AccuKnox

DEPLOYMENT GUIDE

CLOUD DETECTION AND RESPONSE

AWS CDR Deployment Guide

Standalone AWS Account

A step by step build of the AWS CDR integration by hand, without Terraform. You create the S3 bucket and CloudTrail trail, stand up a Lambda log forwarder, point CloudTrail at it, and confirm the events land in the AccuKnox SIEM. The last part wires alerts to GitHub Actions so common AWS misconfigurations get fixed automatically, using the 7 detection rules that ship with the deployment.

WHAT YOU BUILD

A CloudTrail to Lambda to AccuKnox log pipeline in one AWS account, plus an automated remediation path through GitHub Actions.

WHO IT IS FOR

Cloud and security engineers with console access to the target AWS account and admin rights on a GitHub repository.

BEFORE YOU START

Get the ingestion URL and Basic Auth credentials from the AccuKnox team. Budget about 90 minutes for the full walkthrough.

DEPLOYMENT ARCHITECTURE

CloudTrail ▶ Amazon S3 ▶ Lambda forwarder ▶ AccuKnox CDR ▶ Automated response

What this guide covers

Twelve steps, in order. Each one opens with why it exists, then the console path, then the exact values to enter. 51 annotated console screenshots show what the screen should look like when the step is done. Step 12 lists the 7 detection rules that ship with the deployment, and three worked use cases close the guide.

> **Skip ahead if your account is already logging**
>
> Already have an S3 bucket and a CloudTrail trail writing to it? Start at step 1, the Lambda execution role. The bucket and trail setup sits in an optional section just before it, marked **Only if not already configured**. Check first that the existing trail delivers to the existing bucket with an empty log prefix.

01 Create the Lambda execution IAM role
An execution role Lambda can assume, with CloudWatch Logs write access.

02 Add the S3 read inline policy
Scoped s3:GetObject on the CloudTrail log bucket only.

03 Create the Lambda function
Python 3.10 function on x86_64, wired to the execution role.

04 Upload the Lambda function code
The forwarder that unzips, batches and POSTs CloudTrail records.

05 Configure Lambda environment variables
Ingestion URL and Basic Auth credentials from the AccuKnox team.

06 Configure the Amazon S3 trigger
ObjectCreated notifications invoke the function on every new log file.

07 Validate the Lambda invocation
Read the CloudWatch log stream and confirm batches were posted.

08 Verify CloudTrail logs in the SIEM UI
Filter on cloud = aws and confirm events are searchable.

09 Configure the GitHub remediation repository
Actions workflow plus repository secrets for the target cloud.

10 Verify alerts in the AccuKnox platform
Confirm CDR alerts fire for the rules you plan to remediate.

11 Configure the GitHub webhook integration
Repository Dispatch call that starts the remediation workflow.

12 Review the out-of-the-box AWS rules
Seven pre-built detections, each with an automated response.

AUTOMATED REMEDIATION, PROVEN END TO END

Use case 1. Public S3 buckets
Block Public Access re-enabled by Cloud Custodian.

Use case 2. Exposed security group ports
Inbound 0.0.0.0/0 rule revoked automatically.

Use case 3. EC2 with public IPs
Public IP disassociated, instance keeps running.

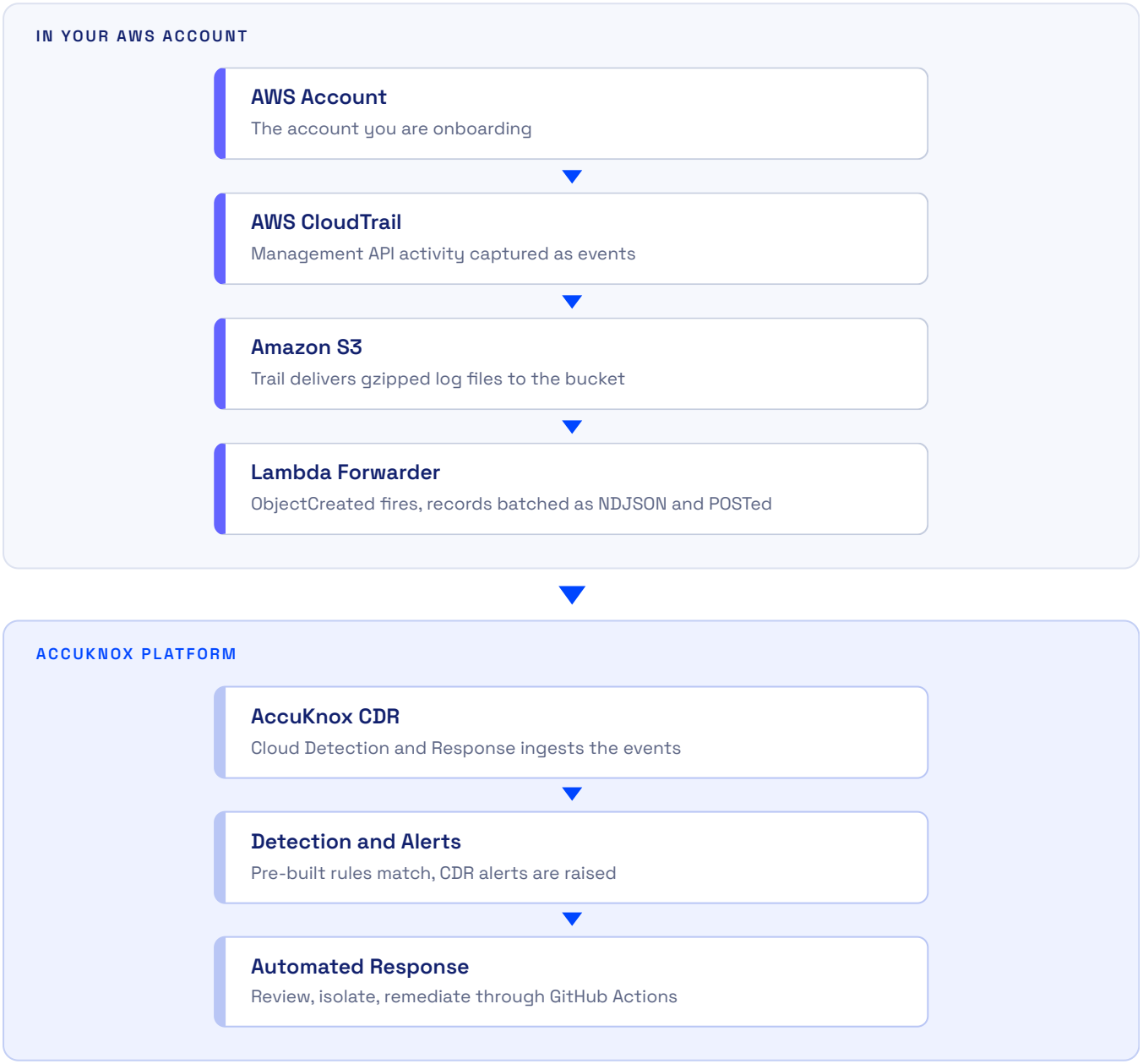
Conventions. Console paths render as chips, for example `AWS Console` ▶ `Lambda`. Values you type appear in a table next to the setting name. Red bordered boxes are notes you should read before clicking. Screenshots came from the source export at reduced resolution, so fine console text may look soft in print.

Overview

This document describes the manual deployment process for the AWS CDR integration.

The guide walks through creating all required AWS resources, configuring the Lambda log forwarder, enabling CloudTrail log collection, and forwarding CloudTrail events to the AccuKnox SIEM/CDR platform.

High-Level Deployment Architecture



Prerequisite

If you already have an **Amazon S3 bucket** and an **AWS CloudTrail** trail configured, skip the two prerequisite steps below and start at Step 1, the Lambda execution IAM role.

Note: Before you start Step 1, check that your existing CloudTrail trail delivers logs to the existing Amazon S3 bucket.

Verify the CloudTrail Storage Configuration

Setting	Value
Storage	Use Existing S3 Bucket
Bucket	Your existing CloudTrail log bucket (e.g., `accuknox-cloudtrail-sabpaisa`)
Log Prefix	Leave Empty

CloudTrail should store log files in the following location:

```
s3://<your-existing-cloudtrail-bucket>/AWSLogs/<Account-ID>/
```

Already have both? Use the section below only to check the existing configuration. Otherwise follow its two prerequisite steps to create the bucket and the trail.

ONLY IF NOT ALREADY CONFIGURED **Verify Existing S3 Bucket & CloudTrail Configuration**

Prerequisite Step 1, Create an Amazon S3 Bucket

Purpose

Create an Amazon S3 bucket to store AWS CloudTrail log files. CloudTrail requires an S3 bucket as the destination for storing audit logs generated from AWS API activity.

Procedure

Navigate to:

AWS Console ▶ Amazon S3 ▶ Create Bucket

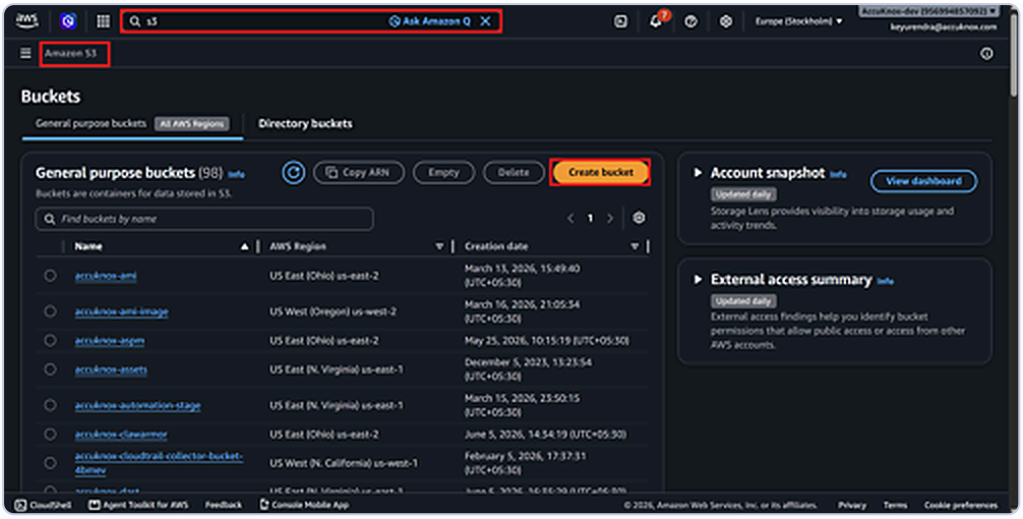


Figure 1. Prerequisite Step 1, Create an Amazon S3 Bucket

Bucket Configuration

Setting	Value
Region	Europe (Stockholm)
Bucket Type	General Purpose
Namespace	Global Namespace
Bucket Name	`accuknox-cloudtrail-sabpaisa`
Copy Settings	None
Object Ownership	ACL Disabled
Block Public Access	Enable all (Default)
Versioning	Disabled
Tags	Optional
Default Encryption	SSE-S3
Bucket Key	Disabled
Advanced Settings	Default

After configuring the above settings, click Create Bucket.

Note: The bucket name must be globally unique across all AWS accounts. If the suggested name is already in use, choose a different unique name.

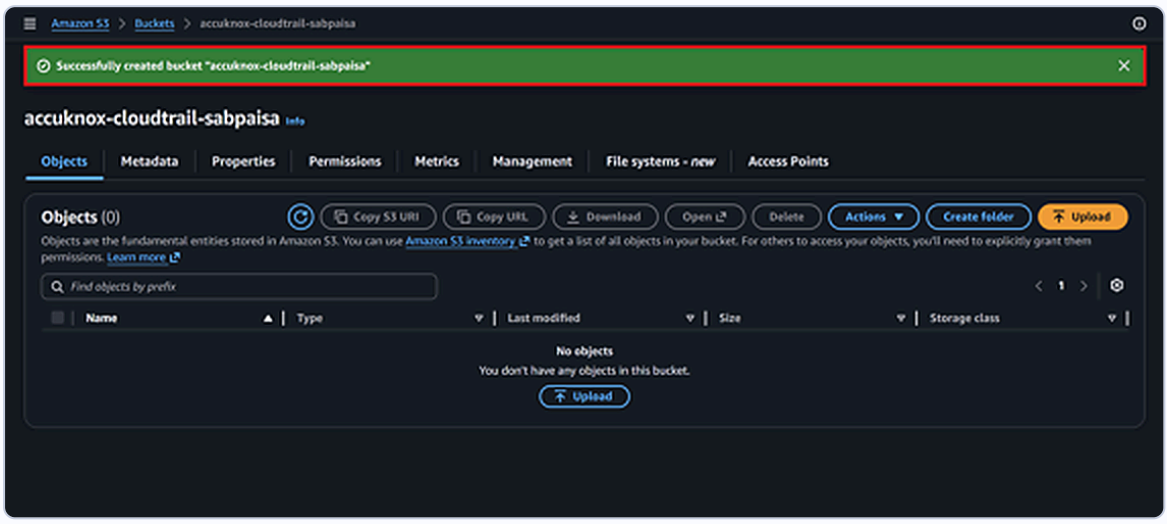


Figure 2. Bucket Configuration

Prerequisite Step 2, Create an AWS CloudTrail Trail

Purpose

Create an AWS CloudTrail trail to capture AWS management API activity and deliver audit logs to the Amazon S3 bucket created in the previous step.

Procedure

Navigate to:

AWS Console ▶ CloudTrail ▶ Trails ▶ Create Trail

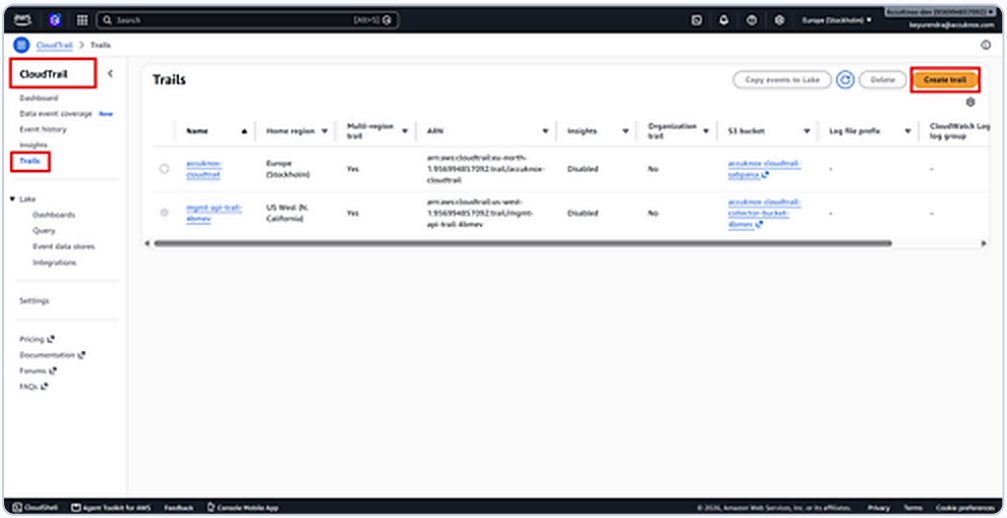


Figure 3. Prerequisite Step 2, Create an AWS CloudTrail Trail

General Details

Setting	Value
Trail Name	`accuknox-cloudtrail`
Trail Type	Single Account Trail
Enable for all accounts in my organization	No (Disabled)

Note: This deployment is designed for a single AWS account. Organization trails are not required.

