

AmazonConnectによる自動電話通知（7.複数連絡先への電話通知〈構築⑧〉）

AWS



×

ZABBIX

×



AmazonConnectによる自動電話通知 （7.複数連絡先への電話通知〈構築⑧〉）

2021.11.12 2021.11.06

[【前回】 AmazonConnectによる自動電話通知（7.複数連絡先への電話通知〈構築⑦〉）](#)[【次回】 AmazonConnectによる自動電話通知（7.複数連絡先への電話通知〈構築⑨〉）](#)[【簡易版】 AmazonConnectによる自動電話通知（まとめ）](#)

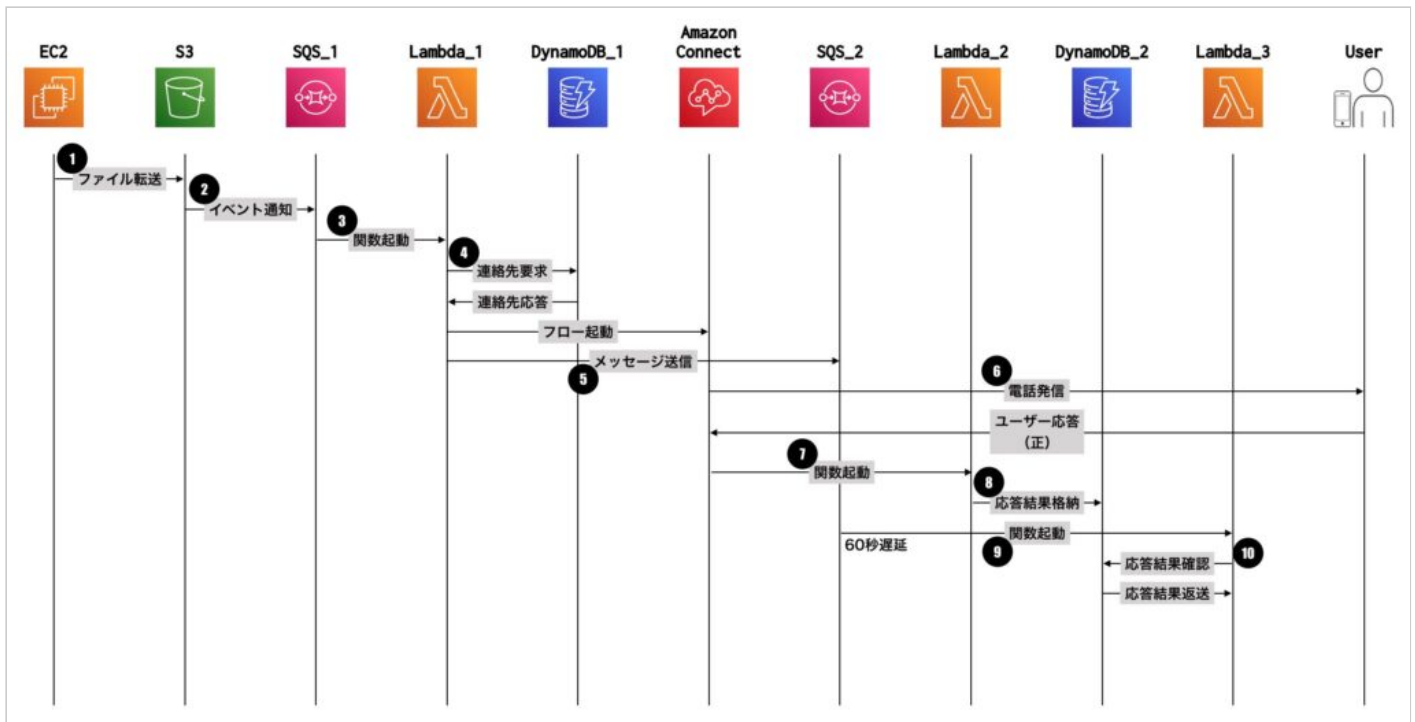
監視サーバーで障害を検知した際に、自動で電話通知できるようにしていきます。ネットワークエンジニアも利用することの多い監視サーバー(Zabbix)で障害検知し、AWS上のAmazonConnectを利用し自動電話を発信します。

今回は下記の条件を満たせるようにAWSの各サービスを利用して自動電話通知の仕組みを導入します。

- 複数の通知先を登録した連絡先リストを持たせる。
- 連絡先リストに優先度(通知順)を設定する。
- 優先度が高い人に最初に電話する。
- 応答が無かった場合、次の優先度の人に順番に電話する。
- 連絡先リストの最後まで電話しても応答が無かった場合、最初に戻って継続する。

