

## Assignment 1: Autonomous Monitoring of Cryptographic Activity With KMS

Report by: Tonny Odhiambo, CS-CNS06-24028

### Introduction.

In this lab, I explored the critical aspects of autonomously monitoring cryptographic activity using AWS Key Management Service (KMS). The ability to provide traceability and automatically react to security events within an application is paramount for maintaining a high security posture. By monitoring key activity metrics, we can detect potentially malicious behaviour early and respond appropriately. This lab focused on creating an autonomous feedback loop to ensure cloud administrators are informed before a serious event occurs.

I utilized AWS CloudTrail to capture API-based activities within an AWS account, and Amazon CloudWatch to contextualize these events and create mechanisms for automatic reactions. This integration allowed me to produce adequate alerting and visibility to significant system events triggered by key activity metrics. Specifically, I monitored KMS activity, which is crucial for encryption and decryption within secure architecture designs. By detecting abnormal activity beyond predefined thresholds, I aimed to identify early warning signals of potential data exfiltration attempts.

Throughout this lab, I deployed the base infrastructure, configured the ECS repository, deployed the application stack, and set up workload logging and alarms. I also tested the workload functionality and cleaned up the resources. This comprehensive approach provided a hands-on understanding of autonomous monitoring and reaction strategies in a secure cloud environment.

### 1. Deploying the Lab Base Infrastructure

In this section, I built out a Virtual Public Cloud (VPC), complete with public and private subnets across two Availability Zones, an Internet Gateway, and a NAT gateway, along with the necessary routes for both public and private subnets. This VPC served as the baseline network architecture within which the application would run.

To deploy from the command line, I ensured that the AWS CLI was set up and configured with the appropriate credentials. Here are the steps I followed:

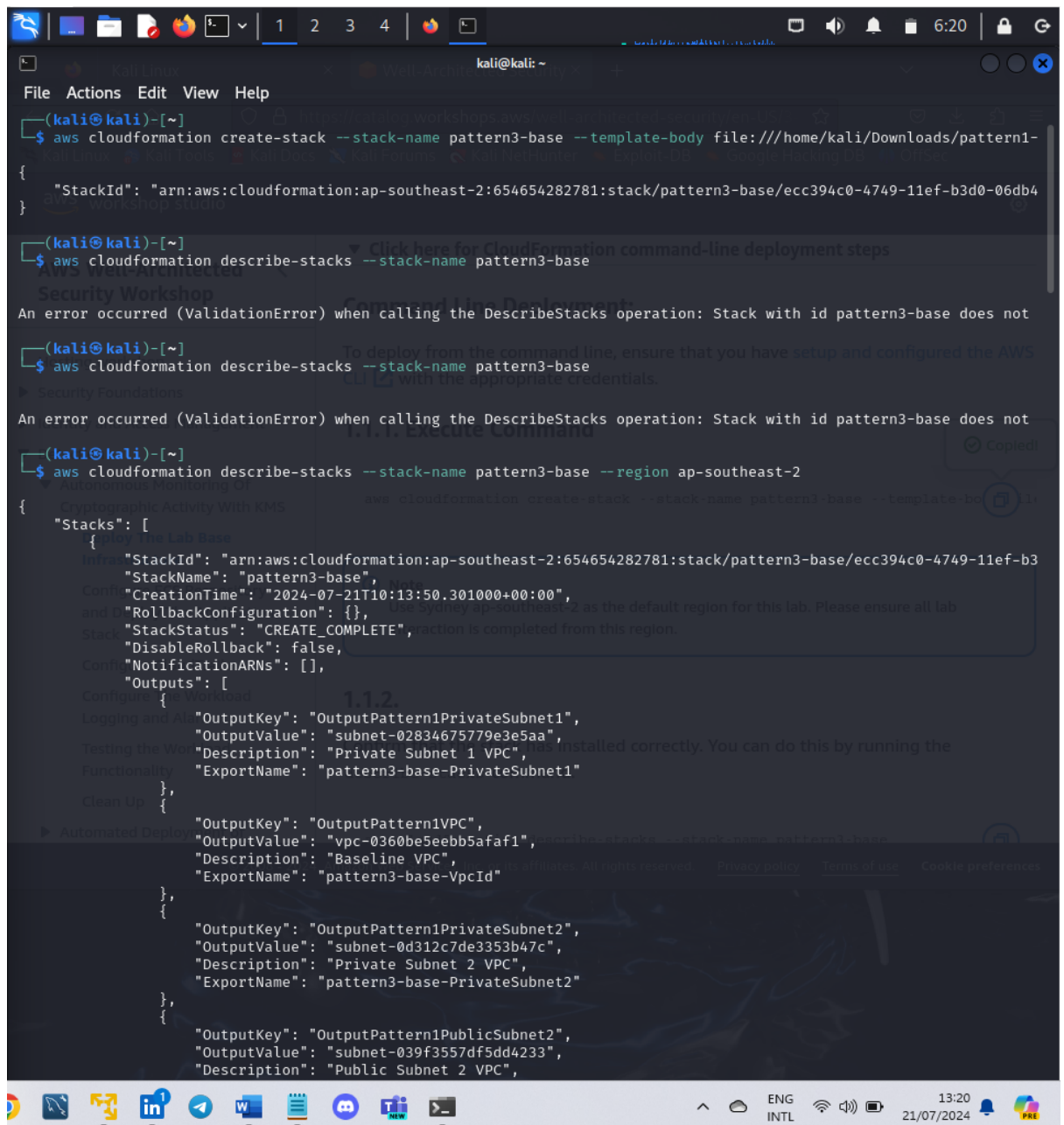
a) Execute Command:

```
aws cloudformation create-stack --stack-name pattern1-base --template-body  
file://pattern1-base.yml --region ap-southeast-2
```

b) Confirm Stack Installation:

