



Lambda Automation Deepdive

Activity Guide

We'll be using AWS remote environments for our labs. Every student MUST HAVE AN AWS ACCOUNT CREATED IF YOU'D LIKE TO FOLLOW ALONG IN ACTIVITIES DURING CLASS.

NOTE: The code you'll use in this class is located at <https://github.com/rich-morrow/autom8d-deepdive>. You can install with:

```
git clone https://github.com/rich-morrow/autom8d-deepdive.git
```

Also helpful to know:

- *It's imperative that in each lab, you pick one Region and stay in that Region throughout the entire activity. If you frequently move between Regions in your normal AWS account, please always double check that the console didn't switch you to another Region mid-activity (it may do that if you use multiple Regions outside of class)*
- *For internet browser, we strongly recommend using Google Chrome or Firefox. The AWS web gui which we'll use extensively doesn't play Microsoft's game of "write a completely different version of your website to deal with MS' inability to abide by internet standards like Javascript", so avoid IE, Edge, or whatever Microsoft is calling their crappy browser today.*
- *In Linux (which we'll use exclusively), you can "Tab Complete" long filenames. For example, if I had a directory named "i-am-a-super-long-hard-to-type-directory", I can cd into it by typing "cd i-am-a" then just press tab. If there are files or directories that match those first few characters, linux will list them out for you, and if there's only a single match, it will automatically fill in the remaining characters for you. Carpal Tunnel BEGONE!!!*
- *Commands you're asked to enter on the terminal are formatted as so in this document:*

```
cd ~/Desktop
```

Activity #1: Automated WAF rule updates via GuardDuty finding notification

Git clone the code locally

At the bottom of the window, you'll see a terminal. In the terminal, enter:

```
git clone https://github.com/rich-morrow/autom8d-deepdive.git
```

```
#then cd into the codebase, activity1
```

```
cd autom8d-deepdive/activity-1
```

NOTE: This lab has only been tested to work in the us-east-1 (Virginia) region. Please make sure you switch to that Region prior to starting the lab.

NOTE2: If you wish to see WAF rules enforced, you'll need to also set up two EC2 instances (across two AZs) and put an ALB load balancer (balanced across those AZs) in front of them, all open to port 80, 443, and 22 access to the world (CIDR 0.0.0.0/32). If you do not perform this step, you'll not be able to see the rules actually block your traffic.

NOTE3: Prior to starting the lab, you'll want to copy the two zipfiles in “~/activity-1/archives” to an S3 bucket (and subfolder... the code does need a subfolder) in us-east-1. These files will need to be publicly readable. Note the bucket name and folder for these files

NOTE4: This lab uses some open source code from AWS. You can find the origin of that code here: <https://github.com/aws-samples/amazon-guardduty-waf-acl.git>, and a blog writeup (that references the older WAF, but still holds true as far as functionality) here: <https://aws.amazon.com/blogs/security/how-to-use-amazon-guardduty-and-aws-web-application-firewall-to-automatically-block-suspicious-hosts/>

Launch CloudFormation template to create resources

At the bottom of the window, you'll see a terminal. In the terminal log into your AWS console and browse to the [Cloudformation console](#), and verify that you are in the us-east-1 (N. Virginia) Region. In the console, press “Create Stack” button, then choose “Upload a template file”, and provide the template in the “~/activity-1/templates/guarddutytoacl.template” (wherever you cloned the github repo code for this class). Press “Next”.

On the next page in the flow, under “Stack name”. Give this stack the name “activity-1-gd-to-acl”. For notification email, put in an email you have ready access to while doing the lab. Enter the bucket name and folder of where you uploaded the assets (two lambda functions that will set and delete an IP block) into prior to starting the lab. You can also use rich's upload area:

Bucket: advanced-lambda-automation

Folder: activity-1-assets/

Press “Next” to go to the Configure Stack Options page. Press “Next” on this page as well.

On the final confirmation page, tick the checkbox at page bottom that says “I acknowledge that AWS CloudFormation might create IAM resources.” and press “Create stack”

The stack will take approximately 15minutes to create, so while it's working your instructor may guide you through the Events, Resources, Parameters, and Templates tabs.

NOTE: Do Not Proceed in the activity until you see “Create Complete” on the ENTIRE STACK.

Also, now check your email for a “SNS Subscription Confirmation” email from AWS, and click the link inside of that email to confirm that you wish to receive emails from AWS.

The screenshot shows the AWS CloudFormation 'Specify Stack Details' page. The 'Stack name' field is filled with 'activity-1-gd-to-acl'. Under 'GD2ACL Configuration', the 'Notification email (REQUIRED)' field is set to 'rich@quickcloud.com'. In 'Artifact Configuration', the 'S3 bucket for artifacts' is 'advanced-lambda-automation' and the 'S3 path to artifacts' is 'activity-1-assets/'. Under 'Other parameters', both 'CloudFrontWebACL' and 'RegionalWebACL' are set to 'false'. At the bottom right, there are 'Cancel', 'Previous', and a large orange 'Next' button.

Browse the resources that the template launched.

Open tabs in your browser for VPC (then browse to “Security→Network ACLs” in the left rail nav), Lambda, AWS WAF, DyanmoDB, and EventBridge. Notice the various assets that your CF template created & how they are interconnected. Note that there is currently no WACL and that there were two IPsets created under WAF.

Your instructor will now browse through the Lambda function & describe the function of the various piece-parts and how we may want to extend / enhance in a real production environment.

Your instructor will also describe how the template set up the hooks from GuardDuty to fire the Lambda function (via an EventBridge hook).

Set up a test event.

Because GuardDuty findings can sometimes take minutes to generate, we’re going to kick ours off with test of our Lambda function. In order to do that, we’ll have to first configure a test object representing the JSON ‘events’ object that lambda would get passed in the case of a real GD finding occurring. Pull up your Lambda function tab, browse to the “guardduty_to_acl_lambda.py” script, and open the“Code” tab as show below.

The screenshot shows the AWS Lambda Code editor interface. The top navigation bar includes tabs for Code, Test, Monitor, Configuration, Aliases, and Versions. The Code tab is selected. Below the tabs is a toolbar with File, Edit, Find, View, Go, Tools, Window, a Test dropdown, and Deploy buttons. A sidebar on the left shows the environment variables and a code editor window containing the Python script. The script imports boto3, math, time, json, logging, os, and botocore.exceptions. It defines a logger and sets its level to INFO. It then defines several environment variables: API_CALL_NUM_RETRIES, ACLETABLERESET, REGIONAL_IP_SET, CLOUDFRONT_IP_SET, and REGIONAL_IP_SET. The code ends with a copyright notice. To the right of the code editor is a message about invoking the function without saving an event. Below that is a Test event action section with Create new event and Edit saved event buttons. The event name is set to test-event. At the bottom is an Event JSON panel showing a JSON object with various fields like port, portname, localportdetails, protocol, blocked, resourceRole, additionalInfo, sample, eventFirstSeen, eventLastSeen, archived, count, severity, createdAt, updatedAt, title, and description.

```
1 # Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
2 #
3 # MIT No Attribution
4 # Permission is hereby granted, free of charge, to any person obtaining a copy of this
5 # software and associated documentation files (the "Software"), to deal in the Software
6 # without restriction, including without limitation the rights to use, copy, modify,
7 # merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
8 # permit persons to whom the Software is furnished to do so.
9 #
10 # THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
11 # INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
12 # PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
13 # HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
14 # OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
15 # SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
16
17 import boto3
18 import math
19 import time
20 import json
21 import logging
22 import os
23 import json
24 from botocore.dynamodb.conditions import Key, Attr
25 from botocore.exceptions import ClientError
26
27 logger = logging.getLogger()
28 logger.setLevel(logging.INFO)
29
30
31 #-----
32 # Variables
33 #-----
34
35 API_CALL_NUM_RETRIES = 1
36 ACLETABLERESET = os.environ['ACLETABLERESET']
37 REGIONAL_IP_SET = os.environ['REGIONAL_IP_SET']
38 CLOUDFRONT_IP_SET = os.environ['CLOUDFRONT_IP_SET']
39 REGIONAL_IP_SET = os.environ['REGIONAL_IP_SET']
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
```

To invoke your function without saving an event, modify the event, then choose Test. Lambda uses the modified event to invoke your function, but does not overwrite the original event until you choose Save changes.

Test event action

Create new event Edit saved event

Event name: test-event

Event JSON

```
117     "port": 32794,
118     "portname": "Unknown"
119   },
120   "localportdetails": {
121     "port": 22,
122     "portname": "SSH"
123   },
124   "protocol": "TCP",
125   "blocked": false
126 },
127   "resourceRole": "TARGET",
128   "additionalInfo": [
129     {
130       "sample": true
131     }
132   ],
133   "eventFirstSeen": "2018-05-11T14:56:39.976Z",
134   "eventLastSeen": "2018-05-11T14:56:39.976Z",
135   "archived": false,
136   "count": 1
137 },
138   "severity": 2,
139   "createdAt": "2018-05-11T14:56:39.976Z",
140   "updatedAt": "2018-05-11T14:56:39.976Z",
141   "title": "198.51.100.0 is performing SSH brute force attacks against i-999999999. ",
142   "description": "198.51.100.0 is performing SSH brute force attacks against i-999999999. "
143 }
```

Open the contents of

“`~/activity-1/templates/gd2acl_test_event.json`” in a text editor and browse for “subnetId”. Replace the text “Replace with valid Subnet ID” with a valid subnet from one of your VPCs in this region.

To the right of the orange “Test” button back on the main Lambda page, click the downward facing triangle & choose “Configure test event”. In the popup that appears, give the event the name ‘test-event’ and then paste the “SubnetID” replaced

contents of “~/activity-1/templates/gd2acl_test_event.json” over the 4 lines of boilerplate code that is prepopulated. & press the orange “Save” button at the lower right.

Back in the main view of Lambda, press the “Test” button proper. This should leave you with a screen that looks similar to this:

```
Tools Window Test | Deploy
guardduty_to_ac X Execution result X
Execution results
Test Event Name test-event
Response null
Function Logs
View Log
{
  "value": "GeneratedFindingInstanceTagValue?"}, {"key": "GeneratedFindingInstanceTag?", "value": "GeneratedFindingInstanceTag?"}, {"key": "GeneratedFindingInstanceTag?", "value": "GeneratedFindingInstanceTag?"}
[INFO] 2022-05-10T14:33:23.510Z 011eb5ca-37e8-4f99-9eb3-9eb0849a1975 log -- gd2acl attempting to process finding-data: instanceID: i-09999899 - SubnetID: subnet-05febaaa - RemoteHostIP: ['198.51.100.0/32']
[INFO] 2022-05-10T14:33:23.724Z 011eb5ca-37e8-4f99-9eb3-9eb0849a1975 log -- adding new rule 71, HostIP 198.51.100.0, to NACL acl-0e128015.
[INFO] 2022-05-10T14:33:23.724Z 011eb5ca-37e8-4f99-9eb3-9eb0849a1975 log -- adding new rule 71, HostIP 198.51.100.0, to NACL acl-0e128015.
[INFO] 2022-05-10T14:33:23.942Z 011eb5ca-37e8-4f99-9eb3-9eb0849a1975 log -- successfully add DDB state entry for rule 71, HostIP 198.51.100.0, NACL acl-0e128015.
[INFO] 2022-05-10T14:33:23.942Z 011eb5ca-37e8-4f99-9eb3-9eb0849a1975 log -- rule count for NACL acl-0e128015 is 1.
[INFO] 2022-05-10T14:33:23.942Z 011eb5ca-37e8-4f99-9eb3-9eb0849a1975 log -- adding regional and CloudFront waf IP set entry for host, ['198.51.100.0/32'] from CloudFront Ip set CloudFrontBlocklistIPSet
[INFO] 2022-05-10T14:33:23.992Z 011eb5ca-37e8-4f99-9eb3-9eb0849a1975 log -- creating waf object
[INFO] 2022-05-10T14:33:23.992Z 011eb5ca-37e8-4f99-9eb3-9eb0849a1975 log -- waf_update_ip_set c9c658ab-8ca7-4ade-82ee-5aacccef8b58 IP ['198.51.100.0/32'] - type REGIONAL successfully...
[INFO] 2022-05-10T14:33:24.166Z 011eb5ca-37e8-4f99-9eb3-9eb0849a1975 log -- waf_update_ip_set 2f1ceef1-4e10-43d1-806a-6ebf71ad6f59 IP ['198.51.100.0/32'] - type CLOUDFRONT successfully...
[INFO] 2022-05-10T14:33:25.701Z 011eb5ca-37e8-4f99-9eb3-9eb0849a1975 log -- send notification sent to SNS Topic: arn:aws:sns:us-east-1:143230255374:guardduty-to-aci-WORKS-GuardDutytoACLSNSTopic-B03H
[INFO] 2022-05-10T14:33:25.701Z 011eb5ca-37e8-4f99-9eb3-9eb0849a1975 log -- processing GuardDuty finding completed successfully
REPORT RequestId: 011eb5ca-37e8-4f99-9eb3-9eb0849a1975 Duration: 2287.43 ms Billed Duration: 2288 ms Memory Size: 1024 MB Max Memory Used: 83 MB
Request ID
011eb5ca-37e8-4f99-9eb3-9eb0849a1975
```

Validate blacklisting rules

Verify that the test that you just ran:

- Sent you an email with the Subject line “AWS GD2ACL Alert”
- Entered a rule (probably “71”) in the NACL associated with the subnet you specified, blocking IP address 198.51.100.0/32 .
- Entered an item in the DynamoDB table you created, also for address 198.51.100.0/32
- Entered an IP for the “RegionalBlocklistIPSetV4-XXXXXX” IPSet that was created earlier

RESOLVE ANY OF THESE ISSUES PRIOR TO PROCEEDING IN THE ACTIVITY

Remove rules from NACL, DDB

Because our Lambda code is triggered from the pre-existing rules in the NACL or DDB, let’s delete those entries now & then verify that they are gone. Your instructor will remove the entries from NACL and DDB.

