www.ine.com

Lab 4 - Serverless Capstone Project

SPECIAL NOTE: If this lab is accomplished outside the INE-supplied sandbox environment, INE is not and cannot be responsible for any charges from AWS for resources created by this lab.

IMPORTANT: Video Overview

Accompanying this lab are videos. Please watch the overview to understand the significant lab.

There are also walkthrough videos you can watch if you prefer. Those videos will track with these instructions. So, if you think you would rather follow along with an instructor versus reading the steps, take advantage of that option.

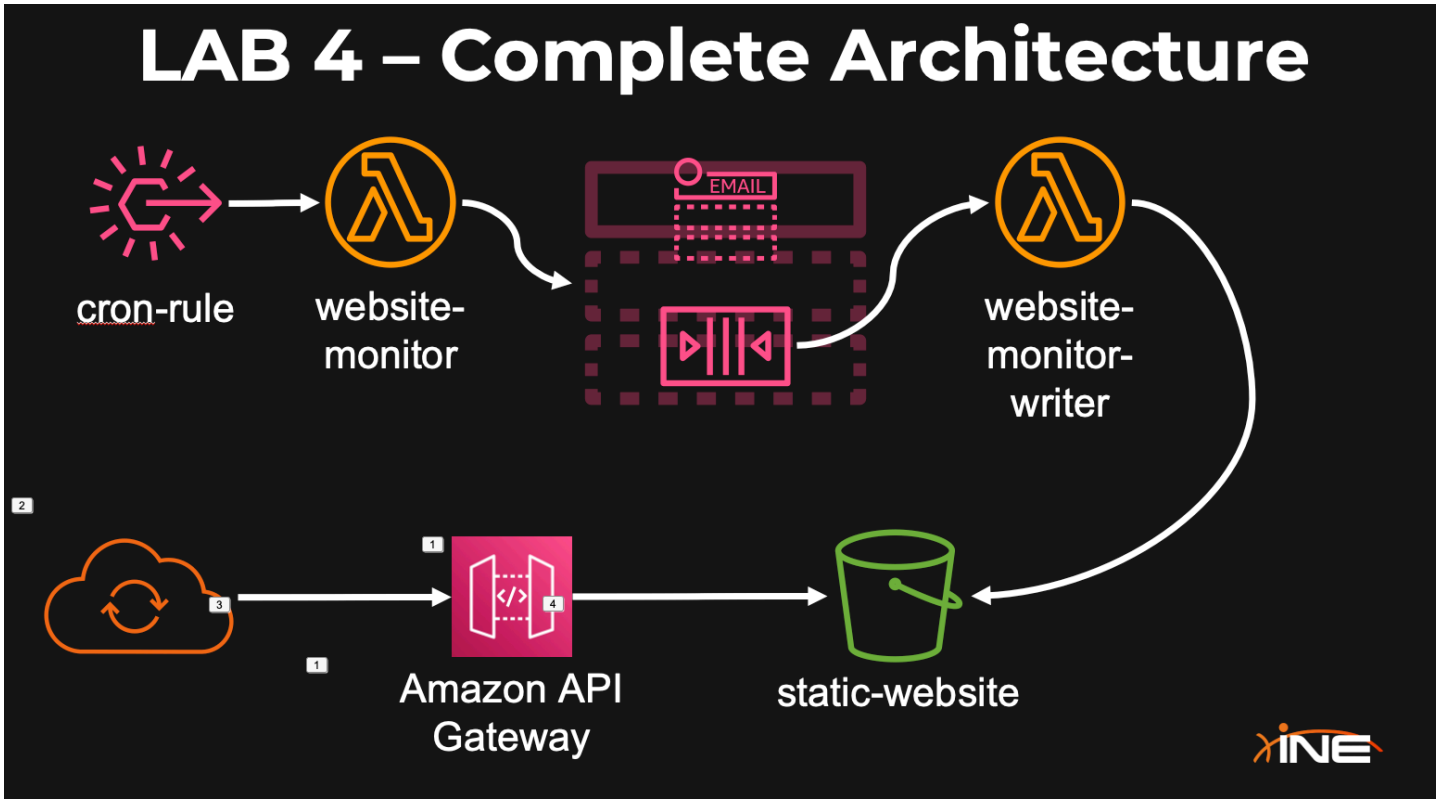
Scenario / Introduction

In this lab, you will implement the entire solution: an automated, serverless infrastructure for monitoring websites the boss likes to shop on. That means everything from the EventBrigde, Lambda functions to SNS and will include a new SQS queue, static website in S3, and an API Gateway to access the web page.