To confirm that the stack was installed correctly, I ran the following command:

```
aws cloudformation describe-stacks --stack-name pattern3-base
```



```
(kali@kali)-[~]
$ aws cloudformation create-stack --stack-name pattern3-base --template-body file:///home/kali/Downloads/pattern1-
{
  "StackId": "arn:aws:cloudformation:ap-southeast-2:654654282781:stack/pattern3-base/ecc394c0-4749-11ef-b3d0-06db4
}

(kali@kali)-[~]
$ aws cloudformation describe-stacks --stack-name pattern3-base
An error occurred (ValidationError) when calling the DescribeStacks operation: Stack with id pattern3-base does not

(kali@kali)-[~]
$ aws cloudformation describe-stacks --stack-name pattern3-base
An error occurred (ValidationError) when calling the DescribeStacks operation: Stack with id pattern3-base does not

(kali@kali)-[~]
$ aws cloudformation describe-stacks --stack-name pattern3-base --region ap-southeast-2
{
  "Stacks": [
    {
      "StackId": "arn:aws:cloudformation:ap-southeast-2:654654282781:stack/pattern3-base/ecc394c0-4749-11ef-b3
      "StackName": "pattern3-base",
      "CreationTime": "2024-07-21T10:13:50.301000+00:00",
      "RollbackConfiguration": {},
      "StackStatus": "CREATE_COMPLETE",
      "DisableRollback": false,
      "NotificationARNs": [],
      "Outputs": [
        {
          "OutputKey": "OutputPattern1PrivateSubnet1",
          "OutputValue": "subnet-02834675779e3e5aa",
          "Description": "Private Subnet 1 VPC",
          "ExportName": "pattern3-base-PrivateSubnet1"
        },
        {
          "OutputKey": "OutputPattern1VPC",
          "OutputValue": "vpc-0360be5eebb5afaf1",
          "Description": "Baseline VPC",
          "ExportName": "pattern3-base-VpcId"
        },
        {
          "OutputKey": "OutputPattern1PrivateSubnet2",
          "OutputValue": "subnet-0d312c7de3353b47c",
          "Description": "Private Subnet 2 VPC",
          "ExportName": "pattern3-base-PrivateSubnet2"
        },
        {
          "OutputKey": "OutputPattern1PublicSubnet2",
          "OutputValue": "subnet-039f3557df5dd4233",
          "Description": "Public Subnet 2 VPC",
          "ExportName": ""
        }
      ]
    }
  ]
}
```

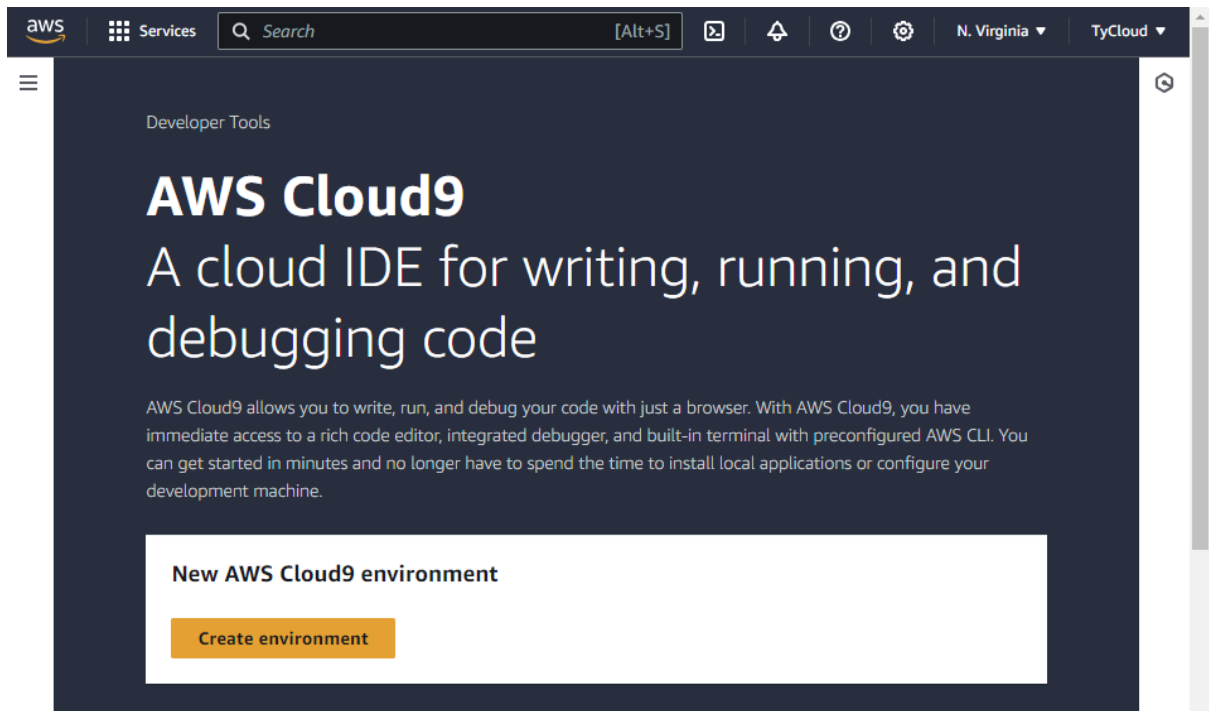
## 2. Configure the ECS Container Repository

### Environment Setup

To simplify the process of pushing the container to the repository, I used AWS Cloud9 IDE. This IDE comes pre-configured with all the necessary Docker components. I followed these steps to set up the environment:

#### a) Navigate to AWS Cloud9:

I accessed the AWS Cloud9 service and chose **"Create Environment"** from the welcome screen.



**b) Enter Environment Details:**

I provided the following details for the new environment:

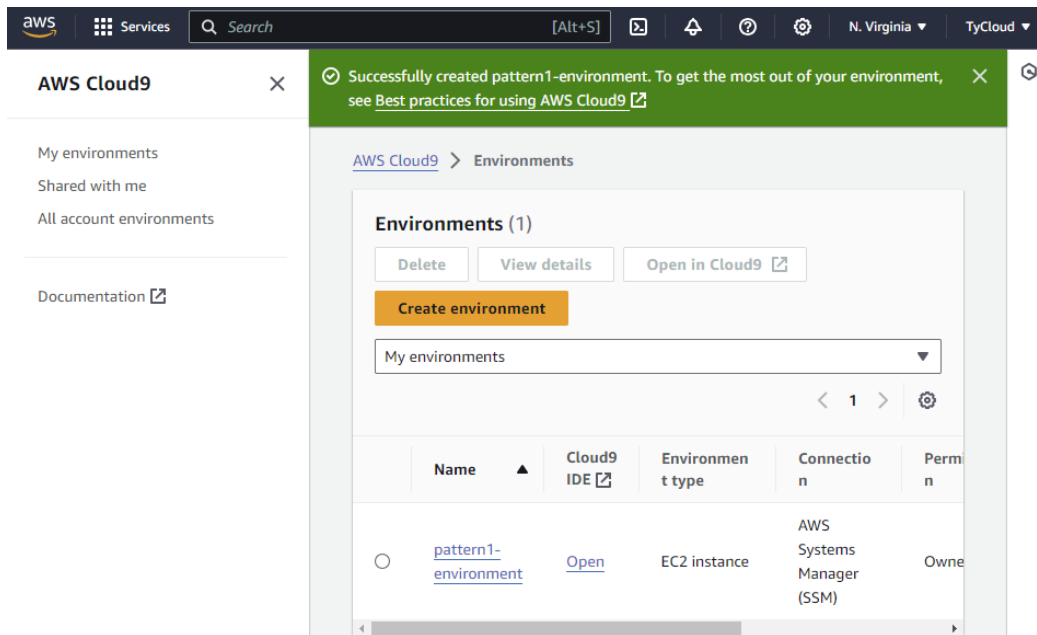
- a. Name: pattern1-environment
- b. Description: pattern1 environment

