USING APIS TO LAUNCH INSTANCES

Mike Macdonald

www.mike-macdonald.co.uk

CONTENTS

1. Brief 2
2. Explanation 2
3. Lambda function creation 3
   3.1 Lambda for starting EC2s 3
   3.2 Lambda for stopping EC2s 4
   3.3 Timeout for lambda functions 5
4. Creating an http api 6
   4.1 Create the API and its Integrations 6
   4.2 Configure the routes 7
   4.3 Configure the stages 7
   4.4 HTTP API Overview 8
5. Configure IAM to allow ec2 changes 8
6. Practical Results 9
7. EventBridge action 11
8. Conclusion 12

1. BRIEF

As part of a personal project that I was working on with someone else I wanted to create an API that would call lambda functions to start EC2s with specific tags (related to the project). Then to prevent extra costs being incurred, each day to stop the EC2s (in case someone had forgotten) at midnight each night.

2. EXPLANATION

APIs (Application Programming Interfaces) are a way of seamlessly connecting front end and back end services without exposing the backend to the public and therefore creating security risks. They can also help isolate the business logic of a service to allow logic changes in only one place. APIs are used everywhere, every day. For example, the Spotify AP allows developers to access music catalog data, user playlists, playback controls, and other music-related features. It enables integration with the Spotify platform, allowing applications to retrieve music recommendations, create playlists, and control playback. Using AWS API Gateway we can create various types of API (including REST and HTTP) to call other AWS functions such as Lambda.

In this example, I wanted to create an API with two methods that would call two different lambda functions. One method would start EC2s if they were stopped, and the other would stop EC2s if they were running. The user would go to the API link and the appropriate path. This invokes the appropriate Lambda function which, in turn, starts or stops any EC2 instance with a tag key of “Project” and a value of “projectx”.

The key stages for this are as follows:

1. Create the two lambda fuctions to filter by tag and then start and stop the EC2s that meet the criteria.
2. Create the API – Create an HTTP API with two ANY methods (/start and /stop) that triggers the appropriate lambda function when the API is called.
3. Integrate the API methods with the appropriate lambda function.
4. Ensure there is an IAM role to allow the API to trigger the Lambda function
5. Ensure there is an IAM role to allow the lambda functions to describe change the state of EC2 instances.

One part often overlooked is the creation of the IAM roles to give each part permission to complete its task.

3. LAMBDA FUNCTION CREATION

3.1 LAMBDA FOR STARTING EC2S

The Lambda function was written in python and the basic process is:

1. Set the filters (stopped instances with the project tag: “projectx”)
2. Get the Instance Ids that match the filters (Stopped and “projectx” tag)
3. Start the instances and return the IDs.
4. If there aren’t instances that match the filters return that there aren’t any.
5. If it fails return an error.

Note: This code is specifically designed to work in eu-west-2 region. If this were to be run elsewhere this would need to be changed.

Here is the code:

```
import json
import boto3


def lambda_handler(event, context):
    # Specify your desired filter criteria
    filters = [{'Name': 'instance-state-name', 'Values': ['stopped']},
               {'Name': 'tag:Project', 'Values': ['projectx']}]

    # Create an EC2 client
    ec2 = boto3.client('ec2', region_name='eu-west-2')

    try:
        # Describe instances matching the specified filters
        response = ec2.describe_instances(Filters=filters)
        instances = [instance['InstanceId'] for reservation in
response['Reservations']
                     for instance in reservation['Instances']]

        if instances:
            # Start the instances
            ec2.start_instances(InstanceIds=instances)


            return {
                'statusCode': 200,
                'body': json.dumps(f'Success: Started instances - {instances}')
            }
```

```
        else:
            return {
                'statusCode': 200,
                'body': json.dumps('Info: No instances matching the filter criteria')
            }
    except Exception as e:
        return {
            'statusCode': 500,
            'body': json.dumps(f'Error: {str(e)}')
        }
```