Verify reachability of instances, load balancer

If you created instances and a load balancer, verify that all are publicly accessible.

Create WAF Rule, include IPSet, associate with ALB

Our initial CF template setup didn't actually create the WAF rule, so let's go about doing that now. In your WAF console, click "Web ACLs" on the left rail nav & press the "Create web ACL" orange button in the main page. On the next page that appears, give your WACL the name "activity-1-wacl", accept all other defaults and press "Next". Add the Load Balancer, if you set one up. On the "Add rules and rule groups" page, choose "Add rules → Add my own rules and rule groups". On the page that pops up, choose "IP Set", pick the "IPV4" IP Set, and give the rule the name "blacklist-ips". Click "Add rule", "Next", "Next", "Next" to get to the final page, then press "Create Web ACL" on the final page.

Blacklist your own IP

Browse to <http://whatismyip.com> & record the IP address that your machine is known as on the internet.

Open a local copy of your "gd2acl_test_event.json" in a text editor. Making sure to keep the "Subnet ID" replacement you made previously, do a global replace of "198.51.100.0" with your actual IP address. There should be 4 replacements total.

Back in the Lambda interface, overwrite your "test-event" with this new event object. When you run a test this time, it will actually show your IP address as the culprit, and because we've now associated this WACL with your load balancer, you should immediately be blocked from seeing the instances or the Load Balancer.

Questions:

1. Why are the instances blocked (there's no WACL associated w/them)?
2. Is there any way we could programmatically test this function without causing a real finding to occur?
3. How could we improve our Lambda function?

***** END OF ACTIVITY #1 STOP HERE!!! *****

You now have a good understanding of everything involved with using AWS Lambda with to automatically respond to GuardDuty findings and implement a WAF rule block.

Activity #2: Automated EC2 and S3 scanning and remediation using Inspector, Macie and Lambda action

Recording will follow

This being the first run of the class, we've run out of in-class time to perform all labs. Rich will follow up with a recording of this activity (and updated instructions) no later than Wed, May 25.

***** END OF ACTIVITY #2 STOP HERE!!! *****

You can now hook up a toolchain to use Inspector and Macie with Lambda remediations on EC2 and S3 violations.

Activity #3: Setting up a fully automated DevOps pipeline using Lambda triggers

NOTE: This lab has only been tested to work in the us-east-1 (Virginia) region. Please make sure you switch to that Region prior to starting the lab.

NOTE2: This lab uses some open source code from AWS Quickstarts. You can find the origin of that code here: <https://github.com/aws-quickstart/quickstart-trek10-serverless-enterprise-cicd#readme>, and a blog writeup here: <https://aws.amazon.com/quickstart/architecture/serverless-cicd-for-enterprise/>. Your instructor will browse to the blog writeup, and go through the “View Deployment details” to set up your environment.

Start EC2 instance

If you’re not running linux (or a linux shell via MacOS/Darwin or Windows/Cygwin), fire up a small linux EC2 instance and SSH into it as a user with super user privileges.

Install git, ssh-agent, aws cli, configure aws cli

In your linux box, ensure that git, ssh-agent, and the aws cli are all installed and configured appropriately. Your instructor will now show you how to set up an EC2 for this purpose.

Git clone the code locally

At the bottom of the window, you’ll see a terminal. In the terminal, enter:

```
git clone https://github.com/rich-morrow/autom8d-deepdive.git
#then cd into the codebase, activity3
cd autom8d-deepdive/activity-3
```

Your instructor will point out several directories and their contents.

Deploy solution

In the [deployment guide](#) for this quickstart, follow the instructions along with your instructor to deploy the solution.

Walk through deployed assets

As assets are deploying, your instructor will guide you through the following Resources that deployed, and point out details of their configuration:

[IAM](#), [S3](#), [CodePipeline](#), [CodeCommit](#), [CodeBuild](#), [Lambda](#), [Secrets Manager](#), [KMS](#)

Perform Sample Deployment

Perform the last step of [deployment guide](#) (run the git-setup.sh code) to actually perform a deployment, and your instructor will guide you through the various activities that take place.

Rein in permissions

The roles that were created in this lab were a tad permissive for the real world, let’s look at ways we can use Resources in the the PDs to implement rule of least privilege.

Run CloudFormation Delete

As a final cleanup step, let's run CF delete in our respective accounts to clean up our work.

******* END OF ACTIVITY #3 STOP HERE!!! *******

You can now create, modify, use, and delete a full, production ready CI/CD toolchain using Lambda invocation triggers.

BONUS Activity #4: Real world Lambda function in Cloud9 IDE

Log into your AWS console and browse to the [Cloud9 area](#) (we use us-east-1, but you can switch Regions if you like):

NOTE: As of 2 May, 2022, these are all the default settings.

Environment Name: lambda-automation-basics

Environment type: "Create a new EC2 instance for environment (direct access)

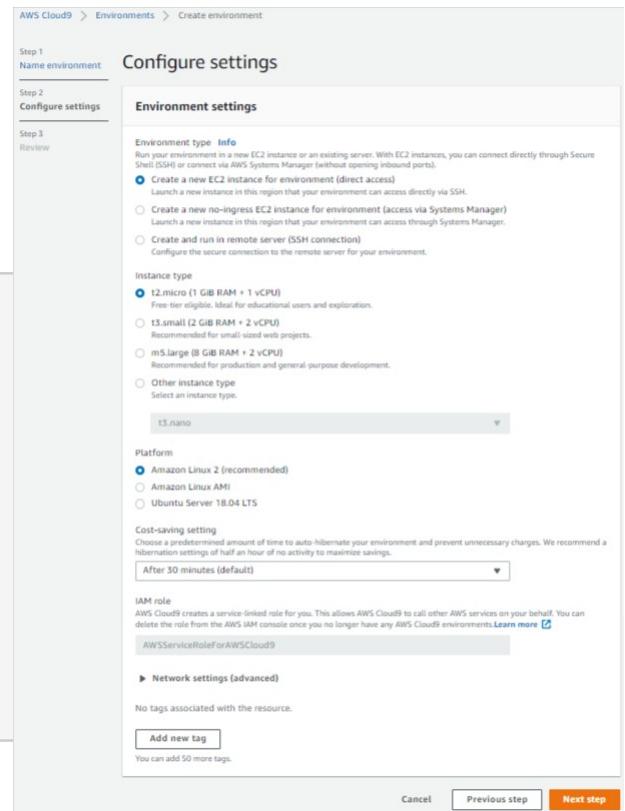
Instance type: "t2.micro"

Platform: "Amazon Linux 2"

Cost-saving setting: "After 30 minutes"

IAM Role: "AWSLambdaServiceRoleForAWSCloud9"

Click "Next Steps" until you get to "Review" page, then click "Create Environment". You will see messages like "We are creating your AWS Cloud9 environment. This can take a few minutes", then "Connecting". When your environment is complete, you'll see a page that says "Welcome to your development environment".



Git clone the code locally

At the bottom of the window, you'll see a terminal. In the terminal, enter:

```
git clone https://github.com/rich-morrow/autom8d-foundations.git
```

```
#then cd into the codebase, activity1
```

```
cd autom8d-foundations/activity1
```

```
#finally, cat the code out:
```

```
cat check-set-tags.py
```

Browse the code

Grab the terminal divider (double lines just above the terminal) and drag it to a larger view so it looks something like this:

A screenshot of the Cloud9 IDE interface. The top navigation bar includes File, Edit, Find, View, Go, Run, Tools, Window, Support, Preview, and a green "Run" button. The left sidebar shows a file tree with a folder named "lambda-automatic" containing "autom8d-foundations" and "README.md". The main workspace contains a terminal window titled "Untitled1" with the following code:

```
rich:~/environment $ git clone https://github.com/rich-morrow/autom8d-foundations.git
rich:~/environment $ git clone https://github.com/rich-morrow/autom8d-foundations.git
Cloning into 'autom8d-foundations'...
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 16 (delta 1), reused 7 (delta 0), pack-reused 0
Receiving objects: 100% (16/16), 4.19 MiB | 26.65 MiB/s, done.
Resolving deltas: 100% (1/1), done.
rich:~/environment $ cd autom8d-foundations/activity-1
rich:~/environment/autom8d-foundations/activity-1 (main) $ cat check-set-tags.py
import boto3
import json
from pprint import pprint

# Set to True to get the lambda to assume the Role attached on the Config Service (useful for cross-account).
ASSUME_ROLE_MODE = False

# This gets the client after assuming the Config service role
# either in the same AWS account or cross-account.
def get_client(service, event):
    """Return the service boto client. It should be used instead of directly calling the client.
    Keyword arguments:
    service -- the service name used for calling the boto.client()
    event -- the event variable given in the lambda handler
    """
    if not ASSUME_ROLE_MODE:
        return boto3.client(service)
    credentials = get_assume_role_credentials(event["executionRoleArn"])
    return boto3.client(service, aws_access_key_id=credentials['AccessKeyId'],
                        aws_secret_access_key=credentials['SecretAccessKey'],
                        aws_session_token=credentials['SessionToken'])

# Helper function used to validate input
def check_defined(reference, reference_name):
    if not reference:
        raise Exception('Error: ', reference_name, 'is not defined')
    return reference

# Check whether the message is OversizedConfigurationItemChangeNotification or not
def check_size(item):
    if item['SizeInBytes'] > 1024 * 1024 * 1024:
        return "OversizedConfigurationItemChangeNotification"
    else:
        return "StandardConfigurationItemChangeNotification"

# This gets the client after assuming the Config service role
# either in the same AWS account or cross-account.
def get_client(service, event):
    """Return the service boto client. It should be used instead of directly calling the client.
    Keyword arguments:
    service -- the service name used for calling the boto.client()
    event -- the event variable given in the lambda handler
    """
    if not ASSUME_ROLE_MODE:
        return boto3.client(service)
    credentials = get_assume_role_credentials(event["executionRoleArn"])
    return boto3.client(service, aws_access_key_id=credentials['AccessKeyId'],
                        aws_secret_access_key=credentials['SecretAccessKey'],
                        aws_session_token=credentials['SessionToken'])

# Helper function used to validate input
def check_defined(reference, reference_name):
    if not reference:
        raise Exception('Error: ', reference_name, 'is not defined')
    return reference
```

This is basically the code we'll use for activity #2. For now, don't worry too much about what it does... let's open it in Cloud9 by choosing “File→Open”, then selecting “activity-1/check-set-tags.py”. Drag your terminal view down smaller so you can better see the code. As shown below:

A screenshot of the Cloud9 IDE interface. The top navigation bar includes File, Edit, Find, View, Go, Run, Tools, Window, Support, Preview, and a green "Run" button. The left sidebar shows a file tree with a folder named "lambda-automatic" containing "autom8d-foundations" and "README.md". The main workspace contains a terminal window titled "Untitled1" with the same code as above, and a separate terminal window titled "bash - ip-172-31-0-91 ec2x" with the following output:

```
rich:~/environment $ git clone https://github.com/rich-morrow/autom8d-foundations.git
rich:~/environment $ git clone https://github.com/rich-morrow/autom8d-foundations.git
Cloning into 'autom8d-foundations'...
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 16 (delta 1), reused 7 (delta 0), pack-reused 0
Receiving objects: 100% (16/16), 4.19 MiB | 26.65 MiB/s, done.
Resolving deltas: 100% (1/1), done.
rich:~/environment $ cd autom8d-foundations/activity-1
```

You can attempt to run the code directly by pushing the green “play” button named “Run” at page top, but it will fail as boto3 has not been loaded into our Cloud9 environment. Let's fix that by doing a pip install in the terminal at page bottom:

```
pip install boto3
```

