

Lambda経由でECインスタンスのstatusを
slackに通知

キム テフン

jkdnekki@naver.com

目次

■ EC2StatusをAWS Lambda経由でslack通知

- ・ 概要
- ・ IAM roleを作成
- ・ AWS Lambdaを作成
- ・ CloudWatchを作成
- ・ 応用編

概要

前回の課題時にやってみたいと思った「EC2ステータスの変化をSlackに通知」を実現しました。

EC2インスタンスのステータスがRunningとStoppedになった時にCloudWatchがそれを検知し、ステータス変更のメッセージがLambda経由でslackに通知する構成となります。



IAM roleを作成

IAMとは

Identity and Access Managementは、AWSリソースへのアクセス権限を設定するサービスです。今回はEC2のステータスの情報アクセス権限が必要なので、Read権限を設定します。

The screenshot shows the AWS IAM console interface. On the left, a sidebar menu is open under the 'Access management' section, with 'Role' selected (marked by a red box labeled '1'). In the main content area, there are two tabs: 'Role creation' (marked by a red box labeled '2') and 'Delete role'. A large red box highlights the 'Role creation' tab. Below it, a table lists existing roles. The table has columns for 'Role name', 'Trusted entity', and 'Last activity'. One row is highlighted with a red box and labeled 'Role selection after role creation click'.

Role name	Trusted entity	Last activity
AWSServiceRoleForAmazonGuardDuty	AWS サービス: guardduty (サービスにリンク...)	昨日
AWSServiceRoleForCloud9	AWS サービス: cloud9 (サービスにリンクさ...)	今日
AWSServiceRoleForCloudWatchCrossAccount	AWS サービス: cloudwatch-crossaccount (サ...)	21 日間
AWSServiceRoleForElasticLoadBalancing	AWS サービス: elasticloadbalancing (サービ...)	18 日間
AWSServiceRoleForOrganizations	AWS サービス: organizations (サービスにリ...)	なし
AWSServiceRoleForRDS	AWS サービス: rds (サービスにリンクされた...)	17 日間
AWSServiceRoleForSupport	AWS サービス: support (サービスにリンクさ...)	なし
AWSServiceRoleForTrustedAdvisor	AWS サービス: trustedadvisor (サービスにリ...)	なし
botSampleRole	AWS サービス: lambda	今日

IAM roleを作成

□ ロールの作成

信頼されたエンティティの種類を選択

1 2 3 4

AWS サービス EC2、Lambda、およびその他
別の AWS アカウント お客様またはサードパーティに属しています
ウェブ ID Cognito または任意の OpenID プロバイダ
SAML 2.0 フェデレーション 企業ディレクトリ

AWS のサービスによるアクションの代行を許可します。 詳細は[こちら](#)

このロールを使用するサービスを選択

EC2 Allows EC2 instances to call AWS services on your behalf.

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

API Gateway CodeBuild EKS Kinesis S3
AWS Backup CodeDeploy EMR Lambda SMS
AWS Chatbot CodeStar Notifications ElastiCache Lex SNS
AWS Support Comprehend Elastic Beanstalk License Manager SWF
Amplify Config Elastic Container Service Machine Learning SageMaker
AppStream 2.0 Connect Elastic Transcoder Macie Security Hub
AppSync DMS Elastic Load Balancing MediaConvert Service Catalog
Application Auto Scaling Data Lifecycle Manager Forecast Migration Hub Step Functions
Application Discovery Service Data Pipeline Global Accelerator OpsWorks Storage Gateway
DataSync Glue Personalize Textract

□ ロールの作成

1 2 3 4

▼ Attach アクセス権限ポリシー

新しいロールにアタッチするポリシーを 1 つ以上選択します。

ポリシーの作成

ポリシーのフィルタ ▾ 検索 ec2read 1 件の結果を表示中

ポリシー名 次として使用

AmazonEC2ReadOnlyAccess Permissions policy (1)

Lambdaサービスを選択後→EC2ReadOnlyAccessを検索→選択

IAM roleを作成

1 2 3 4

ロールの作成

確認

以下に必要な情報を指定してこのロールを見直してから、作成してください。

任意のロール名を入力後ロールの作成をクリック

ロール名*

LambdaEc2readRo|

英数字と「+=,@-_」を使用します。最大 64 文字。

ロールの説明

Allows Lambda functions to call AWS services on your behalf.

最大 1000 文字。英数字と「+=,@-_」を使用します。

信頼されたエンティティ AWS のサービス: lambda.amazonaws.com

ポリシー



AmazonEC2ReadOnlyAccess

アクセス権限の境界 アクセス権限の境界が設定されていません

追加されたタグはありません。

* ロールの作成後、Lambdaの作成に進みます

* 必須

キャンセル

戻る

ロールの作成

Lambdaを作成

AWS Lambdaとは

AWSでeventが発生した時に処理するコンソール環境のようなものです。
サーバー管理を気にせず、コードをAWSにアップロードすることで色々なAWSサービスを自動的に制御することができます。

CloudWatchのtriggerとなるLambda関数を作成します。
この関数によってCloudWatchが検知したEC2ステータスをSlackに送ることが可能になります。

The screenshot shows the AWS Lambda console interface. On the left, there's a sidebar with links: ダッシュボード, アプリケーション, **関数**, and Layers. The main area is titled "Lambda > 関数" and shows a table with 5 rows. The columns are labeled: 関数名 (Function Name), 説明 (Description), ランタイム (Runtime), コードサイズ (Code Size), and 最終更新日時 (Last Updated). A red box highlights the "関数の作成" (Create Function) button in the top right corner of the header. Another red box highlights the text "AWSLambda → 関数の作成クリック" (Click to create a function).