3.2 LAMBDA FOR STOPPING EC2S

The lambda function for stopping the EC2s is nearly identical except it has ec2.stop_instances instead of ec2.start_instances.

```
import json
import boto3

def lambda_handler(event, context):
    # Specify your desired filter criteria
    filters = [{'Name': 'instance-state-name', 'Values': ['running']},
               {'Name': 'tag:Project', 'Values': ['projectx']}]

    # Create an EC2 client
    ec2 = boto3.client('ec2', region_name='eu-west-2')

    try:
        # Describe instances matching the specified filters
        response = ec2.describe_instances(Filters=filters)
        instances = [instance['InstanceId'] for reservation in
response['Reservations']
                     for instance in reservation['Instances']]

        if instances:
            # Stop the instances
            ec2.stop_instances(InstanceIds=instances)

            return {
                'statusCode': 200,
                'body': json.dumps(f'Success: Stopped instances - {instances}')
            }
        else:
```

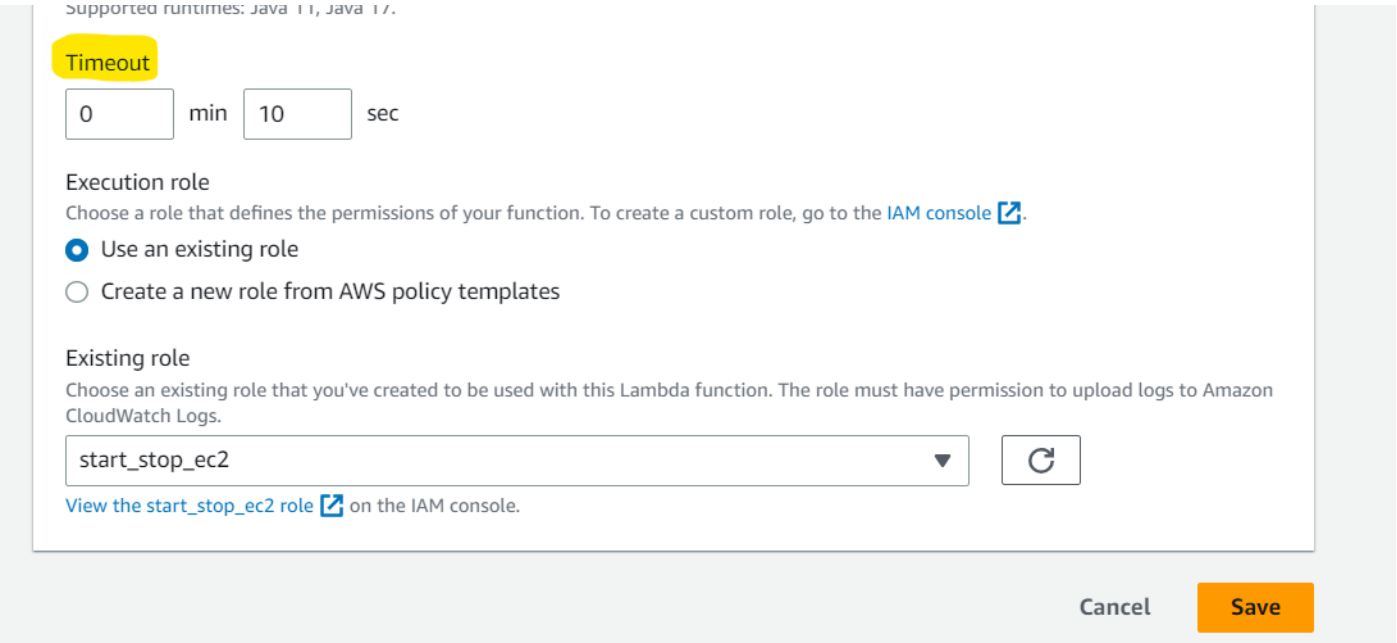
```
        return {
            'statusCode': 200,
            'body': json.dumps('Info: No instances matching the filter criteria')
        }
    except Exception as e:
        return {
            'statusCode': 500,
            'body': json.dumps(f'Error: {str(e)}')
        }
```

3.3 TIMEOUT FOR LAMBDA FUNCTIONS

Sometimes Lambda functions can be created correctly but either not work, or return an error.

This can be due to the functions *timeout configuration.* When creating a lambda function the default timeout is 3s. However depending on the function this can be too short a time for the function to complete so it's critical to change this appropriately so that the Lambda function doesn't timeout before it has completed.

