Automating CloudFormation Stack Drift Remediation Using AWS Lambda and Amazon EventBridge

Description

To deploy resources with AWS CloudFormation, a stack template is used to specify unique configurations for each resource. Once deployed, resources can be updated through a CloudFormation stack update, or manually using the AWS console, CLI, or APIs. However, this freedom to update deployed resources outside of CloudFormation can impact the consistency of the resource configurations and should be avoided.

With that being said, if an unmanaged update occurs to a resource outside of CloudFormation, developers can utilize the built-in **drift detection** feature. Drift detection can be used to detect stack and resource level changes that misalign resource configurations from their definitions in the stack template. Once **stack drift** is detected, developers can manually update the configurations to bring them back in sync with a stack or develop an automated solution to handle the entire drift detection and remediation process.

In this lab, you will use an AWS Lambda function and an Amazon EventBridge schedule, to continuously monitor a CloudFormation stack using drift detection. When stack drift is detected, your Lambda function will automatically restore the resource settings to realign them with the settings defined in the stack template.

Note: The general solution architecture covered in this hands-on lab can be attributed to the Implement automatic drift remediation for AWS CloudFormation using Amazon CloudWatch and AWS Lambda AWS blog post. For more architecture examples that relate to Cloud Operations and DevOps on AWS, check out the following AWS blogs:

- AWS Cloud Operations & Migrations
- AWS DevOps

Learning Objectives

Upon completion of this advanced-level lab, you will be able to:

- Deploy an AWS Security Group with AWS CloudFormation
- Detect unmanaged resource updates with AWS CloudFormation Drift Detection
- Create an AWS Lambda function that remediates drifted resource configurations
- Schedule automatic drift detection and remediation with an Amazon EventBridge Schedule

Intended Audience

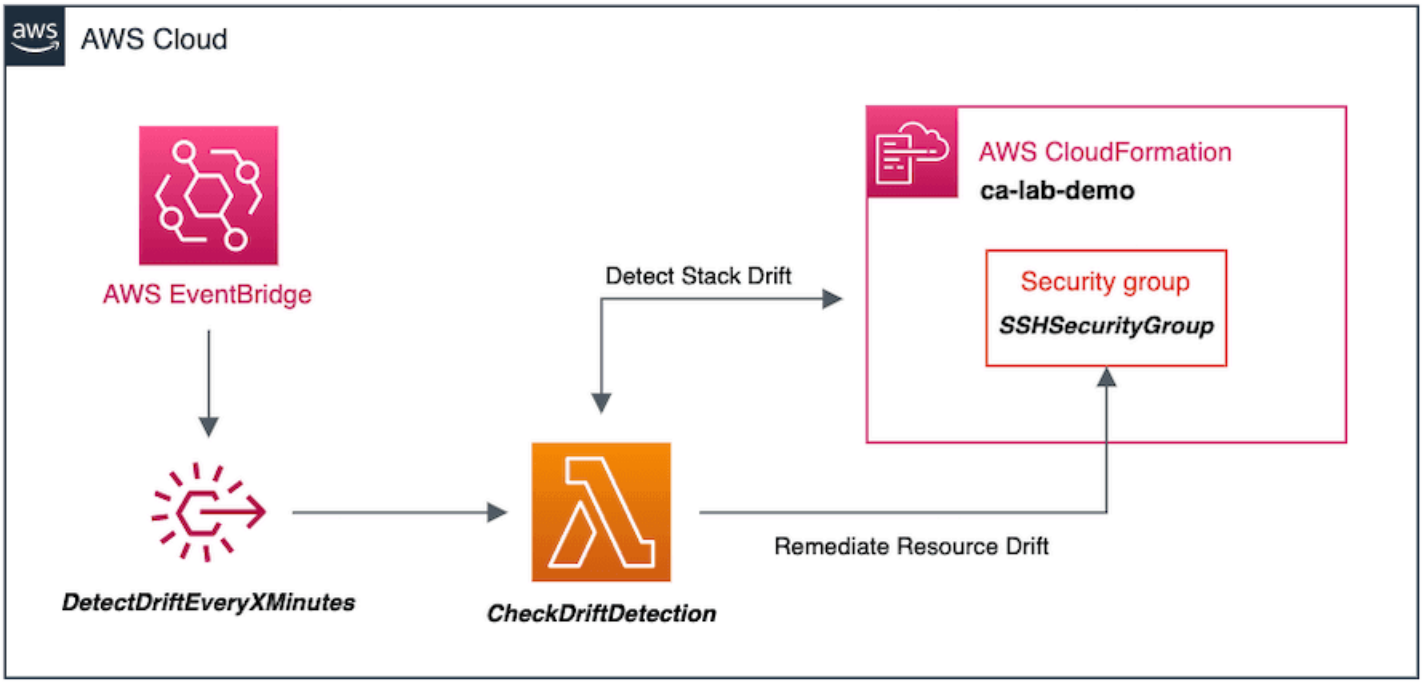
- Candidates for the AWS Certified DevOps Engineer - Professional Exam
- DevOps Engineers
- Cloud Architects
- Software Engineers

Prerequisites

Familiarity with the following will be beneficial but is not required:

- AWS CloudFormation
- AWS Lambda
- Amazon EventBridge

Environment after



Lab steps

- Logging In to the Amazon Web Services Console
- Deploying a Simple AWS CloudFormation Stack
- Detecting Unmanaged Resource Changes with Drift Detection
- Restoring Drifted Resource Settings with AWS Lambda
- Scheduling Automatic Drift Remediation with the Amazon EventBridge Scheduler

Deploying a Simple AWS CloudFormation Stack

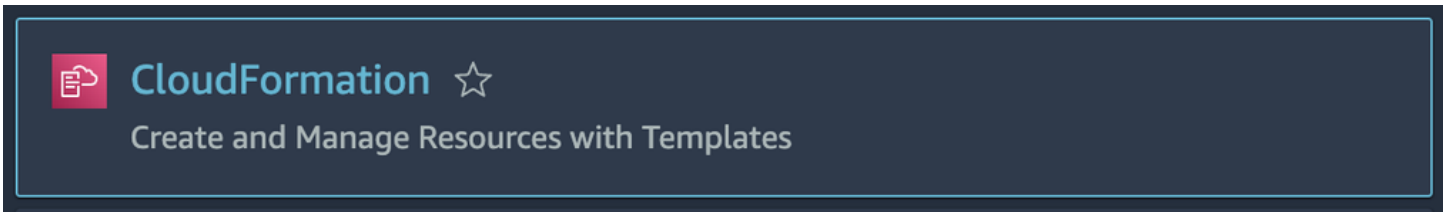
Introduction

In this lab step, you will deploy a single AWS Security Group using AWS Cloudformation. This security group will be updated throughout this lab to illustrate the drift detection and

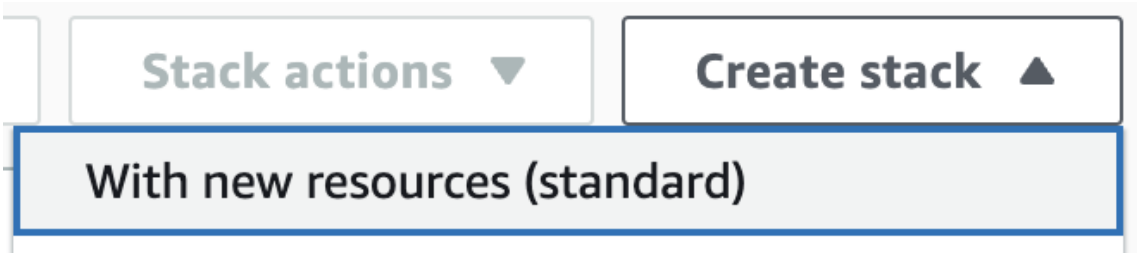
remediation process. The CloudFormation stack in this lab has been simplified in order to effectively illustrate the functionality of the drift detection feature.

