GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 4813A Spring 2021
Lab: Serverless Photo Gallery Application using AWS Lambda, API Gateway, S3, DynamoDB and Cognito

---

References:

[1] A. Bahga, V. Madisetti, "Cloud Computing Solutions Architect: A Hands-On Approach", ISBN: 978-0996025591

[2] https://aws.amazon.com/documentation/

Due Date:

The lab report will be **due on February 17, 2021.**

This lab is about creating a serverless Photo Gallery application. ***Make sure your AWS region is set for us-east-1 or N. Virginia (located at the top right of the navigation bar)**

The application uses the following:

1. A web interface implemented in HTML & JS, which can be served through S3 static website hosting
2. AWS API Gateway endpoints
3. Lambda functions
4. Cognito for user pool management
5. DynamoDB for storing records of photos
6. S3 for storing photos

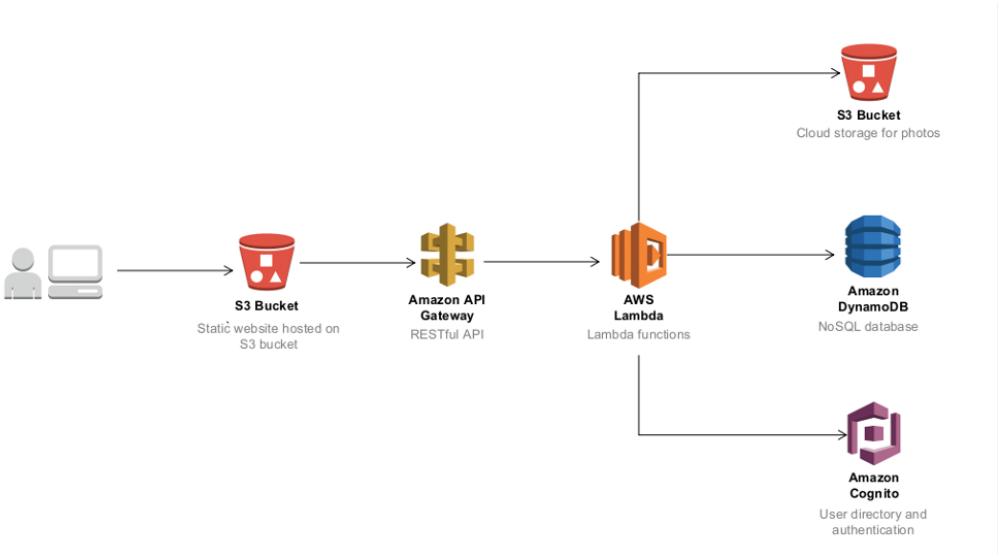


Fig. Architecture diagram of the Photo Gallery application showing AWS services used

Follow the steps below to set up the Photo Gallery application in your AWS account.

1. Create an S3 Bucket for hosting the static website for the application

- Create a new S3 bucket for hosting the static website and enable static website hosting for the bucket. **Uncheck Block all public access under Bucket setting for Block Public Access section**
- The bucket name **must be unique**. Click here to know the rules. For the name of the bucket, use **staticwebsite-lastname-year-courseNumber** (e.g., **staticwebsite-corporan-2021-4813**)
- Add a bucket policy below to enable public access to the photos uploaded. Replace '**mybucketname**' with the name of the S3 bucket created.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::mybucketname/*"
        }
    ]
}
```

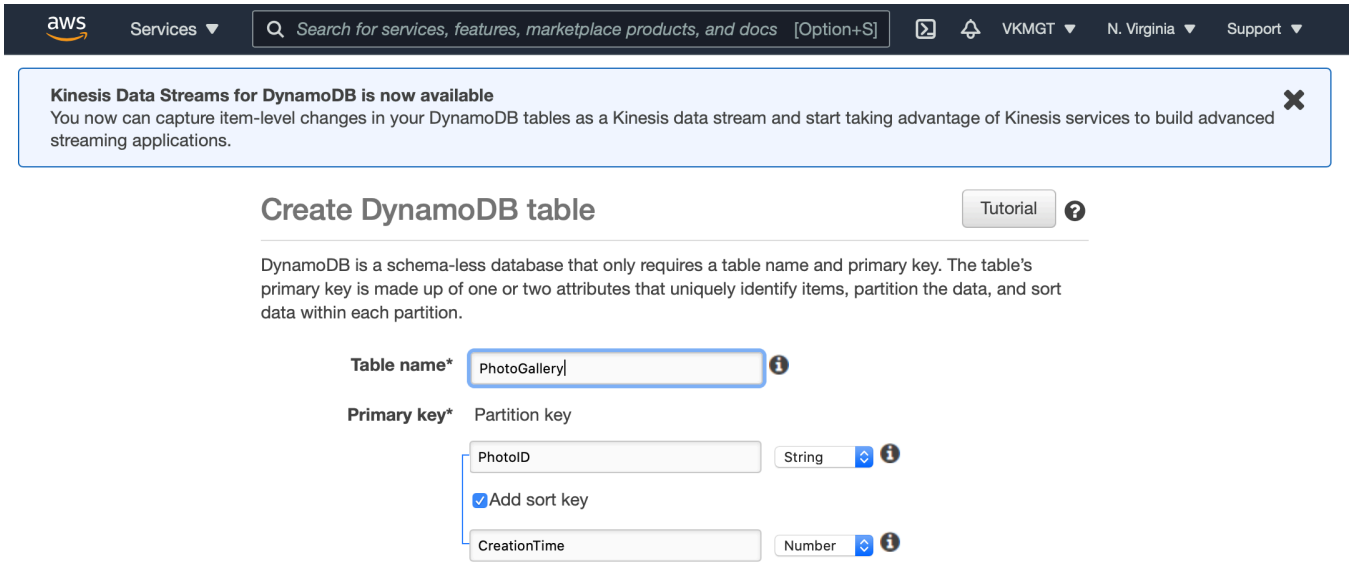
2. Create an S3 Bucket for storing photos

- Create a new S3 bucket for storing photos. **Uncheck Block all public access under Bucket setting for Block Public Access section**
- For the name of the bucket, use **photobucket-lastname-year-courseNumber** (e.g., **photobucket-corporan-2021-4813**)
- Create a folder named 'photos' in this bucket.
- Add a bucket policy below to enable public access to the photos uploaded. Replace '**mybucketname**' with the name of the S3 bucket created.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::mybucketname/photos/*"
        }
    ]
}
```

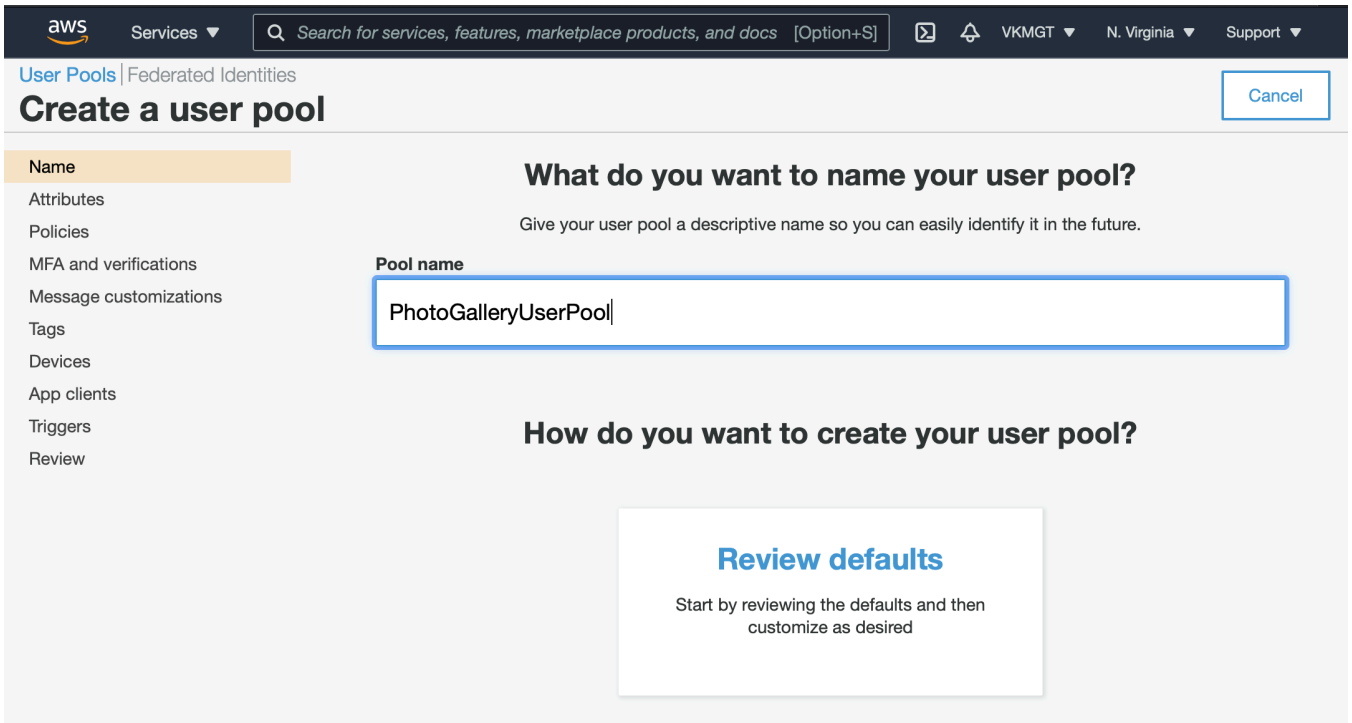
3. Create a DynamoDB table for storing records of photos

- Create a DynamoDB table named **PhotoGallery** with a partition key and a sort key named **PhotoID** and **CreationTime**, respectively, as shown below.

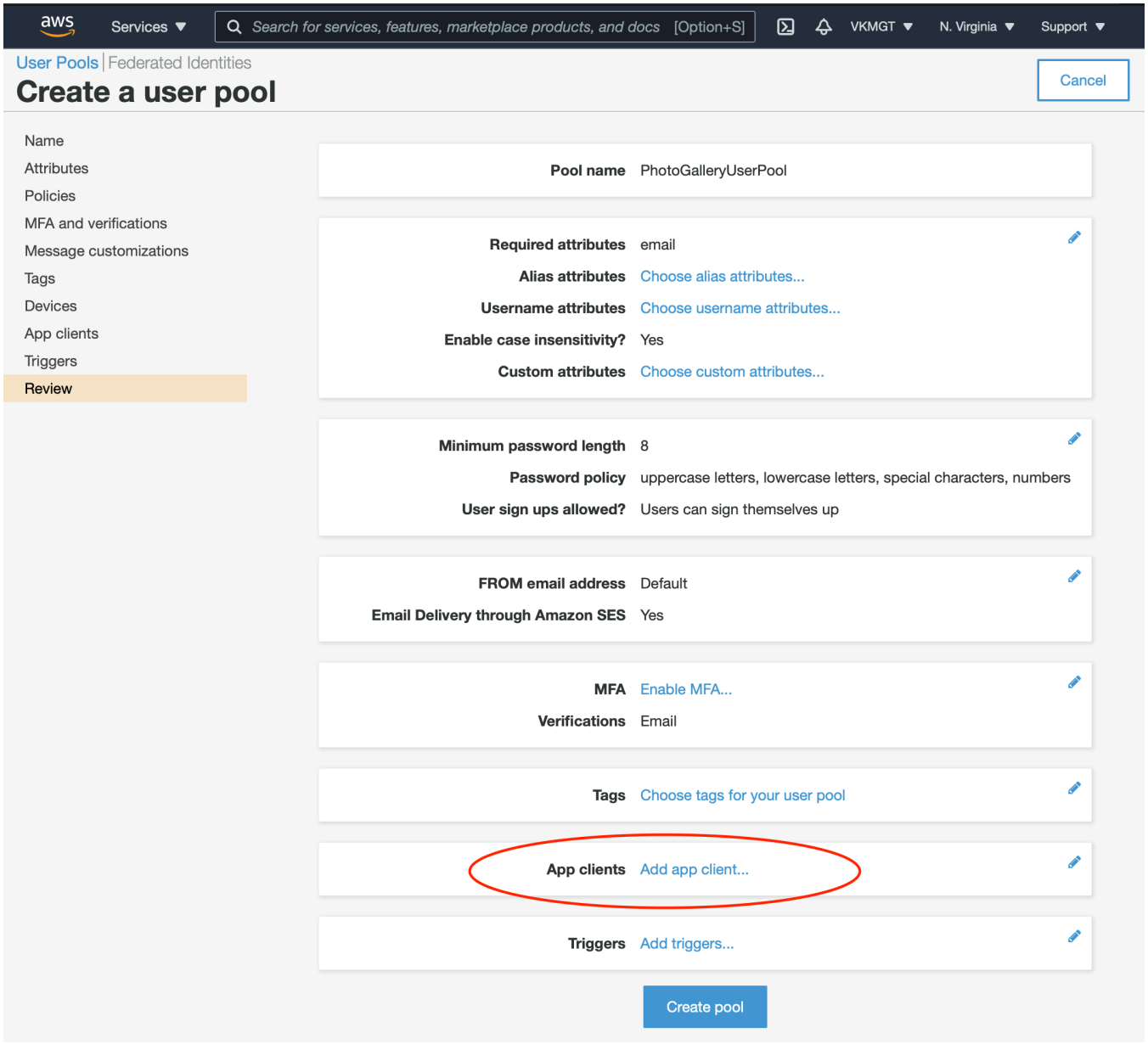


4. Create Cognito User Pool

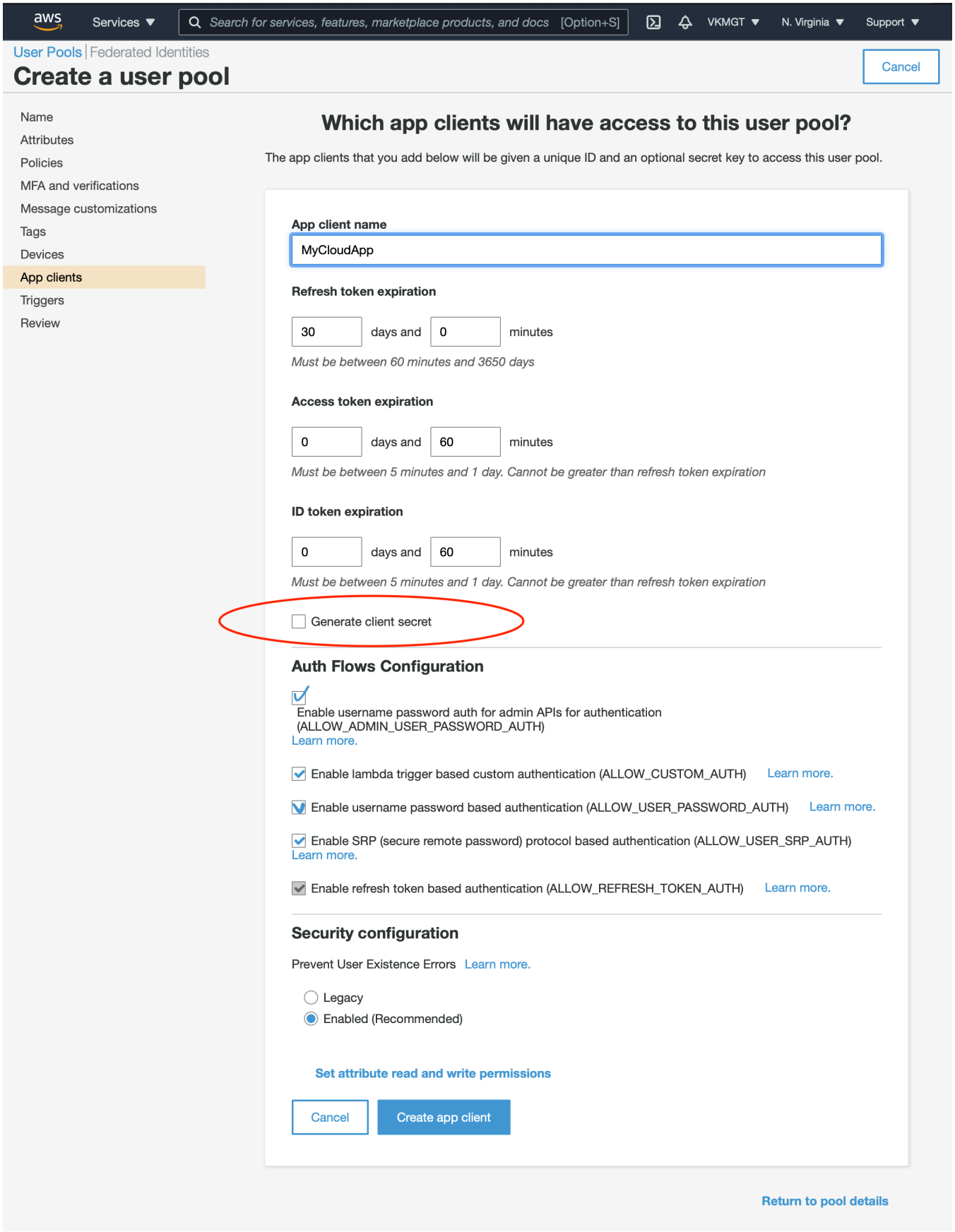
- Navigate to the AWS Cognito console
- Create a new **User Pool** called **PhotoGalleryUserPool** as shown below. Click "**Review defaults**"



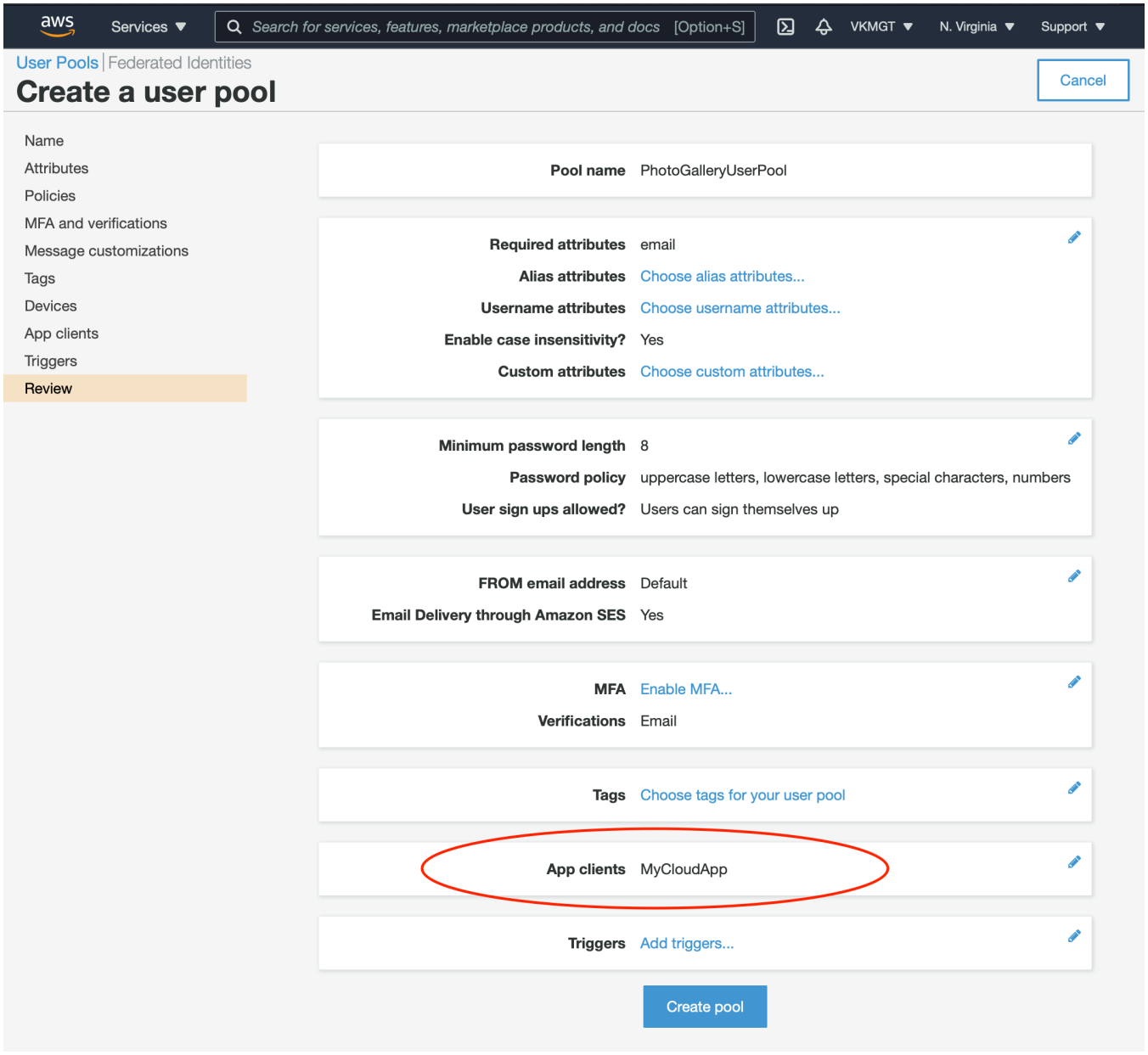
Click “**add app client**” under **App Clients**, as shown below



Click “**Add an app client**” and name it **MyCloudApp**. Uncheck “**Generate Client Secret**”, **enable all** the authentication checkbox, and click “**Create app client**” as shown below

