# AWS Deployment - Troubleshooting & FAQ Guide

This document provides solutions to common issues and answers frequently asked questions that may arise during the deployment of the Serverless News Aggregator on AWS.

## Common Issues & Solutions

### Issue 1: Website Prompts to "Save" Instead of Displaying

* **Symptom:** When you navigate to your S3 website URL, the browser opens a "Save File" dialog for index.html instead of showing the webpage.
* **Cause:** The index.html file in your S3 bucket has the wrong Content-Type metadata. The browser thinks it's a generic file (binary/octet-stream) instead of a webpage (text/html).
* **Solution:**
  1. Go to your S3 bucket in the AWS Console.
  2. Select the index.html file by checking the box next to it.
  3. Click **Actions** -> **Edit metadata**.
  4. Scroll down to the "Metadata" section.
  5. Change the Content-Type value to **text/html**.
  6. Click **Save changes**. Refresh your website URL.

### Issue 2: Website Shows "An Error Occurred - No articles found"

* **Symptom:** The website loads but displays an error saying no articles were found.
* **Cause:** This is expected right after deployment. The FetchNewsFunction runs on an hourly schedule and hasn't had a chance to run yet, so your DynamoDB database is empty.
* **Solution:** Manually run the fetcher function once to populate the database.
  1. Go to the AWS Lambda console and select your FetchNewsFunction.
  2. Navigate to the **Test** tab.
  3. Create a new test event. You can name it ManualRun and leave the default {} JSON content.
  4. Click **Save**, and then click the **Test** button.
  5. Wait for the execution to complete (it may take up to a minute). If it succeeds, your database will be populated.
  6. Refresh your website. The articles should now appear.

### Issue 3: Lambda Fails with "Unable to import module 'lambda\_function'"

* **Symptom:** When testing your Lambda function, the execution fails with an ImportModuleError.
* **Cause:** Lambda's configuration is looking for a file named lambda\_function.py by default, but our files have different names (e.g., fetch\_news\_lambda.py).
* **Solution:** Update the Lambda "Handler" setting.
  1. In your Lambda function's configuration, find the **Runtime settings** panel and click **Edit**.
  2. The **Handler** value is in the format filename.function\_name.
  3. Update it to match your actual file and function. For example: **fetch\_news\_lambda.lambda\_handler**.
  4. Click **Save** and test the function again.

## Deployment FAQs

#### Q: How do I create the Lambda .zip file correctly?

**A:** You must zip the **contents** of the package folder, not the folder itself.

1. Open the folder (e.g., fetcher-package).
2. Select all files and subfolders inside (Ctrl+A or Cmd+A).
3. Right-click on the selected items and choose **Send to -> Compressed (zipped) folder** (Windows) or **Compress** (macOS).

#### Q: Do I need to enable CORS for each API resource separately?

**A:** Yes. You must select /news, click **Actions -> Enable CORS**, and complete the process. Then, you must select /track-view and do the exact same thing.

#### Q: Do I need to deploy each API resource separately?

**A:** No. You deploy the entire API at once. After you have configured all your resources (/news, /track-view, etc.), select the top-level root resource (/), click **Actions -> Deploy API**, and choose your stage. This will publish all changes to all resources.

#### Q: What is an API Gateway "Stage"?

**A:** A stage is like a version or environment for your API (e.g., dev, test, prod). When you deploy your API to a stage, you get a unique URL (the "Invoke URL") for that version, which you use in your application. For this project, creating a single stage named prod is sufficient.

Issues I Faced

**Issue-1**

PS C:\Users\Srinu\OneDrive\Documents\Social\_Media\_News\_Aggregator\fetcher-package> pip install feedparser -t

Usage:

  pip install [options] <requirement specifier> [package-index-options] ...

  pip install [options] -r <requirements file> [package-index-options] ...

  pip install [options] [-e] <vcs project url> ...

  pip install [options] [-e] <local project path> ...

  pip install [options] <archive url/path> ...

**Solution:**

It looks like you're on the right track for creating the Lambda deployment package, but you've run into a small syntax issue with the pip command.

The error message you're seeing is pip's way of saying, "You used the -t flag, but you didn't tell me *where* to install the package."