**c) Configure Settings:**

I accepted the default settings on the Configure Settings dialog box and chose "Next Step."

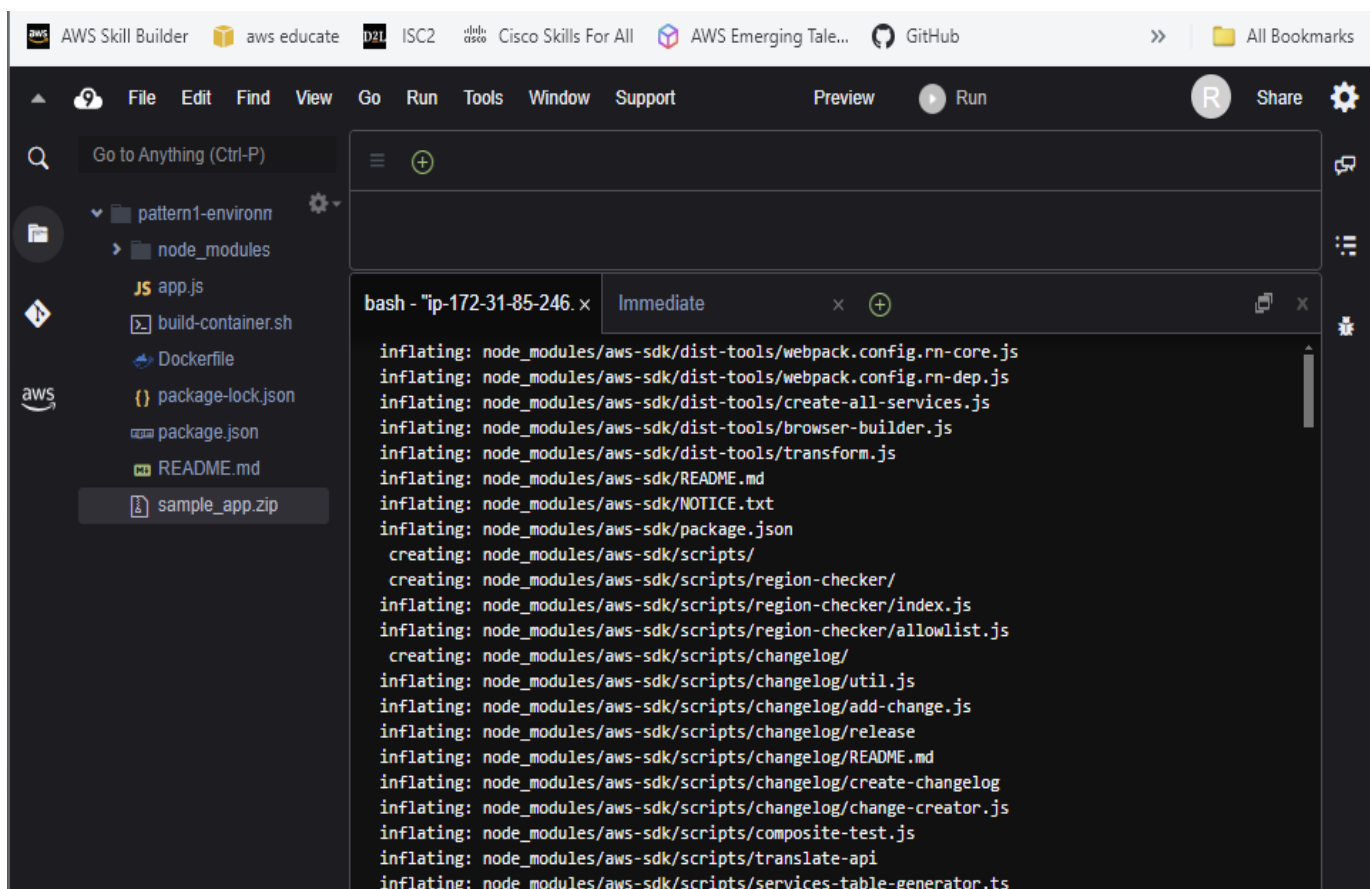
**d) Create Environment:**

On the Review dialog box, I selected "Create Environment." The Cloud9 IDE environment was then built, integrating the AWS command line and Docker components necessary for the lab. This process took a few minutes.



