



# USING APIS TO LAUNCH INSTANCES



Mike Macdonald

## 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}')
            }
    except:

```

```

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*

Supported runtimes: Java 11, Java 17.

**Timeout**

0 min 10 sec

**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☒ Use an existing role

☐ Create a new role from AWS policy templates

**Existing role**  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

start\_stop\_ec2

[View the start\\_stop\\_ec2 role](#) on the IAM console.

Cancel Save

## 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

#### Create and configure integrations

Specify the backend services that your API will communicate with. These are called integrations. For a Lambda integration, API Gateway invokes the Lambda function and responds with the response from the function. For HTTP integration, API Gateway sends the request to the URL that you specify and returns the response from the URL.

Integrations [Info](#)

Lambda

Remove

AWS Region

Lambda function

Version [Learn more.](#)

eu-west-2

arn:aws:lambda:eu-west-2:601403623821:function:start\_ec2\_v2

×

2.0

Lambda

Remove

AWS Region

Lambda function

Version [Learn more.](#)

eu-west-2

arn:aws:lambda:eu-west-2:601403623821:function:stop\_ec2\_v2

×

2.0

Add integration

API name

An HTTP API must have a name. This name is cosmetic and does not have to be unique; you will use the API's ID (generated later) to programmatically refer to this API.

start\_stop\_ec2\_api

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

## 4.2 CONFIGURE THE ROUTES

**Configure routes** [Info](#)

API Gateway uses routes to expose integrations to consumers of your API. Routes for HTTP APIs consist of two parts: an HTTP method and a resource path (e.g., GET /pets). You can define specific HTTP methods for your integration (GET, POST, PUT, PATCH, HEAD, OPTIONS, and DELETE) or use the ANY method to match all methods that you haven't defined on a given resource.

Method	Resource path	Integration target	
ANY ▼	/start	→ start_ec2_v2 ▼	<button>Remove</button>
ANY ▼	/stop	→ stop_ec2_v2 ▼	<button>Remove</button>

Add route

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.

**Configure stages** [Info](#)

Stages are independently configurable environments that your API can be deployed to. You must deploy to a stage for API configuration changes to take effect, unless that stage is configured to autodeploy. By default, all HTTP APIs created through the console have a default stage named \$default. All changes that you make to your API are autodeployed to that stage. You can add stages that represent environments such as development or production.

Stage name	Auto-deploy	
\$default	<input checked="" type="checkbox"/>	<button>Remove</button>

Add stage



## 4.4 HTTP API OVERVIEW

start\_stop\_ec2\_api

Edit

API details

API ID	Protocol	Created
mtd0ndia71	HTTP	2023-06-08
Description	Default endpoint	
No Description	Enabled	

Stages for start\_stop\_ec2\_api

Find resources

Stage name	Invoke URL	Attached deployment	Auto deploy	Last updated
\$default	https://mtd0ndia71.execute-api.eu-west-2.amazonaws.com	le1iay	enabled	2023-06-08

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.

Instances (3) <a href="#">Info</a>							<a href="#">Connect</a>
<input type="text" value="Find instance by attribute or tag (case-sensitive)"/>							
<div>Instance state = stopped </div> <div>Clear filters</div>							
<input type="checkbox"/>	Name ▾	Instance ID	Instance state ▾	Instance type ▾			
<input type="checkbox"/>	API Test 1	<a href="#">i-0f634c2e80f07851a</a>	Stopped	t2.micro			
<input type="checkbox"/>	API Test 2	<a href="#">i-01c6211e958bfab32</a>	Stopped	t2.micro			
<input type="checkbox"/>	API Test 3 (Wrong Tag)	<a href="#">i-01b106160cb0ad847</a>	Stopped	t2.micro			













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:

Instances (3) <a href="#">Info</a>						<a href="#">Connect</a>
<input type="text" value="Find instance by attribute or tag (case-sensitive)"/>						
<div><div>Instance state = running </div><div>Instance state = stopped </div><div>Clear filters</div></div>						
<input type="checkbox"/>	Name ▾	Instance ID	Instance state ▾	Instance type ▾		
<input type="checkbox"/>	API Test 1	i-0f634c2e80f07851a	 Running  	t2.micro		
<input type="checkbox"/>	API Test 2	i-01c6211e958bfab32	 Running  	t2.micro		
<input type="checkbox"/>	API Test 3 (Wrong Tag)	i-01b106160cb0ad847	 Stopped  	t2.micro		

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

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

Instances (3) <a href="#">Info</a>						<a href="#">Connect</a>
<input type="text" value="Find instance by attribute or tag (case-sensitive)"/>						
<div><div>Instance state = running </div><div>Instance state = stopped </div><div>Clear filters</div></div>						
<input type="checkbox"/>	Name ▾	Instance ID	Instance state ▾	Instance type ▾		
<input type="checkbox"/>	API Test 1	i-0f634c2e80f07851a	 Stopped  	t2.micro		
<input type="checkbox"/>	API Test 2	i-01c6211e958bfab32	 Stopped  	t2.micro		
<input type="checkbox"/>	API Test 3 (Wrong Tag)	i-01b106160cb0ad847	 Stopped  	t2.micro		

## 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.

### Schedule pattern

Occurrence [Info](#)  
You can define an one-time or recurrent schedule.

☐ One-time schedule

☒ Recurring schedule

Schedule type  
Choose the schedule type that best meets your needs.

☒ Cron-based schedule  
A schedule set using a cron expression that runs at a specific time, such as 8:00 a.m. PST on the first Monday of every month.

☐ Rate-based schedule  
A schedule that runs at a regular rate, such as every 10 minutes.

Cron expression [Info](#)  
Define the cron expression for the schedule

cron (

0

Minutes

0

Hours

\*

Day of month

\*

Month

?

Day of the week

\*

Year

)

Copy

Clear

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

Amazon EventBridge PutEvents	Kinesis Data Firehose PutRecord	Amazon Inspector V1 StartAssessmentRun
Kinesis Data Streams PutRecord	AWS Lambda Invoke	SageMaker StartPipelineExecution
AWS Step Functions StartExecution	Amazon SNS Publish	Amazon SQS SendMessage

### Invoke

AWS Lambda

Universal target definition

Lambda function

stop\_ec2\_v2

Create new Lambda function

11

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 looking 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.