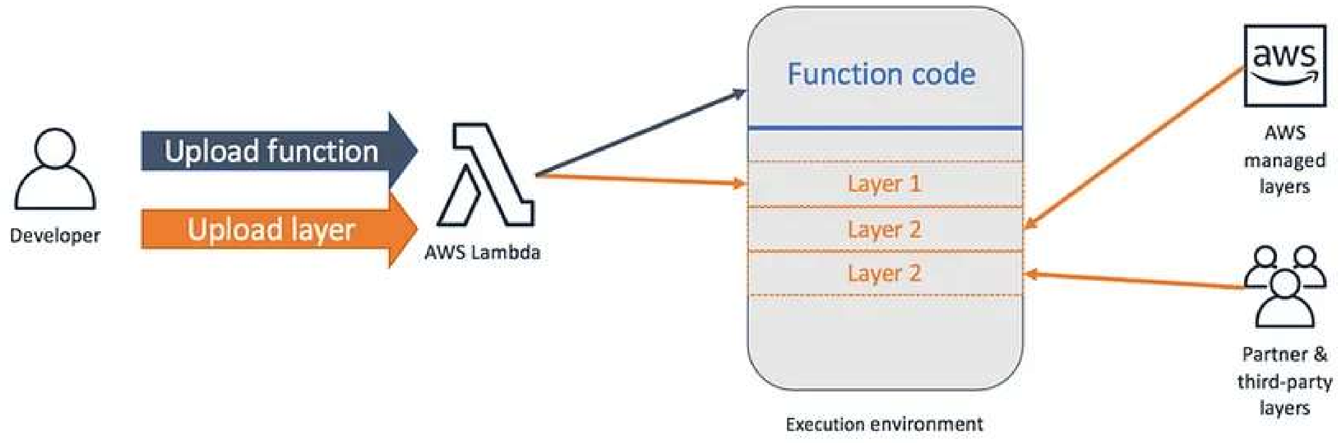
Easiest Way to Create Lambda Layers with the Required Python Version

Amit Duwal · Follow
Published in AWS in Plain English · 6 min read · 3 days ago

A tutorial on using Amazon CloudShell to create lambda layers zip file.

As a developer, I've always been excited about serverless computing and the potential it offers for scaling applications effortlessly. AWS Lambda, with its pay-as-you-go model, seemed like the perfect choice for my latest project. However, my enthusiasm was short-lived as I encountered a frustrating roadblock: adding external Python libraries to my Lambda functions.

I began my journey with AWS Lambda by creating a simple function to fetch data from an API and process it. The built-in AWS Lambda environment offered a seamless way to create and deploy my code. But as my project grew, so did the need for additional libraries to enhance its functionality.



Lambda layers

The first library I needed was **Numpy**, a popular Python library for processing datasets. Simple enough, I thought. I added it to my project's requirements.txt file, zipped up my code, and uploaded it to Lambda. It should have been a walk in the park, right? Wrong.

The first hurdle was Lambda's execution environment. Lambda runs on **Amazon Linux**, and any external libraries I added had to be compatible with this environment. After several failed attempts and cryptic error messages, I realized that the compiled versions of the libraries I was trying to use weren't compatible with Lambda. To resolve this, I had to rebuild these libraries in an Amazon Linux environment.

There were two options: use an EC2 instance or use Cloudshell

First, I attempted to solve the library compatibility issue using an **AWS EC2 instance**. While this approach allowed me to have complete control over the environment and manually install and compile the necessary libraries, it also introduced a level of complexity that I hadn't initially anticipated. Setting up the EC2 instance, configuring it, and ensuring it mirrored Lambda's execution environment precisely required significant time and effort. Additionally, I had to manage the EC2 instance's lifecycle and associated costs.

On the other hand, **AWS CloudShell** provided a more straightforward and user-friendly experience. CloudShell is a managed development environment within the AWS Console that comes preconfigured with many AWS CLI tools and SDKs, including those needed for Lambda development. It allowed me to create a Lambda-compatible environment quickly without the overhead of setting up an EC2 instance. Plus, CloudShell resources are automatically provisioned and billed on a per-second basis, making it cost-effective for occasional use.

