

Step 2C): Modify your IAM Policy

1. Click on the **Services** menu and choose **IAM**.
2. Click on the **Policies** link in the left menu.
3. In the **Search** bar, input `edx-ddb-dragonsearch-policy`.
4. Click on the **edx-ddb-dragonsearch-policy** link.
5. Click on the **Edit policy** button.

   If you are familiar with IAM Policies and prefer to do this yourself, feel free to click on the **JSON** tab and edit the policy manually based on what you found in the IAM Policy Conditions documentation. You can skip to the next task to test your Lambda function.
6. Click on the **DynamoDB** section to expand it.

7. The Service is set to DynamoDB which is what you want it to be.
8. Click on the **Actions** section to expand it.
9. Remove the checkmark next to **All DynamoDB actions**.
10. Click on the breadcrumb next to **Read** as your Lambda function is only reading from DynamoDB.
11. Place a checkmark next to all of the API calls made to DynamoDB used in the code.
12. Select the two methods that your code uses: **Query** and **Scan**.
13. Click on the **Resources** section to expand it.
14. Click the radio button next to **Specific**.
15. In the **table** section, click on the **Add ARN** link.
16. In the **Specify ARN for table** field, **replace the entire content** with the **ARN of the DynamoDB table** that you found in the previous section. It will automatically populate the entries for Region, Account and Table name. Region should be set to us-east-1. Account should be set to your account ID (12 digits). Table name should be set to dragon_stats.
17. Click on the **Add** button.
18. Click on the **Request conditions** section to expand it.
19. Click on the **Add condition** link.
20. In the **Condition key** dropdown, select **dynamodb:Attributes** near the bottom of the list to specify a list of attributes that the user can get back.
21. In the **Qualifier** dropdown, select **For all values in request** to match every values in the specified **ProjectionExpression** parameter in the API call to DynamoDB.
22. In the **Operator** dropdown, select **StringEquals**.
23. In the **Value** field, input the value of an attribute name that you found while looking at the code. Click on the **Add another condition value** and input the next attribute name. Repeat this step until you have filled all the attributes. The order doesn't have any importance, but make sure that all the attributes are listed. They should be: "**dragon_name**", "**family**", "**protection**", "**damage**", "**description**".
24. Click on the **Add** button.
25. To enforce the use of the **ProjectionExpression** parameter in the API Call, you need to specify that the `dynamodb:Select` attribute is set to `SPECIFIC_ATTRIBUTES`. If you don't do that and you only enforce the attributes, then if the **ProjectionExpression** parameter isn't used, the attributes list won't be enforced.
    1. Click on the **Add another condition** link.
    2. In the **Condition key** dropdown, select **dynamodb:Select** at the bottom of the list.
    3. Leave the **Qualifier** to **Default**.
    4. In the **Operator** dropdown, select **StringEquals**.
    5. In the **Value** field, input `SPECIFIC_ATTRIBUTES`.

6. Click on the **Add** button.

26. Click on the **Review policy** button.
27. Click on the **Save changes** button.
28. Your policy should look *similar* to this:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "dynamodb:Scan",
                "dynamodb:Query"
            ],
            "Resource": "arn:aws:dynamodb:us-east-
1:761424745283:table/dragon_stats",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:Attributes": [
                        "dragon_name",
                        "family",
                        "protection",
                        "damage",
                        "description"
                    ]
                },
                "StringEqualsIfExists": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
                }
            }
        }
    ]
}
```

What this is doing is preventing a malicious coder requesting more data back than we want them to have. We don't need the website havng access to the `location_*` information for example, so locking this down at the database level using IAM least privilege, prevents a malicious coder bypassing the current coded projection.

Step 2D): Test the Lambda function

1. You will need a valid session again for use in the test case, so you'll need to use the AWS Cloud9 terminal to get a new session.

