

AWS Lambda

目录

- 一、使用控制台创建和调用 Lambda 函数
 - (一) 创建Lambda函数
 - (二) 调用Lambda函数
- 二、使用容器镜像部署 Python Lambda 函数

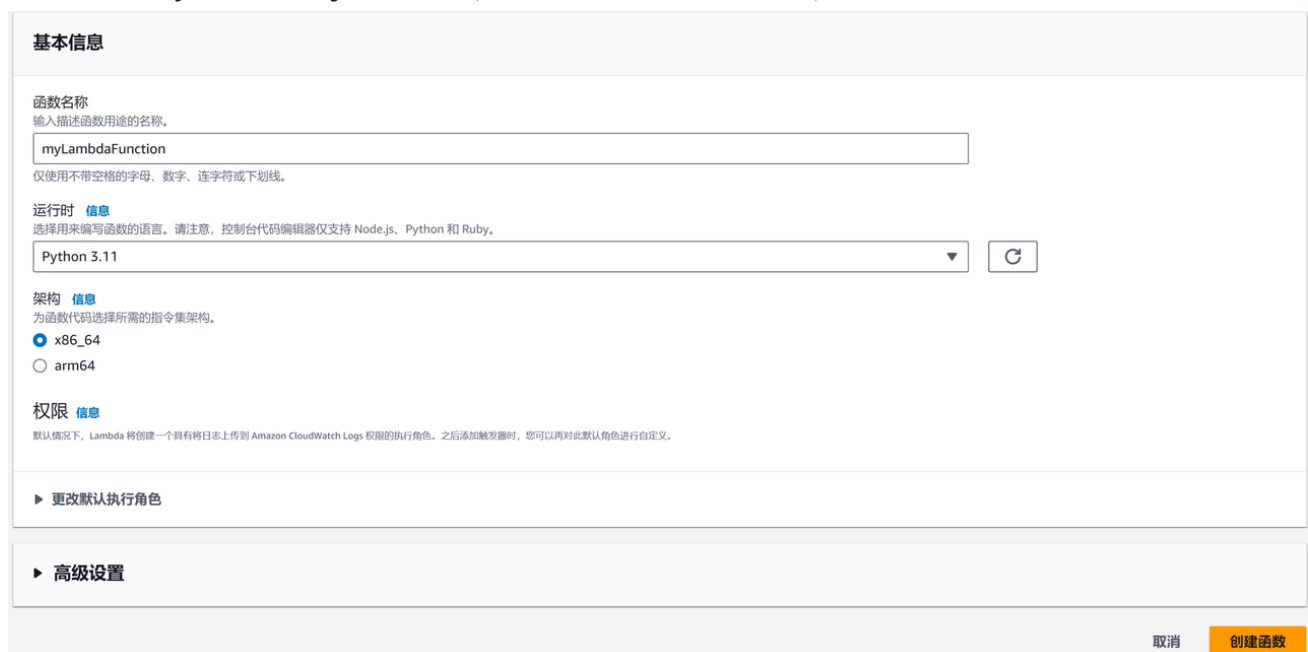
一、使用控制台创建和调用Lambda函数

(一) 创建Lambda函数

1. 打开Lambda控制台的 [Functions page](#)，选择创建函数



2. 选择从头开始编写。在基本信息窗格中，函数名称输入自己的函数名称。对于运行时系统，选择 Node.js 18.x 或 Python 3.11，保留架构设置为 x86_64，然后选择创建函数。



- 以Python 3.11为例。选择节点选项卡。在控制台的内置代码编辑器中，会显示Lambda创建的函数代码。如果代码编辑器中没有显示 `lambda_function.py` 选项卡，请在文件资源管理器中选择 `lambda_function.py`。



- 修改`lambda_function.py`中代码，以下是一个示例：

```
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    # Get the length and width parameters from the event object. The
    # runtime converts the event object to a Python dictionary
    length=event['length']
    width=event['width']

    area = calculate_area(length, width)
    print(f"The area is {area}")

    logger.info(f"CloudWatch logs group: {context.log_group_name}")

    # return the calculated area as a JSON string
    data = {"area": area}
    return json.dumps(data)

def calculate_area(length, width):
    return length*width
```