This lab will give you the experience of how, as a builder, you work in the console to develop architectural solutions in AWS.

Warning: This is a long lab. It could take anywhere from thirty minutes to just over an hour to complete. Regardless, take your time and understand what you are doing at each step. Don't just follow these instructions (especially if you use the lab instructions instead of the lab walkthrough video). Do your best to understand how each service works and how the overall architecture works together.

Here is an overview slide of the architecture you can use for reference:



1. Start Lab

From the course list, click the **Start Lab** button and follow the instructions for opening the lab.

2. Login

# 1	Login to the AWS Management Console using the student username and password.	Logged into the AWS console
# 2	In the search for services bar, enter **Lambda** and then select the Lambda service from the search list.	The Lambda console appears.
# 3	Verify you are in the *N. Virginia* region. Change to that region if necessary.	**N. Virginia** appears in the menu bar.

3. Build the Site Monitor Lambda function

Step	Instructions	Result
# 1	Navigate to the Lambda console and click the **Create function** button at the top right of the window.	The **Create function** page is displayed.
# 2	Leave the selection at the top set to **Author from scratch**.	NA
# 3	In the **Basic information** section, set the **Function name** to *website-monitor*	NA
# 4	Change the **Runtime** value to *Python 3.9*	NA
# 5	Expand the **Change default execution role** section. We will leave this set to **Create a new role with basic Lambda permissions**. (*Note the dialogue box under the options indicating that creating the role may take minutes. Keep this in mind during the build, so you do not mistake a long-running process with thinking the Lambda has failed to build correctly.*)	NA
# 6	Scroll down and expand the **Advanced settings** section.	The advanced settings for the Lambda are shown.
# 7	Select **Enable tags**, click the **Add new tag** button, and set the key to *Project* and the value to *site-monitor*	You can now search the infrastructure for elements related to your site monitor project.
# 8	Scroll to the bottom of the page and click the **Create function**.	The **Site-Monitor** Lambda function details page is displayed.

Add the code to the Lambda function.

Step	Instructions	Result
# 1	Still in the details page for our Lambda function, scroll down to the **Code source** section.	NA
# 2	In the right pane, for the file lambda_function.py, you will see the default code. Delete all code shown so the code pane is empty.	The code pane has no code.
# 3	Copy the source code below and paste it into the Lambda source code window.	The source code from site-monitor.py is now in the lambda source code section.

```
import os
import boto3
from datetime import datetime
from urllib.request import Request, urlopen

# temp_sites = ["https://www.amazon.com", "https://www.ebay.com"]   # Uncomment for local
execution
# temp_site_checks = ["amazon", "ebay"]                             # Uncomment for local
execution
TEMP_SITES = os.environ['sites']                                    # Uncomment for Lambda
execution
TEMP_SITES_CHECK = os.environ['site_check']                         # Uncomment for Lambda
execution
TEMP_SNS_ARN = os.environ['sns_arn']


def validate(res):
    EXPECTED = [item for item in TEMP_SITES_CHECK.split(",") if item]
    return_value = False
    for check in EXPECTED:
        if check in res:
            return_value = True
            break
    return return_value

def send_message_to_sns(message):
    sns = boto3.client('sns')
    response = sns.publish(
        TopicArn=TEMP_SNS_ARN,
```

```
        Message=message,
        Subject='Site Monitor'
    )
    print(response)

def check_site(event, context):
    SITES = [item for item in TEMP_SITES.split(",") if item]
    all_sites_reachable = True

    for site in SITES:
        NOW = str(datetime.now())
        print('Checking ', site)
        try:
            req = Request(site, headers={'User-Agent': 'AWS Lambda'})
            if not validate(str(urlopen(req).read())):
                print('SNS call from try section.')
        except:
            all_sites_reachable = False
            print('Check failed!')
            print('Call from the except section.')
            send_message_to_sns('Check failed for ' + site + ' at ' + NOW)
        else:
            print('Check passed!' + NOW)
            print('Call from the else section.')
        finally:
            print('End of loop run at: ' + NOW)

    if all_sites_reachable:
        send_message_to_sns('All sites are reachable.')

def lambda_handler(event, context):
    check_site(event, context)
```

Step	Instructions	Result
# 4	Click the **Deploy** button just above the top of the code window.	A success message appears at the top of the window.
# 5	At the tabs just above the code window (Code, Test, Monitor, Configuration, and so on), click on the tab for **Configuration**.	The Lambda configuration tab is displayed.
	Click **General Configuration** in the left menu, and then to the right click	