This can be done in *Configuration – General Configuration – Edit*



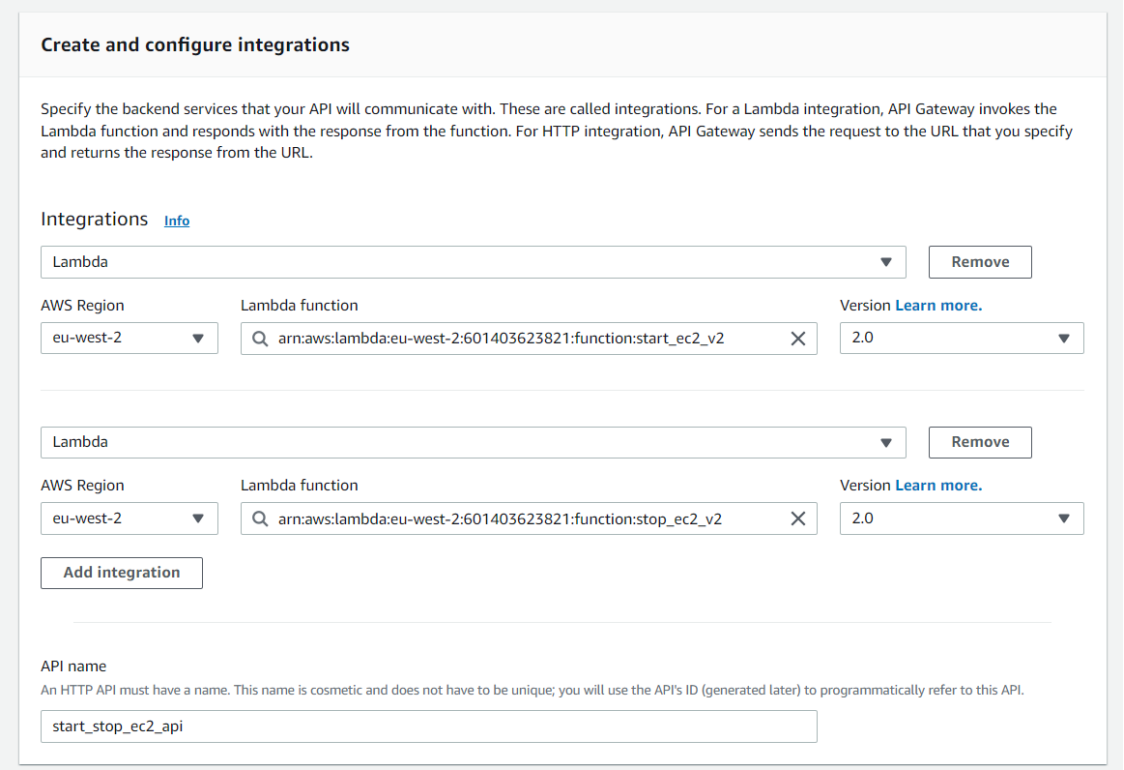
4. CREATING AN HTTP API

API Gateway can make various types of API including HTTP and REST APIs. HTTP APIs can only have Lambda and HTTP end points. They're cost effective, have low latency and easier to set up than REST APIs but they lack some of the advanced features that REST APIs provide including some security (AWS WAF and backend authentication certificates are only available through REST APIs).

There are three main steps to create an HTTP API.