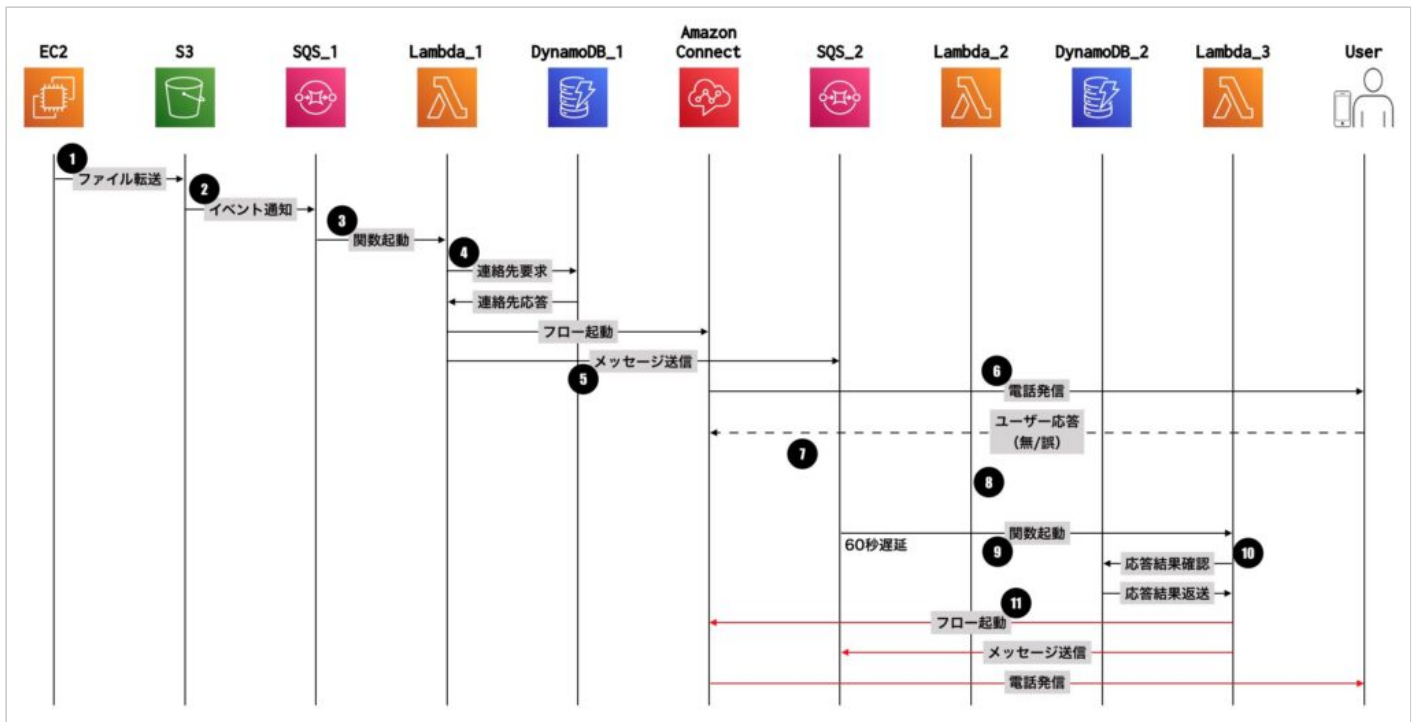
自動電話通知フロー

電話に応答した場合のフロー



1. EC2上の監視サーバーで障害を検知し、S3へトリガーファイルを格納
2. S3のイベント通知機能で、SQS_1にメッセージを送信
3. SQS_1をトリガーとして、Lambda_1を起動
4. Lambda_1がDynamoDB_1から連絡先を取得し、AmazonConnectを起動
5. Lambda_1がAmazonConnectを起動すると同時に、SQS_2へメッセージを送信
6. AmazonConnectがユーザーへ自動電話通知を実施
7. ユーザーが正常応答し、AmazonConnectがLambda_2を起動
8. Lambda_2が応答結果をDynamoDB_2に保存(応答OK)
9. 60秒後にSQS_2をトリガーとしてLambda_3を起動
10. Lambda_3がDynamoDB_2の応答結果を確認(正常応答しているため、何もせずに処理完了)

電話に応答しなかった場合のフロー



1. EC2上の監視サーバーで障害を検知し、S3へトリガーファイルを格納
2. S3のイベント通知機能で、SQS_1にメッセージを送信
3. SQS_1をトリガーとして、Lambda_1を起動
4. Lambda_1がDynamoDB_1から連絡先を取得し、AmazonConnectを起動
5. Lambda_1がAmazonConnectを起動すると同時に、SQS_2へメッセージを送信
6. AmazonConnectがユーザーへ自動電話通知を実施
7. ユーザーが正常応答せず、AmazonConnectがLambda_2を起動
8. Lambda_2が応答結果をDynamoDB_2に保存(応答NG)
9. 60秒後にSQS_2をトリガーとしてLambda_3を起動
10. Lambda_3がDynamoDB_2の応答結果を確認
11. 正常応答していないため、再度AmazonConnectを起動(以降、5から繰り返す)

Lambda_1の修正

Lambda_1を下記の通り修正します。

```

import json
import boto3
from boto3.dynamodb.conditions import Key, Attr

# boto3からDynamoDBへアクセスするためのオブジェクトを取得
dynamodb = boto3.resource('dynamodb')
  
