Unit 8: Automation

Contents

Lab Questions

1, 2, 3, 4, 5, 6, 7

Getting Started

In this unit, we'll combine concepts in previous units to collect data from an external source, process it, generate a report, and deliver it. We'll do this using cloud services, specifically Amazon Web Services (https://aws.amazon.com/) so an AWS account will be required. The services we'll use qualify for the free tier (https://aws.amazon.com/free/).

In this example, our report will consist of simple stock portfolio information including shares, price, and total value.

While we'll do some initial setup using the web console, most of the work we'll do with AWS will be done programmatically using Python. To do this, we'll need the *boto3* (https://aws.amazon.com/sdk-for-python/) library, which can be installed with `pip` .

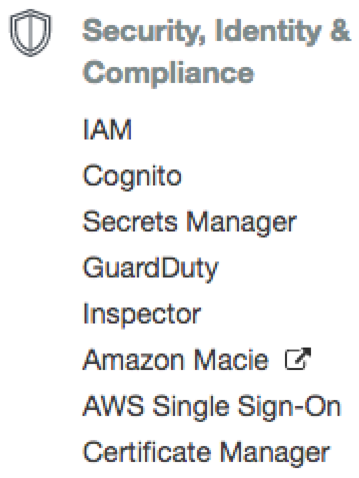
```
In [3]: import sys
        !{sys.executable} -m pip install boto3
```

```
Requirement already satisfied: boto3 in /usr/local/lib/python3.6/site-pac
kages (1.7.4)
Requirement already satisfied: botocore<1.11.0,>=1.10.4 in /usr/local/li
b/python3.6/site-packages (from boto3) (1.10.4)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /usr/local/lib/p
ython3.6/site-packages (from boto3) (0.9.3)
Requirement already satisfied: s3transfer<0.2.0,>=0.1.10 in /usr/local/li
b/python3.6/site-packages (from boto3) (0.1.13)
Requirement already satisfied: docutils>=0.10 in /usr/local/lib/python3.
6/site-packages (from botocore<1.11.0,>=1.10.4->boto3) (0.14)
Requirement already satisfied: python-dateutil<2.7.0,>=2.1 in /usr/local/
lib/python3.6/site-packages (from botocore<1.11.0,>=1.10.4->boto3) (2.6.
1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/site-
packages (from python-dateutil<2.7.0,>=2.1->botocore<1.11.0,>=1.10.4->bot
o3) (1.11.0)
```

Generating Keys

Before we can use boto with our AWS accounts, we need to create an access key and a secret key that will be used to identify our account and the fact that we are authorized to make changes to our accounts. To generate keys, we'll use the AWS web console.

Once logged into the AWS console, open the IAM interface by selecting "IAM" in the "Security, Identity & Compliance" section within the services menu.

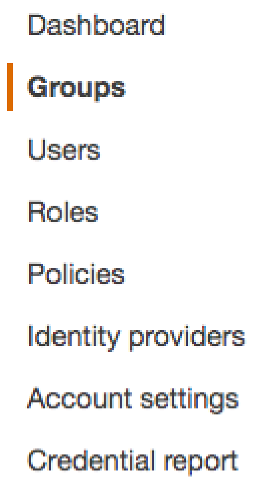


Security, Identity & Compliance Services

IAM, or Identity and Access Management, allows us to manage access to an AWS account. IAM allows us to create users and groups with granular permissions. In multi-user organizations, IAM can be used to ensure that people only have access to services they need and don't have access to other services. Access control for programs we create can also be managed through IAM.

Often groups of users will have the same set of permissions. Rather than repeatedly assigning the same permissions to those users, we instead create a group to which the permissions are assigned and then indicate that users are members of the group. This provides additional flexibility by allowing different users to be members of different groups.

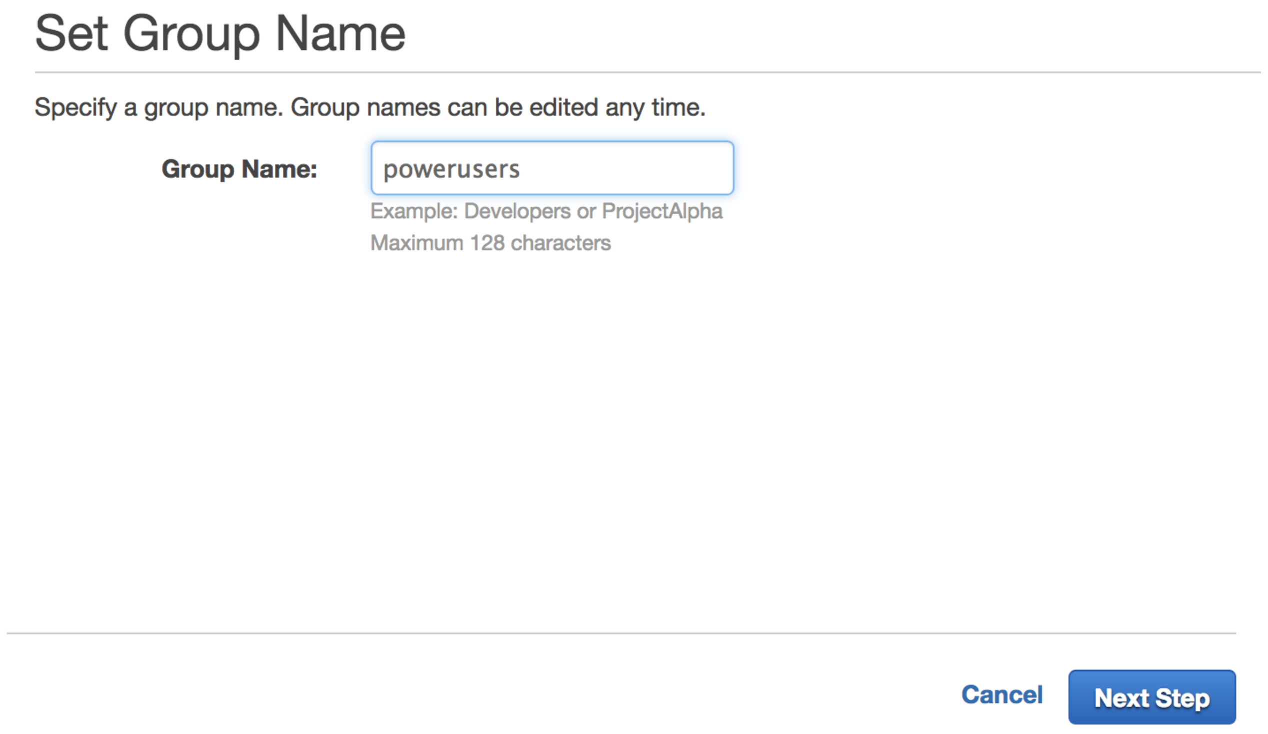
To begin, select "Groups" from the left side of the console.



IAM Menu

To create a new group, click the "Create New Group" button. While it is generally best practice to make permissions as limited as possible, this can be a time consuming process. For our purposes, we'll create a group with "power user" access - access that allows members the group to create, modify, delete, and manage most of the AWS services.

For the new group's name, enter a descriptive value that reflects the permissions that will be assigned to the group such as "powerusers". Click "Next Step" to continue.



Set Group Name

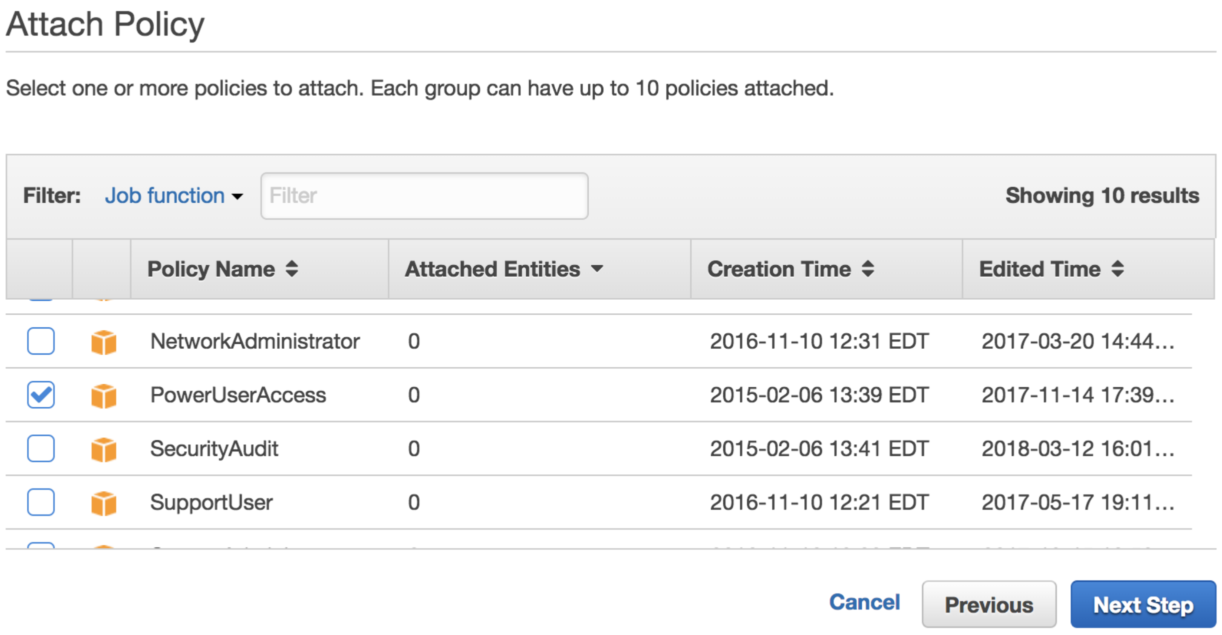
Specify a group name. Group names can be edited any time.

Group Name: powerusers

Example: Developers or ProjectAlpha
Maximum 128 characters

Specify a Group Name

Next, we attach a policy to the group. Policies define the permissions a group has. We can filter these by choosing a filter from the drop-down list. Filter by "Job function". Among the policy names, we see "PowerUserAccess" - this provides full control to all AWS services with the exception of user management. Click "Next Step" to continue.