Instructions

1. In the AWS Console search bar, search for *cloudformation* and click the **CloudFormation** result under **Services**:



2. Click **Create stack** on the right to expand the dropdown menu and click **With new resources (standard)**:

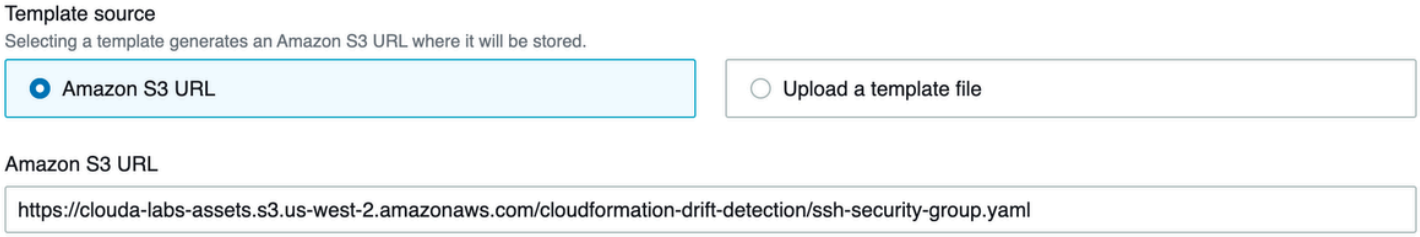


The **Create stack** form will appear.

3. In the **Specify template** section, ensure **Amazon S3 URL** is selected for **Template source**, and enter the following URL in the **Amazon S3 URL** field:

Copy code

```
https://clouda-labs-assets.s3.us-west-2.amazonaws.com/cloudformation-drift-detection/ssh-security-group.yaml
```



Note: Leave the default value for the **Prepare template** section on this page.

The template to be deployed contains a single Security Group resource definition as shown below:

```
Parameters:
  VPCId:
    Description: Default VPC that SG is deployed into
    Type: AWS::EC2::VPC::Id

Resources:
  SSHSecurityGroup:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupDescription: SSH Only Ingress Security Group
      SecurityGroupIngress:
        - CidrIp: "10.0.0.0/20"
          FromPort: 22
          ToPort: 22
          IpProtocol: tcp
      VpcId: !Ref VPCId
```

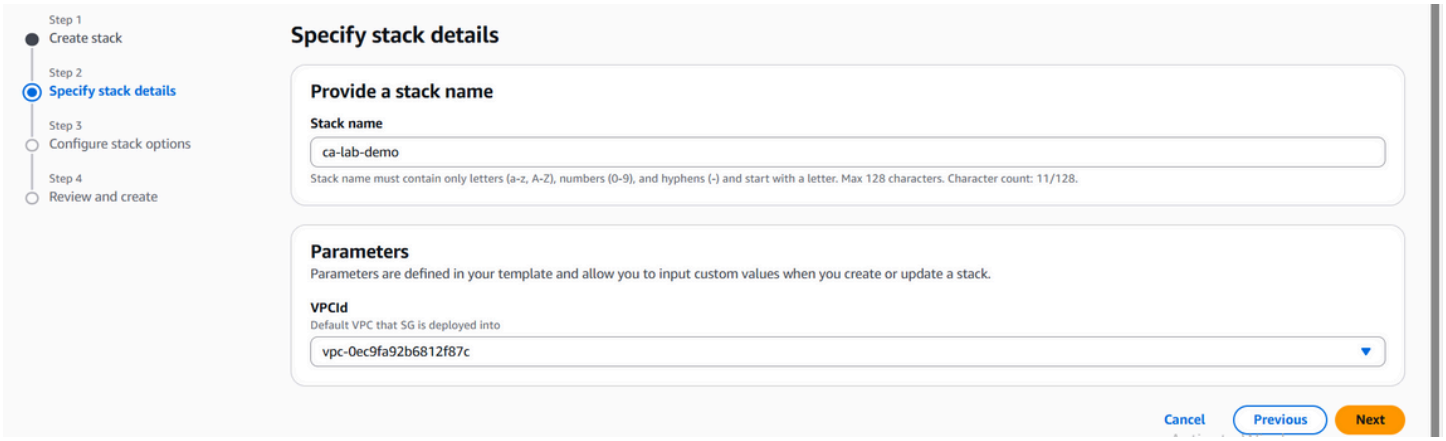
The **SSHSecurityGroup** resource is configured with an inbound rule that will allow SSH traffic on port **22** using the **TCP** protocol. SSH traffic is only allowed from the **10.0.0.0/20CIDR IP** range.

In the stack creation process, you will be prompted to select a VPC for the security group to be provisioned into. The security group will reside in the default VPC of the AWS account.

4. Click **Next**:

5. On the **Specify stack details** section, enter the following values, then click **Next**:

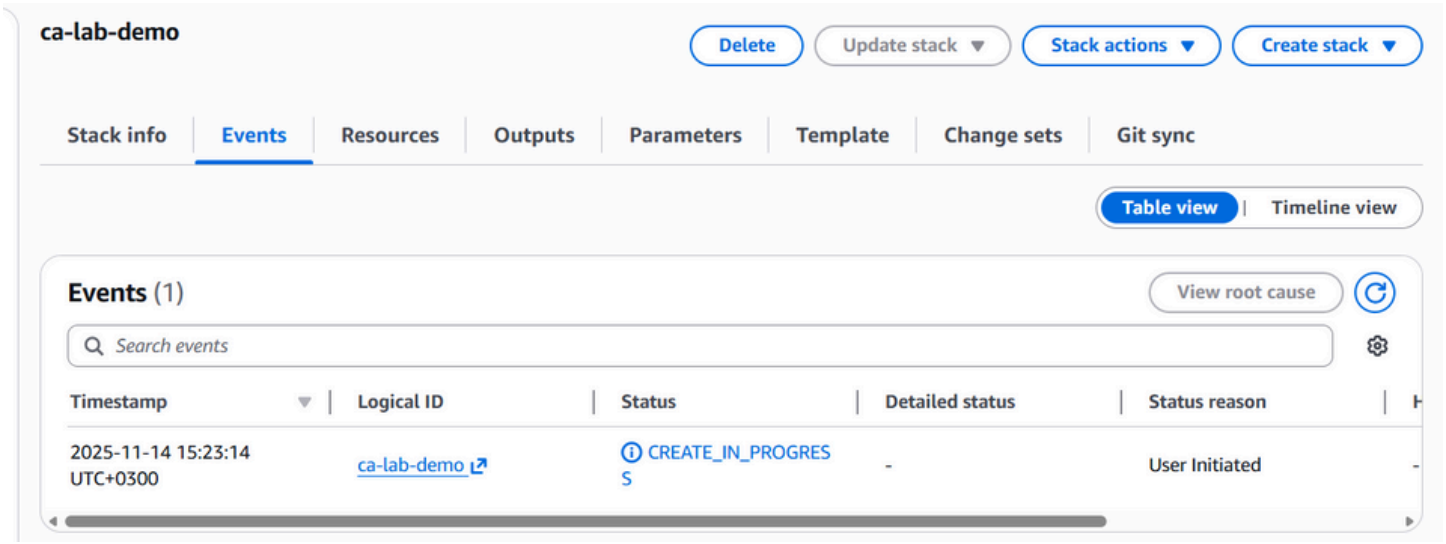
- **Stack name**: Enter *ca-lab-demo*
- **VPCId**: Select the single **vpc-XXXX** ID from the dropdown menu. This is the default VPC of the account.



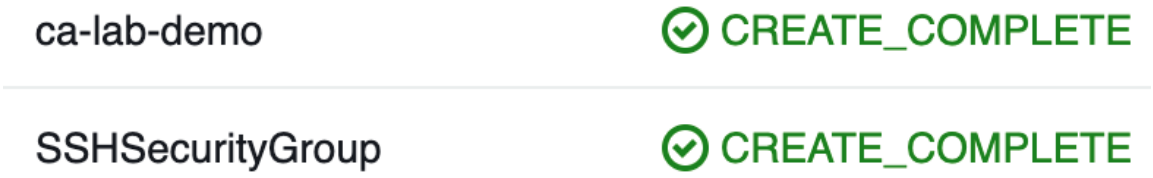
6. You will not be configuring stack options in the **Configure stack options** section. Scroll to the bottom of the page and click **Next** to proceed:

7. On the **Review ca-lab-demo** section, scroll to the bottom of the page and click **Submit**:

You will be redirected to the **Events** page of the **ca-lab-demo** stack.



The security group creation should take less than 1 minute to complete. You may need to refresh the page to view the completed status:



Summary

In this lab step, you deployed an AWS Security Group resource using AWS CloudFormation.