```

```
# "amazonconnect-contact-list"へアクセスするためのオブジェクトを取得
```

```
contactlist = dynamodb.Table("amazonconnect-contact-list")
```

```
# "amazonconnect-response-status"へアクセスするためのオブジェクトを取得
```

```
responsestatus = dynamodb.Table("amazonconnect-response-status")
```

```
# CallStatusを"NG"に変更する関数
```

```
def status_ng():
```

```
    response = responsestatus.update_item(
        Key={
            'No': 1,
            'Name': "Response"
        },
        UpdateExpression="set CallStatus=:c",
        ExpressionAttributeValues={
            ':c': "NG"
        },
        ReturnValues="UPDATED_NEW"
    )
    return response
```

```
# "amazonconnect-contact-list"の内容を返す関数
```

```
def operation_scan():
```

```
    scanData = contactlist.scan()
    items=scanData['Items']

    return scanData["Items"]
```

```
# 指定されたプライオリティの電話番号を返す関数
```

```
def get_phone_number(json_contactinfo, now_priority):
```

```
    for line in json_contactinfo:
        if line['Priority']==now_priority:
            phone_number = line['Phone']

    return phone_number
```

```
def lambda_handler(event, context):

    # 応答結果を初期化
    status_ng()

    # 電話番号リストを取得
    contactinfo = operation_scan()

    # 初回のプライオリティを"1"に設定
    priority = 1

    # 指定したプライオリティの電話番号を取得
    phone_number = get_phone_number(contactinfo, priority)

    # boto3からAmazonConnectへアクセスするためのオブジェクトを取得
    connect = boto3.client('connect')

    # AmazonConnectの問い合わせフローを呼び出し電話発信
    connect.start_outbound_voice_contact(
        DestinationPhoneNumber=phone_number,
        ContactFlowId='*****',
        InstanceId='*****',
        SourcePhoneNumber='+81*****',
    )

    # boto3からSQSへアクセスするためのオブジェクトを取得
    sqs = boto3.resource('sqs')

    # "amazonconnect-queue-confirm"へアクセスするためのオブジェクトを取得
    name = 'amazonconnect-queue-confirm'
    queue = sqs.get_queue_by_name(QueueName=name)

    # 現在のプライオリティをセットしてステータス確認用のキューにメッセージ送信
    response = queue.send_message(MessageBody=json.dumps({"priority": priority}))
```

ContactFlowId: 問い合わせフローID

InstanceId: インスタンスID

SourcePhoneNumber: 発信元の電話番号

※国番号をつけて記述（日本の050の番号の場合、+8150*****）

```

1 import json
2 import boto3
3 from boto3.dynamodb.conditions import Key, Attr
4
5 # boto3からDynamoDBへアクセスするためのオブジェクトを取得
6 dynamodb = boto3.resource('dynamodb')
7
8 # "amazonconnect-contact-list"へアクセスするためのオブジェクトを取得
9 contactlist = dynamodb.Table("amazonconnect-contact-list")
10
11 # "amazonconnect-response-status"へアクセスするためのオブジェクトを取得
12 responsestatus = dynamodb.Table("amazonconnect-response-status")
13
14 # CallStatusを"NG"に変更する関数
15 def status_ng():
16     response = responsestatus.update_item(
17         Key={
18             'No': 1,
19             'Name': "Response"
20         },
21         UpdateExpression="set CallStatus=:c",
22         ExpressionAttributeValues={
23             ':c': "NG"
24         },
25         ReturnValues="UPDATED_NEW"
26     )
27     return response
28
29 # "amazonconnect-contact-list"の内容を返す関数
30 def operation_scan():
31     scanData = contactlist.scan()
32     items=scanData['Items']
33
34     return scanData["Items"]
35
36 # 指定されたプライオリティの電話番号を返す関数
37 def get_phone_number(json_contactinfo, now_priority):
38
39     for line in json_contactinfo:
40         if line['Priority']==now_priority:
41             phone_number = line['Phone']
42
43     return phone_number
44
45 def lambda_handler(event, context):
46
47     # 応答結果を初期化
48     status_ng()
49
50     # 電話番号リストを取得
51     contactinfo = operation_scan()
52
53     # 初回のプライオリティを"1"に設定
54     priority = 1
55
56     # 指定したプライオリティの電話番号を取得
57     phone_number = get_phone_number(contactinfo, priority)
58
59     # boto3からAmazonConnectへアクセスするためのオブジェクトを取得
60     connect = boto3.client('connect')
61
62     # AmazonConnectの問い合わせフローを呼び出し電話発信
63     connect.start_outbound_voice_contact(
64         DestinationPhoneNumber=phone_number,
65         ContactFlowId='...',
66         InstanceId='...',
67         SourcePhoneNumber='+81...',
68     )
69
70     # boto3からSQSへアクセスするためのオブジェクトを取得
71     sqs = boto3.resource('sqs')
72
73     # "amazonconnect-queue-confirm"へアクセスするためのオブジェクトを取得
74     name = 'amazonconnect-queue-confirm'
75     queue = sqs.get_queue_by_name(QueueName=name)
76
77     # 現在のプライオリティをセットしてステータス確認用のキューにメッセージ送信
78     response = queue.send_message(MessageBody=json.dumps({"priority": priority}))
79
80
81
82
83

```

Lambda_2の修正

Lambda_2を下記の通り修正します。

```
import boto3
import json
from boto3.dynamodb.conditions import Key, Attr

# boto3からDynamoDBへアクセスするためのオブジェクトを取得
dynamodb = boto3.resource('dynamodb')

# "amazonconnect-response-status"へアクセスするためのオブジェクトを取得
responsestatus = dynamodb.Table("amazonconnect-response-status")

# CallStatusを"OK"に変更する関数
def status_ok():
    response = responsestatus.update_item(
        Key={
            'No': 1,
            'Name': "Response"
        },
        UpdateExpression="set CallStatus=:c",
        ExpressionAttributeValues={
            ':c': "OK"
        },
        ReturnValues="UPDATED_NEW"
    )
    return response

def lambda_handler(event, context):

    # 通知先の電話番号を取得
    destphone = event['Details']['ContactData']['CustomerEndpoint']['Address']

    # ユーザーの応答結果を取得
    pressnum = event['Details']['ContactData']['Attributes']['pressnum']

    # ユーザーの応答結果がOKの場合、CallStatusをOKに変更し、通知成功のログを出力
    if pressnum == "1":
        status_ok()
        print(destphone + "への通知に成功しました。")

    # ユーザーの応答結果がOKの場合、通知失敗のログを出力
```

```

else:
    print(destphone + "への通知に失敗しました。")

return {
    'statusCode': 200
}

