# Use AWS Runtimes to protect AWS Lambda functions

When the total size of all layers used allow it, it's preferable to configure the Application Security custom runtimes, since no code changes are required to your AWS lambda functions.

Given the limit of 250 MB for the total size of all uncompressed layers configured for a serverless function, it might not be possible to use the Application Security Custom Runtime, given its size.

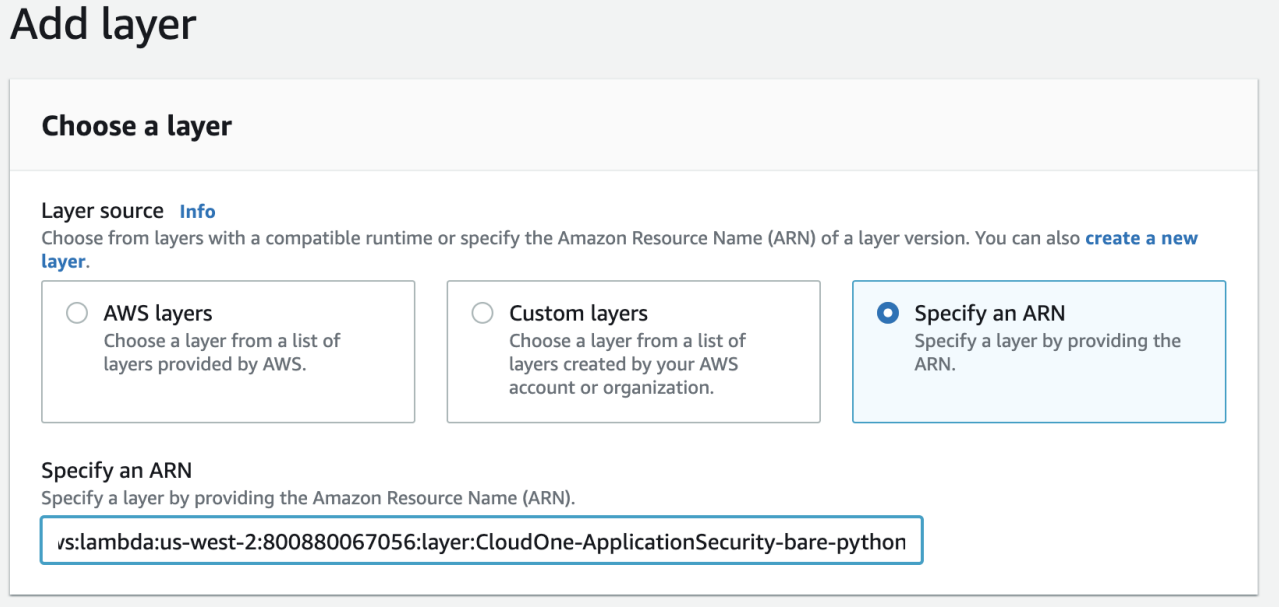
Application Security protection for AWS lambda functions is also available packaged as a layer, and to be used with official AWS runtimes. The Application Security layers can be used with official AWS runtimes, are much smaller in size, but require a few lines of code to be added to your lambda function.