```
node ../lab5/solution/login.js test dave@dragoncardgame001.com apple
```

This should give you a new session like before:

```
Local test to log in a user with email of dave@dragoncardgame001.com
$2b$10$Q5.0VU2CA5JFnc0r6hSXfeNFpm2XoVXlKMVnniR7pPivIMb7wvoVy apple
Password is correct
{ ConsumedCapacity: { TableName: 'sessions', CapacityUnits: 1 } }
AWAITED a1080978-ab39-4666-ac0c-7d1f98529f4c
null { user_name_str: 'davey65',
  session_id_str: 'a1080978-ab39-4666-ac0c-7d1f98529f4c' }
```

1. Click on the **Services** menu and choose **Lambda**.
2. Click on the **DragonSearch** lambda function link.
3. In the dropdown next to the *Test* button, edit `withSession` test payload to include the new session where you see the <FMI>:

```
{
  "user_name_str": "davey65",
  "session_id_str": "<FMI>",
  "dragon_name_str": "Fireball"
}
```

1. Click on the **Test** button.

The output should fail and look like the following:

```
{
  "errorType": "string",
  "errorMessage": "nope",
  "trace": []
}
```

If you look at the log output section of the Execution result, you will see output like this:

```
    START RequestId: a77751dd-d7d0-485c-b997-6219ab3f05b9 Version: $LATEST
    2019-06-11T20:06:27.983Z    a77751dd-d7d0-485c-b997-6219ab3f05b9    INFO
To run a Local test in Cloud 9 use `node scan_dragons.js test`
    2019-06-11T20:06:27.983Z    a77751dd-d7d0-485c-b997-6219ab3f05b9    INFO
running in Lambda
    2019-06-11T20:06:27.983Z    a77751dd-d7d0-485c-b997-6219ab3f05b9    INFO
davey65 18b3dcf4-06b1-4d2e-98ad-a70f4e0b40ac
```

```
    2019-06-11T20:06:29.024Z    a77751dd-d7d0-485c-b997-6219ab3f05b9    INFO    {
AccessDeniedException: User: arn:aws:sts::628920026067:assumed-role/edx-ddb-
dragonsearch-role/DragonSearch is not authorized to perform: dynamodb:Query on
resource: arn:aws:dynamodb:us-east-1:628920026067:table/sessions
        at Request.extractError (/var/runtime/node_modules/aws-
sdk/lib/protocol/json.js:51:27)
        at Request.callListeners (/var/runtime/node_modules/aws-
sdk/lib/sequential_executor.js:106:20)
        at Request.emit (/var/runtime/node_modules/aws-
sdk/lib/sequential_executor.js:78:10)
        at Request.emit (/var/runtime/node_modules/aws-sdk/lib/request.js:683:14)
        at Request.transition (/var/runtime/node_modules/aws-
sdk/lib/request.js:22:10)
        at AcceptorStateMachine.runTo (/var/runtime/node_modules/aws-
sdk/lib/state_machine.js:14:12)
        at /var/runtime/node_modules/aws-sdk/lib/state_machine.js:26:10
        at Request.<anonymous> (/var/runtime/node_modules/aws-
sdk/lib/request.js:38:9)
        at Request.<anonymous> (/var/runtime/node_modules/aws-
sdk/lib/request.js:685:12)
        at Request.callListeners (/var/runtime/node_modules/aws-
sdk/lib/sequential_executor.js:116:18)
      message:
       'User: arn:aws:sts::628920026067:assumed-role/edx-ddb-dragonsearch-
role/DragonSearch is not authorized to perform: dynamodb:Query on resource:
arn:aws:dynamodb:us-east-1:628920026067:table/sessions',
      code: 'AccessDeniedException',
      time: 2019-06-11T20:06:28.904Z,
      requestId: '96GLOQ57HC2VKE7K3IC25D0KV3VV4KQNSO5AEMVJF66Q9ASUAAJG',
      statusCode: 400,
      retryable: false,
      retryDelay: 24.719242158357947 }
    2019-06-11T20:06:29.024Z    a77751dd-d7d0-485c-b997-6219ab3f05b9    ERROR
Invoke Error    {"errorType":"Error","errorMessage":"nope","stack":["Error:
nope","    at _homogeneousError (/var/runtime/CallbackContext.js:12:12)","    at
postError (/var/runtime/CallbackContext.js:29:51)","    at callback
(/var/runtime/CallbackContext.js:40:7)","    at
/var/runtime/CallbackContext.js:103:16","    at /var/task/index.js:24:24","    at
Response.<anonymous> (/var/task/index.js:57:21)","    at Request.<anonymous>
(/var/runtime/node_modules/aws-sdk/lib/request.js:364:18)","    at
Request.callListeners (/var/runtime/node_modules/aws-
sdk/lib/sequential_executor.js:106:20)","    at Request.emit
(/var/runtime/node_modules/aws-sdk/lib/sequential_executor.js:78:10)","    at
Request.emit (/var/runtime/node_modules/aws-sdk/lib/request.js:683:14)"]}
    END RequestId: a77751dd-d7d0-485c-b997-6219ab3f05b9
```