Detecting Unmanaged Resource Changes with Drift Detection

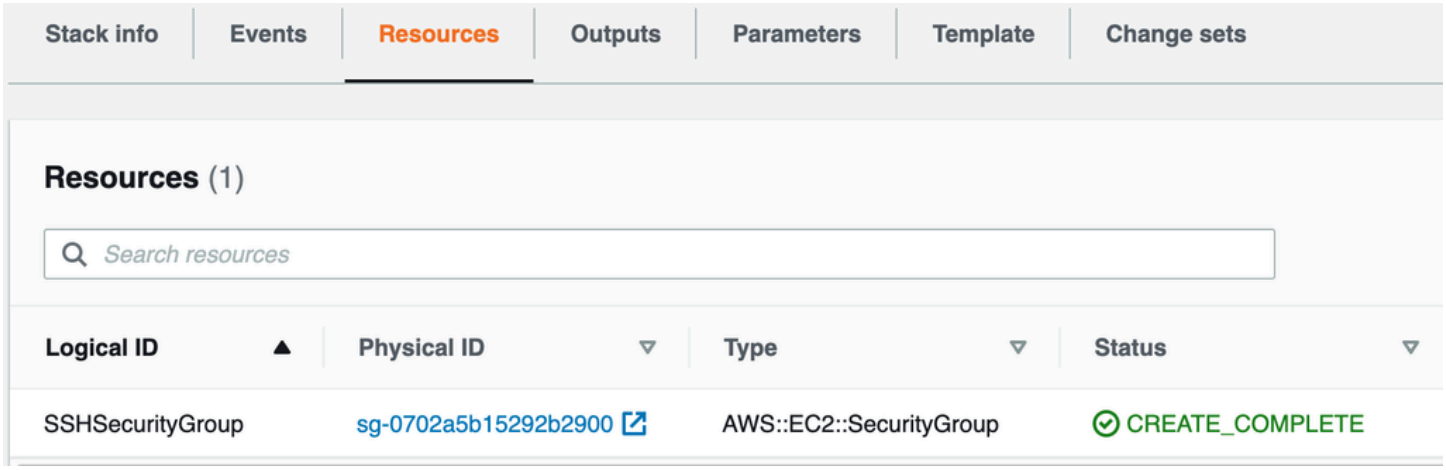
0 out of 1 validations checks passed

Introduction

In this lab step, you will learn how to detect resource updates that occur outside of AWS CloudFormation with drift detection. These unmanaged changes can be done in the AWS console or using the service APIs or CLI tools. You will update the inbound rule of your **SSHSecurityGroup** resource that you deployed in the previous lab step.

Instructions

1. From the **ca-lab-demo** stack detail page, click the **Resources** tab to view the single resource created:

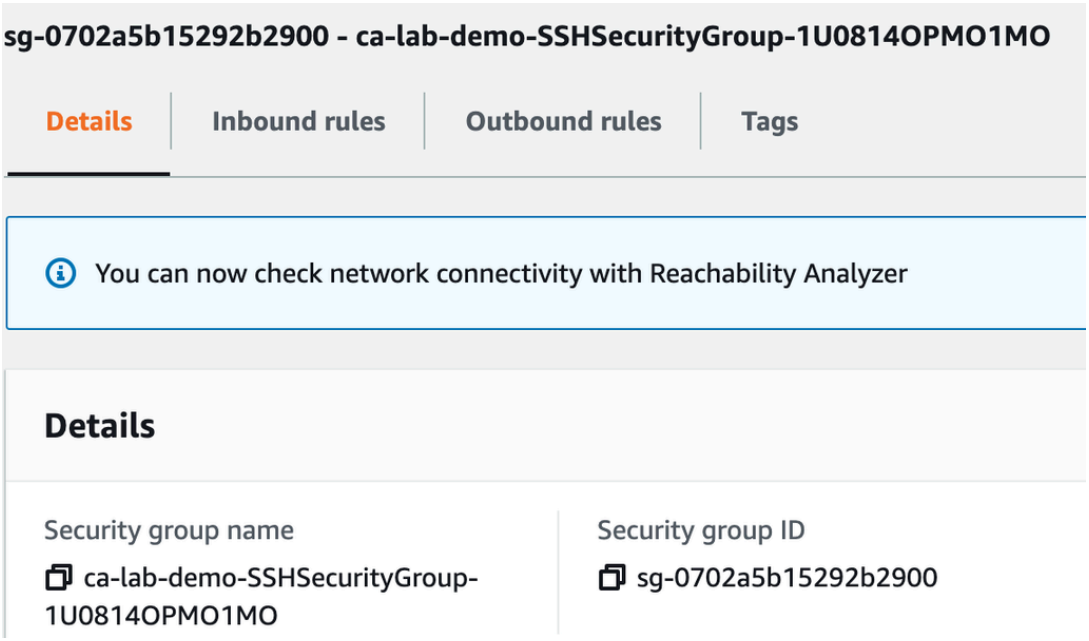


2. Click the **Physical ID** of the **SSHSecurityGroup** resource that follows the pattern **sg-XXXX**:

sg-0702a5b15292b2900

This will redirect you to the details page of the **SSHSecurityGroup** resource in a new browser tab.

On the bottom half of the screen, you will see the **Details** tab currently selected:



Note: You will be referring back to the CloudFormation console throughout the remainder of this lab, so keep the browser tab open.

3. Click the **Inbound rules** tab

The security group consists of the single inbound rule for SSH traffic on port 22 that was defined in the stack template:

Security group rule ID	IP version	Type	Protocol	Port range	Source
sgr-0eb53b3f893588bc6	IPv4	SSH	TCP	22	10.0.0.0/20

It's considered best practice to update stack resources through a CloudFormation stack update in order to maintain consistency between your resources and their definitions in the stack template. However, you have the ability (with the right IAM permissions) to update these settings manually.

4. Click **Edit inbound rules**

You will update the security group rule to allow SSH requests from the public Internet.

5. Below **Source**, select **Anywhere-IPv4** from the dropdown menu:

The CIDR range value will also automatically be updated to **0.0.0.0/0**.

6. Click **Save rules**

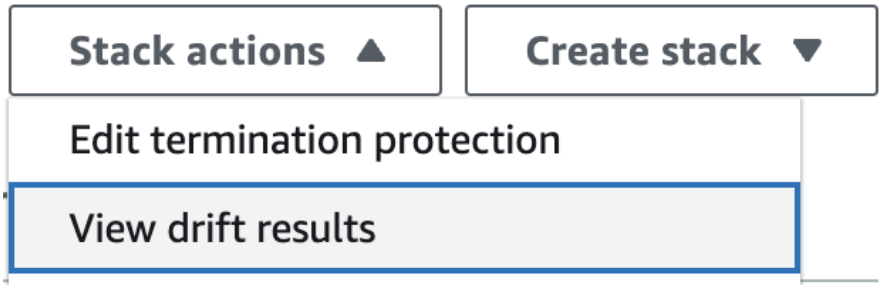
Although you've updated the physical settings of the security group rule to allow public Internet traffic, the **ca-lab-demo** stack template still only allows traffic from the **10.0.0.0/20**

CIDR IP range.

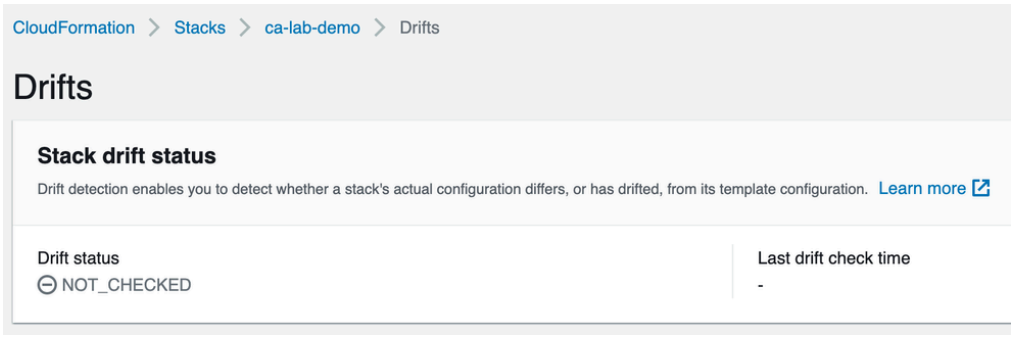
Unmanaged changes like this may affect how other developers interact with the shared stack resources and settings that are out of sync with the stack template will cause the next CloudFormation stack update to fail.