Storage Location

Configure CloudTrail to store logs in the Amazon S3 bucket created in Step 1.

Setting	Value
Storage	Use Existing S3 Bucket
Bucket	`accuknox-cloudtrail-sabpaisa`
Log Prefix	Leave Empty

CloudTrail will store logs in the following path:

```
s3://accuknox-cloudtrail-sabpaisa/AWSLogs/<Account-ID>/
```

Encryption & Additional Settings

Setting	Value	Remarks
Log File SSE-KMS Encryption	Disabled	Not required by this deployment.
Log File Validation	Enabled	Confirms CloudTrail log integrity.
SNS Notification Delivery	Disabled	Not used in this deployment.
CloudWatch Logs	Disabled	CloudTrail logs are written to Amazon S3. Lambda execution logs are sent to CloudWatch Logs separately.
Tags	Optional	Add organization-specific tags if required.

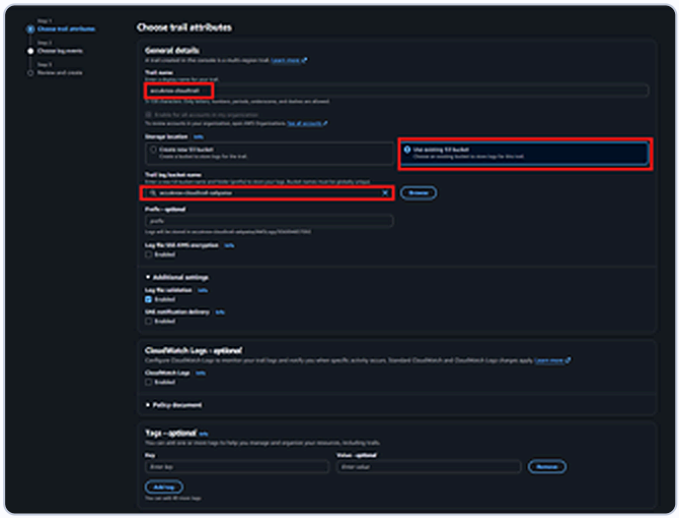


Figure 4. Encryption & Additional Settings

Event Settings

Event Type	Selection
Management Events	Enabled
Data Events	Disabled
Insights Events	Disabled
Network Activity Events	Disabled

Management Event Configuration

Setting	Value
API Activity	Read + Write
Exclude AWS KMS Events	Enabled
Exclude Amazon RDS Data API Events	Enabled

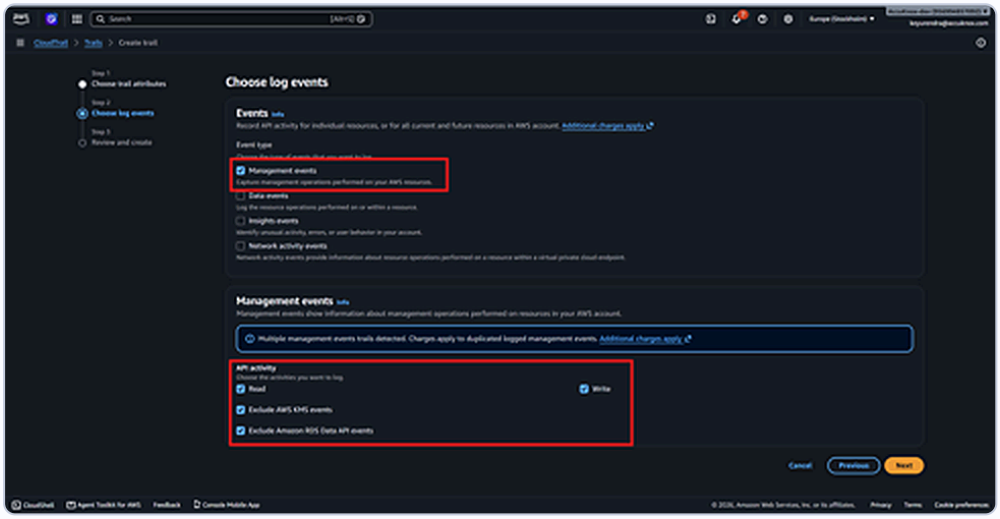


Figure 5. Management Event Configuration

After reviewing the configuration, click Create Trail.

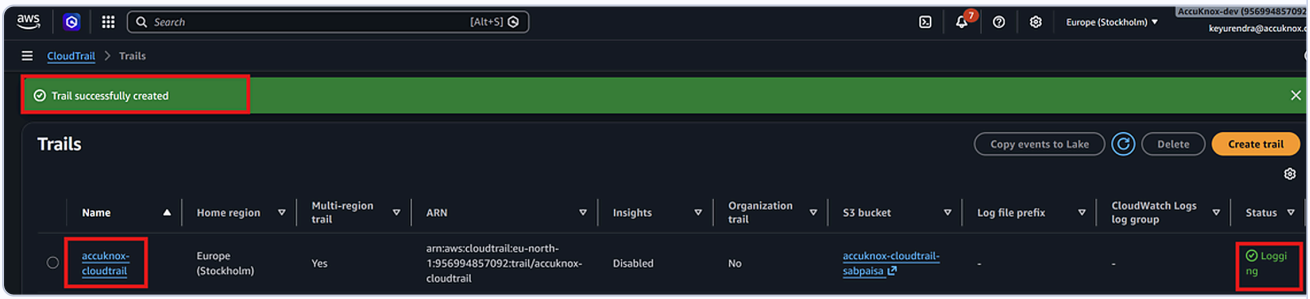


Figure 6. Management Event Configuration

Step 1. Create the Lambda Execution IAM Role

Purpose

Create an IAM execution role that allows the AWS Lambda function to write execution logs to Amazon CloudWatch Logs and access CloudTrail log files stored in the Amazon S3 bucket.

Procedure

Navigate to:

AWS Console ▶ IAM ▶ Roles ▶ Create Role

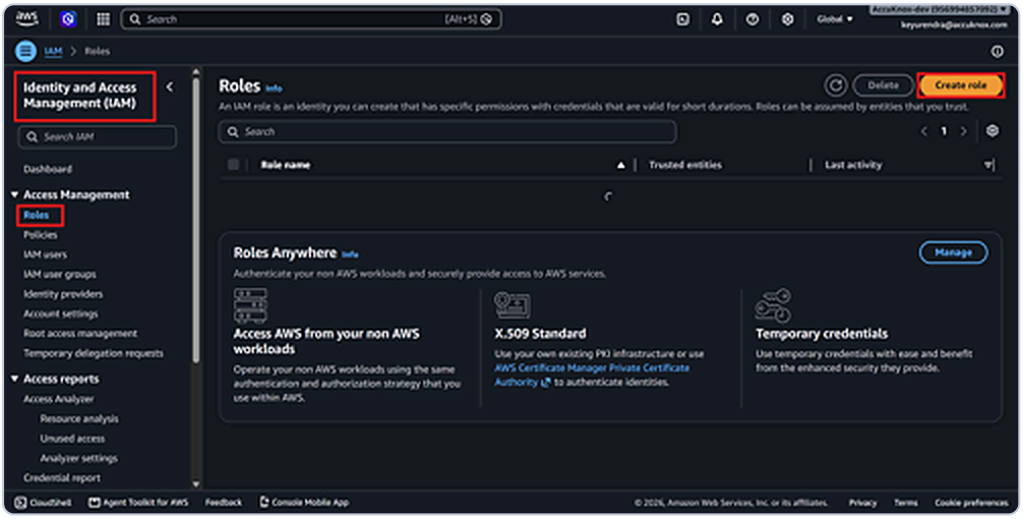


Figure 7. Step 1. Create the Lambda Execution IAM Role

Trusted Entity

Configure the following:

Setting	Value
Trusted Entity Type	AWS Service
Use Case	Lambda

Click Next.

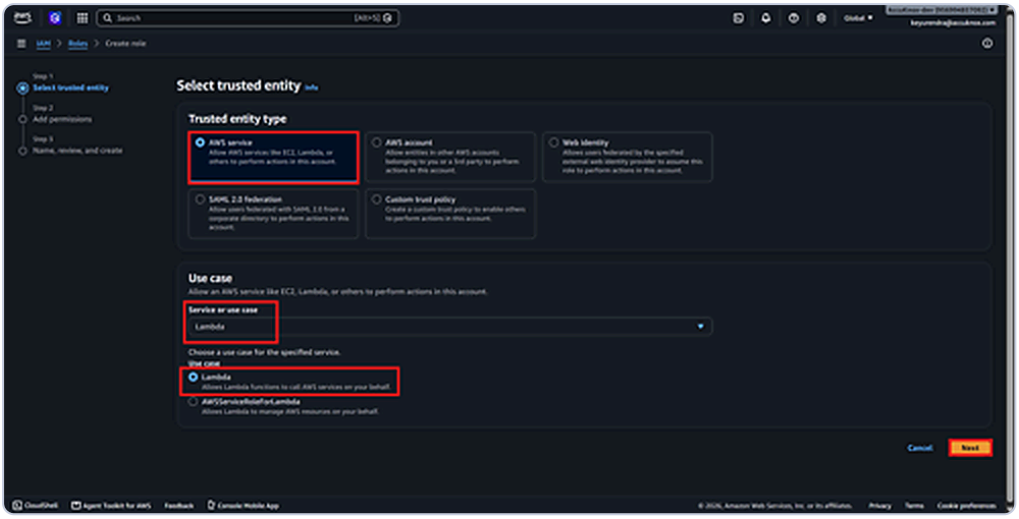


Figure 8. Trusted Entity

Permissions

Attach the following AWS managed policy:

Policy	Type
`AWSLambdaBasicExecutionRole`	AWS Managed Policy

This policy grants the Lambda function permission to:

- Create CloudWatch Log Groups
- Create CloudWatch Log Streams
- Write CloudWatch Log Events

Click Next.

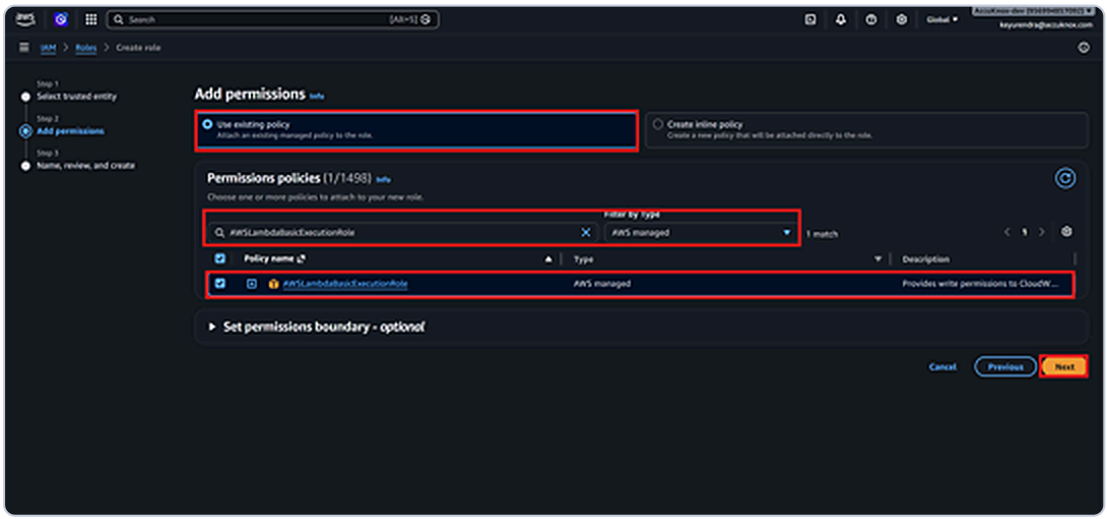


Figure 9. Permissions

Role Details

Configure the role using the following values:

Setting	Value
Role Name	`accuknox-lambda-exec-role`
Description	Allows the AccuKnox CloudTrail Lambda function to read CloudTrail logs from Amazon S3 and forward them to AccuKnox SIEM/CDR.

Trust Policy

Leave the default trust policy unchanged.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      }
    }
  ]
}
```

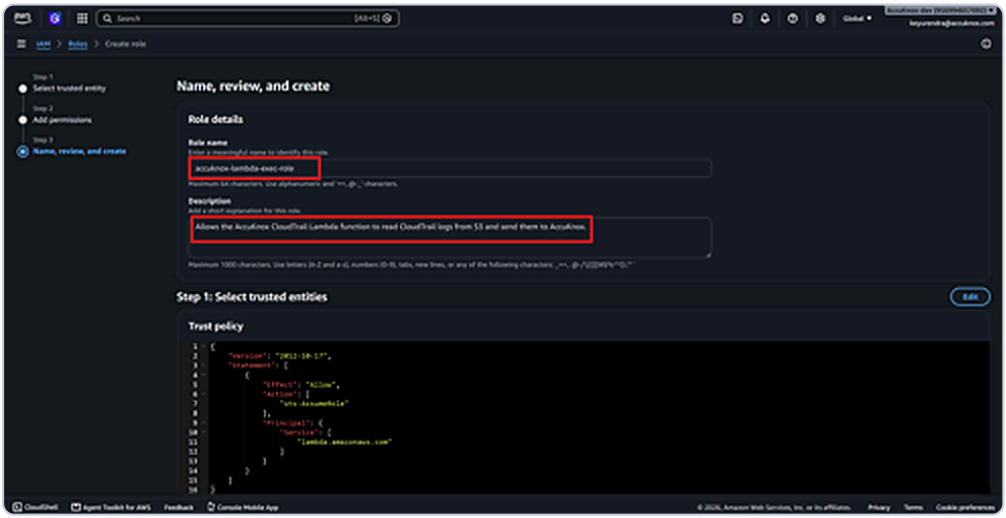


Figure 10. Trust Policy

Tags (Optional)

Key	Value
Owner	AccuKnox CDR Collector

After reviewing the configuration, click Create Role.

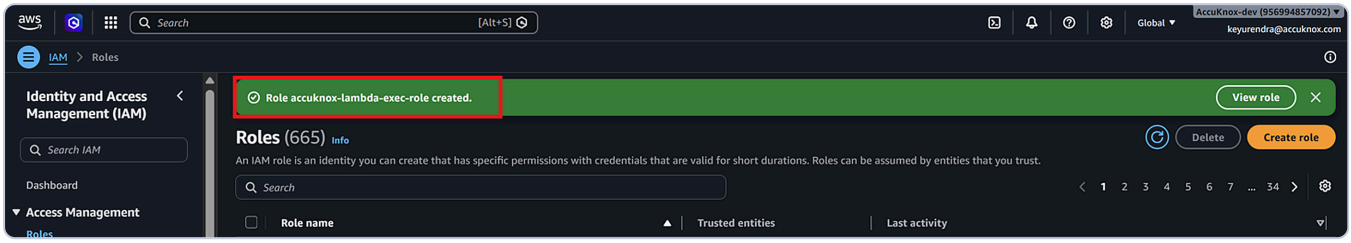


Figure 11. Tags (Optional)

Step 2. Add the S3 Read Inline Policy

Purpose

Attach an inline IAM policy to the Lambda execution role to grant read access to CloudTrail log files stored in the Amazon S3 bucket.

Procedure

Navigate to:

AWS Console ▶ IAM ▶ Roles ▶ accuknox-lambda-exec-role ▶ Add Permissions ▶ Click Attach Policies

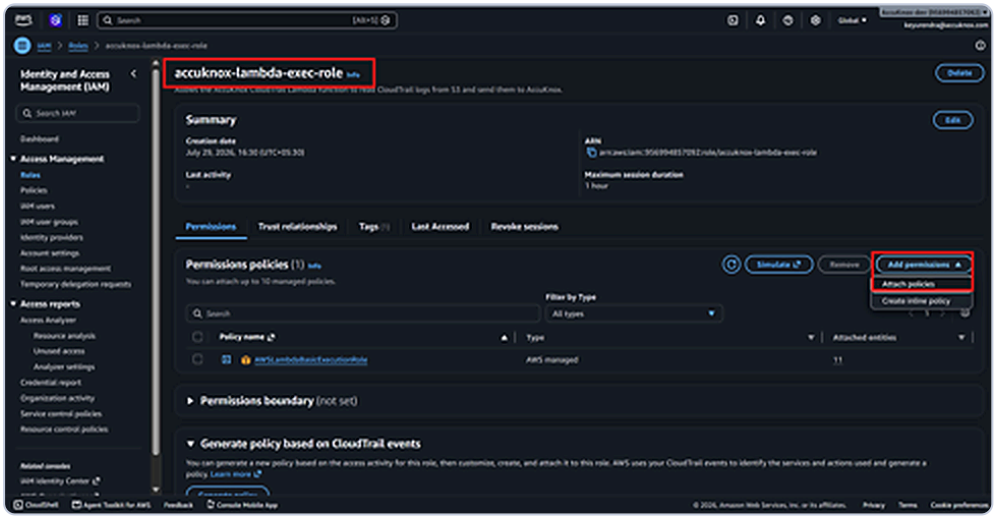


Figure 12. Step 2. Add the S3 Read Inline Policy

Select the JSON editor and replace the policy document with the following:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::accuknox-cloudtrail-sabpaisa/*"
    }
  ]
}
```

Click Next.