```
    REPORT RequestId: a77751dd-d7d0-485c-b997-6219ab3f05b9  Duration: 1229.91 ms
   Billed Duration: 1300 ms    Memory Size: 128 MB Max Memory Used: 94 MB
```

So this is good news and bad news.

The good news is that the improved policy is working! #win

The bad news is that this is reminding us (doh!) that the `session` tables is contacted in this code too. We need to modify our policy to include access to the `sessions` table, otherwise our IAM polciy is so tight that it won't let the code communicate to the authentication resource table called `sessions`, basically blocking everyone!

Let's edit our policy some more: * Head over to **IAM** * Click on **Policies** * Select the `edx-ddb-dragonsearch-policy` policy * Click **Edit Policy** and then click the **JSON** tab. * Replace the contents with the following, remember to replace your account number in your ARN where you see the <FMI>s.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "dynamodb:Scan",
                "dynamodb:Query"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:<FMI>:table/dragon_stats",
            "Condition": {
                "StringEquals": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
                },
                "ForAllValues:StringEquals": {
                    "dynamodb:Attributes": [
                        "dragon_name",
                        "family",
                        "protection",
                        "damage",
                        "description"
                    ]
                }
            }
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
```

```
            "Action": [
                "dynamodb:Query"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:<FMI>:table/sessions"
        }
    ]
}
```

Notice we added one more statement to allow a query. We could have tightened up the projection here too, to prevent the "Expires At attribute being retuned", however this is not really a big deal and would involve editing your code and adding a projection.

Click **Review Policy** and then **Save changes**.

Ok, let's try and test again using the Lambda console.

1. In the dropdown next to the *Test* button, select **withSession**. *Hopefully you still have a valid session for the test case* withSession . *If not, grab one again like you did before and pass in a valid* session_id_str *into that payload.
2. Click on the **Test** button.

The output should look like the following:

```
[
  {
    "family": {
      "S": "red"
    },
    "damage": {
      "N": "2"
    },
    "description": {
      "S": "Fireball is a young dragon in training. He is learning how to control
his fire, but is still lethal."
    },
    "protection": {
      "N": "6"
    },
    "dragon_name": {
      "S": "Fireball"
    }
  }
]
```

Ok now it is all working, let's pretend to be the malicious coder and get the `location_city` attribute back in our results.

Let's try replacing the **ProjectionExpression** of the **justThisDragon** function to include another value `location_city`:

1. In your Lambda console, replace a line of code (line 86) which is currently this:

```
    ProjectionExpression: "dragon_name, #family, protection, damage,
description",
```

and change it to this (to include `location_city`):

```
ProjectionExpression: "dragon_name, #family, protection, damage, description,
location_city",
```

AND

Replace line 106 of the scanTable function. Currently the code looks like this:

```
ProjectionExpression: "dragon_name, #family, protection, damage, description"
```

And you will replace that with this:

```
ProjectionExpression: "dragon_name, #family, protection, damage, description,
location_city"
```

NOTE: No trailing comma on line 106.