Lambdaを作成

下記のように任意の関数名を入力して作成します。

- Python3.6と既存のロール選択ご関数の作成をクリック

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。

1 instance_status_check

半角英数字、ハイphen、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 情報
関数を記述する言語を選択します。

2 Python 3.6

▼

アクセス権限 情報
Lambda は、Amazon CloudWatch Logs にログをアップロードするアクセス権限を持つ実行ロールを作成します。トリガーを追加すると、アクセス権限をさらに設定および変更できます。

▼ 実行ロールの選択または作成

実行ロール
関数のアクセス許可を定義するロールを選択します。カスタムロールを作成するには、[IAM コンソール](#)に移動します。

3 基本的な Lambda アクセス権限で新しいロールを作成
 既存のロールを使用する
 AWS ポリシーテンプレートから新しいロールを作成

既存のロール
この Lambda 関数で使用するために作成した既存のロールを選択します。このロールには、Amazon CloudWatch Logs にログをアップロードするアクセス許可が必要です。

4 LambdaEc2readRole

IAM コンソールで [LambdaEc2readRoleロールを表示](#)します。

▼ G

キャンセル 関数の作成

* Lambda関数の作成が終、Slackの設定に進めます。

Lambdaを作成

Slack設定

自分のslack apiで下記のようにappを作成します。

slack api <https://api.slack.com/apps>

The screenshot shows the Slack API App creation interface. On the left, a modal window titled 'Create a Slack App' is open, showing fields for 'App Name' (set to 'ec_instance_check') and 'Development Slack Workspace' (set to 'SH-TALK'). Below these are terms of service and a 'Create App' button. On the right, the main 'Your Apps' page lists the created app. The 'Incoming Webhooks' tab is selected, showing a table with one row:

Webhook URL	Channel	Added By
https://hooks.slack.com/services/TM... Copy	#simsimhada	kth Dec 5, 2019

A red box highlights the 'Copy' button next to the webhook URL, with a callout pointing to it labeled 'Lambda関数に使うのでコピー'. Another red box highlights the 'Activate Incoming Webhooks' switch, which is turned 'On'. A third red box highlights the 'Request to Allow' button at the bottom of the page, labeled 'リクエストを許可するをクリック'.

Slack appをチャンネルで使うため、Webhookの設定をします

Activate Incoming Webhooks

Incoming webhooks are a simple way to post messages from external sources into Slack. They make use of normal HTTP requests with a JSON payload, which includes the message and a few other optional details. You can include message attachments to display richly-formatted messages.

Each time your app is installed, a new Webhook URL will be generated.

If you deactivate incoming webhooks, new Webhook URLs will not be generated.

ec_instance_check が SH-TALK Slack ワークスペースにアクセスする権限をリクエストしています

チャンネル

general
sh-talk
simsimhada
talktalk

ダイレクトメッセージ

@botsampletest アプリ

チャンネルを検索...

キャンセル 許可する

Lambdaを作成

pythonコード作成

AWS Lambdaメニューに戻り下記のようにpythonコードを作成します。

- urllibとboto3を使って、ec2のステータスをslackで指定したチャンネルにメッセージを送信

```
import urllib
import boto3

def lambda_handler(event, context):

    endpoint = https://hooks.slack.com/xxxxxxxxxx #slackのwebhook url
    ec2 = boto3.client('ec2')

    #ec2 status
    ec2state= ec2.describe_instances(
        Filters=[{'Name':'instance-id', 'Values':['checkするインスタンスID']}]
    )["Reservations"][0]["Instances"][0]['State']['Name']

    if ec2state == 'stopped':
        print("stop")
        state_param = urllib.parse.urlencode({"payload": {"text": "ec2インスタンスが停止" }})

    elif ec2state == 'running':

        state_param = urllib.parse.urlencode({"payload": {"text": "ec2インスタンスが起動！" }})

    param = state_param.encode("utf-8")

    function = urllib.request.urlopen(endpoint, param)

    print(function.read())

    return
```