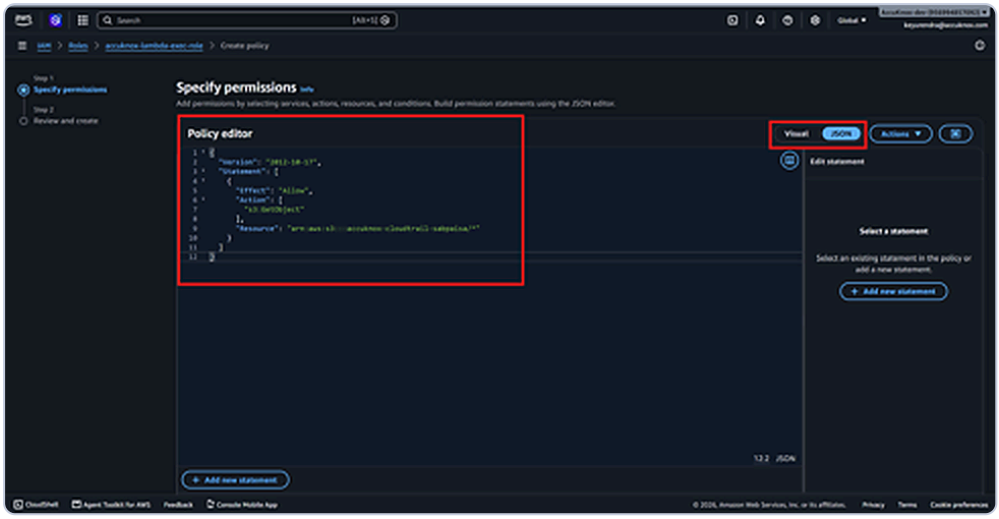


Figure 13. Step 2. Add the S3 Read Inline Policy

Configure the policy using the following values:

Setting	Value
Policy Name	`S3ReadPolicy`

Click Create Policy.

Figure 14. Step 2. Add the S3 Read Inline Policy

IAM Role Configuration

After creating the inline policy, the Lambda execution role should have the following permissions:

Policy	Type
`AWSLambdaBasicExecutionRole`	AWS Managed Policy
`S3ReadPolicy`	Inline Policy

The execution role is now configured to:

- Read CloudTrail log files from the Amazon S3 bucket.

- Write Lambda execution logs to Amazon CloudWatch Logs.

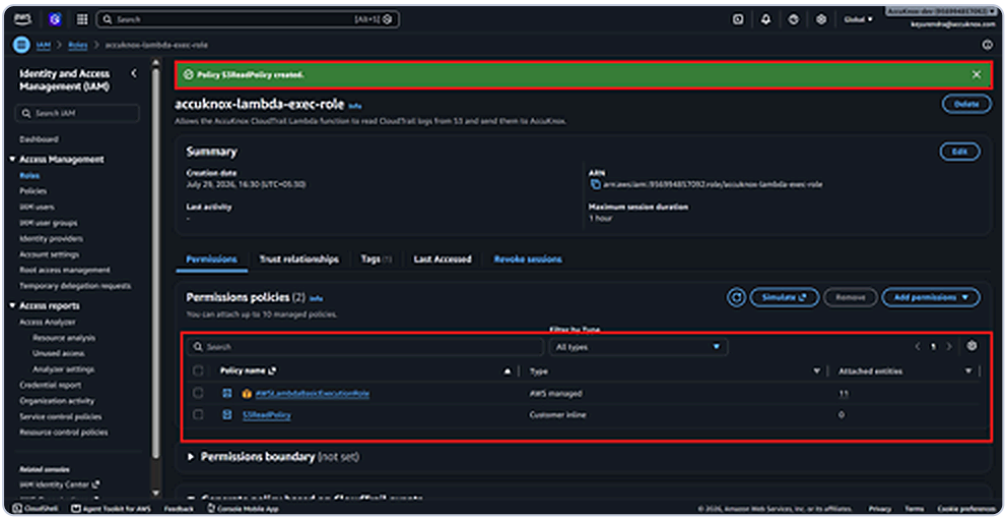


Figure 15. IAM Role Configuration

Step 3. Create the Lambda Function

Purpose

Create the AWS Lambda function responsible for processing CloudTrail log files uploaded to the Amazon S3 bucket and forwarding them to the AccuKnox SIEM/CDR ingestion endpoint.

Procedure

Navigate to:

AWS Console ▶ Lambda ▶ Create Function

Select Author from scratch.

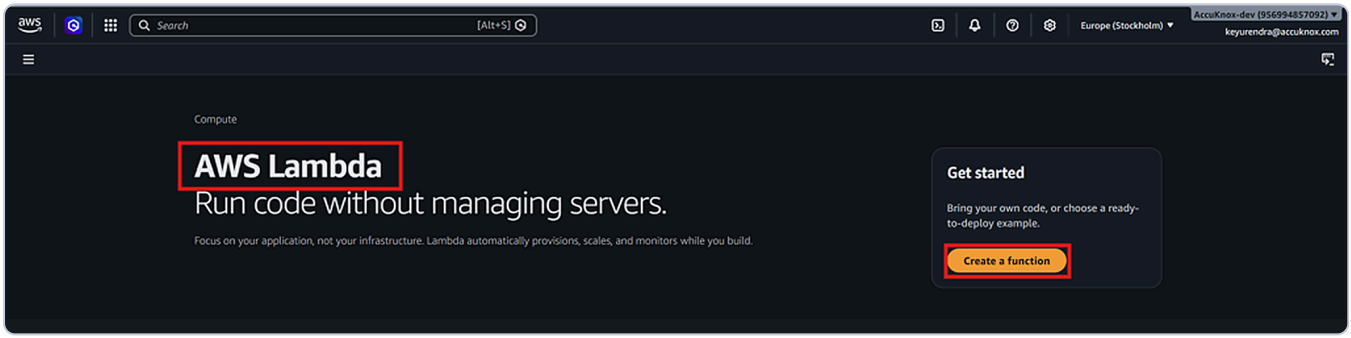


Figure 16. Step 3. Create the Lambda Function

Basic Information

Configure the Lambda function using the following settings:

Setting	Value
Function Name	`accuknox-cloudtrail-log-forwarder`
Runtime	Python 3.10

Note: The Terraform configuration specifies Python 3.9. However, Python 3.9 is no longer available for creating new Lambda functions in many AWS regions. Python 3.10 is the closest supported runtime and is fully compatible with this deployment.

Permissions

Expand Change default execution role and configure the following:

Setting	Value
Execution Role	Use an existing role
Existing Role	`accuknox-lambda-exec-role`

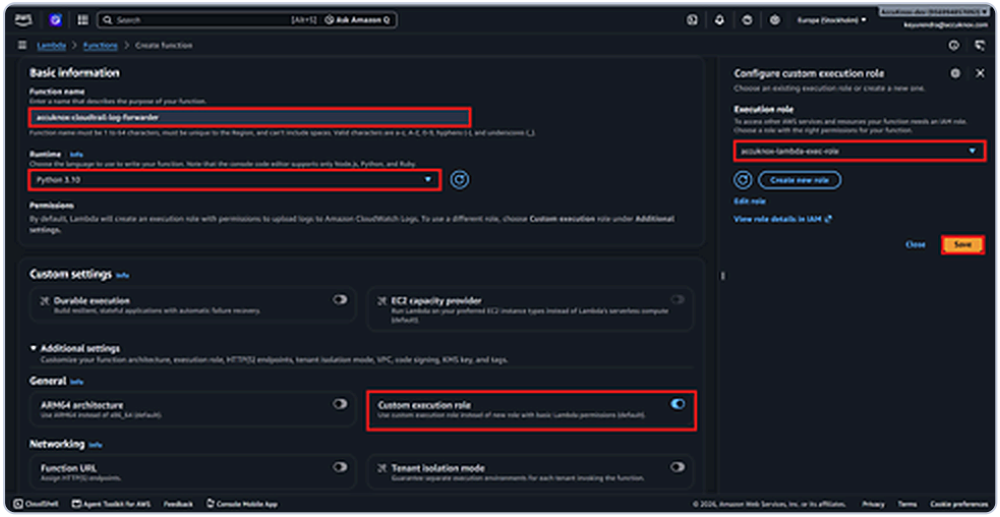


Figure 17. Permissions

The execution role should already include the following permissions:

Policy	Type
`AWSLambdaBasicExecutionRole`	AWS Managed Policy
`S3ReadPolicy`	Inline Policy

These permissions allow the Lambda function to:

- Read CloudTrail log files from the Amazon S3 bucket.
- Write execution logs to Amazon CloudWatch Logs.

Lambda Configuration Summary

Setting	Value
Function Name	`accuknox-cloudtrail-log-forwarder`
Runtime	Python 3.10
Architecture	`x86_64`
Execution Role	`accuknox-lambda-exec-role`
Function URL	Disabled
VPC	None
Code Signing	Disabled
Encryption	AWS Managed Key (Default)
Tags	Optional

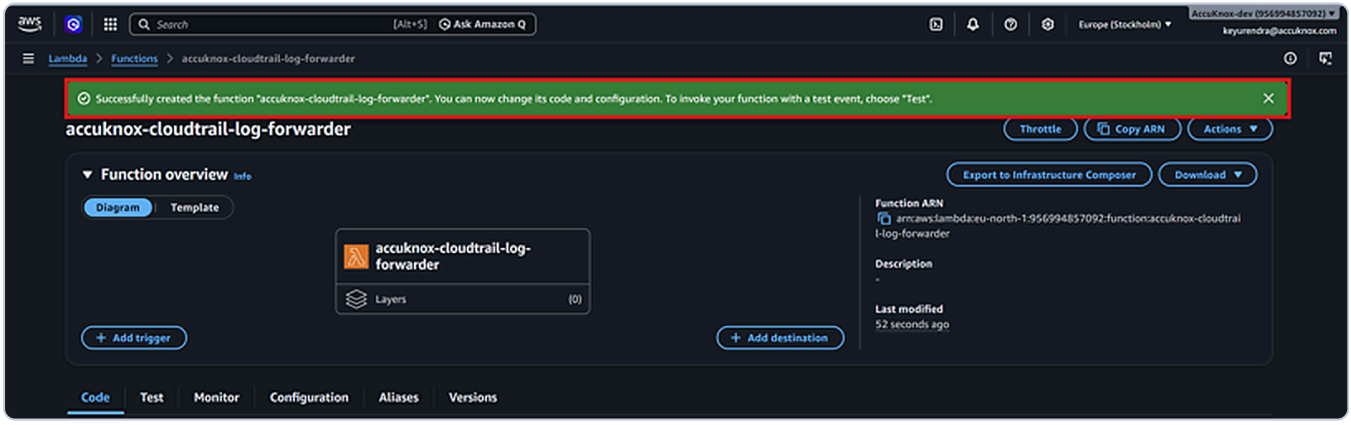


Figure 18. Lambda Configuration Summary

Step 4. Upload the Lambda Function Code

Purpose

Upload the Lambda function source code that processes CloudTrail log files, converts the events into NDJSON format, batches the records, and forwards them to the AccuKnox SIEM/CDR ingestion endpoint.

Procedure

Navigate to:

AWS Console ▶ Lambda ▶ accuknox-cloudtrail-log-forwarder

Open the Code tab.

Replace the default Lambda source code with the provided Python implementation.