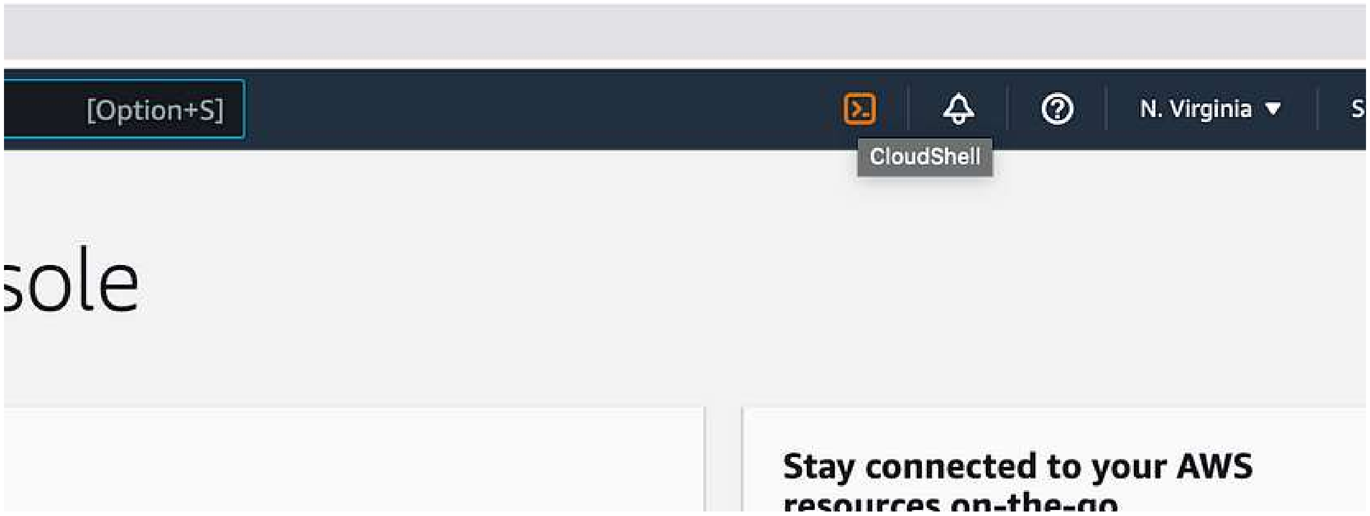
Using AWS CloudShell for creating a Lambda Layer ZIP file is advantageous for several reasons:

1. **Integrated Environment:** CloudShell comes pre-configured with AWS CLI tools, making it easy to work with AWS resources directly from the terminal without the need for additional setup.
2. **Ephemeral and Managed:** CloudShell is a fully managed service, so you don't have to worry about provisioning, managing, or maintaining any underlying infrastructure. It automatically cleans up after your session ends.
3. **Cost-Effective:** You only pay for the compute resources used during your CloudShell session, making it cost-effective for occasional tasks like creating Lambda Layers.
4. **Seamless Integration:** CloudShell integrates seamlessly with other AWS services, simplifying the process of uploading files to S3 and creating Lambda Layers.

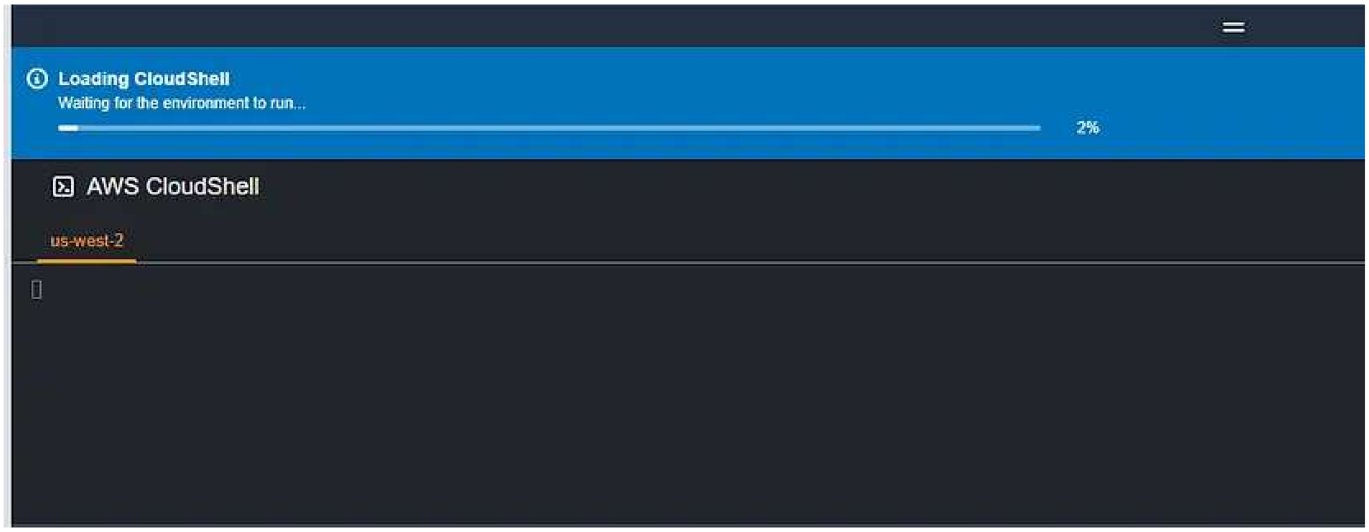
Below are the steps to create a Layer ZIP file in CloudShell:

1. **Access AWS CloudShell:**

- Log in to the AWS Management Console.
- Click on CloudShell icon left of the notification icon



A window like this will open:

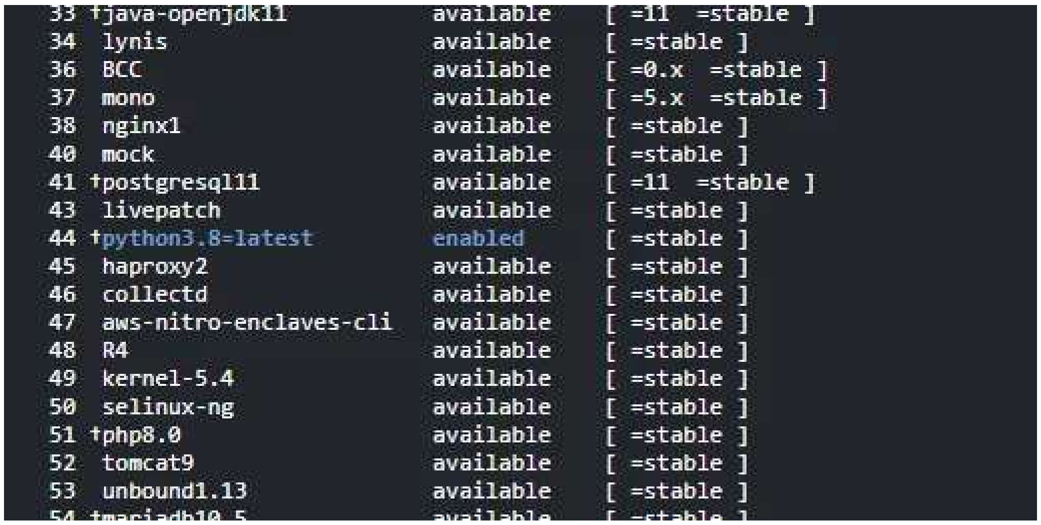

Loading CloudShell
Waiting for the environment to run...
2%
AWS CloudShell
us-west-2

2. Install the required Python version

- Wait for the the terminal to load.
- The default version of cloudshell is python 3.7.
- If you want to install python version 3.8 use the following command.

```
sudo amazon-linux-extras install python3.8
```

This will add python3.8 but python3.7 will still be the default python version.



```
 33 †java-openjdk11          available    [ =11  =stable ]
 34  lynis                   available    [ =stable ]
 36  BCC                     available    [ =0.x  =stable ]
 37  mono                    available    [ =5.x  =stable ]
 38  nginx1                  available    [ =stable ]
 40  mock                    available    [ =stable ]
 41 †postgresql11            available    [ =11  =stable ]
 43  livepatch               available    [ =stable ]
 44 †python3.8=latest        enabled      [ =stable ]
 45  haproxy2                available    [ =stable ]
 46  collectd                available    [ =stable ]
 47  aws-nitro-enclaves-cli  available    [ =stable ]
 48  R4                      available    [ =stable ]
 49  kernel-5.4              available    [ =stable ]
 50  selinux-ng              available    [ =stable ]
 51 †php8.0                  available    [ =stable ]
 52  tomcat9                 available    [ =stable ]
 53  unbound1.13             available    [ =stable ]
 54 †mariadb10.5             available    [ =stable ]
```

3. Prepare Your Layer Content:

- Create a directory named “packages”, then change into the directory by running the commands below one after the other

```
mkdir packages
```

```
cd packages
```

Next, create a virtual environment named venv and activate the virtual environment by running the commands below one after the other using the required python version.

```
python3.8 -m venv venv
source venv/bin/activate
```

Your virtual environment should be activated now. You can check the python version with code below.

```
python --version
```

Make sure you remain in the same directory. What you will do next is create a folder named "python" and change into the python directory by running the commands below one after the other.

```
mkdir python
cd python
```

Its important you create a directory named python. This is mandatory when working with lambda layers. Now that you are in python directory, install the required libraries.

```
pip install numpy -t .
```

We will need to zip the python folder where the library is but before we do that, let’s save space and delete objects with “.dis-info” extension from the folder. They are not needed. The best way to this is running the command below

```
rm -rf *dist-info
```

After running the above, we should be left with only the relevant packages needed.