Attach Policy

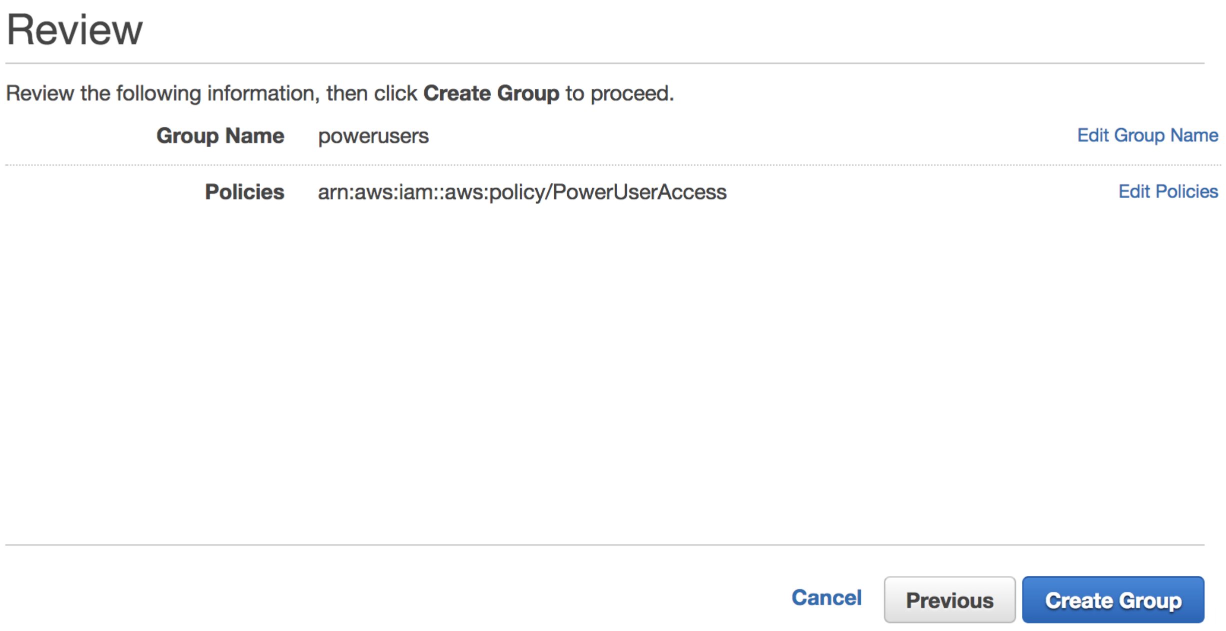
Select one or more policies to attach. Each group can have up to 10 policies attached.

Filter: Job function | Showing 10 results

		Policy Name	Attached Entities	Creation Time	Edited Time
☐		NetworkAdministrator	0	2016-11-10 12:31 EDT	2017-03-20 14:44...
✓		PowerUserAccess	0	2015-02-06 13:39 EDT	2017-11-14 17:39...
☐		SecurityAudit	0	2015-02-06 13:41 EDT	2018-03-12 16:01...
☐		SupportUser	0	2016-11-10 12:21 EDT	2017-05-17 19:11...

Attach a Policy

As a final step, we have a chance to review the settings for the new group. If everything appears correct, click "Create Group". Click "Cancel" or "Previous" if changes need to be made.



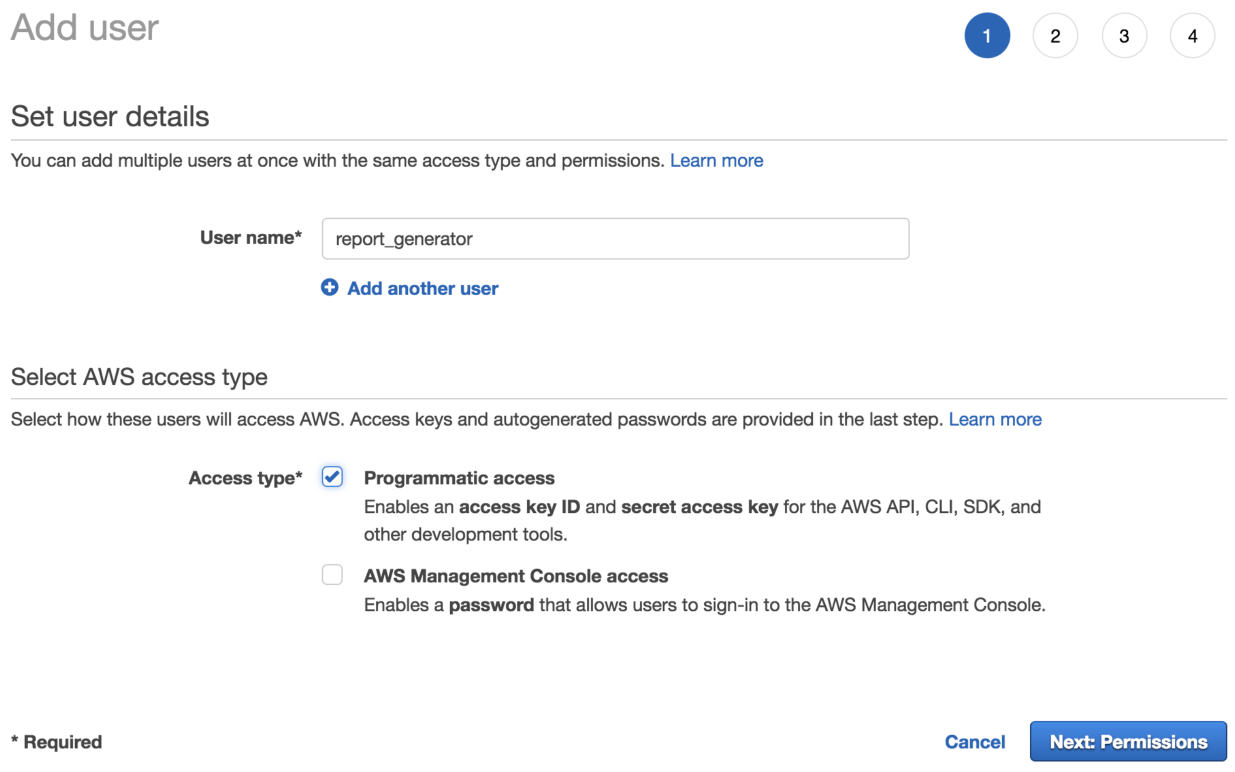
Review

Review the following information, then click **Create Group** to proceed.

Group Name	powerusers	Edit Group Name
Policies	arn:aws:iam::aws:policy/PowerUserAccess	Edit Policies

Review Group Options

Now that we have a group, we'll add a user. On the left, select "Users". We see the list of existing users. To add a user, click the "Add User" button. This will begin the user creation process. In the first step, we will specify the user name an the access type. If we were creating a user for a person, we might choose a user name based on the person's name; because we are creating a user name for a program, we should choose a descriptive name like "report_generator". We'll use this account for programmatic access so select the "Programmatic access" type. Click "Next: Permissions" to continue.


Add user

1 2 3 4

Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name* report_generator

Add another user

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more

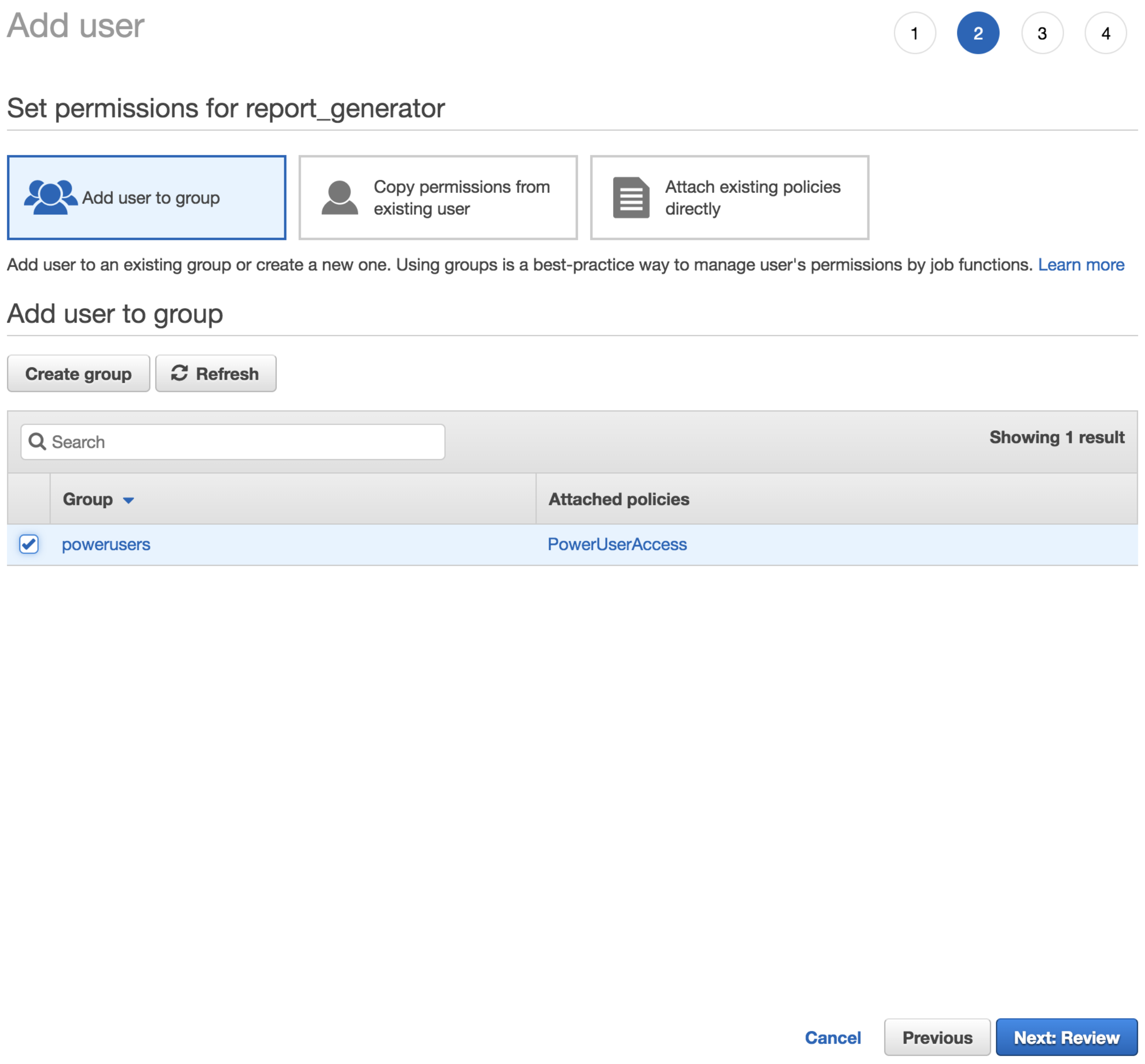
Access type* [✓] **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

[ ] **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

* Required

Specify a User Name and Access Type

To assign permissions to a user, we specify the groups to which the user will belong. For our purposes, the new user will be a member of the group we created; select the appropriate group. Click "Next: Review" to continue.