ONLY IF NOT ALREADY CONFIGURED

Lambda Function Source Code (Python)

```
import boto3
import gzip
import json
import urllib.request
import logging
import os
import base64
import ssl

# Initialize logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

s3 = boto3.client('s3')


def get_env_config():
    return {
        "target_url": os.environ.get("TARGET_URL"),
        "chunk_size": int(os.environ.get("CHUNK_SIZE_LIMIT", 3 * 1024 * 1024)),
        "line_limit": int(os.environ.get("LINE_LIMIT", 1000)),
        "user": os.environ.get("BASIC_USER"),
        "password": os.environ.get("BASIC_PASS"),
        "insecure_tls": os.environ.get("INSECURE_TLS_SKIP", "false").lower() == "true"
    }


def handler(event, context):

    config = get_env_config()

    if not config["target_url"]:
        logger.error("TARGET_URL environment variable is missing.")
        return {"statusCode": 500}

    logger.info(f"Target URL: {config['target_url']}")
    logger.info(f"Chunk Size Limit: {config['chunk_size']} bytes")
    logger.info(f"Line Limit: {config['line_limit']}")

    ssl_context = None
    if config["insecure_tls"]:
        logger.warning("SSL verification is disabled.")
        ssl_context = ssl._create_unverified_context()

    for record in event.get("Records", []):

        bucket = record.get("s3", {}).get("bucket", {}).get("name")
        key = record.get("s3", {}).get("object", {}).get("key")

        if not bucket or not key:
            continue

        logger.info(f"Processing S3 Object: s3://{bucket}/{key}")
```

```
        try:

            response = s3.get_object(Bucket=bucket, Key=key)

            with gzip.GzipFile(fileobj=response["Body"]) as gf:
                content = gf.read()

            data = json.loads(content.decode("utf-8"))

            records_list = data.get("Records", [])

            logger.info(f"Loaded {len(records_list)} CloudTrail records")

            current_chunk = []
            current_size = 0
            batch_number = 1

            for item in records_list:

                item_str = json.dumps(item) + "\n"
                item_bytes = item_str.encode("utf-8")
                item_size = len(item_bytes)

                if (
                    current_size + item_size > config["chunk_size"]
                    or len(current_chunk) >= config["line_limit"]
                ):

                    if current_chunk:

                        logger.info(
                            f"Sending Batch #{batch_number}: "
                            f"{len(current_chunk)} records, "
                            f"{current_size} bytes"
                        )

                        send_data(
                            "".join(current_chunk),
                            config,
                            ssl_context
                        )

                        batch_number += 1
                        current_chunk = []
                        current_size = 0

                current_chunk.append(item_str)
                current_size += item_size

            if current_chunk:

                logger.info(
                    f"Sending Final Batch #{batch_number}: "
                    f"{len(current_chunk)} records, "
                    f"{current_size} bytes"
                )

                send_data(
                    "".join(current_chunk),
```

```
                    config,
                    ssl_context
                )

        except Exception as e:
            logger.error(f"Error processing {key}: {str(e)}")

    return {"statusCode": 200}


def send_data(payload, config, ssl_context):

    data_bytes = payload.encode("utf-8")

    logger.info(f"Posting payload size: {len(data_bytes)} bytes")

    headers = {
        "Content-Type": "application/x-ndjson"
    }

    if config["user"] and config["password"]:

        auth_str = f"{config['user']}:{config['password']}"

        encoded_auth = base64.b64encode(
            auth_str.encode("ascii")
        ).decode("ascii")

        headers["Authorization"] = f"Basic {encoded_auth}"

    req = urllib.request.Request(
        config["target_url"],
        data=data_bytes,
        headers=headers,
        method="POST"
    )

    try:

        with urllib.request.urlopen(
            req,
            timeout=15,
            context=ssl_context
        ) as response:

            logger.info(
                f"Successfully posted {len(data_bytes)} bytes. "
                f"Status: {response.getcode()}"
            )

    except Exception as e:

        logger.error(f"HTTP Post failed: {str(e)}")
        raise e
```

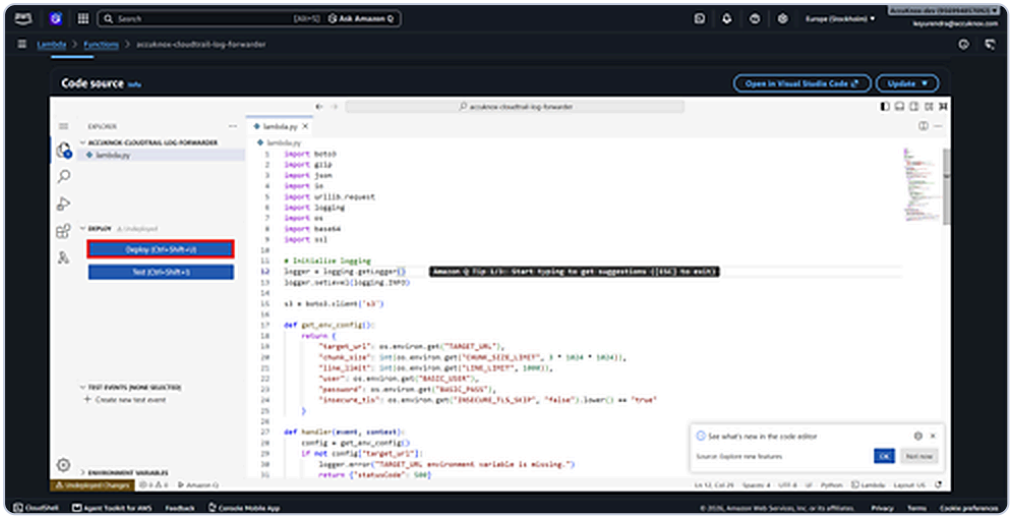


Figure 19. Step 4. Upload the Lambda Function Code

Configure the Lambda Handler

Navigate to:

Configuration ▶ Runtime Settings ▶ Edit

Configure the handler as follows:

Setting	Value
Handler	`lambda_function.handler`

Note: If the source file is renamed, update the handler accordingly. The handler format is `<filename>.<function_name>`. For example, if the file is named `lambda_function.py` and the function is `handler`, the value should be `lambda_function.handler`.

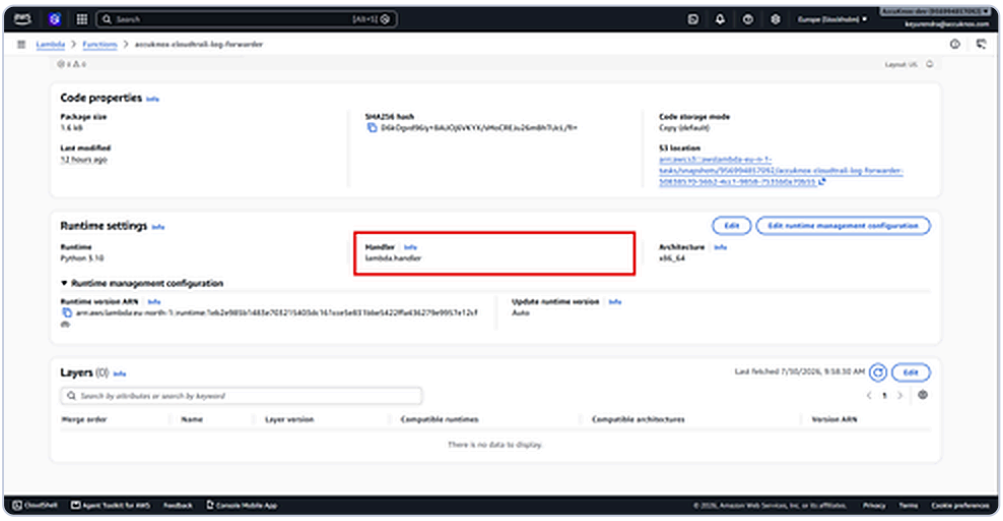


Figure 20. Configure the Lambda Handler

Step 5. Configure Lambda Environment Variables

Purpose

Configure the required environment variables used by the Lambda function to authenticate with and forward CloudTrail logs to the AccuKnox SIEM/CDR ingestion endpoint.

Procedure

Navigate to:

AWS Console ▶ Lambda ▶ accuknox-cloudtrail-log-forwarder ▶ Configuration ▶ Environment Variables ▶ Edit

Configure the following environment variables:

Environment Variable	Value
`TARGET_URL`	AccuKnox SIEM/CDR ingestion endpoint
`BASIC_USER`	Username used for Basic Authentication
`BASIC_PASS`	Password used for Basic Authentication
`INSECURE_TLS_SKIP`	`false`

> **Note:** Confirm that the `TARGET_URL`, `BASIC_USER`, and `BASIC_PASS` values are provided by the AccuKnox team before deployment.

After adding all the environment variables, click Save.

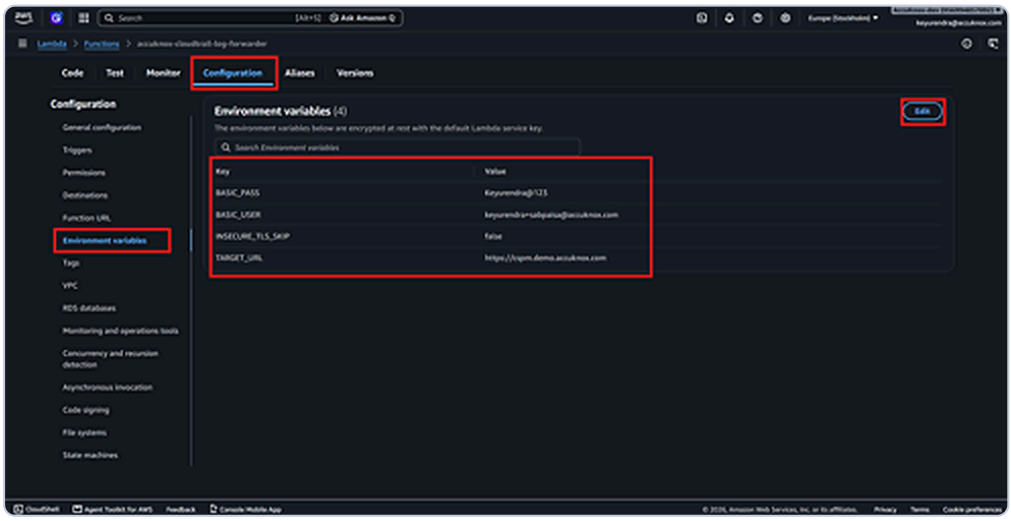


Figure 21. Step 5. Configure Lambda Environment Variables

Step 6. Configure the Amazon S3 Trigger

Purpose

Configure an Amazon S3 event notification to automatically invoke the Lambda function whenever AWS CloudTrail writes a new log file to the S3 bucket.

Procedure

Navigate to:

AWS Console ▶ Lambda ▶ accuknox-cloudtrail-log-forwarder ▶ Add Trigger

Trigger Configuration

Configure the trigger using the following settings:

Setting	Value
Trigger	Amazon S3
Bucket	`accuknox-cloudtrail-sabpaisa`
Event Type	All object create events
Prefix	Leave Empty
Suffix	Leave Empty
Recursive Invocation Acknowledgement	Enabled

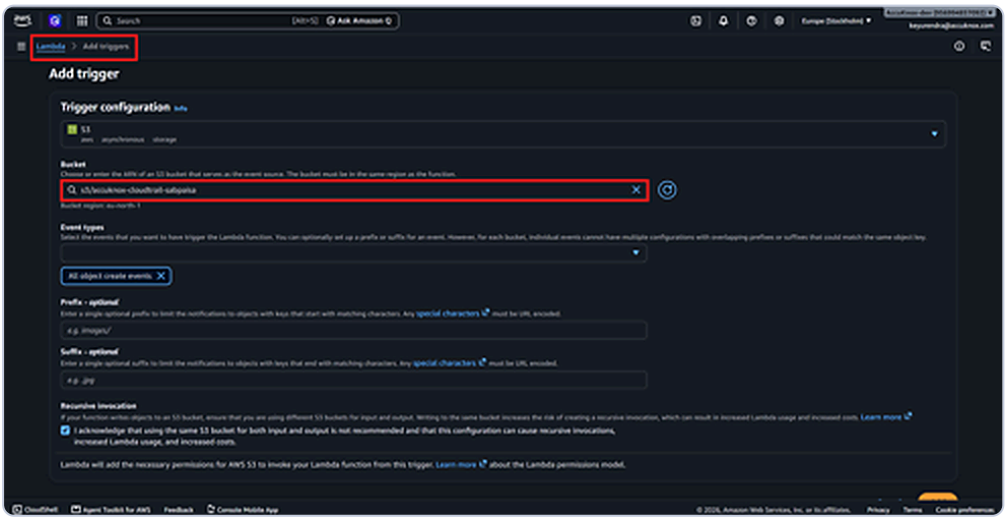


Figure 22. Trigger Configuration

After reviewing the configuration, click Add.

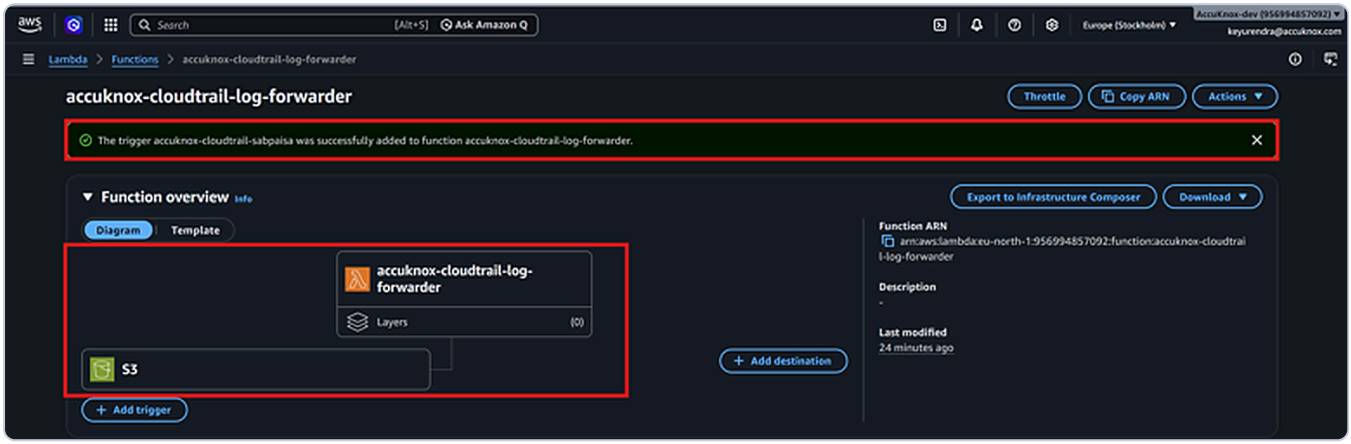


Figure 23. Trigger Configuration

Step 7. Validate the Lambda Invocation

Purpose

Verify that the Amazon S3 trigger is correctly invoking the Lambda function whenever a new AWS CloudTrail log file is uploaded.