- Click on the Save button.
- Click on the Test button while making sure that withSession is selected.
- *TIP: Hopefully you still have a valid session for the test case `withSession`. If not grab one again like you did before and pass in a valid `session_id_str` into that payload then click test.*

With a valid session, you should get an Runtime.ExitError similar to the following when you run the `withSession` test case.

```
{
  "errorType": "Runtime.ExitError",
  "errorMessage": "RequestId: 02e19c49-ee86-4432-ac14-5a58ef7b0db4 Error: Runtime
exited with error: exit status 129"
}
```

If we take a closer look at the logs:

```
START RequestId: 417f1e42-e03b-428d-956a-7478bf82dd1b Version: $LATEST
2019-06-11T20:48:50.710Z    417f1e42-e03b-428d-956a-7478bf82dd1b    INFO    To
run a Local test in Cloud 9 use `node scan_dragons.js test`
2019-06-11T20:48:50.748Z    417f1e42-e03b-428d-956a-7478bf82dd1b    INFO
running in Lambda
2019-06-11T20:48:50.748Z    417f1e42-e03b-428d-956a-7478bf82dd1b    INFO
davey65 0a9d4731-cd54-4040-b516-60e50641514f
2019-06-11T20:48:51.368Z    417f1e42-e03b-428d-956a-7478bf82dd1b    INFO    match
2019-06-11T20:48:51.368Z    417f1e42-e03b-428d-956a-7478bf82dd1b    INFO    Full
scan all
2019-06-11T20:48:51.591Z    417f1e42-e03b-428d-956a-7478bf82dd1b    ERROR
Uncaught Exception  {"errorType":"AccessDeniedException","errorMessage":"User:
arn:aws:sts::628920026067:assumed-role/edx-ddb-dragonsearch-role/DragonSearch is
not authorized to perform: dynamodb:Scan on resource: arn:aws:dynamodb:us-east-
1:628920026067:table/dragon_stats","code":"AccessDeniedException","stack":
["AccessDeniedException: User: arn:aws:sts::628920026067:assumed-role/edx-ddb-
dragonsearch-role/DragonSearch is not authorized to perform: dynamodb:Scan on
resource: arn:aws:dynamodb:us-east-1:628920026067:table/dragon_stats","    at
Request.extractError (/var/runtime/node_modules/aws-
sdk/lib/protocol/json.js:51:27)","    at Request.callListeners
(/var/runtime/node_modules/aws-sdk/lib/sequential_executor.js:106:20)","    at
Request.emit (/var/runtime/node_modules/aws-
sdk/lib/sequential_executor.js:78:10)","    at Request.emit
(/var/runtime/node_modules/aws-sdk/lib/request.js:683:14)","    at
Request.transition (/var/runtime/node_modules/aws-sdk/lib/request.js:22:10)","
at AcceptorStateMachine.runTo (/var/runtime/node_modules/aws-
sdk/lib/state_machine.js:14:12)","    at /var/runtime/node_modules/aws-
sdk/lib/state_machine.js:26:10","    at Request.<anonymous>
(/var/runtime/node_modules/aws-sdk/lib/request.js:38:9)","    at Request.
<anonymous> (/var/runtime/node_modules/aws-sdk/lib/request.js:685:12)","    at
Request.callListeners (/var/runtime/node_modules/aws-
sdk/lib/sequential_executor.js:116:18)"],"message":"User:
arn:aws:sts::628920026067:assumed-role/edx-ddb-dragonsearch-role/DragonSearch is
not authorized to perform: dynamodb:Scan on resource: arn:aws:dynamodb:us-east-
1:628920026067:table/dragon_stats","time":"2019-06-
11T20:48:51.591Z","requestId":"E60G2SNQ2E8PT44H6GSK56FRABVV4KQNSO5AEMVJF66Q9ASUAA
JG","statusCode":400,"retryable":false,"retryDelay":26.91456449885179}
END RequestId: 417f1e42-e03b-428d-956a-7478bf82dd1b
REPORT RequestId: 417f1e42-e03b-428d-956a-7478bf82dd1b  Duration: 1752.91 ms
Billed Duration: 1800 ms    Memory Size: 128 MB Max Memory Used: 44 MB
RequestId: 417f1e42-e03b-428d-956a-7478bf82dd1b Error: Runtime exited with error:
exit status 129
Runtime.ExitError
```