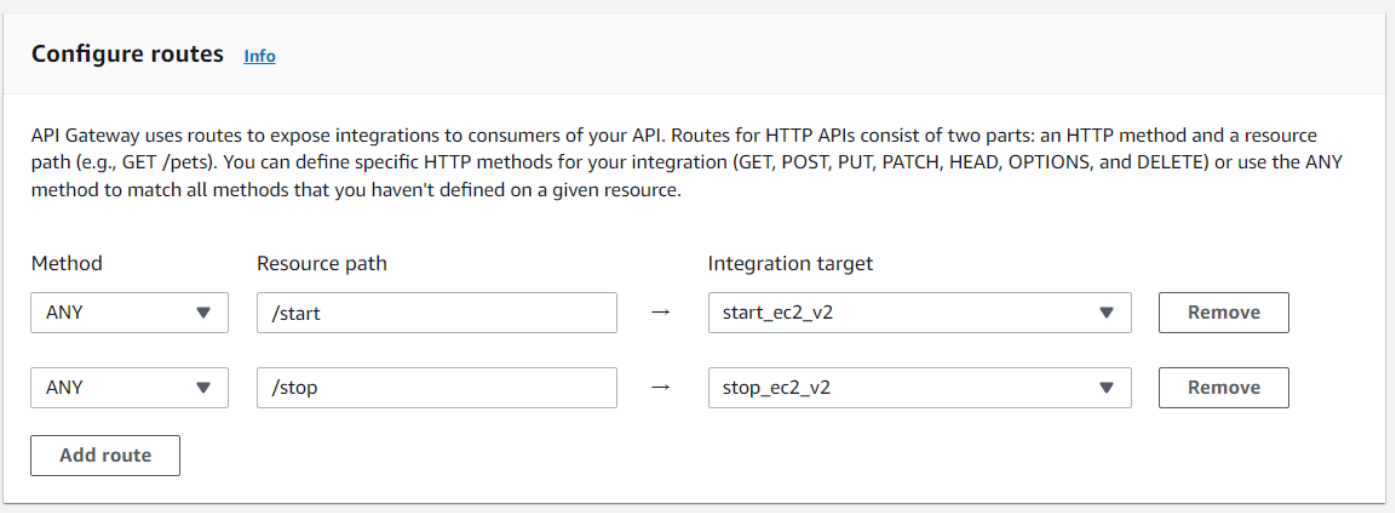
1. Create the API with its integrations
2. Configure the routes
3. Define the stages