This is supported only by Python and Node.js programming languages. {: .note }

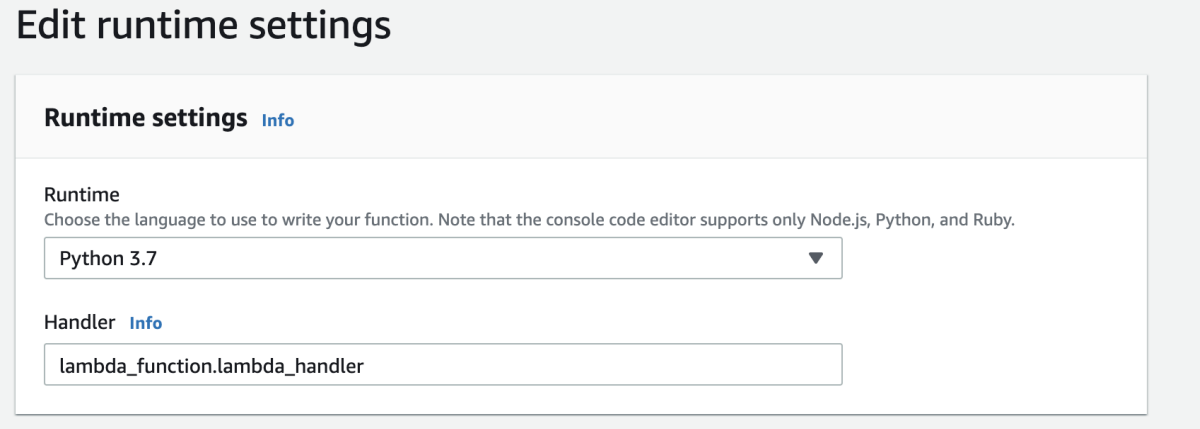
## Install Application Security using official AWS runtimes

If it's not possible to configure the Application Security Custom Runtime, you can use the Application Security layer with the official AWS runtimes by adding a few lines of code to the function to initialize manually.

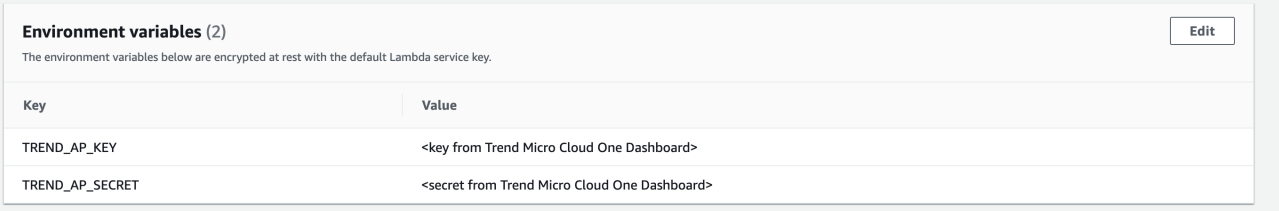
1. In your AWS account, go to **Lambda > Functions** and select the function you'd like to protect.
2. In the **Function Overview**, select **Layers** to be taken to the layers configuration view.
3. Select **Add layer**. On the **Choose a layer** view, select **Specify an ARN**. Specify the Application Security protection layer ARN in the text box **Specify an ARN**. Find the appropriate ARN in Python layers for AWS Lambda or NodeJS layers for AWS Lambda.




4. Do one of the following:
   - From the function overview, scroll to the **Runtime settings** panel, then select **Edit**, then select the required **Runtime** from the menu.
   - Navigate back to the Functions configuration. In the **Runtime settings** view, select **Edit**, and then select the required **Runtime** from the menu.




5. In order for the Application Security layer to protect your function, the Application Security group Key and Secret need to be configured as environment variables, for your function. In the **Environment variables** section, enter the values as seen in the below screenshot.




To find your Application Security group Key and Secret, go to the Application Security dashboard and navigate to your group's setting page. The Key and Secret are displayed under **Group Credentials**. {: .note }

## Amazon Resource Names (ARNs) {#arns}

The Application Security protection layer is available in multiple AWS regions.

The protection layer ARN depends on the lambda function programming language and the language version. The ARN structure for python functions is:

```
arn:aws:lambda:<aws region>:800880067056:layer:CloudOne-
ApplicationSecurity-bare-<language and version>:<layer version>
```

and the ARN structure for NodeJS functions is:

```
arn:aws:lambda:<aws region>:800880067056:layer:CloudOne-
ApplicationSecurity-runtime-<language and version>:<layer version>
```

- `aws region` = eu-north-1, ap-south-1, eu-west-1, eu-west-2, eu-west-3, ap-northeast-1, ap-northeast-2, ap-southeast-1, ap-southeast-2, sa-east-1, ca-central-1, eu-central-1, us-east-1, us-east-2, us-west-1, or us-west-2

