

Lambda service example- CI, Infrastructure, Development, and Slack setup

This document contains an explanation of a reference architecture in the introduction, and then has detailed information, including screenshots, about how to set up a Lambda function and continuous integration pipeline with all the necessary AWS supports. It concludes with a little information about how to set up Slack to call an AWS Lambda function.

Introduction

In 2019, Mozilla's new product development will require rapid development and deployment of consumer-facing web services, often experimental in nature. We'll need to be able to deliver fast, experiment fast, scale fast, and fail fast.

To enable developers and the business to meet our ambitious goals, we need to be able to deliver software quickly and safely. Serverless technologies hold the promise that developers will not have to concern themselves with any of the details of the underlying infrastructure that supports a service. Developers and business users can focus on delivering customer needs and experiments.

Container-based architectures (like Docker on Kubernetes) have opened up ways for developers to deploy code without worrying about server configurations (anything that can run Docker can host a container). However, there is still a level of machine configuration and dependency management that must be managed for all deployments. Serverless removes this complication, and allows SRE's to better optimize use of expensive computing resources. A serverless function is much less expensive to operate than a Docker-based web service orchestrated via Kubernetes.

Goals

I had multiple goals in taking on this project:

1. To demonstrate that it is possible to have a CI pipeline that enables a developer to deploy a change to a Lambda function in the least time possible. (With this architecture, I am able to deploy changes to Lambda and have them reach anywhere from 1%-100% of user requests in less than 2 minutes)
2. To create a reference example for Lambda development, CI pipeline setup, and AWS config that can be used for learning and can be refined as Mozilla learns more about our new product needs.
3. To deliver a functional Slack extension that enables Mozillians to fulfill a collaboration need without leaving Slack.

Use Cases

This project, including this document and the code in the Github repos at the end of the article, delivers each of the following use cases:

As a developer, I want to be able to write Lambda functions and deploy them directly to production from my development environment without having to manage container dependencies (supported by this doc and the linked repos and resources)

As a Site Reliability Engineer, I want to set up and maintain a continuous integration pipeline, a Lambda function, API Gateway, and other services needed to enable developers and product managers to quickly deploy new business services (supported by this doc and the linked repos and resources)

As a Mozilla community member, I want to be able to search for bugs in Bugzilla from within a Slack conversation, using a simple command line (supported by the command delivered in Slack)

As a Mozilla Product Manager, I want to be able to define quick consumer-facing service experiments and be able to see these experiments deployed and tracked in the quickest possible way (supported by the architecture developed for this use case, and by accompanying education provided to Mozilla technical staff)

As a Mozilla software architect, I want to have a reference to follow when helping Mozilla teams set up development and deploy services aimed at achieving Mozilla's new product development goals.

Process

I started this work with a hypothesis that a developer and an SRE could quickly and easily stand up all of the services needed to take code from a developer's workstation and safely deploy into a consumer-facing web service with minimal outside help.

All of the work is done in my own accounts, since it was unclear how to provision all needed services within Mozilla at this time. Accounts used include Github, TravisCI, Amazon Web Services, and Slack.