4.1 CREATE THE API AND ITS INTEGRATIONS



With this API I have two integrations. One to start and one to stop the EC2s

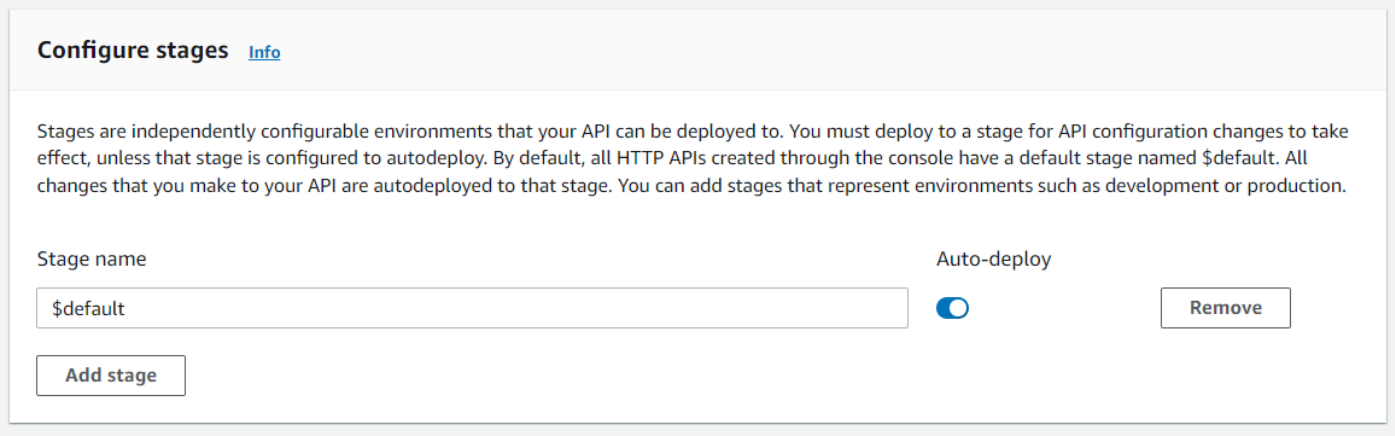
4.2 CONFIGURE THE ROUTES



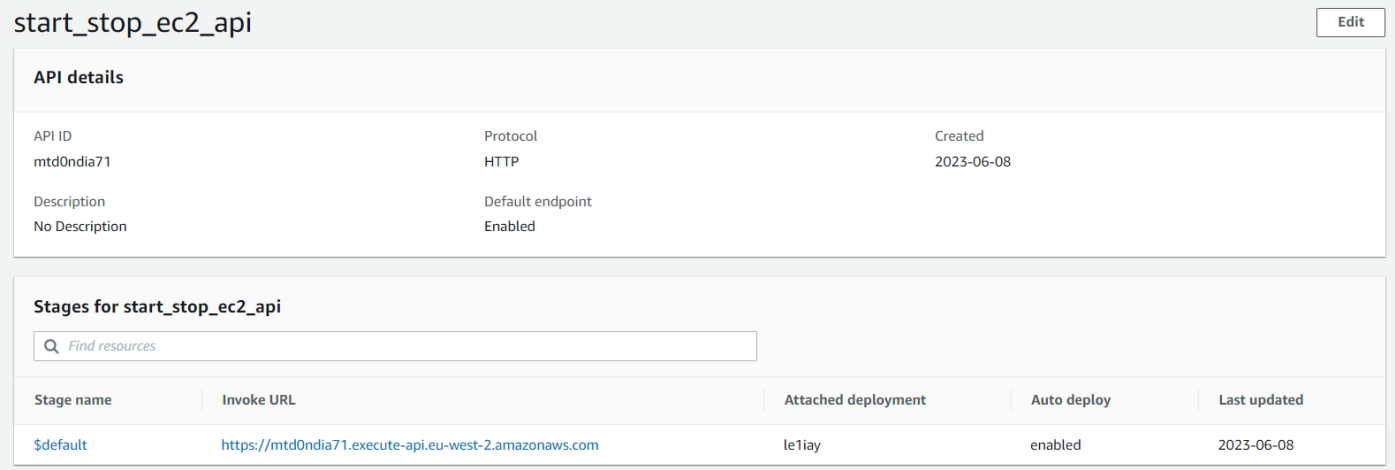
Here I've defined the path that will call the API. Later I will have a URL and if I use the path URL/start it will trigger the Lambda function to start the EC2s and the same for URL/stop.

4.3 CONFIGURE THE STAGES

When developing an API you can have different stages which help managed the lifecycle of an API. Often you might have a "Development" stage, a "Testing" stage, a "Production" stage and then versioning as the API is updated over time. With creating an HTTP API the API auto-deploys to a default stage but that is changeable if desired.



4.4 HTTP API OVERVIEW



This completed HTTP API gives an invocation URL that can be called with the appropriate path to invoke the API and its respective methods.

5. CONFIGURE IAM TO ALLOW EC2 CHANGES

It's not enough to just create a lambda function to control the EC2s. A lambda function assigned to it the appropriate IAM policy via an IAM Role. The IAM role need permissions to describe EC2 instances and then start and stop them as appropriate.