然后选择部署更新函数代码。当Lambda更改后，控制台会显示一个横幅，告知函数已成功更新。

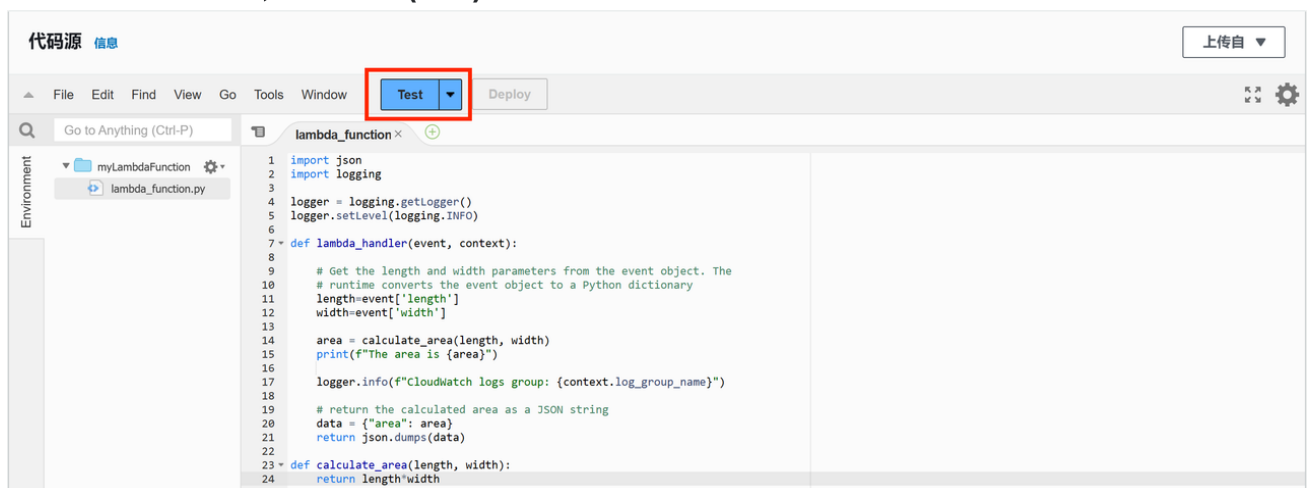
🟢 已成功更新函数 myLambdaFunction。

Lambda代码介绍：

- Lambda处理程序：Lambda函数需包含一个名为 `lambda_handler` 的Python函数。在Python中，Lambda函数可以包含多个Python函数，但处理程序函数始终是代码的入口点。当调用函数时，Lambda会运行此方法。
- Lambda事件对象：函数 `lambda_handler` 采用 `event` 和 `context` 这两个参数。Lambda中的事件是JSON格式的文档，其中包含要处理的函数数据。如果函数被其他AWS服务调用，则事件对象会包含有关导致调用的事件的信息。
- Lambda上下文对象：函数采用的第二个参数是 `context`。Lambda会自动将上下文对象传递给函数。上下文对象包含有关函数调用和执行环境的信息。

(二) 调用Lambda函数

1. 在代码源窗格中，选择测试(Test)。



The screenshot shows the AWS Lambda console's code editor interface. At the top, there is a 'Test' button highlighted with a red box. Below it, the code editor displays the following Python code:

```
1 import json
2 import logging
3
4 logger = logging.getLogger()
5 logger.setLevel(logging.INFO)
6
7 def lambda_handler(event, context):
8
9     # Get the length and width parameters from the event object. The
10    # runtime converts the event object to a Python dictionary
11    length=event['length']
12    width=event['width']
13
14    area = calculate_area(length, width)
15    print(f"The area is {area}")
16
17    logger.info(f"CloudWatch logs group: {context.log_group_name}")
18
19    # return the calculated area as a JSON string
20    data = {"area": area}
21    return json.dumps(data)
22
23 def calculate_area(length, width):
24    return length*width
```