Lambdaを作成

pythonコード作成

instance_status_check

実行結果: 成功 (ログ)
▶ 詳細

スロットリング 限定条件 ▾ アクション ▾ mytest テスト 保存

テストして正常に出来るのか確認 ×

設定 アクセス権限 モニタリング

* 問題なければ、CloudWatch設定に進みます。



コードエントリ タイプ
コードをオンラインで編集

ランタイム
Python 3.6

ハンドラ 情報
lambda_function.lambda_handler

File Edit Find View Go Tools Window

Environment

instance_state_checker λ lambda_function

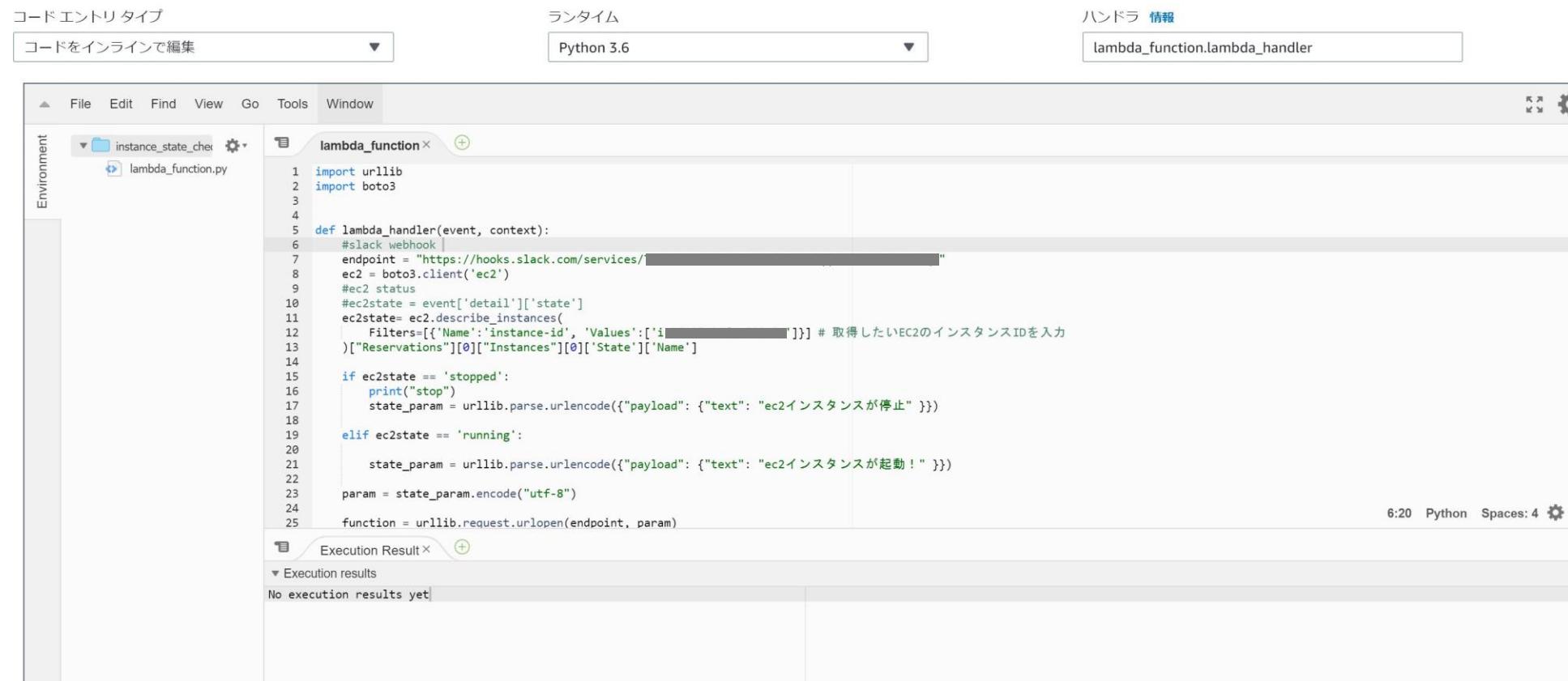
lambda_function.py

```
1 import urllib
2 import boto3
3
4
5 def lambda_handler(event, context):
6     #slack webhook
7     endpoint = "https://hooks.slack.com/services/[REDACTED]"
8     ec2 = boto3.client('ec2')
9     #ec2 status
10    #ec2state = event['detail']['state']
11    #ec2states= ec2.describe_instances(
12    #    Filters=[{'Name': 'instance-id', 'Values': ['[REDACTED]']}]) # 取得したいEC2のインスタンスIDを入力
13    #)[("Reservations")[0]["Instances"][0]['State']['Name']]
14
15    if ec2state == 'stopped':
16        print("stop")
17        state_param = urllib.parse.urlencode({"payload": {"text": "ec2インスタンスが停止" }})
18
19    elif ec2state == 'running':
20
21        state_param = urllib.parse.urlencode({"payload": {"text": "ec2インスタンスが起動！" }})
22
23    param = state_param.encode("utf-8")
24
25    function = urllib.request.urlopen(endpoint, param)
```

Execution Result

No execution results yet!

6:20 Python Spaces: 4



CloudWatchを設定

ルールの作成

CloudWatch

ダッシュボード

アラーム

アラーム

不足

OK

請求

ログ

ロググループ

インサイト

メトリクス

イベント

ルール

イベントバス

ServiceLens 新規

Service Map

Traces

Contributor Insights 新規

設定 新規

お気に入り

ルール

ルールにより、選:

ルールの作成

ルールのクリックして次の画面で詳細の設定を行います。

* ルールを作成してEC2のステータスを検知出来るようにします。

ルールの定義

名前*

ec2_status_check

説明

状態 有効化

CloudWatch Events はターゲットに必要な権限を追加し、このルールがトリガーされたときに呼び出せるようにします。

* 必須

キャンセル

戻る

ルールの作成

CloudWatchを設定

ルールの作成

サービス名はEC2を選択、イベントタイプは下記のようにEC2 Instance State-change Notificationにします。

イベントソース

イベントパターンを構築またはカスタマイズする、またはスケジュールを設定してターゲットを呼び出します。

イベントパターン タイプ スケジュール

サービス別に一致するイベントパターンの構築

サービス名 1 EC2

イベントタイプ 2 EC2 Instance State-change Notification

任意の状態

特定の状態

running stopped

任意のインスタンス

特定のインスタンス ID

i-0

4

インスタンスIDはチェックしたインスタンスのIDを入れます。

ターゲット

イベントがイベントパターンに一致するか、スケジュールがトリガーされたときに呼び出すターゲットを選択します。

Lambda 関数

関数* instance_status_check

バージョン / エイリアスの設定

入力の設定

+ ターゲットの追加 *

5

作った既存のLambda関数を選択します。

*以上で設定はおわります。次はEC2インスタンスを起動と停止させてslackにちゃんと通知が来るのかを確認します。

CloudWatchを設定

動作確認

以下のように通知が来ることを確認する。

 **ec_instance_check** アプリ 04:20
ec2インスタンスが起動！

 **ec_instance_check** アプリ 04:25
ec2インスタンスが停止

定期的にEC2instanceを操作する場合(追加)