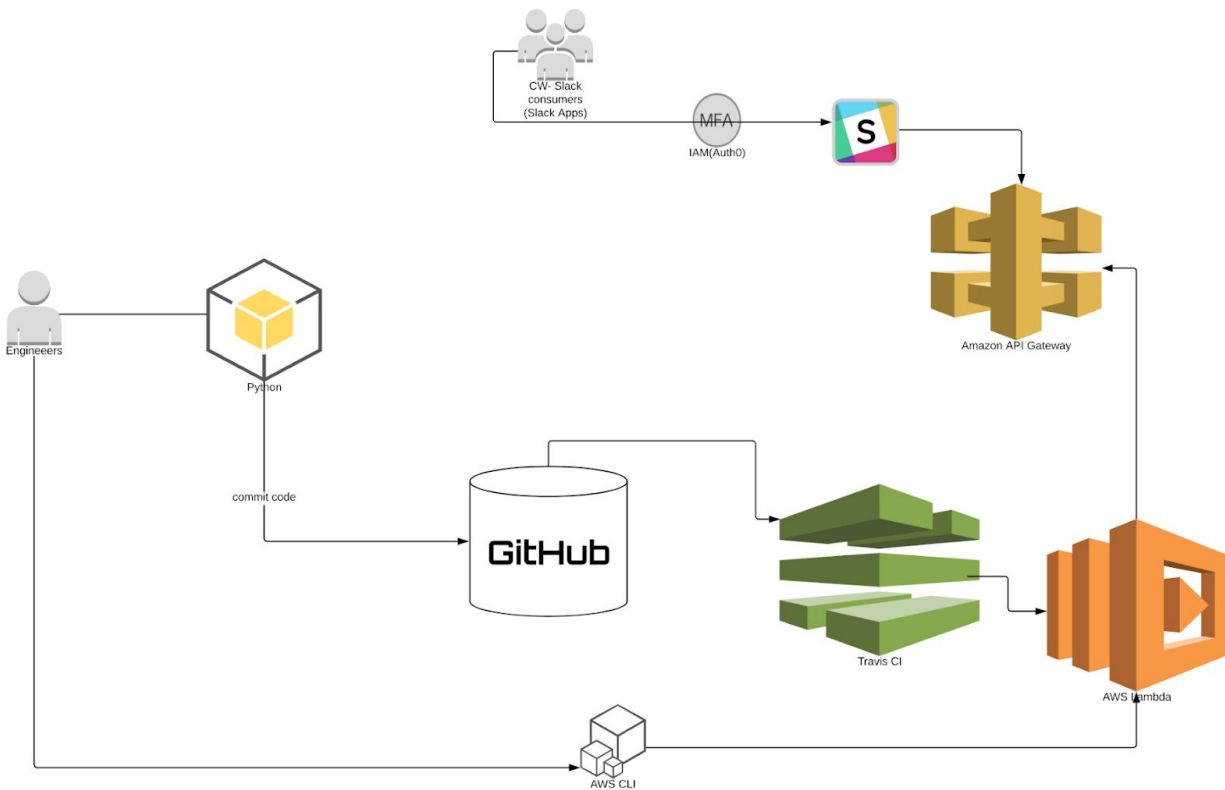
The Slack command is currently deployed in Mozilla's sandbox Slack instance.

The original target Slackbot was a bot that would allow a user to book a meeting room without leaving Slack. But complications around Google authentication and resource allocation, as well as the availability of off-the-shelf solutions that easily fulfill this use case, compelled a pivot.

The Slack use case that is implemented here is taken from a request Emma Humphries made in the [12/17/2018 Project Call](#), that someone at Mozilla make a Slack command that would allow users to call the BMODuplicates web service from within Slack.

TECHNICAL DETAILS OF ARCHITECTURE AND SETUP

This diagram illustrates the process by which the Slack app I've created is built, deployed, and accessed, and the following document goes into detail about how all these pieces go together:



Lambda setup

The Lambda function that will back the Slack command is easily setup in the AWS Lambda Management console, or through the AWS Command Line Interface. The function needs to be setup using a Python 3.6 runtime, and a custom IAM role (using the default `lambda_basic_execution` role).

The function will require an environment variable for `BOT_TOKEN`, and that will be available from Slack after we configure the Slack application.

Here are what the various configuration options will look like after the `BOT_TOKEN` is installed:

The screenshot shows the AWS Lambda console configuration page for a function named `findBmoDupesPy`. The page is divided into several sections:

- Designer:** This section shows the function's configuration. On the left, there is a list of triggers: API Gateway, AWS IoT, Application Load Balancer, CloudWatch Events, CloudWatch Logs, and CodeCommit. In the center, there is a diagram showing the function `findBmoDupesPy` with a `Layers` section (0 layers) and an `API Gateway` trigger (3 triggers). On the right, there is a section for `Amazon CloudWatch Logs` with a note: "Resources that the function's role has access to appear here".
- Function code:** This section shows the function's code configuration. It includes a `Code entry type` dropdown set to `Edit code inline`, a `Runtime` dropdown set to `Python 3.6`, and a `Handler` dropdown set to `lambda_function.lambda_handler`.
- Environment variables:** This section shows the function's environment variables. It includes a table with the following entries:

Key	Value	Action
<code>BOT_TOKEN</code>	<code>xxxx-xxxx-xxxx-xxxx-xxxx</code>	Remove
<code>Key</code>	<code>Value</code>	Remove