Add user

Set permissions for report_generator

Add user to group
Copy permissions from existing user
Attach existing policies directly

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. Learn more

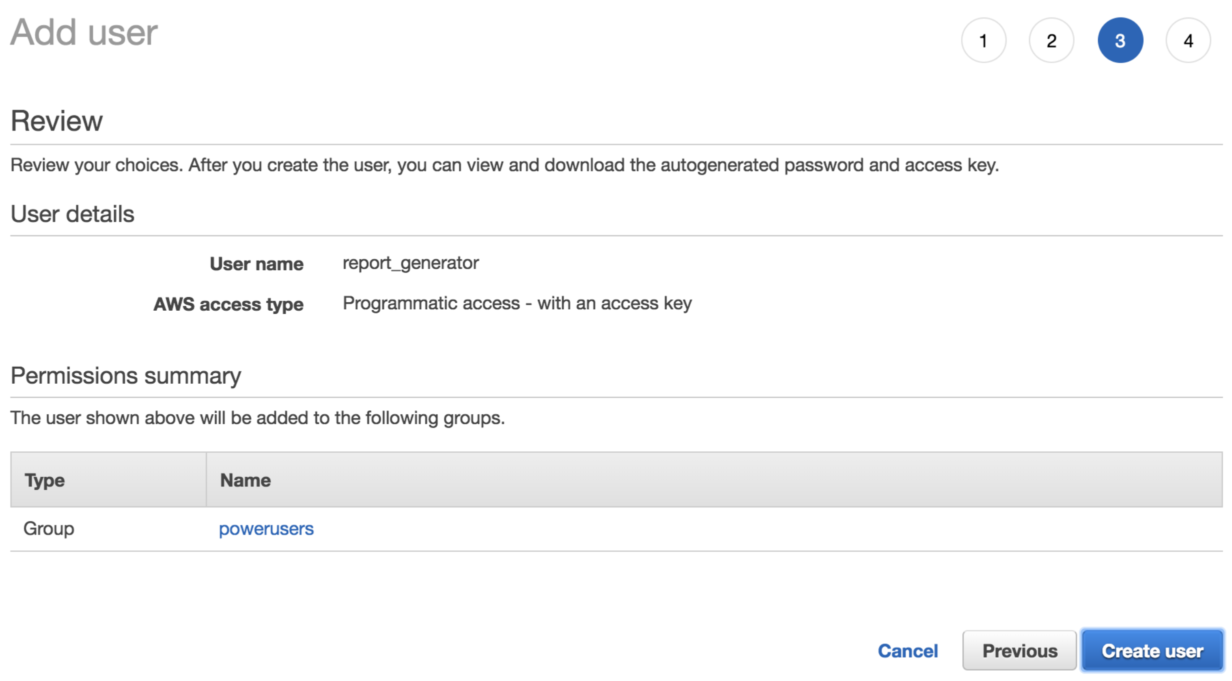
Add user to group

Showing 1 result

✓	Group	Attached policies
✓	powerusers	PowerUserAccess

Specify Group Membership

Next, we have a chance to review the settings for the new user. If everything appears correct, click "Create User". Click "Cancel" or "Previous" if changes need to be made.



Add user

1 2 3 4

Review

Review your choices. After you create the user, you can view and download the autogenerated password and access key.

User details

User name	report_generator
AWS access type	Programmatic access - with an access key

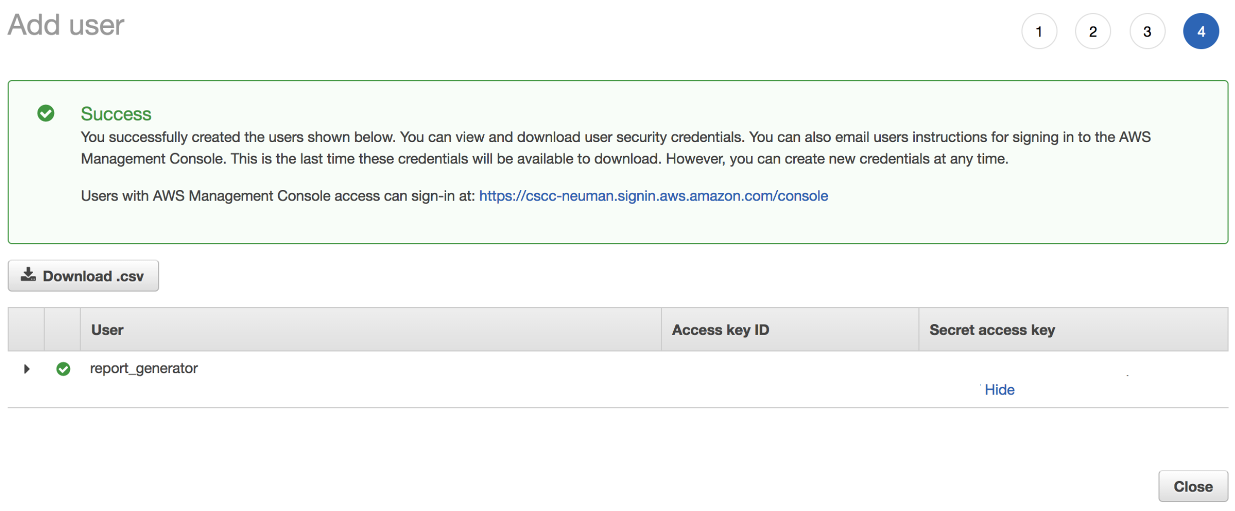
Permissions summary

The user shown above will be added to the following groups.

Type	Name
Group	powerusers

Review User Options

Finally, we see a success message and the access key for the new user. We can also choose to see the secret access key. Enter these values in the cell below.



Add user

1 2 3 4

Success

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: https://cscc-neuman.signin.aws.amazon.com/console

		User	Access key ID	Secret access key
▸	✓	report_generator		Hide

User Access and Secret Keys

Lab 1 In the cell below, set the values for `ACCESS_KEY` and `SECRET_KEY` using the values displayed in the AWS console.

```
In [154]: ACCESS_KEY = ""
          SECRET_KEY = ""
```

Creating a Data Store

Our objective is to automate data collection and processing and report generation and delivery. Once we've collected the data and completed some initial processing, we'll likely need to store it for future use. While we could create a relational database using the Amazon Relational Database Service (https://aws.amazon.com/rds/), we'll instead rely on a NoSQL store using Amazon SimpleDB (https://aws.amazon.com/simpledb/).

To work with SimpleDB from boto, we must first create a *client()* (https://boto3.readthedocs.io/en/latest/guide/quickstart.html) indicating the service we intend to use and with our credentials.

For this example, we'll use the "us-east-1" region as it includes all the AWS services we need.

In [130]:
```
import boto3
sdb_client = boto3.client('sdb',
                          aws_access_key_id = ACCESS_KEY,
                          aws_secret_access_key = SECRET_KEY,
                          region_name="us-east-1")
```

Instead of databases and tables, SimpleDB uses *domains* to categorize data. To create a new domain, we can use the client's *create_domain()* (https://boto3.readthedocs.io/en/latest/reference/services/sdb.html#SimpleDB.Client.create_domain) method.

In [131]:
```
sdb_client.create_domain(DomainName="test_domain")
```

Out[131]:
```
{'ResponseMetadata': {'BoxUsage': '0.0055590278',
  'HTTPHeaders': {'connection': 'keep-alive',
   'content-type': 'text/xml',
   'date': 'Sun, 22 Apr 2018 18:00:56 GMT',
   'server': 'Amazon SimpleDB',
   'transfer-encoding': 'chunked',
   'vary': 'Accept-Encoding'},
  'HTTPStatusCode': 200,
  'RequestId': '65344e29-625b-708a-f290-9b3856a5e21c',
  'RetryAttempts': 0}}
```

If successful, the *create_domain()* method returns information related to the domain. Note that an HTTP status code of 200 (https://en.wikipedia.org/wiki/List_of_HTTP_status_codes#2xx_Success) indicates success. We can also verify that the domain was created successfully by viewing its metadata. To do this, we can use the client's *domain_metadata()* (https://boto3.readthedocs.io/en/latest/reference/services/sdb.html#SimpleDB.Client.domain_metada method. When calling the method, we specify the domain name using the `DomainName` keyword argument.

Lab 2 In the cell below, retrieve the metadata for the recently-created domain using the *domain_metadata_method()*.

```
In [132]: sdb_client.domain_metadata(DomainName='test_domain')
```

```
Out[132]: {'AttributeNameCount': 0,
 'AttributeNamesSizeBytes': 0,
 'AttributeValueCount': 0,
 'AttributeValuesSizeBytes': 0,
 'ItemCount': 0,
 'ItemNamesSizeBytes': 0,
 'ResponseMetadata': {'BoxUsage': '0.0000071759',
  'HTTPHeaders': {'connection': 'keep-alive',
   'content-type': 'text/xml',
   'date': 'Sun, 22 Apr 2018 18:01:02 GMT',
   'server': 'Amazon SimpleDB',
   'transfer-encoding': 'chunked',
   'vary': 'Accept-Encoding'},
  'HTTPStatusCode': 200,
  'RequestId': '2c593445-06e5-965d-6b1c-5315659983e3',
  'RetryAttempts': 0},
 'Timestamp': 1524420062}
```

With a domain created, we can now begin adding data; to do this, we can use the *put_attributes()* (https://boto3.readthedocs.io/en/latest/reference/services/sdb.html#SimpleDB.Client.put_attributes) method. Data in SimpleDB is stored as items where each item can have a set of attributes associated with it. In the code below, we add an item named `employee1` to represent an employee. In the example, there are two attributes associated with the employee, full name and department. To associate these with the item, we specify a list of dictionaries for attributes where each dictionary has two key/value pairs. Each attribute dictionary contains a `Name` and `Value` key where the corresponding values are the data for each attribute. If we think of our data as being tabular, we can think of the `ItemName` as identifying a row, the attribute `Name` key as specifying a column name, and the attribute `Value` key as being used to provide a field value.