Lambda関数を作成する時にCloudWatch Eventsをトリガーとして追加することで cronの設定も可能です。

-EC2instance以外にRDSなどの起動、停止も簡単に定期的に自動することが可能です。設定手順は次のようになります。

CloudWatchを設定

定期的にEC2instanceを操作する場合(手順)

Lambda関数の作成で、以下の通り選択します。ロールは既存のロール(PowerUser)を使います。

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。
半角英数字、ハイphen、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 情報
関数を記述する言語を選択します。

アクセス権限 情報
Lambda は、Amazon CloudWatch Logs にログをアップロードするアクセス権限を持つ実行ロールを作成します。トリガーを追加すると、アクセス権限をさらに設定および変更できます。

▼ 実行ロールの選択または作成

実行ロール
関数のアクセス許可を定義するロールを選択します。カスタムロールを作成するには、[IAM コンソール](#)に移動します。

基本的な Lambda アクセス権限で新しいロールを作成
 既存のロールを使用する
 AWS ポリシーテンプレートから新しいロールを作成

既存のロール
この Lambda 関数で使用するために作成した既存のロールを選択します。このロールには、Amazon CloudWatch Logs にログをアップロードするアクセス許可が必要です。
 IAM コンソールで [slack_Control_role ロールを表示](#) します。

[キャンセル](#) [関数の作成](#)

CloudWatchを設定

定期的にEC2instanceを操作する場合(手順)

トリガーを追加をクリックし、cronの設定をします。

The screenshot shows two parts of the AWS Lambda function configuration interface:

- Left Panel (Function Configuration):** Shows the function name "ec2_auto_func". A red box highlights the "Trigger" button under the "Designer" tab.
- Right Panel (Trigger Configuration):** Shows the "Trigger settings" screen for CloudWatch Events. A red box highlights the "Create new rule" button. Below it, a red box highlights the "Name" input field. Further down, a red box highlights the "Schedule type" section with the "Scheduled" radio button selected. A red box also highlights the "Schedule expression" input field where the cron expression "cron(0 8 * * ? *)" is entered.

cron設定は下記のようにします。

cron(分 時 日 月 週 年)
ex.毎月1日午前8時にRUNするように設定したい場合 : **cron(0 8 1 * ? *)**

参考:<https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/ScheduledEvents.html>

CloudWatchを設定

定期的にEC2instanceを操作する場合(手順)

Lambda関数のpython コードは下記のようになります。

・参考:https://aws.amazon.com/premiumsupport/knowledge-center/start-stop-lambda-cloudwatch/?nc1=h_ls

instanceを停止

```
import boto3

region = 'ap-northeast-1'
instances = ['i-0fxxxxxxxxxxxx', 'i-0f3xxxxxxxxxx']

def lambda_handler(event, context):
    ec2 = boto3.client('ec2', region_name=region)
    ec2.stop_instances(InstanceIds=instances)
    print('stopped インスタンス:' + str(instances))
```

instanceを起動

```
import boto3

region = 'ap-northeast-1'
instances = ['i-0fxxxxxxxxxxxx', 'i-0f3xxxxxxxxxx']

def lambda_handler(event, context):
    ec2 = boto3.client('ec2', region_name=region)
    ec2.start_instances(InstanceIds=instances)
    print('started インスタンス: ' + str(instances))
```

設定が上手く出来たら以下ののようにEC2通知アプリ(ec_instance_check)からメッセージがきます。

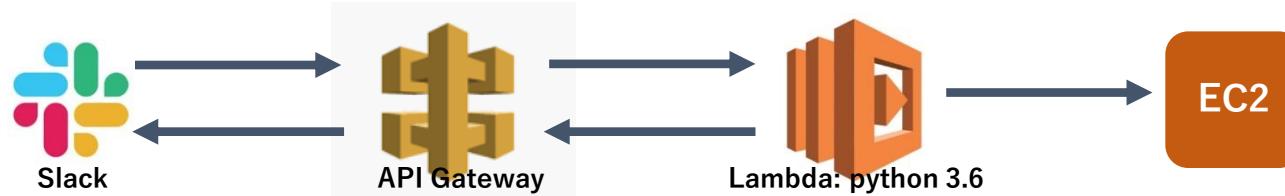


| ログ
ロググループ
インサイト
メトリクス
イベント
ルール
イベントバス
ServiceLens 新規
Service Map
Traces

イベントのフィルター	
時間 (UTC +00:00)	メッセージ
すべて 2019-12-09 (19:40:50) ▾	
2019-12-10	いまのところ古いイベントはありません。再試行。
▼ 19:00:49	START RequestId: 59a93a0d-742a-4489-b9dd-519c699ec8a5 Version: \$LATEST
	START RequestId: 59a93a0d-742a-4489-b9dd-519c699ec8a5 Version: \$LATEST
▼ 19:00:51	stopped インスタンス:['i-0f7a55544ec3aff1e', 'i-0f3bad5b5cc6ea73f']
	stopped インスタンス:['i-0f7a55544ec3aff1e', 'i-0f3bad5b5cc6ea73f']

応用編

今回はSlackのコマンドラインからEC2 instanceを操作できるようにします。



手順

1. Lambda関数の作成(既存のロール(PowerUserAccess)を使用)
2. API Gatewayの追加と設定

The screenshot shows two side-by-side configuration dialogs.