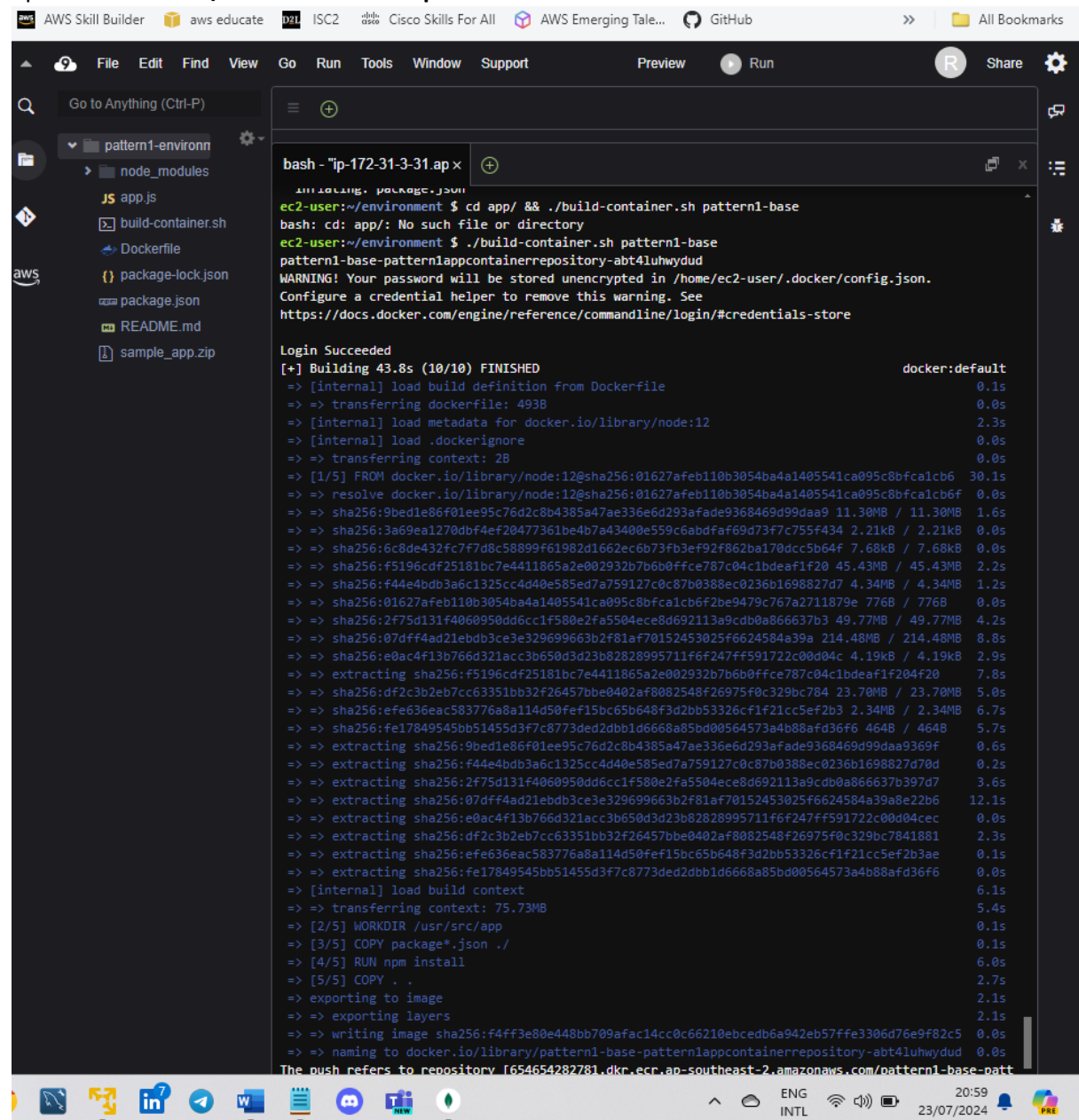
### e) Download and Unzip Application Files:

Once the environment was set up, I downloaded the application files by running the following command:



## f) Build and Upload Docker Image:

I navigated to the application directory and executed a script to build the Docker image and upload it to ECR: `./build-container.sh pattern1-base`



```
bash - "ip-172-31-31.ap x
Installing package.json
ec2-user:~/environment $ cd app/ && ./build-container.sh pattern1-base
bash: cd: app/: No such file or directory
ec2-user:~/environment $ ./build-container.sh pattern1-base
pattern1-base-pattern1appcontainerrepository-abt4luhwydud
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[+] Building 43.8s (10/10) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.1s
=> => transferring dockerfile: 493B                                0.0s
=> [internal] load metadata for docker.io/library/node:12         2.3s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                       0.0s
=> [1/5] FROM docker.io/library/node:12@sha256:01627afeb110b3054ba4a1405541ca095c8bfca1cb6 30.1s
=> => resolve docker.io/library/node:12@sha256:01627afeb110b3054ba4a1405541ca095c8bfca1cb6f 0.0s
=> => sha256:9bed1e86f01ee95c76d2c8b4385a47ae336e6d293afade9368469d99daa9 11.30MB / 11.30MB 1.6s
=> => sha256:3a69ea1270dbf4ef20477361be4b7a43400e559c6abdfaf69d73f7c755f434 2.21kB / 2.21kB 0.0s
=> => sha256:6c8de432fc7f7d8c58899f61982d1662ec6b73fb3ef92f862ba170dc5b64f 7.68kB / 7.68kB 0.0s
=> => sha256:f5196cdf25181bc7e4411865a2e002932b7b6b0ffce787c04c1bdeaf1f20 45.43MB / 45.43MB 2.2s
=> => sha256:f44e4bdb3a6c1325cc4d40e585ed7a759127c0c87b0388ec0236b169882d7d 4.34MB / 4.34MB 1.2s
=> => sha256:01627afeb110b3054ba4a1405541ca095c8bfca1cb6f2be9479c767a2711879e 776B / 776B 0.0s
=> => sha256:2f75d131f4060950dd6cc1f580e2fa5504ece8d692113a9cdb0a866637b3 49.77MB / 49.77MB 4.2s
=> => sha256:07dff4ad21ebdb3ce3e329699663b2f81af70152453025f6624584a39a 214.48MB / 214.48MB 8.8s
=> => sha256:e0ac4f13b766d321acc3b650d3d23b82828995711f6f247ff591722c00d04c 4.19kB / 4.19kB 2.9s
=> => extracting sha256:f5196cdf25181bc7e4411865a2e002932b7b6b0ffce787c04c1bdeaf1f204f20 7.8s
=> => sha256:df2c3b2eb7cc63351bb32f26457bbe0402af8082548f26975f0c329bc784 23.70MB / 23.70MB 5.0s
=> => sha256:efe636eac583776a8a114d50fef15bc65b648f3d2bb53326cf1f21cc5ef2b3 2.34MB / 2.34MB 6.7s
=> => sha256:fe17849545bb51455d3f7c8773ded2dbb1d6668a85bd00564573a4b88afd36f6 464B / 464B 5.7s
=> => extracting sha256:9bed1e86f01ee95c76d2c8b4385a47ae336e6d293afade9368469d99daa9369f 0.6s
=> => extracting sha256:f44e4bdb3a6c1325cc4d40e585ed7a759127c0c87b0388ec0236b169882d7d0d 0.2s
=> => extracting sha256:2f75d131f4060950dd6cc1f580e2fa5504ece8d692113a9cdb0a866637b397d7 3.6s
=> => extracting sha256:07dff4ad21ebdb3ce3e329699663b2f81af70152453025f6624584a39a8e22b6 12.1s
=> => extracting sha256:e0ac4f13b766d321acc3b650d3d23b82828995711f6f247ff591722c00d04cec 0.0s
=> => extracting sha256:df2c3b2eb7cc63351bb32f26457bbe0402af8082548f26975f0c329bc7841881 2.3s
=> => extracting sha256:efe636eac583776a8a114d50fef15bc65b648f3d2bb53326cf1f21cc5ef2b3a6 0.1s
=> => extracting sha256:fe17849545bb51455d3f7c8773ded2dbb1d6668a85bd00564573a4b88afd36f6 0.0s
=> [internal] load build context                                  6.1s
=> => transferring context: 75.73MB                                  5.4s
=> [2/5] WORKDIR /usr/src/app                                    0.1s
=> [3/5] COPY package*.json ./                                  0.1s
=> [4/5] RUN npm install                                         6.0s
=> [5/5] COPY . .                                                2.7s
=> => exporting to image                                             2.1s
=> => writing image sha256:f4ff3e80e448bb709afac14cc0c66210ebcd6b6a942eb57ffe3306d76e9f82c5 0.0s
=> => naming to docker.io/library/pattern1-base-pattern1appcontainerrepository-abt4luhwydud 0.0s
The push refers to repository [654654282781.dkr.ecr.ap-southeast-2.amazonaws.com/pattern1-base-patt
```