You can see that IAM prevents us from accessing that location_str attribute. #win

So that we don't break our website we are going to revert these lines now:

Replace line 106 of the **scanTable** function.

From:

```
ProjectionExpression: "dragon_name, #family, protection, damage, description,
location_city"
```

To:

```
ProjectionExpression: "dragon_name, #family, protection, damage, description"
```

NOTE: No trailing comma on line 106.

AND

Replace a line of code (line 86) which is currently this:

```
ProjectionExpression: "dragon_name, #family, protection, damage, description,
location_city",
```

Change it to this (to remove `location_city`)

```
ProjectionExpression: "dragon_name, #family, protection, damage, description",
```

Click save at the top of the Lambda console, and then do a quick test to make sure it still works like before. Use the test `withSession` in the Lambda Console. If you want a list of all dragons, simple remove the `dragon_name_str` value from your `withSession` test event to look like this:

```
{
  "user_name_str": "davey65",
  "session_id_str": "<FMI>"
}
```

You have successfully implemented the least privilege principle by using Fine-Grained Access Controls on your IAM Policy for your Lambda function. It's time to work on the next task from Steve. In this task, you will protect access to only allow CODE acting inside the VPC to access DynamoDB.

Step 3: Configure VPC resources and Lambda to use it

You will first test the **DragonSearch** function using the CLI to make sure it's in a working state.

Then, you need to create a Virtual Private Cloud, a Subnet and a Security Group that will be used by the Lambda function. To create those resources, the AWS CLI will be used in your Cloud9 environment via a script called `lab6/create-vpc-resources.sh`. You will need to take note of the output of that script as it will be used in the next steps. Feel free to look at the `lab6/create-vpc-resources.sh` script to understand how it was done, however this isn't the purpose of the exercise.

You will then execute a CLI command to instruct Lambda to attach your Lambda function to the VPC you created.

Finally, you will test the **DragonSearch** function to see if it still works. Spoiler alert: it won't!

By now your sesssion may have expired, so lets grab a fresh one: (you should be in /lab6 path)

```
cd /home/ec2-user/environment/lab6
```

```
node ../lab5/solution/login.js test dave@dragoncardgame001.com apple
```

Then use that session instead of the `<FMI>` in the command below.

1. This command will test the Lambda function usinbg the CLI passing in a payload, similar to the way you have been testing things in the lambda console. Replace the `<FMI>` with your new `session_id_str` above.

```
aws lambda invoke --function-name DragonSearch --payload '{"user_name_str":
"davey65", "session_id_str": "<FMI>", "dragon_name_str": "Dexler"}' \
lambda-output.json  && cat lambda-output.json | jq
```

You should see an output that looks like the following:

```
{
    "ExecutedVersion": "$LATEST",
    "StatusCode": 200
}
[
  {
    "family": {
      "S": "green"
    },
    "damage": {
      "N": "4"
    },
    "description": {
      "S": "Dexler is a protector of the earth and forests. He is as green as the
earth and burrows into the ground for protection and extra defense."
    },
```

```
    "protection": {
      "N": "2"
    },
    "dragon_name": {
      "S": "Dexler"
    }
  }
]
```

1. Now that you know that your Lambda function can communicate with DynamoDB while not running from within the VPC, you need to lock this down. You will need to configure the VPC resources first. Execute the following command to create the resources:
2. Enable the script to be executed with the following command: (*You should still be in the /lab6 path.*)

```
chmod +x resources/create_vpc_resources.sh
```