Left Dialog: トリガーを追加 (Add Trigger)

- トリガーの設定 (Trigger settings):** Shows a dropdown menu with "API Gateway" selected, and other options like "api", "application-services", and "aws serverless".
- 説明 (Description):** Text explaining how to set up proxy integration between API Gateway and Lambda.
- API (API):** A dropdown menu with "既存の API を選択するか、新しい API を作成します。" (Select an existing API or create a new one). This menu is highlighted with a red box.
- Template (Template):** Two options:
 - HTTP API Beta: Ideal for building REST APIs that proxy to Lambda functions.
 - REST API: Ideal for building REST APIs where you want full control over the request and response with API management capabilities.

Right Dialog: セキュリティ (Security)

- セキュリティ (Security):** Text stating "API エンドポイントのセキュリティメカニズムを設定します。" (Set security mechanisms for the API endpoint). A "Open" button is highlighted with a red box.
- 警告 (Warning):** Text stating "警告: 作成した API エンドポイントは公開され、すべてのユーザーが呼び出せるようになります。" (Warning: The created API endpoint will be publicly accessible and can be called by all users).
- 追加の設定 (Additional settings):**
 - API 名 (API Name):** "ec2_control_bot-API" is entered in the input field, highlighted with a red box.
 - デプロイされるステージ (Deployment stage):** "stage" is entered in the input field, highlighted with a red box.

応用編

手順

3. API Gatewayで詳細の設定をする

The screenshot shows the AWS Lambda function configuration interface for the function 'ec2_control_bot'. The top navigation bar includes tabs for 'スロットリング' (Slot Rerouting), '限定条件' (Request Restriction), 'アクション' (Action), 'テストイベントの選択' (Select Test Event), 'テスト' (Test), and '保存' (Save). Below the tabs, there's a section titled 'Designer' which displays the function's configuration. A large box labeled 'ec2_control_bot' contains an icon of a Lambda function and a 'Layers' section with '(0)'. Below this box is a section for 'API Gateway' which has a blue border, indicating it is selected or active. This section includes a 'Layers' dropdown, a 'Layers' button, and a 'Add destination' button. At the bottom left of this section is a '+ トリガーを追加' (Add Trigger) button. The bottom part of the interface is a summary section titled 'API Gateway' containing the following information:

ec2_control_bot-API	← クリックするとAPI Gateway設定メニューに移動します。	有効	削除
arn:aws:execute-api:ap-northeast-1:174906440569:6xdklc845a/*/*/ec2_control_bot			
▶ API: api-gateway/6xdklc845a/*/*/ec2_control_bot API Type: rest API エンドポイント: https://6xdklc845a.execute-api.ap-northeast-1.amazonaws.com/stage/ec2_control_bot			

ec2_control_bot-API ← クリックするとAPI Gateway設定メニューに移動します。

有効

削除

arn:aws:execute-api:ap-northeast-1:174906440569:6xdklc845a/*/*/ec2_control_bot

▶ API: [api-gateway/6xdklc845a/*/*/ec2_control_bot](https://6xdklc845a.execute-api.ap-northeast-1.amazonaws.com/stage/ec2_control_bot) API Type: rest API エンドポイント: https://6xdklc845a.execute-api.ap-northeast-1.amazonaws.com/stage/ec2_control_bot

