Creating a Simple, Low-Cost Twitter Bot, utilising Serverless Technologies.

Craig Godden-Payne
Jun 30 · 6 min read ★

People have a love-hate relationship with twitter bots.

They can be useful for retweeting content that is relevant for things you are looking for, but they can also be annoying if they tweet too much, or about stuff that you don't care for.

Creating a Twitter bot is relatively straightforward.

In true serverless style, it is possible to build your own, in a cost-effective way.

This post will focus on creating a Twitter Bot, in Node, hosted in AWS Lambda.

. . .

Misconceptions of serverless technologies:

If you want to learn more about some of the misconceptions with serverless technologies, check out my other post:

Three misconceptions about Serverless, and why Serverless is often misunderstood?

The trend of serverless continues, but there are still misconceptions, and these often this can lead to decisions to…

medium.com

. . .

Why serverless?

Using serverless patterns for a Twitterbot makes complete sense.

The trigger.

The trigger is going to be based on a cron job.

In true serverless style, I will be using cloudwatch and utilise a cron like expression, to trigger my lambda.

Here is my event, driven through terraform.

```
resource aws_cloudwatch_event_rule every_one_hour {
  name                = "every-one-hour"
  description         = "Fires every one hour"
  schedule_expression = "rate(1 hour)"
}
```

A cloudwatch event rule will run every one hour.

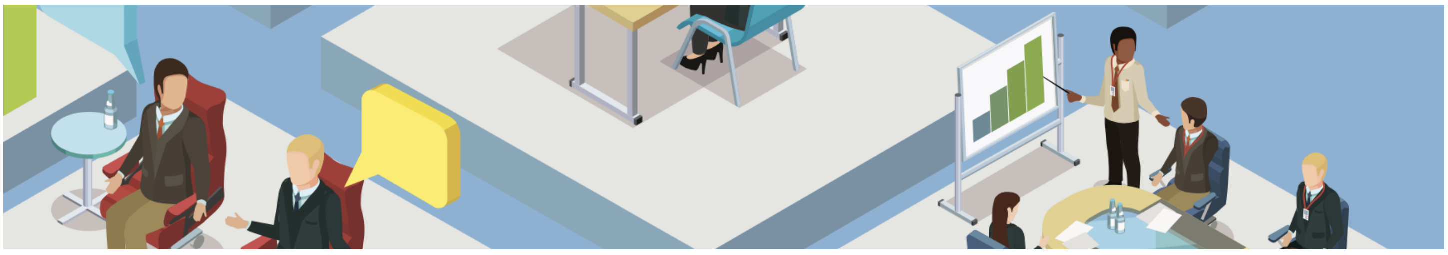
When this runs, I want it to call a lambda, so I also need to set up an event target.

```
resource aws_cloudwatch_event_target twitter_bot {
  rule      = aws_cloudwatch_event_rule.every_one_hour.name
  target_id = "lambda"
  arn       = aws_lambda_function.lambda.arn
}
```

The Only Step You Need to Progress in Your Career as a Developer or Engineer.

I don’t usually like to write articles about myself, as it feels a bit self-indulgent, but I thought it would be useful…

medium.com



Not forgetting to set up to allow the function to be executed.

```
resource aws_lambda_permission allow_cloudwatch_to_call_check_foo {
  statement_id  = "AllowExecutionFromCloudWatch"
  action        = "lambda:InvokeFunction"
  function_name = aws_lambda_function.lambda.function_name
  principal     = "events.amazonaws.com"
  source_arn    = aws_cloudwatch_event_rule.every_one_hour.arn
}
```

. . .

The code.

So now there is a method for triggering the lambda, we now need some code for it to run.

I picked node for the framework, as I just really like javascript. It would be just as easy to write this in a whole host of other languages.

The dependencies I am going to use for this project are (package.json):

```
{
  "name": "twitterbot",
  "version": "1.0.0",
  "description": "My twitterbot",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "twitter": "^1.7.1"
  }
}
```

I then need to create the entrypoint for where AWS Lambda will invoke my function (index.js):

```
let TwitterRetweetClient = require('./TwitterClient');

const consumer_key = process.env.TWITTER_CONSUMER_KEY;
const consumer_secret = process.env.TWITTER_CONSUMER_SECRET;
const access_key = process.env.TWITTER_ACCESS_KEY;
const access_secret = process.env.TWITTER_ACCESS_SECRET;
const find_hashtag = process.env.FIND_HASHTAG;
const retweet_count = process.env.RETWEET_COUNT;

exports.handler =  async function(event, context) {
    console.log("Lambda:Invoke");
    let twitterClient = new TwitterRetweetClient(consumer_key,
consumer_secret, access_key, access_secret);
    console.log("RetweetCount: " + retweet_count);
    twitterClient.retweet(find_hashtag, parseInt(retweet_count));
}
```

As you can see, all the configuration is injected via environment variables.