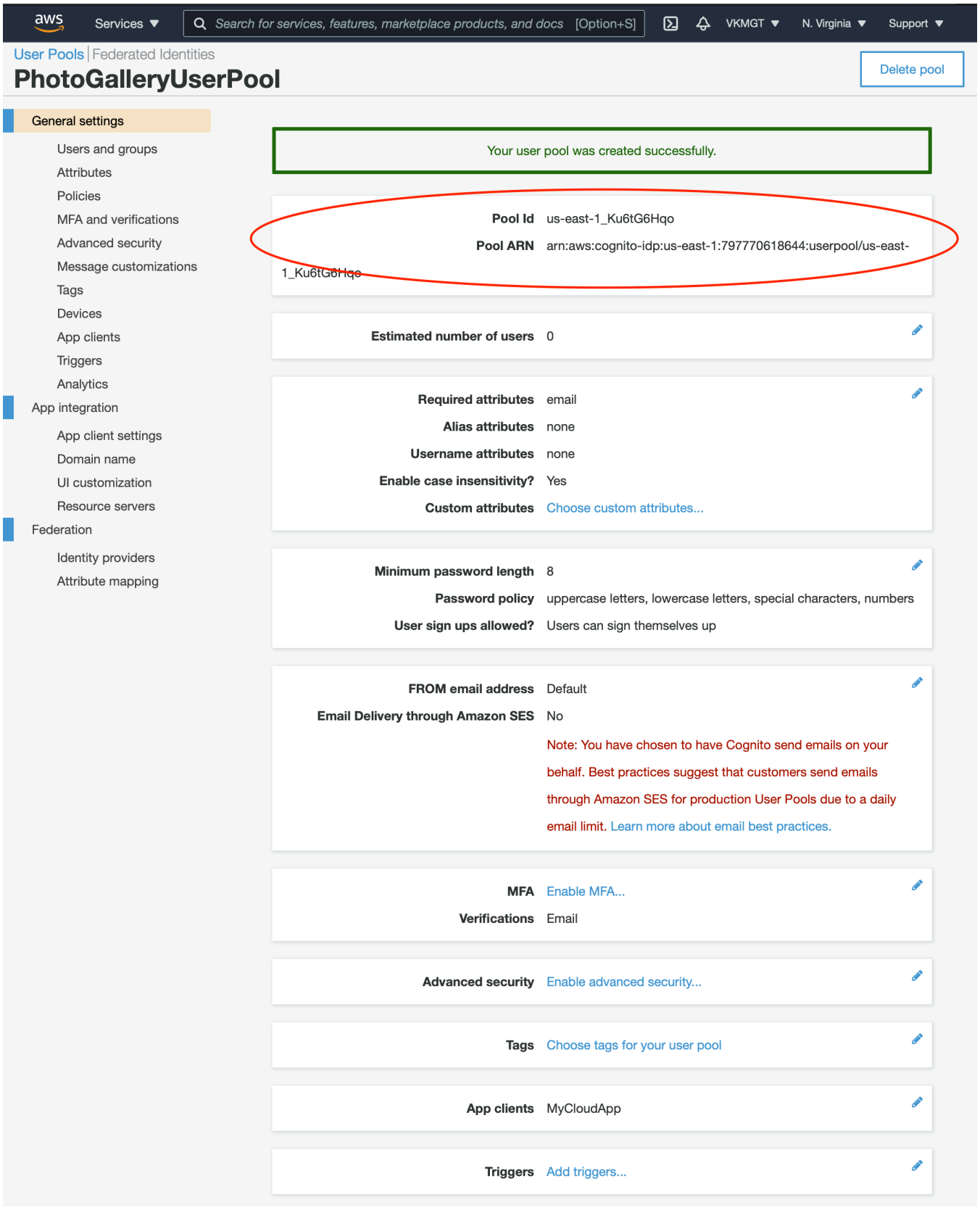


After creating the app client, return to the pool details by clicking "**Return to pool details**". The new app client should be included int eh **App Client** section, as shown below

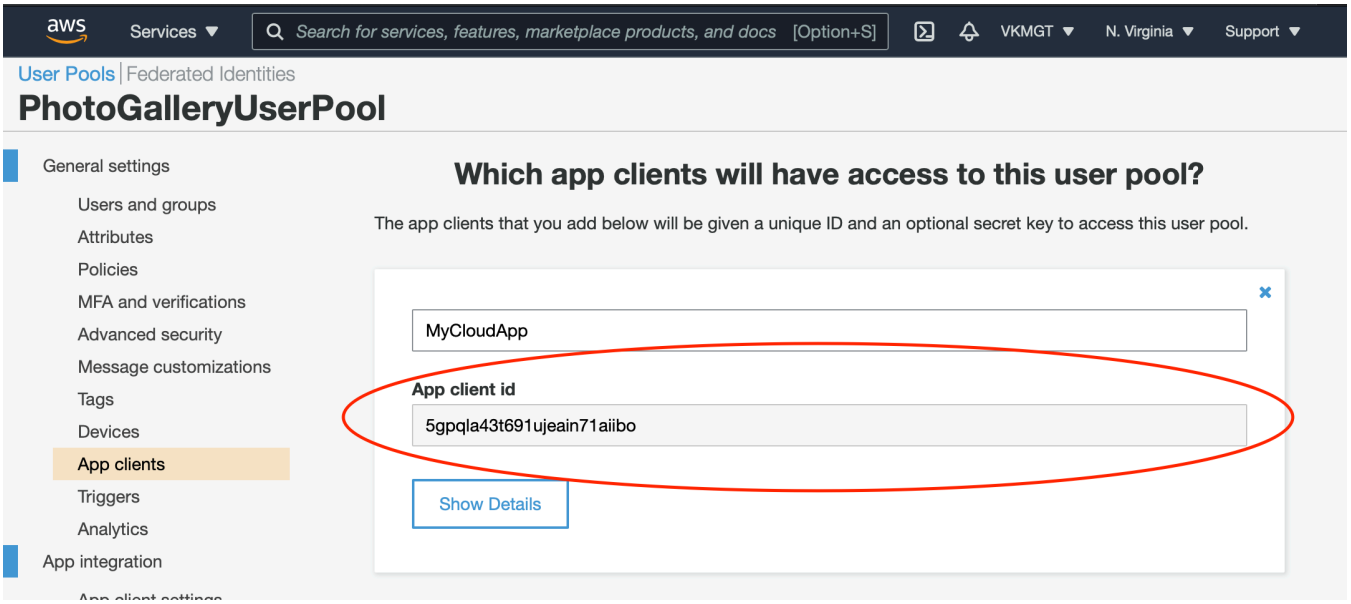


Create the User Pool by clicking "**Create Pool**".

After creating the pool, save the **Pool Id** for later; we will use this identifier for our serverless function.

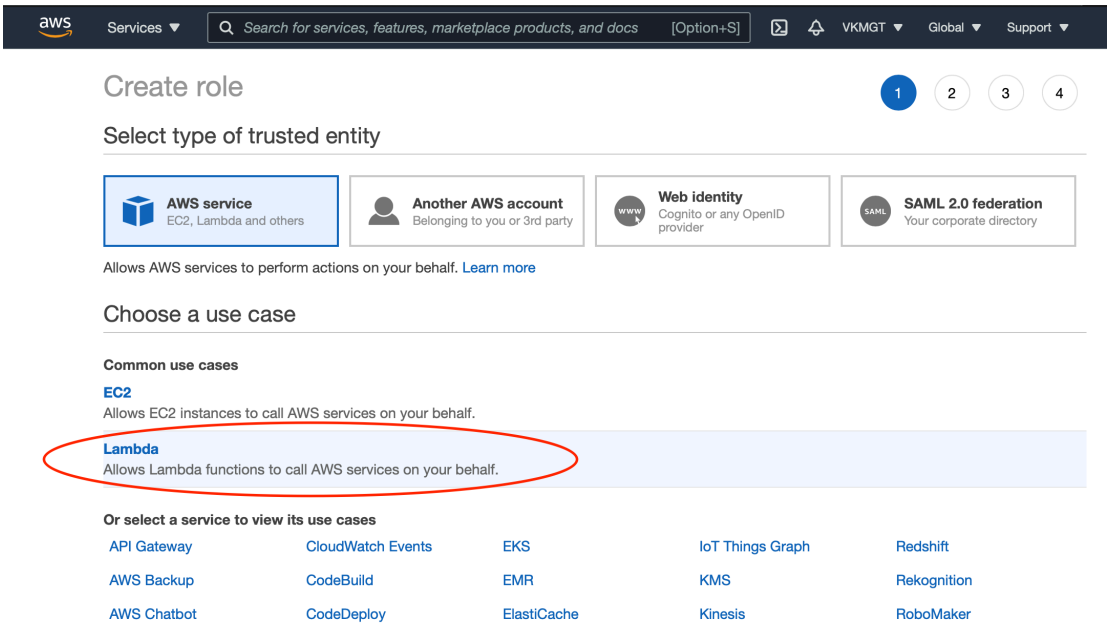


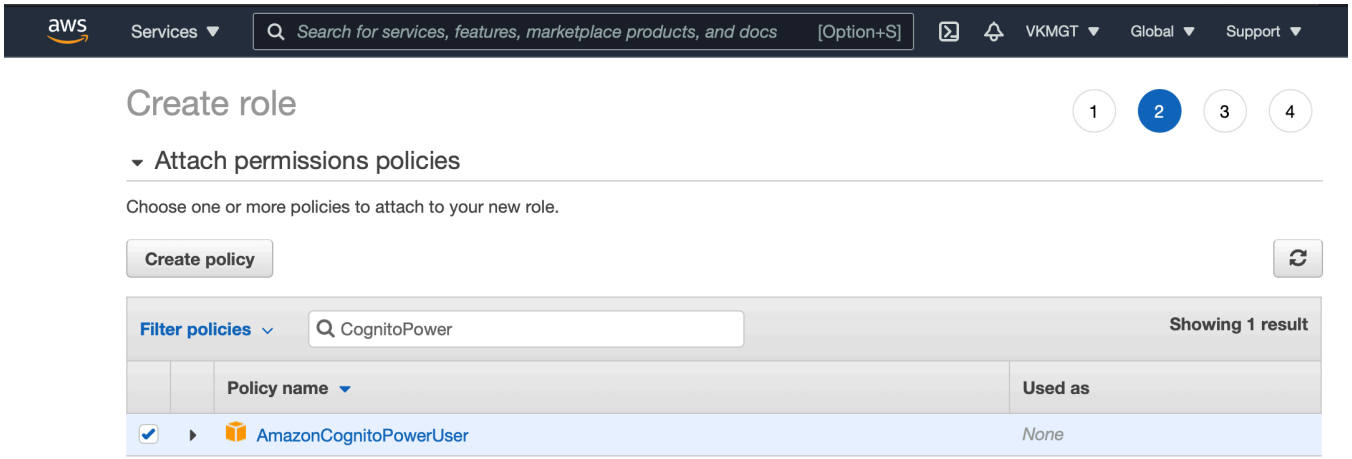
From the left column, click **App Clients**. Under the new app client (**MyCloudApp**), copy and save the **App client id**; we will use this identifier for our serverless function.

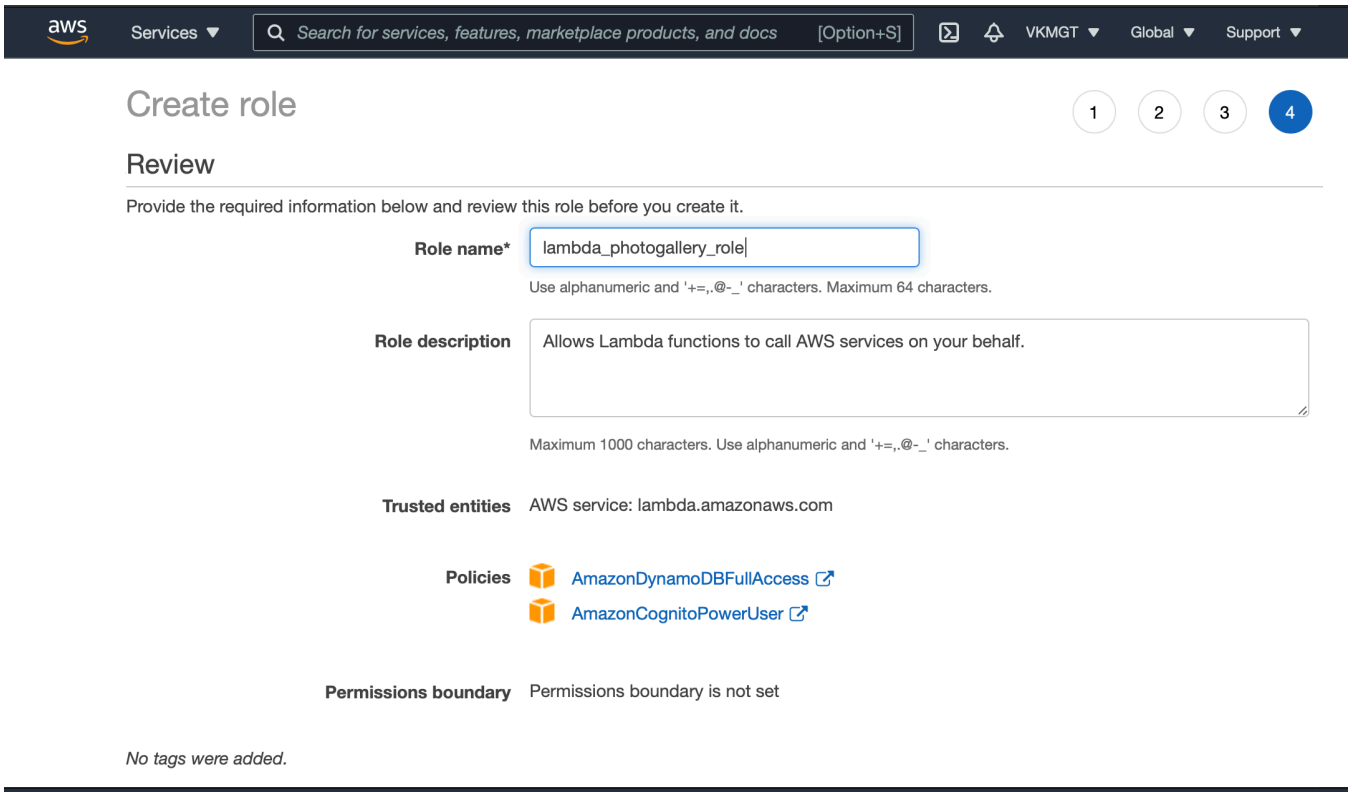


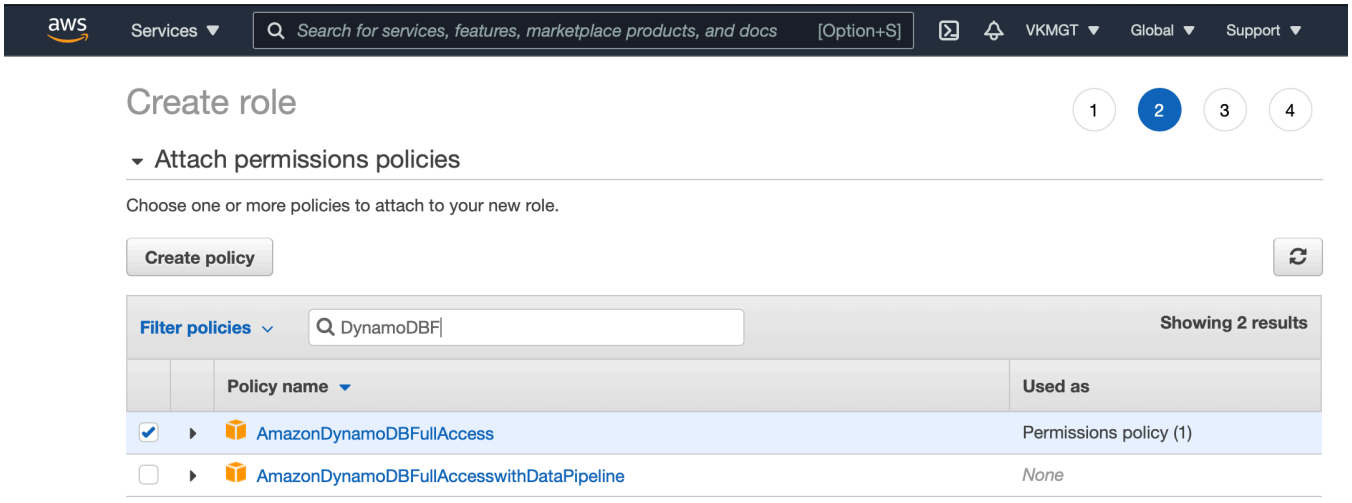
5. Create IAM Roles

- In the AWS IAM console, create a new IAM role called '**lambda_photogallery_role**' for Lambda service, as shown below. Attach policy called **AmazonDynamoDBFullAccess** to this role. Then attach policy called **AmazonCognitoPowerUser** to this role.

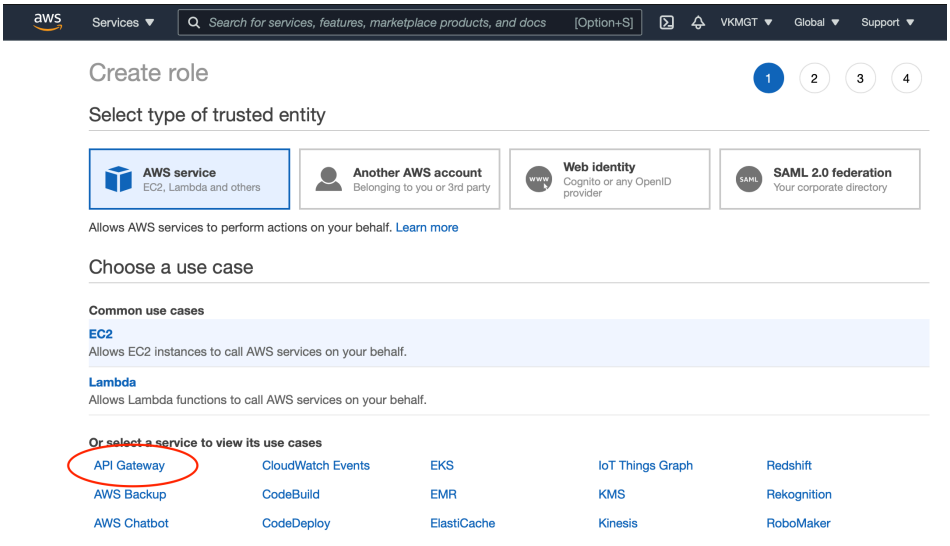


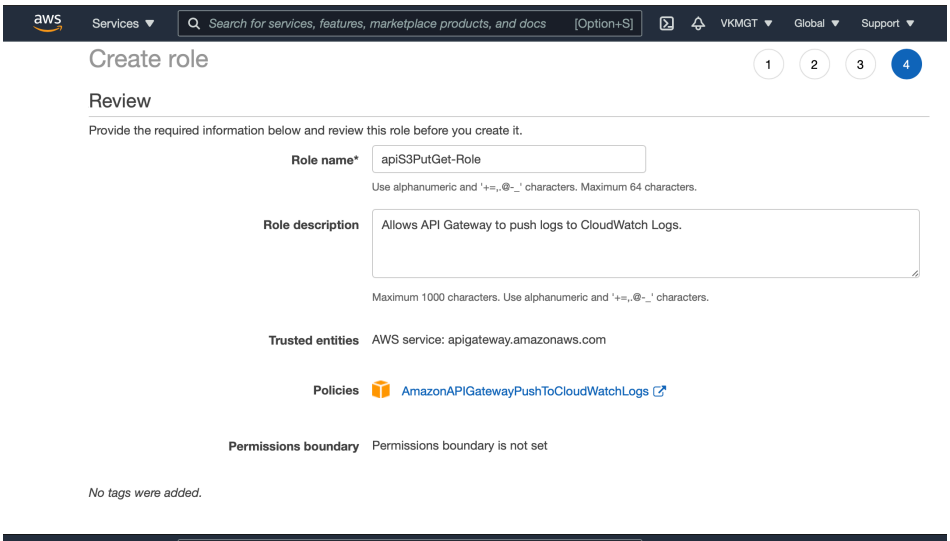

aws
Services
Search for services, features, marketplace products, and docs [Option+S]
VKMGT
Global
Support
Create role
1 2 3 4
Attach permissions policies
Choose one or more policies to attach to your new role.
Create policy
Filter policies
CognitoPower
Showing 1 result
Policy name
Used as
AmazonCognitoPowerUser
None

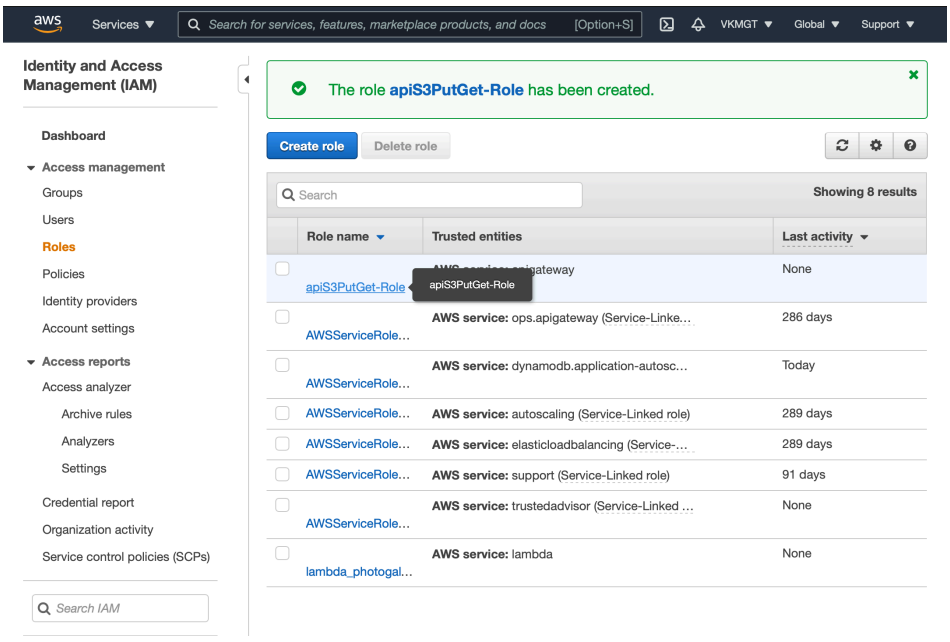

aws
Services
Search for services, features, marketplace products, and docs [Option+S]
VKMGT
Global
Support
Create role
1 2 3 4
Review
Provide the required information below and review this role before you create it.
Role name*
lambda_photogallery_role
Use alphanumeric and '+=,.@-_' characters. Maximum 64 characters.
Role description
Allows Lambda functions to call AWS services on your behalf.
Maximum 1000 characters. Use alphanumeric and '+=,.@-_' characters.
Trusted entities
AWS service: lambda.amazonaws.com
Policies
AmazonDynamoDBFullAccess
AmazonCognitoPowerUser
Permissions boundary
Permissions boundary is not set
No tags were added.