I have created this role with the below policy.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "EC2StartStopInstances",
         "Effect": "Allow",
         "Action": [
            "ec2:DescribeInstances",
            "ec2:StartInstances",
            "ec2:StopInstances"
         ],
         "Resource": "*"
      }
   ]
}
```

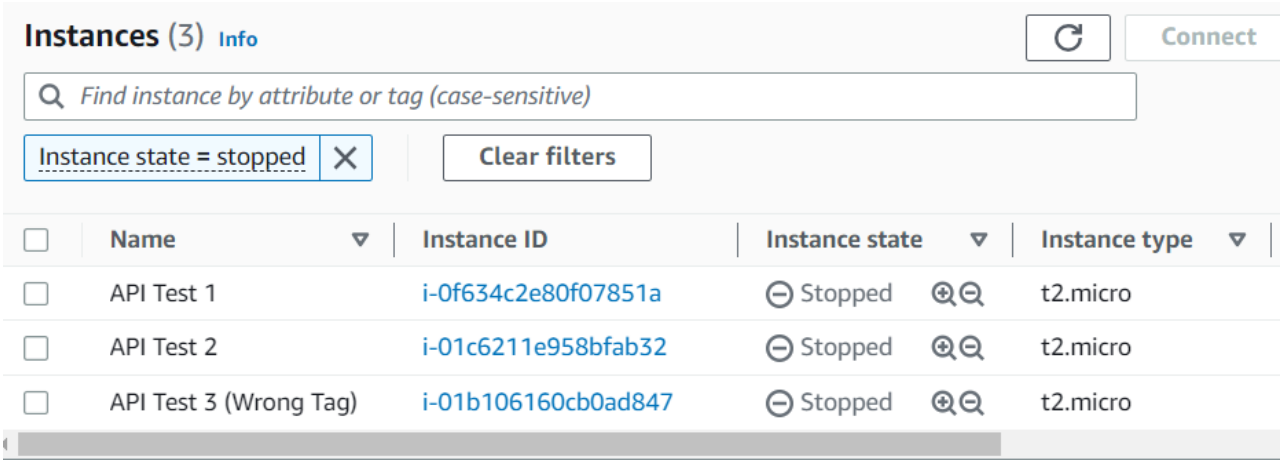
This policy allows the lambda function to describe the instances of any and all EC2s within the account and then start and stop them. This is a very basic policy and isn't inherently very secure, but was created purely

for this demonstration. To make it more secure the resource could be restricted by limiting based on region, account etc.