Execution role	Basic settings
<p>Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. Learn more about Lambda execution roles.</p> <p>Choose an existing role ▼</p> <p>Existing role You can use an existing role with this function. Lambda must be able to assume this role, and the role must have Amazon CloudWatch Logs permissions.</p> <p>bmodupes_role ▼</p>	<p>Description</p> <p>Memory (MB) Info Your function is allocated CPU proportional to the memory configured.</p> <p>128 MB</p> <p>Timeout Info 0 min 3 sec</p>

Here's the config information for my example Lambda function:

```
{
  "FunctionName": "findBmoDupesPy",
  "FunctionArn": "arn:aws:lambda:us-east-2:458411641808:function:findBmoDupesPy",
  "Runtime": "python3.6",
  "Role": "arn:aws:iam::458411641808:role/bmodupes_role",
  "Handler": "lambda_function.lambda_handler",
  "CodeSize": 1350,
  "Description": "",
  "Timeout": 3,
  "MemorySize": 128,
  "LastModified": "2018-12-26T21:08:53.008+0000",
  "CodeSha256": "3SxAApGfzGRuYpPsG+PT/fHbIrafIbvYh3tcXBTtXE=",
  "Version": "$LATEST",
  "VpcConfig": {
    "SubnetIds": [],
    "SecurityGroupIds": [],
    "VpcId": ""
  },
  "Environment": {
    "Variables": {
      "BOT_TOKEN": "xxxxxxx (removed- available from Slack)"
    }
  },
  "TracingConfig": {
    "Mode": "PassThrough"
  },
  "RevisionId": "233a1eff-542d-4d2c-acd8-6766188ec579"
}
```

Here's the IAM role bmo_dupes:

```
{
  "Role": {
    "Path": "/",
    "RoleName": "bmodupes_role",
    "RoleId": "AROAJJ2RKW23FEZA2ZC6O",
```

```
"Arn": "arn:aws:iam::458411641808:role/bmodupes_role",
"CreateDate": "2018-12-18T21:20:43Z",
"AssumeRolePolicyDocument": {
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
},
"MaxSessionDuration": 3600
}
```

API Gateway

With the Lambda function in place, we need to set up API Gateway, so that Slack will have a consistent URL to access the function.

In the API Gateway console, we create an API, add a resource, and add the HTTP command setup. We'll also need to create a mapping template so that the HTTP response returns JSON to Slack.

1. Create an API- hit the Create API button, and specify REST and a New API.

Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

☒ REST ☐ WebSocket


Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

☒ New API ☐ Clone from existing API ☐ Import from Swagger or Open API 3 ☐ Example API

Settings

Choose a friendly name and description for your API.

API name*	<input type="text" value="testApiForSlack"/>
Description	<input type="text" value="Showing the API config"/>
Endpoint Type	<div>Regional </div>

* Required

Create API

2. Select the Resources entry for the new API, and click on the Actions button. Select Create Resource. Give the resource a name and a useful path (the specified path will be the resource name for the API URL which you will provide to Slack). Do not select proxy, and CORS is not necessary. (Proxy may be useful but I struggled with getting Slack to successfully communicate with an API configured as Proxy)

Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

☒ REST ☐ WebSocket


Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

☒ New API ☐ Clone from existing API ☐ Import from Swagger or Open API 3 ☐ Example API

Settings

Choose a friendly name and description for your API.

API name*	<input type="text" value="testApiForSlack"/>
Description	<input type="text" value="Showing the API config"/>
Endpoint Type	<div>Regional </div>

* Required

Create API

3. With the resource created, select the resource, click on Actions, choose “Create Method”. A dropdown will appear under the resource. Select either POST or ANY (Slack will POST to the gateway), and configure a Lambda function integration type. Type the name of the Lambda function. Your API gateway is now setup, but we need to make sure it returns JSON in a way Slack will be able to handle.

Resources

Actions ▾

▼ /

▼ /testresource

ANY

GET

/testresource - ANY - Setup

Choose the integration point for your new method.

Integration type ☒ Lambda Function ⓘ

☐ HTTP ⓘ

☐ Mock ⓘ

☐ AWS Service ⓘ

☐ VPC Link ⓘ

Use Lambda Proxy integration ☐ ⓘ

Lambda Region

Lambda Function

Use Default Timeout ☒ ⓘ

4. Select the POST or ANY method you just created, and in the diagram that appears, select Integration Response. When this screen appears, hit the dropdown next to the listed method, and at the bottom hit the Mapping Templates. Verify that there is an application/json mapping- if not, create one (it should be there by default though)

resources

Actions

▼ /

▼ /testresource

POST

← Method Execution

/testresource - POST - Integration Response

First, declare response types using [Method Response](#). Then, map the possible responses from the backend to this method's response types.