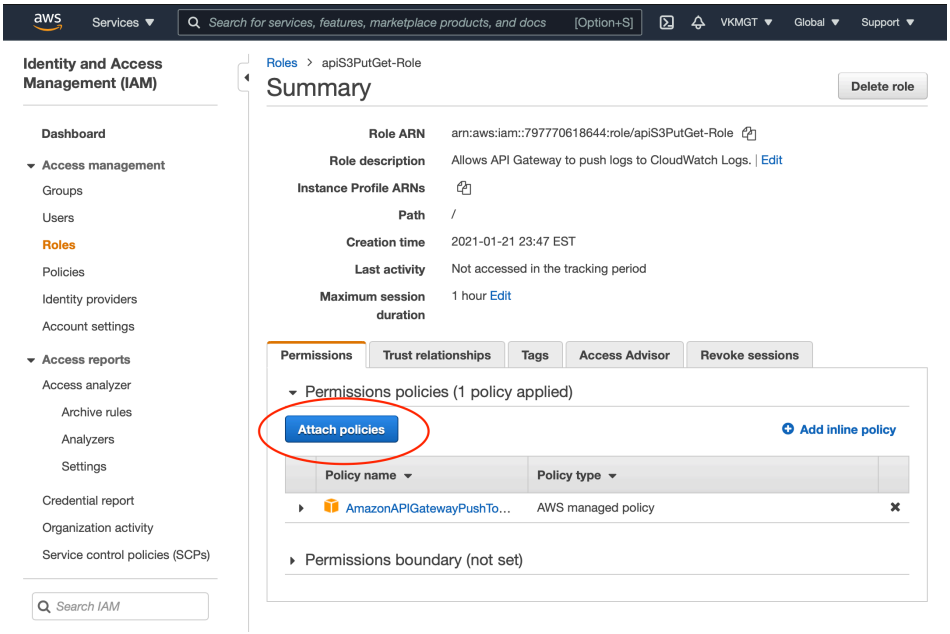

aws
Services
Search for services, features, marketplace products, and docs [Option+S]
VKMGT
Global
Support
Create role
1 2 3 4
Attach permissions policies
Choose one or more policies to attach to your new role.
Create policy
Filter policies
DynamoDBF
Showing 2 results
Policy name
Used as
AmazonDynamoDBFullAccess
Permissions policy (1)
AmazonDynamoDBFullAccesswithDataPipeline
None

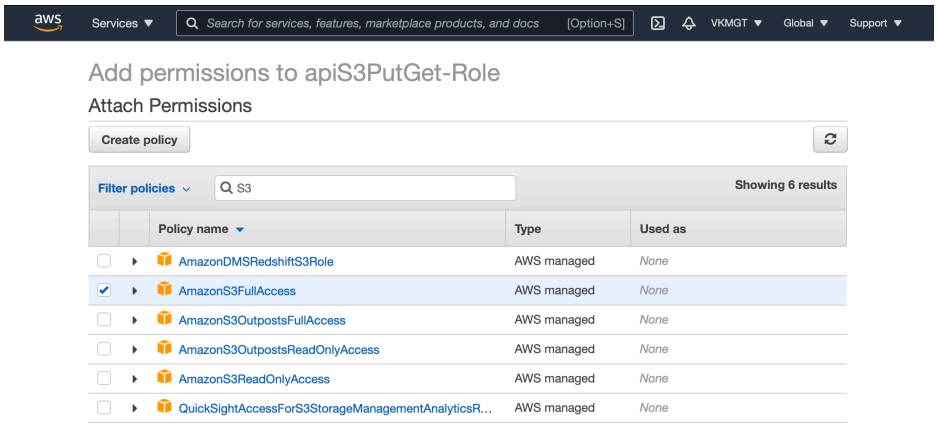
- Create another IAM role called '**apiS3PutGet-Role**' to upload the photos to the S3 Bucket service, as shown below. Attach policy called **AmazonAPIGatewayPushToCloudWatchLogs** to this role. Then attach policy called **AmazonS3FullAccess** to this role.

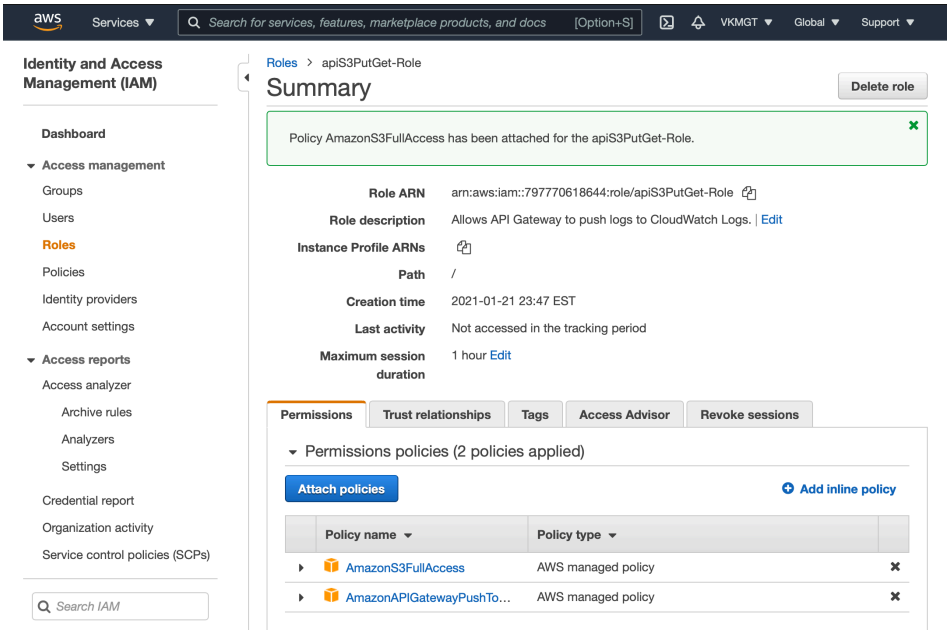






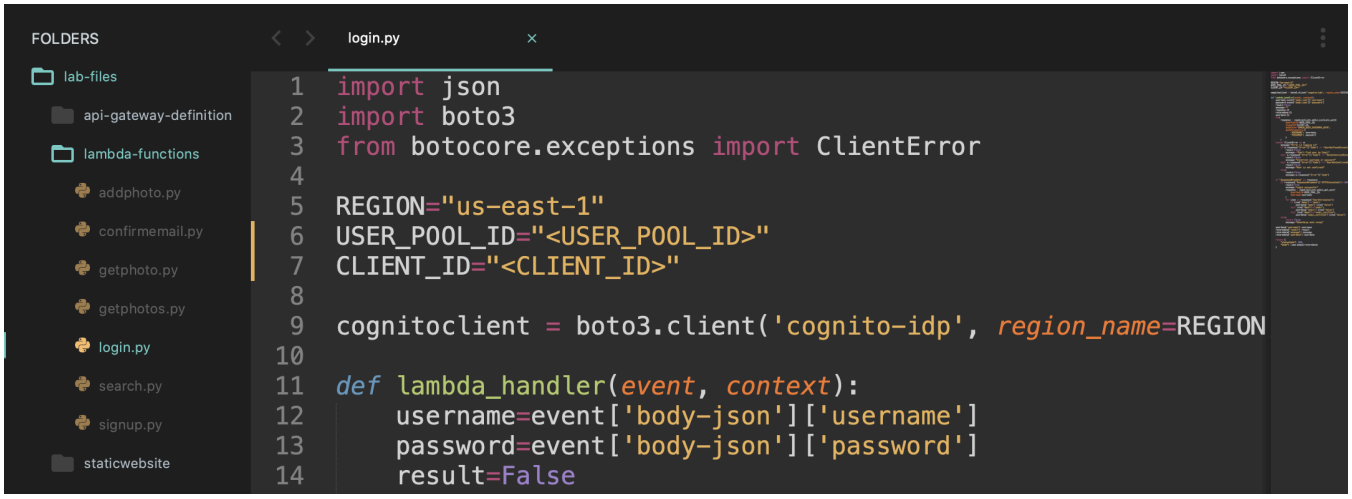

Services
Search for services, features, marketplace products, and docs [Option+S]
VKMGT
Global
Support
Identity and Access Management (IAM)
Dashboard
Access management
Groups
Users
Roles
Policies
Identity providers
Account settings
Access reports
Access analyzer
Archive rules
Analyzers
Settings
Credential report
Organization activity
Service control policies (SCPs)
Search IAM
Roles > apiS3PutGet-Role
Summary
Delete role
Role ARN arn:aws:iam::797770618644:role/apiS3PutGet-Role
Role description Allows API Gateway to push logs to CloudWatch Logs. | Edit
Instance Profile ARNs
Path /
Creation time 2021-01-21 23:47 EST
Last activity Not accessed in the tracking period
Maximum session duration 1 hour Edit
Permissions
Trust relationships
Tags
Access Advisor
Revoke sessions
Permissions policies (1 policy applied)
Attach policies
Add inline policy
Policy name
Policy type
AmazonAPIGatewayPushTo…
AWS managed policy
Permissions boundary (not set)

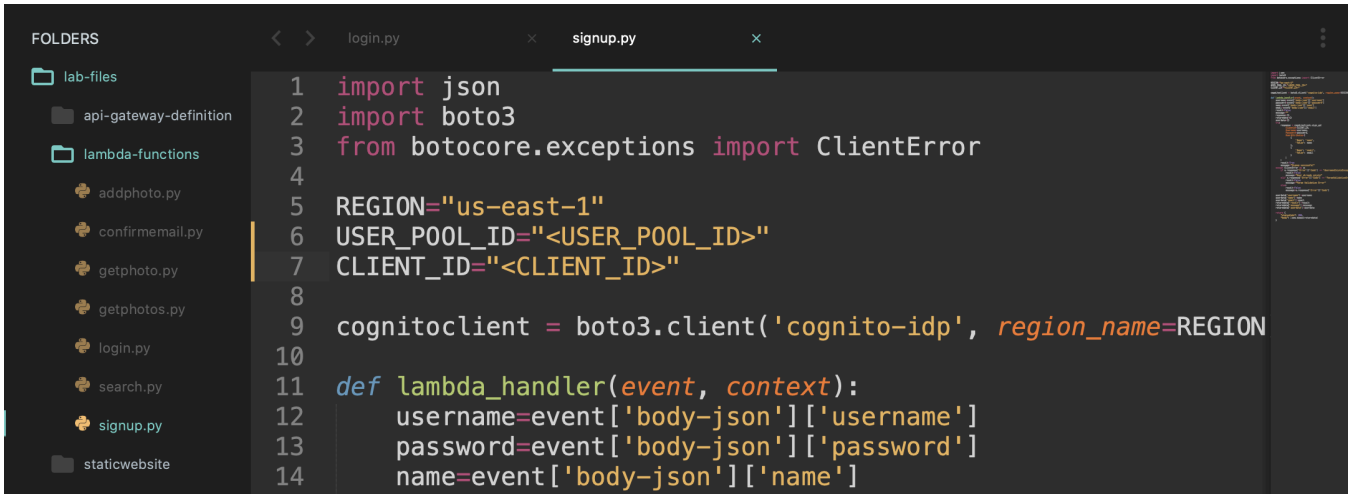

Services
Search for services, features, marketplace products, and docs [Option+S]
VKMGT
Global
Support
Add permissions to apiS3PutGet-Role
Attach Permissions
Create policy
Filter policies
S3
Showing 6 results
Policy name
Type
Used as
AmazonDMSRedshiftS3Role AWS managed None
AmazonS3FullAccess AWS managed None
AmazonS3OutpostsFullAccess AWS managed None
AmazonS3OutpostsReadOnlyAccess AWS managed None
AmazonS3ReadOnlyAccess AWS managed None
QuickSightAccessForS3StorageManagementAnalyticsR… AWS managed None


Services
Search for services, features, marketplace products, and docs [Option+S]
VKMGT
Global
Support
Identity and Access Management (IAM)
Dashboard
Access management
Groups
Users
Roles
Policies
Identity providers
Account settings
Access reports
Access analyzer
Archive rules
Analyzers
Settings
Credential report
Organization activity
Service control policies (SCPs)
Search IAM
Roles > apiS3PutGet-Role
Summary
Delete role
Policy AmazonS3FullAccess has been attached for the apiS3PutGet-Role.
Role ARN arn:aws:iam::797770618644:role/apiS3PutGet-Role
Role description Allows API Gateway to push logs to CloudWatch Logs. | Edit
Instance Profile ARNs
Path /
Creation time 2021-01-21 23:47 EST
Last activity Not accessed in the tracking period
Maximum session duration 1 hour Edit
Permissions
Trust relationships
Tags
Access Advisor
Revoke sessions
Permissions policies (2 policies applied)
Attach policies
Add inline policy
Policy name
Policy type
AmazonS3FullAccess AWS managed policy
AmazonAPIGatewayPushTo… AWS managed policy

6. Review Code

Under the **lambda-functions** directory, open **login.py**, **signup.py**, and **confirmemail.py**; update the <USER_POOL_ID> and the <CLIENT_ID>, in lines 6 and 7, respectively, with the ones you just created.



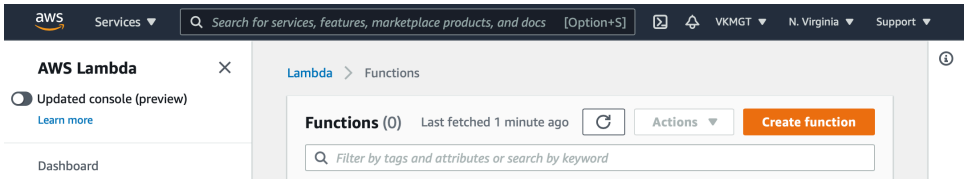


7. Create Lambda functions

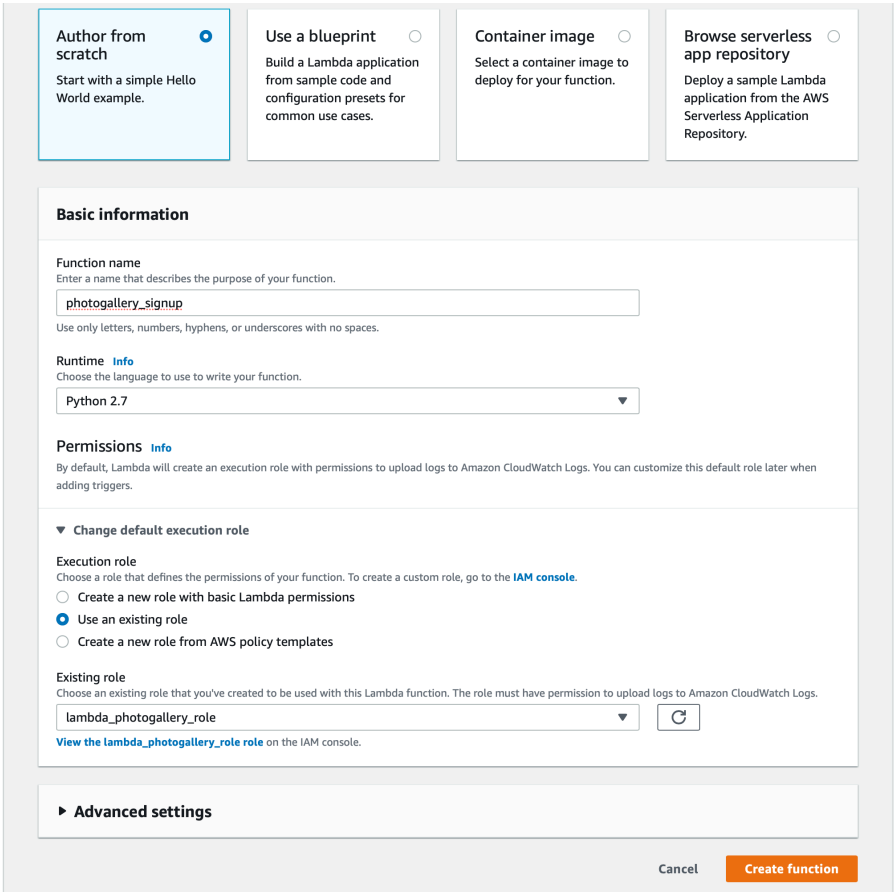
We are going to create seven serverless functions under the AWS Lambda console. Each function follow the same process to deploy them. Please, follow each of the screenshots below.

A. User signup (using source code in signup.py file provided):

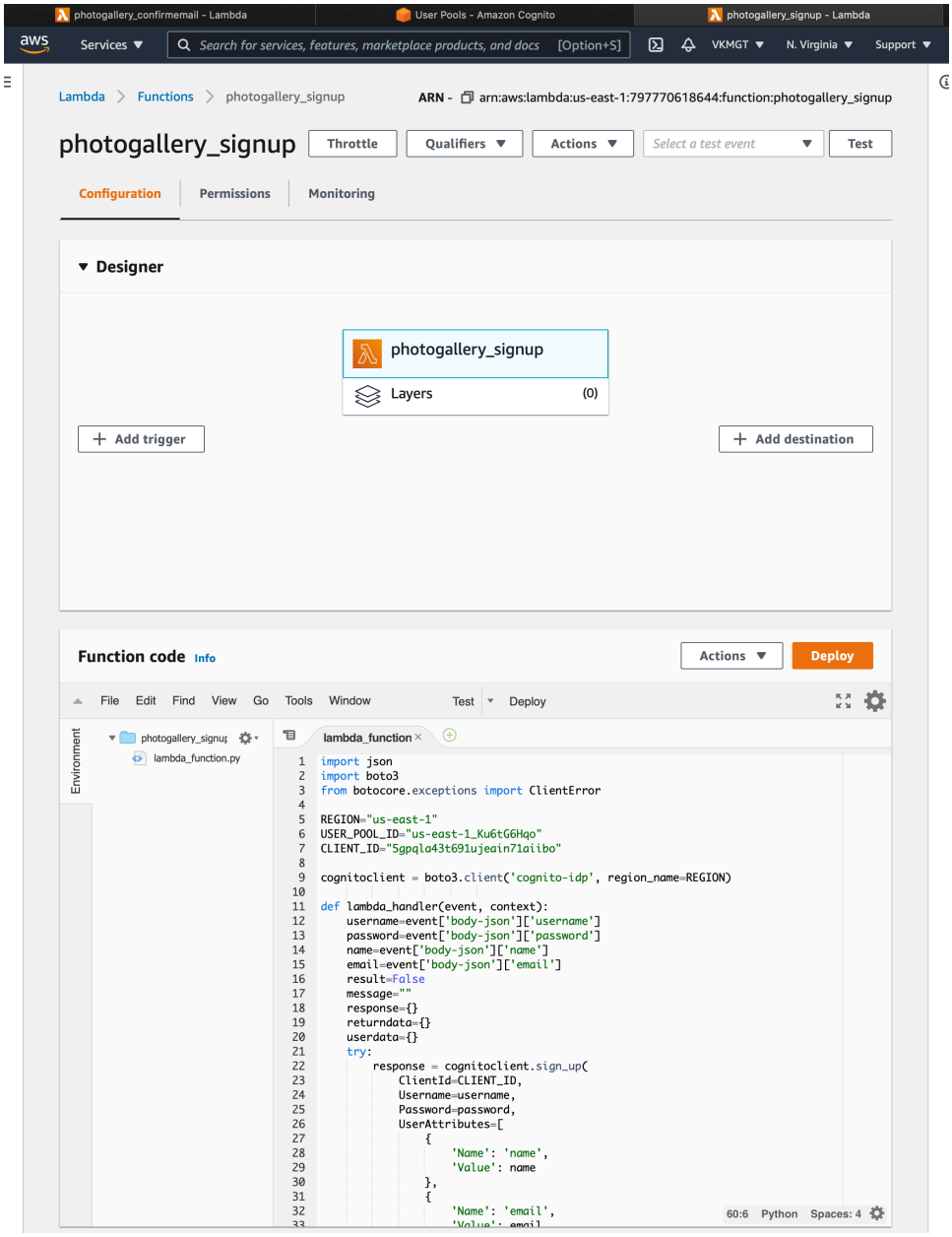
- Under AWS Lambda console, create a new function



- Give this function a name (**photogallery_signup**), attached the role previously created (**lambda_photogallery_role**) to this function, and then click "**Create function**"



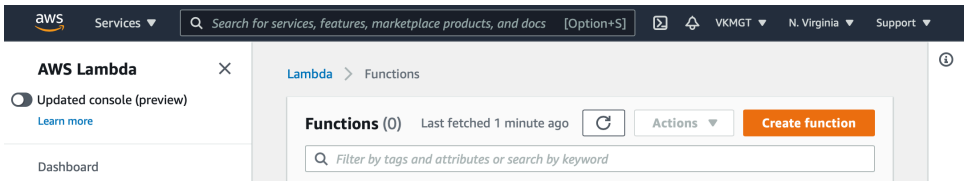
- After creating the function, under "**Function Code**" copy and paste the code provided.



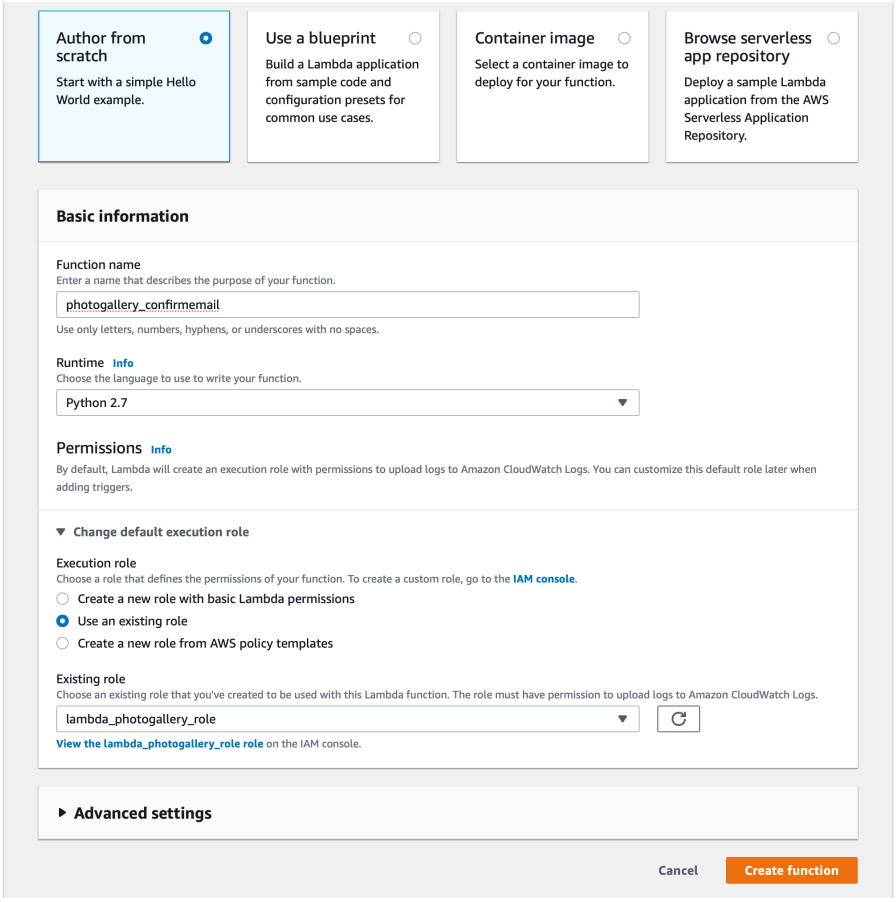
- Click "**Deploy**" to release the function

B. Confirming user email after signup (using source code in confirmemail.py file provided):

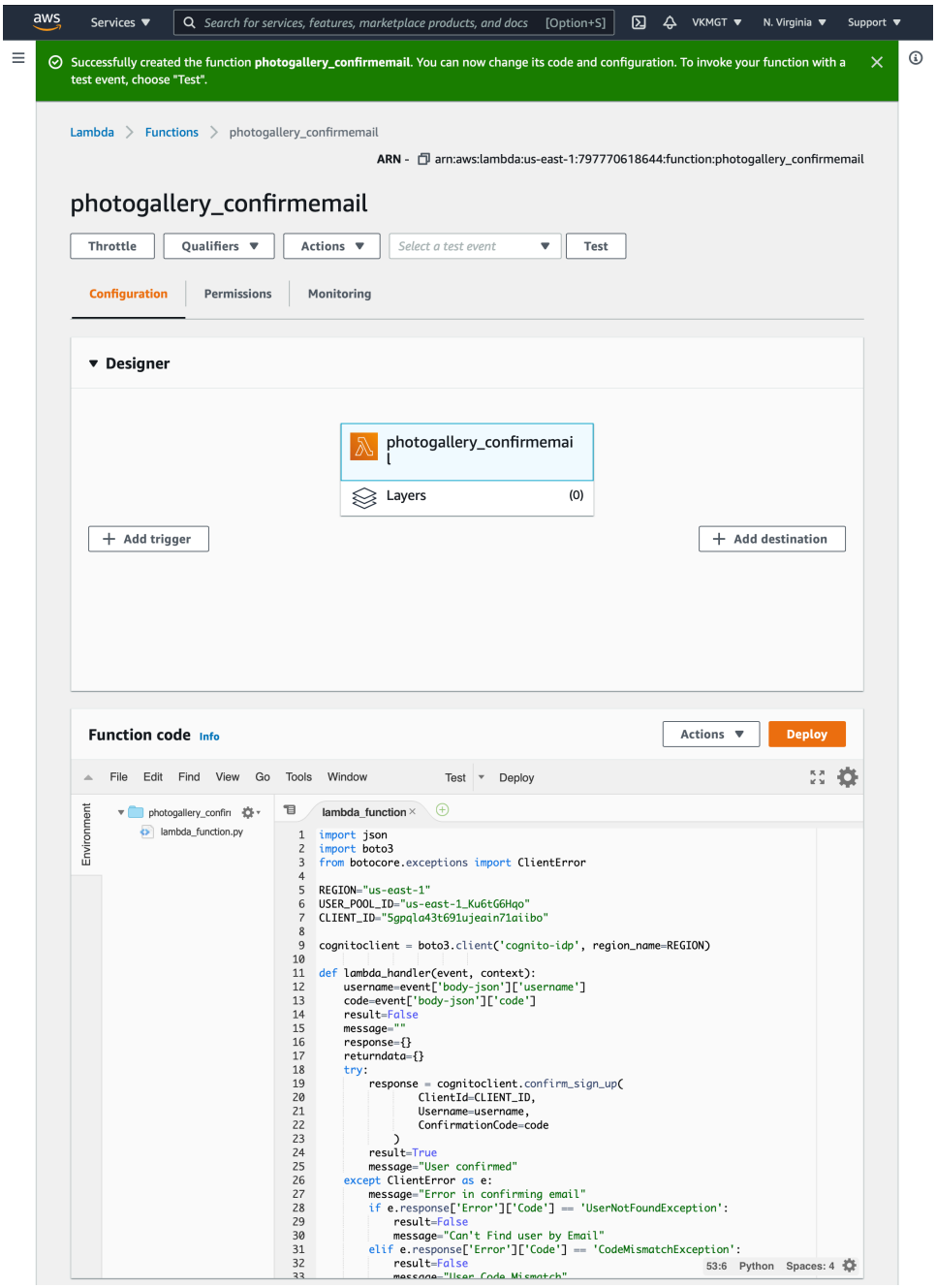
- Under AWS Lambda console, create a new function



- Give this function a name (**photogallery_confirmemail**), attached the role previously created (**lambda_photogallery_role**) to this function, and then click "**Create function**"



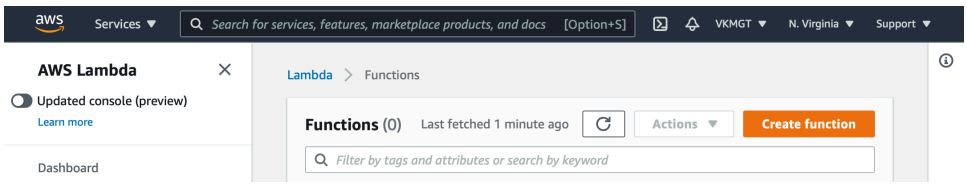
- After creating the function, under "**Function Code**" copy and paste the code provided.