Your install should look something like this:

```
python3 -ip-172-31-0-91.x | Immediate | autom8d-foundations/act | +  
ComplianceEventItem : invoking_event configurationItem [ 'resourceId' ,  
'ComplianceType' : compliance_value,  
'OrderingTimestamp' : invoking_event['configurationItem']['configurationItemCaptureTime']  
,  
,  
],  
rich:/environment/autom8d-foundations/activity-1 (main) $ pip install boto3  
Defaulting to user installation because normal site-packages is not writable  
Collecting boto3  
  Downloading boto3-1.22.4-py3-none-any.whl (132 kB)  
    ██████████ | 132 kB 6.0 MB/s  
Collecting s3transfer<0.6.0,>=0.5.0  
  Downloading s3transfer-0.5.2-py3-none-any.whl (79 kB)  
    ██████████ | 79 kB 8.9 MB/s  
Collecting botocore<1.26.0,>=1.25.4  
  Downloading botocore-1.25.4-py3-none-any.whl (8.7 MB)  
    ██████████ | 8.7 MB 42 kB/s  
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/local/lib/python3.7/site-packages (from boto3) (1.0.0)  
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/local/lib/python3.7/site-packages (from botocore<1.26.0,>=1.25.4->boto3) (1.26.9)  
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/local/lib/python3.7/site-packages (from botocore<1.26.0,>=1.25.4->boto3) (2.8.2)  
Requirement already satisfied: six<=1.5 in /usr/local/lib/python3.7/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.26.0,>=1.25.4->boto3) (1.16.0)  
Installing collected packages: botocore, s3transfer, boto3  
Successfully installed boto3-1.22.4 botocore-1.25.4 s3transfer-0.5.2  
rich:/environment/autom8d-foundations/activity-1 (main) $
```

Expand autom8d-foundations→activity-1 in the left hand navigation. Double click the “event.json” object to open it in a new tab. This event object is a mock of what would be passed to the Lambda function in a real, live execution. We can alter it later on if we choose

The code editor shows the event.json file with the following content:

```
1 {  
2   "invokingEvent": "{\"configurationItem\":{\"configurationItemCaptureTime\":\"2016-02-17T01:36:34.043Z\"},  
3   \"ruleParameters\": \"\\\"desiredInstanceId\\\" : \\\"t2.micro\\\"",  
4   \"resultToken\": \"myResultToken\",  
5   \"eventLeftScope\": false,  
6   \"executionRoleArn\": \"arn:aws:iam::143230255374:role/config-role\",  
7   \"configRuleArn\": \"arn:aws:config:us-east-1:143230255374:config-rule/config-rule-0123456\",  
8   \"configRuleName\": \"change-triggered-config-rule\",  
9   \"configRuleId\": \"config-rule-0123456\",  
10  \"accountId\": \"143230255374\",  
11  \"version\": \"1.0\"  
12 }
```

The terminal window shows the pip install command for botocore.

```
python3 -ip-172-31-0-91.x | autom8d-foundations/act | +  
Downloading botocore-1.25.4-py3-none-any.whl (8.7 MB)  
    ██████████ | 8.7 MB 42 kB/s  
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/local/lib/python3.7/site-packages (from boto3) (1.0.0)
```

select the “check-set-tags.py” tab to look at our code again. Notice how there are several “def” functions defined. Your instructor will walk through each, describing at a high level what’s going on.

Note, also that since we’re running this code in the IDE, we have to:

1. Manually load our events.json code (lines 208,209)
2. Directly invoke our “lambda_handler” method (line 213). When this function runs natively in Lambda (as you’ll see in Activity #2), you’ll see that “lambda_handler” is what is called automatically for us by the Lambda service.

Also, we should point out that if you’re doing a lot of local development and/or dealing with event objects frequently, there is a more elegant way to do this – with the SAM CLI. Although what we’ve done here will work just fine for our demo purposes (and it also nicely highlights some python operations like file open, read, etc), if you want to learn the “right way” to deal with Lambda in Cloud9, check out this AWS user guide: [Working with AWS serverless applications using the AWS Toolkit](#).

Now that boto is installed, we should be all ready to run our code. With our focus on the “check-set-tags.py” tab, let’s hit “Run” again at the page top!

```

File "/home/ec2-user/environment/autom8d-foundations/activity-1/check-set-tags.py", line 192, in lambda_handler
    configuration_item, rule_parameters)
File "/home/ec2-user/environment/autom8d-foundations/activity-1/check-set-tags.py", line 143, in evaluate_change_notification_compliance
    if (configuration_item['configuration']['instanceType'] in rule_parameters['desiredInstanceType']):
KeyError: 'instanceType'

Process exited with code: 0

```

Uh oh. We see errors again: “`KeyError: ‘instanceType’`”. This time, however, the error is expected (our “Configuration” object doesn’t have an expected “`instanceType`” for the mock). We can correct this, but then we’ll see other errors for “tags”, etc. For our purposes, this is just fine. We’ve validated that our code works... it’s just a bad “events” object being passed in. In Activity 2, we’ll see proper operation of this code when we come back and watch it run in a live scenario with real “Configuration” data passed into it from AWS Config.

***** END OF ACTIVITY #4 STOP HERE!!! *****

You now know basic operation (env create, terminal usage, code loading/running/debugging) of Cloud9, and you’ve gotten some experience with a fairly advanced Lambda function.

BONUS Activity #5: Automating tag enforcement with AWS Lambda and AWS Config

In this lab, we’re going to set up our ‘check-set-tags’ Lambda function to automatically be invoked by AWS Config when it detects any configuration change on our EC2 instances. The pseudocode for this function looks like:

```

foreach ec2 instance {
    if(instance is approved instanceType) {
        if(instance is tagged with "env=prod") {
            if(instance is not tagged with "owner") {
                tag instance with "owner=rich@quicloud.com"
            }
            mark COMPLIANT and EXIT
        } else {
            stop instance
    }
}

```

```
mark NON_COMPLIANT
```

```
}
```

COMPLIANT OR NON_COMPLIANT will appear in the AWS Config Dashboard, as well

Start up two (or more) EC2 instances

Before we start our function running, we'll need some test instances for the script to check. In the AWS Console, browse to EC2, and start up two instances. One of size t2.micro, and one of size t3.nano. You may still have a t2.micro running from activity 1 where we played with Cloud9. If desired, you may also start instances of other sizes as well (just be careful... larger instance types can incur significant costs in AWS). Once your instances are started, you should see something like this below (we still have our Cloud9 instance running).

Name	Instance ID	Instance state	Instance type	Status check	Alarm status
-	i-08ee56ddc5eea7ea0	Running	t3.nano	2/2 checks passed	No alarms
-	i-0658e41211fab1d20	Running	t2.micro	2/2 checks passed	No alarms
-	i-09129e988694da220	Stopped	m5.large	-	No alarms
aws-cloud9-la...	i-0d7e56f1b7254f179	Running	t2.micro	2/2 checks passed	No alarms

Create appropriate Lambda role

The Lambda function we'll use needs proper **execution permissions** to perform all of it's interactions with other AWS Services. We'll use our pre-created policy document to create a role. From the git package you downloaded earlier (we just used the Cloud9 bash shell), browse into the 'activity-2' directory and run a directory listing.

```
cd autom8d-foundations/activity-2/
```

```
dir
```

```
rich:~/environment $ dir
autom8d-foundations README.md
rich:~/environment $ cd autom8d-foundations/activity-2/
rich:~/environment/autom8d-foundations/activity-2 (main) $ dir
check-set-tags.py lambda-automation-activity-2-role-policy.txt
rich:~/environment/autom8d-foundations/activity-2 (main) $
```

cat out the contents of the 'lambda-automation-activity-2-rolepolicy.txt'

```
cat lambda-automation-activity-2-role-policy.txt
```

Copy the entire statement (from opening "{" to closing "}") into your copy buffer with CTRL-C. Your instructor will take a moment to explain the various ramifications of the statements inside of the document.

```
rich:~/environment/autom8d-foundations/activity-2 (main) $ cat lambda-automation-activity-2-role-policy.txt
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "config:BatchGet*",
        "config:Describe*",
        "config:Get*",
        "config:List*",
        "config:Put*",
        "config:Select*",
        "ec2:)",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "tag:GetResources"
      ],
      "Resource": "*"
    }
  ]
rich:~/environment/autom8d-foundations/activity-2 (main) $
```

Back in your browser, open another tab to the [IAM service](#). Browse to "Policies" on the left hand navigation.

You should see a view that looks similar to:

The screenshot shows the AWS IAM Policies list interface. On the left, there's a navigation sidebar with options like 'Identity and Access Management (IAM)', 'Dashboard', 'Access management' (with 'User groups', 'Users', 'Roles', and 'Policies' selected), 'Access reports' (with 'Access analyzer', 'Archive rules', 'Analyzers', 'Settings', 'Credential report', 'Organization activity', and 'Service control policies (SCPs)'), and 'Identity providers' and 'Account settings'. The main area has a header 'Policies (954) Info' with a note: 'A policy is an object in AWS that defines permissions.' It includes a search bar 'Filter policies by property or policy name and press enter', a page number '1' (of 48), and a 'Create Policy' button. Below is a table with columns 'Policy name', 'Type', and 'Use'. The table lists several policies, such as 'AmazonS3ReadOnlyAccess-201502181240' (Customer managed, Non-compliant), 'AWSControlTowerAdminPolicy' (Customer managed, Permitted), 'AWSControlTowerCloudTrailRolePolicy' (Customer managed, Permitted), 'AWSControlTowerStackSetRolePolicy' (Customer managed, Permitted), 'AWSLambdaBasicExecutionRole-24e5ac4e-71ca-4434-87ec-4140faf86a30' (Customer managed, Permitted), 'AWSLambdaBasicExecutionRole-fe680df1-9077-4fdcc-92ec-c7927b820b46' (Customer managed, Permitted), 'lambda-s3-thumbnailer' (Customer managed, Permitted), and 's3-get-object-for-developers' (Customer managed, Permitted).

Click the “Create Policy” button on the upper right, choose the “JSON” tab, and paste in the policy document you copied above, overwriting the stub that was pre-entered.

The screenshot shows the 'Create policy' wizard. At the top, it says 'Create policy' with three numbered steps: 1 (selected), 2, and 3. Below is a note: 'A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. [Learn more](#)'.

Below the note, there are tabs: 'Visual editor' (selected) and 'JSON'. To the right is a 'Import managed policy' button. The main area contains a code editor with the JSON policy document pasted in. The code is as follows:

```
1  {
2      "Version": "2012-10-17",
3      "Statement": [
4          {
5              "Effect": "Allow",
6              "Action": [
7                  "config:BatchGet*",
8                  "config:Describe*",
9                  "config:Get*",
10                 "config>List*",
11                 "config:Put*",
12                 "config:Select*",
13                 "ec2:*",
14                 "logs>CreateLogGroup",
15                 "logs>CreateLogStream",
16                 "logs:PutLogEvents",
17                 "tag:GetResources"
18             ],
19             "Resource": "*"
20         }
21     ]
22 }
```

At the bottom of the code editor, there are status indicators: 'Security: 0', 'Errors: 0', 'Warnings: 0', 'Suggestions: 0'.

Next: 273 of 6,144.

Cancel

Next: Tags

Click “Next: Tags”, then “Next: Review”. Give the policy a name of “activity-2-lambda-automation-basics-policy”. You can give a description if you like. Then push “Create policy” button

Create policy

1 2 3

Review policy

Name*

activity-2-lambda-automation-basics-policy

Use alphanumeric and '+=_@-' characters. Maximum 128 characters.

Description

Maximum 1000 characters. Use alphanumeric and '+=_@-' characters.

Summary

Allow (4 of 325 services) Show remaining 321			
Service	Access level	Resource	Request context
CloudWatch Logs	Limited: Write	All resources	None
Config	Full: List Limited: Read, Write	All resources	None
EC2	Full access	All resources	None
Resource Group Tagging	Limited: Read	All resources	None

Tags

Key	Value
No tags associated with the resource.	

Cancel

Previous

Create policy

Now we'll attach this policy to a role that Lambda will assume when it needs to.

From the left hand nav in IAM, select "Roles". And click "Create role" button in upper right.

Identity and Access Management (IAM) > Roles > Create role

Introducing the new IAM roles experience

Step 1 Select trusted entity

Step 2 Add permissions

Step 3 Name, review, and create

Access management

- User groups
- Users
- Roles**
- Policies
- Identity providers
- Account settings

Access reports

Select trusted entity

Trusted entity type

- AWS service Allow AWS services like EC2, Lambda, or others to perform actions in this account.
- AWS account Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- Web identity Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

Use case

Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Common use cases

- EC2 Allows EC2 instances to call AWS services on your behalf.
- Lambda Allows Lambda functions to call AWS services on your behalf.

Use cases for other AWS services:

Choose a service to view use case

Cancel Next

For "Trusted entity type", choose "AWS service", and for "Use case" choose "Lambda", then push "Next".

On the “Step 2: Add permissions” page, select the “activity-2-lambda-automation-basics-policy” that you entered before (it should appear at the top of the list). If you do not see this policy, you must repeat the steps above to enter the policy correctly. Push “Next” button.

On the “Step 3: Name, review, and create” page, name your role “activity-2-lambda-automation-basics-role”, give it an optional description, and then press “Create role” button to finalize the creation.

The screenshot shows the AWS IAM Role creation process. It consists of three main sections:

- Step 1: Select trusted entity**: Shows the "activity-2-lambda-automation-basics-policy" selected.
- Step 2: Add permissions**: Shows the "activity-2-lambda-automation-basics-policy" selected in the list of available policies.
- Step 3: Name, review, and create**: Shows the role named "activity-2-lambda-automation-basics-role" and a description "Allows Lambda functions to call AWS services on your behalf".