The -t flag stands for --target, and it requires a directory path immediately after it. To install the package into the **current directory**, you need to add a dot (.) after -t.

**Here is the correct command:**

Bash

pip install feedparser -t .

**Why this works:**

* pip install feedparser: Tells pip to install the feedparser library.
* -t .: Tells pip to install it into the target (-t) directory specified, which is the current directory (.).

After you run this corrected command, you should see feedparser and its dependencies appear in your fetcher-package folder, right alongside your fetch\_news\_lambda.py file. Then you'll be ready to zip them up for deployment!

**Issue-2**

○      Zip the contents of

the fetcher-package folder. Do not zip

the folder itself.

**Solution**

You've hit on one of the most important and common tripping points when creating AWS Lambda packages! That instruction is crucial for your function to work correctly.

Here’s a simple breakdown of what it means and how to do it right.

**What it Means**

When AWS Lambda runs your code, it looks for your handler file (e.g., fetch\_news\_lambda.py) in the top-level directory of the ZIP file you upload.

* **Incorrect Way:** Zipping the fetcher-package folder itself.
  + The structure inside the ZIP becomes: fetcher-package/fetch\_news\_lambda.py
  + AWS Lambda will fail because it can't find fetch\_news\_lambda.py at the root.
* **Correct Way:** Zipping the *contents* of the fetcher-package folder.
  + The structure inside the ZIP becomes: fetch\_news\_lambda.py (and other folders like feedparser).
  + AWS Lambda will find the file immediately and run successfully.

**How to Do It Correctly**

Here are the step-by-step instructions for Windows and macOS:

**On Windows:**

1. **Open** the fetcher-package folder.
2. **Select all** the files and folders inside it (e.g., press Ctrl + A).
3. **Right-click** on any of the selected files.
4. Go to **Send to** > **Compressed (zipped) folder**.
5. Rename the new ZIP file to fetcher-package.zip.

**On macOS:**

1. **Open** the fetcher-package folder in Finder.
2. **Select all** the files and folders inside it (e.g., press Cmd + A).
3. **Right-click** on any of the selected files.
4. Click **Compress X Items**.
5. Rename the resulting Archive.zip file to fetcher-package.zip.

By following these steps, you ensure that your Python file and the necessary libraries are at the top level of the archive, which is exactly what Lambda needs.

**Issue -3**

1.    Enable CORS:

○      Select the /news resource. Click Actions and Enable CORS.

○      Click Enable CORS and

replace existing CORS headers.

○      Repeat for the /track-view resource.

**Solution**

You are absolutely correct. Those are the exact steps you need to follow to allow your S3-hosted website to communicate with your API Gateway.

Here's a bit more detail on why this is necessary and what's happening behind the scenes.

**Why is this necessary?**

For security reasons, web browsers enforce a "Same-Origin Policy." This policy prevents a web page from making requests to a different domain than the one it was served from. In your case:

* Your website is on an **S3 domain** (e.g., http://your-bucket-name.s3-website.us-east-1.amazonaws.com).
* Your API is on an **API Gateway domain** (e.g., https://abcdef123.execute-api.us-east-1.amazonaws.com).