Using Microservice Patterns in an increasingly Serverless world

When working on an application domain, it is beneficial to use the microservices software design pattern.

medium.com

TwitterClient.js

```
class TwitterRetweetClient {
    constructor(consumer_key, consumer_secret, access_key,
access_secret) {
        let Twitter = require('twitter');
        this.client = new Twitter({
            consumer_key: consumer_key,
            consumer_secret: consumer_secret,
            access_token_key: access_key,
            access_token_secret: access_secret
        });
    }

    execute = (tweets) => {
        for(let tweetId of tweets) {
            this.client.post('statuses/retweet/' + tweetId, function
(error, tweet, response) {
                if (!error) {
                    console.log(tweet);
                }
                else {
                    console.log("unable to retweet: " +
JSON.stringify(error));
                }
            });
        }
    };

    retweet = (find_hashtag, retweet_count) => {
        let self = this;
        this.client.get('search/tweets', {q: find_hashtag, count:
30, lag: 'en'}, function(error, tweets, response) {
            let selected_tweets = [];
            retweet_count = retweet_count > tweets.length ?
tweets.length: retweet_count;
            for(let tweet of tweets.statuses) {

                if(selected_tweets.length === retweet_count) {
                    self.execute(selected_tweets);
                    return true;
                }

                //do not retweet sensitive
                if(tweet.possibly_sensitive)
                    continue;

                //only retweet english
                if(tweet.lang !== 'en')
                    continue;

                //dont retweet spammy hashtags
                if((tweet.text.match(/#/g) || []).length > 2)
                    continue;

                console.log(tweet.text);
                selected_tweets.push(tweet.id_str);
            }
```

```
                console.log("Unable to find enough tweets");
                return false;
            });
        };
    }

    module.exports = TwitterRetweetClient;
```

I basically search for a hashtag and try to filter out the more spammy tweets.

Why not check out one of my other posts: Importing Existing Infrastructure into Terraform

It is always much simpler to create the terraform first, you would only ever import when you need to do something…