Below these steps, the "Permissions policy summary" section shows the attached policy: "activity-2-lambda-automation-basics-policy" (Customer managed). A blue banner at the bottom of this section says: "Introducing the new IAM roles experience. We've redesigned the IAM roles experience to make it easier to use. [Let us know what you think.](#)"

The final step shows a green success message: "Role activity-2-lambda-automation-basics-role created" with a "View role" button.

You should finally see a screen that looks similar to the following, showing that your role has been successfully created.

This screenshot shows the AWS IAM Roles list. It displays a single role entry:

Role name
activity-2-lambda-automation-basics-role

At the top right, there are "Create role", "Delete", and "Info" buttons. Below the table, there is a search bar and a navigation bar with pages 1 and 2.

Create our Lambda function

Now that we have the appropriate permissions (the Lambda role) for our function to work, we can go ahead with the creation of the function itself. From wherever you installed the git package for this class, cat out the "check-set-tags.py" python code.

```
cat check-set-tags.py
```

Copy the entirety of this code (from "import boto3" at top to "resultToken']" at bottom) by selecting it and pushing CTRL-C to copy it into your clipboard.

Open a new tab in your browser, and browse to the [Lambda section of the console](#). Press "Create function" button in the upper right.

Choose "Author from scratch", give your function the name "check-set-tags", Choose the runtime of "Python 3.8", and under "Permissions", expand "Change the default execution role", then select your "activity-2-lambda-automation-basics-role" from the list. Then push the "Create function" button.

The screenshot shows the AWS Lambda 'Create function' wizard. It consists of five main steps:

- Step 1: Choose function type**
 - Author from scratch** (selected): Start with a simple Hello World example.
 - Use a blueprint**: Build a Lambda application from sample code and configuration presets for common use cases.
 - Container image**: Select a container image to deploy for your function.
 - Browse serverless app repository**: Deploy a sample Lambda application from the AWS Serverless Application Repository.
- Step 2: Basic information**
 - Function name**: Enter a name that describes the purpose of your function. (Input: check-set-tags)
 - Runtime**: Choose the language to use for your function. Note that the console code editor supports only Node.js, Python, and Ruby. (Selected: Python 3.8)
 - Architecture**: Choose the instruction set architecture you want for your function code. (Selected: x86_64)
- Step 3: Permissions**
 - Execution role**: Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console. (Selected: Use an existing role: activity-2-lambda-automation-basics-role)
 - Existing role**: Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs. (Selected: activity-2-lambda-automation-basics-role)
- Step 4: Advanced settings**
- Step 5: Review and Create**
 - Create function** button (orange)

On the next page, under “lambda_function” under “Code source”, paste your “check-set-tags” code in the area, overwriting the stub that was pre-entered. NOTE: Your changes ARE NOT LIVE until you push the “Deploy” button. Whenever you make changes to your code, make sure to always push the “Deploy” button to push it live. Go ahead and push the “Deploy” button as well.

Select the dropdown to the right of the “Test” button, and choose “Configure test event”. Copy the contents of our “events.json” object from activity-1 into the “Event JSON”, and give the event the name “config_event” and press “Save”.

Test event action

Create new event Edit saved event

Event name

config_event

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

Private This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn more [\[?\]](#)

Shareable This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more [\[?\]](#)

Template - optional

hello-world

Event JSON

```

1 - [{}]
2   "invokingEvent": "{\"configurationItem\":{\"configurationItemCaptureTime\":\"2016-02-17T01:30:00Z\",\"configurationItemVersion":1,"configurationItemStatus":"OK","configurationItemType":"AWS::EC2::EIP","configurationItemResourceType":"AWS::EC2::EIP","configurationItemResourceARN":"arn:aws:ec2:us-east-2:123456789012:instance/i-00000000","configurationItemRelationships":[]},\"ruleParameters\":{\"desiredInstanceType\":\"t2.micro\"},\"resultToken\":\"myResultToken\", \"eventLeftScope\": false, \"executionRoleArn\": \"arn:aws:iam::143230255374:role/config-role\", \"configurationRuleName\": \"change-triggered-config-rule\", \"configurationRuleId\": \"config-rule-0123456\", \"accountId\": \"143230255374\", \"version\": \"1.0\"}
12 []

```

Format JSON

Cancel Save

Now that we have a valid test object to test with, go ahead and press the “Test” button under Code source. You should see output that looks similar to the following. Notice that we see the same “instanceType” failure that we had from activity 1. This is to be expected, and at least validates that our function is set up correctly, the code is working properly, and that our function has the appropriate permissions to do what it needs to do.

Code source Info

AWS Cloud9 File Edit Find View Go Tools Window Test Deploy

Execution results Execution result

Status: Failed | Max memory used: 77 MB | Time: 1962.48 ms

Test Event Name config_event

Response

```
{
  "errorMessage": "'instanceType'",
  "errorType": "KeyError",
  "stackTrace": [
    "File \"/var/task/lambda_function.py\", line 192, in lambda_handler\n      compliance_value = evaluate_change_notification_compliance(\n        compliance_value + rule['compliance_value'] if (configuration_item['configuration_type'] in rule['parameters']['desired_instance_type']) else\n    )\n"
  ]
}
```

Function Log

```

START RequestID: #5C1B#42-03e0-4ee8-8968-3bd06efdee9b Version: $LATEST
CONFIGURATION_ITEM_DUMP
('ARN': 'arn:aws:ec2:us-east-2:123456789012:instance/i-00000000',
 'availabilityZone': 'us-east-2a',
 'autoAssignPublicIp': True,
 'awsRegion': 'us-east-2',
 'configuration': {'foo': 'bar'},
 'configurationItemArn': 'arn:aws:config:us-east-2:2016-02-17T01:36:34.043Z',
 'configurationItemStatus': 'OK',
 'relationships': [{name: 'Is attached to ElasticIp',
   resourceType: 'AWS::EC2::EIP',
   resourceId: 'i-00000000',
   resourceType2: 'AWS::EC2::Instance',
   tags: {Foo: 'Bar'}}],
   [ERROR] KeyError: 'instanceType'
Traceback (most recent call last):

```

Create custom AWS Config rule using our Lambda function

We'll now set up AWS Config to monitor our EC2 instances, reporting in-or-out of compliance for the instances in the Config Dashboard. First, open a new tab in your browser and browse to [AWS Config](#) in that tab. Click "1-click setup" button, then push "Confirm" (accepting all defaults) to enable AWS Config.

AWS Config > Set up AWS Config

Step 1
Settings

Step 2
Rules

Step 3
Review

Review

Review your AWS Config setup details. You can go back to edit changes for each section. Choose **Confirm** to finish setting up AWS Config.

General settings

Resource types to record
All resources (excluding global resources)

AWS Config role
AWSConfigRoleForConfig

Delivery method

S3 bucket name
config-bucket-143230255374

AWS Config rules (0)

Cancel Previous Confirm

Now that config is enabled, you should see a dashboard that looks similar to the following.

AWS Config

Dashboard

Conformance packs

Rules

Resources

Aggregators

Conformance packs

Rules

Resources

Authorizations

Advanced queries

Settings

What's new

Documentation

AWS Config > Dashboard

Dashboard

Compliance status

Rules Resources

0 Noncompliant rule(s) 0 Noncompliant resource(s)

0 Compliant rule(s) 0 Compliant resource(s)

Noncompliant rules by noncompliant resource count

Name	Compliance

AWS Config usage metrics

AWS Config usage metrics by resource type

Choose Resource types

All

1h 3h 12h 1d 3d 1w Custom

No data available.

Configuration Items Recorded

1

0.5 Try adjusting the dashboard time range.

1

0.5 Try adjusting the dashboard time range.

From the left-hand navigation, select "Rules", then push the "Add rule" button. On the next screen, choose "Create custom Lambda rule" and press "Next"

AWS Config > Rules > Add rule

Step 1
Specify rule type

Step 2
Configure rule

Step 3
Review and create

Specify rule type

Add rules to define the desired configuration setting of your AWS resources. Customize any of the following rules to suit your needs, or create a custom rule. To create a custom rule, you must create an AWS Lambda function for the rule.

Select rule type

Add AWS managed rule
Customize any of the following rules to suit your needs.

Create custom Lambda rule
Create custom rules and add them to AWS Config. Associate each custom rule with an AWS Lambda function, which contains the logic that evaluates whether your AWS resources comply with the rule.

Create custom rule using Guard
Create custom rules using Guard Custom Policy that evaluates whether your AWS resources comply with the rule.

check-set-tags

Function overview Info

Throttle Copy ARN Actions

check-set-tags

Layers (0)

+ Add trigger

+ Add destination

Description

Last modified 19 minutes ago

Function ARN arn:aws:lambda:us-east-1:143230255374:function:check-set-tags

Function URL Info

Back on your Lambda browser tab, copy the ARN of your "check-set-tags"

lambda function by pressing the “copy” icon as shown in this screenshot:

Back in your AWS Config tab, give your rule the name “check-set-tags”, give it a description, paste your ARN under “AWS Lambda function ARN”, select “Trigger type” of “When configuration changes”, and then under parameters, set a “Key” of “desiredInstanceType” and a value of “t2.micro,t3.nano”. These Config parameters are hooked into our Lambda function to control the instance types we allow in our account, and you are welcome to change them later.

Press “Next” button, then on the Confirmation page, press “Add rule” button to finally add the rule.

The screenshot shows the 'Configure rule' wizard with the following details:

- Details:** Name: check-set-tags, Description: check EC2 instances for compliance with instance type and proper tagging, AWS Lambda function ARN: arn:aws:lambda:us-east-1:143230255374:function:check
- Trigger:** Trigger type: When configuration changes (checked), Scope of changes: All changes (checked), Resources and Tags options are unselected.
- Parameters:** Key: desiredInstanceType, Value: t2.micro,t3.nano, Add another row button.
- Rule tags - optional:** No tags defined.

You should now see the following:

The screenshot shows the 'Rules' page with the following details:

- A green banner at the top says: "The rule: check-set-tags has been added to your account".
- Navigation: AWS Config > Rules.
- Section: Rules. Sub-section: Rules represent your desired configuration settings. AWS Config evaluates whether your resource configurations comply with relevant rules and summarizes the compliance results.
- Table: Rules

Name	Remediation action	Type	Compliance
check-set-tags	Not set	Custom Lambda	-

And if you click back into Dashboard from the left hand nav, you should now start to see Compliance / Non-Compliance for your assets as so:

The screenshot shows the 'Dashboard' with the following details:

- Left sidebar: AWS Config > Dashboard.
- Section: Compliance status
 - Rules: 0 Noncompliant rule(s) (red), 0 Compliant rule(s) (green).
 - Resources: 2 Noncompliant resource(s) (red), 2 Compliant resource(s) (green).
- Section: Noncompliant rules by noncompliant resource count
 - Table: Name | Compliance
 - Row: check-set-tags | 2 Noncompliant resource(s)
- Link: View all noncompliant rules.

Test our Config Rule by tweaking EC2 tags

In your browser, open a new tab for CloudWatch Logs. We'll revist this tab frequently as it's where our "print" statements from our Lambda function will land.

Iteratively re-visit your EC2 tab in your browser, and try the following:

1. Set "env" tag to "prod" (then watch Config/Lambda tag your instances with "owner=rich@quicloud.com")
2. Set "env" tag to anything besides prod (then watch Config / Lambda stop your instances)
3. Start new EC2s with proper "env=prod" "owner=your@email.com" and watch them go compliant in the Config Dashboard.

Questions:

4. What are some limitations of Config (can it check all resources, is it "real-time")?
5. If we needed real-time, or other resource checking, what might be a better service?
6. How could we improve our Lambda function in a real-world environment?
7. How could we improve our permissioning of our Lambda function?

***** END OF ACTIVITY #5 STOP HERE!!! *****

You now have a good understanding of everything involved with using AWS Config with custom Lambda functions to programmaticaly and proactively enforce tagging policies.

Activity #6: Automating EC2 log processing and alerting

In this lab, we're going to use a CloudWatch Logs agent to ship the /var/log/secure SSH logs into CloudWatch Logs. We'll then route specific log entries to a Lambda to trigger with CloudWatch Subscription Filters. The Lambda function will log when unauthorized SSH attempts happen on our server.

NOTE: Previous versions of our slides for this class referenced an "EventBridge" solution. This activity has been changed to simply use CloudWatch Subscription Filters as they are a much more elegant solution for this particular use case.

Create CloudWatch Logs Role & launch EC2 Instance

1. Follow the documented [AWS instructions](#) (Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances) to create the role necessary for our EC2 instance to write its logs to CloudWatch Logs. At the end of the process, you will have created a role called "CloudWatchAgentAdminRole"
2. From the AWS EC2 console, launch a t2.micro instance with Amazon Linux (or, optionally re-use one of the ones from activity #2, if you still have it). Give the instance the name "activity-3". Under "key pair name", choose "Create a new key pair", and give the keypair the name "activity-3". If you use linux to SSH, leave the default ".pem", or if you use Putty to SSH, select the ".ppk" option. Your keypair will download locally – note the location it saved to as you'll need it later. Expand "Advanced details", and under "IAM instance profile", select the "CloudWatchAgentAdminRole" you created above. Accept all other defaults (including "Allow SSH traffic from Anywhere" – generally a BAD idea!). Note this instance ID.