# 6	the **Edit** button. When the edit settings page is displayed, change the value for **Timeout** to 10 seconds. Then at the bottom click **Save**.	
# 7	In the left menu, click the menu item **Environmental variables**.	The environmental variables details page is shown to the right.
# 8	Click the **Edit** button, click the **Add environment variable** button, and then add the following key-value pairs: **Key**: site_check **Value:** Amazon,eBay **Key:** sites **Value:** https://www.amazon.com, https://www.ebay.com **Key:**sns_arn **Value:** 0. (*Note: You will update this value from 0 to your SNS ARN after you create it below.*)	NA
# 9	Click the **Save** button.	The Environmental variables details page is redisplayed but now shows to variables.
# 10	Go back to the tabs and click **Code**.	The code window for Lambda is displayed.
# 11	Click the **Test** button at the top of the code window.	The **Configure test event** page is displayed.
# 12	For **Event name** enter *Main-test*.	NA
# 13	Leave all other settings as is, scroll to the bottom of the page and click the **Save** button.	The Lambda details page is shown.
# 14	Click the **Test** button again to run the *Main-test* test.	An **Execution results** tab is opened.
# 15	On the **Execution results** pane, at the top, will appear an error. This is due to the fact that we have yet to configure the SNS ARN with a correct value. You can ignore this error.	Ignore the error at the top of the **Execution results** pane.
#	In the **Function Logs** section of the execution results, look for a line that says *Check passed* with the date and time shown immediately after. This indicates that the function successfully ran a test against both sites, which (hopefully) should be up and return a result.	NA

With that done, you can build the EventBridge and then revisit the Lambda and verify the trigger for the code.

---

3. EventBridge build and configure

Lab 2 covered building the EventBridge. We will build and configure it here and then go back to the monitor Lambda function and verify its trigger.

Search for EventBridge in the service search bar at the top. When it appears, right-click and open it in a new tab. (*Note:* You will find it useful to open a new tab for each service since we will be going between them as we progress.)

Step	Instructions	Result
# 1	Once in the EventBridge console, click the **Create rule** button at the right of the window.	The **Define rule detail** page is displayed.
# 2	Set the **Name** to *WebsiteMonitorScheduledCall*	NA
# 3	In **Description**, enter a description you like for your rule (it's your rule, after all ;)).	NA
# 4	Ensure the **Event bus** is set to *default*.	NA
# 5	For the **Rule type**, change the selection to **Schedule**.	NA
# 6	Click the **Continue to create rule** button.	The **Define schedule** page is displayed.
# 7	For the **Schedule pattern** option, select **A schedule that runs at a regular rate,...**	NA
# 8	For the **Rate expression**, set the value to *1* and click the drop-down for the units and change it to **Minutes**. Then click the **Next** button.	The **Select target(s)** is displayed.
# 9	For **Target 1**, ensure that the **AWS service** is selected.	NA
# 10	In the **Select a target** drop-down, search for and select **Lambda function**.	NA
# 11	In the **Function** drop-down, select **website-monitor**.	NA
# 12	Expand the **Additional settings** section, and for the **Retry attempts** set the value to *5*.	NA
# 13	Choose **Next** to display the **Configure tags** page. Create a new tag with a key value of **Project** and a value of *site-monitor*. Then click the **Next** button.	The **Review and create** page is displayed.
# 14	Review the details page, and at the bottom, click the **Create rule** button.	The **Rules** page is displayed, showing the new rule.

Now let's head back to the Lambda console and verify the trigger for our site monitor.

Step	Instructions	Result
# 1	Navigate to the Lambda console or switch to the tab where you currently have the Lambda console open.	The Lambda console is displayed.
# 2	Click on the entry for the site monitor service if you are not already in the details for the function.	The details for the site monitor are displayed.
# 3	You should now see the EventBridge rule listed as a trigger to the function. If not, reload the page, and the EventBridge trigger will appear.	
# 4	Wait two to three minutes and then go to the **Monitor** tab. You should see invocations of your function. If not, continue reloading the page for a few more minutes until you see them.	NA
# 5	Go back to the EventBridge console, select the **WebsiteMonitorScheduledCall,** and click the **Disable** button.	NA

You may note that the **Error Count & Success rate** chart is not zero. The error is occurring because of the unconfigured SNS ARN. We will update that value shortly.

We've got the first two elements of our architecture completed. Now on to the SNS queue.

---

4. Setup the SNS Topic and update the the website monitor Lambda

In this step, you build the SNS topic and then update the site monitor Lambda with the ARN (Amazon resource name) of the newly created SNS topic.