## g) Deploy The Application Stack

Next I deployed the application to [Amazon ECS](#)

```

(kali㉿kali)-[~]
$ aws cloudformation create-stack --stack-name pattern1-app \
  --template-body file:///home/kali/Downloads/pattern1-app.yml \
  --parameters ParameterKey=BaselineVpcStack,ParameterValue=pattern1-base \
    ParameterKey=ECRImageURI,ParameterValue=654654282781.dkr.ecr.ap-southeast-2.amazonaws.com/pattern1appcontainerrepository-abt4luhwydud:latest \
  --capabilities CAPABILITY_NAMED_IAM \
  --region ap-southeast-2
{
  "StackId": "arn:aws:cloudformation:ap-southeast-2:654654282781:stack/pattern1-app/c7f0c080-492b-11ef-b50f-0a9410"
}
(kali㉿kali)-[~]
$

```

I confirmed that the stack deployed successfully and was in CREATE\_COMPLETE state.

The screenshot displays the AWS Management Console interface. The main pane shows the 'CloudFormation' service with the 'Stacks' tab selected, displaying a list of stacks. The 'pattern1-app' stack is highlighted, showing its status as 'CREATE\_COMPLETE' with a green checkmark icon. The right-hand pane provides detailed information for the 'pattern1-app' stack, including buttons for 'Delete', 'Update', 'Stack actions', and 'Create stack'. The 'Stack info' tab is active, showing the stack's ID and other details. The 'Overview' tab is also visible at the bottom of the right-hand pane.

## h) Testing the Application

In this part of the lab, I tested the encrypt API of the sample application I just deployed. The application takes a JSON payload with "Name" and "Text" keys. It encrypts the value under the "Text" key using a designated KMS key, then stores the encrypted text in the RDS database with "Name" as the primary key.

```
586c0b414da7: Pushed
0bfd290f2c17: Pushed
6d75cd01c26c: Pushed
95904c181913: Pushed
df69bfa94785: Pushed
f35deb8d96fc: Pushed
f6c2459e2059: Pushed
f8323fb3a55c: Pushed
2f4dc9775f33: Pushed
latest: digest: sha256:4f85ffccae260c9fbca65031f6eabcba1d299d05d8425dd81d9915ed249b3acd size: 3054
=====
Image URI: 654654282781.dkr.ecr.ap-southeast-2.amazonaws.com/pattern1-base-pattern1appcontainerrepo
sitory-abt41uhwydud:latest
ec2-user:~/environment $ ALBURL="!Sub "${AWS::StackName}-ApplicationEndpoint""
bash: !Sub: event not found
ec2-user:~/environment $ curl --header "Content-Type: application/json" --request POST --data '{"Na
me":"Andy Jassy","Text":"Welcome to ReInvent 2020!"}' $ALBURL/encrypt
curl: (3) URL rejected: No host part in the URL
ec2-user:~/environment $ ALBURL="patter-Patte-BMO4ckXsGDYI-17565291.ap-southeast-2.elb.amazonaws.co
m"
ec2-user:~/environment $ curl --header "Content-Type: application/json" --request POST --data '{"Na
me":"Andy Jassy","Text":"Welcome to ReInvent 2020!"}' $ALBURL/encrypt
ec2-user:~/environment $
```

rer AWS: profile:default

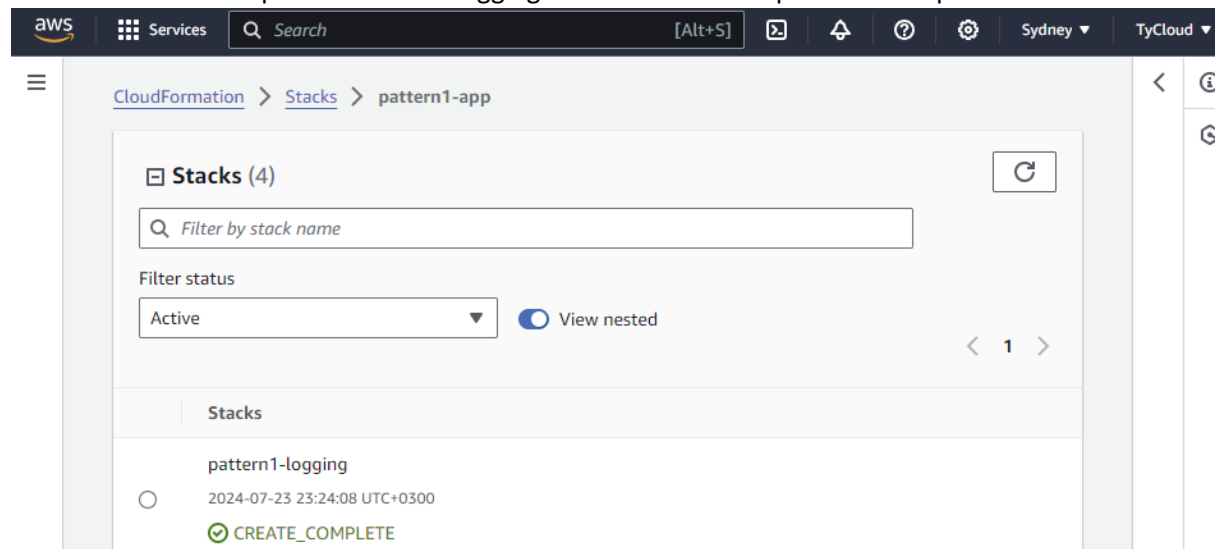
## 3. Configuring CloudTrail

I configured AWS CloudTrail to log all API calls within our architecture. This service will serve as the source of record, allowing us to apply filters later.