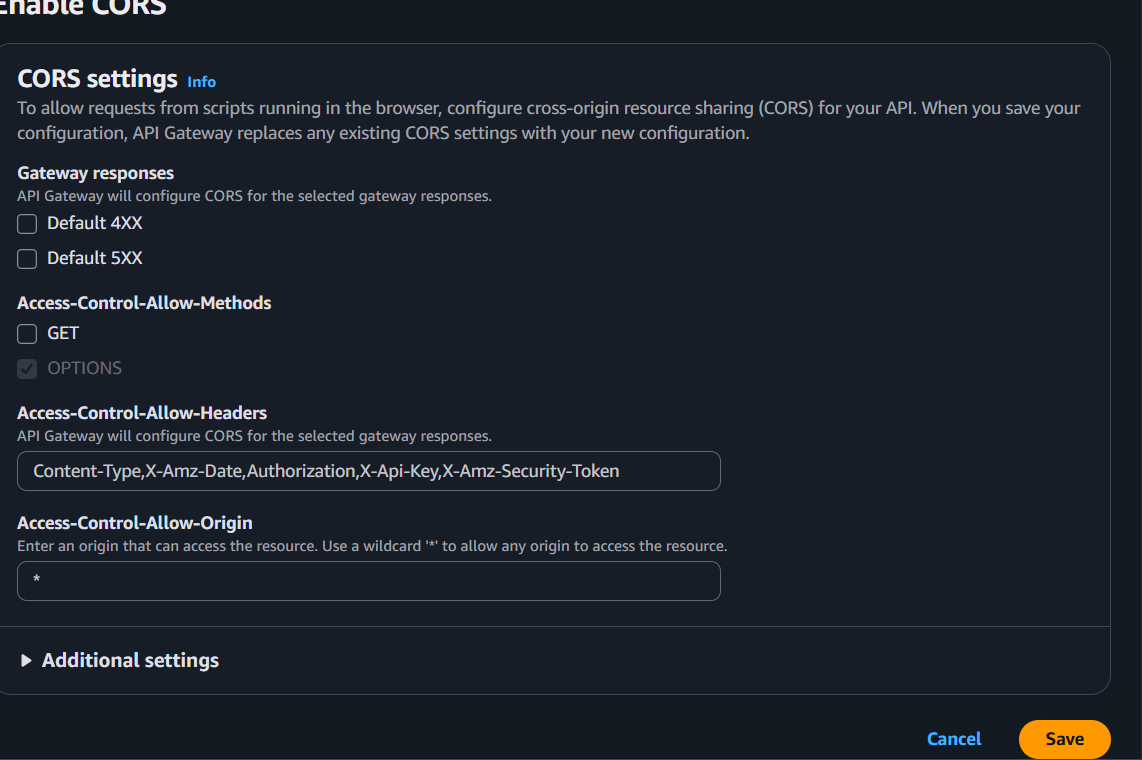
Because these are different domains, you need to enable Cross-Origin Resource Sharing (CORS) on your API to explicitly tell the browser, "It's okay for the S3 website to make requests to me."

**A Visual Guide to Your Steps**

Here is what the process looks like in the AWS Management Console:

1. **Select the Resource:** In your API Gateway, navigate to your API and select the /news resource in the "Resources" tree on the left.
2. **Enable CORS:** Click the **Actions** dropdown menu and select **Enable CORS**, just as your instructions say.
3. **Confirm and Replace:** A new screen will appear. The default settings are usually sufficient. Simply click the button at the bottom that says **"Enable CORS and replace existing CORS headers."**
4. **Repeat for /track-view:** After the process completes for /news, select the /track-view resource from the Resources tree and repeat the exact same steps.

By doing this, API Gateway automatically adds the necessary Access-Control-Allow-Origin headers to your API responses and creates an OPTIONS method for each resource. This OPTIONS method handles "preflight" requests that browsers send to check if the actual request is safe to send.



Excellent, you are on the correct screen! The settings you see in your screenshot are exactly what you should expect when enabling CORS in API Gateway.

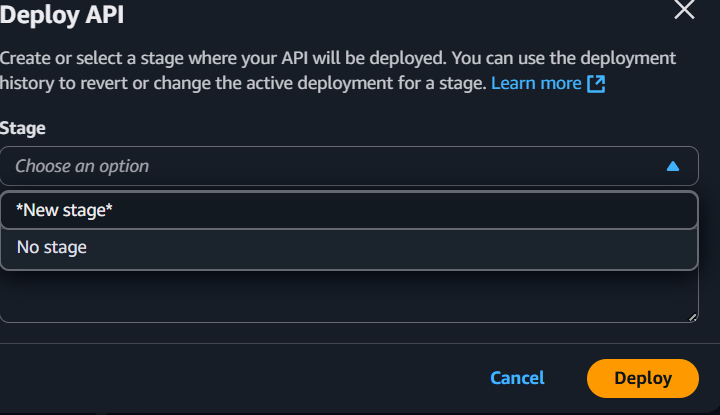
The \* in the Access-Control-Allow-Origin field is correct and means "allow any origin," which is what you need for your public S3 website.