7. Navigate back to the browser tab with the CloudFormation **ca-lab-demo** stack selected.

8. Click the **Stack actions** dropdown menu on the top-right, then click **View drift results**:



You'll be brought to the **ca-lab-demo** drifts table:

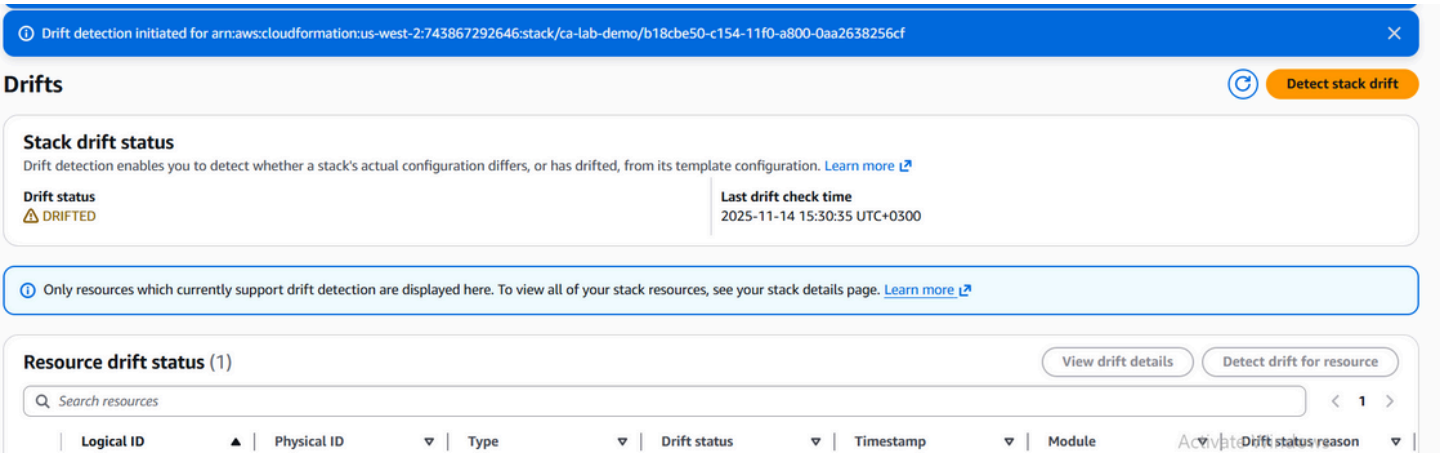


The **Drift status** reads **NOT_CHECKED** since you have yet to detect stack drift.

9. Click **Detect stack drift** at the top-right of the page

The **Drift detection initiated** banner will appear at the top of the page:

10. Click the **Refresh** icon at the top of the page:



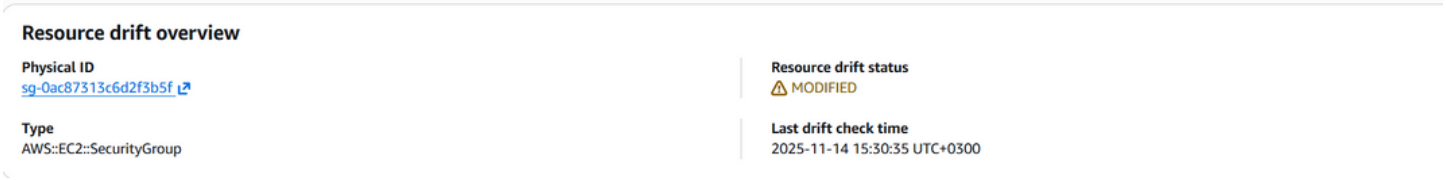
Upon refreshing the page, you will notice the **Drift status** for the stack has been updated to **DRIFTED**.

In the **Resource drift status** table below, you're presented with the resource that has experienced the drift in configuration, the SSHSecurityGroup you just updated.

The **Drift status** for **SSHSecurityGroup** reads **MODIFIED** to indicate that the resource still exists, but was modified in a way that brought it out of alignment with its definition in the stack template.

11. Select the **SSHSecurityGroup** from the table and click **View drift details** to learn more:

The **Drift details** page is split up into three sections:



The **Resource drift overview** section provides the same general information about the drifted resource, current status, and the latest drift detection time stamp.

The **Differences** section summarizes the resource drift and provides the resource **property** and **change status**, along with **expected** and **current** values.

At the bottom of the page, the **Details** section provides a JSON representation of the drifted resource configuration.

In this case, you can manually revert the **CIDR IP** address value for the security group in the console, which will resolve this drifted status. The **NOT_EQUAL** change type is a relatively easy change to revert since it's only a matter of syncing up the two values.

12. Navigate back to the **EC2 Security Groups** browser tab and click **Edit inbound rules** once more for the **SSHSecurityGroup** rule.

13. On the existing SSH rule, under **Source**, click the **X** next to **0.0.0.0/0** to delete it

14. In its place, enter *10.0.0.0/20* and ensure the **Custom** option is selected

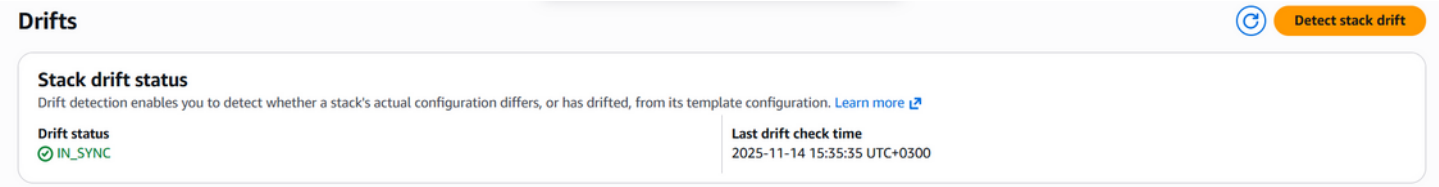
15. Click **Save rules** to submit your changes.

16. Navigate back to the **CloudFormation drift details** browser tab once more, then click **Detect drift for resource**

The page will refresh with a new **Resource drift status** of **IN_SYNC**

17. Click the **Drifts** breadcrumbat the top of the page to return to the Stack drift status page.

The stack **Drift status** also reads **IN_SYNC**:



It's important to verify that both the stack and its underlying resources are in sync. For drifted stacks that have more than one resource, a stack drift status will not appear to be in sync until all resource drifts have been addressed.

Summary

In this lab step, you learned how to detect and remediate stack drifts that are caused by unmanaged resource updates outside of CloudFormation.

Restoring Drifted Resource Settings with AWS Lambda

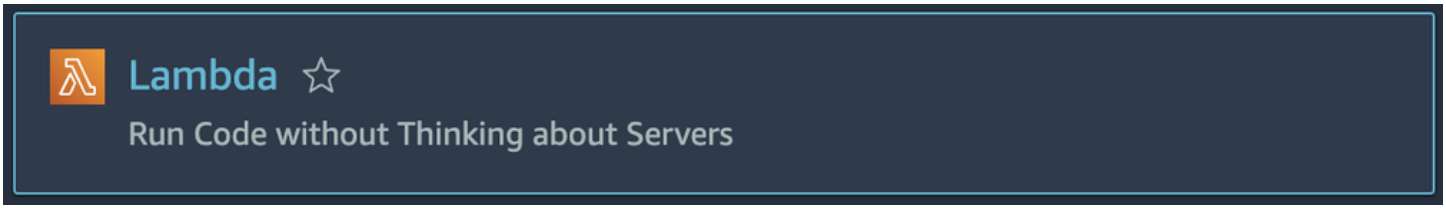
Introduction

Detecting and remediating drifted stack resources can easily be accomplished in the AWS console, but as your stack or team grows, addressing these stack drifts can get tedious. With AWS Lambda, you can develop a function that remediates stack drift by reverting any resources back to their original configurations. To automate this process, you can then configure an AWS EventBridge schedule to invoke this function on a schedule to periodically detect and remediate these drifted resources.

In this lab step, you will develop a Lambda function that accomplishes the detection and remediation phases of the automated process described above.