```

```

1 import boto3
2 import json
3 from boto3.dynamodb.conditions import Key, Attr
4
5 # boto3からDynamoDBへアクセスするためのオブジェクトを取得
6 dynamodb = boto3.resource('dynamodb')
7
8 # "amazonconnect-response-status"へアクセスするためのオブジェクトを取得
9 responsestatus = dynamodb.Table("amazonconnect-response-status")
10
11 # CallStatusを"OK"に変更する関数
12 def status_ok():
13     response = responsestatus.update_item(
14         Key={
15             'No': 1,
16             'Name': "Response"
17         },
18         UpdateExpression="set CallStatus=:c",
19         ExpressionAttributeValues={
20             ':c': "OK"
21         },
22         ReturnValues="UPDATED_NEW"
23     )
24     return response
25
26
27 def lambda_handler(event, context):
28
29     # 通知先の電話番号を取得
30     destphone = event['Details']['ContactData']['CustomerEndpoint']['Address']
31
32     # ユーザーの応答結果を取得
33     pressnum = event['Details']['ContactData']['Attributes']['pressnum']
34
35     # ユーザーの応答結果がOKの場合、CallStatusをOKに変更し、通知成功のログを出力
36     if pressnum == "1":
37         status_ok()
38         print(destphone + "への通知に成功しました。")
39
40     # ユーザーの応答結果がOKの場合、通知失敗のログを出力
41     else:
42         print(destphone + "への通知に失敗しました。")
43
44     return {
45         'statusCode': 200
46     }
47

```

Lambda_3の作成(AmazonConnectの通知結果を確認する関数)

関数の作成

Lambdaの「関数の作成」をクリックします。



下記の通り入力し、「関数の作成」をクリックします。

オプション：一から作成を選択

関数名：任意の名前を入力 ※ここでは、“amazonconnect-status-confirm”としています。

ランタイム：Pythonを選択 ※ここでは、最新版の“Python 3.9”を選択しています。

Lambda > 関数 > 関数の作成

関数の作成 Info

以下のいずれかのオプションを選択して、関数を作成します。

一から作成 Info
シンプルな Hello World の例で開始します。

設計図の使用
一般的ユースケース用のサンプルコードと設定プリセットから Lambda アプリケーションを構築します。

コンテナイメージ
関数にデプロイするコンテナイメージを選択します。

Serverless Application Repository の参照
AWS Serverless Application Repository からサンプル Lambda アプリケーションをデプロイします。

基本的な情報

関数名
関数の目的を名前として入力します。

amazonconnect-status-confirm

半角英数字、ハイフン、アンダースコアのみを使用でき、スペースは使用できません。

ランタイム Info
関数の記述に使用する言語を選択します。コンソールコードエディタは Node.js、Python、および Ruby のみをサポートすることに注意してください。

Python 3.9

アーキテクチャ Info
関数コードに必要な命令セットアーキテクチャを選択します。

☒ x86_64
☐ arm64

アクセス権限 Info
デフォルトでは、Lambda は Amazon CloudWatch Logs にログをアップロードするアクセス許可を持つ実行ロールを作成します。このデフォルトのロールは、後でトリガーを追加するときにカスタマイズできます。

▶ デフォルトの実行ロールの変更

▶ 詳細設定

キャンセル **関数の作成**

フィードバック 日本語 ▼ © 2008 - 2021, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。 プライバシーポリシー 利用規約 Cookie の設定

関数が作成されたことを確認します。

関数 amazonconnect-status-confirm を正常に作成しました。関数コードおよび設定を変更できるようになりました。テストイベントで関数を呼び出すには、[テスト] をクリックします。

Lambda > 関数 > amazonconnect-status-confirm

amazonconnect-status-confirm

スロットリング ARN をコピー アクション ▼

▼ 関数の概要 Info

amazonconnect-status-confirm

Layers (0)

+ トリガーを追加

+ 送信先を追加

説明
-

最終更新
46 秒前

関数の ARN
arn:aws:lambda:ap-northeast-1:~:function:amazonconnect-status-confirm

アクセス権の追加（ロールの設定）

作成された関数の設定タブに移動し、実行ロールをクリックします。

The screenshot shows the 'Execution Role' page in the AWS IAM console. The role name is 'amazonconnect-status-confirm-role'. Under the 'Resources overview' section, it lists resources for Amazon CloudWatch Logs. A table shows the following actions and resources:

リソース	アクション
arn:aws:logs:ap-northeast-1:*:log-group:/aws/lambda/amazonconnect-status-confirm:*	Allow: logs:CreateLogGroup Allow: logs:CreateLogStream Allow: logs:PutLogEvents

Below the table, a note states: 'この情報は以下のポリシーステートメントから取得されました。' (This information was retrieved from the following policy statements.)