Given data that could be stored like this

ID	Fullname	Department
employee1	Bob Smith	Marketing

we can store it in SimpleDB using the following code.

```
In [133]: sdb_client.put_attributes(DomainName='test_domain',
                                    ItemName='employee1',
                                    Attributes=[
                                        {
                                            'Name': 'Fullname',
                                            'Value': 'Bob Smith'
                                        },
                                        {
                                            'Name': 'Department',
                                            'Value': 'Marketing'
                                        }
                                    ])
```

```
Out[133]: {'ResponseMetadata': {'BoxUsage': '0.0000219923',
            'HTTPHeaders': {'connection': 'keep-alive',
             'content-type': 'text/xml',
             'date': 'Sun, 22 Apr 2018 18:01:26 GMT',
             'server': 'Amazon SimpleDB',
             'transfer-encoding': 'chunked',
             'vary': 'Accept-Encoding'},
            'HTTPStatusCode': 200,
            'RequestId': '1de68219-3132-37d2-8bb6-df8742469246',
            'RetryAttempts': 0}}
```

Lab 3 In the cell below, use the *put_attributes()* method to add data for the following item.

ID	Fullname	Department
employee2	Jane Doe	IT

```
In [134]: sdb_client.put_attributes(DomainName='test_domain',
                                    ItemName='employee2',
                                    Attributes=[
                                        {
                                            'Name': 'Fullname',
                                            'Value': 'Jane Doe'
                                        },
                                        {
                                            'Name': 'Department',
                                            'Value': 'IT'
                                        }
                                    ])
```

```
Out[134]: {'ResponseMetadata': {'BoxUsage': '0.0000219923',
            'HTTPHeaders': {'connection': 'keep-alive',
             'content-type': 'text/xml',
             'date': 'Sun, 22 Apr 2018 18:02:22 GMT',
             'server': 'Amazon SimpleDB',
             'transfer-encoding': 'chunked',
             'vary': 'Accept-Encoding'},
            'HTTPStatusCode': 200,
            'RequestId': 'e6061205-37aa-15b4-de8b-ef7dd8c394a6',
            'RetryAttempts': 0}}
```

To retrieve data for a specific item, we can use the *get_attributes()* (https://boto3.readthedocs.io/en/latest/reference/services/sdb.html#SimpleDB.Client.get_attributes) method where we specify the item's name. Note that the returned value is a dictionary containing both item attributes and metadata.

```
In [135]: sdb_client.get_attributes(DomainName='test_domain', ItemName="employee1")
```

```
Out[135]: {'Attributes': [{'Name': 'Department', 'Value': 'Marketing'},
            {'Name': 'Fullname', 'Value': 'Bob Smith'}],
           'ResponseMetadata': {'BoxUsage': '0.0000093282',
            'HTTPHeaders': {'connection': 'keep-alive',
             'content-type': 'text/xml',
             'date': 'Sun, 22 Apr 2018 18:02:24 GMT',
             'server': 'Amazon SimpleDB',
             'transfer-encoding': 'chunked',
             'vary': 'Accept-Encoding'},
            'HTTPStatusCode': 200,
            'RequestId': '66888e0f-c247-9f9b-01a4-40103608c3b2',
            'RetryAttempts': 0}}
```

To get a specific attribute for a specific item, we can specify the attribute name in a list with the `AttributeNames` keyword argument.

```
In [136]: sdb_client.get_attributes(DomainName='test_domain', ItemName="employee1", At
```

```
Out[136]: {'Attributes': [{'Name': 'Department', 'Value': 'Marketing'}],
           'ResponseMetadata': {'BoxUsage': '0.0000093222',
            'HTTPHeaders': {'connection': 'keep-alive',
             'content-type': 'text/xml',
             'date': 'Sun, 22 Apr 2018 18:02:25 GMT',
             'server': 'Amazon SimpleDB',
             'transfer-encoding': 'chunked',
             'vary': 'Accept-Encoding'},
            'HTTPStatusCode': 200,
            'RequestId': 'be126a52-adfb-7f18-19ff-8a61467e7cee',
            'RetryAttempts': 0}}
```

We can also use the *select()* (https://boto3.readthedocs.io/en/latest/reference/services/sdb.html#SimpleDB.Client.select) method to find data using SQL-like queries. For example, we can use the following code to find all employees in the marketing department. Notice that the domain name is used like a table name in the query.

```
In [137]: sdb_client.select(SelectExpression="select * from test_domain where Departme
```

```
Out[137]: {'Items': [{'Attributes': [{'Name': 'Department', 'Value': 'Marketing'},
               {'Name': 'Fullname', 'Value': 'Bob Smith'}],
             'Name': 'employee1'}],
           'ResponseMetadata': {'BoxUsage': '0.0000228616',
            'HTTPHeaders': {'connection': 'keep-alive',
             'content-type': 'text/xml',
             'date': 'Sun, 22 Apr 2018 18:02:31 GMT',
             'server': 'Amazon SimpleDB',
             'transfer-encoding': 'chunked',
             'vary': 'Accept-Encoding'},
            'HTTPStatusCode': 200,
            'RequestId': 'a34a3876-80a9-2ddd-02da-9538b7a6aee3',
            'RetryAttempts': 0}}
```

Lab 4 In the cell below, use the *select()* method to find data where the employee's full name is "Bob Smith".

```
In [138]: sdb_client.select(SelectExpression="select * from test_domain where Fullname
```

```
Out[138]: {'Items': [{'Attributes': [{'Name': 'Department', 'Value': 'Marketing'},
              {'Name': 'Fullname', 'Value': 'Bob Smith'}],
             'Name': 'employee1'}],
           'ResponseMetadata': {'BoxUsage': '0.0000228616',
            'HTTPHeaders': {'connection': 'keep-alive',
             'content-type': 'text/xml',
             'date': 'Sun, 22 Apr 2018 18:02:58 GMT',
             'server': 'Amazon SimpleDB',
             'transfer-encoding': 'chunked',
             'vary': 'Accept-Encoding'},
            'HTTPStatusCode': 200,
            'RequestId': 'c8de410e-2aac-efe2-79ce-ba894dcf1197',
            'RetryAttempts': 0}}
```

We can update attributes for records using the *put_attributes()* method. To do this, we specify the appropriate attribute `Name` and `Value` keys/value pairs along with `Replace: True` . For example, we can change `employee1` 's name.

```
In [139]: sdb_client.put_attributes(DomainName='test_domain',
                                    ItemName='employee1',
                                    Attributes=[
                                        {
                                            'Name': 'Fullname',
                                            'Value': 'Robert Smith',
                                            'Replace': True
                                        }
                                    ])
```

```
Out[139]: {'ResponseMetadata': {'BoxUsage': '0.0000219909',
            'HTTPHeaders': {'connection': 'keep-alive',
             'content-type': 'text/xml',
             'date': 'Sun, 22 Apr 2018 18:02:59 GMT',
             'server': 'Amazon SimpleDB',
             'transfer-encoding': 'chunked',
             'vary': 'Accept-Encoding'},
            'HTTPStatusCode': 200,
            'RequestId': '70e97883-0557-5e79-6d3e-7c6ff3a7da6b',
            'RetryAttempts': 0}}
```

We can confirm the change using the *get_attributes()* method.

```
In [140]: sdb_client.get_attributes(DomainName='test_domain', ItemName="employee1")
```

```
Out[140]: {'Attributes': [{'Name': 'Department', 'Value': 'Marketing'},
            {'Name': 'Fullname', 'Value': 'Robert Smith'}],
           'ResponseMetadata': {'BoxUsage': '0.0000093282',
            'HTTPHeaders': {'connection': 'keep-alive',
             'content-type': 'text/xml',
             'date': 'Sun, 22 Apr 2018 18:03:02 GMT',
             'server': 'Amazon SimpleDB',
             'transfer-encoding': 'chunked',
             'vary': 'Accept-Encoding'},
            'HTTPStatusCode': 200,
            'RequestId': 'fff0890f-a234-2673-ef60-9c2ef4d822fa',
            'RetryAttempts': 0}}
```

We can delete items using the [*delete_attributes()*] method, specifying the domain and item names.

```
In [141]: sdb_client.delete_attributes(DomainName='test_domain', ItemName="employee1"
```

```
Out[141]: {'ResponseMetadata': {'BoxUsage': '0.0000219907',
            'HTTPHeaders': {'connection': 'keep-alive',
             'content-type': 'text/xml',
             'date': 'Sun, 22 Apr 2018 18:03:05 GMT',
             'server': 'Amazon SimpleDB',
             'transfer-encoding': 'chunked',
             'vary': 'Accept-Encoding'},
            'HTTPStatusCode': 200,
            'RequestId': 'd3f2cb21-41ef-d5fb-3e7a-fc7843e684fc',
            'RetryAttempts': 0}}
```

Lab 5 In the cell below, use the *delete_attributes()* method to delete the `employee2` item.

```
In [142]: sdb_client.delete_attributes(DomainName='test_domain', ItemName="employee2"
```

```
Out[142]: {'ResponseMetadata': {'BoxUsage': '0.0000219907',
            'HTTPHeaders': {'connection': 'keep-alive',
             'content-type': 'text/xml',
             'date': 'Sun, 22 Apr 2018 18:03:27 GMT',
             'server': 'Amazon SimpleDB',
             'transfer-encoding': 'chunked',
             'vary': 'Accept-Encoding'},
            'HTTPStatusCode': 200,
            'RequestId': '51e43e34-6c25-d959-f487-35b262a98347',
            'RetryAttempts': 0}}
```