To deploy CloudTrail, I used the following CloudFormation command:

**aws cloudformation create-stack --stack-name pattern1-logging --template-body**  
<file:///path-to-logging-template.yml>

This command sets up the CloudTrail logging stack based on the provided template.



```
(kali@kali)-[~]
$ aws cloudformation create-stack --stack-name pattern1-logging --template-body file:///home/kali/Downloads/patter
n1-logging.yml --parameters ParameterKey=AppECSTaskRoleArn,ParameterValue="arn:aws:iam::654654282781:role/pattern1-a
pp-Pattern1ECSTaskRole" ParameterKey=EmailAddress,ParameterValue=yano.odhiambo@gmail.com --capabilities CAPABILITY_N
AMED_IAM --region ap-southeast-2
{
  "StackId": "arn:aws:cloudformation:ap-southeast-2:654654282781:stack/pattern1-logging/83b64380-4931-11ef-a570-0a
188faeed0d"
}
(kali@kali)-[~]
$
```

3.1. Command Line Deployment

Download the logging template

#### 4. Configuring Workload Logging and Alarm

##### a) Create CloudWatch Metric Filter

To create a CloudWatch metric filter, I navigated to CloudWatch, selected Log Groups, and chose pattern1-logging-loggroup. I then clicked Actions and selected Create metric filter, entering the filter pattern: `{ $.errorCode = "*" && $.eventSource= "kms.amazonaws.com" && $.userIdentity.sessionContext.sessionIssuer.arn = "arn:aws:iam::654654282781:role/pattern1-app-Pattern1ECSTaskRole" }`

I named the filter pattern1-logging-metricfilter, set the namespace to Pattern1Application/KMSecurity, and the metric name to KMSecurityError, with a metric value of 1, and created the metric filter.

aws

Services

Search

[Alt+S]

CloudWatch

Favorites and recents

Dashboards

Alarms 0 0 0 1

Logs

Log groups

Log Anomalies

Live Tail

Logs Insights

Contributor Insights

Metrics

X-Ray traces

Events

Application Signals New

Network monitoring

Insights

Settings

Getting Started

What's new

CloudWatch > Log groups > pattern1-logging-Pattern1CloudWatchLogGroup-1KrFYS78MEMx > Create metric filter

Step 1  
Define pattern

Step 2  
Assign metric

Step 3  
Review and create

Assign metric

Create filter name

Log events that match the pattern you define are recorded to the metric that you specify. You can graph the metric and set alarms to no

Filter name

pattern1-logging-metricfilter

Filter pattern

{ \$.errorCode = "\*" && \$.eventSource= "kms.amazonaws.com" && \$.userIdentity.sessionContext.sessionIssuer.arn =

Metric details

Metric namespace

Namespaces let you group similar metrics. Learn more

Pattern1Application/KMSecurity

Namespaces can be up to 255 characters long; all characters are valid except for colon(:) at the start of the name.

Metric name

Metric name identifies this metric, and must be unique within the namespace. Learn more

KMSecurityError

Metric name can be up to 255 characters long; all characters are valid except for colon(:), asterisk(\*), dollar(\$), and space( ).

Metric value

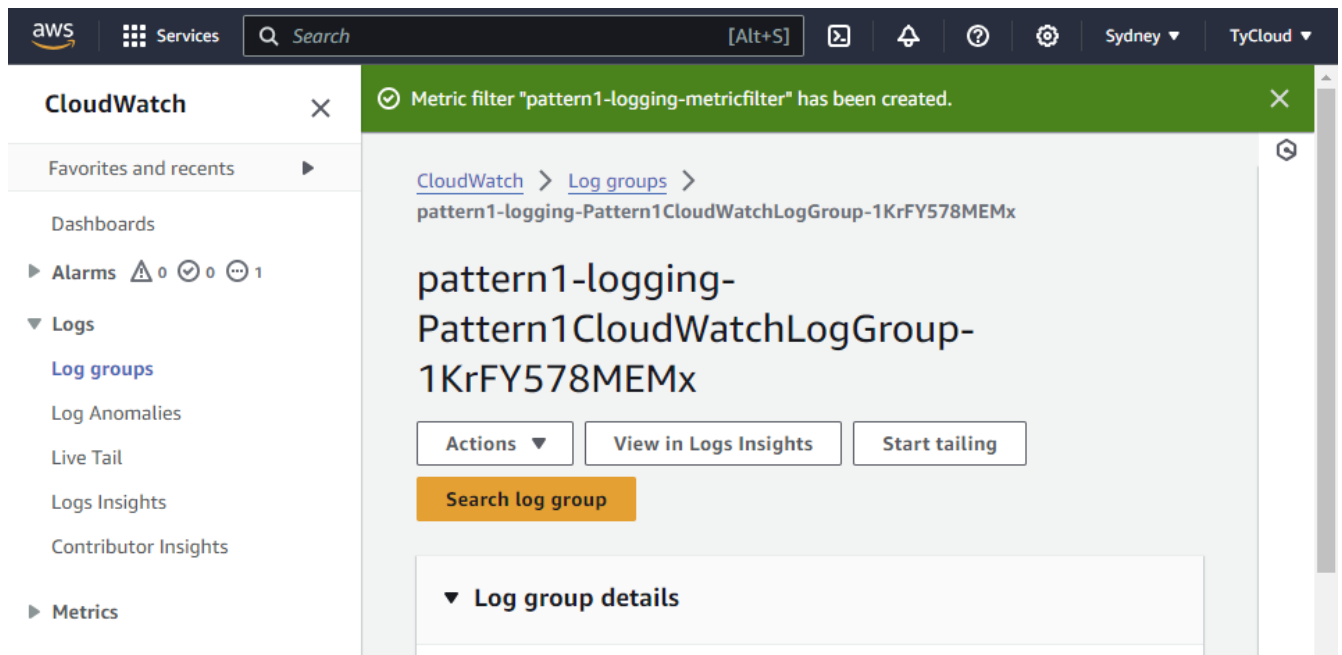
Metric value is the value published to the metric name when a Filter Pattern match occurs.