Instructions

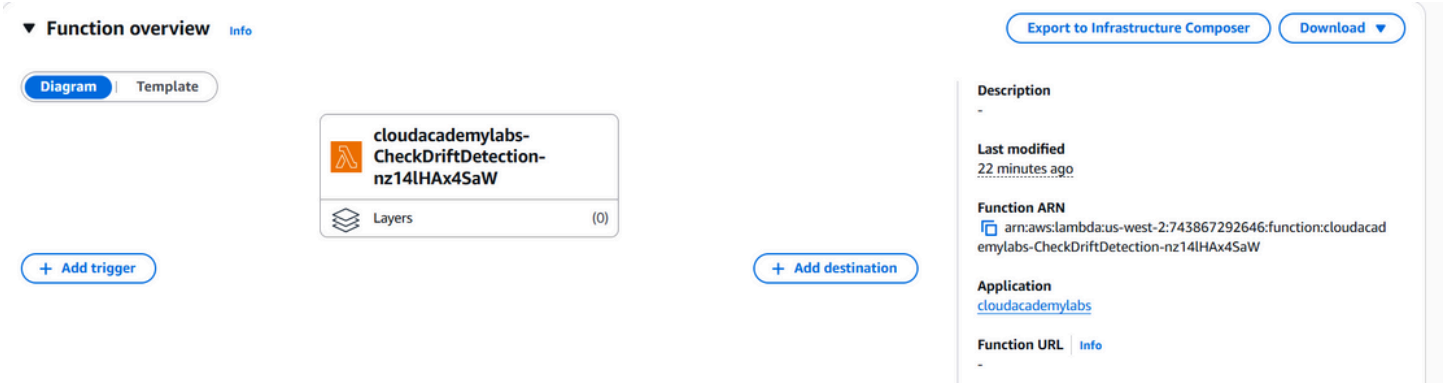
1. In the AWS Console search bar, search for *lambda* and click the **Lambda** result under **Services**:



The **Functions** table will appear with a preconfigured Lambda function.

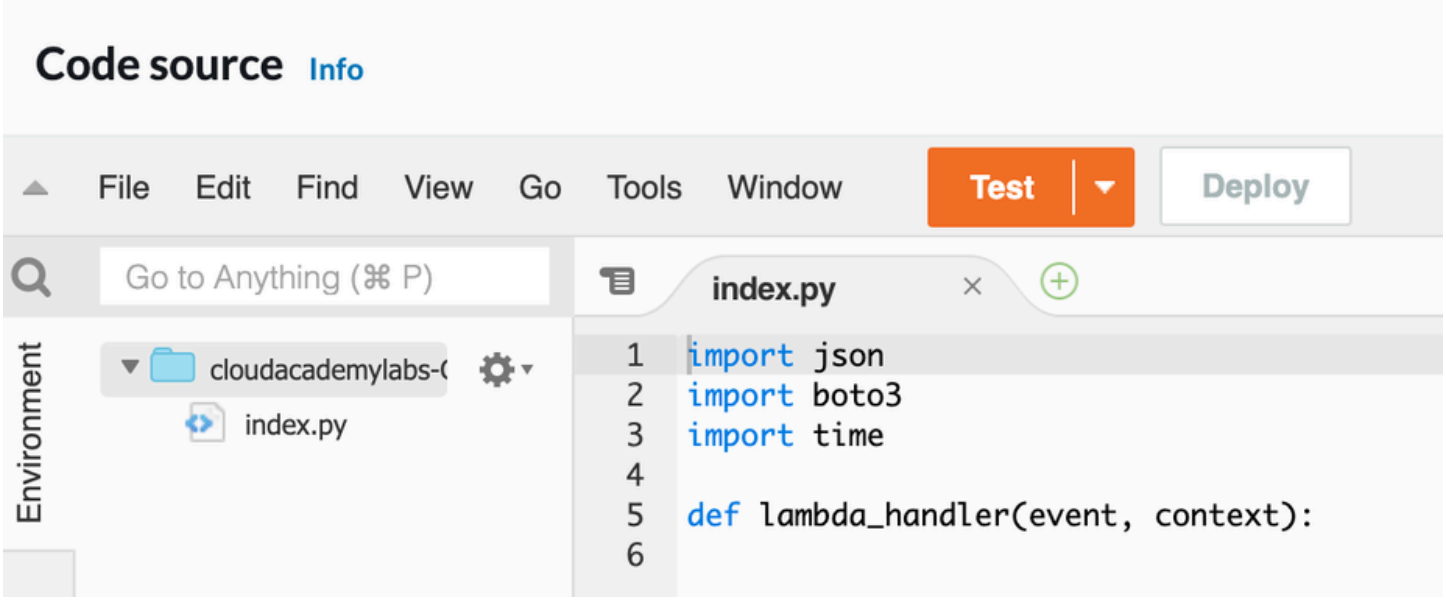
2. Click the **cloudacademylabs-CheckDriftDetection-XXXX** function name:

This function was preconfigured for you at the start of this lab. It uses Python 3.6 runtime and has a Lambda execution role attached to allow the SDK calls described later in this lab step.



3. Scroll down to the **Code source** section.

4. The **index.py** file will be displayed in the code editor with some placeholder code:



5. Replace the contents of the **index.py** file with the following code:

Copy code

```
import json

import boto3

import time


STACK_NAME = "ca-lab-demo"

RESOURCE_TYPE = "AWS::EC2::SecurityGroup"

SECURITY_GROUP_NAME = "SSHSecurityGroup"


CF_CLIENT = boto3.client('cloudformation')

EC2_CLIENT = boto3.client('ec2')


def restore_ssh_security_group(resource_id, expected_properties):

    # Instantiate Security Group resource

    security_group = boto3.resource('ec2').SecurityGroup(resource_id)

    # Retrieve all security group rules
```

```
    rules = EC2_CLIENT.describe_security_group_rules(Filters=[{'Name': 'group-id', 'Values': [resource_id]}])

    # Revoke ALL ingress security group rules
    # Skip egress rules
    if len(rules['SecurityGroupRules']) > 0:
        revoked_rules = []
        for rule in rules['SecurityGroupRules']:
            if rule['IsEgress'] == True:
                continue
            revoked_rules.append(rule['SecurityGroupRuleId'])
        if len(revoked_rules) > 0:
            security_group.revoke_ingress(SecurityGroupRuleIds = revoked_rules)
    # In the event of a deleted expected security group rule,
    # authorize ingress security rule using expected properties
    security_group.authorize_ingress(
                            CidrIp=expected_properties['CidrIp'],
                            FromPort=expected_properties['FromPort'],
                            ToPort=expected_properties['ToPort'],
                            IpProtocol=expected_properties['IpProtocol']
                        )
    print("Restored SSH Security Group Successfully")
    return

def lambda_handler(event, context):
    # Initiate a stack drift detection
```

```
    stack_drift_detection = CF_CLIENT.detect_stack_drift( StackName=STACK_NAME )
    stack_drift_detection_id = stack_drift_detection["StackDriftDetectionId"]
    print(f"Stack Drift Detection ID: {stack_drift_detection_id}")
    drift_detection_status = ""

    while drift_detection_status not in ["DETECTION_COMPLETE",  "DETECTION_FAILED"]:
        check_stack_drift_detection_status = CF_CLIENT.describe_stack_drift_detection_status(
            StackDriftDetectionId=stack_drift_detection_id
        )
        drift_detection_status = check_stack_drift_detection_status["DetectionStatus"]
        # Delay status check for 1 second to avoid CloudFormation API throttling
        time.sleep(1)
    print(f"Completed. Detection Status: {drift_detection_status}")

    # Alert if detection fails then continue with successfully reported resources
    if drift_detection_status == "DETECTION_FAILED":
            print("The stack drift detection did not complete successfully")

    # Check if the stack has drifted
    if check_stack_drift_detection_status["StackDriftStatus"] == "DRIFTED":

        # Retrieve resources that have drifted in ca-lab-demo stack
        stack_resource_drift = CF_CLIENT.describe_stack_resource_drifts(
            StackName=STACK_NAME
        )
```

```
    # Iterate over drifted resources
    for drifted_stack_resource in stack_resource_drift["StackResourceDrifts"]:
        resource_type = drifted_stack_resource["ResourceType"]
        security_group_name = drifted_stack_resource["LogicalResourceId"]
        resource_id = drifted_stack_resource["PhysicalResourceId"]

        # If the drifted resource is the SSH Security Group resource,
        # restore security group rules using expected resource properties
        if resource_type == RESOURCE_TYPE and security_group_name == SECURITY_GROUP_NAME:
            expected_properties = json.loads(drifted_stack_resource["ExpectedProperties"])["SecurityGroupIngress"][0]
            restore_ssh_security_group(resource_id, expected_properties)
    else:
        print("No Drift Detected")
```

The Lambda function code can be broken down into the following sections

The imported libraries include **json**, **boto3**, and **time**. If you're familiar with Boto3 you'll recognize the resource instantiations that occur throughout the code. The Boto3 library consists of easy-to-use clients and resources to be able to interact with the services in your AWS account. The **cloudformation** and **ec2** clients are used in this function.