At this point, our domain should be empty. We can confirm this using the *domain_metadata()* method.

```
In [143]: sdb_client.domain_metadata(DomainName='test_domain')
```

```
Out[143]: {'AttributeNameCount': 0,
 'AttributeNamesSizeBytes': 0,
 'AttributeValueCount': 0,
 'AttributeValuesSizeBytes': 0,
 'ItemCount': 0,
 'ItemNamesSizeBytes': 0,
 'ResponseMetadata': {'BoxUsage': '0.0000071759',
  'HTTPHeaders': {'connection': 'keep-alive',
   'content-type': 'text/xml',
   'date': 'Sun, 22 Apr 2018 18:03:28 GMT',
   'server': 'Amazon SimpleDB',
   'transfer-encoding': 'chunked',
   'vary': 'Accept-Encoding'},
  'HTTPStatusCode': 200,
  'RequestId': '2f0ec258-3f47-91eb-e656-9c619cfd87d1',
  'RetryAttempts': 0},
 'Timestamp': 1524420208}
```

To delete the domain entirely, we can use the *delete_domain()* (https://boto3.readthedocs.io/en/latest/reference/services/sdb.html#SimpleDB.Client.delete_domain) method.

```
In [144]: sdb_client.delete_domain(DomainName='test_domain')
```

```
Out[144]: {'ResponseMetadata': {'BoxUsage': '0.0055590278',
  'HTTPHeaders': {'connection': 'keep-alive',
   'content-type': 'text/xml',
   'date': 'Sun, 22 Apr 2018 18:03:32 GMT',
   'server': 'Amazon SimpleDB',
   'transfer-encoding': 'chunked',
   'vary': 'Accept-Encoding'},
  'HTTPStatusCode': 200,
  'RequestId': '56b81611-5479-d1f7-a55a-cbfcf4b97778',
  'RetryAttempts': 0}}
```

SimpleDB provides finer control over the data such as deleting individual attributes rather than entire items. The methods we've looked at will be sufficient for our task.

Now that we have a way of storing data, we need a way of obtaining the data and processing it.

Data Retrieval and Processing

Suppose we had a simple portfolio consisting of shares of `goog` and `appl`. Our report generation process would need to retrieve prices for each. To do this, use the IEX API (https://iextrading.com/developer/docs/#getting-started). For our simple report, we'll use only the latest price data accessible using the price (https://iextrading.com/developer/docs/#price) endpoint.

To retrieve and process the data into Python objects, we'll use the Requests (http://docs.python-requests.org/en/master/) library. In the example below, we get the latest price for `goog` . Recall that the response contain JSON data that can be processed using the *json()* method.

In [145]:
```
import requests

response = requests.get("https://api.iextrading.com/1.0/stock/goog/price")
price = response.json()
price
```

Out[145]:
```
1072.96
```

We can store price data and number of shares for each stock in a SimpleDB domain. To start, we'll create the domain and an item for each stock where the stock's name will be used as the item name. Price and number of shares will be stored as attributes.

In [146]:
```
sdb_client.create_domain(DomainName='portfolio')

for name in ['aapl', 'goog']:
    sdb_client.put_attributes(DomainName='portfolio',
                              ItemName=name,
                              Attributes=[
                                  {
                                      'Name': 'price',
                                      'Value': '0'
                                  },
                                  {
                                      'Name': 'shares',
                                      'Value': '5'
                                  }
                              ])
```

We can confirm that the items existing using the *select()* method.

```
In [147]: sdb_client.select(SelectExpression="select * from portfolio")
```

```
Out[147]: {'Items': [{'Attributes': [{'Name': 'shares', 'Value': '5'},
              {'Name': 'price', 'Value': '165.72'},
              {'Name': 'price', 'Value': '0'}],
             'Name': 'aapl'},
            {'Attributes': [{'Name': 'shares', 'Value': '5'},
              {'Name': 'price', 'Value': '1072.96'},
              {'Name': 'price', 'Value': '0'}],
             'Name': 'goog'}],
           'ResponseMetadata': {'BoxUsage': '0.0000320033',
            'HTTPHeaders': {'connection': 'keep-alive',
             'content-type': 'text/xml',
             'date': 'Sun, 22 Apr 2018 18:03:44 GMT',
             'server': 'Amazon SimpleDB',
             'transfer-encoding': 'chunked',
             'vary': 'Accept-Encoding'},
            'HTTPStatusCode': 200,
            'RequestId': '3207ece2-0428-4b2a-3c8b-2d1516b86224',
            'RetryAttempts': 0}}
```

We can now write a function that use the IEX API to update the price data in the SimpleDB domain.

```
In [148]: def update_portfolio(sdb_client):
              results = sdb_client.select(SelectExpression="select * from portfolio")
              for item in results['Items']:
                  name = item['Name']
                  url = "https://api.iextrading.com/1.0/stock/{0}/price".format(name)
                  price = requests.get(url).json()
                  sdb_client.put_attributes(
                      DomainName='portfolio',
                      ItemName=name,
                      Attributes=[
                          {
                              'Name': 'price',
                              'Value': str(price),
                              'Replace': True
                          }
                  ])
```

Note that we values in SimpleDB are stored as strings so we have to convert the price data to a string when storing it.

Let's execute the function.

```
In [149]: update_portfolio(sdb_client)
```

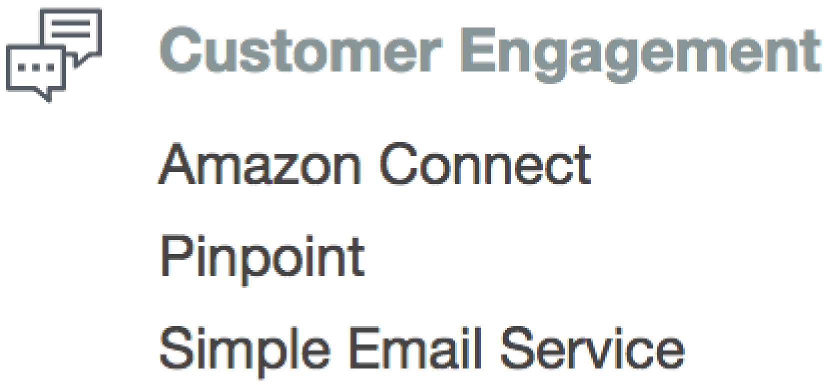
To see if the domain was successfully updated, we can use the *select()* method.

```
In [150]: sdb_client.select(SelectExpression="select * from portfolio")['Items']

Out[150]: [{'Attributes': [{'Name': 'shares', 'Value': '5'},
             {'Name': 'price', 'Value': '165.72'}],
            'Name': 'aapl'},
           {'Attributes': [{'Name': 'shares', 'Value': '5'},
             {'Name': 'price', 'Value': '1072.96'}],
            'Name': 'goog'}]
```

Now that we have a way of retrieving the data and updating the SimpleDB domain, we work on reporting with the data. For this example, our report will be a simple text summary that we email to ourselves.

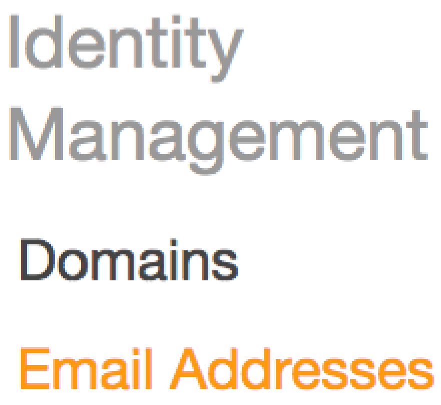
In order to generate and send the email, we will use the [AWS Simple Email Service (https://aws.amazon.com/ses/)](https://aws.amazon.com/ses/). In order to use it, we have to first verify an email address.

Log into the AWS console and select "Simple Email Service" from the "Customer Engagement" section. Make sure the "US East (N. Virginia)" region is selected in the upper right corner of the console.



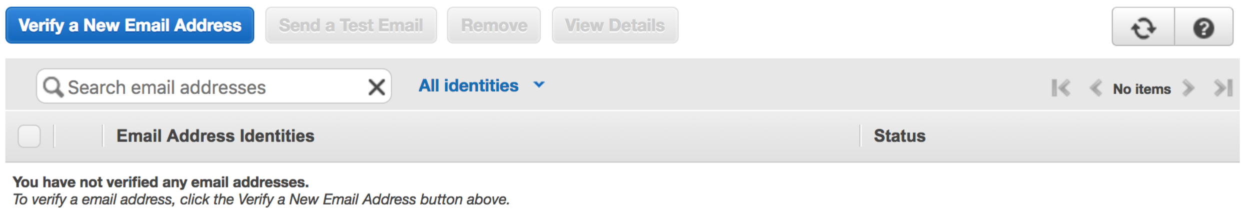
Customer Engagement Services

On the left, click the "Email Adresses" link in the "Identity Management" section.



SES Identity Management

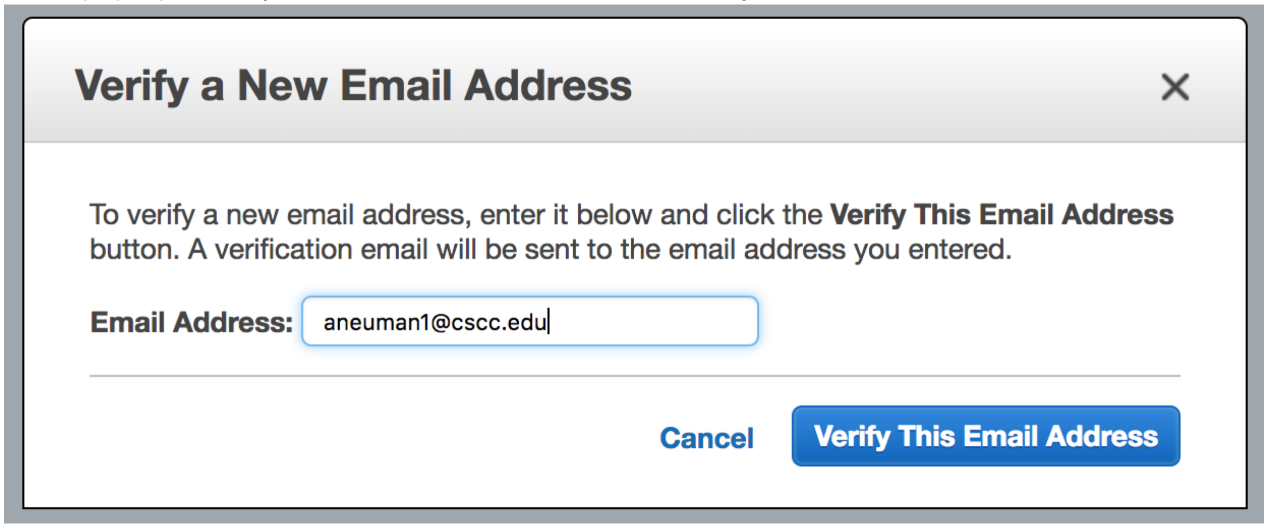
In order to send an email, we'll need to provide a source address. In order to use that address, we have to verify that we own/control it. Similarly, we'll have to verify that the destination address is willing to receive email. To simplify things, we'll use the same email address as the sender and recipient. To begin the verification process, click "Verify a New Email Address".