The API also needs permissions to invoke a lambda function however this can be done automatically during the creation of the API.

6. PRACTICAL RESULTS

I have created 3 t2.micro instances for the test. Two of which have the Project: projectx tag and one doesn't.



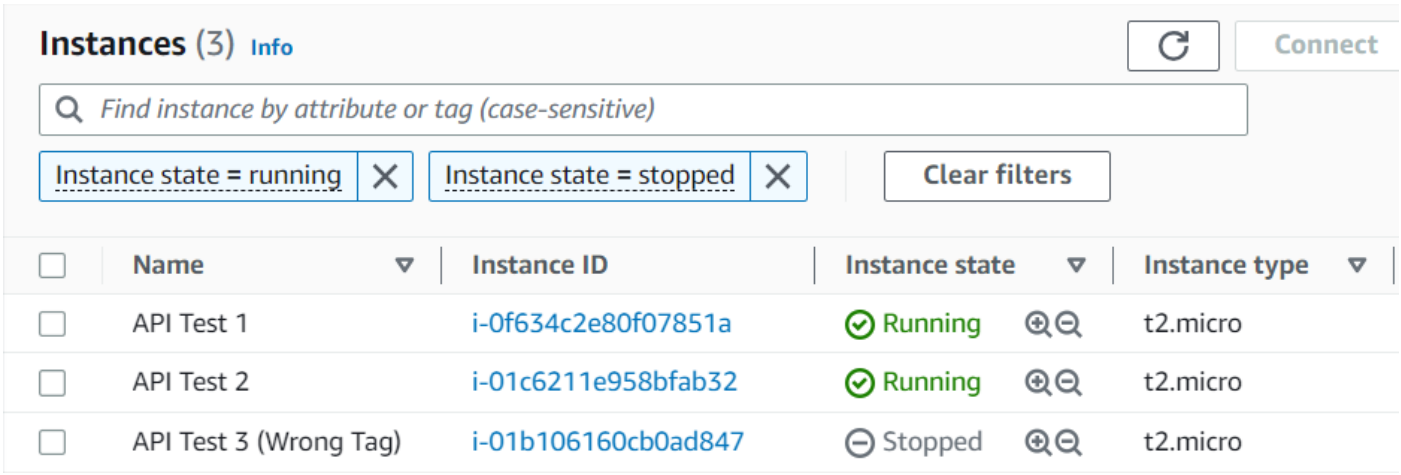
Using the API invoke URL with the path "/start" this will then start two out of the three instances.

https://mtd0ndia71.execute-api.eu-west-2.amazonaws.com/start

The page returns:

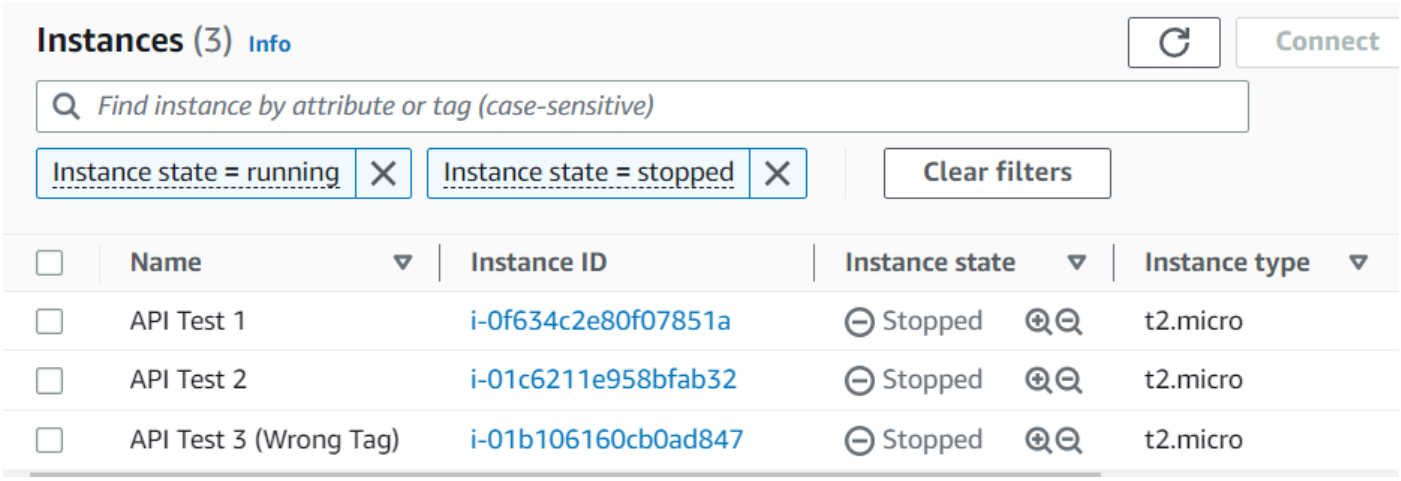
```
"Success: Started instances - ['i-0f634c2e80f07851a', 'i-01c6211e958bfab32']"
```

We can see that two out of the three instances have started:



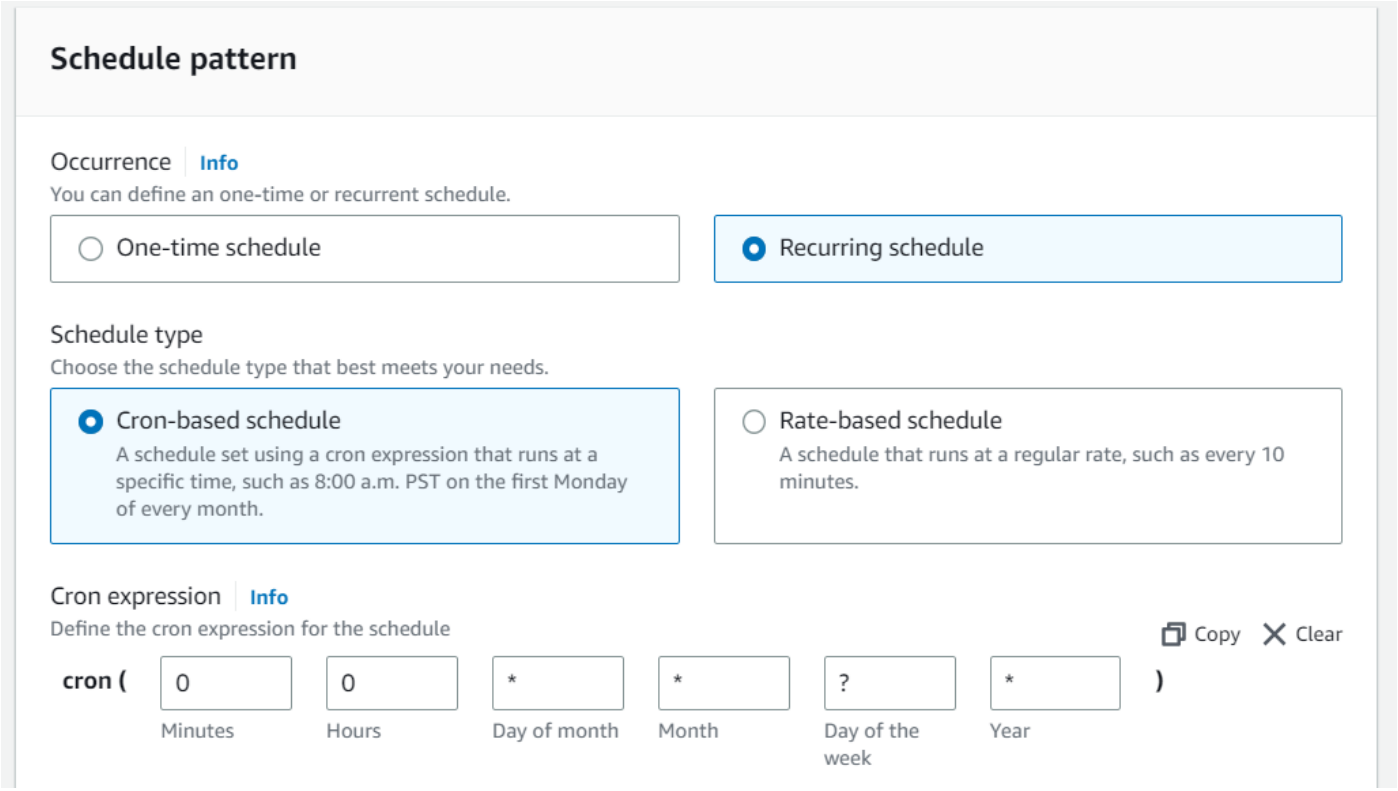
When the URL is invoked with "/stop" as the path these EC2s stop.