- 管理ポリシー AWSLambdaBasicExecutionRole-..., ステートメント 0
- 管理ポリシー AWSLambdaBasicExecutionRole-..., ステートメント 1

「ポリシーをアタッチします」をクリックします。

The screenshot shows the 'Policy Attachments' page in the AWS IAM console. The role name is 'amazonconnect-status-confirm-role'. The 'Permissions policies' section shows a list of policies. A red box highlights the 'ポリシーをアタッチします' (Attach policy) button. The table below shows the following policy:

ポリシー名	ポリシータイプ
AWSLambdaBasicExecutionRole-...	管理ポリシー

Below the table, there is a section for 'Permissions boundary (not set)' and a section for 'CloudTrail イベントに基づいてポリシーを生成' (Generate policy based on CloudTrail events).

The 'CloudTrail イベントに基づいてポリシーを生成' section contains the following text:


この ロールの アクセスアクティビティに基づいて新しいポリシーを生成し、カスタマイズおよび作成したり、このロールにアタッチしたりできます。AWS は CloudTrail イベントを使用して、使用されるサービスとアクションを識別し、ポリシーを生成します。 [詳細はこちら](#)


Share your [feedback](#) and help us improve the policy generation experience.


[ポリシーの生成](#)

過去 7 日間におけるポリシー生成リクエストはありません。

“connect”で検索し、「AmazonConnect_FullAccess」にチェックを入れます。


amazonconnect-status-confirm-role- にアクセス権限を追加する
アクセス権限をアタッチする

ポリシーの作成 


ポリシーのフィルタ  10 件の結果を表示中


	ポリシー名	タイプ	次として使用
<input checked="" type="checkbox"/>	AmazonConnect_FullAccess	AWS による管理	Permissions policy (3)
<input type="checkbox"/>	AmazonConnectReadOnlyAccess	AWS による管理	なし
<input type="checkbox"/>	AmazonConnectVoiceIDFullAccess	AWS による管理	なし
<input type="checkbox"/>	AmazonMSKConnectReadOnlyAccess	AWS による管理	なし
<input type="checkbox"/>	AWSConnector	AWS による管理	なし
<input type="checkbox"/>	AWSDirectConnectFullAccess	AWS による管理	なし
<input type="checkbox"/>	AWSDirectConnectReadOnlyAccess	AWS による管理	なし
<input type="checkbox"/>	EC2InstanceConnect	AWS による管理	なし
<input type="checkbox"/>	ServerMigrationConnector	AWS による管理	なし
<input type="checkbox"/>	VMImportExportRoleForAWSConnector	AWS による管理	なし


キャンセル **ポリシーのアタッチ**

フィードバック 日本語  © 2008 - 2021, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。 プライバシーポリシー 利用規約 Cookie の設定

“sqs”で検索し、「AmazonSQSFullAccess」にチェックを入れます。


amazonconnect-status-confirm-role- にアクセス権限を追加する
アクセス権限をアタッチする

ポリシーの作成 

ポリシーのフィルタ  3 件の結果を表示中

	ポリシー名	タイプ	次として使用
<input checked="" type="checkbox"/>	AmazonSQSFullAccess	AWS による管理	Permissions policy (1)
<input type="checkbox"/>	AmazonSQSReadOnlyAccess	AWS による管理	なし
<input type="checkbox"/>	AWSLambdaSQSQueueExecutionRole	AWS による管理	なし

キャンセル **ポリシーのアタッチ**

フィードバック 日本語  © 2008 - 2021, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。 プライバシーポリシー 利用規約 Cookie の設定

“dynamo”で検索し、「AmazonDynamoDBFullAccess」にチェックを入れ、「ポリシーのアタッチ」をクリックします。

amazonconnect-status-confirm-role- にアクセス権限を追加する
アクセス権限をアタッチする

ポリシーの作成

ポリシーのフィルタ 4件の結果を表示中

ポリシー名	タイプ	次として使用
<input checked="" type="checkbox"/> AmazonDynamoDBFullAccess	AWS による管理	Permissions policy (2)
<input type="checkbox"/> AmazonDynamoDBReadOnlyAccess	AWS による管理	なし
<input type="checkbox"/> AWSLambdaDynamoDBExecutionRole	AWS による管理	なし
<input type="checkbox"/> AWSLambdaInvocation-DynamoDB	AWS による管理	なし

キャンセル **ポリシーのアタッチ**

フィードバック 日本語 ▼ © 2008 - 2021, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。 プライバシーポリシー 利用規約 Cookie の設定

ポリシーがアタッチされたことを確認します。

Identity and Access Management (IAM)

ダッシュボード

- アクセス管理
 - ユーザーグループ
 - ユーザー
 - ロール**
 - ポリシー
 - ID プロバイダー
 - アカウント設定
- アクセスレポート
 - アクセスアナライザー
 - アーカイブルール
 - アナライザー
 - 設定
 - 認証情報レポート
 - 組織アクティビティ
 - サービスコントロールポリシー (SCP)

IAM の検索

AWS アカウント ID: 206013296236

ロール > amazonconnect-status-confirm-role- 概要

ロールの削除

ロール ARN: arn:aws:iam:::role/service-role/amazonconnect-status-confirm-role-
ロールの説明: [編集](#)
インスタンスプロファイル ARN: [編集](#)
パス: /service-role/
作成時刻:
最後のアクティビティ:
最大セッション時間: 1 時間 [編集](#)

アクセス権限 信頼関係 タグ アクセスアドバイザー セッションの無効化

Permissions policies (4 適用済みポリシー)

[ポリシーをアタッチします](#) [インラインポリシーの追加](#)