**What to do now:**

1. **Click the "Save" button** at the bottom right of the screen.
2. **VERY IMPORTANT:** After you save, you must do two more things:
   * **Repeat for /track-view:** You have to do this exact same process for your other API resource. Select /track-view from the "Resources" list on the left, click "Actions," and enable CORS for it as well.
   * **Redeploy your API:** CORS changes do not take effect until you deploy the API. After enabling CORS for both /news and /track-view, click the **Actions** button again and select **Deploy API**. Choose your deployment stage (e.g., prod or default) and click **Deploy**.

Once you have redeployed the API, the "save" prompt on your website should disappear, and it should start fetching and displaying the news.

**Issue -3:Deplyoment issue on API gateway**



**Solution:**

You're at the final and most important step for making your API live! This screen is asking you to create a "deployment stage."

Think of a stage as a version or an environment for your API. Common names are dev (development), test, or prod (production). Since this is your live website, using prod is a good practice.

**Here's what you need to do on this screen:**

1. **Select Stage:** Click the dropdown and choose **\*New stage\*** (it's already selected in your screenshot).
2. **Enter Stage Name:** Two new fields will appear. In the **Stage name** field, type a simple name like **prod**.
3. **Deploy:** Click the **Deploy** button.

Once you click "Deploy," you will be taken to the Stage Editor. At the top of this new page, you will see a blue banner with the **Invoke URL**. This URL is the public endpoint for your live API.

**This Invoke URL is critical.** You will need to copy it and paste it into your index.html file to replace the placeholder API URL.

----------------------------------------------------------------------------------------------------

**Q. previously I selected news and deployed now instead of both of them I selected /.**

That's perfectly fine, and in fact, **that's the recommended way to do it!**

When you select the root resource (/) and click "Deploy API," you are telling API Gateway to deploy the **entire API**, which includes all the resources underneath it.

So, by deploying from /, you have successfully deployed both /news and /track-view at the same time.

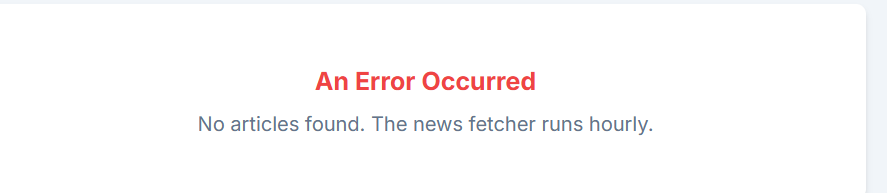
Think of it like a folder structure:

* / (Root)
  + /news (Child resource)
  + /track-view (Child resource)

Deploying the parent (/) automatically includes all the children. This is much more efficient than deploying each resource one by one.

**In summary: Your action was correct. Both of your API endpoints are now live.**

**Issue -4:**



**Solution:**

This error message is actually a good sign! It means:

1. Your S3 website is working.
2. Your API Gateway and ApiHandlerFunction are working correctly.
3. The API successfully queried your DynamoDB table and correctly reported back that it's empty.