You have not verified any email addresses.
To verify a email address, click the Verify a New Email Address button above.

Verify a New Email Address

In the pop-up, enter your email address and click the "Verify This Email Address Button".



Verify a New Email Address

To verify a new email address, enter it below and click the **Verify This Email Address** button. A verification email will be sent to the email address you entered.

Email Address: aneuman1@cscc.edu

Enter an Email Address

Check your email account - you should receive an email with a link to verify the address. Click it to complete the verification process.

To send an email, we'll need to create a client using Boto3 in much the same way we did for SimpleDB.

In [151]:
```
ses_client = boto3.client('ses',
                          aws_access_key_id = ACCESS_KEY,
                          aws_secret_access_key = SECRET_KEY,
                          region_name="us-east-1")
```

To send an email, we can use the [*send_email()* (https://boto3.readthedocs.io/en/latest/reference/services/ses.html#SES.Client.send_email)](https://boto3.readthedocs.io/en/latest/reference/services/ses.html#SES.Client.send_email) method, specifying the source and destination addresses as well as the message.

Lab 6 Update the code below to store your email address in the `ADDRESS` variable. After executing the cell, you should receive a test message.

```
In [101]: ADDRESS = ""
CHARSET = "UTF-8"

ses_client.send_email(
    Source=ADDRESS,
    Destination={
        'ToAddresses': [
            ADDRESS
        ]
    },
    Message={
        'Body': {
            'Text': {
                'Charset': CHARSET,
                'Data': "Testing SES"
            }
        },
        'Subject': {
            'Charset': CHARSET,
            'Data': "This is a test"
        }
    }
)
```

```
Out[101]: {'MessageId': '01000162ee28281f-862b375c-74e4-4ee6-903b-93b948b3e61d-0000
00',
 'ResponseMetadata': {'HTTPHeaders': {'content-length': '326',
   'content-type': 'text/xml',
   'date': 'Sun, 22 Apr 2018 16:20:31 GMT',
   'x-amzn-requestid': '141f6a57-4649-11e8-9b8a-3bd3697c786c'},
  'HTTPStatusCode': 200,
  'RequestId': '141f6a57-4649-11e8-9b8a-3bd3697c786c',
  'RetryAttempts': 0}}
```

We can now combine the ability to retrieve data from SimpleDB with our ability to send email using the Simple Email Service, to create and send a report. To do this, we'll create two functions. The first will generate the report and the second will send the report email.

In [152]:
```python
def report(sdb_client):
    email_lines = []
    total = 0

    results = sdb_client.select(SelectExpression="select * from portfolio")
    for item in results['Items']:
        name = item['Name']
        shares = int(sdb_client.get_attributes(DomainName='portfolio',
                                                ItemName=name,
                                                AttributeNames=['shares'])['
        price = float(sdb_client.get_attributes(DomainName='portfolio',
                                                 ItemName=name,
                                                 AttributeNames=['price'])['

        stock_total = shares * price
        message = f"{name}, {shares} shares @ ${price}: ${stock_total}"
        email_lines.append(message)
        total += stock_total

    email_lines.append(f"Total: ${total}")
    return "\n".join(email_lines)

print(report(sdb_client))
```

```
aapl, 5 shares @ $165.72: $828.6
goog, 5 shares @ $1072.96: $5364.8
Total: $6193.400000000001
```

To send the email, we have the following.

In [153]:
```python
def send_report(ses_client, dest_addr, message):
    CHARSET = "UTF-8"

    ses_client.send_email(
        Source=dest_addr,
        Destination={
            'ToAddresses': [
                dest_addr
            ]
        },
        Message={
            'Body': {
                'Text': {
                    'Charset': CHARSET,
                    'Data': message
                }
            },
            'Subject': {
                'Charset': CHARSET,
                'Data': "Report"
            }
        }
    )
```

We can now test report generation and message delivery.

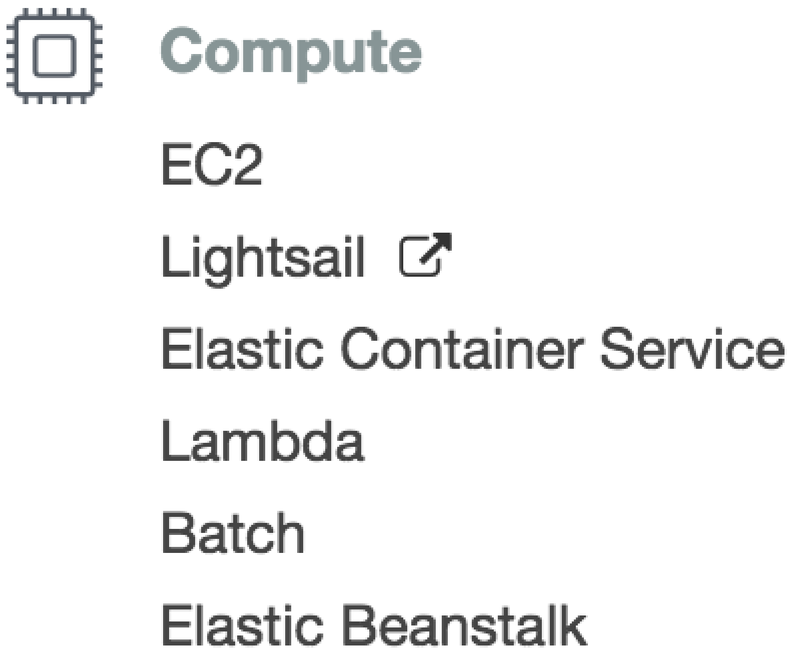
Lab 7 Update the code below with the appropriate email address to test report generation and delivery.

In [127]:
```
ADDRESS = ""
message = report(sdb_client)
send_report(ses_client, ADDRESS, message)
```

Automating the Process

While we could manually execute code when necessary, we can improve reliability by automating as much of the process as possible. There are a variety of way of automating execution including creating scheduled tasks in Windows (https://msdn.microsoft.com/en-us/library/windows/desktop/aa383614%28v=vs.85%29.aspx) and using cron (https://en.wikipedia.org/wiki/Cron) in Unix-like environments. For our reporting job, we'll use AWS Lambda (https://aws.amazon.com/lambda/), a service that allows us to execute code in the cloud without having to worry about managing servers.

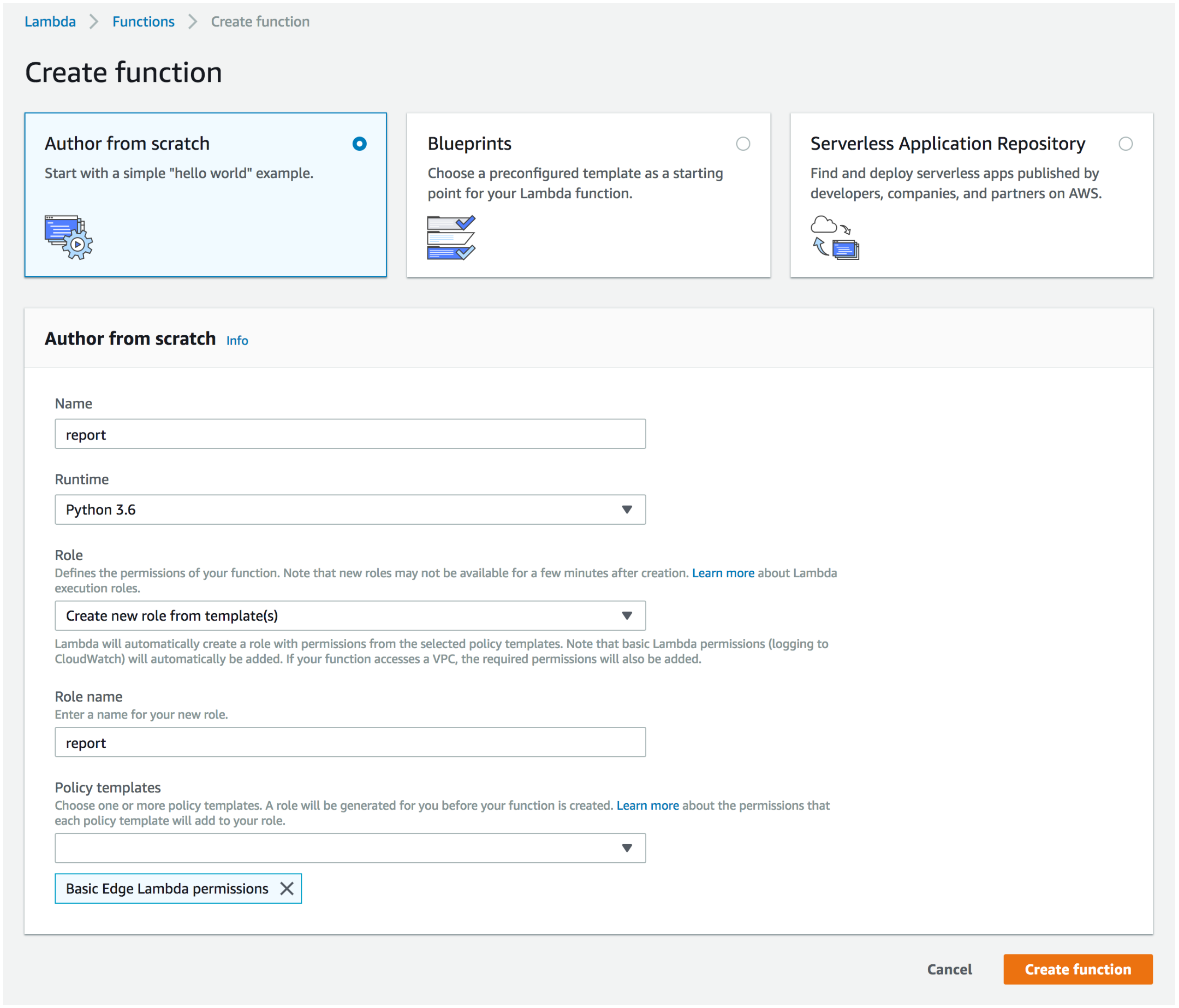
While we could programmatically set up Lambda using Boto, it is much simpler to to it via the console. From the Services menu, select "Lambda" in the "Compute" section.