- SSH into your “activity-3” instance (user is “ec2-user”), and then install the cloudwatch agent (it’s a standard package)

```
sudo yum install amazon-cloudwatch-agent
```

Once the command runs, you should see the output as shown on the right.

```
ec2-user@ip-172-31-0-66 ~
https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-0-66 ~]$ sudo yum install amazon-cloudwatch-agent
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core
amzn2extra-docker
amzn2extra-kernel=5.10
(1/7): amzn2-core/2/x86_64/group_gz | 3.7 kB 00:00
(2/7): amzn2-core/2/x86_64/updateinfo | 3.0 kB 00:00
(3/7): amzn2extra-docker/2/x86_64/updateinfo | 2.5 kB 00:00
(4/7): amzn2extra-kernel=5.10/2/x86_64/group_gz | 470 kB 00:00
(5/7): amzn2extra-kernel=5.10/2/x86_64/updateinfo | 6.2 kB 00:00
(6/7): amzn2extra-kernel=5.10/2/x86_64/primary_db | 30 kB 00:00
(7/7): amzn2-core/2/x86_64/primary_db | 8.5 MB 00:00
Dependencies Resolved
=====
Package           Arch      Version            Repository   Size
=====
Installing:
amazon-cloudwatch-agent x86_64    1.247350.0-1.amzn2 amzn2-core 45 M
Transaction Summary
Install 1 Package
=====
Total download size: 45 M
Installed size: 45 M
1 transaction, 1 file(s) downloaded, 0 verify errors
Downloaded:
amazon-cloudwatch-agent-1.247350.0-1.amzn2.x86_64.rpm | 45 MB 00:01
Running transaction test
Running transaction test
Transaction test succeeded
Running transaction
  Creating user cwapagent, result: 0
  Creating group cwapagent, result: 0
  Installing : amazon-cloudwatch-agent-1.247350.0-1.amzn2.x86_64
  Verifying  : amazon-cloudwatch-agent-1.247350.0-1.amzn2.x86_64
=====
Installed:
amazon-cloudwatch-agent.x86_64 0:1.247350.0-1.amzn2
=====
Complete!
[ec2-user@ip-172-31-0-66 ~]$
```

Configure CloudWatch Logs Agent

The logfile that we’re going to analyze is /var/log/secure on Amazon Linux 2. This log has all the entries whenever an SSH attempt is made against this server. Cat the file out to see our current logged in session.

```
sudo cat /var/log/secure
```

Now, let’s tail the log file to see what an invalid access would look like.

```
sudo tail -f /var/log/secure
```

```
[ec2-user@ip-172-31-0-66 ~]$ sudo cat /var/log/secure
May 3 02:17:17 ip-172-31-0-66 useradd[3072]: user: name=ec2-user, GID=1000
May 3 02:17:17 ip-172-31-0-66 useradd[3072]: user: name=ec2-user, UID=1000, home=/home/ec2-user, shell=/bin/bash
May 3 02:17:17 ip-172-31-0-66 useradd[3072]: add 'ec2-user' to group 'adm'
May 3 02:17:17 ip-172-31-0-66 useradd[3072]: add 'ec2-user' to group 'wheel'
May 3 02:17:17 ip-172-31-0-66 useradd[3072]: add 'ec2-user' to group 'systemd-journal'
May 3 02:17:17 ip-172-31-0-66 useradd[3072]: add 'ec2-user' to shadow group 'adm'
May 3 02:17:17 ip-172-31-0-66 useradd[3072]: add 'ec2-user' to shadow group 'wheel'
May 3 02:17:17 ip-172-31-0-66 sshd[3154]: Server listening on 0.0.0.0 port 22.
May 3 02:17:17 ip-172-31-0-66 sshd[3154]: Server listening on :: port 22.
May 3 02:17:19 ip-172-31-0-66 sshd[3176]: Server listening on 0.0.0.0 port 22.
May 3 02:17:19 ip-172-31-0-66 sshd[3176]: Server listening on :: port 22.
May 3 02:19:41 ip-172-31-0-66 sshd[3351]: error: AuthorizedKeyCommand /opt/aws/bin/eic_runAuthorizedKeys ec2-user SHA256:opCN7TyEovvLQmWw9DyQJQ3KffqBhNkydOmMg8 failed, status 22
May 3 02:19:41 ip-172-31-0-66 sshd[3351]: Accepted publickey for ec2-user from 73.217.17.5 port 61021 ssh2: RSA SHA256:opCN7TyEovvLQmWw9DyQJQ3KffqBhNkydOmMg8
May 3 02:19:41 ip-172-31-0-66 sshd[3351]: Accepted publickey for ec2-user from 73.217.17.5 port 61021 ssh2: RSA SHA256:opCN7TyEovvLQmWw9DyQJQ3KffqBhNkydOmMg8
May 3 02:19:44 ip-172-31-0-66 sshd[3351]: pam_unix(sshd:session): session opened for user ec2-user by (uid=0)
May 3 02:19:44 ip-172-31-0-66 sudo: ec2-user : TTY-pts/0 : FWD-/home/ec2-user ; USER=root ; COMMAND=/bin/yum install amazon-cloudwatch-agent
May 3 02:19:44 ip-172-31-0-66 sudo: pam_unix(sudo:session): session opened for user root by ec2-user(uid=0)
May 3 02:19:45 ip-172-31-0-66 groupadd[3433]: group added to /etc/group; name=cwapagent, GID=993
May 3 02:19:45 ip-172-31-0-66 useradd[3433]: user added to /etc/passwd; name=ec2-user, GID=993
May 3 02:19:45 ip-172-31-0-66 useradd[3433]: new user: name=cwapagent, UID=993, home=/home/cwapagent, shell=/sbin/nologin
May 3 02:19:50 ip-172-31-0-66 pam_unix(sudo:session): session closed for user root
May 3 02:19:51 ip-172-31-0-66 sudo: ec2-user : TTY-pts/0 : FWD-/home/ec2-user ; USER=root ; COMMAND=/bin/cat /var/log/secure
May 3 02:19:51 ip-172-31-0-66 sudo: pam_unix(sudo:session): session opened for user root by ec2-user(uid=0)
May 3 02:19:51 ip-172-31-0-66 sudo: pam_unix(sudo:session): session closed for user root
May 3 02:19:52 ip-172-31-0-66 sudo: ec2-user : TTY-pts/0 : FWD-/home/ec2-user ; USER=root ; COMMAND=/bin/tail -f /var/log/secure
May 3 02:19:52 ip-172-31-0-66 sudo: pam_unix(sudo:session): session opened for user root by ec2-user(uid=0)
May 3 02:52:54 ip-172-31-0-66 sudo: ec2-user : TTY-pts/0 : FWD-/home/ec2-user ; USER=root ; COMMAND=/bin/tail -f /var/log/secure
May 3 02:52:54 ip-172-31-0-66 sudo: pam_unix(sudo:session): session opened for user root by ec2-user(uid=0)
```

Open a 2nd putty session to this server, and attempt to log in with another user than the authorized “ec2-user”. In the example below, we’re trying to login as the user “xyz”. Note the failed “invalid user” line.

```
[ec2-user@ip-172-31-0-66 ~]$ sudo tail -f /var/log/secure
May 3 02:54:08 ip-172-31-0-66 sshd[3604]: Connection closed by 73.217.17.5 port 61772 [preauth]
May 3 02:54:46 ip-172-31-0-66 sudo: pam_unix(sudo:session): session closed for user root
May 3 02:54:49 ip-172-31-0-66 sudo: ec2-user : TTY-pts/0 ; FWD-/home/ec2-user ; USER=root ; COMMAND=/bin/cat /var/log/secure
May 3 02:54:49 ip-172-31-0-66 sudo: pam_unix(sudo:session): session opened for user root by ec2-user(uid=0)
May 3 02:54:49 ip-172-31-0-66 sudo: pam_unix(sudo:session): session closed for user root
May 3 02:56:36 ip-172-31-0-66 sshd[3611]: Invalid user from 64.62.197.152 port 25988
May 3 02:56:36 ip-172-31-0-66 sshd[3611]: input_uauth_request: invalid user [preauth]
May 3 02:56:40 ip-172-31-0-66 sshd[3611]: Connection closed by 64.62.197.152 port 25988 [preauth]
May 3 02:58:08 ip-172-31-0-66 sudo: ec2-user : TTY-pts/0 ; FWD-/home/ec2-user ; USER=root ; COMMAND=/bin/tail -f /var/log/secure
May 3 02:58:08 ip-172-31-0-66 sudo: pam_unix(sudo:session): session opened for user root by ec2-user(uid=0)
```

Let’s now configure the CloudWatch Logs agent to ship these logs into CloudWatch. The CloudWatch Agent installs under /opt/aws/amazon-cloudwatch-agent/, so we’ll update our configuration there. The configuration files we’ll create will be stored under the “/etc” directory in that path, so let’s cd into there first and browse the contents.

```
cd /opt/aws/amazon-cloudwatch-agent/etc
```

```
ls -lat
```

You'll only see two one file in there currently, "common-config.toml", and one directory "amazon-cloudwatch-agent.d". Following AWS recommendations, we'll install a config file here named "amazon-cloudwatch-agent.json". Copy the contents of our git "/activity-3/amazon-cloudwatch-agent.json" file into here (using vi, nano, or whatever editor works for you). NOTE: It may be faster/easier to "git clone" our base code on this server and copy the file directly if you have problems with copy/paste in your SSH client). Your instructor will now pull up the contents of the file and walk you through the details.

Once you've browsed the file contents and you're aware of what they each do, we'll go ahead and start our cloudwatch agent with the following command:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-  
config -m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-  
agent.json
```

To ensure Cloudwatch started correctly, let's first look at our log file...

```
sudo cat /opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log
```

If the agent started successfully, we should see something similar to the output at the right.

```
[root@ip-172-31-0-66 etc]# sudo cat /opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log  
2022/05/03 03:41:37 I! Detected the instance is EC2  
2022/05/03 03:41:37 Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json ...  
/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json does not exist or cannot read. Skipping it.  
2022/05/03 03:41:37 Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/file_amazon-cloudwatch-agent.json ...  
2022/05/03 03:41:37 unable to scan config dir /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d with error: unable to parse json, error: invalid character ''! after array element  
2022/05/03 03:41:37 unable to scan config dir /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d with error: unable to parse json, error: invalid character ''! after array element  
No json config files found, please provide config, exit now  
  
2022/05/03 03:41:37 I! Return exit error: exit code=99  
2022/05/03 03:41:37 I! there is no json configuration when running translator  
2022/05/03 03:46:28 I! Detected the instance is EC2  
2022/05/03 03:46:28 Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json ...  
/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json does not exist or cannot read. Skipping it.  
2022/05/03 03:46:28 Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/file_amazon-cloudwatch-agent.json ...  
Valid Json input schema.  
I! Detecting run as user...  
No csm configuration found.  
No metric configuration found.  
Configuration validation first phase succeeded  
  