Now run this command:

`./resources/create_vpc_resources.sh`

The output should be similar to the following:

```
Creating VPC...
  VPC ID 'vpc-0675ae344c5dd885e' CREATED.
  Main Route Table ID is 'rtb-08761a962ab5495c0'.
Creating Private Subnet...
  Subnet ID 'subnet-0048dd3333f83d25f' CREATED.
Creating Security Group for Lambda...
Lambda Security Group ID 'sg-047d1c2fee7131ffc' CREATED.
COMPLETED
```

Take a note of this output as it will be used in the in the next tasks.

All of the resources are created and you are ready to modify the **DragonSearch** Lambda function so it uses the VPC instead.

1. Execute the following command to do that. You will need to replace the `<FMI>`s. You will need to replace the `<SUBNET ID>` with the Subnet ID from the previous command (eg. subnet-018e23...). You will also need to replace the `<SECURITY GROUP ID>` with the Lambda Security Group ID from the previous command (eg. sg-001453...).

```
aws lambda update-function-configuration \
--function-name DragonSearch \
--vpc-config SubnetIds=<FMI>,SecurityGroupIds=<FMI>
```

You should see a JSON description of your Lambda function similar to the following:


```
    {
        "FunctionName": "DragonSearch",
        "LastModified": "2019-05-18T21:27:33.536+0000",
        "RevisionId": "be1a71da-1e72-421f-971f-45015e6b1cd6",
        "MemorySize": 128,
        "Version": "$LATEST",
        "Role": "arn:aws:iam::123456789012:role/edx-ddb-dragonsearch-role",
        "Timeout": 10,
        "Runtime": "nodejs8.10",
        "TracingConfig": {
            "Mode": "PassThrough"
        },
        "CodeSha256": "NOKI11IJrYf7fgrxPr1OnYO188jc1u1uu291MGI1i4A=",
        "Description": "",
        "VpcConfig": {
            "SubnetIds": [
                "subnet-018e23c152b1e1afb"
            ],
            "VpcId": "vpc-0831c011881f2d3b4",
            "SecurityGroupIds": [
                "sg-001453b5187613c7d"
            ]
        },
        "CodeSize": 718,
        "FunctionArn": "arn:aws:lambda:us-east-
1:123456789012:function:DragonSearch",
        "Handler": "index.handler"
    }
```


1. Test the Lambda function to see if you can communicate with DynamoDB now by running the same command as previously. Note that it could take up to 10 seconds for getting the output. That is normal as your Lambda function needs to timeout. However, by now your session may have expired, so let's grab a fresh one first.

```
node ../lab5/solution/login.js test dave@dragoncardgame001.com apple
```

Use that `session_id_str` here (place the `<FMI>`:

```
aws lambda invoke --function-name DragonSearch --payload '{"user_name_str":
"davey65", "session_id_str": "<FMI>", "dragon_name_str": "Dexler"}' \
lambda-output.json  && cat lambda-output.json | jq
```

You should see an output that looks like the following:

```
{
    "FunctionError": "Unhandled",
    "ExecutedVersion": "$LATEST",
    "StatusCode": 200
}
{
  "errorMessage": "2019-06-12T17:39:27.752Z 74d9c774-4380-4b54-b092-45e180075020
Task timed out after 10.01 seconds"
}
```

It sounds like your Lambda function doesn't work anymore, yet all you did was modify it to run within the VPC. So this means it has to be an issue with the communication with DynamoDB. This is what we would expect as normally Lambda would communicate over the internet to the DynamoDB endpoint. If it is inside a VPC, it does not have by default access to reach the internet.

Looks like we are too protected now, we need to allow access to DynamoDB from calls made within the VPC (i.e., where our new Lambda function is sitting).