	Lambda Error Regex	Method response status	Output model	Default mapping	
▼	-	200		Yes	✕

Map the output from your Lambda function to the headers and output model of the 200 method response.

Lambda Error Regex

default

Content handling

Passthrough

Cancel

Save

► Header Mappings

▼ Mapping Templates

Content-Type

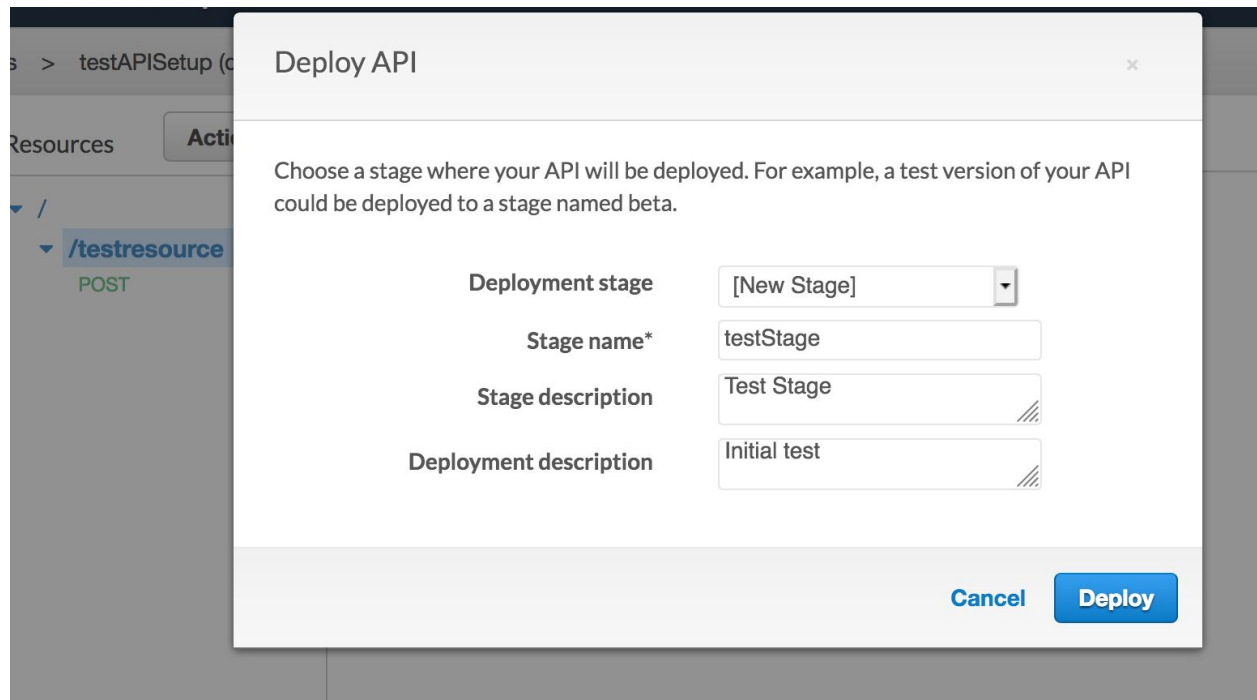
application/json

⊖

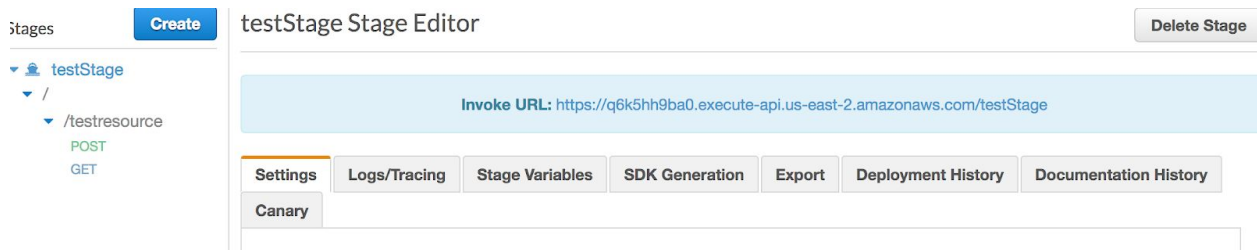
⊕

Add mapping template

5. Now create a stage to host the deployed API. You can do this by doing an initial deploy of the API. In the API Gateway console, select Actions, and Deploy API. Under Deployment Stage, select [New Stage] and complete the screen (you'll use the stage name in your Travis configuration later)