The **json** and **time** libraries are used for input formatting and establishing a timeout in the code that follows.

The environment variables defined in this section hold the CloudFormation stack name, the specific security group resource type we want this function to address, and the unique security group name. For this example, you're targetting a single resource in a single stack.

```python
def restore_ssh_security_group(resource_id, expected_properties):
    # Instantiate Security Group resource
    security_group = boto3.resource('ec2').SecurityGroup(resource_id)
    # Retrieve all security group rules
    rules = EC2_CLIENT.describe_security_group_rules(Filters=[{'Name': 'group-id', 'Values': [resource_id]}])

    # Revoke ALL ingress security group rules
    # Skip egress rules
    if len(rules['SecurityGroupRules']) > 0:
        revoked_rules = []
        for rule in rules['SecurityGroupRules']:
            if rule['IsEgress'] == True:
                continue
            revoked_rules.append(rule['SecurityGroupRuleId'])
        if len(revoked_rules) > 0:
            security_group.revoke_ingress(SecurityGroupRuleIds = revoked_rules)
    # In the event of a deleted expected security group rule,
    # authorize ingress security rule using expected properties
    security_group.authorize_ingress(
                                        CidrIp=expected_properties['CidrIp'],
                                        FromPort=expected_properties['FromPort'],
                                        ToPort=expected_properties['ToPort'],
                                        IpProtocol=expected_properties['IpProtocol']
                                    )
    print("Restored SSH Security Group Successfully")
    return
```

The first function defined is **restore_ssh_security_group**. When this function is called, you will pass in the specific **resource_id** which represents the SSH security group, and its **expected_properties**. These **expected_properties** are determined after this Lambda function detects the drifted resources. If you recall, resource drift is determined by comparing the expected and current values of a specific configuration.

A **security_group** resource variable is instantiated with Boto3 to obtain the existing rules of the security group. Since a security group consists of both inbound and outbound rules, the conditional block contains filters to ignore **egress** (outbound) security group rules.

All inbound security group rules are collected and stored in the **revoked_rules** variable. To simplify the remediation process, this function collects all existing inbound rules and revokes them.

Once all inbound security group rules have been revoked, the **authorize_ingress** method is called. This method uses the values provided in the **expected_properties** variable to create a new security group rule, effectively remediating the drifted status.

```
def lambda_handler(event, context):
    # Initiate a stack drift detection
    stack_drift_detection = CF_CLIENT.detect_stack_drift( StackName=STACK_NAME )
    stack_drift_detection_id = stack_drift_detection["StackDriftDetectionId"]
    print(f"Stack Drift Detection ID: {stack_drift_detection_id}")
    drift_detection_status = ""

    while drift_detection_status not in ["DETECTION_COMPLETE",  "DETECTION_FAILED"]:
        check_stack_drift_detection_status = CF_CLIENT.describe_stack_drift_detection_status(
            StackDriftDetectionId=stack_drift_detection_id
        )
        drift_detection_status = check_stack_drift_detection_status["DetectionStatus"]
        # Delay status check for 1 second to avoid CloudFormation API throttling
        time.sleep(1)
    print(f"Completed. Detection Status: {drift_detection_status}")

    # Alert if detection fails then continue with successfully reported resources
    if drift_detection_status == "DETECTION_FAILED":
            print("The stack drift detection did not complete successfully")

    # Check if the stack has drifted
    if check_stack_drift_detection_status["StackDriftStatus"] == "DRIFTED":

        # Retrieve resources that have drifted in ca-lab-demo stack
        stack_resource_drift = CF_CLIENT.describe_stack_resource_drifts(
            StackName=STACK_NAME
        )

        # Iterate over drifted resources
        for drifted_stack_resource in stack_resource_drift["StackResourceDrifts"]:
            resource_type = drifted_stack_resource["ResourceType"]
            security_group_name = drifted_stack_resource["LogicalResourceId"]
            resource_id = drifted_stack_resource["PhysicalResourceId"]

            # If the drifted resource is the SSH Security Group resource,
            # restore security group rules using expected resource properties
            if resource_type == RESOURCE_TYPE and security_group_name == SECURITY_GROUP_NAME:
                expected_properties = json.loads(drifted_stack_resource["ExpectedProperties"])["SecurityGroupIngress"][0]
                restore_ssh_security_group(resource_id, expected_properties)
    else:
        print("No Drift Detected")
```

In the **lambda_handler** function, the **CF_CLIENT** is used to make the **detect_stack_drift** API call. The drift detection status is stored in **drift_detection_status**, which will indicate **COMPLETE** or **FAILED**. The **while** loop along with the **time.sleep(1)** timeout ensures that the drift detection status is retrieved before moving on to the next segment of the function while avoiding CloudFormation API throttling.

Note: When detecting drift in a CloudFormation stack with more than one resource, a status of **DETECTION_FAILED** can represent the fact that some AWS resources within the stack are not currently supported by drift detection, opens in a new tab. In this case, drift results will only be presented for supported resources and the function will continue.

When a detection operation completes with a status of **DRIFTED**, the function describes all of the drifted resources. In this case, the function only cares about the security group resource type and the specific **SSHSecurityGroup**.

expected_properties are retrieved by the drift results and are passed along with the **resource_id** to the **restore_ssh_security_group** function to remediate the drift.

6. Click **Deploy** to deploy your changes

The following successful update banner will appear at the top of the page:

Before you test the function to ensure it detects and remediates any stack drift, you will edit the security group ingress rule to bring it out of sync.

Note: As in the previous lab step, keep this browser tab open to make it easier to switch between service consoles.

7. Navigate to the **SSHSecurityGroup** browser tab.

If you've already closed the tab, navigate to the EC2 Security Groups page and select the **ca-lab-demo-SSHSecurityGroup-XXXX** security group

8. On the **SSHSecurityGroup** details page, click the **Inbound rules** tab, click **Edit inbound rules.**

9. This time, click **Delete** next to the single security group rule to delete it:

10. Click **Save rules**.

11. Switch back to the **CheckDriftDetection** Lambda function browser tab, click the **Test** dropdown arrow above the code editor, then select **Configure test event**:

12. In the **Configure test event** wizard, enter *TestDriftDetection*under **Event name**

Keep the default values for the remaining settings in the wizard.

The test event data does not matter for this Lambda function since it will be triggered on a schedule, not by a specific service event.

13. Click **Save** at the bottom of the wizard

14. Click **Test** above the code editor to trigger your Lambda function

The Lambda function invocation will succeed with the following log messages:

```
Stack Drift Detection ID: aa42c6f0-a581-11ec-b55d-0aae502645e9
Completed. Detection Status: DETECTION_COMPLETE
Restored SSH Security Group Successfully
```

15. Navigate back to the **SSHSecurityGroup** details browser tab and click the refresh icon for your inbound rules

The inbound rule to allow **SSH** traffic on port **22** from **10.0.0.0/20** has been reinstated on the security group

Now that you have a Lambda function to handle drift detection and remediation of your stack, you will automate the entire process by creating an AWS EventBridge schedule to invoke your Lambda function at a scheduled rate.

Summary

In this lab step, you configured a Lambda function to act on your behalf to detect and remediate the drifted resource settings of your security group.

Scheduling Automatic Drift Remediation with the Amazon EventBridge Scheduler

Introduction

In this lab step, you will tie everything together by configuring an Amazon EventBridge schedule to trigger your drift detection Lambda function every 5 minutes.