2022/05/03 03:46:28 I! Config has been translated into TOML /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.toml  
2022/05/03 03:46:28 Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json ...  
2022/05/03 03:46:28 Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/file_amazon-cloudwatch-agent.json ...  
2022/05/03 03:46:28 I! Detected runAsUser: root  
2022/05/03 03:46:28 I! Changing ownership of [/opt/aws/amazon-cloudwatch-agent/logs /opt/aws/amazon-cloudwatch-agent/etc /opt/aws/amazon-cloudwatch-agent/var] to 0:0  
2022-05-03T03:46:28Z I! Starting AmazonCloudWatchAgent 1.247350.0  
2022-05-03T03:46:28Z I! AWS SDK log level not set  
2022-05-03T03:46:28Z I! Loaded inputs: logfile  
2022-05-03T03:46:28Z I! Loaded aggregators:  
2022-05-03T03:46:28Z I! Loaded processors:  
2022-05-03T03:46:28Z I! Loaded outputs: cloudwatchlogs  
2022-05-03T03:46:28Z I! Tags enabled: host=ip-172-31-0-66.ec2.internal  
2022-05-03T03:46:28Z I! [logagent] Config: Interval:10s, Quiet:false, Hostname:"ip-172-31-0-66.ec2.internal", Flush Interval:1s  
2022-05-03T03:46:28Z I! [logagent] starting  
2022-05-03T03:46:28Z I! [logagent] found plugin cloudwatchlogs is a log backend  
2022-05-03T03:46:28Z I! [logagent] found plugin logfile is a log collection  
2022-05-03T03:46:28Z I! [logagent] piping log from ec2-var-log-secure.log/ec2-var-log-secure.log(/var/log/secure) to cloudwatchlogs with retention = 1  
2022-05-03T03:46:44Z W! [outputs.cloudwatchlogs] Retried 0 time, going to sleep 167.069468ms before retrying.  
[root@ip-172-31-0-66 etc]#
```

The true test of whether or not everything has been set up correctly to date (including the EC2 role), is to actually browse the CloudWatch Logs console. Browse to the [CloudWatch Console](#), select “Log groups” under “Logs” on the right-hand navigation, and then click on our “ec2-var-log-secure.log” group. Once you see the group, click on the stream under “Log streams” (Note: We only have a single stream here, because we only have one agent running on one server. If we had several servers running an agent each, we’d see multiple log files). Click on the single “ec2-var-log-secure.log” stream to view its contents. It should look similar to what we see on the right (showing that we only captured entries which match ‘sshd’).

	Timestamp	Message
▶	2022-05-02T21:46:29.467-06:00	No older events at this moment. Retry
▶	2022-05-02T21:46:29.467-06:00	May 3 02:17:19 ip-172-31-0-66 sshd[3154]: Server listening on 0.0.0.0 port 22.
▶	2022-05-02T21:46:29.467-06:00	May 3 02:17:19 ip-172-31-0-66 sshd[3154]: Server listening on :: port 22.
▶	2022-05-02T21:46:29.467-06:00	May 3 02:17:19 ip-172-31-0-66 sshd[3154]: Received signal 15; terminating.
▶	2022-05-02T21:46:29.467-06:00	May 3 02:17:19 ip-172-31-0-66 sshd[3176]: Server listening on 0.0.0.0 port 22.
▶	2022-05-02T21:46:29.467-06:00	May 3 02:17:19 ip-172-31-0-66 sshd[3176]: Server listening on :: port 22.
▶	2022-05-02T21:46:29.467-06:00	May 3 02:33:41 ip-172-31-0-66 sshd[3351]: error: AuthorizedKeysCommand /opt/a...
▶	2022-05-02T21:46:29.467-06:00	May 3 02:33:42 ip-172-31-0-66 sshd[3351]: error: AuthorizedKeysCommand /opt/a...
▶	2022-05-02T21:46:29.467-06:00	May 3 02:33:42 ip-172-31-0-66 sshd[3351]: Accepted publickey for ec2-user fro...
▶	2022-05-02T21:46:29.467-06:00	May 3 02:33:42 ip-172-31-0-66 sshd[3351]: pam_unix(sshd:session): session ope...
▶	2022-05-02T21:46:29.467-06:00	May 3 02:54:05 ip-172-31-0-66 sshd[3604]: Invalid user xyz from 73.217.17.5 po...
▶	2022-05-02T21:46:29.467-06:00	May 3 02:54:05 ip-172-31-0-66 sshd[3604]: input_userauth_request: invalid use...
▶	2022-05-02T21:46:29.467-06:00	May 3 02:54:05 ip-172-31-0-66 sshd[3604]: Connection closed by 73.217.17.5 po...
▶	2022-05-02T21:46:29.467-06:00	May 3 02:56:36 ip-172-31-0-66 sshd[3611]: Invalid user from 64.62.197.152 por...
▶	2022-05-02T21:46:29.467-06:00	May 3 02:56:36 ip-172-31-0-66 sshd[3611]: input_userauth_request: invalid use...
▶	2022-05-02T21:46:29.467-06:00	May 3 02:56:40 ip-172-31-0-66 sshd[3611]: Connection closed by 64.62.197.152 -
▶	2022-05-02T21:46:29.467-06:00	May 3 03:11:31 ip-172-31-0-66 sshd[3712]: Connection closed by 192.241.212.21-
▶	2022-05-02T21:46:29.467-06:00	May 3 03:19:33 ip-172-31-0-66 sshd[3728]: Did not receive identification stri...
▶	2022-05-02T21:46:29.467-06:00	May 3 03:19:42 ip-172-31-0-66 sshd[3738]: Connection reset by 45.67.34.253 po...
▶	2022-05-02T21:46:29.467-06:00	May 3 03:19:43 ip-172-31-0-66 sshd[3729]: Connection reset by 45.67.34.253 po...
▶	2022-05-02T21:46:29.467-06:00	May 3 03:19:46 ip-172-31-0-66 sshd[3731]: Connection reset by 45.67.34.253 po...

No newer events at this moment. [Auto retry paused. Resume](#)

If you like, you can attempt another failed SSH connection to see another “Invalid user [user-name]” message appear in the logs.

Create Lambda function to process Invalid SSH attempts.

Using the steps you went through in activity #2, go ahead & create a 2nd lambda function, using the screenshot at right as a guide. Once the function is created, go ahead & paste the code for “process-cloudwatch-logs.py” into the function & deploy it by pressing the “deploy” button. We could create an event object to test the code with, but we’re just going to go ahead & create a CloudWatch subscription filter which is actually probably faster than even testing.

Create function Info

Choose one of the following options to create your function.

Author from scratch
Start with a simple Hello World example.

Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

Container image
Select a container image to deploy for your function.

Browse serverless app repository
Deploy a sample Lambda application from the AWS Serverless Application Repository.

Basic information

Function name: Enter a name that describes the purpose of your function.
process-cloudwatch-logs
Use only letters, numbers, underscores, or hyphens with no spaces.

Runtime: Info Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
 Python 3.9

Architecture: Info Choose the instruction set architecture you want for your function code.
 x86_64
 arm64

Permissions: Info By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

Change default execution role

Execution role: Choose a role that defines the permissions of your functions. To create a custom role, go to the IAM console.
 Create a new role with basic Lambda permissions.
 Use an existing role
 Create a new role from AWS policy templates

Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.

Lambda will create an execution role named process-cloudwatch-logs-role-mjv5q412d, with permission to upload logs to Amazon CloudWatch Logs.

Advanced settings

Enable Code signing Info
The code signing configurations to ensure that the code has been signed by an approved source and has not been altered since signing.

Enable Function URL new Info
Use Function URLs to assign HTTPS endpoints to your Lambda functions.

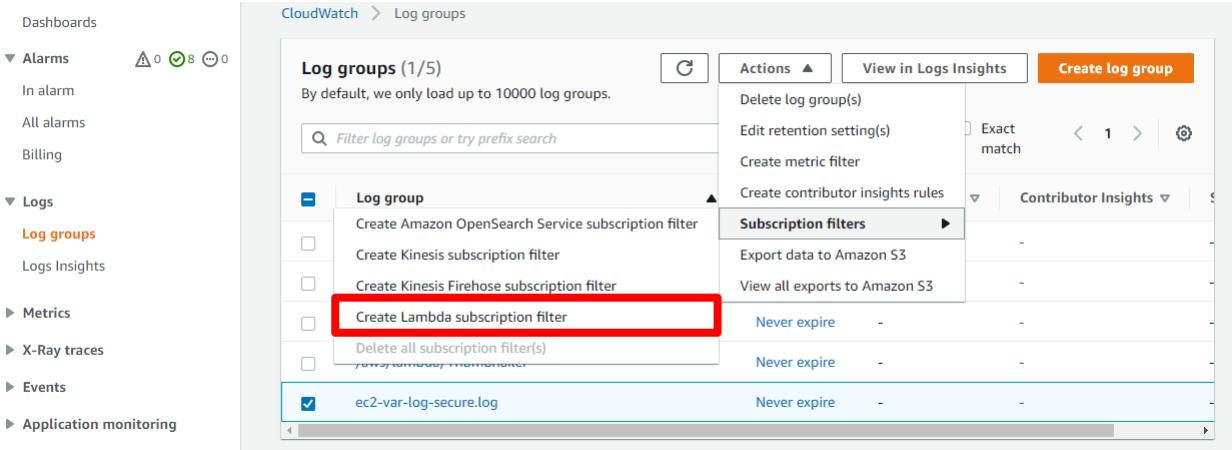
Enable VPC Info
Connect your function to a VPC to access private resources during execution.

Enable Tags Info
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources, track your AWS costs, and enforce attribute-based access control.

[Cancel](#) [Create Function](#)

Create CloudWatch subscription filter to call Lambda log processor and respond to events in EC2 logs

Back in the CloudWatch Console, click “Log groups” under Logs in the right hand nav, tick the checkbox next to “ec2-var-log-secure-log”, and then from the “Actions” dropdown, choose “Subscription filters→Create Lambda subscription filter”. If you’re confused, use the screenshot below as a guide.



This screenshot shows the 'Create Subscription Filter' dialog box. It has several sections: 1) 'Lambda function' dropdown set to 'process-cloudwatch-logs'. 2) A warning message about streaming large amounts of data to Lambda. 3) 'Configure log format and filters' section with 'Log format' set to 'Other' and a 'Subscription filter pattern' input field containing '[w1,w2,w3,w4,w5,w6=Invalid user,w7]'. 4) 'Test pattern' section with 'Select log data to test' dropdown set to 'ec2-var-log-secure.log'. 5) 'Log event messages' section showing several log entries. 6) 'Results' section showing a table with 6 matches. 7) 'Start streaming' button at the bottom right.

On the next page, point the subscription to our function (process-cloudwatch-logs), use “Other” log format, set the “Subscription filter pattern” to “[w1,w2,w3,w4,w5,w6=Invalid user,w7]” (your instructor will explain this pattern in detail), give the subscription a name (we used “invalid-user-subscription”), then under “Test pattern”, select our “ec2-var-log-secure.log” log, and push the “Test pattern” button to validate our Subscription filter pattern.

You should only see the results that match “Invalid user”, and you can see that we’ve tested our results (on the right) with several failed SSH attempts.

If all looks good, go ahead & press “Start streaming” to connect this final piece of our activity.

You’ll be redirected back to the “Log groups” page with a message saying “**Log events streamed to Amazon Lambda.**” at page top.

Validate operation of solution

Attempt to SSH to this server again, but this time do not supply the valid name "ec2-user". Try to log in with anything else like "BAD-USER" or "HACKING-ATTEMPT". Immediately after failed logins, you should see the appropriate entries in "ec2-var-log-secure.log". Here's what ours looked like after a few attempts:

▶ 2022-05-02T22:53:07.173-06:00	May 3 04:53:07 ip-172-31-0-66 sshd[844]: Invalid user I-AM-A-HACKER from 73.217.17.5...
▶ 2022-05-02T22:53:08.924-06:00	May 3 04:53:07 ip-172-31-0-66 sshd[844]: input_userauth_request: invalid user I-AM-A...
▶ 2022-05-02T22:53:13.467-06:00	May 3 04:53:08 ip-172-31-0-66 sshd[844]: Connection closed by 73.217.17.5 port 50024...
▶ 2022-05-02T23:20:31.724-06:00	May 3 05:20:31 ip-172-31-0-66 sshd[1036]: Invalid user BAD-BAD-USER from 73.217.17.5...
▶ 2022-05-02T23:20:32.974-06:00	May 3 05:20:31 ip-172-31-0-66 sshd[1036]: input_userauth_request: invalid user BAD-B...
▶ 2022-05-02T23:20:37.466-06:00	May 3 05:20:32 ip-172-31-0-66 sshd[1036]: Connection closed by 73.217.17.5 port 5095...
▶ 2022-05-02T23:22:09.521-06:00	May 3 05:22:09 ip-172-31-0-66 sshd[1039]: Invalid user HACKING-ATTEMPT from 73.217.1...
▶ 2022-05-02T23:22:11.272-06:00	May 3 05:22:09 ip-172-31-0-66 sshd[1039]: input_userauth_request: invalid user HACKI...
▶ 2022-05-02T23:22:15.466-06:00	May 3 05:22:11 ip-172-31-0-66 sshd[1039]: Connection closed by 73.217.17.5 port 5104...
▶ 2022-05-02T23:46:31.484-06:00	May 3 05:46:31 ip-172-31-0-66 sshd[1136]: Invalid user BAD-USER from 73.217.17.5 port...
▶ 2022-05-02T23:46:33.485-06:00	May 3 05:46:31 ip-172-31-0-66 sshd[1136]: input_userauth_request: invalid user BAD-U...
▶ 2022-05-02T23:46:38.466-06:00	May 3 05:46:33 ip-172-31-0-66 sshd[1136]: Connection closed by 73.217.17.5 port 5200...

After 10-20 seconds, back in "Log groups", you should see a new log group appear "process-cloudwatch-logs".



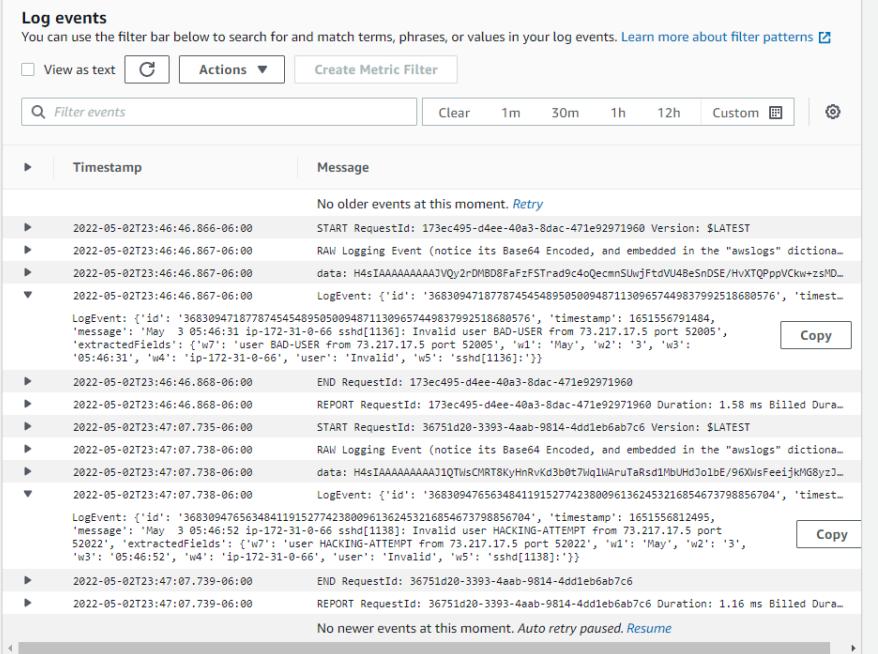
The screenshot shows the AWS CloudWatch Logs console. In the left sidebar, under the 'Logs' section, 'Log groups' is selected. A red arrow points to the 'process-cloudwatch-logs' entry in the main list. The list also includes other log groups like '/aws/lambda/check-set-tags', '/aws/lambda/RequestUnicorn', '/aws/lambda/Thumbtoggler', and 'ec2-var-log-secure.log'. Each entry has columns for 'Log group', 'Retention', 'Metric filters', and 'Contributor Insights'.

Go ahead & click into that group, click into the stream to view the entries inside. You should see something similar to our results on the right.

You can expand out the "LogEvent" entries to see what matched our patterns (note these are the ONLY entries that matched) and what was placed into the w1,w2, etc variables.

This solution could easily be extended to allow Lambda to notify us (via SNS), update a dashboard (which we'll do in the next lab), Add the remote IP to a WAF or SG rule block, etc...

The possibilities are endless!!!



The screenshot shows the AWS CloudWatch Logs console with the 'process-cloudwatch-logs' log group selected. The 'Log events' section displays several log entries. One entry is expanded to show its details. The expanded entry includes a timestamp ('2022-05-02T23:46:46.867-06:00'), a message ('LogEvent: {"id": "36830947187774545489505000497113086574408037992518680576", "timestamp": 1651556791484, "message": "May 3 05:46:31 ip-172-31-0-66 sshd[1136]: Invalid user BAD-USER from 73.217.17.5 port 52005", "extractedFields": {"w1": "user BAD-USER from 73.217.17.5 port 52005", "w2": "May", "w3": "3", "w4": "05:46:31", "w5": "ip-172-31-0-66", "user": "Invalid", "w6": "sshd[1136]"}}, and a 'Copy' button. Below this, there are more log entries and a 'No newer events at this moment. Auto retry paused. Resume' message.

***** END OF ACTIVITY #6 STOP HERE!!! *****

You can now set up an end-to-end log capture, ship, and analysis toolchain using CloudWatch Logs Agent, CloudWatch subscription filters, and Lambda.

BONUS Activity #7: Automating GuardDuty and EventBridge alerts to update CloudWatch dashboards

In this lab, we're going to set up a 'guard-duty-check' Lambda function to automatically be invoked by Amazon EventBridge when it EventBridge detects any GuardDuty finding occurrence in the high, medium, or low severity category. Our Lambda function will be invoked once per finding. The function additionally updates a CloudWatch dashboard with the count of the findings. Pseudocode for this function looks like:

```
get_high_medium_low_counts_from_current_dashboard {  
    determine_severity_of_this_finding()  
    increment_relevant_severity()  
    send sns notice_of_finding()  
    update_current_dashboard()  
}
```

Create SNS Topic, subscribe your email

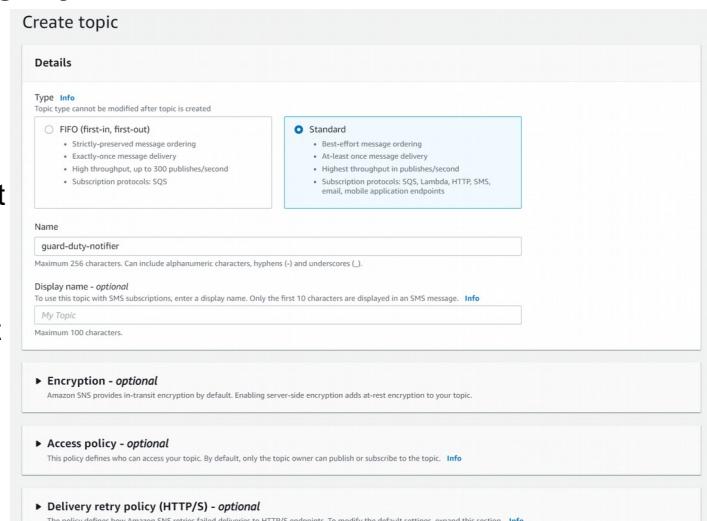
Browse to the [SNS console](#), click "Create topic", choose "Standard" as Topic type, give the topic the name "guard-duty-notifier", and press "Create topic" button.

You should see a message at the top of next page that says "Topic guard-duty-notifier" created successfully.

On this page, click the "Create subscription" button. For "Protocol", choose "Email", and enter an email that you have access to in the "Endpoint" text entry field. Click "Create subscription" button at bottom of page.

This email will have to be confirmed before you can start to receive email from Amazon. Flip over to your email client, and find the email that Amazon just sent to you.

Click the link in the email that says "Confirm subscription" to verify that



AWS Notification - Subscription Confirmation [External](#)

3:52 PM (0 minutes ago)

AWS Notifications <no-reply@sns.amazonaws.com>

to me ▾

You have chosen to subscribe to the topic:
arn:aws:sns:us-east-2:143230255374:guard-duty-notifier

To confirm this subscription, click or visit the link below (if this was in error no action is necessary):
[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)

you own this email.

You'll be directed to a page at AWS that says "Subscription confirmed!".

Enable GuardDuty

In another tab in your browser, open the [GuardDuty console](#), click the "Get Started" button, and then click the "Enable GuardDuty" button to turn GuardDuty on in this Region. (NOTE: AWS gives you a 30day free trial of GuardDuty, which is more than enough to perform the lab for free. Please make sure you drop a note in your calendar to disable GuardDuty before this 30 days runs out so you do not get charged for the service).

You'll be directed to a "Findings" page that looks something like below.

The screenshot shows the AWS GuardDuty console interface. On the left, a sidebar menu includes 'Findings' (which is highlighted in orange), 'Usage', 'Settings', 'Lists', 'S3 Protection', 'Kubernetes Protection' (with a 'New!' badge), and 'Accounts'. Below this are links for 'What's New' and 'Partners'. The main content area is titled 'Findings' and shows a message about 'Amazon GuardDuty for EKS Protection now available in AWS China Regions'. It includes a link to learn more. The main table header includes columns for 'Finding type', 'Resource', 'Last seen', and 'Count'. A message box at the bottom states 'You don't have any findings. GuardDuty continuously monitors your AWS environment and reports findings on this page.' with a 'Learn more' link.

We'll come back to this console later to generate some sample findings, but for now, let's just leave this tab open.

Create CloudWatch Dashboard

If you haven't yet, browse into the "/activity-4/" directory in the github package you downloaded for this class. You'll copy some assets from here into various places in the console and other areas in AWS.

Open (yet) another tab in your browser, and in that tab, browse to the [CloudWatch console](#). From the left rail nav, click on “Dashboards”, then click on the “Create dashboard” button in the main page. Give the dashboard the name “Security”. On the “Add widget” popup that immediately appears, choose the “Text” widget, and in the “Markdown” area, copy and past the text that appears in the “/activity-4/cloudwatch-dashboard.txt”. Push the “Create widget” button, then push the “Save dashboard” button to finish the dashboard creation. You should finally be presented with a page that looks like this:

Create Execution Role for Lambda function

Open another tab in your browser, and in that tab, browse to the [IAM console](#). Choose “Policies” from the left rail nav. Click the “Create Policy” button in the main window. In the page that appears, click the “JSON” tab at page top, and paste the text from “/activity-4/activity-4-lambda-policy-document.txt” over the JSON stub that’s pre-populated here. MAKE SURE TO CAREFULLY REPLACE YOUR “REGION” AND “ACCT ID” over the appropriate places in the “Resource” line for the “sns:Publish” action. You’ll get “ACCT ID” by selecting your “user @ acct” from the far upper right dropdown in the console, and the Region is what you’ve been using for this entire lab (make sure you use the AWS programmatic identifier like “us-east-1” rather than “virginia”). For example, if your Acct ID was “123456789012” and your region was Oregon, your full identifier for that line would look like this:

```
"Resource":  
"arn:aws:sns:us-west-  
2:123456789012:guard-duty-notifier"
```

Press the “Next:Tags” button to advance to the next step, Press “Next:Review”, give the policy the name “activity-4-lambda-policy”, then press “Create policy”.

You should see a green “The policy activity-4-lambda-policy has been created” notice appear on the next screen”.

Step 1: Select trusted entities

```
1 ~ [ {  
2 ~ "Version": "2012-10-17",  
3 ~ "Statement": [  
4 ~ {  
5 ~ "Effect": "Allow",  
6 ~ "Action": [  
7 ~ "sts:AssumeRole"  
8 ~ ]},  
9 ~ "Principal": {  
10 ~ "Service": [  
11 ~ "lambda.amazonaws.com"  
12 ~ ]}  
13 ~ }]  
14 ~ ]  
15 ~ ]  
16 ~ ]
```

Step 2: Add permissions

Policy name	Type	Attached as
activity-4-lambda-policy	Customer managed	Permissions policy

Step 3: Name, review, and create

Name, review, and create

Role details

Role name
activity-4-guardduty-role

Description
Allows Lambda functions to call AWS services on your behalf.

Now, on the left rail nav, choose “Roles”, then press the “Create role” button in the main page body. Leave “AWS service” selected (default), then under “Use case”, select “Lambda” and press the “Next” button. On the “Permissions policies” page, tick the check box next to “activity-4-lambda-policy” to choose it, then scroll to page bottom, and press the “Next” button. On the “Name, review, and create” page, give the Role the name “activity-4-guardduty-role”, scroll to page bottom and press the “Create role” button to finalize the role creation.

Create Lambda function

Open another tab in your browser, and in that tab, open a [Lambda console](#). In the upper right of the main page body, click the “Create function” button. Leave the default “Author from scratch” selected, name the function “guard-duty-check”, select “Python 3.8” as the Runtime, and under “Permissions”, expand “Change default execution role”, and select the radio button next to “Use an existing role”. In the dropdown that gets exposed below, choose the “activity-4-guardduty-role” that you created previously. NOTE: If you don’t see this role, you have to finalize the previous step, and you may have to press the “refresh” button just to the right of this dropdown.

Press the “Create function” button and wait a few seconds until you see the green “Successfully created function **guard-duty-check**” message appear at page top.

Once the message appears, you’ll see the “Code” tab selected in the main page. Browse to the “/activity-4/guard-duty-check.py” code from your github package. Copy that code into your clipboard, and paste it into the code editor in your Lambda console, overwriting the 8 lines of stub code that was pre-populated there for you. Similarly to how you changed YOUR-REGION:YOUR-ACCT-ID in the policy document step earlier, you’ll need to update the “sns_topic_arn” line under “init_global_vars” at the top of the code to put your specific REGION and ACCT_ID in place of the place holders. Press “Deploy” to put your code Live.

NOTE: If you’d like to tinker with the code later on, it’s useful to use “Versions” (far right tab at same level of “Code” to save a version like “Working” once you’ve got a working version of code).

Like all tabs we’ve opened, leave this one open... we’ll come back to it to do some testing later.

Create EventBridge rule to trigger Lambda function

Open another tab in your browser, and in that tab, open the [EventBridge console](#). From the left rail nav, select “Rules” and in the main page body, click the “Create rule” button. Give your rule the name “process-guardduty-

finding”, leave “Rule with an event pattern” (and all other defaults) selected, and press the “Next” button at page bottom.

On the next page, under “Sample event”, in the “Sample events” dropdown, scroll down to “GuardDuty Finding”. The text field will fill with a sample event that would’ve been generated by the GuardDuty service. Click the “Copy” button that appears below that text box to copy this text.

Flip back to your Lambda console tab. To the right of the “Test” button on the Code page, click the dropdown button and choose “Configure test event”. Give the event any name (we’ll use “GuardDuty-sample”), and paste in the event code you just copied from EventBridge in the “Event JSON” field (overwriting the text that was pre-populated there for you). Press “Save”. EventBridge gives us a really handy way to capture these events to feed them into our Lambda functions. We’re going to take advantage of that!

Leave Lambda console open... we’ll come back and test out a sample event in a bit.

Flip back to your EventBridge tab. At the bottom of the page, under “Event Pattern”, under “AWS service”, scroll down to and select “GuardDuty”. That will expose another drop down below it for “Event type”. In this dropdown, select “GuardDuty Finding” (we’re not interested in CloudTrail events). Press the “Next” button to move the workflow forward.

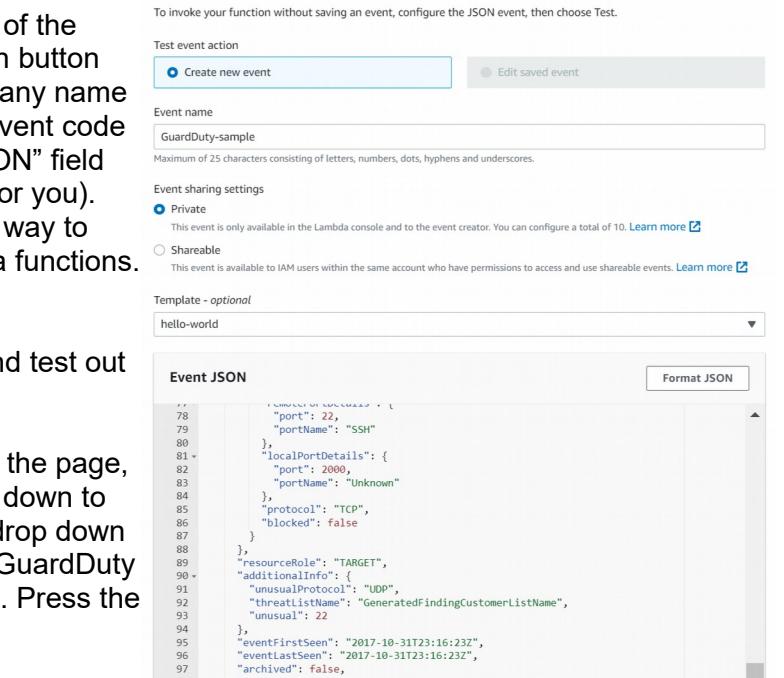
On the next page, for “Select target(s)”, leave “AWS service” (the default) selected, and under the dropdown for “Select a target”, choose “Lambda function”, which will expose a “Function” dropdown. From that dropdown, choose your “guard-duty-check” lambda function. Leave all other defaults as-is, and press the “Next” button at page bottom.

On the “Configure tags” page, choose “Next”. And on the “Review and create” page, verify that your settings resemble what is show at the right, and then press “Create rule” to finalize the creation process.

If successful, you’ll see a “Rule process-guardduty-finding was created successfully” green message appear at page stop and you’ll have your rule appear in the list below.

Test Lambda function with sample event

Before we let our function finally run wild, we probably want to give it a test, and we just set that up in the last step (thanks to EventBridge being kind enough to supply us with that handy event object!).



To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

- Create new event
- Edit saved event

Event name

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

- Private

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)
- Shareable

This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional

Format JSON

Event JSON