ポリシー名	ポリシータイプ	
<input checked="" type="checkbox"/> AmazonConnect_FullAccess	AWS 管理ポリシー	✕
<input checked="" type="checkbox"/> AmazonDynamoDBFullAccess	AWS 管理ポリシー	✕
<input checked="" type="checkbox"/> AmazonSQSFullAccess	AWS 管理ポリシー	✕
<input checked="" type="checkbox"/> AWSLambdaBasicExecutionRole-	管理ポリシー	✕

Permissions boundary (not set)

CloudTrail イベントに基づいてポリシーを生成

このロールのアクセスアクティビティに基づいて新しいポリシーを生成し、カスタマイズおよび作成したり、このロールにアタッチしたりできます。AWS は CloudTrail イベントを使用して、使用されるサービスとアクションを識別し、ポリシーを生成します。 [詳細はこちら](#)

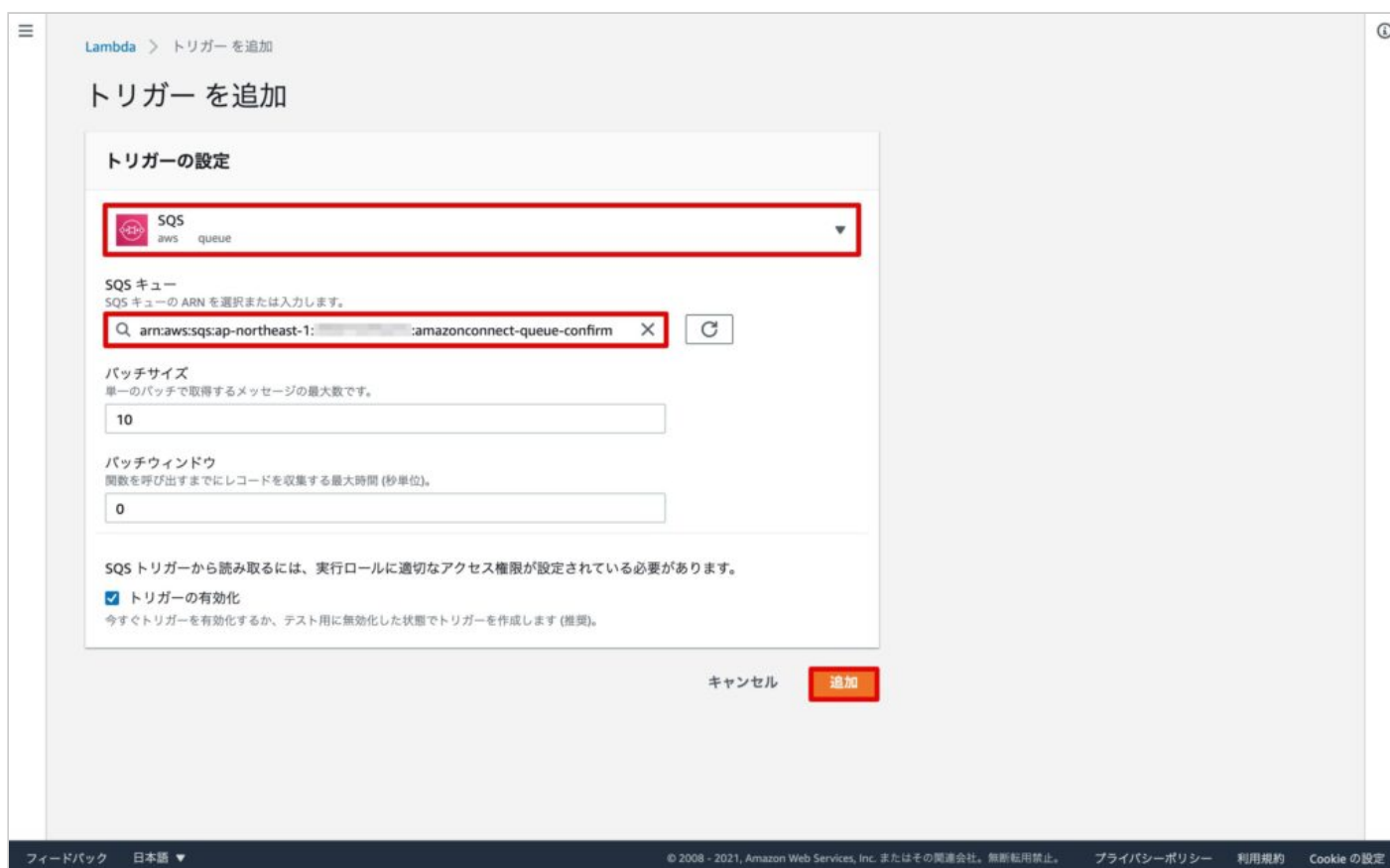
フィードバック 日本語 ▼ © 2008 - 2021, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。 プライバシーポリシー 利用規約 Cookie の設定