You will now be able to hit the API Gateway and get a response! You'll see the URL at the top of the stage editor:



Clicking on the URL as is will give you a "Missing Authentication Token" error. If you have configured ANY, you'll be able to append the name of your API (e.g., /testresource) to the URL and get a 200 "Hello From Lambda" response. You can also POST to the API via curl if you haven't configured a GET.

After this is done, here's the configurations:

The REST API (from aws get-rest-apis)

```
{
  "id": "q6k5hh9ba0",
  "name": "testAPISetup",
  "description": "Demonstrating API gateway config",
  "createdDate": 1545944028,
```

```

    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "REGIONAL"
        ]
    }
},

```

The resource config (from aws-get-resources):

```

{
  "items": [
    {
      "id": "8043no",
      "parentId": "v6s33gmffj",
      "pathPart": "testresource",
      "path": "/testresource",
      "resourceMethods": {
        "GET": {},
        "POST": {}
      }
    },
    {
      "id": "v6s33gmffj",
      "path": "/"
    }
  ]
}

```

The POST method (including the cli call to get this info):

```

mvankleeck-44385:SlackCmdInLambdaForBmoDupes mvankleeck$ aws apigateway get-method
--rest-api-id q6k5hh9ba0 --resource-id 8043no --http-method POST --region us-east-2

```

```

{
  "httpMethod": "POST",
  "authorizationType": "NONE",
  "apiKeyRequired": false,
  "requestParameters": {},
  "methodResponses": {
    "200": {
      "statusCode": "200",
      "responseModels": {
        "application/json": "Empty"
      }
    }
  },
  "methodIntegration": {
    "type": "AWS",
    "httpMethod": "POST",

```

```

    "uri":
"arn:aws:apigateway:us-east-2:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-
2:458411641808:function:testCreateFunc/invocations",
    "passthroughBehavior": "WHEN_NO_MATCH",
    "contentHandling": "CONVERT_TO_TEXT",
    "timeoutInMillis": 29000,
    "cacheNamespace": "8043no",
    "cacheKeyParameters": [],
    "integrationResponses": {
        "200": {
            "statusCode": "200",
            "responseTemplates": {
                "application/json": null
            }
        }
    }
}

```

We now have a fully configured Lambda function and API gateway, so we now turn to the CI pipeline to get code delivered into the function.

GitHub and Travis CI setup

We use a Github repository to handle the code (including the Travis build) for this project. Create a new Github repository. I created this one as a public repository since I am using a public (free) account (this is a potential security leak as we will add a trigger to start a Travis build and a deployment based on commits to master).

Note- always select the Mozilla Public License 2.0 when creating a repo, unless there is a specific reason not to (very rare).

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



mvk-mozilla ▾

Repository name

testLambdaTravisRepo ✓

Great repository names are short and memorable. Need inspiration? How about **stunning-system**.

Description (optional)

a test repo for setting up AWS Lambda deployment via Travis



☒ Public

Anyone can see this repository. You choose who can commit.



☐ Private

You choose who can see and commit to this repository.

☒ Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None ▾

Add a license: Mozilla Public License 2.0 ▾



Create repository

With the Github repo set up, now we need to setup some AWS stuff to support the build. You'll need an S3 bucket to deploy to, an IAM user to do the deployment, and an IAM role for the deployment. All of that is covered in the next section (and in the linked article <https://dev.to/codevbus/deploy-aws-lambda-functions-with-aws-sam-cli-and-travis-ci-part-2-2goh>)

IAM/Lambda/S3 setup to support Travis CI

Travis will rely on the top-level .travis.yml file to build. Follow the information here to properly tailor your file:

<https://docs.travis-ci.com/user/deployment/lambda/>

This tutorial here goes into depth about how to set up S3 to support the build, how to set up IAM, and how to modify the necessary files to make this all work. The SlackbotLambdaExample repo I've set up has working .travis.yml and template.yaml files based on this.