Instructions

1. In the AWS Console search bar, search for *scheduler* and click the **Amazon EventBridge Scheduler** result under **Services**

2. Click **Create schedule** to get started

The **Specify schedule detail** page will appear.

3. Configure the following schedule:

- **Name**: Enter *DetectDriftEvery5Minutes*
- **Description**: Enter *Trigger Lambda function to detect and remediate drifted stack resources every 5 minutes*
- **Schedule group**: Keep the **default** schedule group selected
- **Occurrence**: Select **Recurring schedule**
- **Schedule type**: Select **Rate-based schedule**
- **Rate expression**: Enter *5* for **Value** and select **minutes** for the **Unit**
- **Flexible time window**: Select **Off** from the drop-down menu
- Accept the defaults for options not specified

Schedule name and description

Schedule name

DetectDriftEvery5Minutes

Use only letters, numbers, dashes, dots or underscores. Max 64 characters.

Description - *optional*

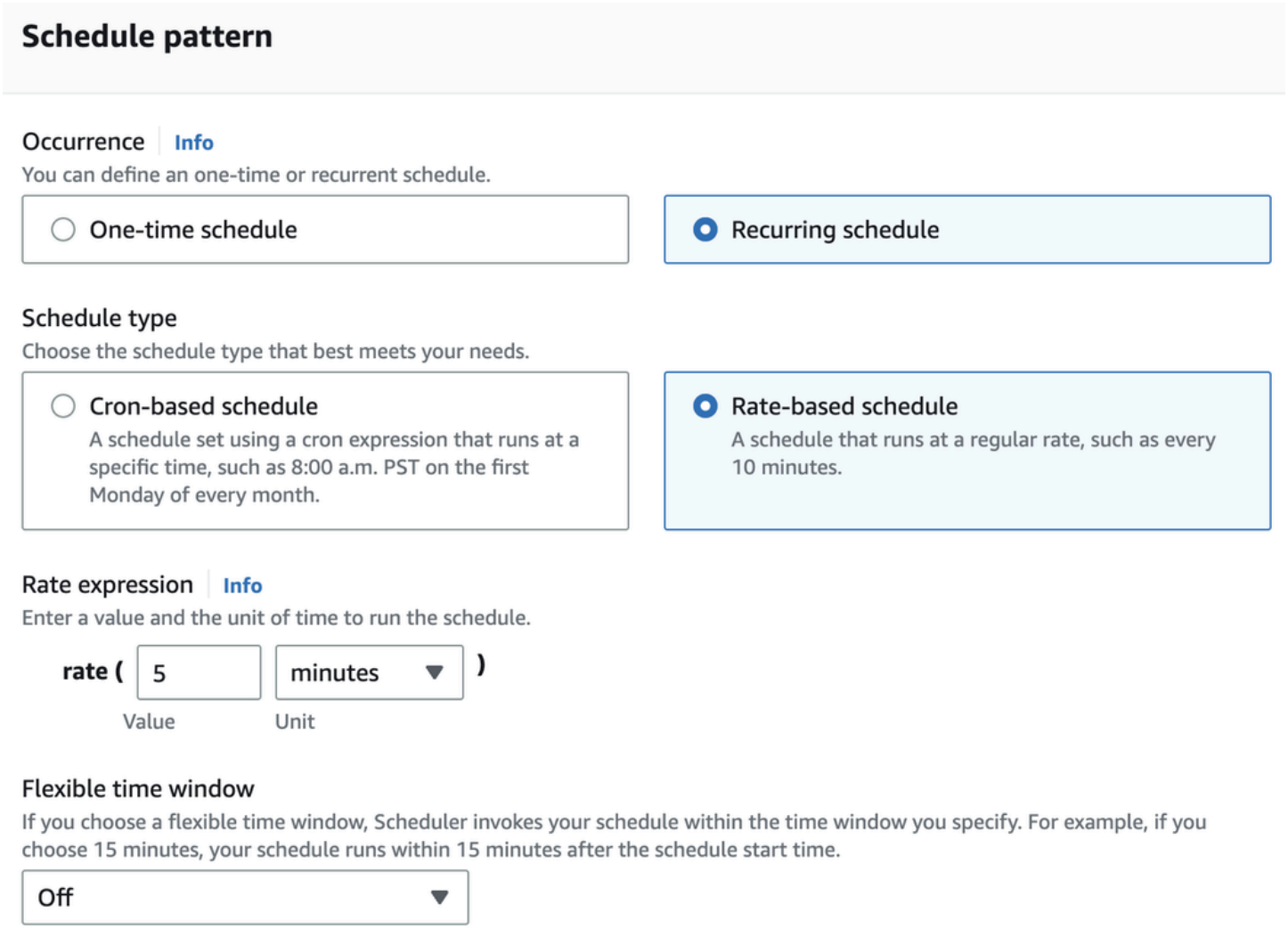
Trigger Lambda function to detect and remediate drifted stack resources every 5 minutes

Maximum of 512 characters.

Schedule group

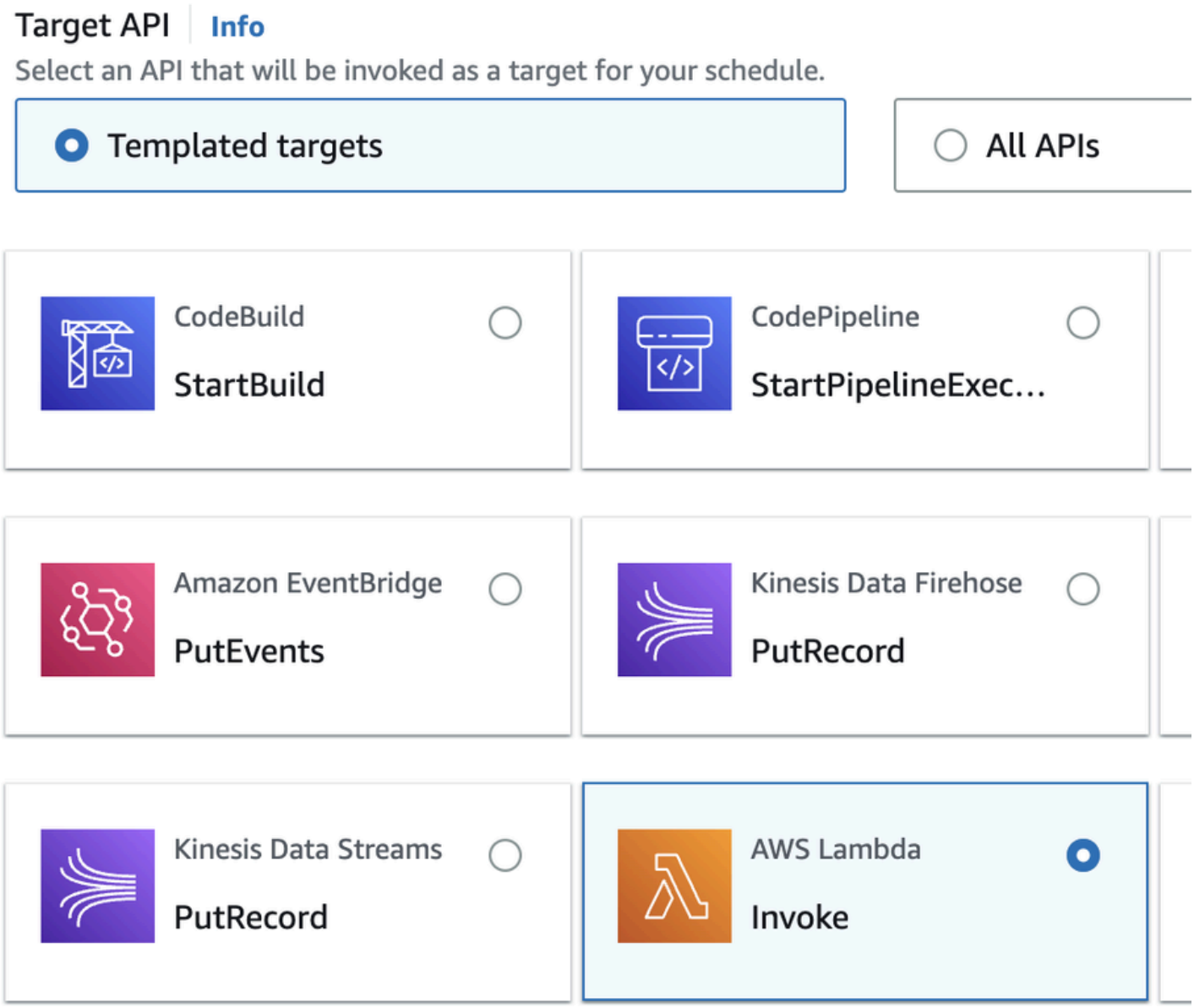
Each schedule needs to be placed in a schedule group. By default, a schedule is placed in the 'Default' **your own schedule group**. You can only add tags to a schedule group, not a schedule.