Step	Instructions	Result
# 1	Search for the SNS service in the service search bar. When it appears in the drop-down, right-click it and open it in a new tab. Once the SNS console is displayed, in the left menu, click **Topics**.	The Topics page is displayed.
# 2	Toward the top right of the page, click the **Create topic** button.	The Create topic page is displayed.
# 3	Change the **Type** to *Standard*.	NA
# 4	For the **Name** and **Display name**, enter *SiteMonitorAlarms*.	NA

# 5	Scroll down and expand the **Access policy** section. Leave the method set to **Basic**, set **Define who can publish...** to **Everyone,** and set **Define who can subscribe** to **Only requesters with certain endpoints**.	NA
# 6	In the textbox for endpoints, enter the domain for the email address you would like to receive notifications. For example, enter *@gmail.com* for Gmail, *@icloud.com* for iCloud, and so on.	You have the domain of your email address entered.
	Before moving on, scroll down the JSON preview to the right. Notice two things: 1) That Principal is set to *****. Could you modify this (if you change the method to **Advanced**) to a specific account or account? 2) Notice the rule *"StringLike": {"SNS:Endpoint" : "@gmail.com"}*. This is the definition for the endpoint domain. Do you see how you could modify this manually for any domain and multiple domains?	NA
# 7	Scroll to the bottom of the page to find the **Tags** section. Enter a new tag with a **Key** value *Project* and **Value** of *site-monitor*.	NA
# 8	At the bottom of the page, click the **Create topic** button.	The details page for SiteMonitorAlarms is displayed.
# 9	At the top of the page in the **Details** section, copy the ARN value for your topic. The value will look something similar to *arn:aws:sns:us-east-1::SiteMonitorAlarms*.	You will paste that value into the site monitor lambda.
# 10	Navigate back to the Lambda console and click on the website-monitor Lambda.	The details for the website monitor are displayed.
# 11	Click on the **Configuration** tab, and then in the left menu, click on **Environmental variables**, and then in the **Environmental variables** box click the **Edit** button.	The environmental variables editor opens.
# 12	Update the value for **sns_arn** with the value copied above.	The sns_arn environment variable's value is now the SiteMonitorAlarms ARN value.
		The function code page is

# 13	Click **Save** at the bottom of the page.	The function code page is redisplayed, and the function automatically redeploys. You should see a green banner at the top of the page if successful.

Now we set up the topic for our subscription just like we did in Lab 3. We will come back to add another subscription after we create the webpage writer Lambda function.

Step	Instructions	Result
# 1	Switch tabs or navigate to the SNS console & the SiteMonitorAlarms topic. On the SiteMonitorAlarms details page, click the **Create subscription** button.	The Create subscription page is displayed.
# 2	For the **Protocol**, click the drop-down and select **Email**.	The **Endpoint** textbox appears.
# 3	In the **Endpoint** textbox, enter the email address you want to use for this lab (from the same domain as the topic settings above).	NA
# 4	At the bottom of the page, click the **Create subscription** button.	The Subscription page is displayed.
# 5	Check the email account you you entered above. You should receive an email from *SiteMonitorAlarms* with the subject of *AWS Notification - Subscription Confirmation*. Open the email and click the *Confirm subscription* link.	A webpage will open with the text *Subscription confirmed!* showing at the top.
# 6	Close the opened page from AWS, and you can also delete the email from SNS.	

Okay...deep breath! You are doing great on this build. Next, let's set up the S3 bucket for our static website. You'll create that first so that when you create the Lambda function that writes the web page, you will have a target for the write operation.

---

5. Create and configure a static website in Amazon S3

5. Create and configure a static website in Amazon S3

Super simple setup here. But...

> **Warning:** *You are configuring a publicly accessible bucket. In the INE lab environment, that's all well and good. In a company, organization, or otherwise important organization's account, not so fast. Ensure that configuring such a bucket is in line with company policy (and won't get you fired builder).*
>
> *As for some 'Wisdom from the Road', if you must have public S3 buckets, use or create a new AWS Organization just for that purpose. At many companies your author worked with, that is the default policy. That way Security can ensure no buckets in those organization will ever be used for sensitive data.*