1

Valid metric values are: floating point number (1, 99.9, etc.), numeric field identifiers (\$1, \$2, etc.), or named field identifiers (e.g. \$.errorCode) followed by alphanumeric and/or underscore ( ) characters.



The metric filter was successfully created.



**b) Creating The Metric Alarm.**

To create the metric alarm, I selected the newly created metric filter, chose **Create Alarm**, named the alarm KMSsecurityError, set the period to 10 seconds, configured the conditions with a static threshold greater than 1, and set missing data treatment to "Ignore." I then configured the notification to create a new topic named pattern1-logging-topic, entered my email address, named the alarm pattern1-logging-alarm, reviewed the settings, and created the alarm.



Simple Notification Service

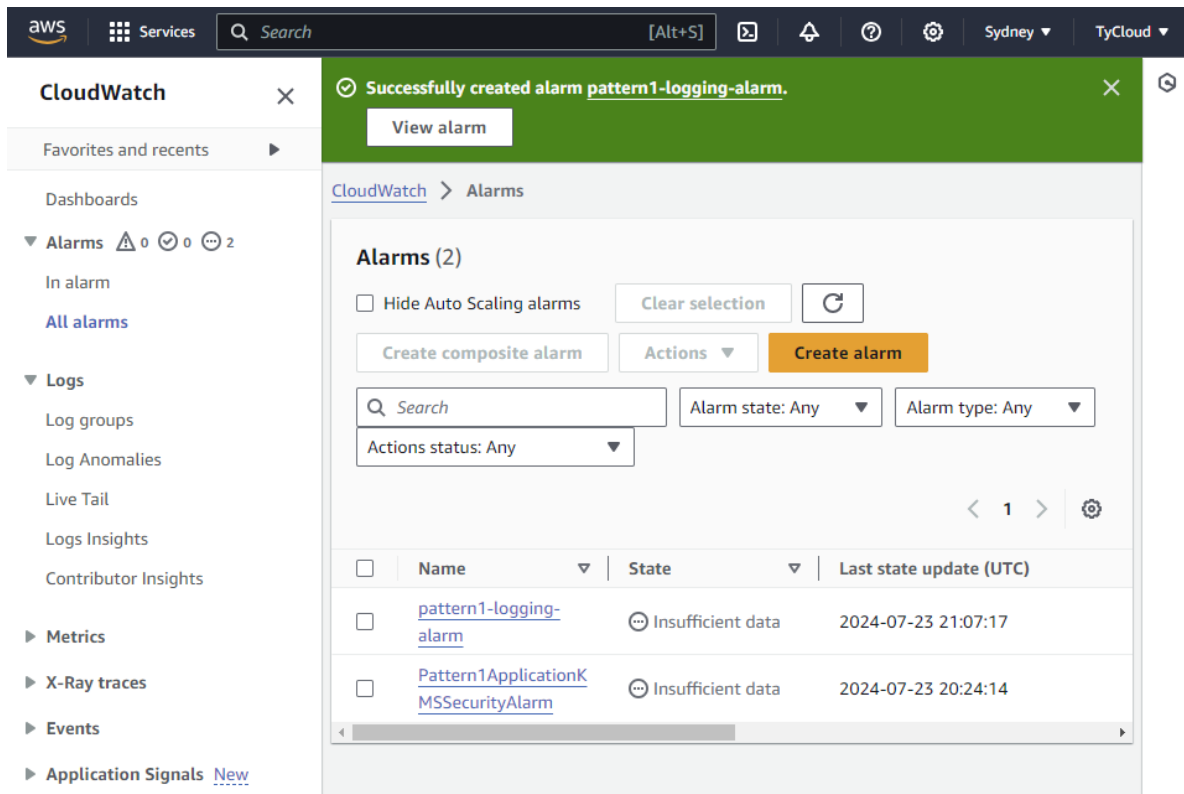
**Subscription confirmed!**

You have successfully subscribed.

Your subscription's id is:

arn:aws:sns:ap-southeast-2:654654282781:pattern1-logging-  
Pattern1CloudWatchAlarmTopic-C2us0wyS7007:2f62067d-9149-48cc-a1a6-  
b73e89f21d9b

If it was not your intention to subscribe, [click here to unsubscribe](#).



## 5. Testing the Workload Functionality

### a) Initiating a successful decryption operation

To initiate a successful decryption operation, I ran the command:

**ALBURL="http://patter-Patte-BMO4cKXsGDYI-17565291.ap-southeast-2.elb.amazonaws.com"**

**curl --header "Content-Type: application/json" --request GET --data '{"Name":"Andy Jassy","Key":"3ebc9c7d-5d1d-49c6-b1af-3b65acda2f81"}' \$ALBURL/decrypt**

```

76523705855C: Pushed
2f4dc9775f33: Pushed
latest: digest: sha256:4f85ffccae260c9fbca65031f6eabcba1d299d05d8425dd81d9915ed249b3acd size: 3054
=====
Image URI: 654654282781.dkr.ecr.ap-southeast-2.amazonaws.com/pattern1-base-pattern1appcontainerrepository-abt41uhwydud:latest
ec2-user:~/environment $ ALBURL="!Sub "${AWS::StackName}-ApplicationEndpoint""
bash: !Sub: event not found
ec2-user:~/environment $ curl --header "Content-Type: application/json" --request POST --data '{"Name":"Andy Jassy","Text":"Welcome to ReInvent 2020!"}' $ALBURL/encrypt
curl: (3) URL rejected: No host part in the URL
ec2-user:~/environment $ ALBURL="patter-Patte-BMO4cKXsGDYI-17565291.ap-southeast-2.elb.amazonaws.com"
ec2-user:~/environment $ curl --header "Content-Type: application/json" --request POST --data '{"Name":"Andy Jassy","Text":"Welcome to ReInvent 2020!"}' $ALBURL/encrypt
ec2-user:~/environment $ curl --header "Content-Type: application/json" --request POST --data '{"Name":"Andy Jassy","Text":"Welcome to ReInvent 2020!"}' $ALBURL/encrypt
{"Message":"Data encrypted and stored, keep your key save","Key":"3ebc9c7d-5d1d-49c6-b1af-3b65acda2f81"}
ec2-user:~/environment $ ALBURL="patter-Patte-BMO4cKXsGDYI-17565291.ap-southeast-2.elb.amazonaws.com"
ec2-user:~/environment $ curl --header "Content-Type: application/json" --request GET --data '{"Name":"Andy Jassy","Key":"3ebc9c7d-5d1d-49c6-b1af-3b65acda2f81"}' $ALBURL/decrypt
{"Text":"Welcome to ReInvent 2020!"}
ec2-user:~/environment $

```

The decryption was successful and the output "Welcome to ReInvent 2020!" was displayed as shown in the screenshot above.

