AWS EC2 to Cloudflare Lambda Function Documentation

Release https://github.com/infamousjoeg/aws-ec2-cloudflare-lambda/releases/tag/v1.0

Joe Garcia, CISSP

Oct 25, 2018

Contents:

CHAPTER 1

Use Case Explanation

I like to use a very specific domain name when I demo [CyberArk Conjur](https://conjur.org) because it's easier for me to remember and it also looks better to the customer. Even so, if it doesn't look good to them – it still looks better to me!

I noticed quickly that when I'd set the public IPv4 address of my [AWS](https://aws.amazon.com) [EC2](https://aws.amazon.com/ec2/) instance that I demo from to an A record for *cdemo.cybr.rocks* in [Cloudflare](https://cloudflare.com)'s DNS, it'd be good for that Instance running. However, after I stopped the EC2 instance and, eventually, restarted it again, it boots with a different public IPv4 address.

Now, I'm sure I could just use an [Elastic IP](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html) and assign it and call it a day… but that costs money – even when it's not being used! I can save money, even over the Elastic IP lease, triggering the [AWS Lambda](https://aws.amazon.com/lambda/) function only when my _Instance **State_** changes.

Finally, to the use case: upon a defined Instance's _Instance **State_** changing to "Running", grab the defined Instance's public IPv4 address and update a defined [A record](https://support.dnsimple.com/articles/a-record/) in Cloudflare's DNS.

CHAPTER 2

My Engineered Solution

- First and foremost, it should be open sourced on GitHub.
  - <https://github.com/infamousjoeg/aws-ec2-cloudflare-lambda>
  - Language should be [Python](https://www.python.org/). _(My most comfortable language currently.)_
- Any custom functions should be put in a [Python module package](https://docs.python.org/2/tutorial/modules.html) format

for easy distribution later.

- The AWS Lambda function's [execution

role](https://docs.aws.amazon.com/lambda/latest/dg/intro-permission-model.html#lambda-intro-execution-role) should have the following permissions:

- [CloudWatch Logs](https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html)
- [SNS](https://aws.amazon.com/sns/) (_Publish_)
- EC2 (_EC2ReadOnlyAccess_ IAM Role)
- The _Instance **State_** change should be a [CloudWatch Event

trigger](https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/Create-CloudWatch-Events-Rule.html) in the function.

- The function should be universally adoptable. It should make the following variables available to the Python script as

Environment Variables:

- *EC2_INSTANCE_NAME*
- *CLOUDFLARE_EMAIL*
- *CLOUDFLARE_API_KEY*
- *CLOUDFLARE_A_NAME*

- *CLOUDFLARE_DNS_ID*
- *CLOUDFLARE_ZONE_ID*
- The function should use the [AWS SDK](https://aws.amazon.com/sdk-for-python/) to describe the EC2 instance provided and

retrieve the public IPv4 address assigned.

- The function should use [Cloudflare's RESTful API](https://api.cloudflare.com/) to update the specified A record in the

DNS & Zone given using the E-Mail and API Key provided for authentication to do so. Updating it to the previously retrieved public IPv4 address.

CHAPTER 3

How I Did It

CHAPTER 4

Decomposition

Coming into this phase, I already knew I was doing something that took _way_ too long and _way_ too many times repeatedly. Immediately, that triggered my "Automation Senses" – like Spidey Senses… but less cool… and not surprisingly dorkier.

Easily repeatable processes are the **GOLDEN RULE** when it comes to easy automation. All that was left was decomposing that easily repeatable process:

My Easily Repeatable Process

- When I start an AWS EC2 Instance, I do the following: * Copy the EC2 Instance's public IPv4 address to my clipboard. * Browse to <https://cloudflare.com>. * Login to my Cloudflare account. * Select *cybr.rocks* from the list of domains managed. * Select the IP address value of the A record named *cdemo.cybr.rocks* and update it with the public IPv4 address on my

clipboard.

CHAPTER 5

“Step” Development

I made a name for this development method because I don’t really know what it’s called. However, it’s a process to development I found to be profoundly effective in a majority of my situations. Those being where I’m a solo developer, tester, and releaser all wrapped into one.

The idea behind Step Development is rather easy – take your decomposed “easily repeatable process” and script it one bullet point at a time. Testing it one bullet point at a time. While also, essentially, releasing it one bullet point at a time.

Step #1: Get the EC2 Instance’s Public IPv4 Address

The first bullet point in my decomposition is to grab the running EC2 instance’s public IPv4 address once it is assigned fresh off a new run state. I created and tested the function released in [pkg/func.py](https://github.com/infamousjoeg/aws-ec2-cloudflare-lambda/blob/master/pkg/func.py#L8) in my Terminal within MacOS.

Step #2: Authenticate to Cloudflare

The second bullet point in my decomposition is to browse to Cloudflare, while my third is to login to Cloudflare. Since this is automation, we’ll kill two birds with one stone over the REST API. Better yet, after investigating the [Cloudflare REST API Documentation](https://api.cloudflare.com/), it looks like we can authenticate _while_ also updating the A record.

SCORE! Nothing to do this step!

Step #3: Update A Record in Cloudflare DNS via REST API

Finally, the meat and potatoes of the function!

I created [cloudflare.py](https://github.com/infamousjoeg/aws-ec2-cloudflare-lambda/blob/master/cloudflare.py) as a means to creating and testing this function released in [pkg/func.py](https://github.com/infamousjoeg/aws-ec2-cloudflare-lambda/blob/master/pkg/func.py#L8). I decided to keep it because it’s a great example of how to interact with Cloudflare’s REST API using Python in a raw manner.

Back to the function, though. It authenticates with the provided Cloudflare e-mail address and API key. After confirmation of successful authentication, it then updates the A record associated with the A record name provided via DNS ID in the domain managed in Cloudflare associated with the provided Zone ID.

CHAPTER 6

Where Is It?

I released my AWS Lambda function under [main.py](https://github.com/infamousjoeg/aws-ec2-cloudflare-lambda/blob/master/main.py). It invokes *main.handler* on trigger action – therefore, you'll find in [main.py](https://github.com/infamousjoeg/aws-ec2-cloudflare-lambda/blob/master/main.py) a defined handler function that contains the actions to take.

You can use the exported [AWS SAM template](https://docs.aws.amazon.com/lambda/latest/dg/serverless_app.html) to orchestrate setting up your AWS Lambda function using the same Python script. Just don't forget to populate values into your Environment Variables before enabling the CloudWatch Event trigger!

![How to Deploy SAM Template](https://github.com/infamousjoeg/joeco.de/blob/gh-pages/assets/images/Messages%20Image(2549426905).png?raw=true)

![CloudFormation Success Screenshot](https://github.com/infamousjoeg/joeco.de/blob/gh-pages/assets/images/Messages%20Image(3292846589).png?raw=true)

![Lambda Function Created Automatically](https://github.com/infamousjoeg/joeco.de/blob/gh-pages/assets/images/Messages%20Image(2459700380).png?raw=true)

## Release

You may download the latest SAM template release from the [GitHub project page](https://github.com/infamousjoeg/aws-ec2-cloudflare-lambda/releases).

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

# AWS EC2 to Cloudflare Lambda Function

[![Documentation Status](https://readthedocs.org/projects/aws-ec2-cloudflare-lambda/badge/?version=latest) {]}(https://aws-ec2-cloudflare-lambda.readthedocs.io/en/latest/?badge=latest)

![AWS Lambda Function](https://joeco.de/assets/images/cdemoupdatecloudflare-screenshot.png)