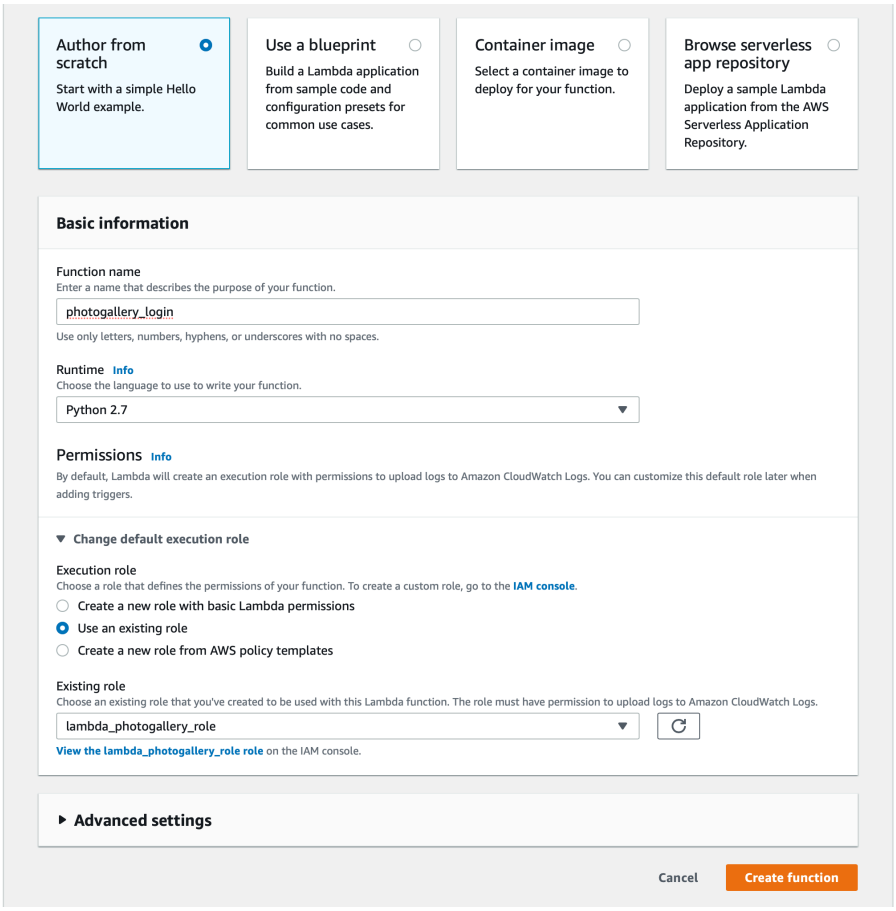
- Click "**Deploy**" to release the function

C. User login (using source code in login.py file provided):

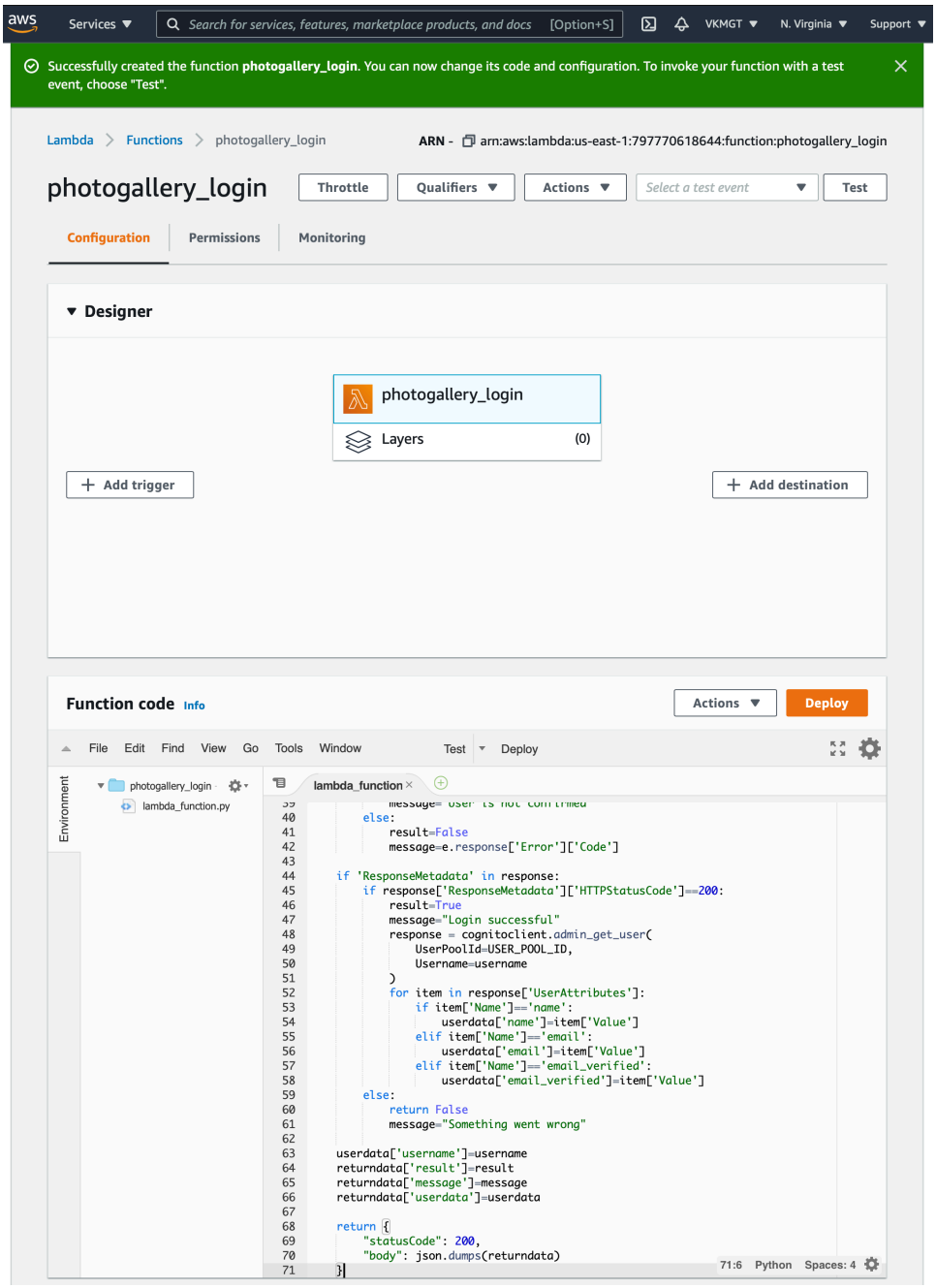
- Under AWS Lambda console, create a new function



- Give this function a name (**photogallery_login**), attached the role previously created (**lambda_photogallery_role**) to this function, and then click "**Create function**"



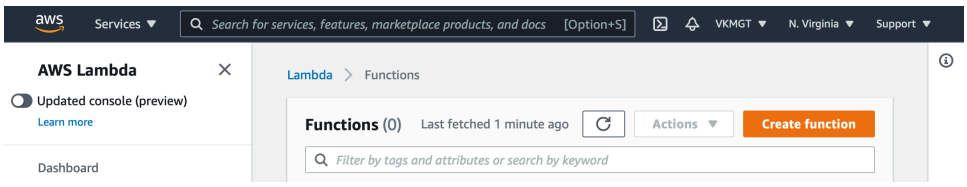
- After creating the function, under "**Function Code**" copy and paste the code provided.



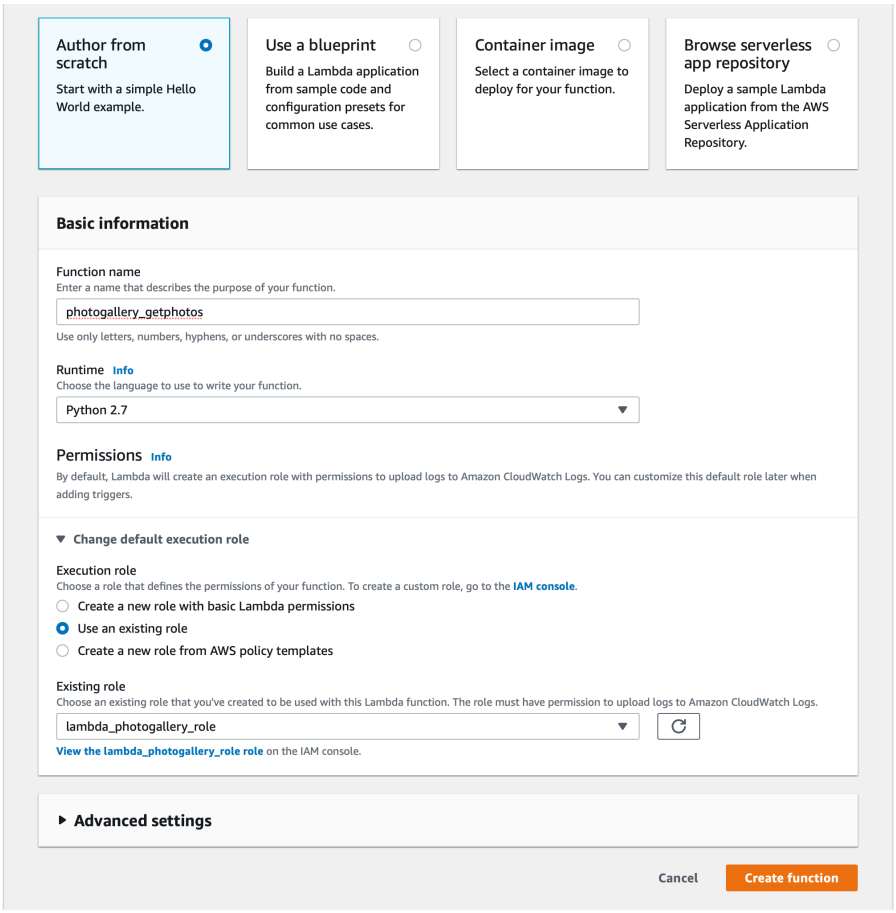
- Click "**Deploy**" to release the function

D. Getting details of all photos (using source code in getphotos.py file provided):

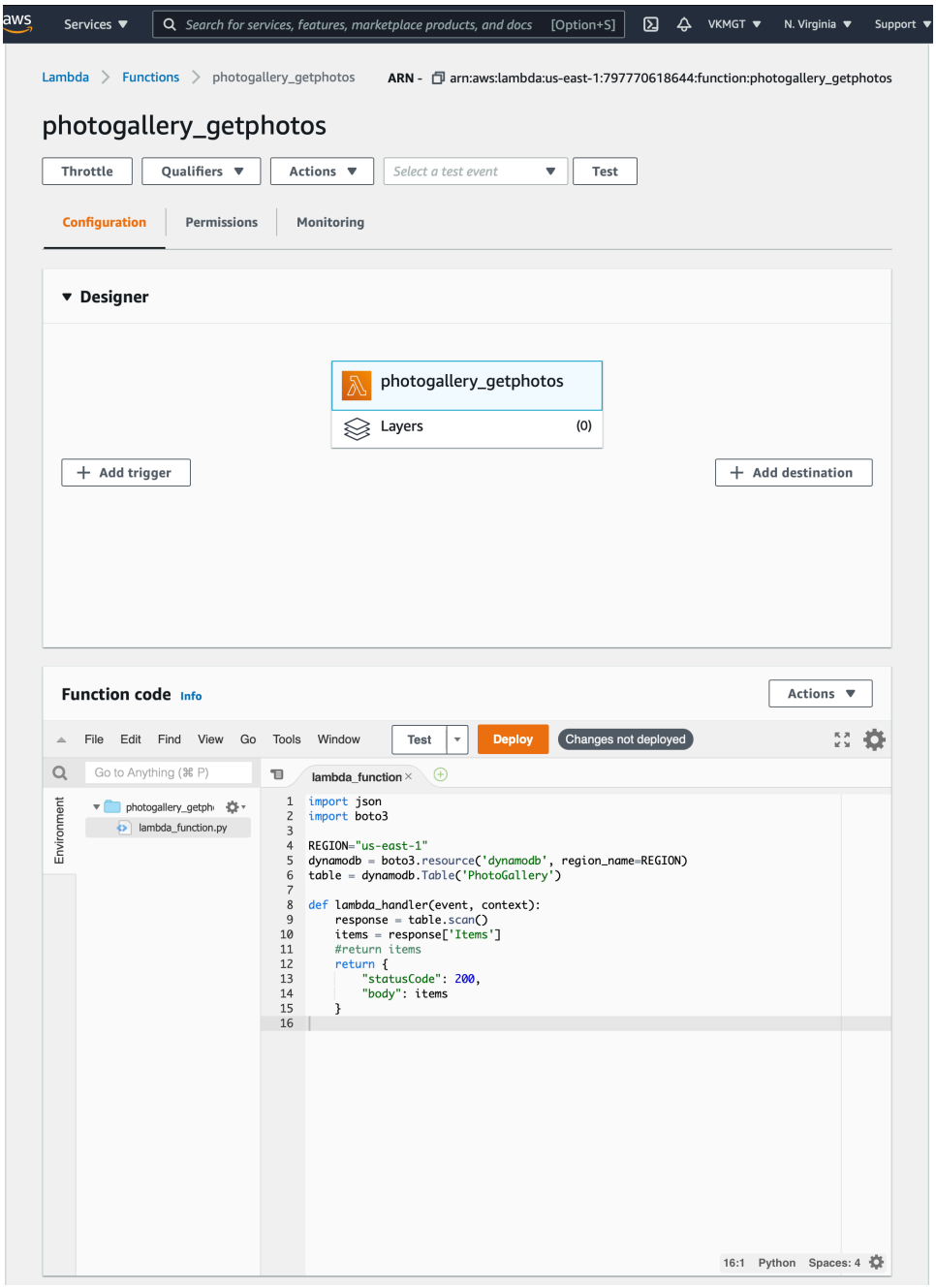
- Under AWS Lambda console, create a new function



- Give this function a name (**photogallery_getphotos**), attached the role previously created (**lambda_photogallery_role**) to this function, and then click "**Create function**"



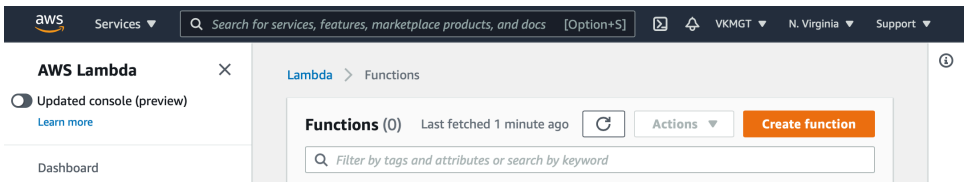
- After creating the function, under "**Function Code**" copy and paste the code provided.



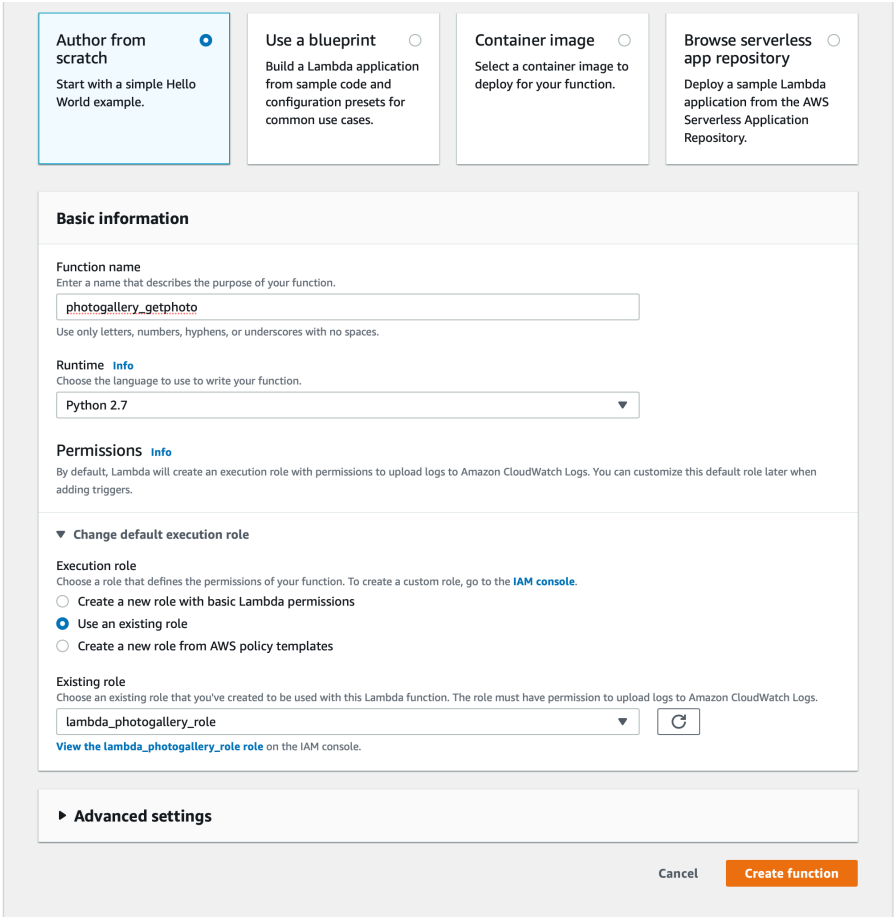
- Click "**Deploy**" to release the function

E. Getting details of a specific photo (using source code in getphoto.py file provided):

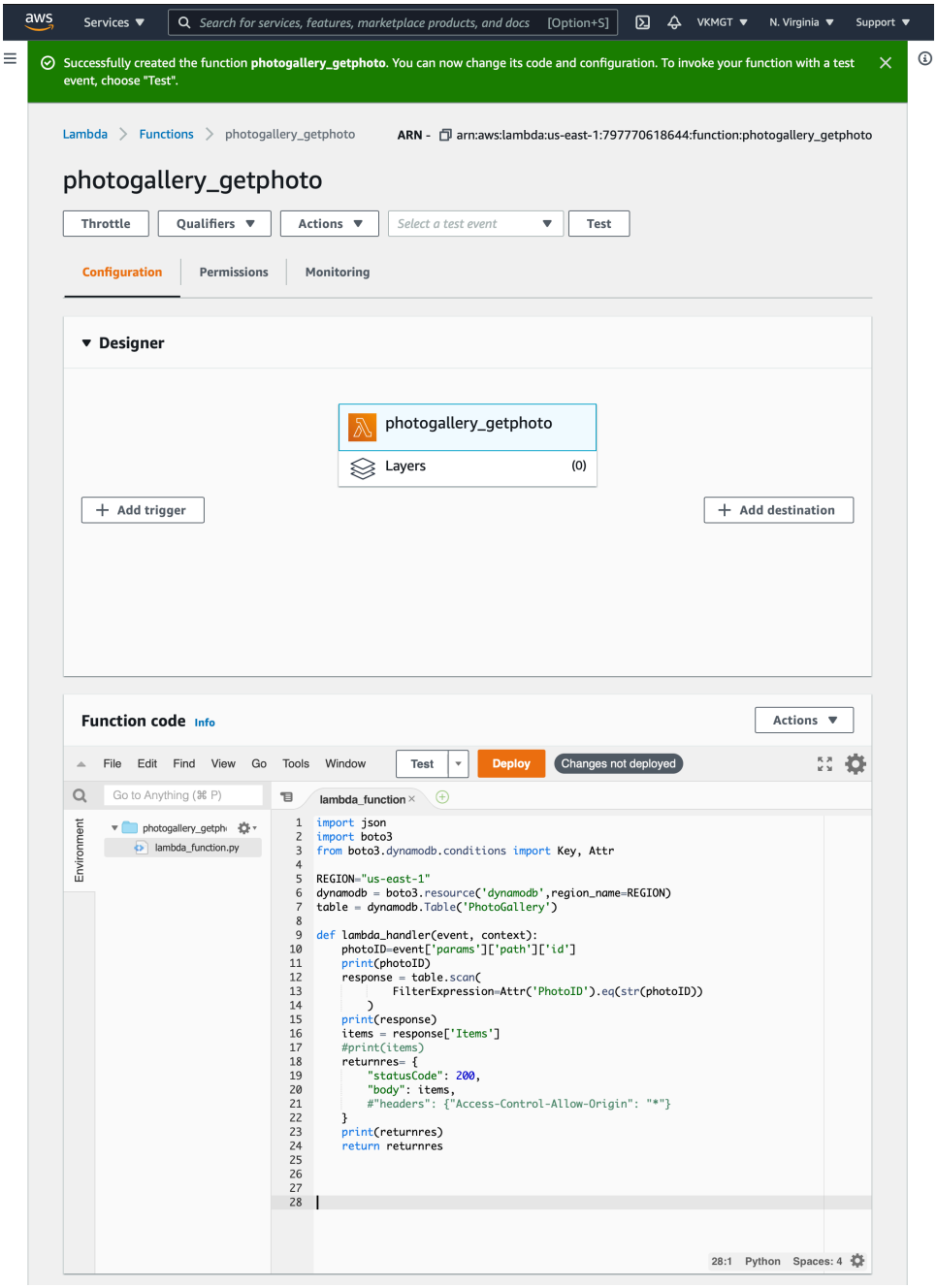
- Under AWS Lambda console, create a new function



- Give this function a name (**photogallery_getphoto**), attached the role previously created (**lambda_photogallery_role**) to this function, and then click "**Create function**"



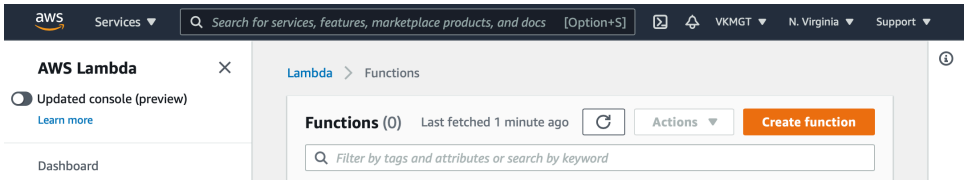
- After creating the function, under "**Function Code**" copy and paste the code provided.