Step	Instructions	Result
# 1	Navigate to the S3 Console (*open in a new tab for convenience*).	The S3 console is displayed.
# 2	Click the **Create bucket** button.	The **Create bucket** page is displayed.
# 3	Name the bucket something containing *sitemonitor*. Since all S3 bucket names must be unique in the AWS partitions, you'll have to make the bucket name unique. Examples: sitemonitor-202212011900 mysuperdeluxesitemonitor downwebsitemonitor Be sure to make a note of your bucket name for future configurations steps.	
# 4	Ensure the **AWS Region** is set to *US East (N. Virginia) us-east-1*	NA
# 5	Scroll down and in the block for public access box deselect **Block *all* public access**.	NA
# 6	In the warning box that displays, click the checkbox to acknowledge you understand that this action could result in objects becoming public.	NA
# 7	Be sure to add tags with a **Key** of *Project*, and **Value** of *site-monitor*, and then scroll to the bottom of the page and click the **Create bucket** button.	The bucket list for the lab account you are using

		using appears.
# 8	In the bucket list, click the name of the bucket you just created.	The details for your bucket are displayed.
# 9	Click on the **Properties** tab and scroll to the bottom of the page to the **Static website hosting** box and click the **Edit** button.	The **edit static website hosting** page is displayed.
# 10	In the **Static website hosting** box select the **Enable** radio button.	The details for the static site are shown.
# 11	Leave all the settings as is except the **Index document** setting. For that, enter *index.html* into the textbox. Then scroll to the bottom and click the **Save changes** button.	The properties for the bucket are displayed.
# 12	Click on the **Permissions** tab and scroll down to the **Bucket policy** box and click the **Edit** button. Paste the policy shown below into the policy window. Be sure to include the leading and final curl braces and replace **YOUR-BUCKET-NAME-HERE** with the name of your bucket. When done, scroll to the bottom of the page and click the **Save changes** button. **NOTE:** Be absolutely sure that the end the Resource line is '/*'	

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicRead",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [

                "s3:GetObject"
```

```
            ],
            "Resource": [
                "arn:aws:s3:::YOUR-BUCKET-NAME-HERE/*"
            ]
        }
    ]
}
```

Step	Instructions	Result
# 13	Your bucket's details page is redisplayed, and the bucket is ready to host a static web page. Note that under the name is a pill-shaped callout with the text **Publicly accessible**.	NA
# 14	Click on the **Properties** tab and scroll to the bottom. Note that in the **Static website hosting** box, at the bottom, is a URL for your bucket. Although this will not be the official URL for your web page, it is another access point for the page.	NA

Okay, the bucket for our static website is ready to display information about our site monitor. Now we need to create the Lambda that will be called by our SQS queue and write the web page.

---

6. Build the page writing Lambda function

Now for our fancy Lambda function that will receive messages from SQS and write an HTML page to our S3 bucket.

Step	Instructions	Result

Step	Instructions	Result
# 1	Open the Lambda console page and click the **Create function** button.	The **Create function** page is displayed.
# 2	For the **Function name** enter *website-monitor-writer*.	NA
# 3	Change the **Runtime** to **Python 3.9**.	NA
# 4	Click **Create function** at the bottom of the page.	The details for the new Lambda are shown.
# 5	Scroll down to the code window and replace the default code show with the following code listing:	

```
import json
import boto3
import os
import datetime

S3_BUCKET = os.environ.get('BUCKET_NAME')

# ------------------------------------------------
def htmlify(subject, message):
  htmlCode = ''
  try:
    htmlCode = '<HTML><BODY><H1>' + subject + '</H1><br/>' + message + '</BODY></HTML>'
  except e:
    htmlCode = '<HTML><BODY><H1>Page render error.</H1><br/>An error occurred during page
rendering.</BODY></HTML>'
    pass

  return htmlCode
# ------------------------------------------------



# ------------------------------------------------
def lambda_handler(event, context):
    session = boto3.Session()

    file_content = ' '
    response_object = ' '

    for key in event['Records']:
      file_content = file_content + str(key)


      if 'Check failed' in file_content:
```

```
        messagePosition1 = file_content.index('Check failed')
        messagePosition2 = file_content.index('Timestamp') - 7
        file_content = file_content[messagePosition1:messagePosition2]
        response_object = htmlify("Unreachable website detected: ", file_content + 'UTC')
      else:
        response_object = htmlify('Website Check Complete: ', 'All configured websites are
reachable at ' + str(datetime.datetime.now()) + 'UTC')