Procedure

1. Navigate to:

   AWS Console
   → Lambda
   → accuknox-cloudtrail-log-forwarder
   → Monitor
   → View CloudWatch Logs

2. Open the latest Log Stream.

3. Verify that the following log sequence appears:

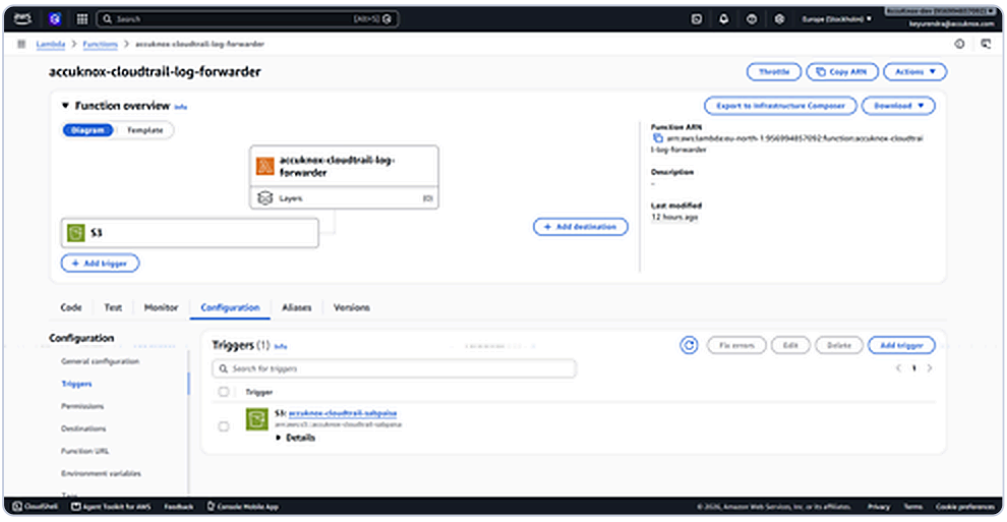


Figure 24. Step 7. Validate the Lambda Invocation

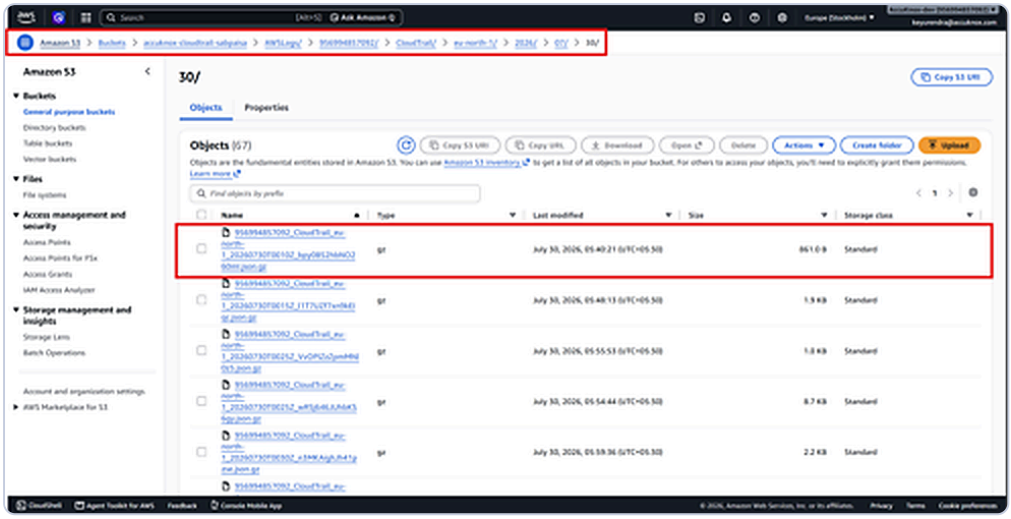


Figure 25. Step 7. Validate the Lambda Invocation

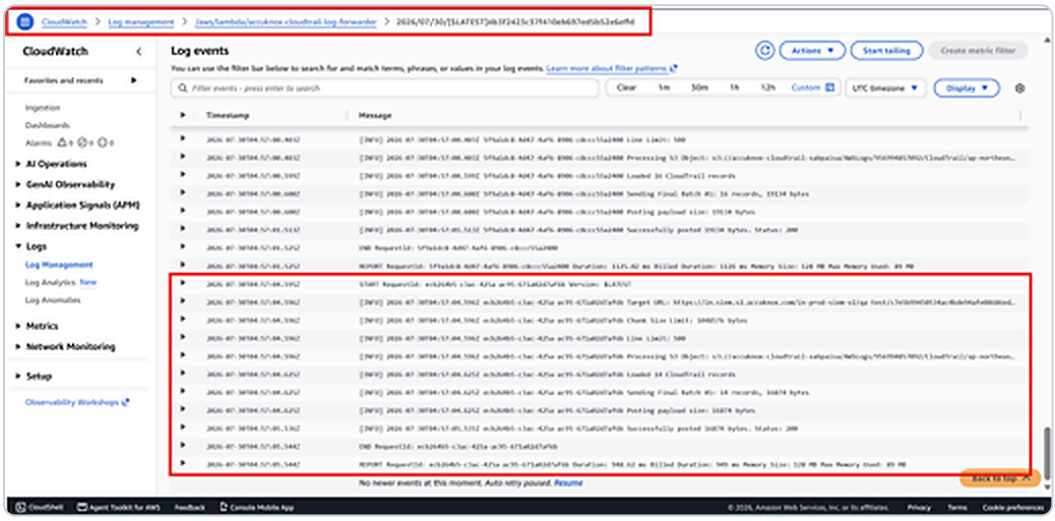


Figure 26. Step 7. Validate the Lambda Invocation

Step 8. Verify CloudTrail Logs in the AccuKnox SIEM UI

Purpose

Verify that the CloudTrail logs forwarded by the Lambda function are successfully ingested and are searchable in the AccuKnox SIEM dashboard.

Procedure

1. Log in to the AccuKnox SIEM Portal

Navigate to your AccuKnox SIEM instance.

Example:

```
https://in.siem.v2.accuknox.com
```

Log in using your credentials.

2. Open the Logs Page

Navigate to:

SIEM ▶ Logs

3. Select the Appropriate Time Range

Choose a recent time range that includes the Lambda execution, for example:

- Last 15 Minutes
- Last 30 Minutes
- Last 1 Hour

4. Search for CloudTrail Logs

In the search bar, apply the following filter:

```
cloud = aws
```

This filters the logs to display AWS CloudTrail events that have been ingested into the SIEM.

5. Verify the Logs

If the integration is successful, AWS CloudTrail logs will be displayed in the **Logs** section.

Verify that the logs contain CloudTrail event details such as:

- Event Time
- AWS Account ID

- AWS Region
- Event Name
- Event Source
- User Identity
- Source IP Address

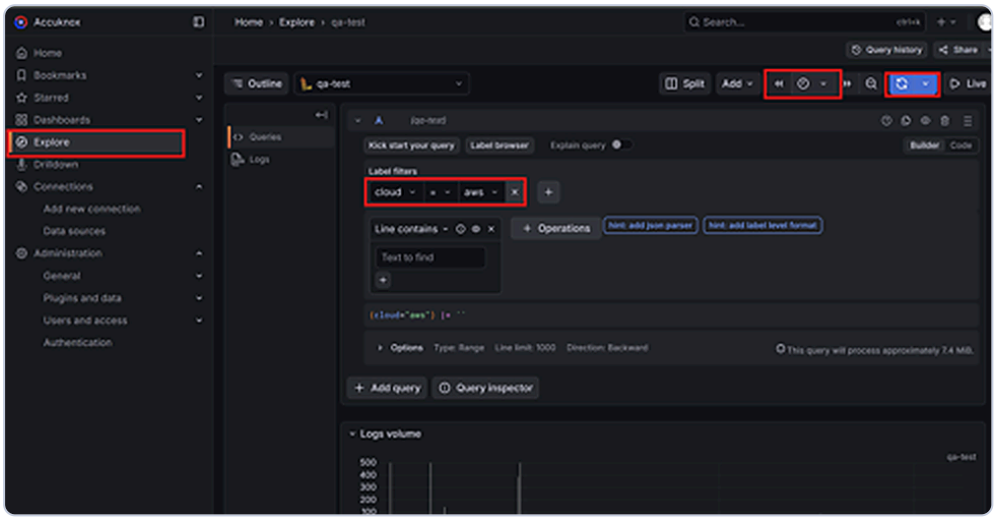


Figure 27. 5. Verify the Logs

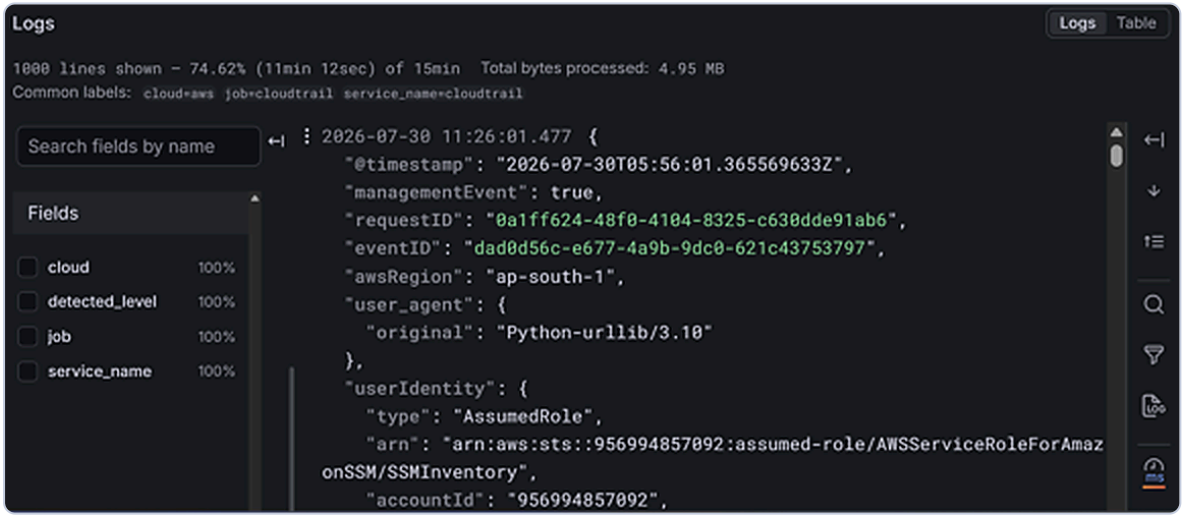


Figure 28. 5. Verify the Logs

Step 9. Configure GitHub Repository for CDR Remediation

Purpose

Configure a GitHub repository that receives remediation requests from AccuKnox and executes the required remediation workflow using GitHub Actions.

Procedure

1. Clone the CDR Demo Repository

```
git clone https://github.com/Keyurendra/cdr-remediation-demo.git
```

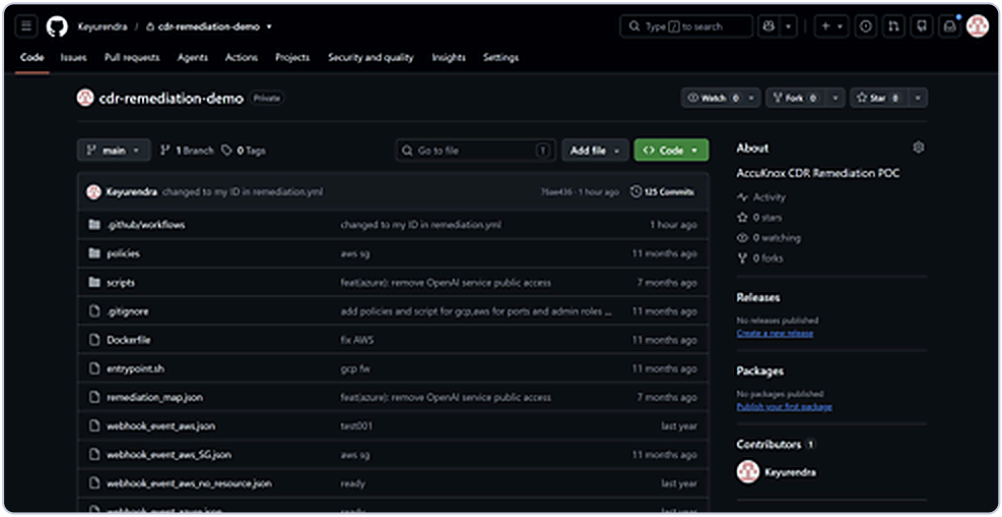


Figure 29. 1. Clone the CDR Demo Repository

This repository contains:

- GitHub Actions workflow
- Webhook integration
- Demo remediation workflow

2. Clone the CDR Remediation Action Repository

```
git clone https://github.com/Keyurendra/cdr-remediation.git
```

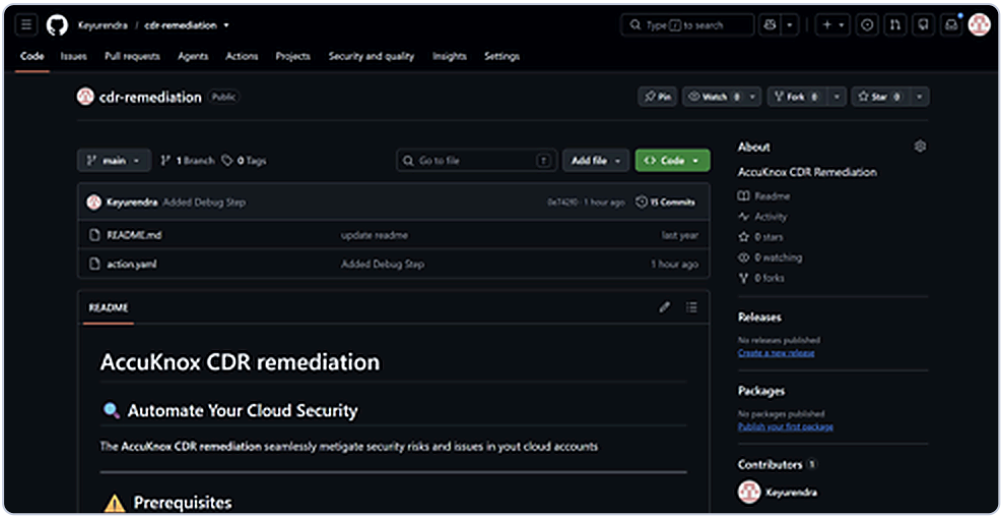


Figure 30. 2. Clone the CDR Remediation Action Repository

This repository contains:

- action.yml
- GitHub Composite Action
- Cloud authentication logic
- Entry point execution

3. Create GitHub Secrets

Navigate to:

Repository ▶ Settings ▶ Secrets and variables ▶ Actions ▶ New repository secret

Create the following secrets:

Secret Name	Description
AWS_DEV_ACCESS_ID	AWS Access Key ID
AWS_DEV_SECRET_ID	AWS Secret Access Key

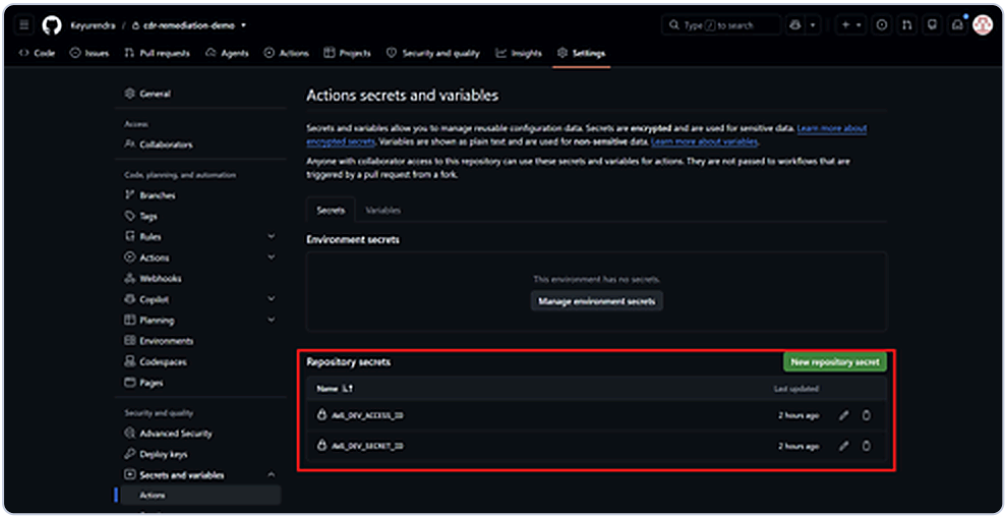


Figure 31. 3. Create GitHub Secrets

4. Changes in the workflow file

Navigate to

```
.github/workflows/remediation.yml
```

Update the workflow by retaining only the credential secrets for the target cloud provider:

```
name: CDR Remediation

on:
  repository_dispatch:
    types: [accuknox-webhook]

jobs:
  remediate:
    runs-on: ubuntu-latest

    container:
      image: public.ecr.aws/k9v9d5v2/cdr/accknox-cdr-remediation:latest
      env:
        CLIENT_PAYLOAD: ${{ toJSON(github.event.client_payload) }}

    steps:
      - name: AccuKnox CDR Action
        uses: Keyurendra/cdr-remediation@main
        with:
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_DEV_ACCESS_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_DEV_SECRET_ID }}
```