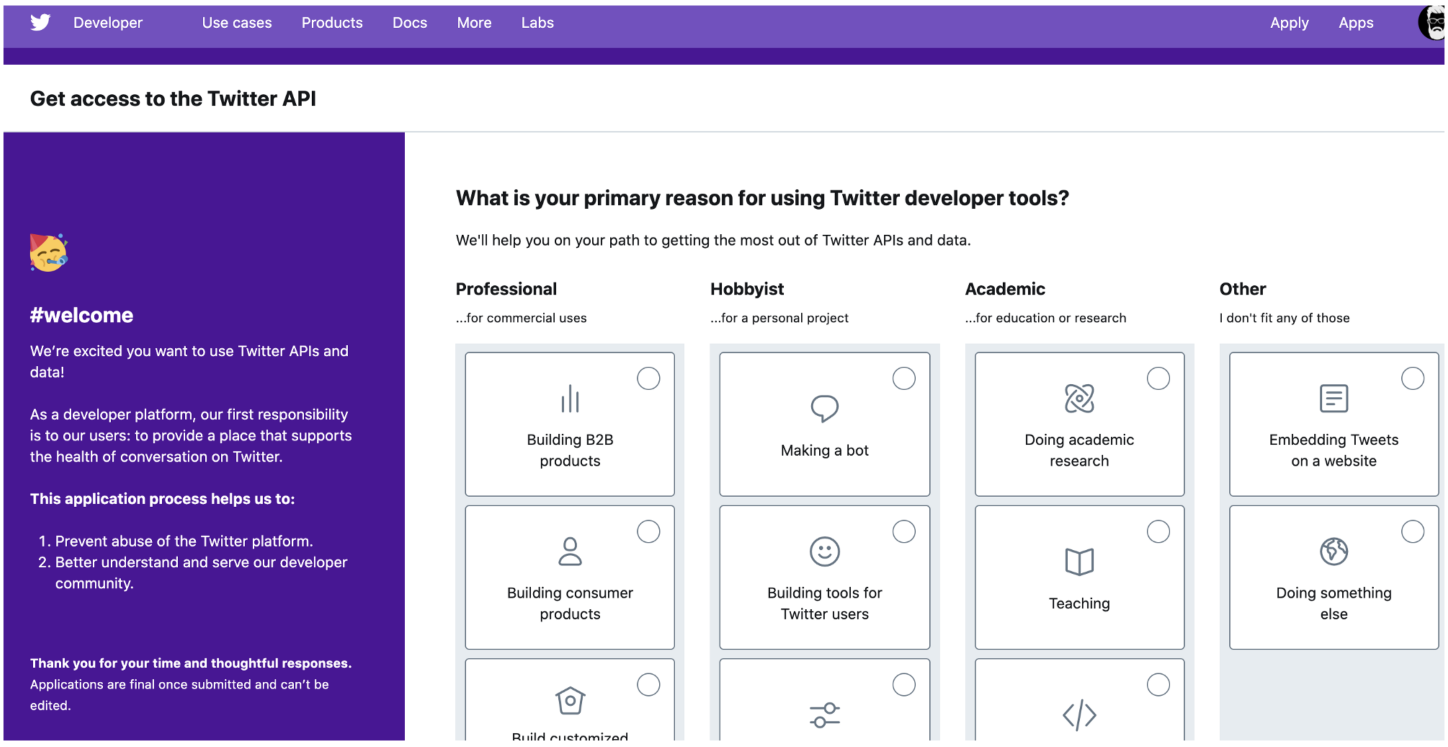
medium.com

. . .

Signing up for a developer account

In order to send tweets using code, you will need to sign up to create a developer account, the process is discussed here:

https://apps.twitter.com/



Get access to the Twitter API

#welcome

We're excited you want to use Twitter APIs and data!

As a developer platform, our first responsibility is to our users: to provide a place that supports the health of conversation on Twitter.

This application process helps us to:

1. Prevent abuse of the Twitter platform.
2. Better understand and serve our developer community.

Thank you for your time and thoughtful responses. Applications are final once submitted and can't be edited.

What is your primary reason for using Twitter developer tools?

We'll help you on your path to getting the most out of Twitter APIs and data.

Professional	Hobbyist	Academic	Other
...for commercial uses	...for a personal project	...for education or research	I don't fit any of those
Building B2B products	Making a bot	Doing academic research	Embedding Tweets on a website
Building consumer products	Building tools for Twitter users	Teaching	Doing something else
Build customized			

Once signed up, make sure to store the:

- Access Token
- Access Token Secret
- Api Key
- Api Key Secret.

. . .

Testing locally

So in order to test this all works, I basically invoke the handler manually, setting the environment variables:

At the end of index.js I add

```
exports.handler();
```

And run the function as:

```
TWITTER_CONSUMER_KEY=xxxxU5FaJU3bgDz6nGt9xxxxx
TWITTER_CONSUMER_SECRET=XXXXXXXXaQsdHuaC4h3oR3URRqYd0jnqoBVcd9HEV1XX
XXXXXX TWITTER_ACCESS_KEY=0000000051433271300-
XXXXXXXXRRUdtAMtHoAHIVXXXXXXXXX
TWITTER_ACCESS_SECRET=xxxxxxxxGalNApvG3yDwPtY2szibxW7TLEImXXXXXXXX
FIND_HASHTAG=#serverless RETWEET_COUNT=5 npm start
```

This will search for up to 30 tweets with the hashtag #serverless, ad will retweet up to 5 of them.

. . .

Building for Lambda

Since node by default will install its packages to node_modules directory, it is just a case of zipping up the directory and pushing to AWS.

Again this is a fairly straightforward thing to do with terraform.

```
data "archive_file" "lambda_zip" {
    type        = "zip"
    source_dir  = "./"
    output_path = "lambda.zip"
}

resource aws_lambda_function my_lambda {
  filename = "lambda.zip"
  source_code_hash =
data.archive_file.lambda_zip.output_base64sha256
  function_name = "my_lambda"
  role = aws_iam_role.lambda_role.arn
  description = "Some AWS lambda"
  handler = "index.handler"
  runtime = "nodejs12.x"
}

resource aws_iam_role lambda_role {
  name = "lambda-role"
  assume_role_policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
  {
    "Action": "sts:AssumeRole",
    "Principal": {
      "Service": "lambda.amazonaws.com"
    },
    "Effect": "Allow",
    "Sid": ""
  }]
}
EOF
}

resource "aws_iam_policy" "policy" {
  name = "policy"
  policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogGroup",
      "logs:CreateLogStream",
      "logs:PutLogEvents"
    ],
    "Resource": "*"
  }]
}
EOF
}

resource aws_iam_policy_attachment policy_attachment {
  name       = "attachment"
  roles      = [aws_iam_role.lambda_role.name]
  policy_arn = aws_iam_policy.policy.arn
}
```

And there you have it, a twitter bot, which retweets a specific hashtag every hour.

Why not check out one of my other posts: Using Microservice Patterns in an increasingly Serverless world

When working on an application domain, it is beneficial to use the microservices software design pattern.

medium.com

Graphic Attributions:
https://www.freepik.com/free-photos-vectors/business
macrovector

Serverless AWS Bots Twitter Terraform