### b) Initiating an unsuccessful decryption operation

To trigger a decryption failure and test the alerting, I re-ran the decryption command using an incorrect key value.

```
1"header "Content-Type: application/json" --request GET --data '{"Name": "Andy Jassy", "Key": "3ebc9c7d-3d1d-79c6-b1af-3b65acda2f81"}' $ALBURL/decrypt
{"Message": "Data decryption failed, make sure you have the correct key"}ec2-user:~/environment $ cu
r1 --header "Content-Type: application/json" --request GET --data '{"Name": "Andy Jassy", "Key": "3ebc9c7d-3d1d-79c6-b1af-3b65acda2f81"}' $ALBURL/decrypt
{"Message": "Data decryption failed, make sure you have the correct key"}ec2-user:~/environment $ cu
r1 --header "Content-Type: application/json" --request GET --data '{"Name": "Andy Jassy", "Key": "3ebc9c7d-3d1d-79c6-b1af-3b65acda2f81"}' $ALBURL/decrypt
{"Message": "Data decryption failed, make sure you have the correct key"}ec2-user:~/environment $ cu
r1 --header "Content-Type: application/json" --request GET --data '{"Name": "Andy Jassy", "Key": "3ebc9c7d-3d1d-79c6-b1af-3b65acda2f81"}' $ALBURL/decrypt
{"Message": "Data decryption failed, make sure you have the correct key"}ec2-user:~/environment $ cu
r1 --header "Content-Type: application/json" --request GET --data '{"Name": "Andy Jassy", "Key": "3ebc9c7d-3d1d-79c6-b1af-3b65acda2f81"}' $ALBURL/decrypt
{"Message": "Data decryption failed, make sure you have the correct key"}ec2-user:~/environment $ cu
r1 --header "Content-Type: application/json" --request GET --data '{"Name": "Andy Jassy", "Key": "3ebc9c7d-3d1d-79c6-b1af-3b65acda2f81"}' $ALBURL/decrypt
{"Message": "Data decryption failed, make sure you have the correct key"}ec2-user:~/environment $ cu
r1 --header "Content-Type: application/json" --request GET --data '{"Name": "Andy Jassy", "Key": "3ebc9c7d-3d1d-79c6-b1af-3b65acda2f81"}' $ALBURL/decrypt
{"Message": "Data decryption failed, make sure you have the correct key"}ec2-user:~/environment $ cu
r1 --header "Content-Type: application/json" --request GET --data '{"Name": "Andy Jassy", "Key": "3ebc9c7d-3d1d-79c6-b1af-3b65acda2f81"}' $ALBURL/decrypt
ec2-user:~/environment $
```

When re-ran with a different key, it fails as shown in the screenshot above. I re-ran it multiple times to trigger the alarm.

**c) Observing the alarm.**

To observe the alarm, I checked the email notification from CloudWatch, clicked the provided URL to access the CloudWatch Alarm resource, and reviewed the state changes under the History section to track activity.

### History (3)

Date (Local)	Type	Description
<a href="#">2024-07-24 00:42:59</a>	Action	Successfully executed action <code>arn:aws:sns:ap-southeast-2:654654282781:pattern1-logging-topic</code>
<a href="#">2024-07-24 00:42:59</a>	State update	Alarm updated from Insufficient data to <b>In alarm</b> .
<a href="#">2024-07-24 00:07:17</a>	Configuration update	Alarm "pattern1-logging-alarm" created

## ALARM: "Pattern1ApplicationKMSSecurityAlarm" in Asia Pacific (Sydney) Inbox x



**AWS Notifications**

00:43 (16 minutes ago)



to me ▾

You are receiving this email because your Amazon CloudWatch Alarm "Pattern1ApplicationKMSSecurityAlarm" in the Asia Pacific (Sydney) region has entered the ALARM state, because "Threshold Crossed: 1 datapoint [2.0 (23/07/24 21:42:00)] was greater than the threshold (1.0)." at "Tuesday 23 July, 2024 21:43:01 UTC".

View this alarm in the AWS Management Console:

<https://ap-southeast-2.console.aws.amazon.com/cloudwatch/deeplink.js?region=ap-southeast-2#alarmsV2:alarm/Pattern1ApplicationKMSSecurityAlarm>

### Alarm Details:

- Name: Pattern1ApplicationKMSSecurityAlarm
- Description: Pattern1 Application KMS Security Alarm.
- State Change: INSUFFICIENT\_DATA -> ALARM
- Reason for State Change: Threshold Crossed: 1 datapoint [2.0 (23/07/24 21:42:00)] was greater than the threshold (1.0).
- Timestamp: Tuesday 23 July, 2024 21:43:01 UTC
- AWS Account: 654654282781
- Alarm Arn: arn:aws:cloudwatch:ap-southeast-2:654654282781:alarm:Pattern1ApplicationKMSSecurityAlarm

### Threshold:

- The alarm is in the ALARM state when the metric is GreaterThanThreshold 1.0 for at least 1 of the last 1 period(s) of 10 seconds.

### Monitored Metric:

- MetricNamespace: Pattern1Application/KMSSecurity
- MetricName: KMSSecurityError
- Dimensions:
- Period: 10 seconds
- Statistic: Sum
- Unit: not specified
- TreatMissingData: ignore

## Conclusion

In this lab, I successfully deployed and configured a comprehensive infrastructure for monitoring cryptographic activity. I began by setting up the base infrastructure and configuring the ECS repository to deploy the application stack. Next, I integrated CloudTrail to capture and log API calls, ensuring all activities were recorded for future analysis. I then configured CloudWatch to monitor workload activities by creating metric filters and setting up alarms to trigger notifications on specific events.

After testing the functionality of the workload, including successful and unsuccessful decryption operations, I verified that the system was working as expected. The CloudWatch alarms provided timely alerts, confirming that the monitoring and alerting mechanisms were functioning correctly. Overall, the lab demonstrated a robust approach to autonomously monitoring cryptographic activities using AWS services, ensuring that all components were properly configured and integrated.