### Python versions for AWS Lambda protection layer {#python-layers}

| language and version | layer version |
|---|---|

| language and version | layer version |
|---|---|
| python3_6 | 4 |
| python3_7 | 4 |
| python3_8 | 1 |

NodeJS versions for AWS Lambda protection layer {#nodejs-layers}

| language and version | layer version |
|---|---|
| nodejs10_x | 3 |
| nodejs12_x | 1 |

For example: If a lambda function is deployed in region us-east-1 and implemented with Python 3.7, the ARN is:

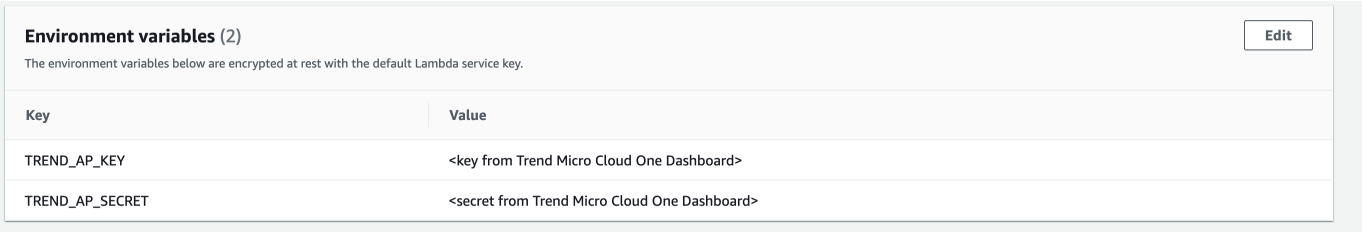
```
arn:aws:lambda:us-east-1:800880067056:layer:CloudOne-ApplicationSecurity-bare-python3_7:4
```

If a lambda function is deployed in region ap-south-1 and implemented with NodeJS 12, the ARN is:

```
arn:aws:lambda:ap-south-1:800880067056:layer:CloudOne-ApplicationSecurity-runtime-nodejs12_x:1
```

## Additional configuration for Lambda on AWS official runtimes

When using the Application Security protection layer, some additional configuration needs to be set as environment variables. Add the following in addition to the TREND_AP_KEY and TREND_AP_SECRET




```
TREND_AP_KEY: < key from Application Security Dashboard >
TREND_AP_SECRET: < secret from Application Security Dashboard >
TREND_AP_READY_TIMEOUT: 30
TREND_AP_TRANSACTION_FINISH_TIMEOUT: 10
TREND_AP_MIN_REPORT_SIZE: 1
TREND_AP_INITIAL_DELAY_MS: 1
TREND_AP_MAX_DELAY_MS: 100
TREND_AP_HTTP_TIMEOUT: 5
TREND_AP_PREFORK_MODE: False
```

```
TREND_AP_CACHE_DIR: /tmp/trend_cache
TREND_AP_LOG_FILE: STDERR
```

For more information about configuring environment variables, see Environment variables.

## Add code to the function to activate agents

When configuring Application Security as a layer rather than a runtime, a few changes need to be added to the code to activate security when the function executes.

### Initialize Library manually

In the **Function code**, add the following code to import Application Security and start security monitoring:

#### Python activation

```
python 3.7
import trend_app_protect.start
from trend_app_protect.api.aws_lambda import protect_handler

@protect_handler
def handler(event, context):

    # Your application code here

    return {
        'statusCode': 200,
        'body': 'Hello from Lambda',
    }
```

#### node.js activation

```
nodejs12.x
var trend_app_protect = require('trend_app_protect');

var _handler = async (event) => {

    // Your application code here

    return {
        statusCode: 200,
        body: 'Hello from Lambda',
    };
};

// Export wrapped handler
exports.handler =
trend_app_protect.api.aws_lambda.protectHandler(_handler);
```