Compute Services

When the Lambda page loads, make sure that your region is set to "US East (N. Virginia)". Click the "Create a function" button to get started. We'll author a function from scratch. Set the following values for the fields and click "Create function".

- Name: report
- Runtime: Python 3.6
- Role: Create new role from template(s)
- Role name: report
- Policy templates: Basic Edge Lambda permissions


Lambda > Functions > Create function

Create function

Author from scratch [✓]
Start with a simple "hello world" example.

Blueprints
Choose a preconfigured template as a starting point for your Lambda function.

Serverless Application Repository
Find and deploy serverless apps published by developers, companies, and partners on AWS.

Author from scratch Info

Name
report

Runtime
Python 3.6

Role
Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. Learn more about Lambda execution roles.
Create new role from template(s)
Lambda will automatically create a role with permissions from the selected policy templates. Note that basic Lambda permissions (logging to CloudWatch) will automatically be added. If your function accesses a VPC, the required permissions will also be added.

Role name
Enter a name for your new role.
report

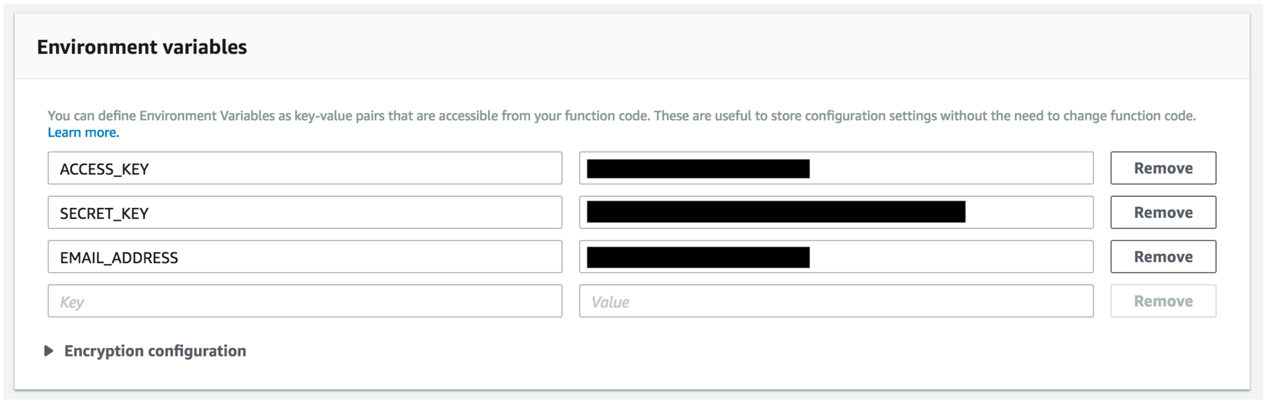
Policy templates
Choose one or more policy templates. A role will be generated for you before your function is created. Learn more about the permissions that each policy template will add to your role.
Basic Edge Lambda permissions

Cancel Create function

New Lambda Function

There are several sections of configuration items; we'll look at a few of them. First, look for the "Environment Variables" section. We can used environment variables to provide data to our function when it runs while avoiding hard-coding the data in the function itself. For example. we can store the AWS keys and the email address that the report will be sent to as environment variables. Create the following three keys and enter the appropriate values.

- ACCESS_KEY
- SECRET_KEY
- EMAIL_ADDRESS


Environment variables

You can define Environment Variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more.

ACCESS_KEY	[redacted]	Remove
SECRET_KEY	[redacted]	Remove
EMAIL_ADDRESS	[redacted]	Remove
Key	Value	Remove

▶ Encryption configuration

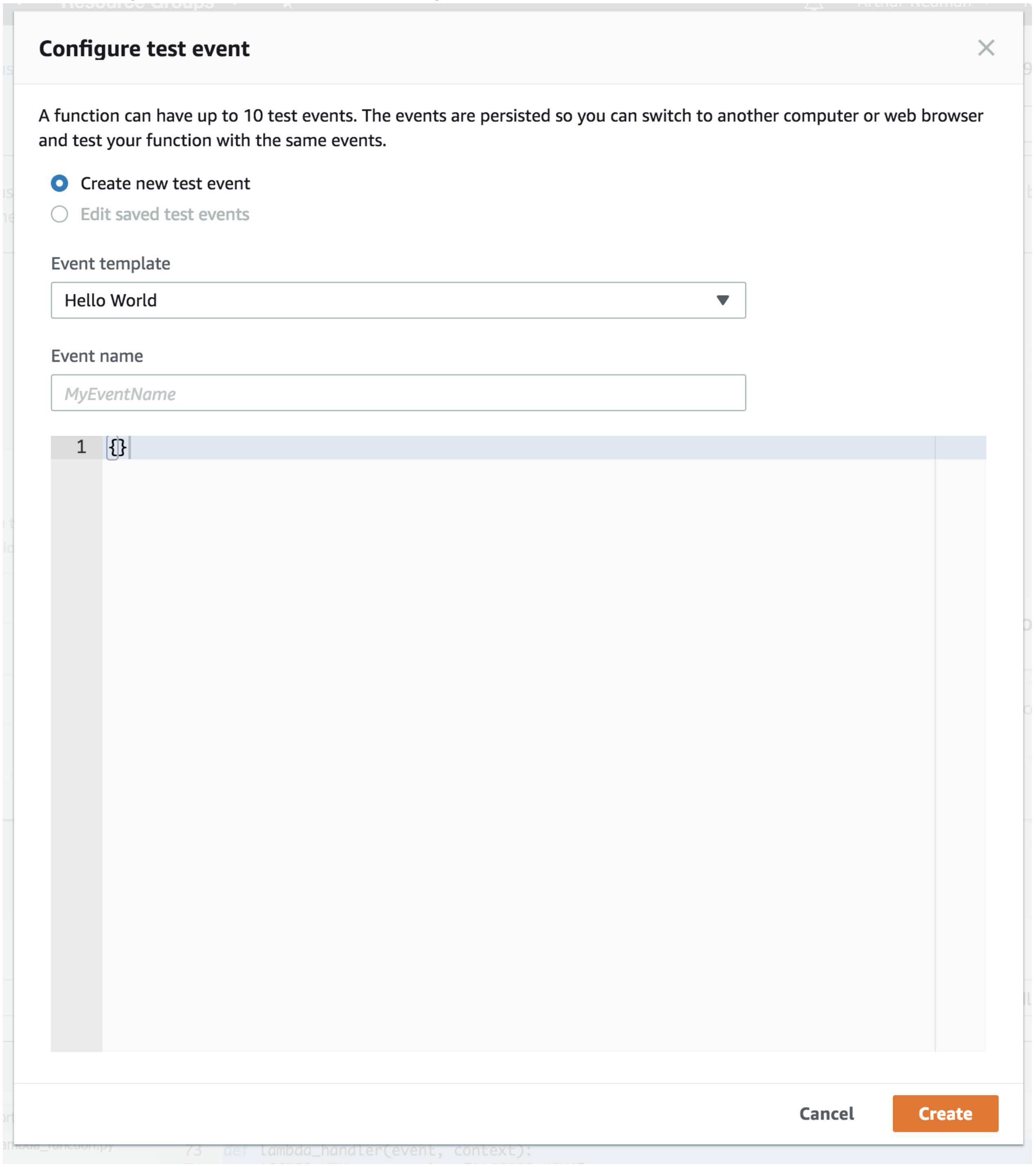
Environment Variables

We'll now add code for our function using the code editor in the "Function code" section. Our code will be based on the *update_portfolio()*, *report()*, and *send_report()* functions we wrote earlier.

```python
import os
import boto3
from botocore.vendored import requests


def update_portfolio(sdb_client):
    results = sdb_client.select(SelectExpression="select * from port
folio")
    for item in results['Items']:
        name = item['Name']
        url = "https://api.iextrading.com/1.0/stock/{0}/price".forma
t(name)
        price = requests.get(url).json()
        sdb_client.put_attributes(
            DomainName='portfolio',
            ItemName=name,
            Attributes=[
                {
                    'Name': 'price',
                    'Value': str(price),
                    'Replace': True
                }
        ])


def report(sdb_client):
    email_lines = []
    total = 0

    results = sdb_client.select(SelectExpression="select * from port
folio")
    for item in results['Items']:
        name = item['Name']
        shares = int(sdb_client.get_attributes(DomainName='portfoli
o',
                                               ItemName=name,
                                               AttributeNames=['shar
es'])['Attributes'][0]['Value'])
        price = float(sdb_client.get_attributes(DomainName='portfoli
o',
                                                ItemName=name,
                                                AttributeNames=['pri
ce'])['Attributes'][0]['Value'])
        stock_total = shares * price
        message = f"{name}, {shares} shares @ ${price}: ${stock_tota
l}"
        email_lines.append(message)
        total += stock_total
    email_lines.append(f"Total: ${total}")
    return "\n".join(email_lines)


def send_report(ses_client, dest_addr, message):
    CHARSET = "UTF-8"

    ses_client.send_email(
        Source=dest_addr,
        Destination={
            'ToAddresses': [
                dest_addr
            ]
        },
        Message={
            'Body': {
                'Text': {
                    'Charset': CHARSET,
                    'Data': message
                }
            },
            'Subject': {
                'Charset': CHARSET,
                'Data': "Report"
            }
        }
    )


def lambda_handler(event, context):
    ACCESS_KEY = os.environ['ACCESS_KEY']
    SECRET_KEY = os.environ['SECRET_KEY']
    EMAIL_ADDRESS = os.environ['EMAIL_ADDRESS']

    sdb_client = boto3.client('sdb',
                              aws_access_key_id = ACCESS_KEY,
                              aws_secret_access_key = SECRET_KEY,
                              region_name="us-east-1")

    ses_client = boto3.client('ses',
                              aws_access_key_id = ACCESS_KEY,
                              aws_secret_access_key = SECRET_KEY,
                              region_name="us-east-1")

    update_portfolio(sdb_client)
    message = report(sdb_client)
    send_report(ses_client, EMAIL_ADDRESS, message)
```

The *lambda_handler()* function serves as the entry point for the code. We start by loading data from the environment variables we set previously. Next, we create the Boto3 clients. Finally, we execute the functions we created earlier in order.