Get out of the the python directory by going a directory backward using the command below

```
cd ..
```

Now, we will zip the python directory and give it a name called “numpy-lambda-package.zip”. Run the command below to achieve this.

```
zip -r numpy-lambda-package.zip python
```

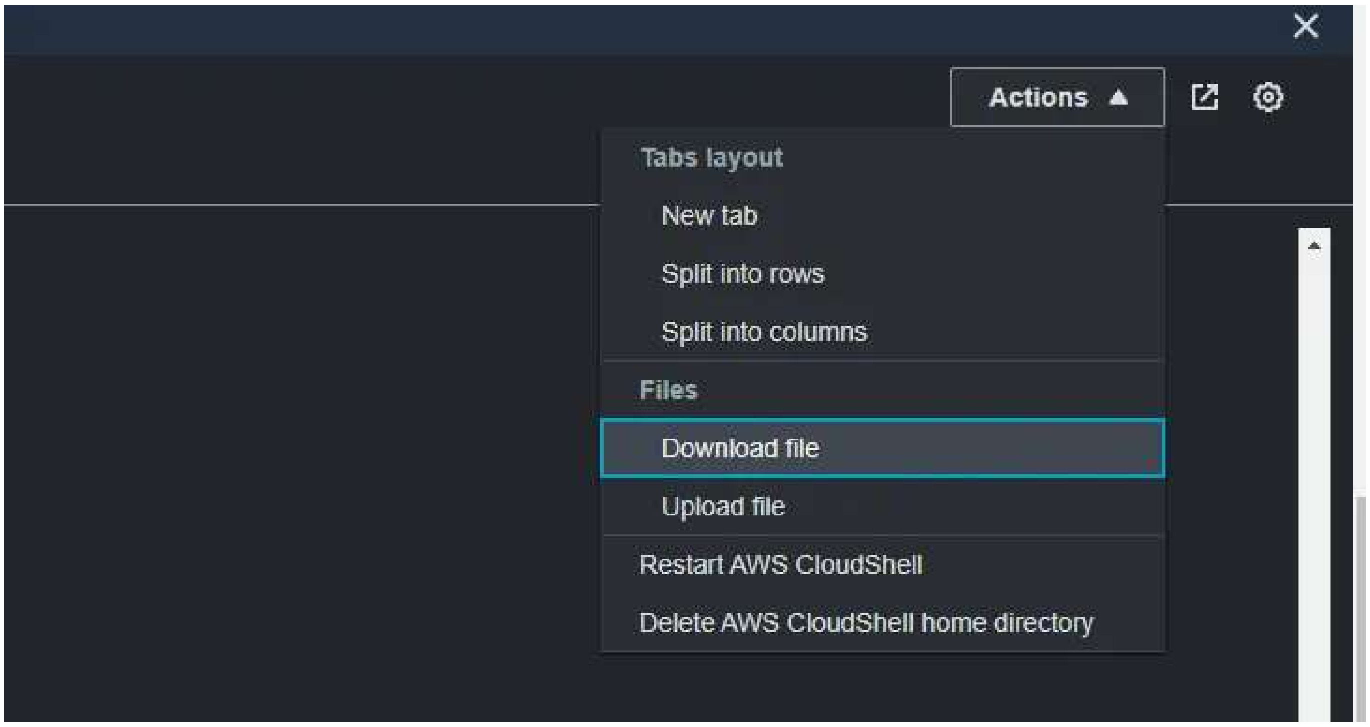
Once done, you should have a zip file named numpy-lambda-package.zip in the current directory.