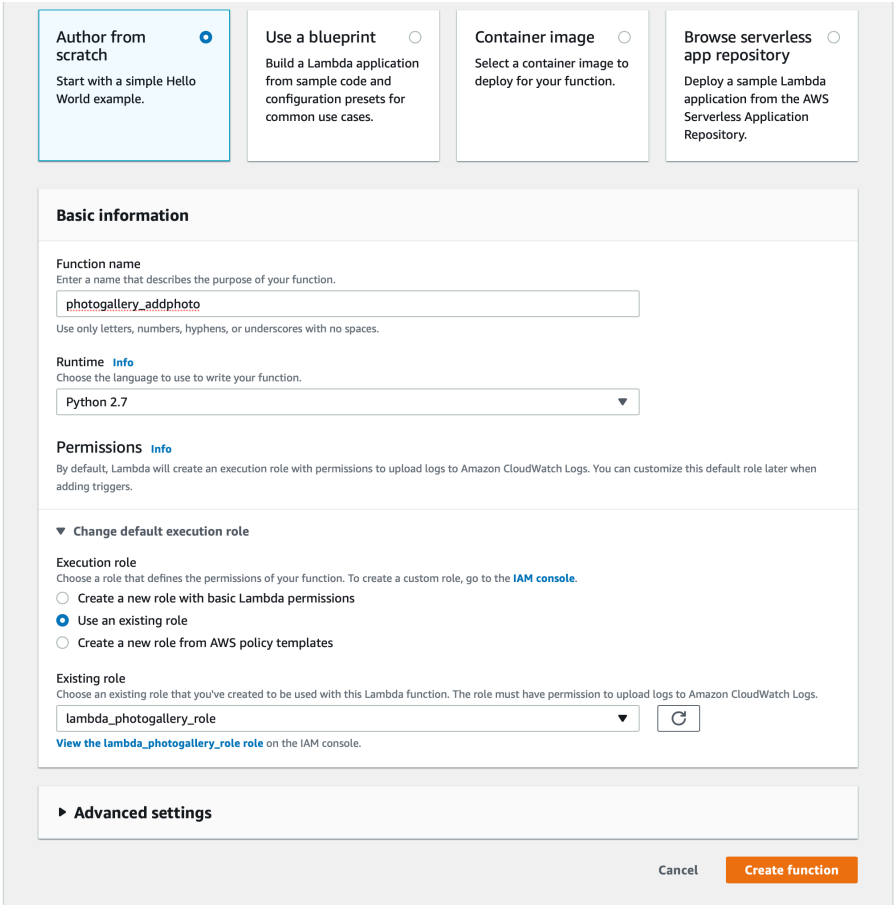
- Click "**Deploy**" to release the function

F. Adding a photo (using source code in addphoto.py file provided):

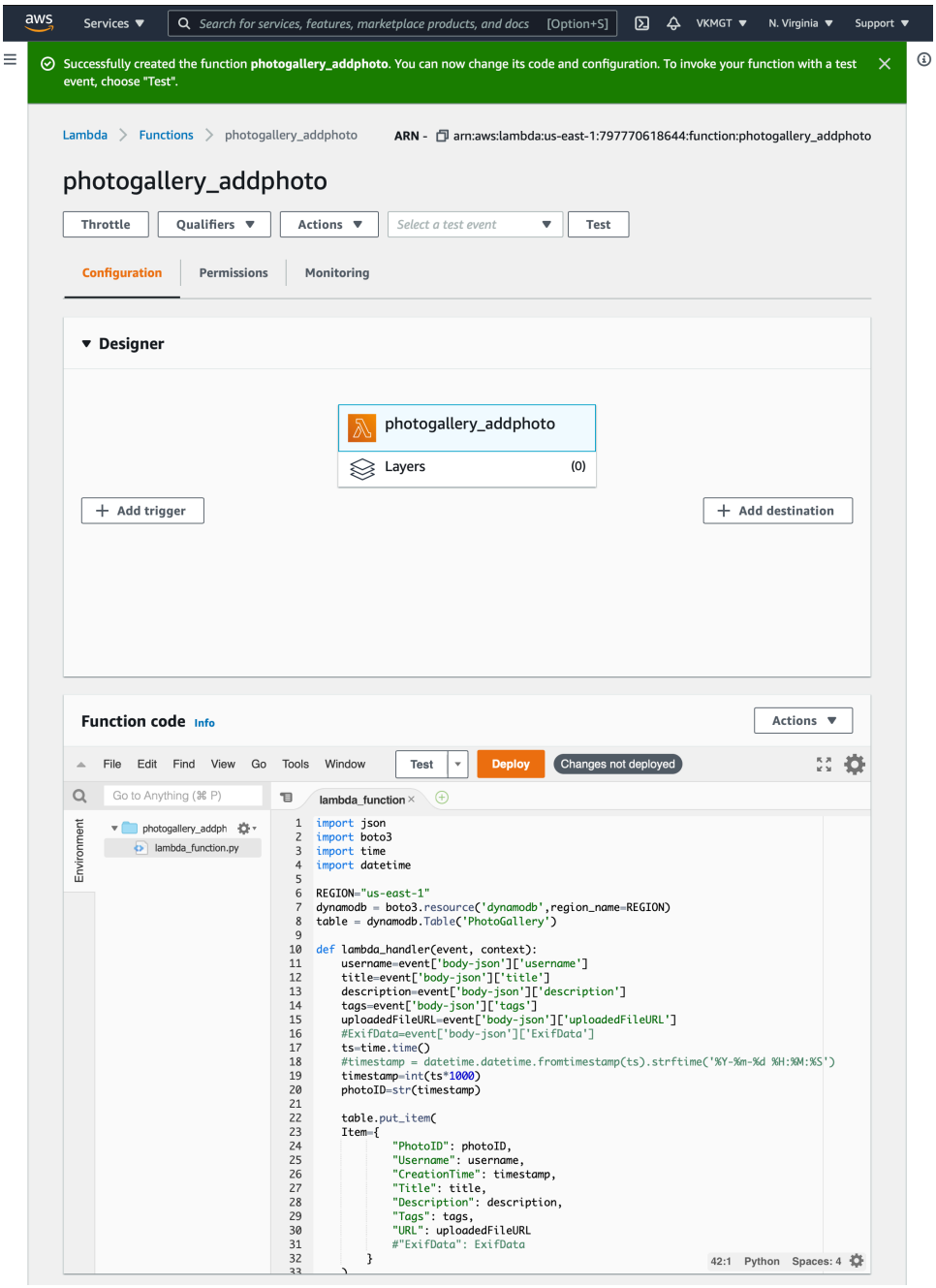
- Under AWS Lambda console, create a new function



- Give this function a name (**photogallery_addphoto**), attached the role previously created (**lambda_photogallery_role**) to this function, and then click "**Create function**"



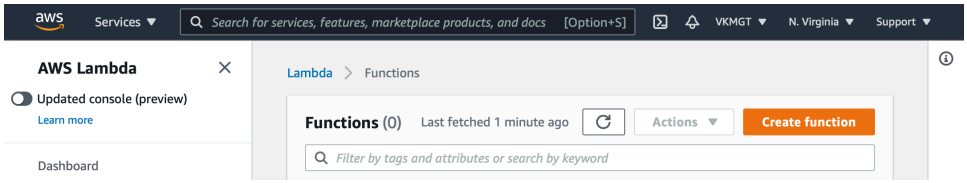
- After creating the function, under "**Function Code**" copy and paste the code provided.



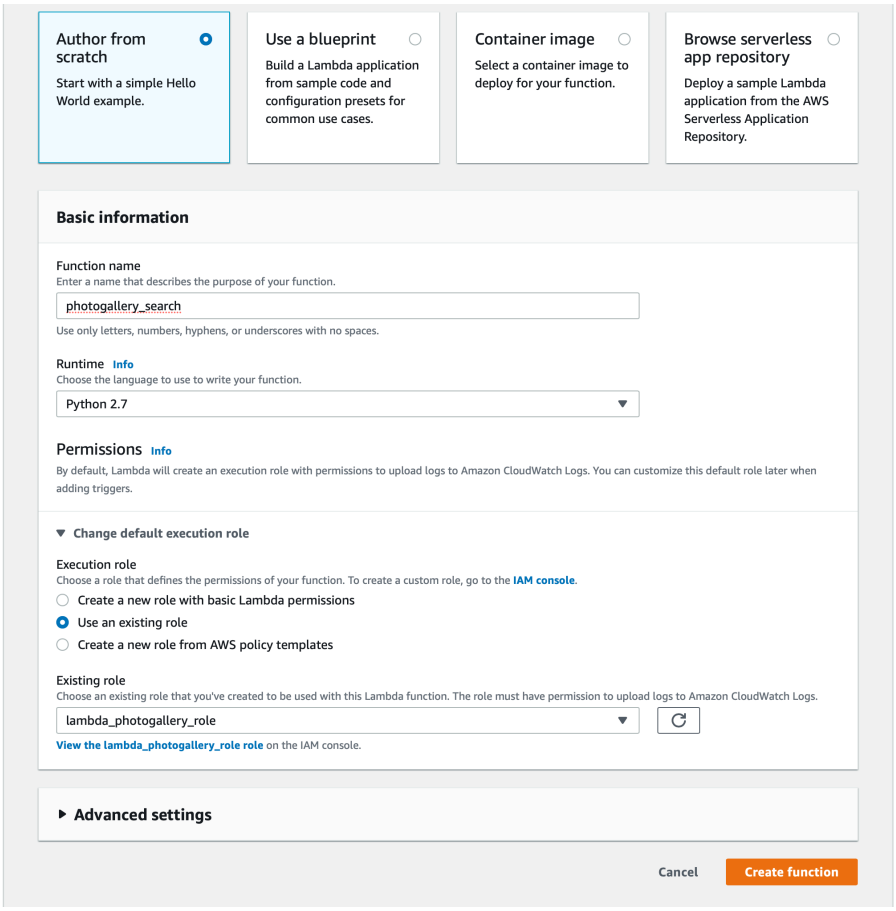
- Click "**Deploy**" to release the function

G. Searching photos (using source code in search.py file provided):

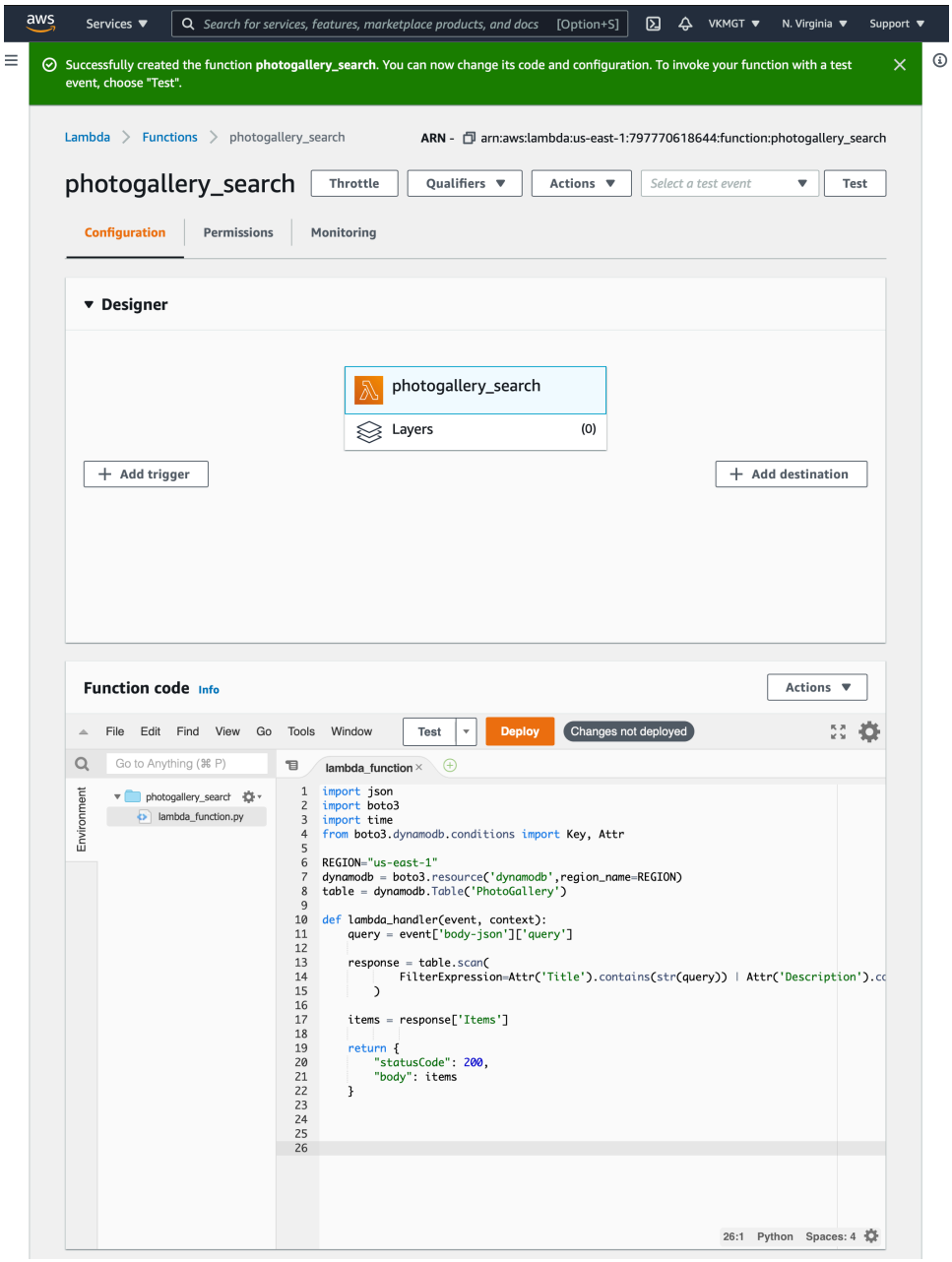
- Under AWS Lambda console, create a new function



- Give this function a name (**photogallery_search**), attached the role previously created (**lambda_photogallery_role**) to this function, and then click "**Create function**"



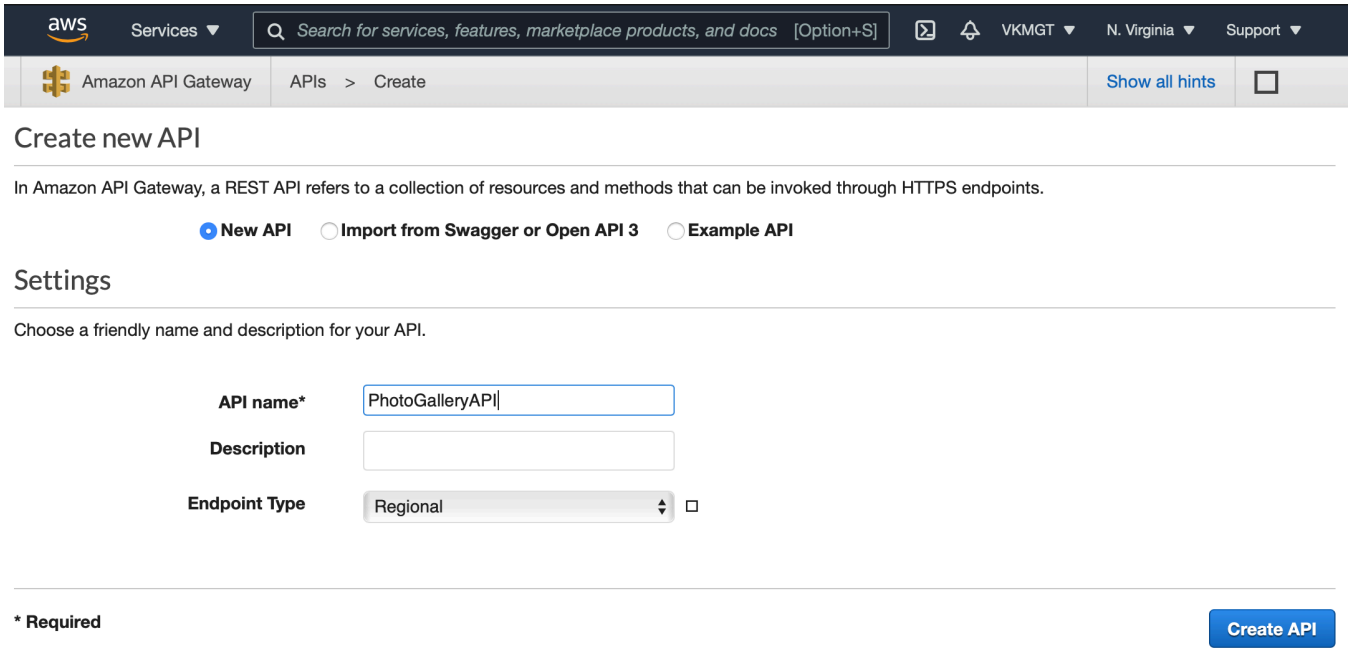
- After creating the function, under "**Function Code**" copy and paste the code provided.



- Click "**Deploy**" to release the function

8. Create a new API from API Gateway

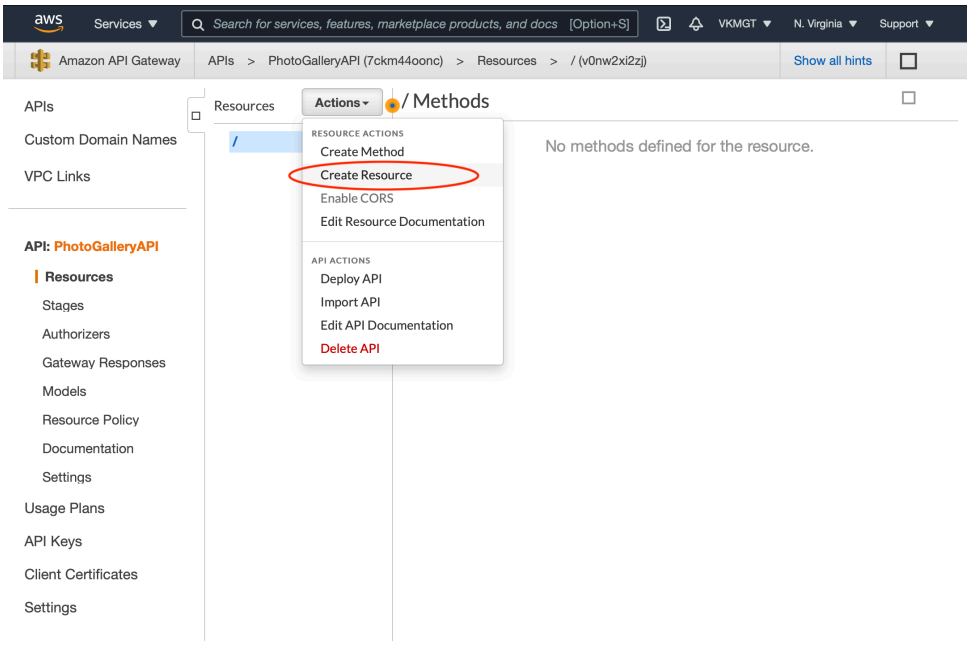
Go to the AWS API Gateway Console. Once you enter the console, it will provide you with an option to create an Example API with some resources and method; disregard that option and select "**New API**". Create a new API called **PhotoGalleryAPI** with the resources, methods, and integration shown below.