Explanation

- Invokes the custom GitHub Action from the `Keyurendra/cdr-remediation` repository.
- Authenticates to AWS using the repository secrets:
  - `AWS_DEV_ACCESS_ID`

- `AWS_DEV_SECRET_ID`

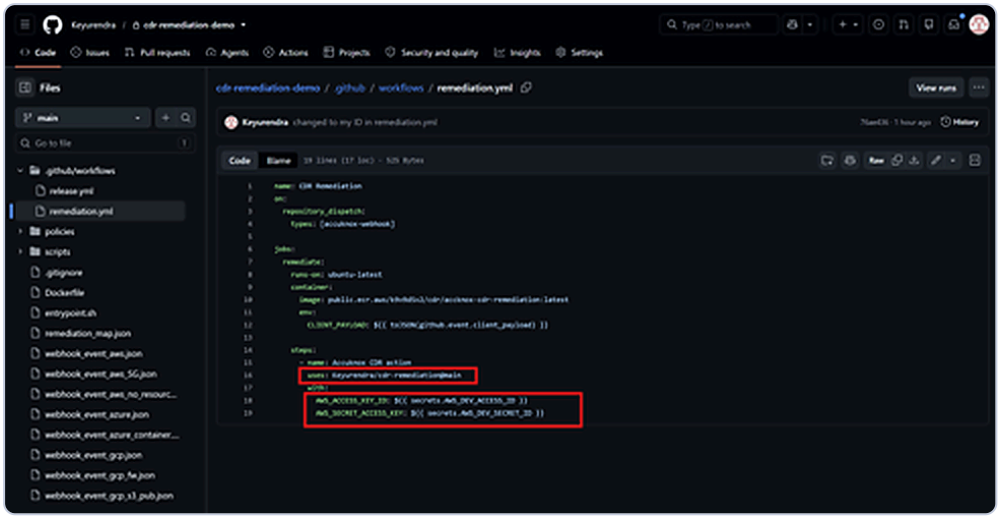


Figure 32. Navigate to

Step 10 - Verify Alerts in the AccuKnox Platform

Purpose

Verify that the ingested AWS CloudTrail logs generate the corresponding CDR alerts in the AccuKnox Platform before configuring automated remediation.

Procedure

1. Log in to the AccuKnox Platform.
2. Navigate to:

Alerts ▶ All Alerts

3. Configure the following filters:

Filter	Value
Module	CDR
Tools	Select the target cloud provider (e.g., AWS)

4. Apply additional filters to locate specific alerts more easily.

Filter	Example Value
Rule	Select the rule corresponding to the use case being validated.
Region	`eu-north-1`

5. Verify that the expected CDR alert is displayed.

Result

The corresponding CDR alert is successfully generated and is available for configuring automated remediation through Triggers and Webhook Integrations.

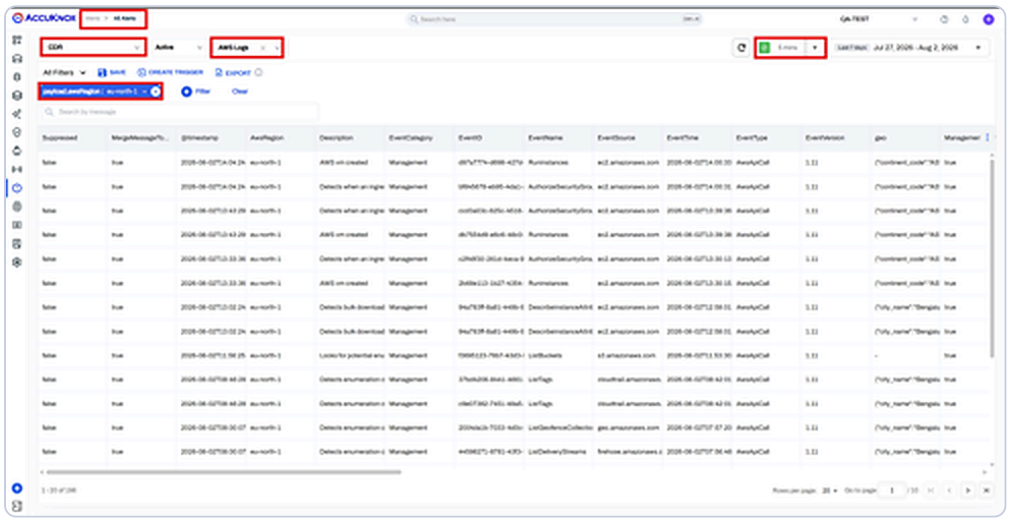


Figure 33. Step 10 - Verify Alerts in the AccuKnox Platform

Step 11. Configure GitHub Webhook Integration

Purpose

Configure an AccuKnox Webhook Integration to invoke the GitHub Repository Dispatch API whenever a remediation event is generated.

Procedure

Navigate to:

Settings ▶ Integrations ▶ Webhook ▶ Connect

Configure the following:

Field	Value
Integration Name	GitHub Webhook
Method	POST
Webhook URL	`https://api.github.com/repos/Keyurendra/cdr-remediation-demo/dispatches`
Success Codes	200, 201, 202, 204

Headers

Header	Value
Authorization	`token <GitHub Personal Access Token>`
Accept	`application/vnd.github+json`

Test Payload

```
{
  "event_type": "accuknox-webhook"
}
```

Note: The Test Payload is only used to validate the webhook connection. The actual payload is sent by the Trigger during runtime.

Click Save to create the Webhook Integration.

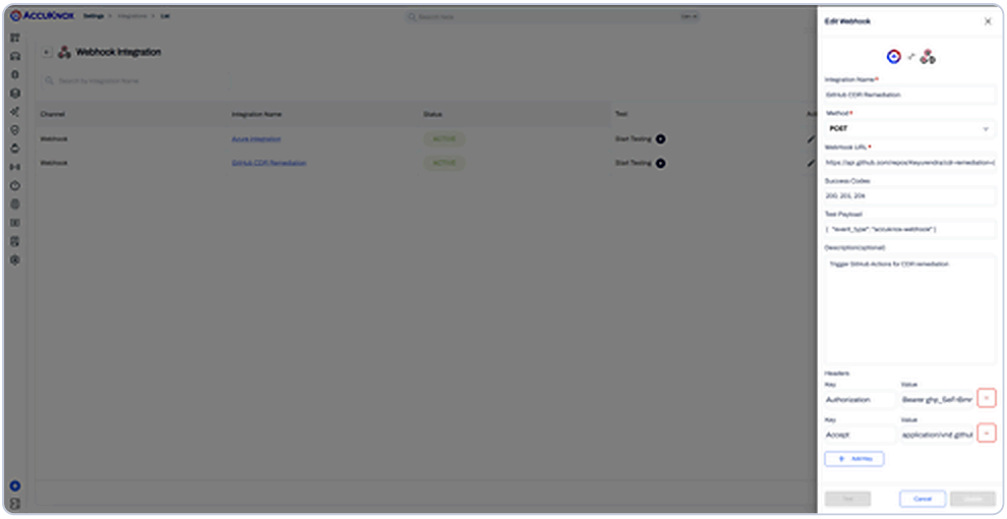


Figure 34. Test Payload

Step 12. AWS Out-of-the-Box Detection & Remediation Rules

Once the AWS CDR deployment is completed, AccuKnox continuously monitors AWS CloudTrail events and applies pre-built detection and remediation policies to help identify and automatically remediate common security misconfigurations across AWS resources.

Detection Rule	Description	Automated Response
AWS S3 Made Public	Detects when an Amazon S3 bucket is configured to allow public access.	Automatically enables all S3 Block Public Access settings to prevent public access to the bucket.
AWS Security Group Port Opened	Detects when sensitive ports such as SSH (22) and RDP (3389) are exposed to the internet through a security group.	Automatically removes the security group ingress rules that allow unrestricted access to ports 22 and 3389.
AWS S3 Bucket Versioning Disabled	Detects when versioning is disabled or not configured for an Amazon S3 bucket.	Automatically enables bucket versioning to improve data protection and recovery capabilities.
AWS CloudTrail Logging Disabled	Detects when AWS CloudTrail logging is disabled for a trail.	Automatically re-enables CloudTrail logging so audit logging of AWS API activity keeps running.
AWS Config Recorder Retention Misconfigured	Detects when the AWS Config recorder retention period does not meet the configured policy requirements.	Automatically configures the AWS Config retention period to 30 days.
AWS EC2 Instance with Public IP	Detects EC2 instances that are launched with a public IP address or exposed to the public internet.	Automatically terminates the exposed EC2 instance (or applies the configured remediation policy) to eliminate unintended public exposure.
AWS Default EBS Encryption Disabled	Detects when default encryption for Amazon EBS volumes is disabled at the AWS account level.	Automatically enables default EBS encryption using the AWS-managed KMS key.

Key Benefits

- Pre-built AWS security detection and remediation policies
- Automatic remediation of common cloud misconfigurations
- Continuous monitoring of AWS CloudTrail events
- Reduced manual effort through policy-driven enforcement
- Helps enforce AWS security best practices and compliance requirements
- Supports proactive cloud security and continuous risk reduction

Note: These out-of-the-box policies are included with the AWS CDR deployment. Additional custom detection and remediation policies can be configured to meet organization-specific security and compliance requirements.

Use Case 1. Automatic Remediation of Public Amazon S3 Buckets

Objective

Automatically detect publicly accessible Amazon S3 buckets and remediate them by enabling Block Public Access, preventing unauthorized access to bucket objects.

Test Scenario

An Amazon S3 bucket named `sabpaisa-test-bucket` was created in the eu-north-1 region, and a sample image was uploaded for testing.

To simulate a security issue:

- Disabled Block Public Access
- Applied a bucket policy allowing public read access
- Verified that the uploaded image was accessible using its S3 Object URL

Result

The object was publicly accessible.

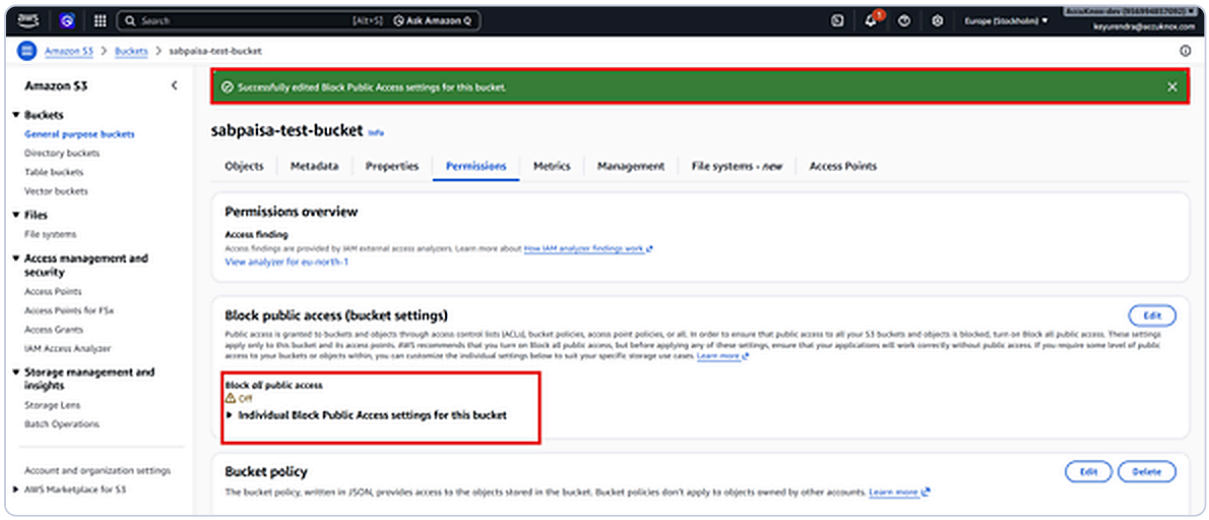


Figure 35. Test Scenario

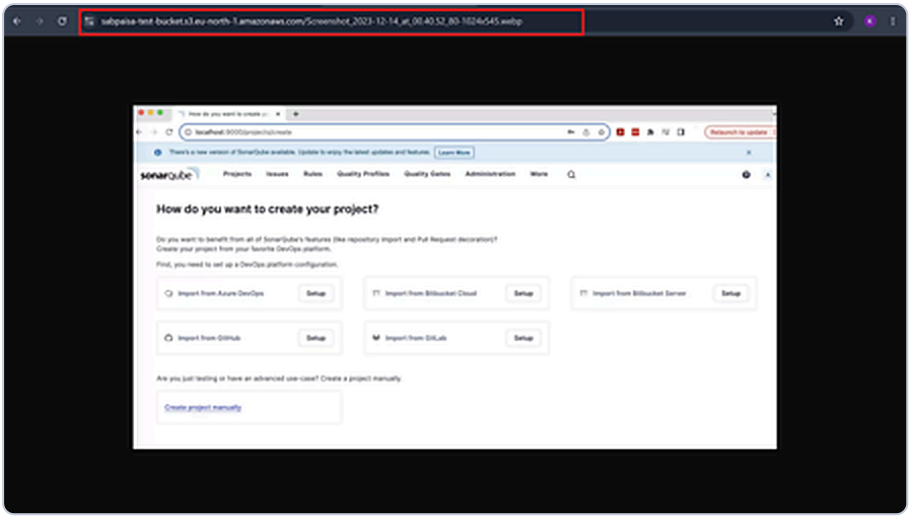


Figure 36. Test Scenario

Alert Detection in AccuKnox

After the Amazon S3 bucket was made publicly accessible, AccuKnox detected the security event and generated an alert under the CDR module.

The alerts were filtered using:

- **Module:** CDR
- **Log Source:** AWS Logs
- **Filter:** payload.rule = AWS S3 made public

The generated alert confirms that AccuKnox successfully identified the security event, which can then be used to configure an automated remediation trigger.

Result

The AWS S3 made public alert was successfully generated in AccuKnox.

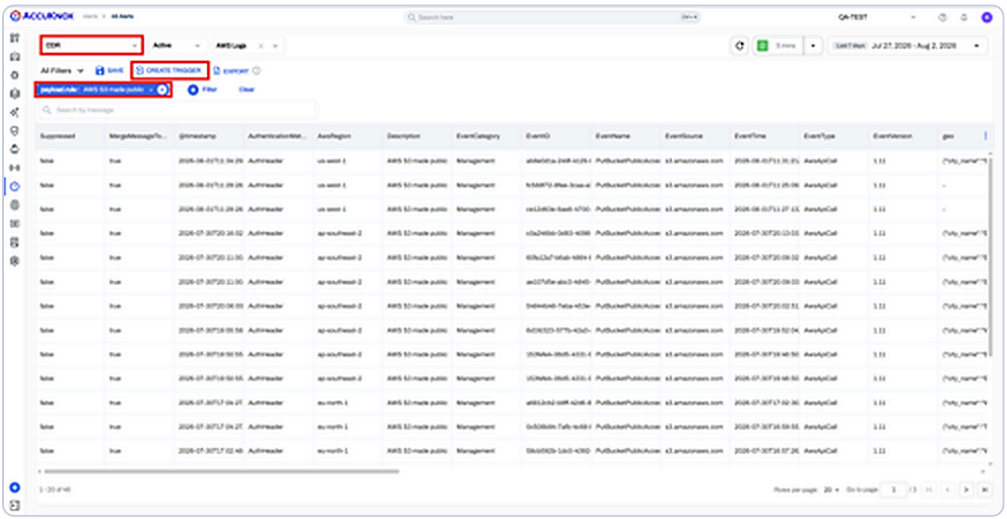


Figure 37. Alert Detection in AccuKnox

Trigger Configuration

A trigger was created in AccuKnox to automatically invoke the GitHub Webhook Integration whenever an Amazon S3 bucket is detected as publicly accessible.