```
"Success: Stopped instances - ['i-0f634c2e80f07851a', 'i-01c6211e958bfab32']"
```

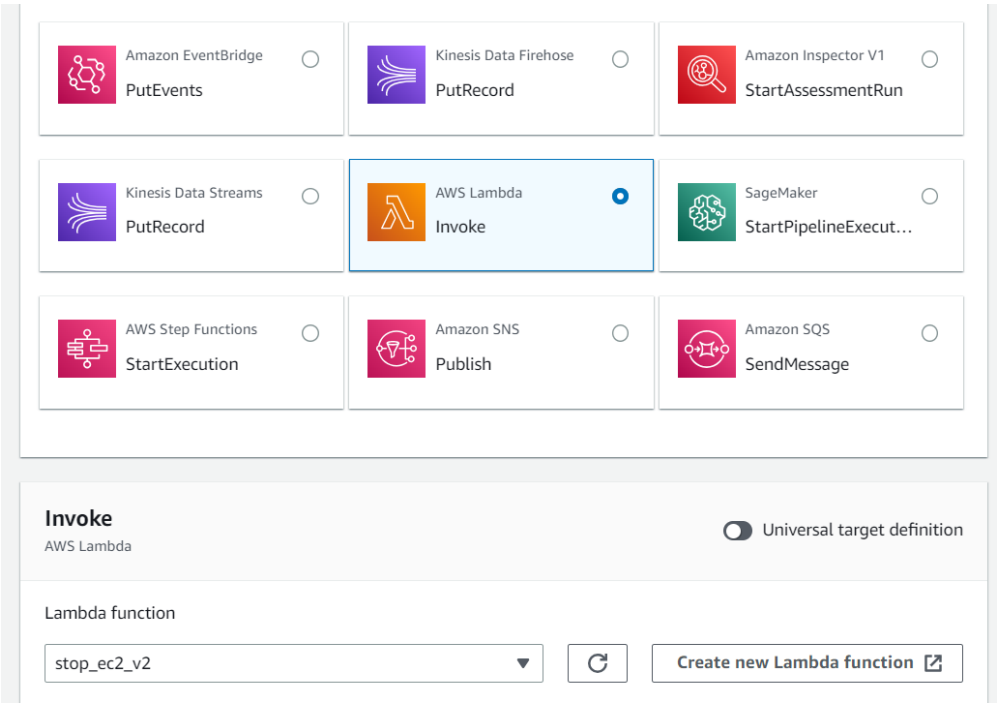


7. EVENTBRIDGE ACTION

Finally I created an EventBridge Schedule to automatically invoke the stop Lambda function every night at midnight. This is done in on a schedule and can be done using a Cron expression to define how often to schedule it.



There are large varieties of integrations that EventBridge provides but in this case I integrated with the previously created Lambda function.



This will now automatically stop any EC2 with the Project: projectx tag at midnight every night. It's possible to write the Lambda return value to a CloudWatch log to then see how often the EC2s needed shutting down. That way you can gently remind your developers to shut them down themselves…

8. CONCLUSION

It's great to scratch the surface of what APIs and Lambda functions can do together, how they'd be useful in real world applications and the power of serverless applications to make life easier and more efficient. I'm looing forward to diving deeper into APIs and how they can be used.

There are a number of things that could be considered to take this project further. These include using a custom domain name to call the API, logging calls in CloudWatch logs, creating a web interface that allows users to select specific tags to start or stop and user authentication and access control to ensure only specific people can call the API.