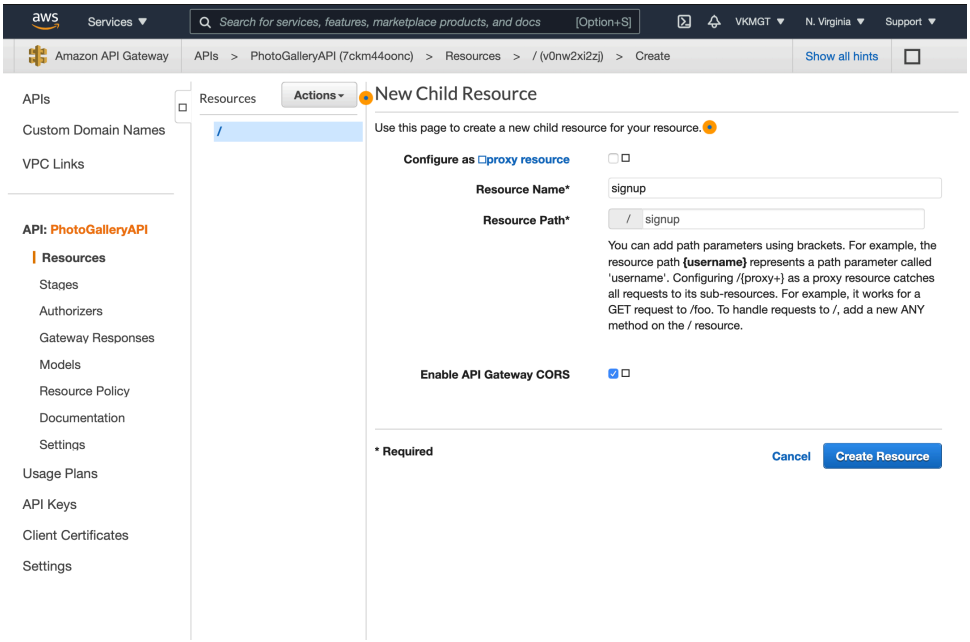
Resource	Method	Integration Request
/signup	POST	Lambda Function: photogallery_signup
/confirmemail	POST	Lambda Function: photogallery_confirmemail
/login	POST	Lambda Function: photogallery_login
/photos	POST	Lambda Function: photogallery_addphoto
/photos	GET	Lambda Function: photogallery_getphotos
/photos/{id}	GET	Lambda Function: photogallery_getphoto
/uploadphoto	PUT	S3 Bucket: photobucket-corporan-2021-4813
/search	POST	Lambda Function: photogallery_search

For **/signup**, a POST request a lambda function:

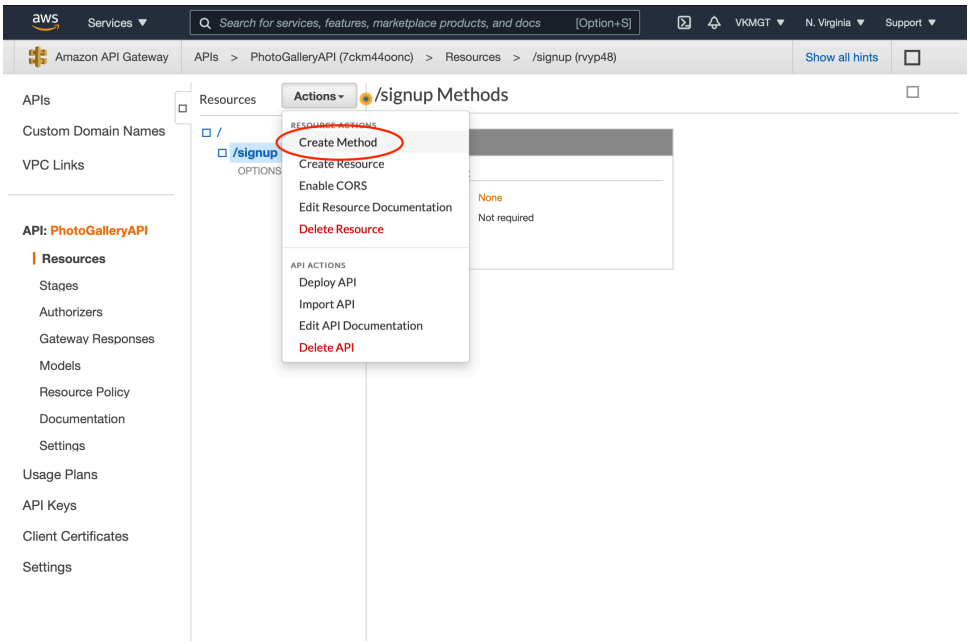
- Create a new **Resource** under the **Actions** dropdown button



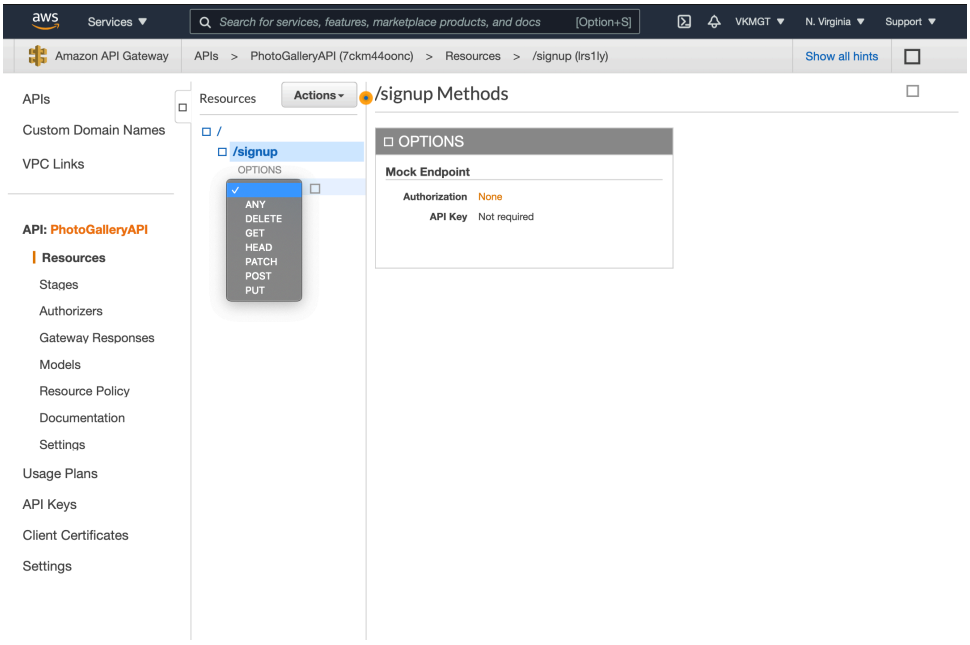
- Give a name to the resource and make sure that "**Enable API Gateway CORS**" is checked



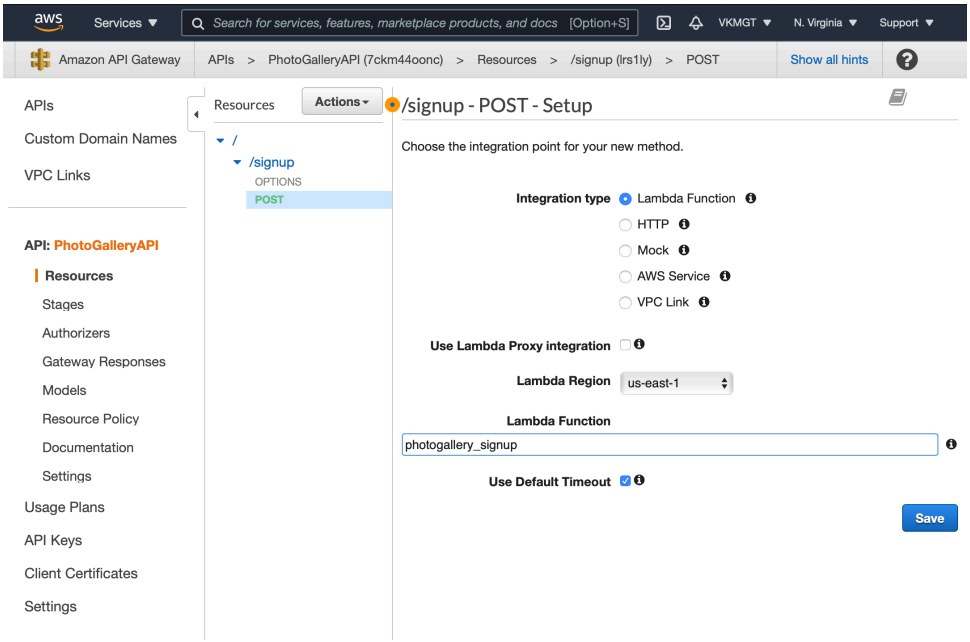
- Once the resource is created, select the resource and create a new **Method** under that resource.



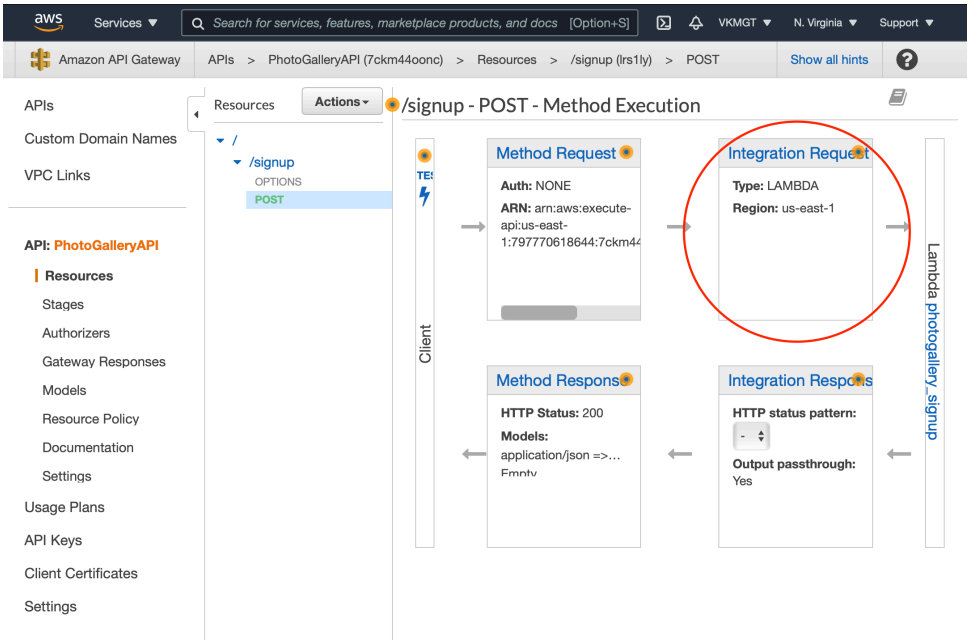
- Select **POST** as the method for this resource

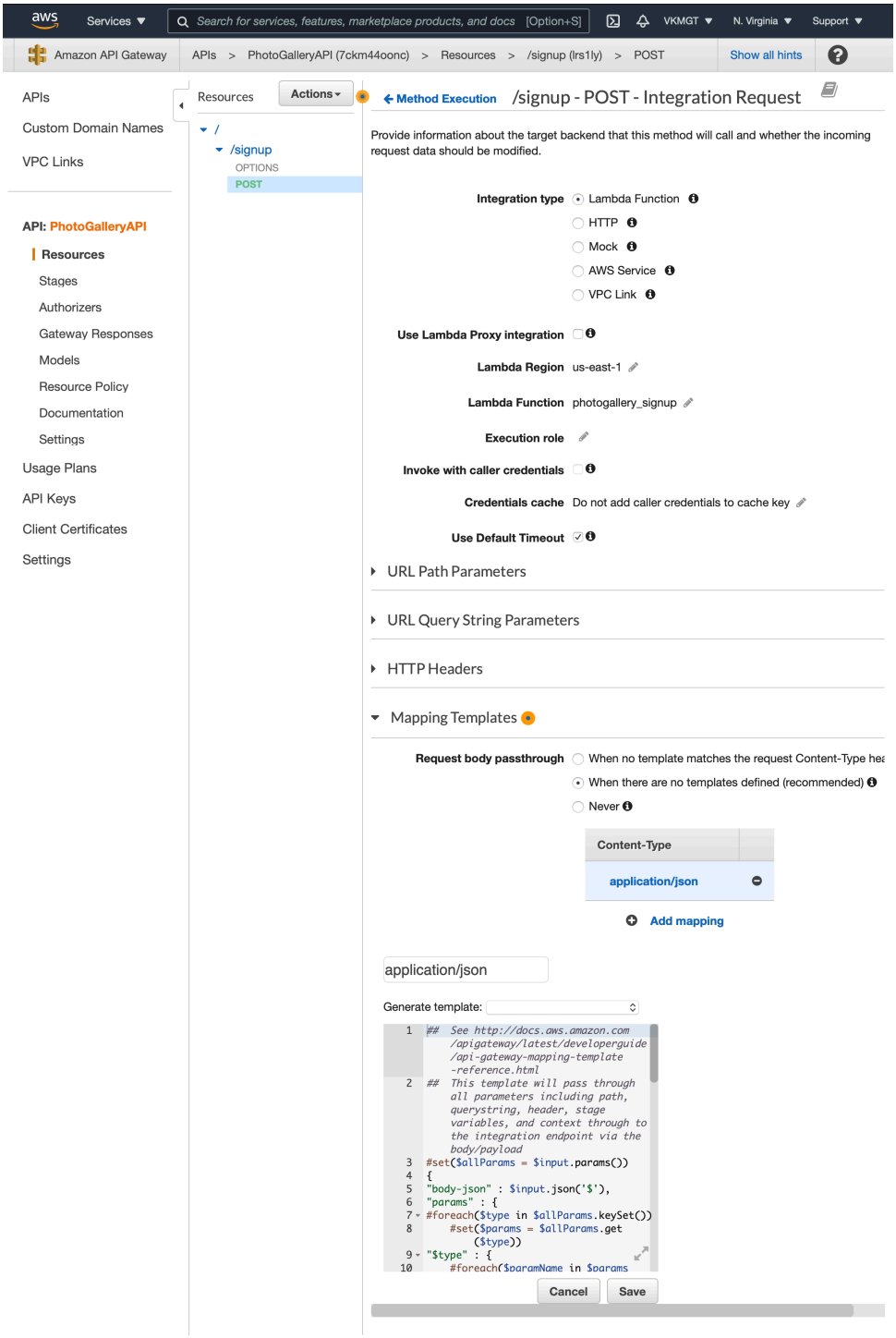


- Select the method created and attach the lambda function as the integration type.

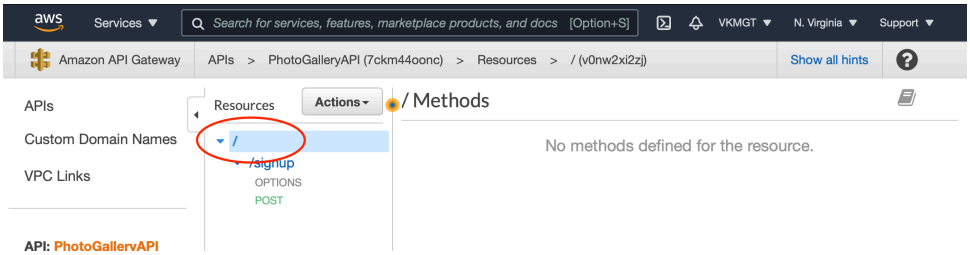


- After saving the integration type, select "**Integration Request**" under the method. Under the "**Mapping Template**", choose "**When there are no templates defined (recommended)**" under the Request body passthrough. Under the **Content-Type**, click "**Add mapping**" and add "**application/json**" to the mapping. Finally, Under the **Generate template** dropdown, select "**Method Request passthrough**" and click "**Save**".





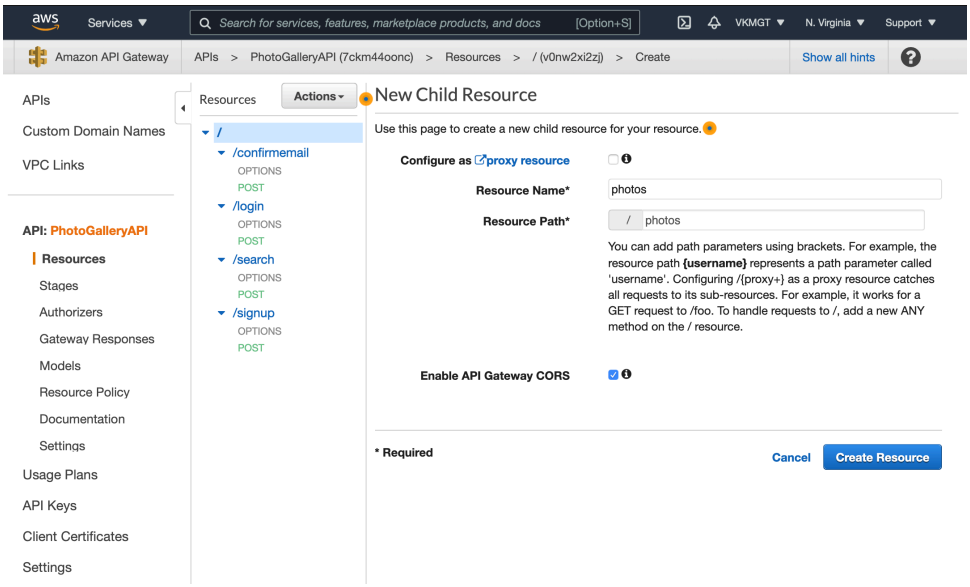
- To create another resource, click "/" under the Resources column



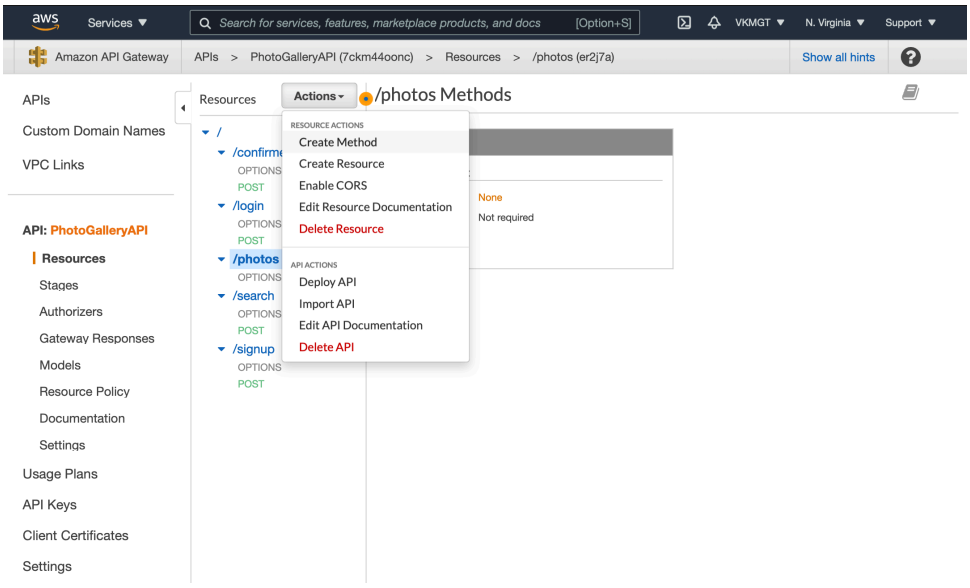
Repeat the same process for **/confirmemail**, **/login**, and **/search**

For **/photos**, a GET request a lambda function:

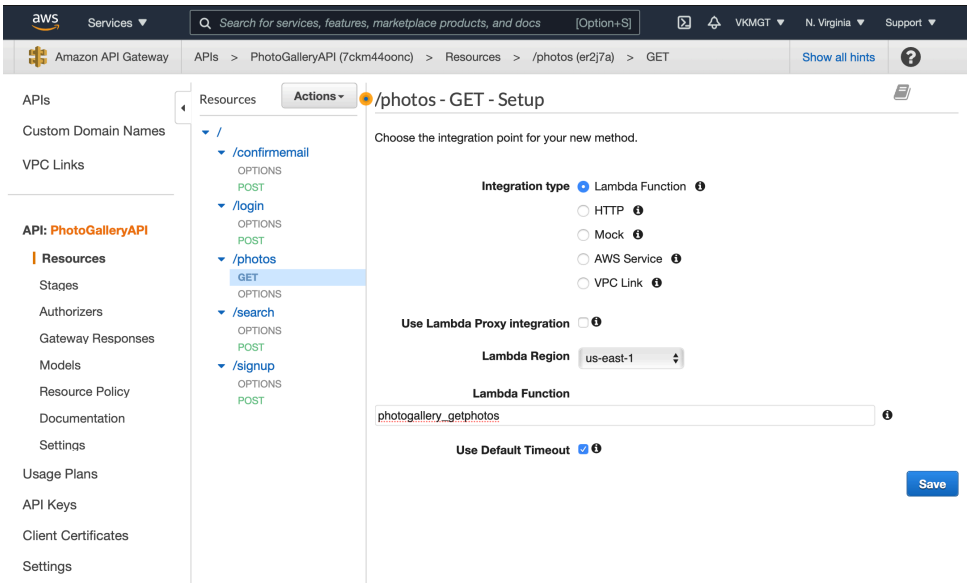
- Create a new **Resource** under the **Actions** dropdown button. Give a name to the resource and make sure that "**Enable API Gateway CORS**" is checked. Set the mapping template for all the method created below.



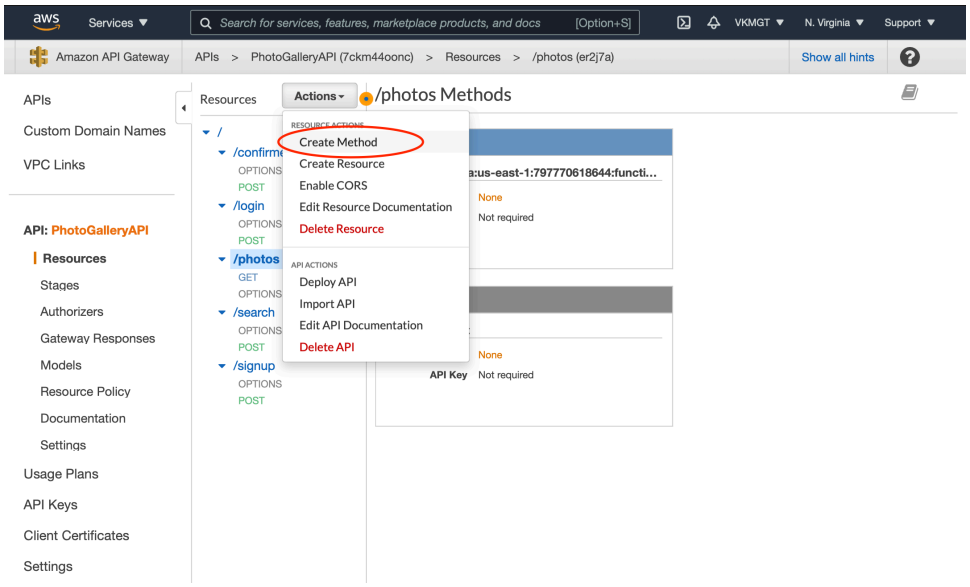
- Once the resource is created, select the resource and create a new **Method** under that resource.