default



4. Click **Next**

5. Ensure **Templated targets** is selected, then select the **AWS Lambda Invoke** API target:

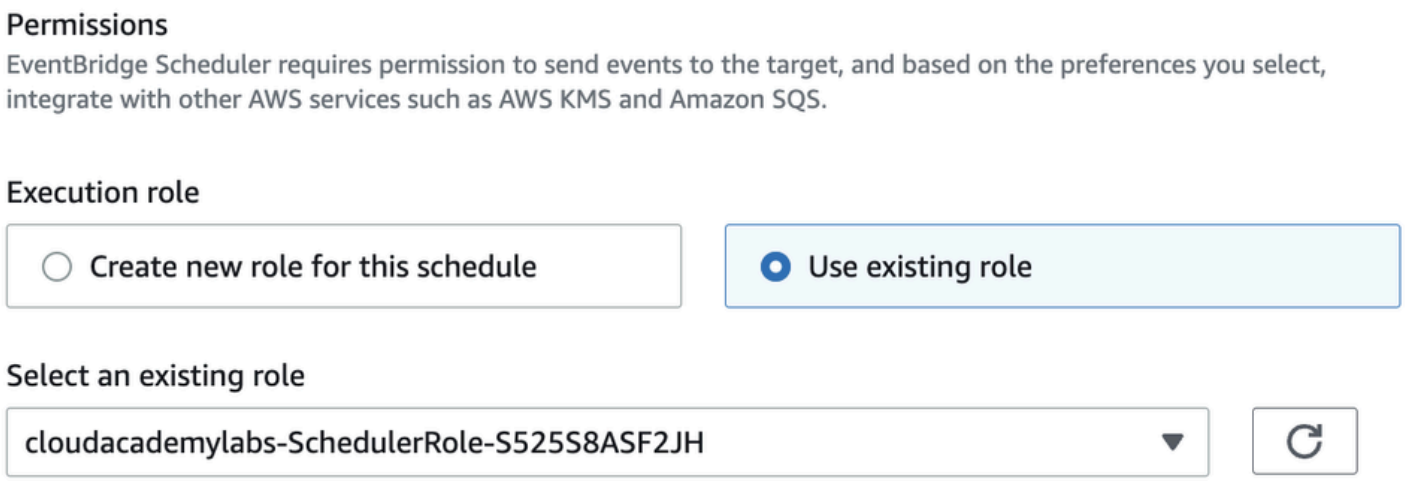


6. Select the **cloudacademylabs-CheckDriftDetection-XXXX** function from the drop-down menu as your **Lambda function** target:

7. Click **Next**

8. In the **Retry policy and dead-letter queue (DLQ)** section, click the **Retry** toggle button under **Retry policy** to disable retry attempts:

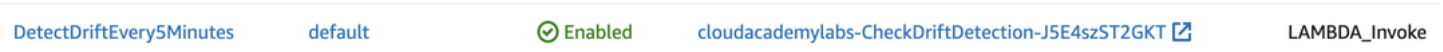
9. Scroll down to the **Permissions** section, select **Use existing role**, then select the **cloudacademylabs-SchedulerRole-###** from the drop-down menu:



10. Click **Next**

11. Click **Create schedule**

Your new EventBridge schedule will appear on the resulting **Schedules** page:



You have successfully created an EventBridge schedule to invoke your **CheckDriftDetection** Lambda function every 5 minutes in order to detect and remediate any drifted stack resource configurations.

Important: Your Lambda function has been written to overwrite all existing inbound rules to ensure only one SSH-onlyrule exists. This style of strict replacement may not work well as you are developing a CloudFormation stack. This type of workflow to automatically revert changes without notice is best suited for stacks that have rigid security requirements. **The example workflow in this lab is for demonstration purposes only**.

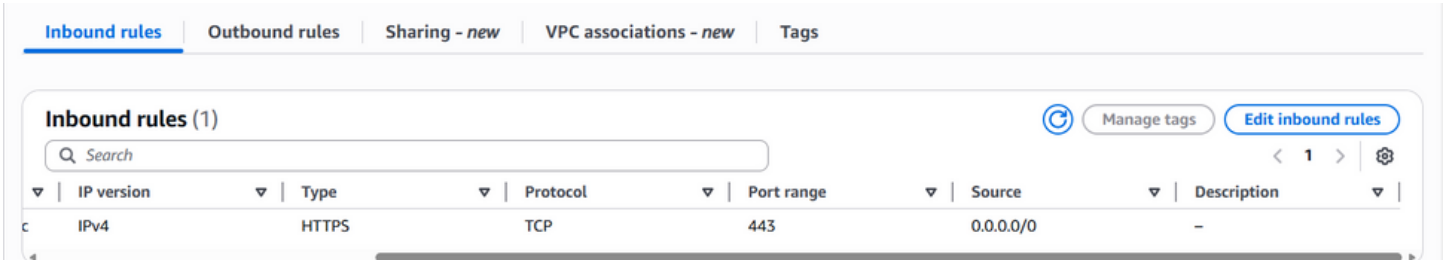
12. To perform a final test, navigate back to the EC2 Security Groups table.

13. Select the security group with the **Security group name** of **ca-lab-demo-SSHSecurityGroup-XXXX**.

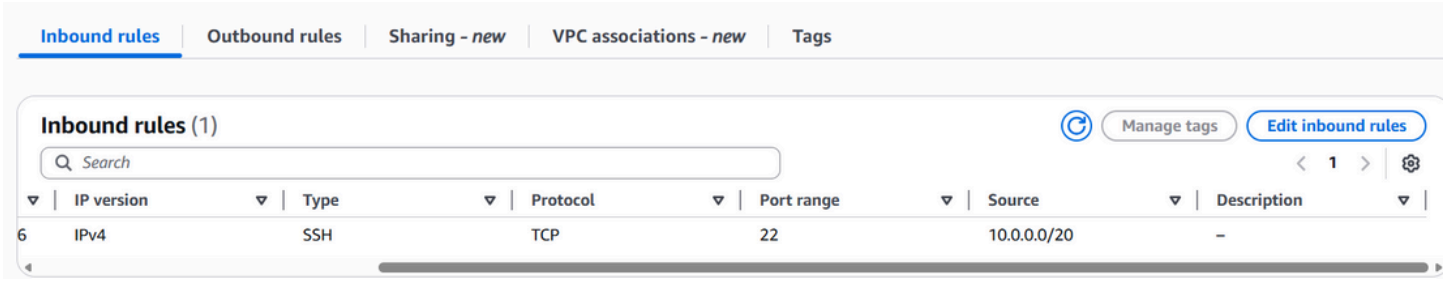
14. Down in the details section, click the **Inbound rules** tab, then **Edit inbound rules**.

15. Change the **Type** of the inbound security group rule to **HTTPS** and the **Source** to **Anywhere-IPv4**

16. Click **Save rules**



After 5 minutes, you will notice the inbound security group rule has been automatically reverted to the original **SSH / 22 / 10.0.0.0/20** configuration.



Note: The EventBridge schedule for the 5 minutes begins as soon as the schedule is created so you may notice the updated security group rule before the 5 minutes are up.

The EventBridge schedule automatically triggers the Lambda function, which detects and remediates any updates to the inbound security group rules on the **SSHSecurityGroup** instance. Stack drift for the **ca-lab-demo** stack will continue to be monitored and remediated every 5 minutes with this automated solution.

Summary

In this lab, you've successfully accomplished the following tasks:

- Created a security group using AWS CloudFormation
- Detected and remediated unmanaged updates using drift detection
- Configured an AWS Lambda function to act on your behalf to detect and remediate drifted settings on a security group
- Automated the entire workflow to run on a schedule of every 5 minutes using an Amazon EventBridge schedule