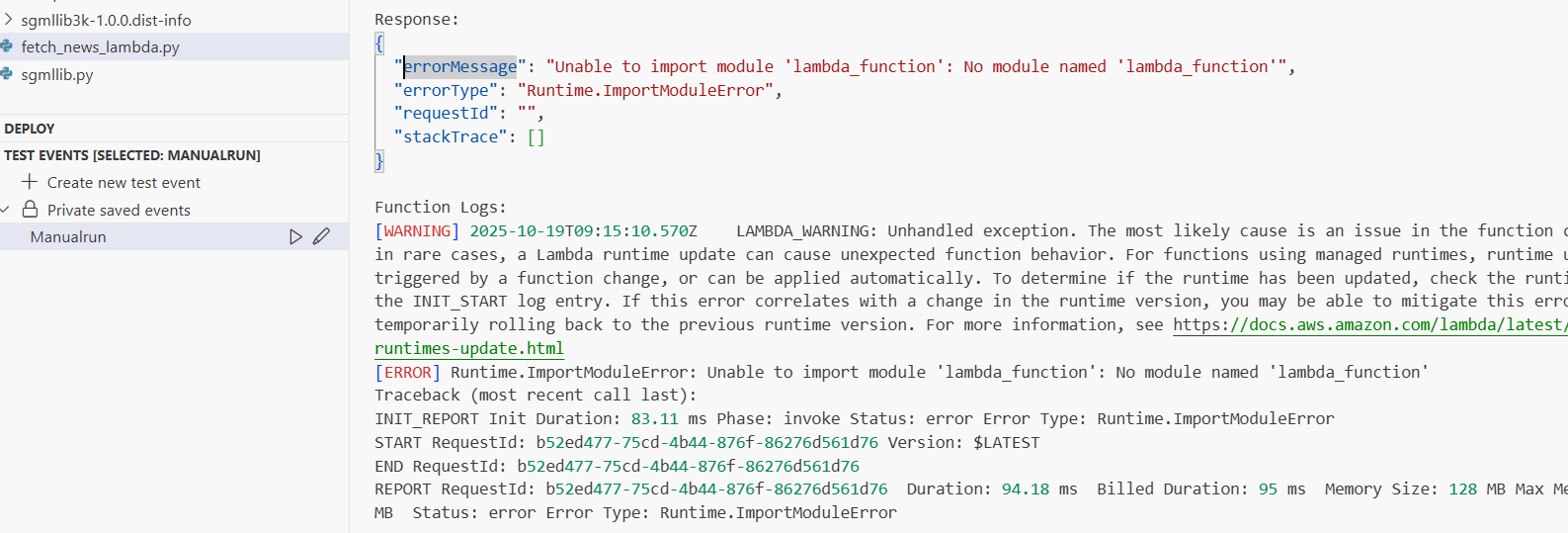
The problem is that your database has no news in it yet. According to your architecture, the FetchNewsFunction is set to run automatically every hour. Since you just deployed everything, it hasn't had a chance to run yet.

**The Solution: Run the Fetcher Manually Once**

You need to manually trigger your FetchNewsFunction one time to populate the database. Here’s how:

1. **Go to the Lambda Console:** Navigate to the AWS Lambda service in the AWS Console.
2. **Select FetchNewsFunction:** Click on your FetchNewsFunction from the list of functions.
3. **Configure a Test Event:**
   * You should see a **Test** tab near the top of the page. Click it.
   * You'll be asked to create a new test event.
   * Give the event a name, for example, ManualRun.
   * In the **Event JSON** box, you can leave the default content, which is just an empty JSON object: {}.
   * Click the **Save** button.
4. **Run the Test:**
   * Now, with your ManualRun event selected, click the blue **Test** button.
   * The function will now execute. This may take 20-30 seconds as it has to fetch data from many different websites.
5. **Check the Result:**
   * After it finishes, you will see an "Execution result" tab. If it says **"Succeeded"** in green, it worked! The log output will show details about the articles it fetched and saved.
   * If it failed, the logs will contain an error message that will help diagnose the problem (often an IAM role permission issue).
6. **Refresh Your Website:** Go back to your news aggregator website and refresh the page. The error message should be gone, and you should now see all the news articles!

**Issue -5:**

****

**Solution:**

This is a very common and solvable error in AWS Lambda! The screenshot you provided is extremely helpful.

The error message **"Unable to import module 'lambda\_function': No module named 'lambda\_function'"** tells us the exact problem. AWS Lambda is looking for a file named lambda\_function.py to start your code, but our file is named fetch\_news\_lambda.py.

The fix is simple: you just need to tell Lambda the correct filename.

**How to Fix the Handler Setting**

1. **Go to your FetchNewsFunction** in the AWS Lambda console.
2. Make sure you are on the **Code** tab (which you are in the screenshot).
3. Scroll down below the code editor until you see the **Runtime settings** panel.
4. Click the **Edit** button in the "Runtime settings" panel.
5. You will see a setting called **Handler**. This is what you need to change.
   * The current, incorrect value is likely lambda\_function.lambda\_handler.
   * Change it to: **fetch\_news\_lambda.lambda\_handler**

This tells Lambda: "Look for a file named **fetch\_news\_lambda**.py, and inside that file, run the function named **lambda\_handler**."

1. Click the **Save** button.

**After Saving**

Now, go back to the **Test** tab and click the blue **Test** button again. The ImportModuleError should be gone, and your function should execute successfully to fetch the news.