```

78     "port": 22,
79     "portName": "SSH"
80   },
81   "localPortDetails": {
82     "port": 2000,
83     "portName": "Unknown"
84   },
85   "protocol": "TCP",
86   "blocked": false
87 }
88 ],
89   "resourceRole": "TARGET",
90   "additionalInfo": {
91     "unusualProtocol": "UDP",
92     "threatListName": "GeneratedFindingCustomerListName",
93     "unusual": 22
94   },
95   "eventFirstSeen": "2017-10-31T23:16:23Z",
96   "eventLastSeen": "2017-10-31T23:16:23Z",
97   "archived": false,
98   "count": 1
99 }
```

Amazon EventBridge > Rules > Create rule

Step 1 Define rule detail

Step 2 Build event pattern

Step 3 Select target(s)

Step 4 - optional Configure tags

Step 5 Review and create

Review and create

Step 1: Define rule detail

Rule name	Status
process-guardduty-finding	Enabled
Description	Rule type
	Standard rule
Event bus	default

Step 2: Build event pattern

Event pattern [Info](#)

```

1 {
2   "source": ["aws.guardduty"],
3   "detail-type": ["GuardDuty Finding"]
4 }
```

[Copy](#)

Step 3: Select target(s)

Targets

Target Name	Type	Arn	Input	Role
guard-duty-check	Lambda function	arn:aws:lambda:us-east-2:143230255374:function:guard-duty-check	Matched event	-

Step 4: Configure tag(s)

Tags

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

Key	Value

[Cancel](#) [Previous](#) [Create rule](#)

Flip back to your Lambda console, and verify that you have successfully set up a Test event (click the dropdown to the right of the “Test” button & verify that you see at least one test – we named ours “GuardDuty-test” in the list that appears. Go ahead & press the “Test” button.

In your email client that you set up to receive your SNS emails, do a search for “to:my@email.com aws notifications”. Verify that you see a single email that looks similar to what we see below:



Additionally, if you flip over to the CloudWatch tab in your browser and browse to CloudWatch logs from the left rail nav, and then you choose “Log groups”, you should now see a “guard-duty-check” Log group”, and if you click into it, you should see at least one Log stream. In the most recent entry in the most recent stream, you’ll see a record of this test event. You should see text similar to what is show below (and you can uncomment lines in code to get more detail in the CW logs).

The screenshot shows the AWS CloudWatch Logs interface. The log group is "/aws/lambda/guard-duty-check". A log stream for the date 2022/05/10 is selected. The log entries are as follows:

Timestamp	Message
	No older events at this moment. Retry
2022-05-10T11:32:57.749-06:00	START RequestId: bbd8064-4508-45f2-ace2-88d17572c193 Version: \$LATEST
2022-05-10T11:32:58.285-06:00	[INFO]/2022-05-10 17:32:58,284/root - SUCCESS: pushed GuardDuty finding to SNS Topic
	[INFO]/2022-05-10 17:32:58,284/root - SUCCESS: pushed GuardDuty finding to SNS Topic
2022-05-10T11:32:58.519-06:00	END RequestId: bbd8064-4508-45f2-ace2-88d17572c193
2022-05-10T11:32:58.519-06:00	REPORT RequestId: bbd8064-4508-45f2-ace2-88d17572c193 Duration: 767.61 ms Billed Duration: 768 ms Memory Size: 128 MB Max M...
	No newer events at this moment. Auto retry paused . Resume

As one final check, we can flip over to our Cloudwatch Dashboard that we created previously, “Security”, and we should see that one of the checks (Medium in our case) incremented by 1.

Great! Now it looks like we’re ready to let our horses run free! Let’s throw a deluge of GuardDuty checks at our system and see how it performs!

Generate sample findings in GuardDuty

GuardDuty is a **fantastic** service, but it may take a while for it to actually generate findings. What if we don’t want to wait? Well GuardDuty has a fix for that! It has a capability to generate sample findings for us so we can actually see what it would do in the case of finding actual incidences. We’ll use this in our final step to throw an avalanche of findings at our little Lambda function to see how it performs under load.

In our [GuardDuty tab](#) (which hopefully we still have open, but if not, take extra care to ensure we're consistent with the same Region we've been using this whole time), select "Settings" from the left rail nav, and in the main page body, scroll down to "Sample findings".

ONLY PROCEED WITH THIS NEXT STEP ONCE YOU'VE VERIFIED ALL IS WORKING.

1. ARE YOU RECEIVING EMAILS?
2. IS YOUR CLOUDWATCH DASHBOARD UPDATING?
3. ARE YOUR CLOUDWATCH LOGS COMING THROUGH?
4. IS YOUR LAMBDA FUNCTION NOT ERRORING AT ALL?

ONLY IF YOU'RE SURE THE ANSWER TO EACH QUESTION ABOVE IS AN ABSOLUTE 'YES' DO YOU PROCEED.

Pressing the "Generate sample findings" button on this interface is going to generate dozens of emails in your account. It is going to fill up dozens of entries in your CloudWatch logs. It is going to result in dozens of firings of your Lambda function. *It may result in AWS costs.* Every. Time. You. Press. It. Exercise this button with caution.

Now that you know the danger, let's go ahead and RELEASE THE KRAKEN!!!

Press the "Generate sample findings" button.

On the left rail nav in GuardDuty, select the "Findings" text to see what you've generated. You should see something similar to what we see in the screenshot below.

New feature: Amazon GuardDuty for EKS Protection now available in AWS China Regions

Amazon GuardDuty for EKS Protection is now available in the Amazon Web Services China (Beijing) Region, operated by Sinnet, and the Amazon Web Services China (Ningxia) Region, operated by NWCD. Amazon GuardDuty for EKS Protection monitors Kubernetes audit logs to identify suspicious activity, such as API operations performed by known malicious or anonymous users, misconfigurations that can result in unauthorized access to Amazon Elastic Kubernetes Services (Amazon EKS) clusters, and patterns consistent with privilege-escalation techniques. GuardDuty customers can enable their 30-day EKS Protection free trial with a few clicks in the Amazon GuardDuty console Kubernetes Protection page. During the trial period you can see the estimated costs in the GuardDuty Console Usage page. To learn more about Amazon GuardDuty for EKS Protection and to find a full list of new EKS finding types, see the Amazon GuardDuty user guide. [Learn more](#)

Showing 102 of 102 12 39 51

Finding type	Resource	Last s...	Count
[SAMPLE] UnauthorizedAccess:EC2/MaliciousIPCaller.Custom	Instance: i-99999999	a minute ago	1
[SAMPLE] CredentialAccess:Kubernetes/TorlPCaller	EKSCluster: GeneratedFindingEKSClusterName	a minute ago	1
[SAMPLE] Recon:IAMUser/MaliciousIPCaller	GeneratedFindingUserName: GeneratedFindingAcce	a minute ago	1
[SAMPLE] Policy:S3/BucketAnonymousAccessGranted	GeneratedFindingUserName: GeneratedFindingAcce	a minute ago	1
[SAMPLE] Impact:Kubernetes/MaliciousIPCaller.Custom	EKSCluster: GeneratedFindingEKSClusterName	a minute ago	1
[SAMPLE] Discovery:Kubernetes/TorlPCaller	EKSCluster: GeneratedFindingEKSClusterName	a minute ago	1
[SAMPLE] Trojan:EC2/DropPoint	Instance: i-99999999	a minute ago	1
[SAMPLE] Trojan:EC2/DGADomainRequest.CIDNS	Instance: i-99999999	a minute ago	1
[SAMPLE] Discovery:IAMUser/AnomalousBehavior	GeneratedFindingUserName: GeneratedFindingAcce	a minute ago	1
[SAMPLE] Backdoor:EC2/DenialOfService.Udp	Instance: i-99999999	a minute ago	1

Notice the 12 / 39 / 51 #s for Low, Med, High severity. Correlate this with the 102 emails you just received.

to:rich@quicloud.com aws notifications

Active | ? | ⚙ | :: | quicloud

Mail Chat & spaces From Any time Has attachment Is unread Advanced search

1–25 of many < > [print]

Rich morrow rich@quicloud.com 303-325-5820

AWS Notifications

quicloud Kubernetes API commonly used in DefenseEvasion tactics invoked from a Tor exit node IP address. - : 11:51 AM

quicloud Unusual outbound communication on port 25 from EC2 instance i-99999999. - : "aws.guardduty", "ac... 11:51 AM

quicloud Command executed inside a pod in kube-system namespace. - : "aws.guardduty", "account": "143230... 11:51 AM

quicloud EC2 instance i-99999999 is behaving in a manner that may indicate it is being used to perform a Deni 11:51 AM

quicloud User GeneratedFindingUserType : GeneratedFindingUserName is anomalously invoking APIs comm... 11:51 AM

quicloud EC2 Instance is performing a Windows Remote Management brute force attack against a remote ho... 11:51 AM

quicloud The Kubeflow Dashboard was exposed to the Internet. - : "aws.guardduty", "account": "14323025537... 11:51 AM

quicloud Kubernetes API commonly used in Impact tactics invoked from an IP address on a custom threat list. 11:51 AM

quicloud DGA domain name queried by EC2 instance i-99999999. - : "aws.guardduty", "account": "1432302553... 11:51 AM

quicloud API GeneratedFindingAPIName was invoked from a remote host potentially running Parrot Security ... 11:51 AM

quicloud Domain related to cryptocurrency queried by EC2 instance i-99999999. - : "aws.guardduty", "account... 11:51 AM

quicloud Kubernetes API commonly used in Persistence tactics invoked from an IP address on a custom thre... 11:51 AM

quicloud Kubernetes API commonly used in Discovery tactics invoked from a Tor exit node IP address. - : "aw... 11:51 AM

quicloud EC2 instance i-99999999 is communicating with a Drop Point. - : "aws.guardduty", "account": "14323... 11:51 AM

quicloud API GeneratedFindingAPIName was invoked from a known malicious IP address. - : "aws.guardduty", "... 11:51 AM

quicloud API GeneratedFindingAPIName was invoked from an IP address on a custom threat list. - : "aws.gua... 11:51 AM

quicloud S3 API GeneratedFindingAPIName was invoked from known malicious IP address 198.51.100.0. - : "aws.guardd... 11:51 AM

quicloud Account password policy was weakened by calling GeneratedFindingAPIName. - : "aws.guardduty", "a... 11:51 AM

Do the counts match?

Let's take a look at your CloudWatch Dashboard. Do the counts match there?

CloudWatch > Dashboards > Security

Switch to your original interface

Security ☆ ☾

1h 3h 12h 1d 3d 1w Custom Actions Save dashboard Add widget

Search dashboards

GuardDuty Checks
High:6 Medium:3 Low:4

Questions:

1. Why do the counts in our CloudWatch Dashboard not match the actual counts we see in GuardDuty?
2. Can we think of a better way to get accurate counts in our Dashboard?
3. How could you rewrite the code (or maybe write other code) to ameliorate the count differences?

***** END OF ACTIVITY #7 STOP HERE!!! *****

You are now able to respond to GuardDuty findings with automated Lambda processors that update dashboards to provide real-time insight into operational issues.