Configuration

- **Name:** AWS S3 Public Bucket Remediation
- **Description:** Automatically triggers GitHub Actions to remediate publicly accessible Amazon S3 buckets.
- **Filter:** payload.rule
  Value: AWS S3 made public
- **Action:** Notify
- **Notification Channel:** GitHub CDR Remediation

Result

Whenever the AWS S3 made public alert is generated, the trigger automatically invokes the configured webhook, initiating the remediation workflow.

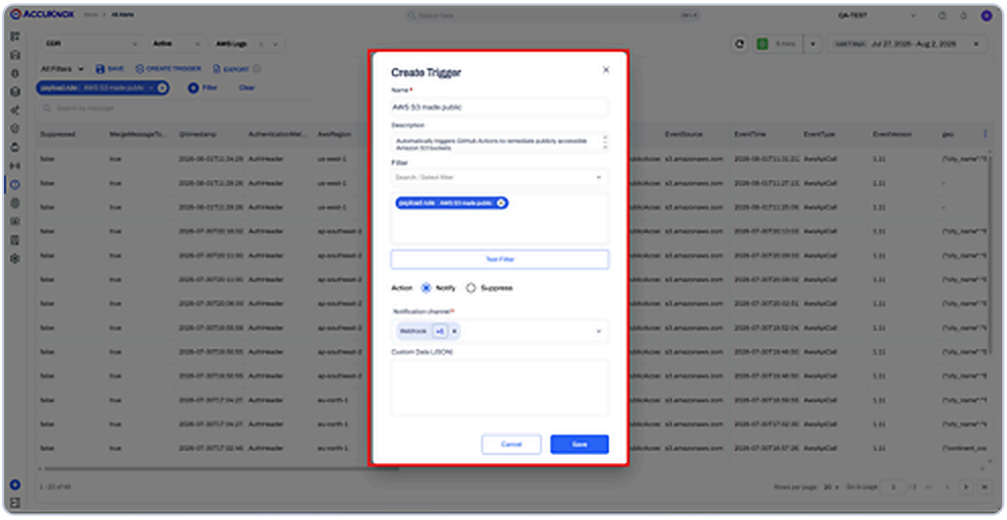


Figure 38. Trigger Configuration

GitHub Workflow Execution

The webhook sent a Repository Dispatch request to GitHub.

The GitHub Actions workflow started automatically and executed the configured remediation workflow.

During execution:

- **Rule identified:** AWS S3 made public
- **Resource identified:** sabpaisa-test-bucket
- **Cloud Provider:** AWS
- **Remediation Tool:** Cloud Custodian
- **Remediation Policy:** s3_make_bucket_private.yaml

Result

GitHub Actions completed successfully.

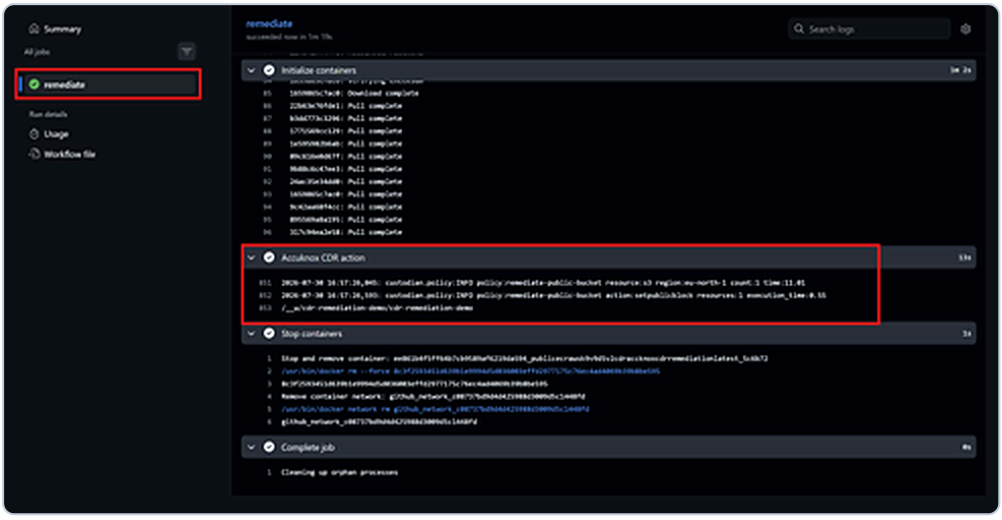


Figure 39. GitHub Workflow Execution

Automatic Remediation

Cloud Custodian executed the setpublicblock action on the affected S3 bucket.

The following S3 security settings were automatically enabled:

- Block Public ACLs
- Ignore Public ACLs
- Block Public Bucket Policies
- Restrict Public Buckets

Result

S3 bucket configuration was updated automatically.

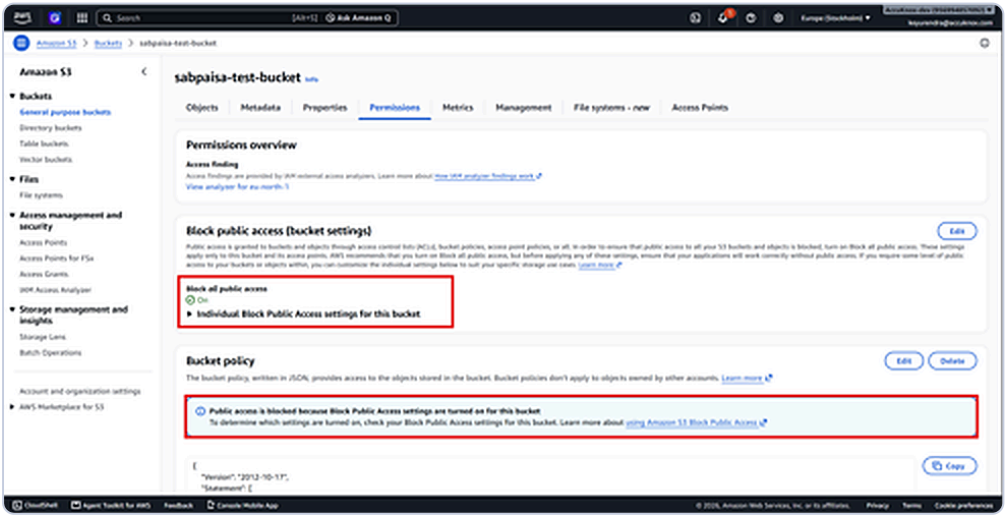


Figure 40. Automatic Remediation

Validation

The same S3 Object URL that was previously accessible was opened again after remediation.

Before Remediation

- Image loaded successfully.

After Remediation

- Access Denied (403 Forbidden)

This confirms that public access to the bucket objects was successfully blocked.

Figure 41. Validation

Use Case 2. Automatic Remediation of Publicly Exposed AWS Security Group Ports

Objective

Automatically detect AWS Security Groups that expose inbound ports to the public Internet (`0.0.0.0/0`) and remediate them by removing the insecure inbound rule, preventing unauthorized network access to the EC2 instance.

Test Scenario

An Amazon EC2 instance named public-ec2-test was launched in the eu-north-1 region using a Security Group configured with an inbound SSH rule allowing access from 0.0.0.0/0.

To simulate a security issue:

- Launched an EC2 instance with a Public IP address.
- Configured the Security Group to allow SSH (TCP Port 22) from 0.0.0.0/0.
- Connected to the EC2 instance using EC2 Instance Connect.
- Verified that the Security Group contained the public inbound rule.

Result

The EC2 instance was publicly accessible through the Security Group.

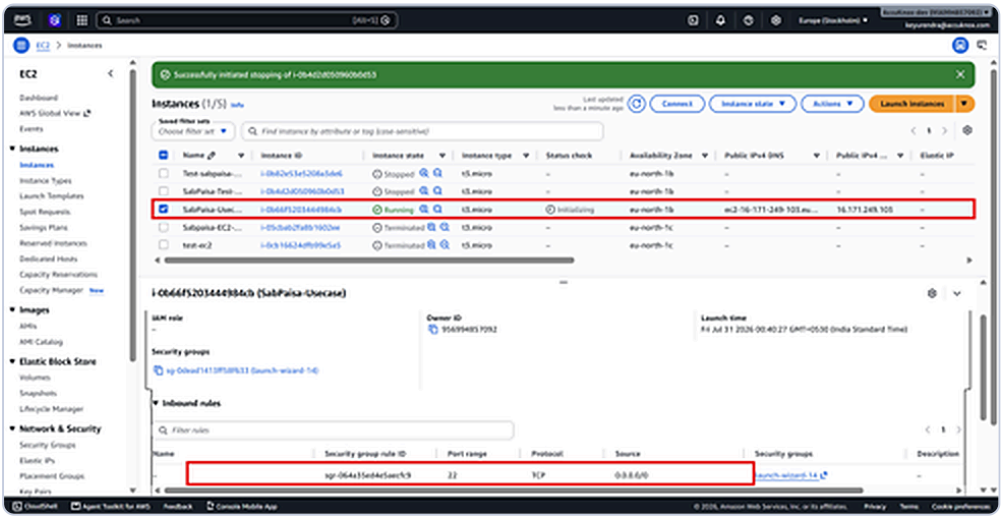


Figure 42. Test Scenario

Alert Detection in AccuKnox

After the AWS Security Group was configured to allow inbound access from 0.0.0.0/0, AccuKnox detected the security event and generated an alert under the CDR module.

The alerts were filtered using:

- **Module:** CDR

- **Log Source:** AWS Logs
- **Filter:** payload.rule = AWS Security Group Port Opened

The generated alert confirms that AccuKnox successfully identified the publicly exposed Security Group, which can then be used to configure an automated remediation trigger.

Result

The AWS Security Group Port Opened alert was successfully generated in AccuKnox.

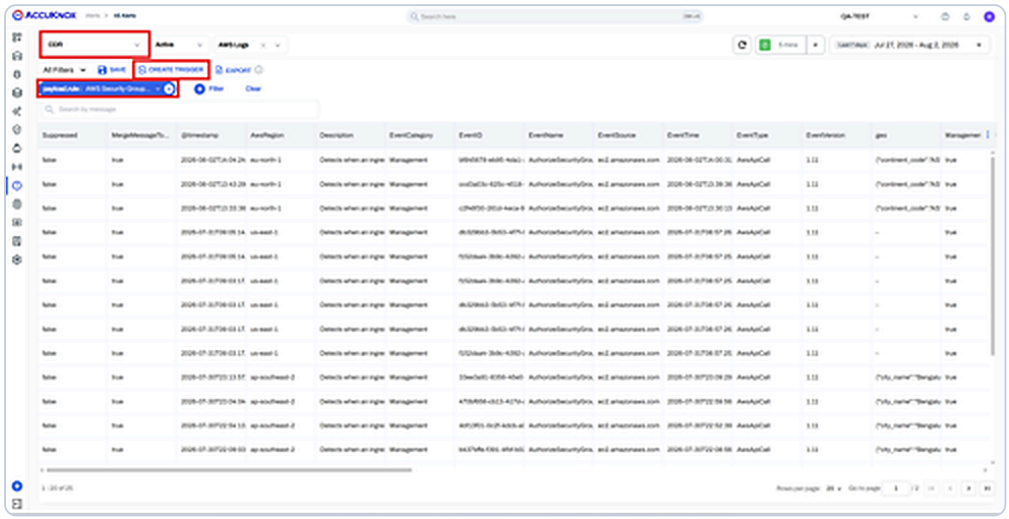


Figure 43. Alert Detection in AccuKnox

Trigger Configuration

A trigger was created in AccuKnox to automatically invoke the GitHub Webhook Integration whenever an AWS Security Group is detected with a publicly exposed inbound port.

Configuration

- **Name:** AWS Security Group Public Port Remediation
- **Description:** Automatically triggers GitHub Actions to remediate publicly exposed AWS Security Group ports.
- **Filter:** payload.rule
- **Value:** AWS Security Group Port Opened
- **Action:** Notify
- **Notification Channel:** GitHub CDR Webhook

Result

Whenever the AWS Security Group Port Opened alert is generated, the configured webhook is automatically invoked, initiating the remediation workflow.

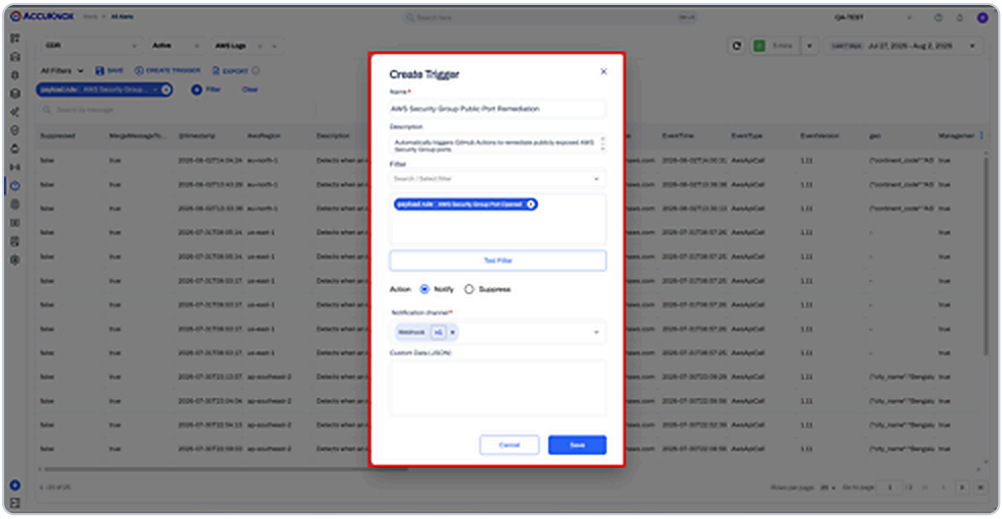


Figure 44. Trigger Configuration

GitHub Workflow Execution

The webhook sent a Repository Dispatch request to GitHub.

The GitHub Actions workflow started automatically and executed the configured remediation workflow.

During execution:

- **Rule identified:** AWS Security Group Port Opened
- **Cloud Provider:** AWS
- **Remediation Tool:** Cloud Custodian
- **Remediation Policy:** Security Group Port Remediation Policy
- **Target Resource:** Affected AWS Security Group

Result

GitHub Actions completed successfully.