    s3 = session.resource('s3')
    object = s3.Object(S3_BUCKET, 'index.html')
    object.put(Body=response_object, ContentType='text/html')
# ------------------------------------------------
```

Step	Instructions	Result
# 8	Once you have pasted the code, click the **Deploy** button.	The updated function is redeployed.
# 9	Click on the **Configuration** tab, and in the left menu scroll down and click on **Environmental variables**.	The variables for this function are displayed.
# 10	Click either of the **Edit** buttons shown and then, on the **Edit environment variables** page, click the **Add environment variable** button.	An empty Key-Value pair is displayed.
# 11	For the **Key** value enter *BUCKET_NAME*, and for the value, enter the name of your S3 static bucket created above. (*Note: You only need to enter the bucket name, the ARN is unnecessary since each bucket is unique.)	NA
# 12	Click the **Save** button at the bottom.	The **Configuration** tab is redisplayed with the new environmental variable.
# 13	Scroll back to the top of the page and on the right side find **Function ARN**. Copy the functions ARN. We'll need in in just a moment when creating the SQS queue.	

Awesome! We are almost there. The only two missing pieces are the SQS queue and our API Gateway. Let's get that queue up and running. When done, we will come back to this function and configure the trigger.

7. Build, configure, & test an SQS queue

This queue will convey messages from the SNS topic to our **website-monitor-writer** Lambda function. It will do this since we will configure SNS to send messages to it and then have the function automatically send messages to our function.

Step	Instructions	Result
# 1	Navigate to the SQS console page and click the **Create queue** button.	The **Create queue** page is displayed.
# 2	In the **Details** box enter *website-monitor-writer-queue*.	NA
# 3	Scroll to the bottom of the page and click the **Create queue** button.	The details page for the website-monitor-writer-queue is displayed.
# 4	Scroll down just a little, and in the tabs displayed, click on **Lambda triggers**, and then click the **Configure Lambda function trigger** button.	The **Trigger AWS Lambda function** page is displayed.
# 5	From the drop-down shown, find the entry for the **website-monitor-writer** function and select it.	NA
# 6	With that done click the **Save** button. **ERROR!!!**	You say "Error!!!"

> When developing a new solution builder, you will encounter security errors...A LOT! The error was intended to occur to build your experience dealing with just such problems.

The error message you recieve should be something similiar to the following:

`The provided execution role does not have permissions to call ReceiveMessage on SQS`

The execution role it is referring to is the IAM Role our *website-monitor-writer* is condfigured to use. Let's fix that!

Update permissions for website-monitor-writer function's execution role

Step	Instructions	Result

Step	Instructions	Result
# 1	Open a new tab, enter *aws.amazon.com*, on the page that opens click **Sign into the Console** it the top right, and then navigate to the Lambda console page.	The Lambda console page is displayed.
# 2	The in the list of function click on **website-monitor-writer**.	The details for your Lambda function are dispalyed.
# 3	Click on the **Configuration** tab, and then in the left column, click **Permissions**.	NA
# 4	At the top of the Permissions tab is the **Execution role**. Click the role name value to open the IAM console to that role. Do that now: click the name of the role name.	A new tab opens the IAM console directly to the role used by the website monitor writer function.
# 5	On the **Permissions** tab shown is the **Permissions policies** list. There should only be one attached policy at this time. To the right of the list is the **Add permissions** button. Click that button, and from the dropdown list, click **Attach policies**.	The permissions policies list appears.
# 6	In the search box (directly under the text **Other permissions policies**), enter the text *SQS* and hit Enter.	Three policies will be shown.
# 7	In the policies that appear, select **AmazonSQSFullAccess**. Clear the filter for SQS and then search again, and this time look for *S3*. In the listing that appears, select the policy **AmazonS3FullAccess**.	
# 7	Click the **Attach policies** button below.	The details for the execution role are displayed with the new SQS and S3 permissions added to the **Permissions policies** list.
# 8	Switch back to your SQS tab, click **Save** again and the update will proceed successfully.	

It may take several minutes (up to five minutes in fact) for the IAM service to update the execution role.

...and such is the life of a builder. Well done!

Remediate access policy and message retention time for the SQS queue

There are another 2 problems that are less obvious to see that need to be fixed before proceeding. The only principal with access to the queue is the root for account you are in. This means that the SNS queue you created will not be able to write onto the queue. In addition, messages will stay in the queue for much longer than we would like after being processed. Let's fix both of them!

Step	Instructions	Result
# 1	Navigate to the SQS console (or switch to it if you already have it open in a tab.)	NAl
# 2	In the list of queues click on your **website-monitor-write-queue** queue.	The details for the queue are displayed.
# 3	Near the top of the page find and the click the button labeled **Edit**.	The details for the for the queue are displayed.
# 4	In the **Configuration** box, on the right side, is the setting for **Message retention period**. Change it's value to **2** and the time increment to **Minutes**.	NA
# 5	Scroll down more til you see the **Access policy** window. Note the first principal listed is the root account. Replace that value with just a star, that is, *****. It should look similar to the list below. Then scroll down and click the **Save** button.	The details page for the function are displayed.

Here's the access policy from step # 5 above.