<https://dev.to/codevbus/deploy-aws-lambda-functions-with-aws-sam-cli-and-travis-ci-part-2-2goh>

Here is an example of .travis.yml based on what I used in the repo to deploy code to an S3 bucket, from where it is installed into Lambda:

```
language: python
python:
  - '3.6'
branches:
  only: master
install:
  - pip install -r requirements.txt
  - pip install awscli
  - pip install aws-sam-cli
script:
  - pytest
  - sam validate
  - sam package --template-file template.yaml --s3-bucket slackbotohdeploybucket
  --output-template-file packaged.yaml
deploy:
  function_name: "testCreateFunc"
  provider: script
  script: sam deploy --template-file packaged.yaml --stack-name travis-serverless-test
  --capabilities CAPABILITY_IAM
  skip_cleanup: true
  on:
    branch: master
notifications:
  email:
    on_failure: always
env:
  global:
    - AWS_DEFAULT_REGION=us-east-2
```

And template.yaml:

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: A simple Slackbot to prove out CI pipeline
Resources:
  SlackbotLambda:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./src
      Handler: SlackbotLambda.lambda_handler
      Timeout: 300
      Runtime: python3.6
```

Following this, we need to create the IAM user and role, based on the article instructions. Follow the article to create the needed policy- we call it TravisSlackbotLambdaDeployPolicy. With it created- here's the IAM Create User screenshots you will follow:

NOTE- when you create the IAM user with this role, you will need to save the provided AWS keys to set up your Travis build!

Add user



Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

[+ Add another user](#)

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)


- Access type* ☒ **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.
- ☐ **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.


* Required


[Cancel](#)


[Next: Permissions](#)

▼ Set permissions

 Add user to group

 Copy permissions from existing user

 Attach existing policies directly

Create policy 

Filter policies ▼

Q Travis

Showing 1 result

	Policy name ▼	Type	Used as	Description
<input type="checkbox"/>	▼ TravisSlackbotLam...	Customer managed	Permissions policy (1)	An attempt to create a deployment policy fo...

TravisSlackbotLambdaDeployPolicy

An attempt to create a deployment policy for the Lambda slackbot, following https-dev.to-codevbus-deploy-aws-lambda-functions-with-aws-sam-cli-and-travis-ci-part-2-2goh

Policy summary

{ } JSON

Edit policy

Q Filter

Service ▼	Access level	Resource	Request condition
-----------	--------------	----------	-------------------

Cancel

Previous

Next: Tags

(you can skip the next screen, tags)

Add user

1 2 3 4 5

Review

Review your choices. After you create the user, you can view and download the autogenerated password and access key.

User details

User name testUser
AWS access type Programmatic access - with an access key
Permissions boundary Permissions boundary is not set

Permissions summary

The following groups and policies will be copied from the selected existing user and attached to the user shown above.

Type	Name
Managed policy	TravisSlackbotLambdaDeployPolicy

Tags

No tags were added.

Cancel

Previous

Create user

CRITICALLY IMPORTANT SCREEN!!! When you get here, save the CSV to a secure place on your local box. You will use these keys to configure Travis.

Add user

1 2 3 4 5



Success

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: <https://458411641808.signin.aws.amazon.com/console>

Download .csv

	User	Access key ID	Secret access key
▶	testUser	AKIAIGKJJ2LGLPAH43Q	***** Show

Here are all the configurations the S3, policy, and user setup has led to (with the commands included to list this info):

```
mvankleeck-44385:SlackCmdInLambdaForBmoDupes mvankleeck$ aws s3api list-buckets
```

```
{
  "Buckets": [
    {
      "Name": "numpy-test-dev-serverlessdeploymentbucket-xisssa8527id",
      "CreationDate": "2018-12-14T22:47:14.000Z"
    },
    {
      "Name": "slackbotohdeploybucket",
      "CreationDate": "2018-12-12T22:37:14.000Z"
    }
  ],
  "Owner": {
    "DisplayName": "mvk",
    "ID": "1395d43c314b45b51af69d5f0309a0e61f5dcff5d5129dd0bab72fcae070ebc7"
  }
}
```

```
mvankleeck-44385:SlackCmdInLambdaForBmoDupes mvankleeck$ aws s3api get-bucket-acl
--bucket slackbotohdeploybucket
```