トリガーの追加

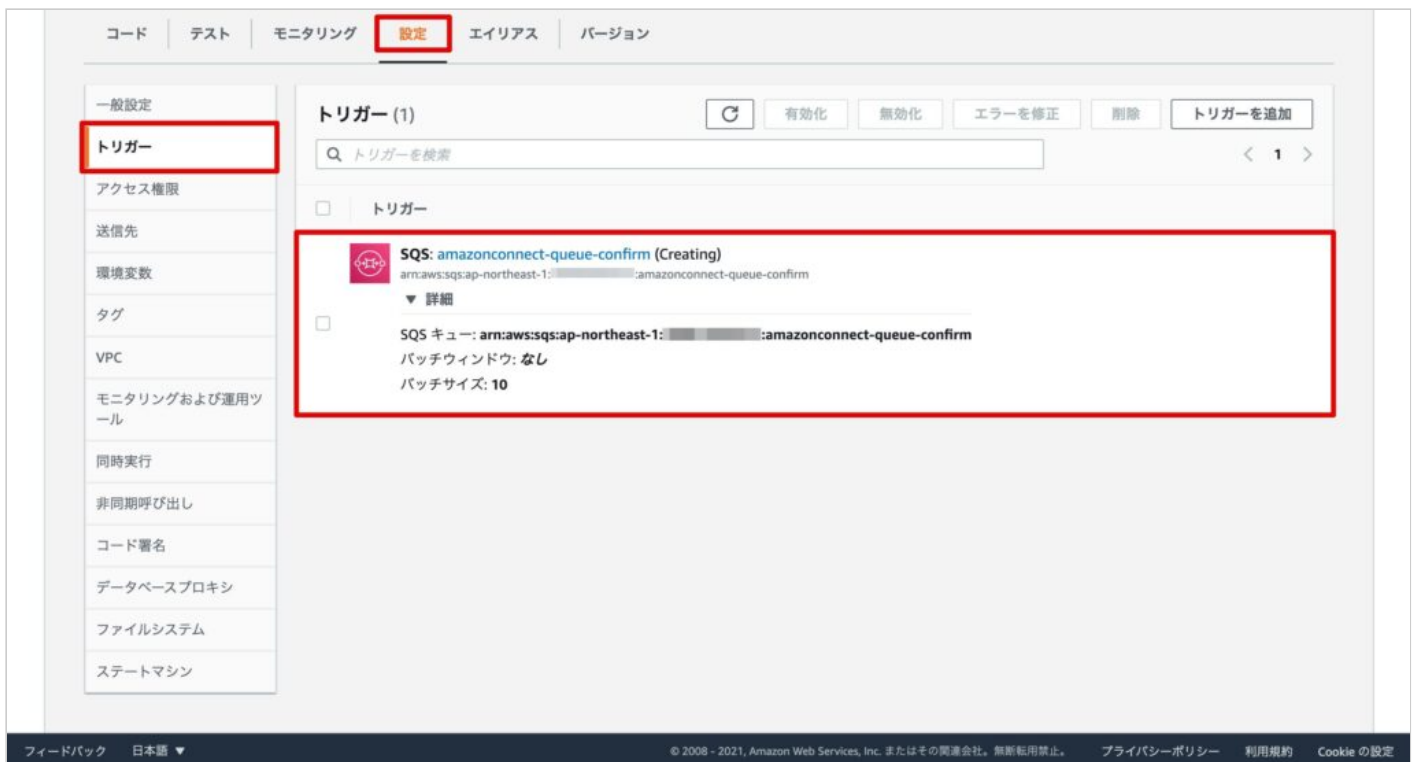
「トリガーを追加」をクリックします。



作成したSQS_2を選択し、「追加」をクリックします。



トリガーが追加されていることを確認します。（設定タブ→トリガー）



コードの記述

Lambdaのコードを記述します。

```
-----  
import json  
import boto3  
from boto3.dynamodb.conditions import Key, Attr  
  
# boto3からDynamoDBへアクセスするためのオブジェクトを取得  
dynamodb = boto3.resource('dynamodb')  
  
# "amazonconnect-contact-list"へアクセスするためのオブジェクトを取得  
contactlist = dynamodb.Table("amazonconnect-contact-list")  
  
# "amazonconnect-response-status"へアクセスするためのオブジェクトを取得  
responsestatus = dynamodb.Table("amazonconnect-response-status")  
  
# "amazonconnect-response-status"の内容を返す関数  
def status_scan():  
  
    scanData = responsestatus.scan()  
    items=scanData['Items']  
  
    return scanData["Items"]
```

"amazonconnect-contact-list"の内容を返す関数

def operation_scan():

scanData = contactlist.scan()

items=scanData['Items']

return scanData["Items"]

電話番号リストの行数を返す関数

def get_contactinfo_item_number(json_contactinfo):

count = 0

for line in json_contactinfo:

count += 1

return count

前回のプライオリティを返す関数

def get_before_priority(sqs_event):

record = sqs_event['Records'][0]

json_record = json.loads(record["body"])

priority = json_record["priority"]

return priority

指定されたプライオリティの電話番号を返す関数

def get_phone_number(json_contactinfo, now_priority):

for line in json_contactinfo:

if line['Priority']==now_priority:

phone_number = line['Phone']