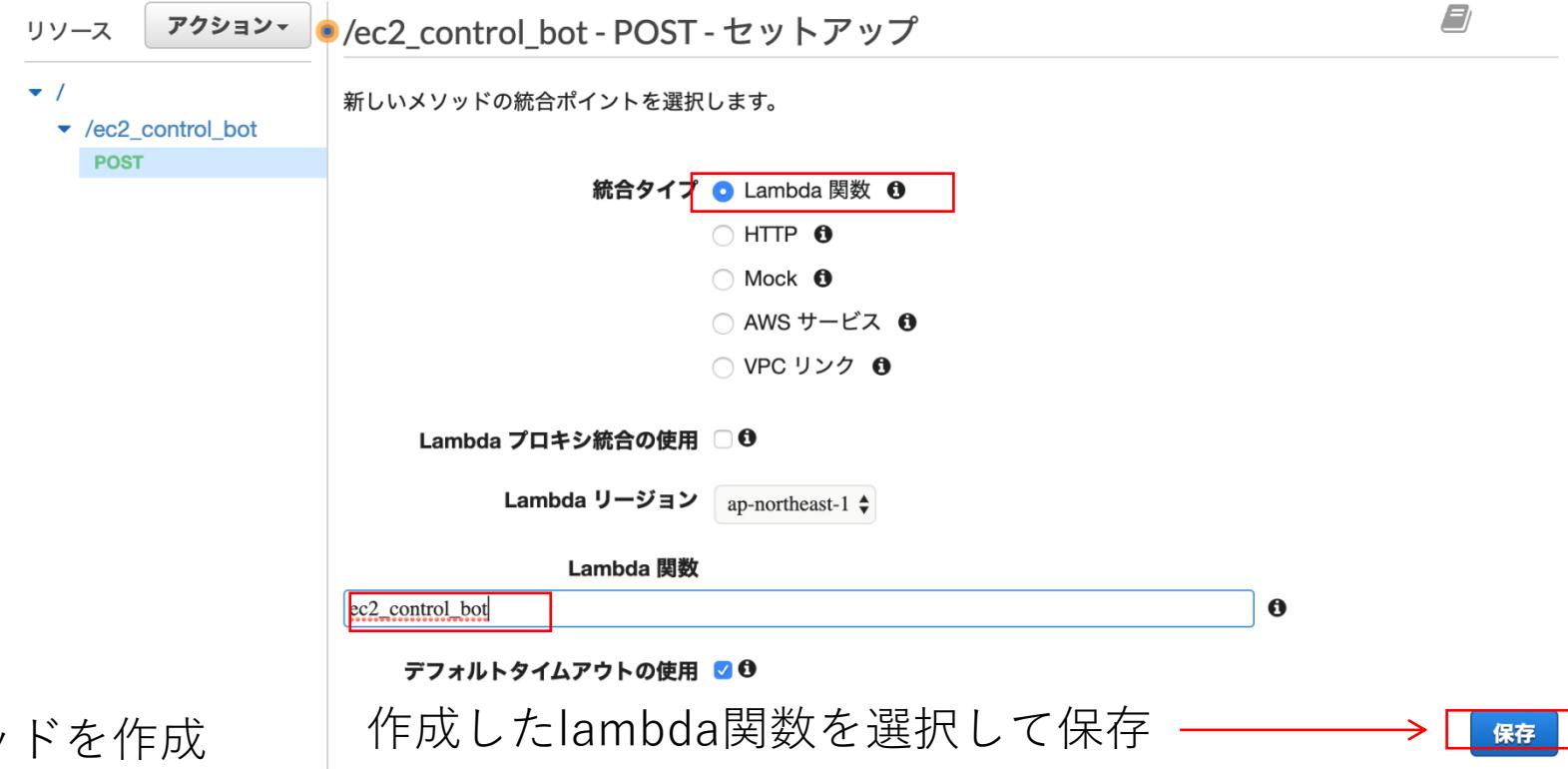
応用編

手順

3. API Gatewayで詳細の設定をする



アクションをクリックしてpostメソッドを作成



作成したlambda関数を選択して保存

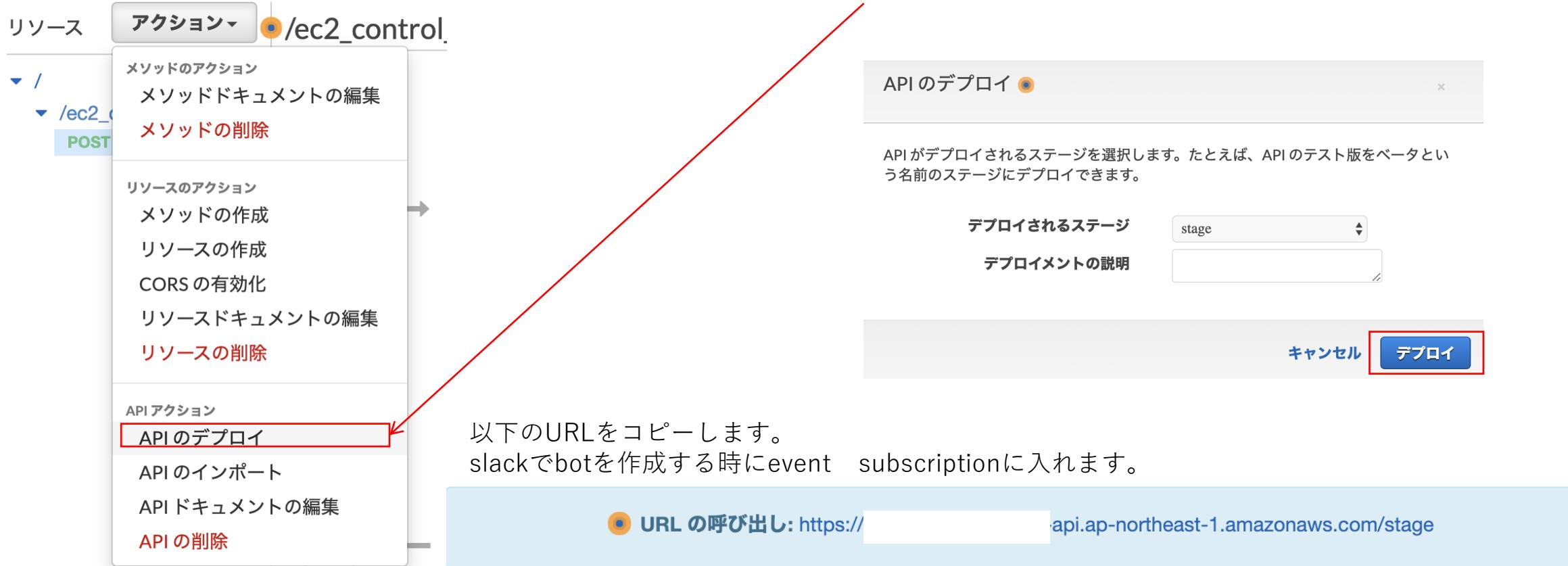
保存

応用編

手順

3. API Gatewayで詳細の設定をする

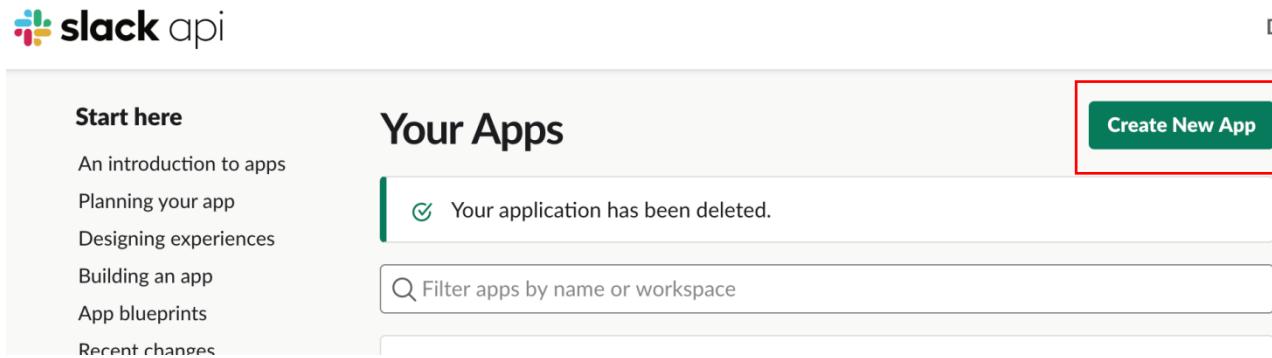
設定を保存後、アクションをクリックしてAPIデプロイ



応用編

手順

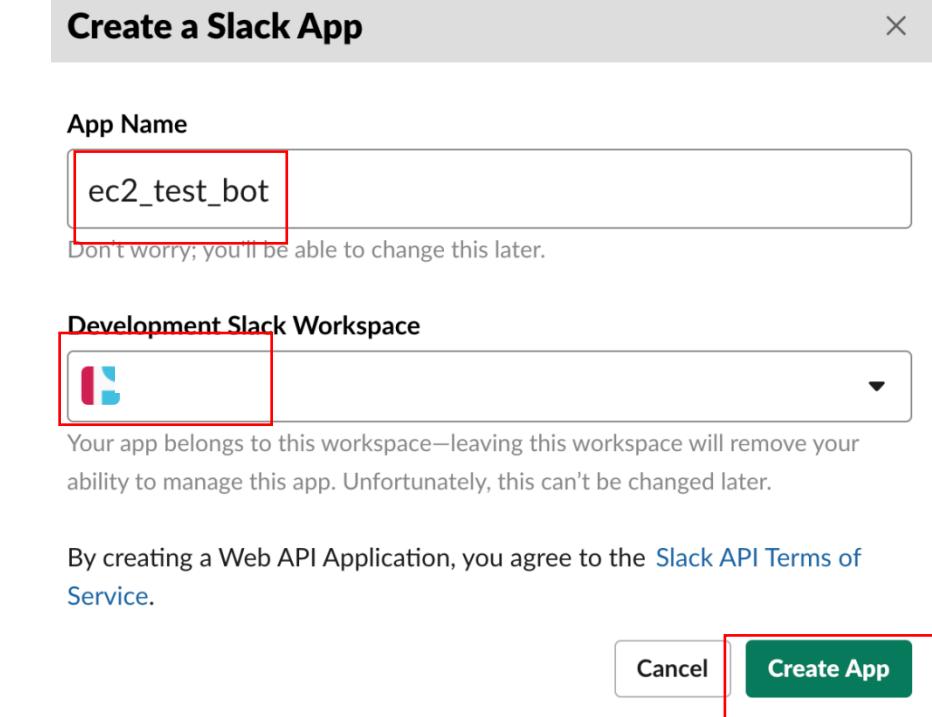
4. Slackでbotを作成



The screenshot shows the Slack API 'Your Apps' interface. On the left, there's a sidebar with 'Start here' links: An introduction to apps, Planning your app, Designing experiences, Building an app, App blueprints, and Recent changes. The main area is titled 'Your Apps' and displays a message: 'Your application has been deleted.' Below it is a search bar labeled 'Filter apps by name or workspace'. A green 'Create New App' button is prominently displayed at the top right of this section, with a red box highlighting it.

Slack APIでcreate new appをクリックしbotを作成します。Appの名前とworkspaceを指定して作成します。

作成が終わったら、bot userを追加します。



The screenshot shows the 'Create a Slack App' dialog box. It has fields for 'App Name' (containing 'ec2_test_bot') and 'Development Slack Workspace' (with a dropdown menu showing a workspace icon). A note below says, 'Don't worry; you'll be able to change this later.' At the bottom, there's a 'Cancel' button and a green 'Create App' button, with a red box highlighting the 'Create App' button.

Create a Slack App

App Name

ec2_test_bot

Development Slack Workspace

Your app belongs to this workspace—leaving this workspace will remove your ability to manage this app. Unfortunately, this can't be changed later.

By creating a Web API Application, you agree to the [Slack API Terms of Service](#).

Cancel Create App

応用編

手順

4. Slackでbotを作成

The screenshot shows the Slack App Builder interface for a bot named "ec2_test_bot". The left sidebar lists various app settings like Basic Information, Collaborators, and Features. The "Bot Users" section is currently selected and highlighted in blue. The main content area is titled "Bot User" and contains fields for "Display name" (set to "ec2testbot") and "Default username" (also set to "ec2testbot"). A note below the default username field specifies that usernames must be lowercase and under 21 characters. An "Always Show My Bot as Online" toggle switch is shown as "Off". On the right side, there are sections for "Features" (including App Home, Incoming Webhooks, etc.) and "Tools" (User ID Translation). The "Display name" field is set to "ec2testbot", and the "Default username" field is also set to "ec2testbot". A note next to the default username field states that if the chosen name is unavailable, it will be slightly modified. The "Always Show My Bot as Online" switch is turned "On". A red box highlights the "Add Bot User" button at the bottom right of the main form.