To test the code, first click "Save" then click "Test" in the upper right corner of the console. This will prompt us to define a new test event; we can create a new test event with the with no initial data. After defining the test event, click "Test" again.



Configure test event

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

[✓ Create new test event]
Edit saved test events

Event template: Hello World

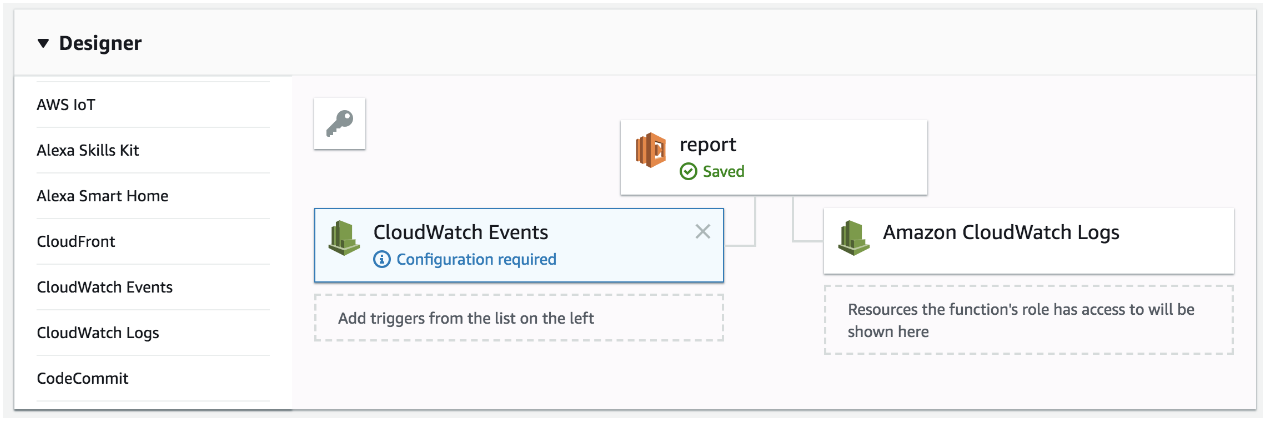
Event name: MyEventName

```
{}
```

New Test Event

If everything is configured properly, you should see a success message in the AWS console and receive a report email.

Finally, to automate the function, we need to schedule its execution. In the "Designer" select "CloudWatch Events". Below report, a "CloudWatch Events" item appears and is selected - we can navigate back to the report function to configure its properties if necessary.

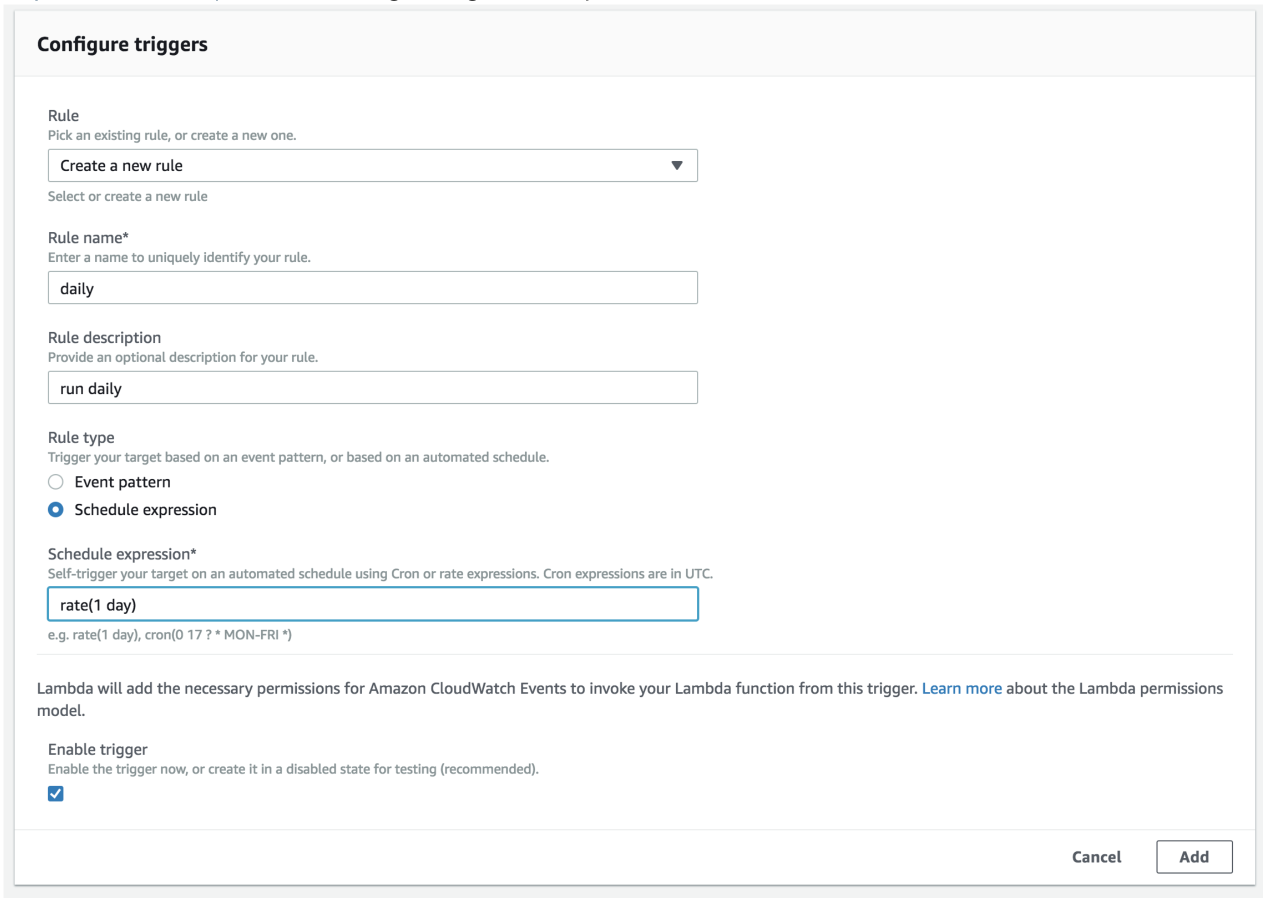


Add triggers from the list on the left

Resources the function's role has access to will be shown here

CloudWatch Events

In the "Configure triggers' section, we can create a new rule based on a scheduled expression to run the Lambda function as frequently as we'd like; the image below shows the configuration for daily execution. See the [AWS Lambda documentation (https://docs.aws.amazon.com/lambda/latest/dg/tutorial-scheduled-events-schedule-expressions.html)](https://docs.aws.amazon.com/lambda/latest/dg/tutorial-scheduled-events-schedule-expressions.html) for details regarding valid expressions.



Configure triggers

Rule
Pick an existing rule, or create a new one.
Create a new rule
Select or create a new rule

Rule name*
Enter a name to uniquely identify your rule.
daily

Rule description
Provide an optional description for your rule.
run daily

Rule type
Trigger your target based on an event pattern, or based on an automated schedule.
○ Event pattern
● Schedule expression

Schedule expression*
Self-trigger your target on an automated schedule using Cron or rate expressions. Cron expressions are in UTC.
rate(1 day)
e.g. rate(1 day), cron(0 17 ? * MON-FRI *)

Lambda will add the necessary permissions for Amazon CloudWatch Events to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.

Enable trigger
Enable the trigger now, or create it in a disabled state for testing (recommended).
☑

CloudWatch Event Trigger

Lab Answers

1. Values vary
2.
```
sdb_client.domain_metadata(DomainName='test_domain')
```
3.
```
sdb_client.put_attributes(DomainName='test_domain',
                          ItemName='employee2',
                          Attributes=[
                              {
                                  'Name': 'Fullname',
                                  'Value': 'Jane Doe'
                              },
                              {
                                  'Name': 'Department',
                                  'Value': 'IT'
                              }
                          ])
```
4.
```
sdb_client.select(SelectExpression="select * from test_domain where Fullname = 'Bob Smith'")
```
5.
```
sdb_client.delete_attributes(DomainName='test_domain', ItemName="employee2")
```
6. Values vary
7. Values vary

Next Steps

Automation should be a goal for most of the tasks we undertake - even complicated ones. While we've only looked at automating a simple task of retrieving text data, creating a text report, and emailing the report as plain text, it is possible to automate more complicated analytics tasks, generate a variety of outputs, and to use the outputs to create more than just emails - we could update dashboards, for example. Continue to explore different data sources and think of what information could be extracted from the data and how that information could be conveyed to interested parties. Try to automate these processes.

Resources

- Automate the Boring Stuff with Python by Al Sweigart (https://automatetheboringstuff.com/)
- Azure for Python Developers (https://docs.microsoft.com/en-us/python/azure/?view=azure-python)
- Boto3 Code Examples (https://boto3.readthedocs.io/en/latest/guide/examples.html)
- Google Cloud Platform: Using the Python Client Library (https://cloud.google.com/compute/docs/tutorials/python-guide)

Exercise

Modify the reporting functions to store and report on several days worth of stock data rather than just the latest price data. Test your code.

```
In [ ]:
```