```
{
  "Version": "2008-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__owner_statement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "SQS:*",
      "Resource": "arn:aws:sqs:us-east-1:0123456789012:website-monitor-writer-queue"
    }
  ]
}
```

> **STOP!!!!**
>
> Think about everything you have completed up til now. Is there anything missing? Sure, I'm going to tell you what is missing, but as a builder, it is going to be your responsibility to ensure there are no problems. So...what might be missing? Think on that for a few minutes and then move on.
>
> The missing element: currently the only subscription we currently have in our topic if for email. At this time, there is nothing that will send messages to your new SNS queue. Let's add the subscription so the messages flow.

Adding a subscription for the new SQS queue

Step	Instructions	Result
# 1	Navigate to, or if you have it open in another tab, switch to the SNS console.	The SNS console page is displayed.
# 2	In the left menu click **Topics**, and then in the topics list click **SiteMonitorAlarms**.	The details for the subscription are shown.
# 3	In the **Subscriptions** box at the bottom note that you have the single subscription to your email address. Click the **Create subscription**.	The **Create subscription** page is displayed.
# 4	On the **Create subscription**, click the **Protocol** dropdown and select **Amazon SQS**.	NA
# 5	In the **Endpoint** dropdown select the ARN for the **website-monitor-writer-queue**, then, at the bottom of the page, click the **Create subscription** button.	The details page for the subscription is displayed.

Now, you have only one more component...the Amazon API Gateway!

---

8. Building and configuring the API Gateway

In this final step of our lab, we will create two API endpoints: an HTTP and a REST endpoint. You will start by creating and testing the HTTP endpoint. Then you will create a REST endpoint...that won't work as expected. You will then review the configuration of the REST endpoint and resolve the issue.

But first...the HTTP endpoint!!!

Building the HTTP API Gateway

Step	Instructions	Result

Step	Action	Result
# 1	Navigate to the API Gateway console page.	The API Gateway page is displayed.
# 2	At the top right of the screen, click the **Create API** button.	NA
# 3	From the list of APIs that can be created, in the **HTTP API** box, click the **Build** button.	The **Create an API** page is displayed.
# 4	In the **Create and configure integrations** box, click the **Add integration** button.	The **Integrations** drop-down is displayed.
# 5	Click the **Integrations** dropdown and select **HTTP**. Leave the **Method** value set to **Any** and paste your static website URL into the **URL endpoint** textbox (S3 Console --> --> Properties tab --> bottom of the page in the *Static website hosting* section).	NA
# 6	In the **API name** textbox name this API *HTTP-Website-Monitor-Gateway* and then click the **Next** button.	The **Configure routes** page is displayed.
# 7	On the **Configure routes** page click **Next**.	The **Define stages** page is displayed.
# 8	On the **Define stages** page leave everything as is and click the **Next**.	The **Reveiw and create** page appears.
# 9	On the **Review and create** page review the settings and then click the **Create** button.	The details page for the API Gateway is displayed.
# 10	To test, in the **Stages for HTTP-Website-Monitor-Gateway** box click or copy and paste the **Invoke URL** value into a new browser tab.	Your static site appears with nothing to report.

Okay... that has our HTTP gateway up and running. Now, we could stop there, but no! You should also have

experience creating a REST endpoint (and fixing a common problem as well). So now, let's build another API Gateway.

Building the REST API Gateway

Step	Instructions	Result
# 1	In the left menu of the API Gateway console click the link (at the top) for **APIs**.	The APIs deployed to this account are shown. You should see the HTTP gateway you just created.
# 2	Click the **Create API** button in the top right. In the API listing click the **Build** button in the **REST API** box.	The **Choose the protocol** page is displayed.
# 3	Leave **REST,** and **New API** options select. In the **API name** box enter *REST-Website-Monitor-Gateway*, enter a description if you would like, leave the **Endpoint Type** set to **Regional** and click the **Create API** button at the bottom.	A blank **Methods** page is displayed.
# 4	In the **Resources** column (between the menu on the left and the Method area), click the **Actions** button and select **Create Method**.	A forward slash will appear in the **Resources** list with a drop-down just below.
# 5	Click the drop-down that appeared and select **Any**. (This will cause your gateway to take any method sent to it [GET, PUT, POST, and so on] and send it to the static website). Then click the little circle with a check (shown below). 	The details for the ANY method are populated on the right-hand side of the console page.
# 6	In the **Integration type** select **HTTP**, enter the URL for your static website in the **Endpoint URL** textbox, leave all other values as is, and then click **Save**.	The **Method Execution** section of the page is show the flow of calling the static website.
	Note: At this point nothing has been deployed. All you have done is told AWS is how you want the gateway to work. Next, we will actually deploy the gateway into this account	
# 7	Click the **Actions** button again, and select **Deploy API**.	The **Deploy API** dislog box

		appears.
# 8	In the **Deployment stage** drop-down select **[New Stage]**. In the texboxes that appear, enter *Dev* for **Stage name**, enter descriptions if you would prefer, and then click **Deploy**.	The **Dev Stage Editor** is displayed.
	Now at this point you have deployed your REST API gateway. But let's do a little common-sense tuning before we continue. This API should not have too much traffic. To ensure we keep it that way, let's take advantage of the throttling capability. We will limit our API to just 10 requests per second.	
# 9	In the **Default Method Throttling** section, set the **Rate** to *10* and the **Burst** to *5*, and then at the bottom click the **Save Changes** button.	Thought the page does not change, the changes will have been saved.
# 10	At the top of the page is an invokation URL for your gateway. Click it and examine the page that is opened.	....not....good!

Here's what I got when I clicked the link:

```
<HTML><BODY><H1>Nothing to report</H1><br/>No down websites detected.  Last updated at:
2022-12-05 18:23:38.306416</BODY></HTML>
```

So, that's the text we expected, but why isn't it displaying as HTML? This isn't a mistake/error, it's simply how the API Gateway is configured by default. Instead of sending *text/html*, which would be displayed as HTML, the endpoint is configured for *application/JSON*. Let's change that.

Configure the REST API to serve HTML

Step	Instructions	Result
# 1	Switch tabs back to the API Gateway console and your REST API.	NA
# 2	In the left menu entry for your API (should be similar to **API: REST-Website-M..**), just under that, click **Resources**.	The **Methods** for your Gateway appear.
# 3	In the far right pane will be a single card for your APIs methods labeled **ANY**. Click on that text **ANY** in the banner of the card to open the method execution.	The **ANY - Method Execution** page will be displayed.

Here you see the round trip for a call to your API's ANY method. We have:

- Method Request
- Integration Request

- Integration Response
- Method Response

And it's the Method Response that's the problem! Notice that the **Models** value is *application/json*, which means we are not returning HTML. Let's fix it!

Step	Instructions	Result
# 4	Click on the text **Method Response** on that card's banner.	The details for **ANY - Method Response** appears.
# 5	Click the small arrow just to the left of the text **200** in the **HTTP Status** table.	The **Response Headers for 200** and **Response Body for 200** appears.
# 6	Under **Response Body for 200**, on the line for content type *application/json*, click the small pencil on the right side of the entry.	The line for application/json will become an editable textbox.
# 7	Change the value from application/json to **text/html** and then click the check circle just to the right.	The update is written.
	Now, with the update, let's redeploy.	
# 8	Click the **Actions** button at the top of the page and select **Deploy API** from the dropdown.	The **Deploy API** dialog box is displayed.
# 9	From the **Deployment state** dropdown select **Dev** and then click the **Deploy** button.	The update will be deployed and the **Dev Stage Editor** dialog will be displayed.
# 10	Click the **Invoke URL** at the top of the page. **NOTE:** When the new tab opens the page may look the same as before. In our experience, reload the page, and the HTML will be rendered properly.	

Now the fun part!!!

Step	Instructions	Result

# 1	Navigate back to the site-monitor Lambda function.	NA
# 2	In the function go to **Configuration**, and in the left menu click **Environmental variables**. Edit the sites variable to include an unreachable URL (i.e.: https://www1.1234qwerasdfzxcv.com). Click **Save** when finished.	NA
# 3	Navigate to the EventBridge console, in the left menu click **Rules**, select the **WebsiteMonitorScheduledCall** checkbox, and then click **Enable**.	NA
# 4	Navigate to the API Gateway console, click on either the HTTP or REST endpoints for the website monitor, and click the invitation URL.	The website monitor page is displayed
# 5	After about a minute, refresh the page. It should report an unreachable website. In addition, you should have an email as well.	NA
# 6	Go back to the site-monitor Lambda function and update the **Environmental variables** by removing the bad URL.	NA
# 7	Wait a minute or two, reload the web page for your static website, and it should now report no issues. You will receive an email reporting the same.	

...and it's all serverless...scales with demand...requires no patching...and is cheap to run!

Conclusion

That, I'm sure, is one of the biggest labs you may have ever accomplished. But it's also realistic. When building a test concept in the console, we often have to continually backup and update a previously created service with values from a service created later.

Next - Challange

Can you do this entire lab again without using the instructions? Simply put, if you are comfortable enough with working through architectures just using your own knowledge, then you really have a strong grasp of the services used by and the concept of serverless architectures in AWS.

Cleanup

Close all tabs running the lab, and finally, on the lab control page, end the lab.

Thanks!