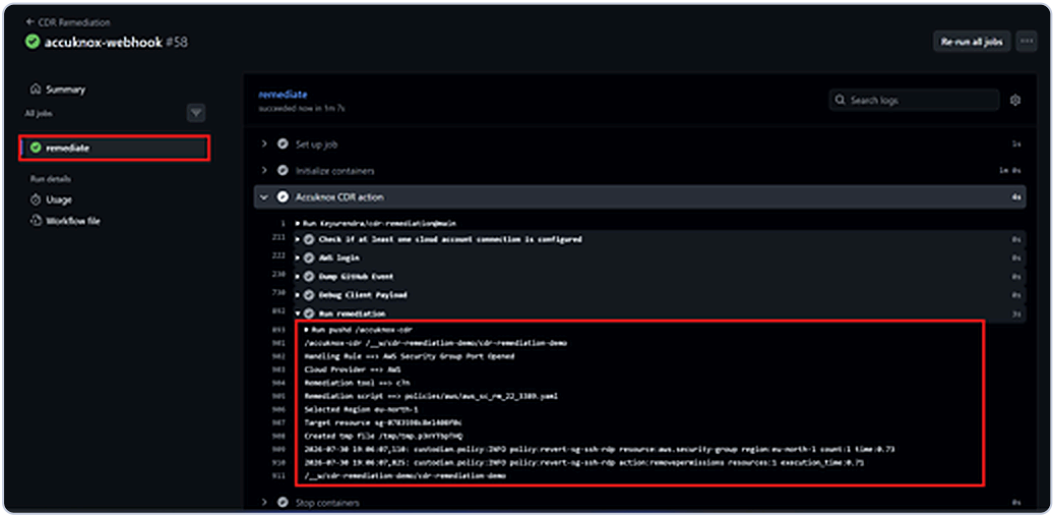


Figure 45. GitHub Workflow Execution

Automatic Remediation

Cloud Custodian automatically revoked the insecure inbound Security Group rule that allowed public access.

The following security change was performed automatically:

- Removed the inbound rule allowing TCP Port 22 from 0.0.0.0/0.

Result

The AWS Security Group configuration was updated automatically.

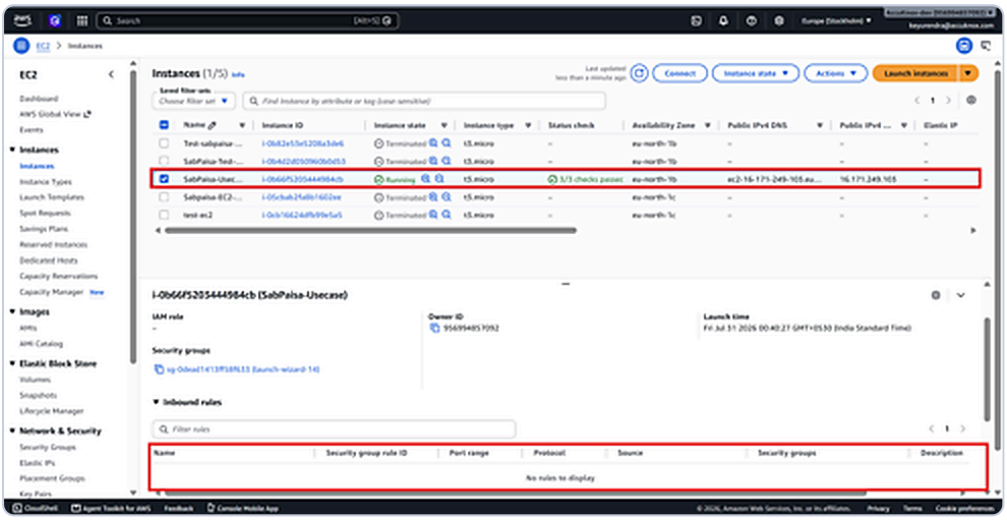


Figure 46. Automatic Remediation

Validation

The Security Group configuration was verified after remediation.

Before Remediation

- Security Group contained an inbound SSH rule (TCP Port 22) with source 0.0.0.0/0.
- The EC2 instance was publicly accessible.

After Remediation

- The inbound Security Group rule allowing 0.0.0.0/0 was automatically revoked.
- Public network access to the EC2 instance through the exposed SSH port was blocked.

Use Case 3. Automatic Remediation of AWS EC2 Instances with Public IP Addresses

Objective

Automatically detect Amazon EC2 instances launched with a public IP address and remediate them by removing the public IP while keeping the instance running.

Test Scenario

An Amazon EC2 instance was launched with Auto-assign Public IP enabled.

To simulate the security issue:

- Created a new EC2 instance with a public IPv4 address.
- Verified that the instance was in the Running state.
- Confirmed that a public IP address was assigned to the instance.

Result

The EC2 instance was running with a publicly accessible IP address.

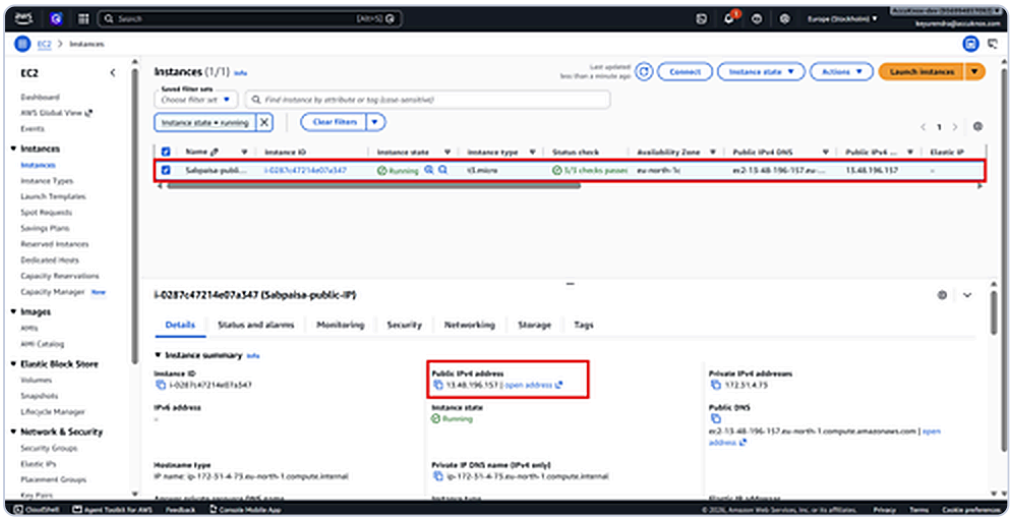


Figure 47. Test Scenario

Alert Detection in AccuKnox

After the EC2 instance was launched with a public IP address, AccuKnox detected the security event and generated an alert under the CDR module.

The alerts were filtered using:

- **Module:** CDR
- **Log Source:** AWS Logs
- **Filter:** payload.rule = AWS vm created

The generated alert confirms that AccuKnox successfully identified the EC2 instance with a public IP, which can then be used to configure an automated remediation trigger.

Result

The AWS vm created alert was successfully generated in AccuKnox.

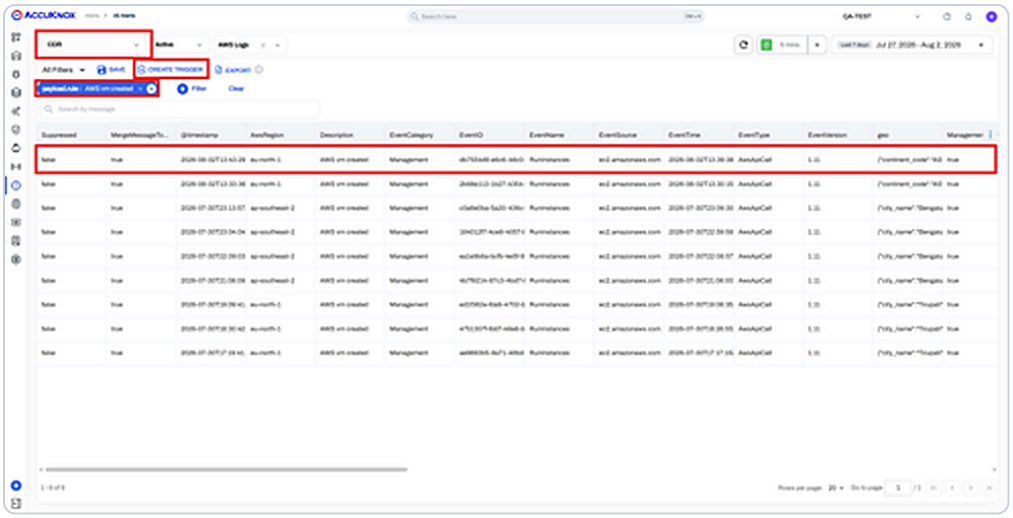


Figure 48. Alert Detection in AccuKnox

Trigger Configuration

A trigger was created in AccuKnox to automatically invoke the GitHub Webhook Integration whenever an Amazon S3 bucket is detected as publicly accessible.

Configuration:

- **Name:** AWS S3 Public Bucket Remediation
- **Description:** Automatically triggers GitHub Actions to remediate publicly accessible Amazon S3 buckets.
- **Filter:** payload.rule
- **Value:** AWS S3 made public
- **Action:** Notify
- **Notification Channel:** Webhook

Result

Whenever the AWS S3 made public alert is generated, the trigger automatically invokes the configured webhook, initiating the remediation workflow.

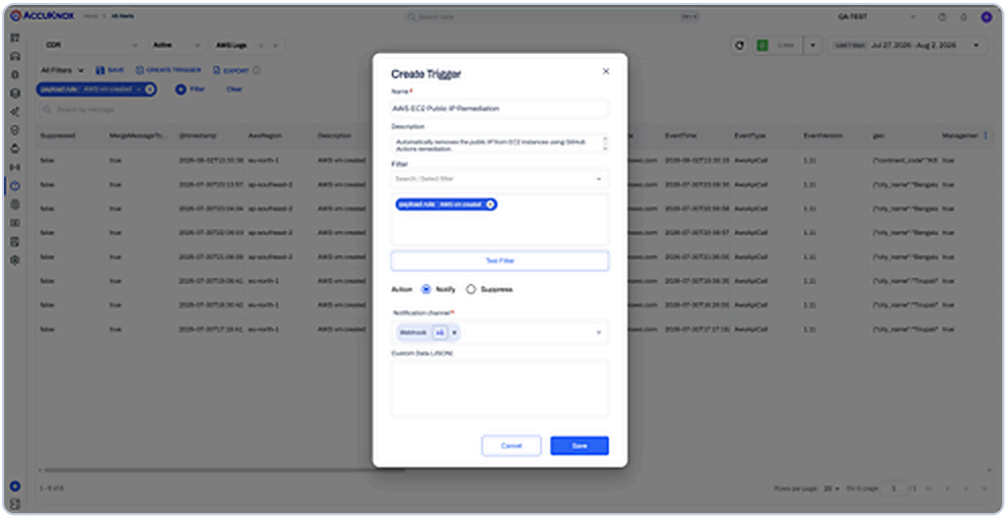


Figure 49. Trigger Configuration

GitHub Workflow Execution

The webhook sent a Repository Dispatch request to GitHub.

The GitHub Actions workflow started automatically and executed the configured remediation workflow.

During execution:

- **Rule identified:** AWS vm created
- **Resource identified:** *EC2 Instance ID*
- **Cloud Provider:** AWS
- **Remediation Script:** remove_public_ip.sh

Result

GitHub Actions completed successfully.

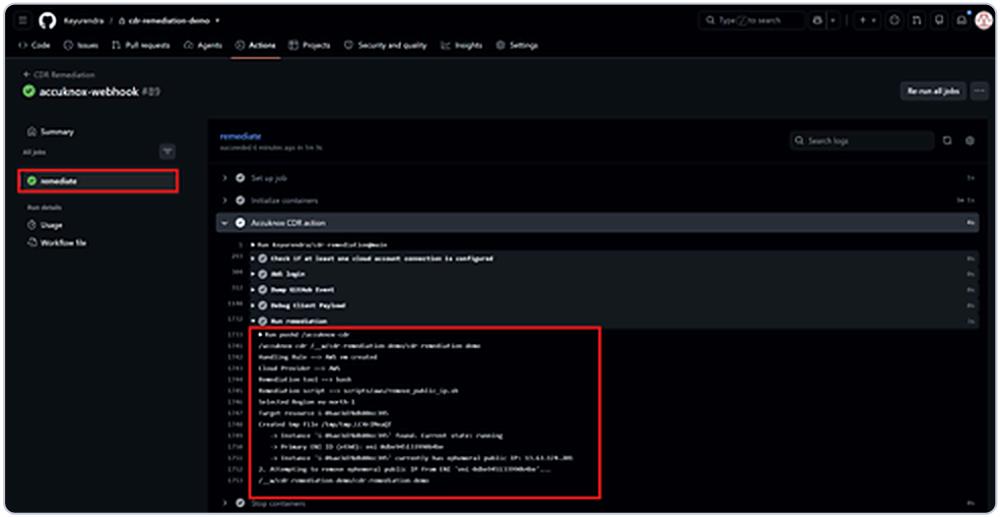


Figure 50. GitHub Workflow Execution

Automatic Remediation

The remediation script executed the required AWS CLI commands to remove the public IP association from the EC2 instance's primary network interface.

The remediation:

- Identified the primary network interface (ENI).
- Removed the associated public IP address.
- Kept the EC2 instance in the Running state without interruption.

Result

The EC2 instance was automatically remediated by removing its public IP address.

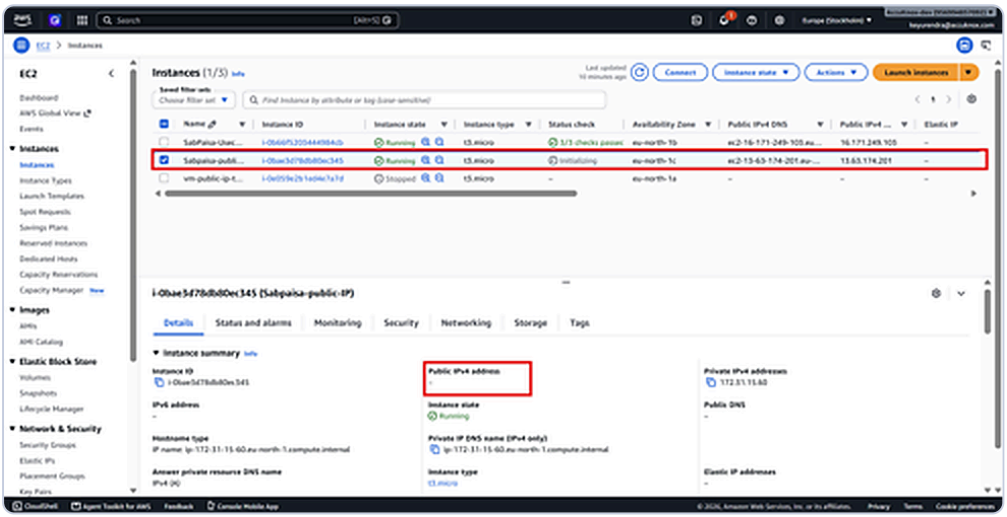


Figure 51. Automatic Remediation

Validation

The EC2 instance details were verified after remediation.

Before Remediation

- Instance State: Running
- Public IPv4 Address: Assigned

After Remediation

- Instance State: Running
- Public IPv4 Address: Not Assigned

This confirms that the public IP address was successfully removed while the EC2 instance continued running normally.