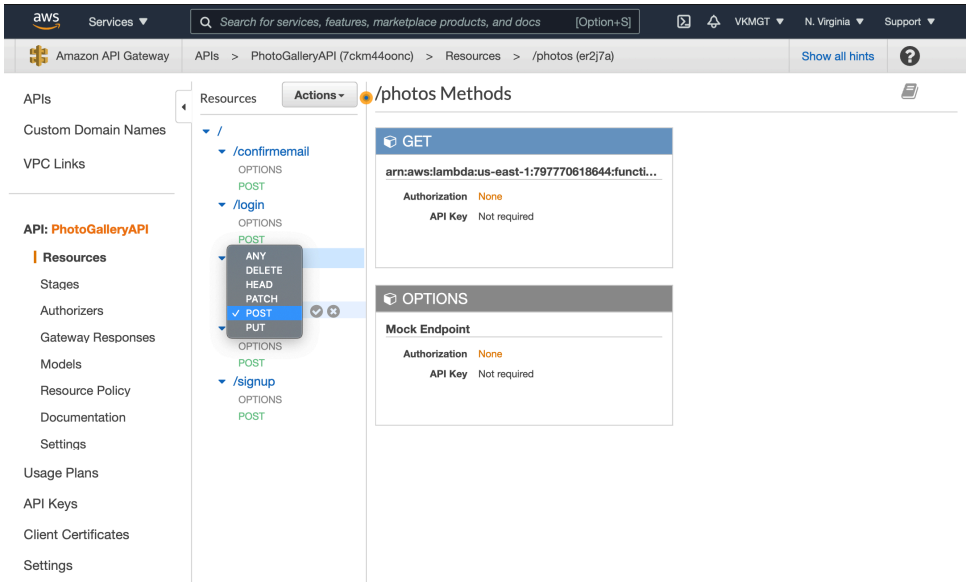


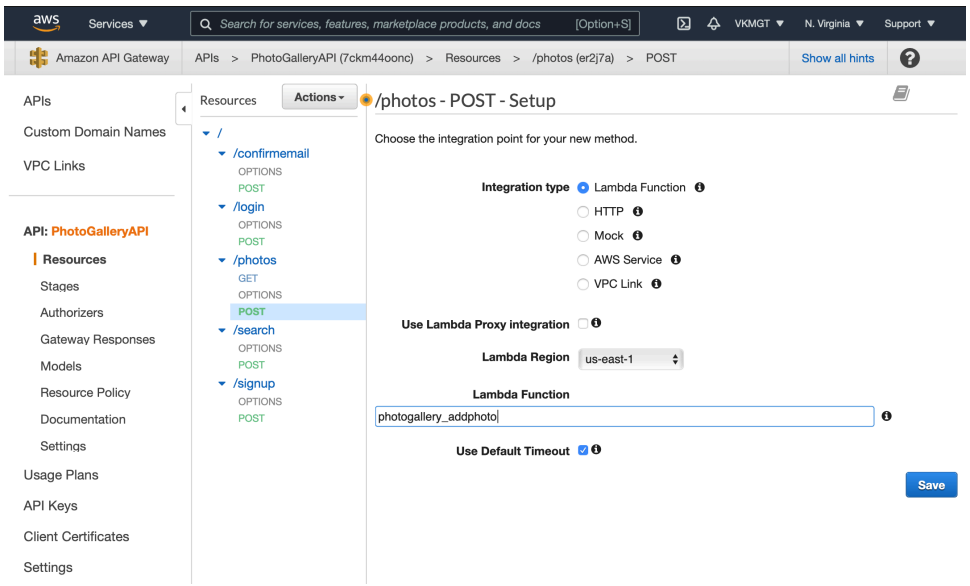
- Select **GET** as the method for this resource. Select the method created and attach the lambda function as the integration type.



- Under the /photos resource, create a new **POST** method. Select the method created and attach the lambda function as the integration type.

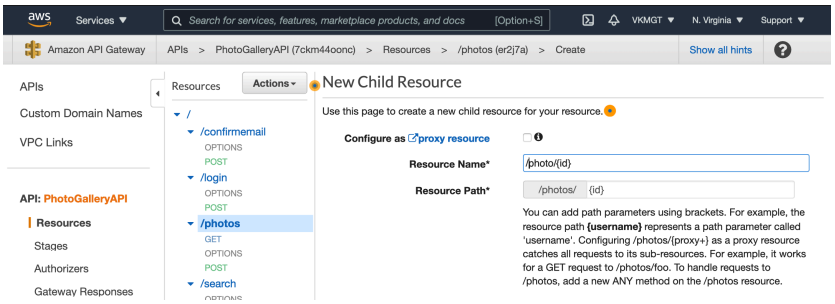


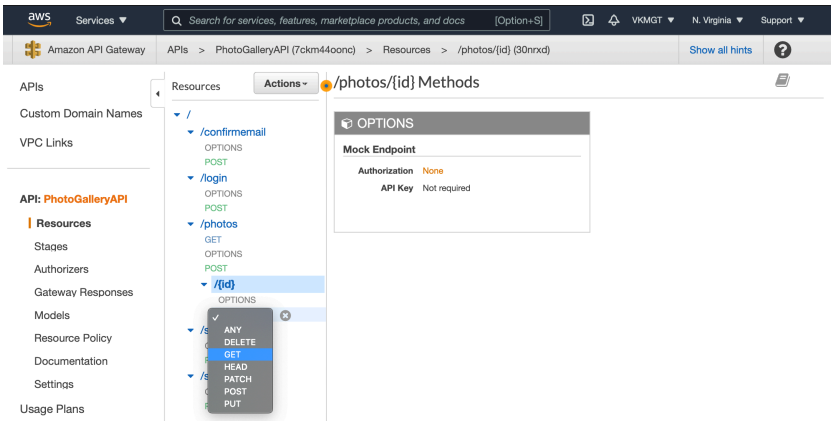


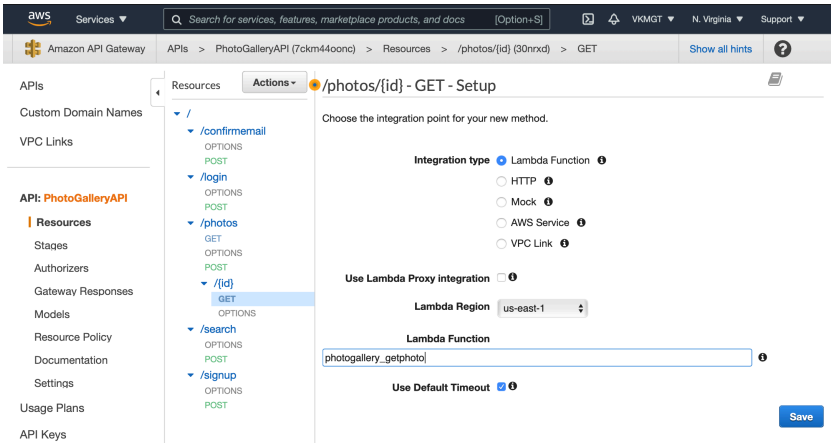


- After saving the integration type, select "**Integration Request**" under both get and post methods. Under the "**Mapping Template**", choose "**When there are no templates defined (recommended)**" under the Request body passthrough. Under the **Content-Type**, click "**Add mapping**" and add "**application/json**" to the mapping. Finally, Under the **Generate template** dropdown, select "**Method Request passthrough**" and click "**Save**".

- Under the /photos resource, create a new resource for /photos/{id}, as shown below. Under the new created resource, create a new **GET** method. Select the method created and attach the lambda function as the integration type.

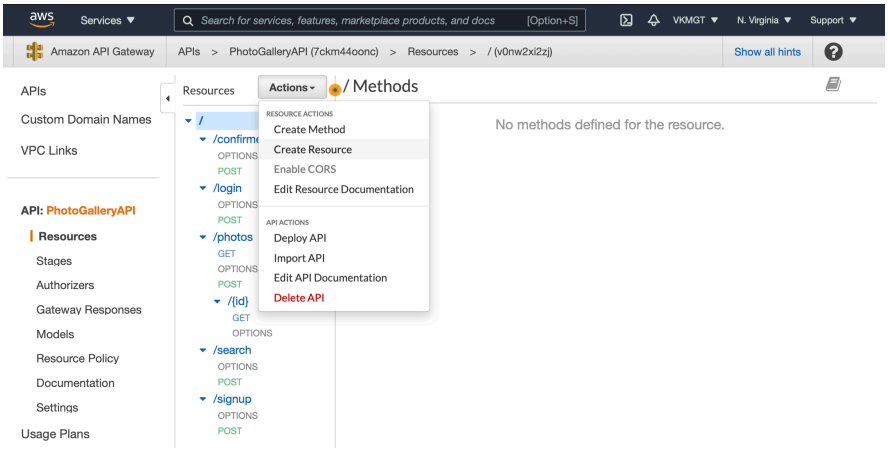


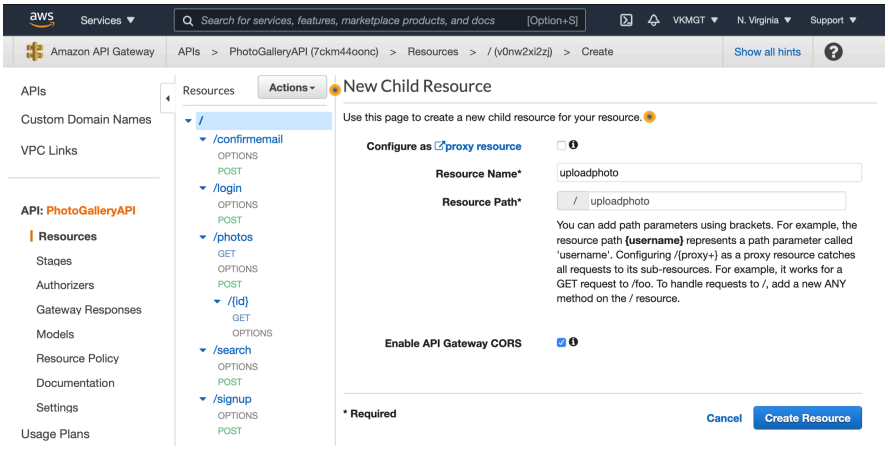




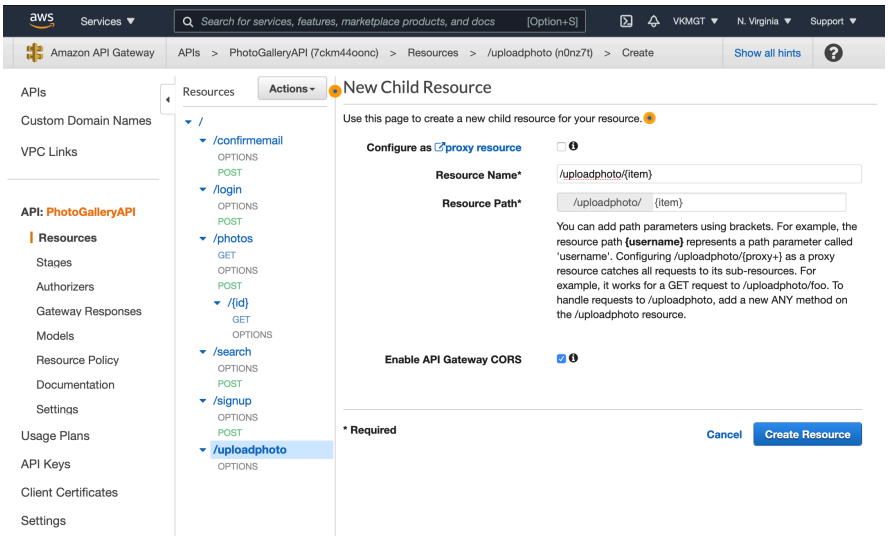
- After saving the integration type, select "**Integration Request**" under the method. Under the "**Mapping Template**", choose "**When there are no templates defined (recommended)**" under the Request body passthrough. Under the **Content-Type**, click "**Add mapping**" and add "**application/json**" to the mapping. Finally, Under the **Generate template** dropdown, select "**Method Request passthrough**" and click "**Save**".

- Finally, Create a new **Resource** at the root ("/") to upload the photos to the S3 bucket. Give a name to the resource and make sure that "**Enable API Gateway CORS**" is checked

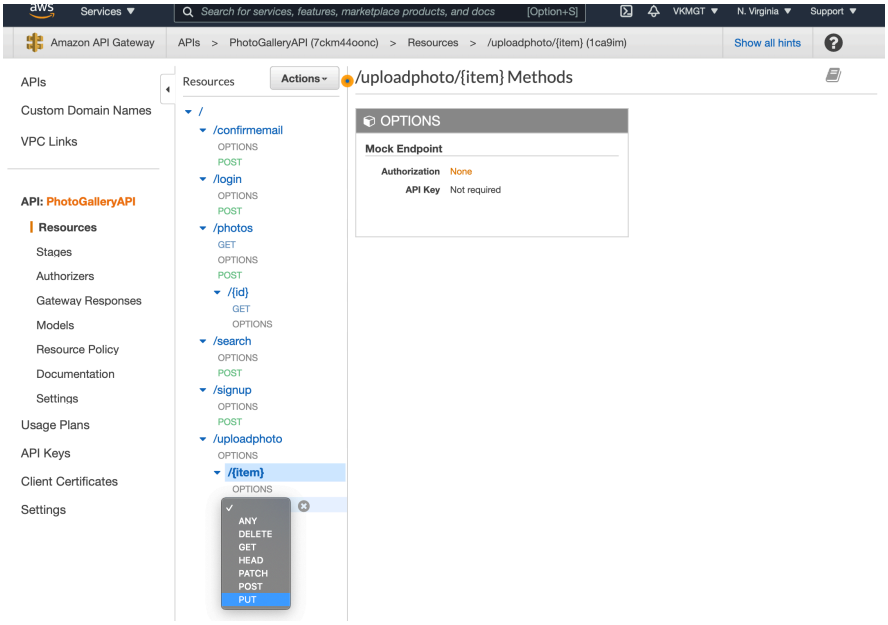




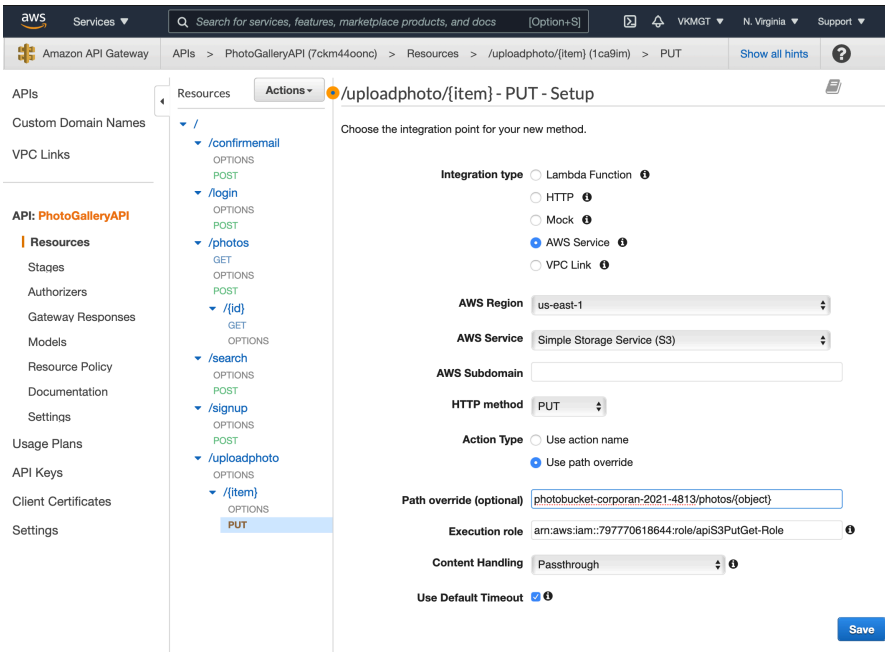
- Within the same resource, create another resource as shown below.



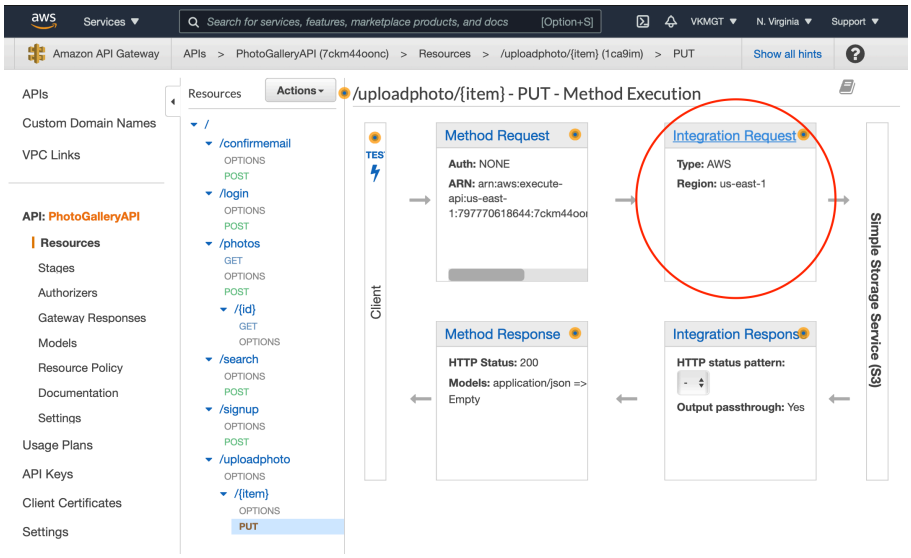
- Under the new created resource, create a new **PUT** method.

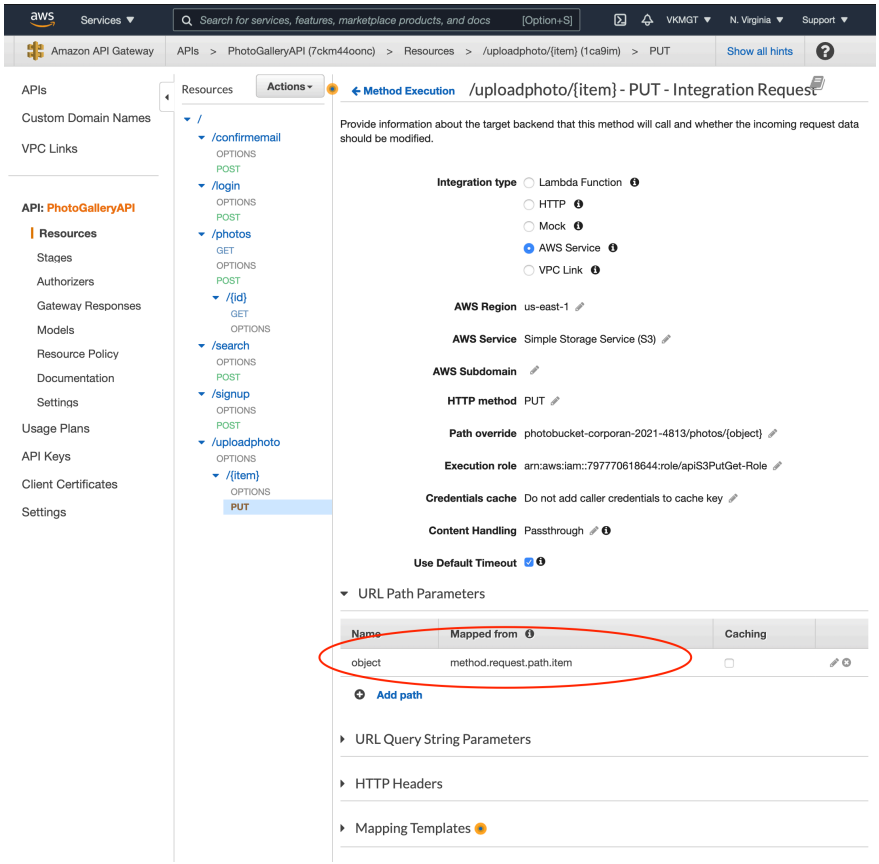


- Select the method created and select the **AWS Region** where the S3 bucket was created (in this case, is **us-east-1** or **N. Virginia**). Under **AWS Service**, choose **Simple Storage Service (S3)**. Under **Action Type**, select "**Use path override**" and add the path override; in this case, it is the path where the photos are going to be stored in the bucket (**photobucket-corporan-2021-4813/photos/{object}**). Attach the execution role previously for the S3 bucket. Click "**Save**" to set the configuration.

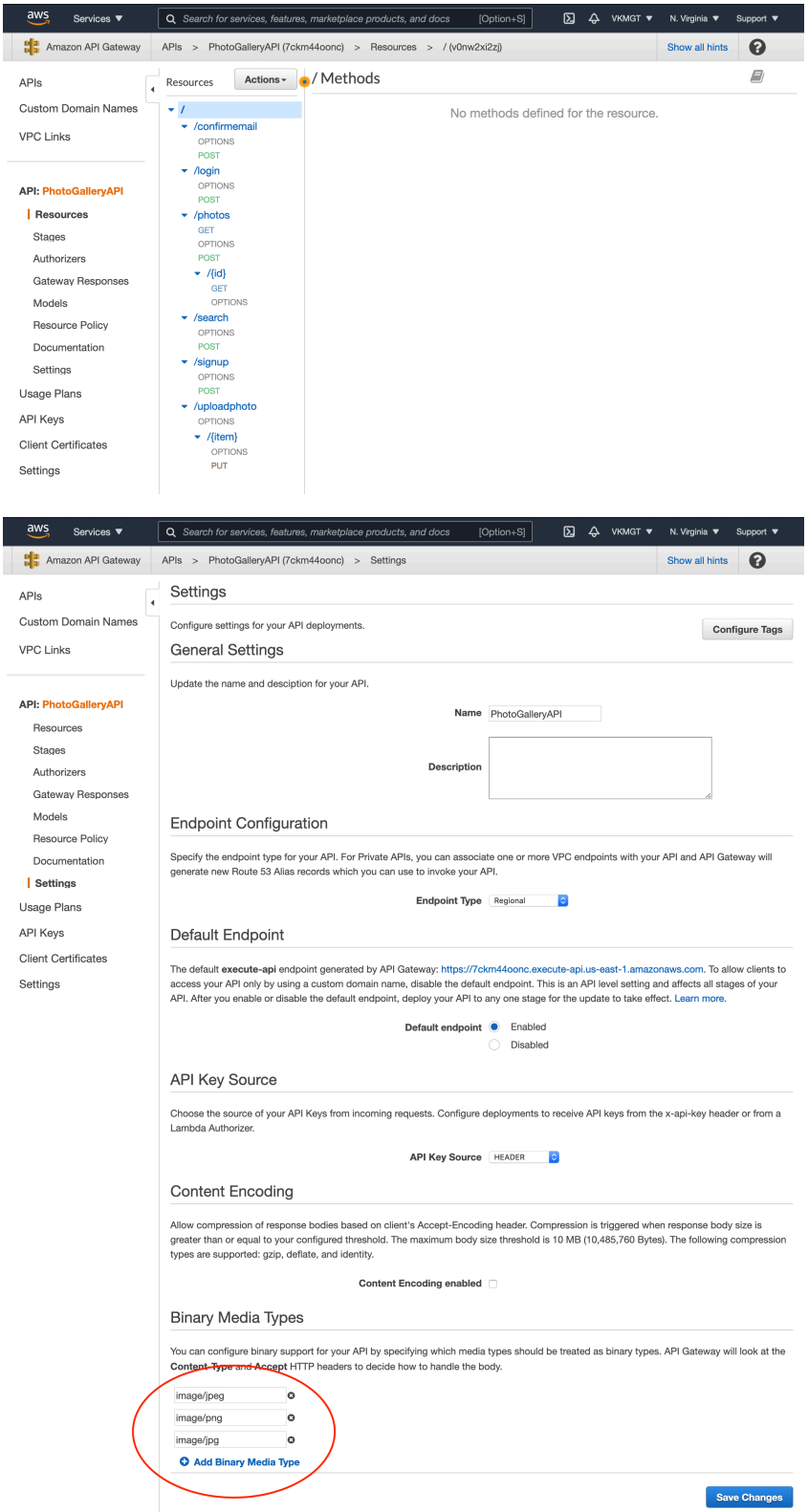


- Once the method is created, select the **Integration Request** under the method and create a mapping configuration as shown below.