```
{
  "Owner": {
    "ID": "1395d43c314b45b51af69d5f0309a0e61f5dcff5d5129dd0bab72fcae070ebc7"
  },
  "Grants": [
    {
      "Grantee": {
        "ID": "1395d43c314b45b51af69d5f0309a0e61f5dcff5d5129dd0bab72fcae070ebc7",
        "Type": "CanonicalUser"
      },
      "Permission": "FULL_CONTROL"
    }
  ]
}
```

```
mvankleeck-44385:SlackCmdInLambdaForBmoDupes mvankleeck$ aws iam get-user --user-name
TravisSlackbotLambdaDeployUser
```

```
{
  "User": {
    "Path": "/",
    "UserName": "TravisSlackbotLambdaDeployUser",
    "UserId": "AIDAI3BFESAXPYSGZQ46O",
    "Arn": "arn:aws:iam::458411641808:user/TravisSlackbotLambdaDeployUser",
    "CreateDate": "2018-12-12T23:50:07Z"
  }
}
```

```

    }
}

mvankleeck-44385:SlackCmdInLambdaForBmoDupes mvankleeck$ aws iam
list-attached-user-policies --user-name TravisSlackbotLambdaDeployUser
{
  "AttachedPolicies": [
    {
      "PolicyName": "TravisSlackbotLambdaDeployPolicy",
      "PolicyArn":
"arn:aws:iam::458411641808:policy/TravisSlackbotLambdaDeployPolicy"
    }
  ]
}

mvankleeck-44385:SlackCmdInLambdaForBmoDupes mvankleeck$ aws iam get-policy
--policy-arn arn:aws:iam::458411641808:policy/TravisSlackbotLambdaDeployPolicy
{
  "Policy": {
    "PolicyName": "TravisSlackbotLambdaDeployPolicy",
    "PolicyId": "ANPAJZAPSW7GNKX6JZJFU",
    "Arn": "arn:aws:iam::458411641808:policy/TravisSlackbotLambdaDeployPolicy",
    "Path": "/",
    "DefaultVersionId": "v6",
    "AttachmentCount": 1,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "Description": "An attempt to create a deployment policy for the Lambda
slackbot, following
https-dev.to/codevbus-deploy-aws-lambda-functions-with-aws-sam-cli-and-travis-ci-part-
2-2goh",
    "CreateDate": "2018-12-12T23:45:42Z",
    "UpdateDate": "2018-12-13T01:28:05Z"
  }
}

```

And here is the policy JSON for the above policy:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::slackbotohdeploybucket",
        "arn:aws:iam::458411641808:role/service-role/SlackPyRole"
      ]
    }
  ]
}

```

```

    },
    {
        "Sid": "VisualEditor1",
        "Effect": "Allow",
        "Action": [
            "s3:PutObject",
            "s3:GetObjectAcl",
            "s3:GetObject",
            "iam:CreateRole",
            "s3:DeleteObject",
            "s3:PutObjectAcl"
        ],
        "Resource": [
            "arn:aws:s3:::slackbotohdeploybucket/*",
            "arn:aws:iam::458411641808:role/SlackbotDeployRole"
        ]
    },
    {
        "Sid": "VisualEditor2",
        "Effect": "Allow",
        "Action": [
            "iam:*",
            "lambda:*",
            "cloudformation:*"
        ],
        "Resource": "*"
    }
]
}

```

Travis CI

With all the configuration done above for Travis (S3 bucket, IAM policy, IAM user, **and keys for the user saved!!!**) we are now ready to set up a Travis build.

This assumes you've created a Lambda function and have committed it to your repo. If not, you can use an example like

<https://github.com/mvk-mozilla/SlackbotLambdaExample/blob/master/src/SlackbotLambda.py>

(it's a simple Slackbot example and includes a `__main__` definition to run a test).

Log in to travis-ci.org and connect it to your Github account if you haven't already. Select the repository for this project, and click Activate Repository:

 mvk-mozilla / SlackCmdInLambdaForBmoDupes

[Current](#) [Branches](#) [Build History](#) [Pull Requests](#)



This is not an active repository

You can activate the repository on your profile,
or by clicking the button below

Activate repository

Once the repo is activated, click on Settings in the repo's menu, and add the AWS keys from your CSV- this will give your build permission to deploy to S3:

General

☒ Build pushed branches ?

☐ Limit concurrent jobs ?

☒ Build pushed pull requests ?