return phone_number


```
def lambda_handler(event, context):

    # 応答結果を確認
    statustinfo = status_scan()

    # 応答結果がNGの場合
    if statustinfo[0]['CallStatus'] == "NG":

        # 電話番号リストを取得
        contactinfo = operation_scan()

        # 電話番号リストの行数を取得
        contactinfo_item_number = get_contactinfo_item_number(contactinfo)

        # 前回のプライオリティを取得
        before_priority = get_before_priority(event)

        # 前回のプライオリティが電話番号リストの行数と同じ場合（リストの最後まで通知した
        # 場合）
        if before_priority == contactinfo_item_number:

            # プライオリティを1に設定（最初に戻る）
            priority = 1

        # 前回のプライオリティが電話番号リストの行数と異なる場合（リストの最後まで通知し
        # ていない場合）
        else:

            # プライオリティに1を足す（次のプライオリティを設定）
            priority = before_priority + 1

        # 指定したプライオリティの電話番号を取得
        phone_number = get_phone_number(contactinfo, priority)

        # boto3からAmazonConnectへアクセスするためのオブジェクトを取得
        connect = boto3.client('connect')

        # AmazonConnectの問い合わせフローを呼び出し電話発信
```

```
connect.start_outbound_voice_contact(  
    DestinationPhoneNumber=phone_number,  
    ContactFlowId='*****',  
    InstanceId='*****',  
    SourcePhoneNumber='+81*****',  
)  
  
# boto3からSQSへアクセスするためのオブジェクトを取得  
sqs = boto3.resource('sqs')  
  
# "amazonconnect-queue-confirm"へアクセスするためのオブジェクトを取得  
name = 'amazonconnect-queue-confirm'  
queue = sqs.get_queue_by_name(QueueName=name)  
  
# 現在のプライオリティをセットしてステータス確認用のキューにメッセージ送信  
response = queue.send_message(MessageBody=json.dumps({"priority": priority}))
```

ContactFlowId: 問い合わせフローID

InstanceId: インスタンスID

SourcePhoneNumber: 発信元の電話番号

※国番号をつけて記述（日本の050の番号の場合、+8150*****）

```
File Edit Find View Go Tools Window Test Deploy Changes deployed
Go to Anything (⌘P)
Environment
amazonconnect-status-confirm
lambda_function.py

1 import json
2 import boto3
3 from boto3.dynamodb.conditions import Key, Attr
4
5 # boto3からDynamoDBへアクセスするためのオブジェクトを取得
6 dynamodb = boto3.resource('dynamodb')
7
8 # "amazonconnect-contact-list"へアクセスするためのオブジェクトを取得
9 contactlist = dynamodb.Table("amazonconnect-contact-list")
10
11 # "amazonconnect-response-status"へアクセスするためのオブジェクトを取得
12 responsestatus = dynamodb.Table("amazonconnect-response-status")
13
14 # "amazonconnect-response-status"の内容を返す関数
15 def status_scan():
16
17     scanData = responsestatus.scan()
18     items = scanData['Items']
19
20     return scanData["Items"]
21
22
23 # "amazonconnect-contact-list"の内容を返す関数
24 def operation_scan():
25
26     scanData = contactlist.scan()
27     items = scanData['Items']
28
29     return scanData["Items"]
30
31
32 # 電話番号リストの行数を返す関数
33 def get_contactinfo_item_number(json_contactinfo):
34
35     count = 0
36
37     for line in json_contactinfo:
38         count += 1
39
40     return count
41
42
43 # 前回のプライオリティを返す関数
44 def get_before_priority(sqs_event):
45
46     record = sqs_event['Records'][0]
47     json_record = json.loads(record['body'])
48     priority = json_record['priority']
49
50     return priority
51
52
53 # 指定されたプライオリティの電話番号を返す関数
54 def get_phone_number(json_contactinfo, now_priority):
55
56     for line in json_contactinfo:
57         if line['Priority'] == now_priority:
58             phone_number = line['Phone']
59
60     return phone_number
61
62
63 def lambda_handler(event, context):
64
65     # 応答結果を確認
66     statustinfo = status_scan()
67
68     # 応答結果がNGの場合
69     if statustinfo[0]['CallStatus'] == "NG":
70
71         # 電話番号リストを取得
72         contactinfo = operation_scan()
73
74         # 電話番号リストの行数を取得
75         contactinfo_item_number = get_contactinfo_item_number(contactinfo)
76
77         # 前回のプライオリティを取得
78         before_priority = get_before_priority(event)
79
80         # 前回のプライオリティが電話番号リストの行数と同じ場合(リストの最後まで通知した場合)
81         if before_priority == contactinfo_item_number:
82
83             # プライオリティを1に設定(最初に戻る)
84             priority = 1
85
86         # 前回のプライオリティが電話番号リストの行数と異なる場合(リストの最後まで通知していない場合)
87         else:
88
89             # プライオリティに1を足す(次のプライオリティを設定)
90             priority = before_priority + 1
91
92         # 指定したプライオリティの電話番号を取得
93         phone_number = get_phone_number(contactinfo, priority)
94
95         # boto3からAmazonConnectへアクセスするためのオブジェクトを取得
96         connect = boto3.client('connect')
97
98         # AmazonConnectの問い合わせフローを呼び出し電話発信
99         connect.start_outbound_voice_contact(
100             DestinationPhoneNumber=phone_number,
101             ContactFlowId='XXXXXXXXXXXXXXXXXXXX',
102             InstanceId='XXXXXXXXXXXXXXXXXXXX',
103             SourcePhoneNumber='+81XXXXXXXXXXXX',
104         )
105
106         # boto3からSQSへアクセスするためのオブジェクトを取得
107         sqs = boto3.resource('sqs')
108
109         # "amazonconnect-queue-confirm"へアクセスするためのオブジェクトを取得
110         name = 'amazonconnect-queue-confirm'
111         queue = sqs.get_queue_by_name(QueueName=name)
112
113         # 現在のプライオリティをセットしてステータス確認用のキューにメッセージ送信
114         response = queue.send_message(MessageBody=json.dumps({'priority': priority}))
115
```

以上で、AmazonConnectによる自動電話通知（7.複数連絡先への電話通知〈構築⑧〉）の説明は完了です！