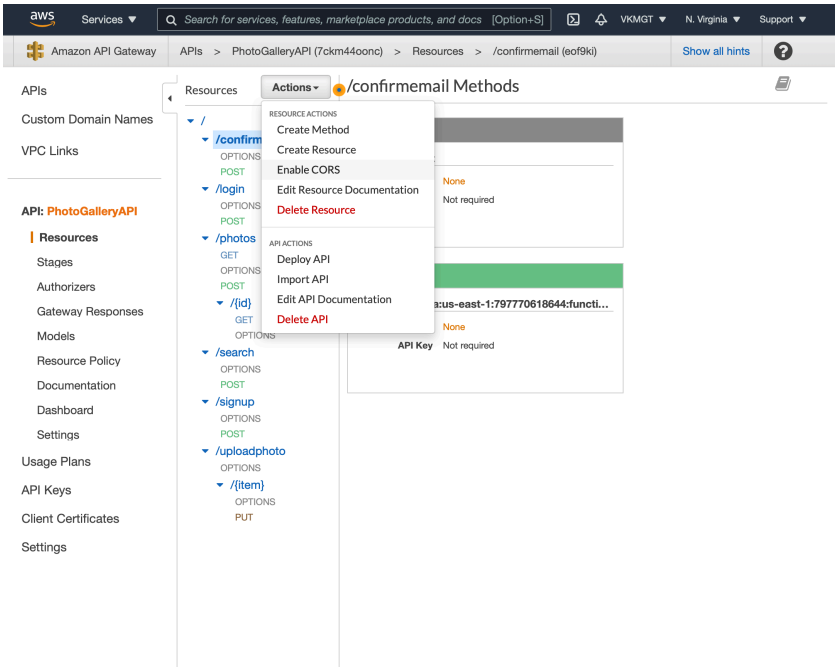




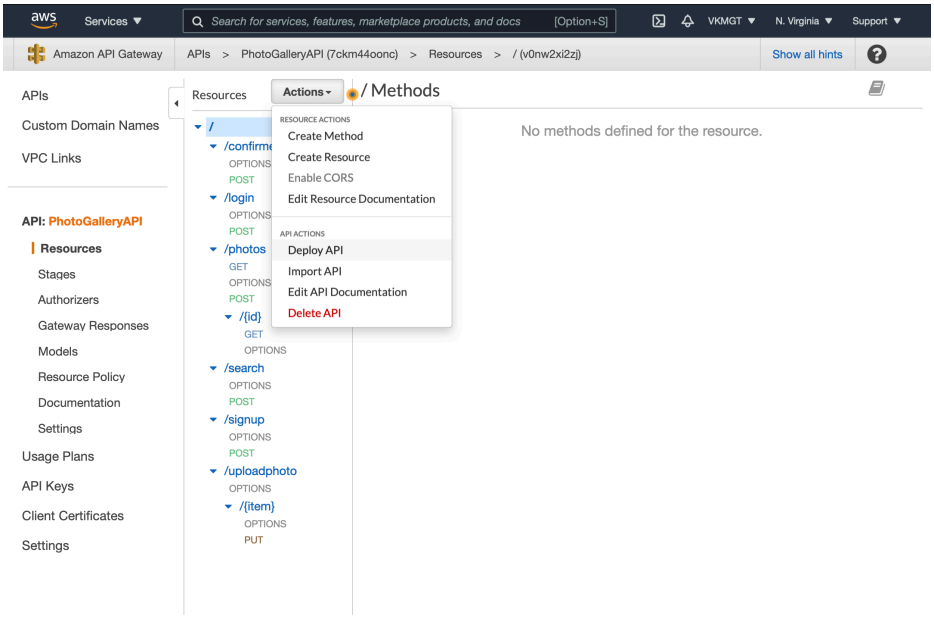
- To allow images to be uploaded to the S3 bucket, we need to add the binary media types. For this case, we are going to allow **JPEG**, **JPG**, and **PNG** to be uploaded to the bucket. To do that, select **Setting** in the left column and add the media types under **Binary Media Types**, as shown below



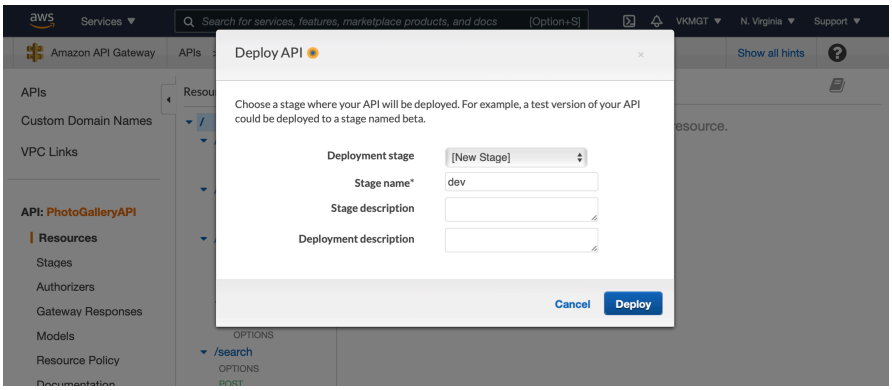
- Make sure to **enable CORS** for the resources. To do that, select the resources and click the **Actions** dropdown and select "**Enable CORS**".



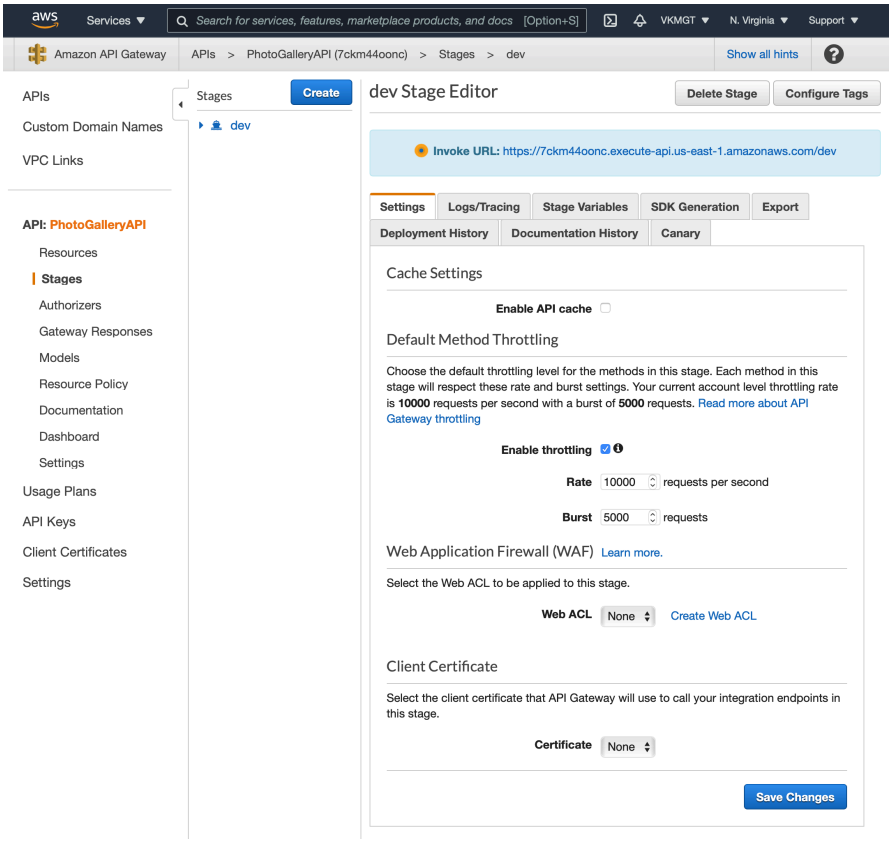
- To deploy the API, click "**Deploy API**", as shown below



- Create a new stage for deployment and called it "**dev**" as shown below



- Copy and save the API URL; will use this URL to access the resources from the website.



9. Updating the website

Add some resources to script used for the website. Update the **process.js** file under the staticwebsite/assets directory. Add the **<API_URL>** and **<PHOTOGALLERY_S3_BUCKET_NAME>** in lines 1 and 2, respectively, as shown below

After making the updates, upload the files from the 'staticwebsite' folder to the S3 bucket for the website.

10.Access the Photo Gallery application in a browser

- Access the URL of the static website for photo gallery application hosted on S3.
- Goto signup page and create a new user.
- Confirm the user's email.
- Goto login page and login using the user created above.
- Goto add photos page and add a new photo.
- Add a more photos and try browsing and searching for photos.

Challenges:

1. Competition of the main functionalities of the website as described in the instructions **(80 points)**
2. Allow users to delete a picture in their own gallery **(10 Points)**
3. Allow users to update the title, description and tags of the pictures after creation **(10 Points)**

Deliverables:

A video that shows the functions of your website, including user signup/login/logout, upload a picture, view pictures under the same tag. If you have completed the challenge tasks, also be sure to include the following example: a deletion of a picture, search for a picture, update of a picture's information, etc.. You can also show the database to prove that the picture has been deleted or modified.

Hints on Debugging Process

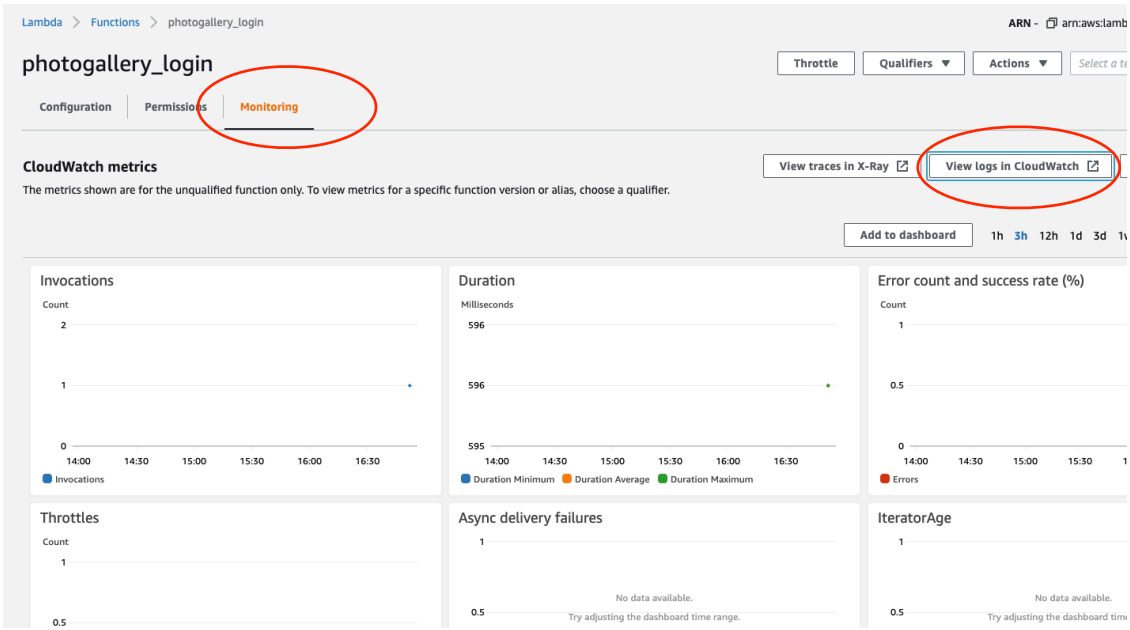
1. Test your code on localhost before uploading to cloud

```
[11:20] (/>ω<)/ ~ $ cd /Users/jolinchen/Desktop/Spring_2021/TA_ECE_4150/Lab1/staticwebsite
[11:20] (/>ω<)/ ~/Desktop/Spring_2021/TA_ECE_4150/Lab1/staticwebsite $ http-server
Starting up http-server, serving ./
Available on:
  http://127.0.0.1:8080
  http://10.0.0.12:8080
Hit CTRL-C to stop the server
```

Go to the root folder of the staticwebsite, type in command line *http-server* to set up HTTP server for testing. You can find the installation information of http-server at: https://www.npmjs.com/package/http-server. Then open the url as indicated in your terminal.

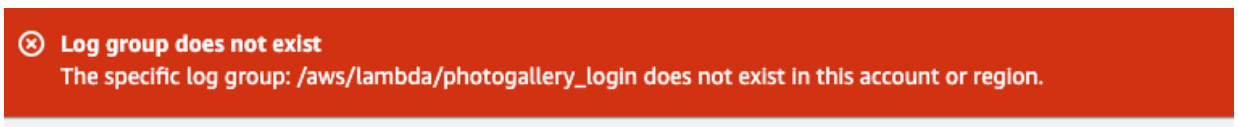
2. Use CloudWatch to monitor data flow

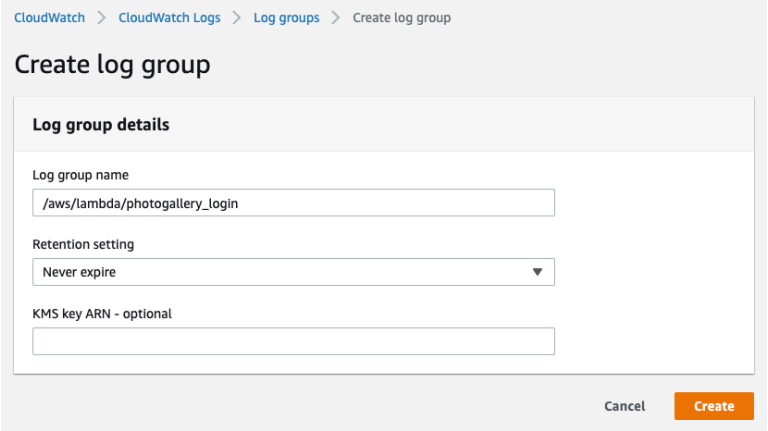
This is to locate problems occurring in lambda functions. Go to your lambda function list and choose any lambda function, go to "**Monitoring**" –> "**view logs in CloudWatch**".



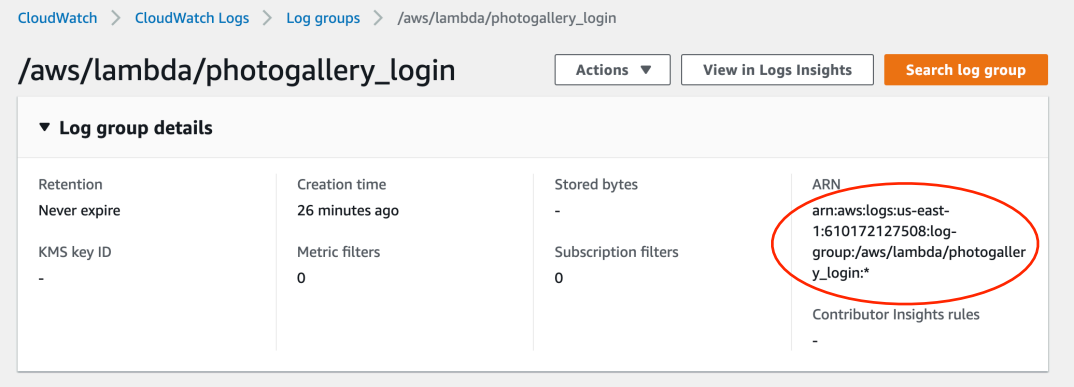
You have to create a log group in order to get the logs. To so do, in your newly opened CloudWatch Management Console page, copy the link above (in this case: group: **/aws/lambda/photogallery_login**):

Go back to Log groups, choose "**Create log group**", in Log group name, paste the link you just copied then create:

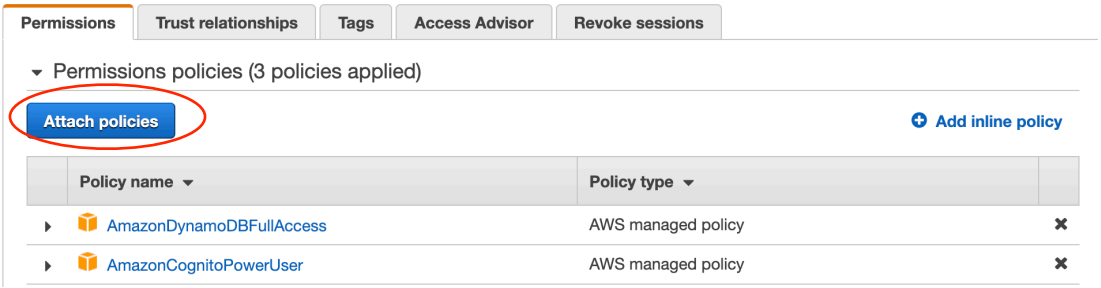




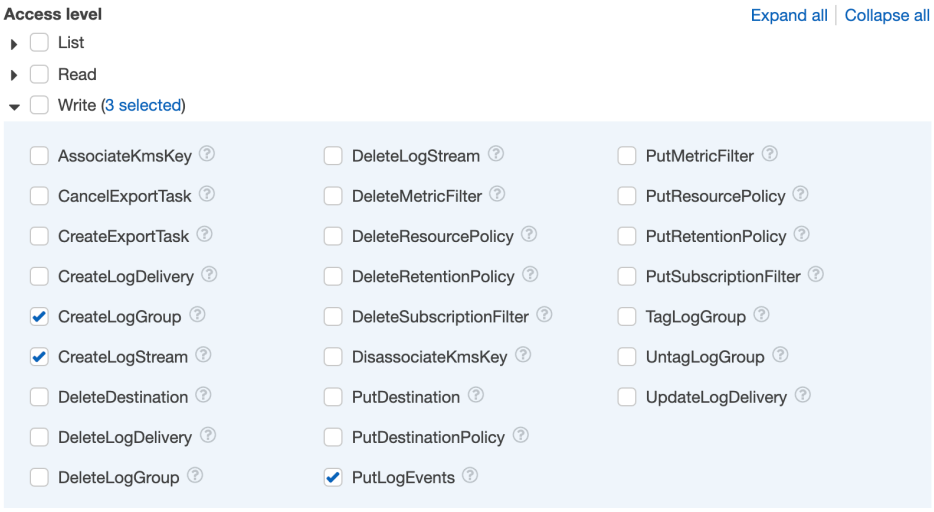
In your newly created log group, click on the dropdown button of "**Log group details**" and copy the ARN in it:



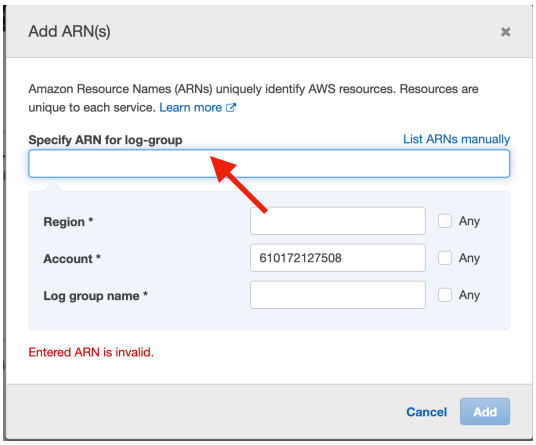
Now we need to grant write access to the log group. Go to IAM console and open roles. Go to **lambda_photogallery_role** you created before and choose "Attach policies":



Choose "**Create policy**". In Service, choose "**CloudWatch Logs**". In Actions - Write, click on the dropdown list and choose as indicated below.

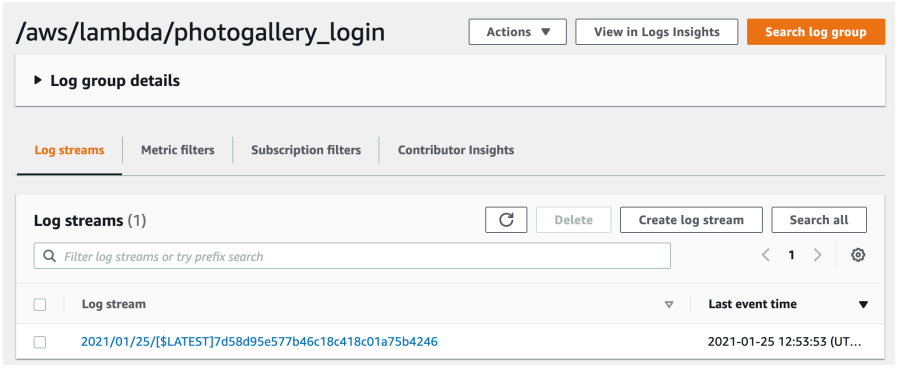


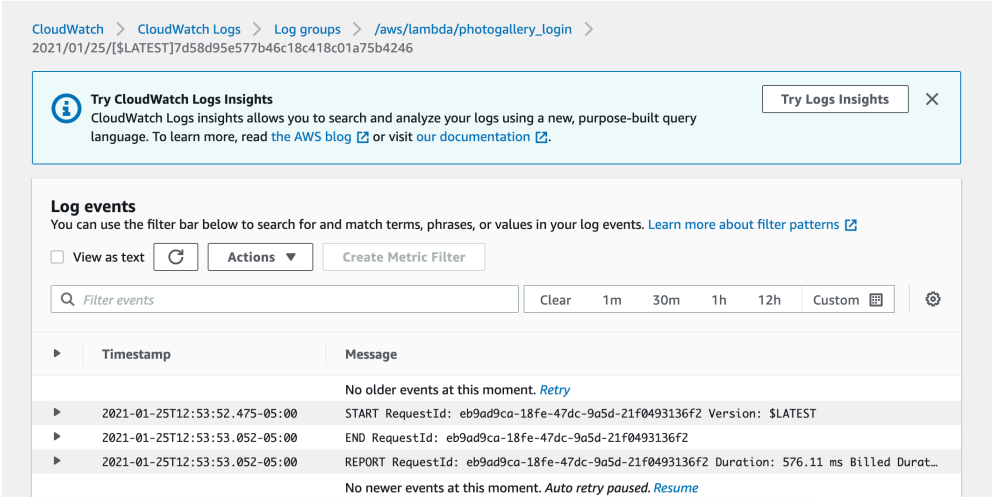
In Resources, choose "**Specific**", click on "**Add ARN**" in **log-group** and paste your copied ARN into it. After finishing everything, check on your json file, which should look like the screenshot below. Click on "**Review Policy**", name your new policy as you like and create. Attach the policy to your role after creation.




```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:us-east-1:610172127508:log-group:/aws/lambda/photogallery_login
                :*"
        }
    ]
}
```

Now as you enter your log group of chosen lambda function, you should be able to monitor and examine all the activities.


/aws/lambda/photogallery_login
Actions
View in Logs Insights
Search log group
Log group details
Log streams
Metric filters
Subscription filters
Contributor Insights
Log streams (1)
Delete
Create log stream
Search all
Filter log streams or try prefix search
1
Log stream
Last event time
2021/01/25/[$LATEST]7d58d95e577b46c18c418c01a75b4246
2021-01-25 12:53:53 (UT…


CloudWatch
CloudWatch Logs
Log groups
/aws/lambda/photogallery_login
2021/01/25/[$LATEST]7d58d95e577b46c18c418c01a75b4246
Try CloudWatch Logs Insights
CloudWatch Logs insights allows you to search and analyze your logs using a new, purpose-built query language. To learn more, read the AWS blog or visit our documentation.
Try Logs Insights
Log events
You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns
View as text
Actions
Create Metric Filter
Filter events
Clear
1m
30m
1h
12h
Custom
Timestamp
Message
No older events at this moment. Retry
2021-01-25T12:53:52.475-05:00
START RequestId: eb9ad9ca-18fe-47dc-9a5d-21f0493136f2 Version: $LATEST
2021-01-25T12:53:53.052-05:00
END RequestId: eb9ad9ca-18fe-47dc-9a5d-21f0493136f2
2021-01-25T12:53:53.052-05:00
REPORT RequestId: eb9ad9ca-18fe-47dc-9a5d-21f0493136f2 Duration: 576.11 ms Billed Durat…
No newer events at this moment. Auto retry paused. Resume