Real world tip: To really lock it down, you would also prevent standard (no VPC) calls from interacting with DynamoDB. However, for purposes of the lab we are not doing that right now.

Step 4: Create the VPC Gateway Endpoint for DynamoDB

In this task, you will first create the **VPC Gateway Endpoint** for DynamoDB by passing the **VPC ID** and **Route Table** you created from the previous step.

You will then test the **DragonSearch** Lambda function to see if it can now communicate with DynamoDB from within the VPC. It should work with no code changes, it's all done via networking.

Finally, you will test everything in the website.

1. To create the VPC Gateway Endpoint, execute the following command. You will need to replace the `<VPC ID>` with the VPC ID from the previous task (eg. vpc-0831c0...). You will also need to replace the `<ROUTE TABLE ID>` with the Main Route Table ID from the previous task (eg. rtb-0fdc313...).

```
aws ec2 create-vpc-endpoint --service-name com.amazonaws.us-east-1.dynamodb \
--vpc-id <VPC ID> --route-table-ids <ROUTE TABLE ID>
```

```
The output of the command should be similar to the following:
```

```
    {
        "VpcEndpoint": {
            "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":
[{\"Effect\":\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}
",
            "VpcId": "vpc-0831c011881f2d3b4",
            "NetworkInterfaceIds": [],
            "SubnetIds": [],
            "PrivateDnsEnabled": false,
            "State": "available",
            "ServiceName": "com.amazonaws.us-east-1.dynamodb",
            "RouteTableIds": [
                "rtb-0fdc3131f0c004447"
            ],
            "Groups": [],
            "VpcEndpointId": "vpce-016c81e1119c128e7",
            "VpcEndpointType": "Gateway",
            "CreationTimestamp": "2019-05-18T21:48:06.000Z",
            "DnsEntries": []
        }
    }
```

```
You can see that a Policy Document has been created automatically that allows
everyone `"Principal":"*"` inside the VPC to do any action `"Action":"*"` against
any DynamoDB resources `"Resource":"*"`. *So if you wanted to only allow certain
actions or resources through the VPC Gateway Endpoint, you could modify this
Policy Document.*
```

1. Now that the VPC Gateway Endpoint for DynamoDB is created, it's time to test the **DragonSearch** Lambda function the same way as you did in the previous task by running the following command.

```
node ../lab5/solution/login.js test dave@dragoncardgame001.com apple
```

Use that `session_id_str` here (replace the `<FMI>`:

```
 aws lambda invoke --function-name DragonSearch --payload '{"user_name_str":
"davey65", "session_id_str": "<FMI>", "dragon_name_str": "Dexler"}' \
 lambda-output.json  && cat lambda-output.json | jq
```

You should see an output exactly the same as the last time this test worked. It should look like the following:

```
{
    "ExecutedVersion": "$LATEST",
    "StatusCode": 200
}
[
  {
    "family": {
      "S": "green"
    },
    "damage": {
      "N": "4"
    },
    "description": {
      "S": "Dexler is a protector of the earth and forests. He is as green as the
earth and burrows into the ground for protection and extra defense."
    },
    "protection": {
      "N": "2"
    },
    "dragon_name": {
      "S": "Dexler"
    }
  }
]
```

Head over to your website (index3.html) *using your bucket name*

```
http://<FMI>.s3-website-us-east-1.amazonaws.com/index3.html
```

Log in as dave

```
dave@dragoncardgame001.com

apple
```

You should see a list of Dragons as if nothing was done. Yet our Lambda function is sat within our VPC to keep Steve happy.

Step 5: Remove the Lambda function from the VPC

For future exercises, it will be easier to have the function outside the VPC.

1. Run the following CLI command from the AWS Cloud9 **bash terminal** to remove the VPC configuration from the **DragonSearch** Lambda function:

```
aws lambda update-function-configuration --function-name DragonSearch \
--vpc-config SubnetIds=[],SecurityGroupIds=[]
```

Congratulations! You have completed exercise 6.