3. Download Your Layer Content:

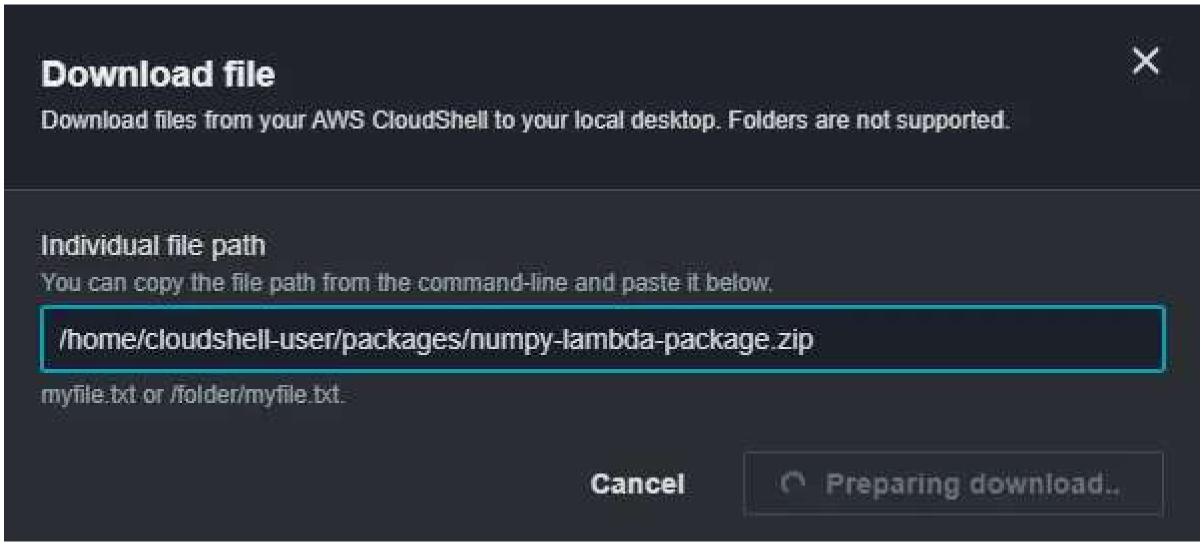
Next, upload the packaged zip into the s3 bucket you created earlier. The command looks like the below:

```
aws s3 cp numpy-lambda-package.zip s3://your-s3-bucket-name/
```

Or you can just download the required zip file using cloudshell to your local computer.



Enter file path:


Download file

Download files from your AWS CloudShell to your local desktop. Folders are not supported.

Individual file path

You can copy the file path from the command-line and paste it below.

```
/home/cloudshell-user/packages/numpy-lambda-package.zip
```

myfile.txt or /folder/myfile.txt.

This downloaded zip file can be uploaded to create a lambda layer manually from console.

Then you will have created a lambda layer successfully with minimal time and effort.

In the end, my journey with AWS Lambda and external Python libraries taught me that while serverless computing can be incredibly powerful, it also comes with its unique set of challenges. Small challenges may keep you occupied for a long time.

However, with persistence, patience, and a willingness to learn, these challenges can be overcome, and the rewards of a scalable, serverless architecture can be fully realized.

I hope this was helpful. Drop comments and feedback to help me improve and do more!

Don't forget to share with friends and connections.

Cheers!

Reference:

1. https://aws.amazon.com/blogs/compute/working-with-aws-lambda-and-lambda-layers-in-aws-sam/
2. https://www.linkedin.com/pulse/add-external-python-libraries-aws-lambda-using-layers-gabe-olokun

In Plain English

Thank you for being a part of our community! Before you go:

- *Be sure to* ***clap*** *and* ***follow*** *the writer!* 👏
- *You can find even more content at* ***PlainEnglish.io*** 🚀
- *Sign up for our* ***free weekly newsletter****.* 🗞️
- *Follow us on* ***Twitter****(X),* ***LinkedIn****,* ***YouTube****, and* ***Discord****.*

AWS Lambda Aws Lambda Layer AWS Aws Ec2 Amazon Web Services

Written by Amit Duwal

Follow

4 Followers · Writer for AWS in Plain English

AWS Enthusiast | Data Engineer

More from Amit Duwal and AWS in Plain English

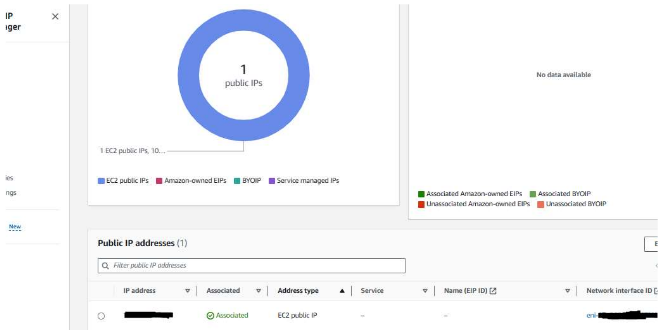
Amit Duwal

Amazon OpenSearch Service: A Review

A brief review of OpenSearch and Amazon OpenSearch Service

7 min read · Aug 23

20



Vivek V in AWS in Plain English

Stop paying $3.75 per month for a public IPv4 address on AWS...

6 min read · Aug 25

291 2