Auto Cancellation

Auto Cancellation allows you to only run builds for the latest commits in the queue. This setting can be applied to builds for Branch builds and Pull Request builds separately. Builds will only be canceled if they are waiting to run, allowing for any running jobs to finish.

☒ Auto cancel branch builds

☒ Auto cancel pull request builds

Environment Variables

Notice that the values are not escaped when your builds are executed. Special characters (for bash) should be escaped accordingly.

AWS_ACCESS_KEY_ID

.....



AWS_SECRET_ACCESS_KEY

.....



Please make sure your secret key is never related to the repository, branch name, or any other guessable string. For more tips on generating keys [read our documentation](#).

Name

Value



Display value in
build log

Add

At this point, you have a working repo, CI, API Gateway, and Lambda function. All that's left to do is to attach it to Slack (and add the Slack API key to the AWS Lambda environment).

Slack config

With the CI pipeline and the Lambda function all set up, we can set up a Slack command that calls the function. Slack will post a JSON even to the function.

This assumes you have access to add apps to a Slack instance I used the Mozilla Slack sandbox mozilla-sandbox-SCIM for this.

1. Create a Slack app

Create a Slack App

App Name

/bmoDupes

Don't worry; you'll be able to change this later.

Development Slack Workspace

m mozilla-sandbox-SCIM

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

2. You'll need the Verification Token for your code. You can get that from this screen. Note that the verification token is deprecated and we'll need to change our code to use the client ID and secret at some point. For now, I use the Verification Token and copy it to the BOT_TOKEN variable in the Lambda configuration (environment variables are in configuration, under the code). Also note, the token below is no longer valid (of course!!!):

App Credentials

These credentials allow your app to access the Slack API. They are secret. Please don't share your app credentials with anyone, include them in public code repositories, or store them in insecure ways.

App ID

AF2KLJY03

Date of App Creation

December 27, 2018

Client ID

371351187216.512666644003

Client Secret

.....

Show

Regenerate

You'll need to send this secret along with your client ID when making your [oauth.access](#) request.

Signing Secret

.....

Show

Regenerate

Slack signs the requests we send you using this secret. Confirm that each request comes from Slack by verifying its unique signature.

Verification Token

JexSCmYNcFAR6feZfpbs4pzD

Regenerate

This deprecated Verification Token can still be used to verify that requests come from Slack, but we strongly recommend using the above, more secure, signing secret instead.

3. With the credential now set up in the Lambda function, Lambda can call back to Slack and post responses. Next, create a slash command. The request URL is the URL for the API Gateway you configured above (to the resource you will POST to):

Edit Command

Command	<input type="text" value="/bmodupes"/> ⓘ
Request URL	<input type="text" value="https://5nn4zlkymi.execute-api.us-e..."/> ⓘ
Short Description	<input type="text" value="BMO Python dupe checker using lambda"/>
	<input type="text" value="pass a BMO search"/>
Usage Hint	Optionally list any parameters that can be passed.

After this, install the app, and boom! You've got a working pipeline!

With this basic setup, a developer can commit and push changes in a matter of minutes.

IDE

I used PyCharm and set up the new AWS Lambda integration so that I could locally run my Lambda before committing and deploying.

TODO- details about PyCharm setup

Links

Articles that provided guidance

<https://serverless.com/blog/serverless-python-packaging/>

<https://dev.to/codevbus/deploy-aws-lambda-functions-with-aws-sam-cli-and-travis-ci-part-2-2goh>

<https://blog.theodo.fr/2017/08/serverless-applications-aws-travis-make-deployment-great/>

<https://api.slack.com/tutorials/aws-lambda>

<https://chatbotslife.com/write-a-serverless-slack-chat-bot-using-aws-e2d2432c380e>

<https://medium.com/@farski/learn-aws-api-gateway-with-the-slack-police-2nd-edition-c5f3af9c1ec7>

Repos

BMO Dupes Glitch project (source code available directly on page)

<https://bmo-find-dupes.glitch.me/>

SlackbotLambdaExample Git repository

<https://github.com/mvk-mozilla/SlackbotLambdaExample>

Configured to trigger a Travis build that deploys the service every time a commit is pushed to master

SlackCmdInLambdaForBmoDupes Git repository

<https://github.com/mvk-mozilla/SlackCmdInLambdaForBmoDupes>

Contains a Lambda function that takes a request from Slack, searches Bugzilla with it, and returns a list of bugs formatted for Slack display through a callback to Slack