You can bundle a bot user with your app to interact with users in a more conversational manner. Learn more about [how bot users work](#).

Display name

ec2testbot

Names must be shorter than 80 characters, and can't use punctuation (other than apostrophes and periods).

Default username

ec2testbot

If this username isn't available on any workspace that tries to install it, we will slightly change it to make it work. Usernames must be all lowercase. They cannot be longer than 21 characters and can only contain letters, numbers, periods, hyphens, and underscores.

Always Show My Bot as Online

Install App
Manage Distribution

Features

App Home BETA
Incoming Webhooks
Interactive Components
Slash Commands
OAuth & Permissions
Event Subscriptions
Bot Users
User ID Translation

Tools

Update to Granular Scopes
Slack ❤️
Help

Display name

ec2testbot

Names must be shorter than 80 characters, and can't use punctuation (other than apostrophes and periods).

Default username

ec2testbot

If this username isn't available on any workspace that tries to install it, we will slightly change it to make it work. Usernames must be all lowercase. They cannot be longer than 21 characters and can only contain letters, numbers, periods, hyphens, and underscores.

Always Show My Bot as Online

Add Bot User

ONを入れてAdd Bot Userをクリックします。

応用編

手順

4. Slackでbotを作成

The screenshot shows the 'Event Subscriptions' configuration page for the 'ec2_test_bot' app. On the left, there's a sidebar with 'Settings' and 'Features' sections, and the 'Event Subscriptions' tab is selected. The main area has a heading 'Event Subscriptions' and a sub-section 'Enable Events' with a green 'On' toggle switch. Below it is a 'Request URL' field containing 'https://aws.com/stage/ec2' with a 'Verified' badge and a 'Change' button. A note explains that AWS API Gateway will send HTTP POST requests to this URL when events occur. The 'Subscribe to bot events' section is expanded, showing a table with one row: 'Event Name' (app_mention) and 'Description' (Subscribe to only the message events that mention your app or bot). At the bottom is a 'Add Bot User Event' button.

AWS API Gatewayと連携する為、Event SubscriptionにAPI Gatewayの設定が終わった時にコピーしたURLを入れます。Verifiedになると大丈夫です。

bot eventsにapp.mentionも入れます。

応用編

手順

4. Slackでbotを作成

最後にappをインストールした完了です。

インストールが終わったら、bot user access Tokenと verify tokenをコピーしてLambda関数にいれます。
verify tokenは Basic Informationで確認できます。

The screenshot shows the 'Install App to Your Team' section of the Slack app configuration. At the top left is a dropdown menu showing 'ec2_test_bot'. Below it is a sidebar with 'Settings' (Basic Information, Collaborators, **Install App**, Manage Distribution) and 'Features'. The main area contains a box with the text: 'Install your app to your Slack workspace to test your app and generate the tokens you need to interact with the Slack API. You will be asked to authorize this app after clicking Install App to Workspace.' A red box highlights the 'Install App to Workspace' button at the bottom of this box.

The screenshot shows the 'Installed App Settings' page. At the top right is a 'Copy' button. Below it is a section titled 'OAuth Tokens for Your Team' with the sub-section 'OAuth Access Token'. A red box highlights the 'Bot User OAuth Access Token' input field, which contains a long token. Another red box highlights the 'Copy' button next to it. At the bottom is a green 'Reinstall App' button.

応用編

手順

5. 動作確認

Slackで以下のようにコマンドを叩いて(@botの名前 コマンド(ex.run) で)
EC2インスタンスが起動出来たら成功です。



応用編

Lambda用のpythonコード

```
import os
import logging
import json
import urllib.request
import boto3

# ログ設定
logger = logging.getLogger()
logger.setLevel(logging.INFO)

# インスタンスregion,IDを設定
region = 'ap-northeast-1'
instances = ['i-0a66e058f11e10f51']

def lambda_handler(event, context):

    # 受信データをCloud Watchログに出力
    logging.info(json.dumps(event))

    # Slack APIの認証
    if "challenge" in event:
        return event["challenge"]

    # tokenのチェック
    if not is_verify_token(event):
        return "OK"

    if not is_app_mention(event):
        return "OK"

    # Slackにメッセージを投稿
    post_message_to_channel(event.get("event").get("channel"), "ec2
running...")

    #ec2start
    ec2 = boto3.client('ec2', region_name=region)
    ec2.start_instances(InstanceIds=instances)

    return 'OK'
```

```
def post_message_to_channel(channel, message):
    url = "https://slack.com/api/chat.postMessage"
    headers = {
        "Content-Type": "application/json; charset=UTF-8",
        "Authorization": "Bearer {}".format(os.environ["SLACK_BOT_USER_ACCESS_TOKEN"])
    }
    data = {
        "token": os.environ["SLACK_BOT_VERIFY_TOKEN"],
        "channel": channel,
        "text": message,
    }

    req = urllib.request.Request(url, data=json.dumps(data).encode("utf-8"), method="POST",
headers=headers)
    urllib.request.urlopen(req)

def is_verify_token(event):
    # トークンをチェック
    token = event.get("token")
    if token != os.environ["SLACK_BOT_VERIFY_TOKEN"]:
        return False

    return True

def is_app_mention(event):
    return event.get("event").get("type") == "app_mention"
```