2. 选择创建新事件。在事件名称中输入对应函数的事件名称。在事件 JSON 面板中，将需要的参数传给 `event` 来调用Lambda函数，然后选择选择 保存(Save)。以下是一个示例：

配置测试事件 ✕

测试事件是一个 JSON 对象，它模拟 AWS 服务为调用 Lambda 函数而发出的请求结构。使用它来查看函数的调用结果。

要调用函数而不保存事件，请配置 JSON 事件，然后选择“测试”。

测试事件操作

创建新事件 编辑已保存的事件

事件名称

myTestEvent

最多 25 个字符，可包含字母、数字、点、连字符和下划线。

事件共享设置

私有
此事件仅在 Lambda 控制台中可用以及仅事件创建者可用。您总共可以配置 10 个。[了解详情](#)

可共享
此事件适用于同一账户中有权访问和使用可共享事件的 IAM 用户。[了解详情](#)

模板 - 可选

hello-world

事件 JSON JSON 格式

```

1 {
2   "length": 6,
3   "width": 7
4 }
```

取消 调用 保存

- 测试函数并在控制台中查看调用记录。在代码源窗格中，选择测试。函数完成运行后，执行结果选项卡中将显示响应和函数日志。

代码源 信息 上传自 ▾

File Edit Find View Go Tools Window Test ▾ Deploy

Go to Anything (Ctrl-P)

Environment

- myLambdaFunction
 - lambda_function.py

Execution results Status: Succeeded | Max memory used: 40 MB | Time: 1.60 ms

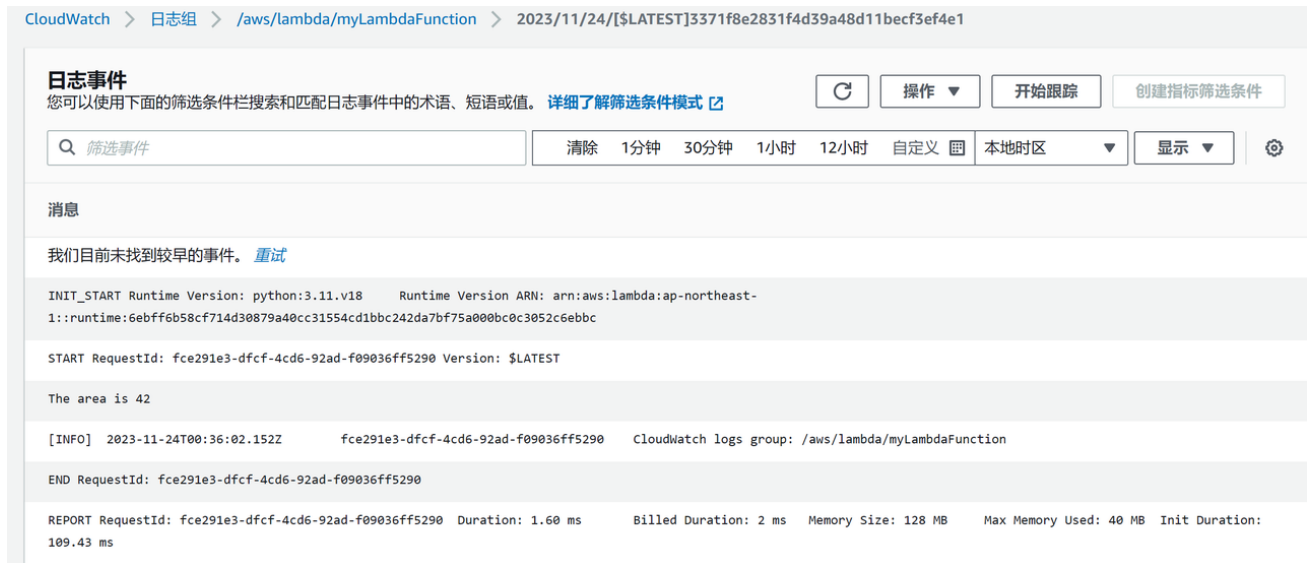
Test Event Name
myTestEvent

Response
{"area": 42}

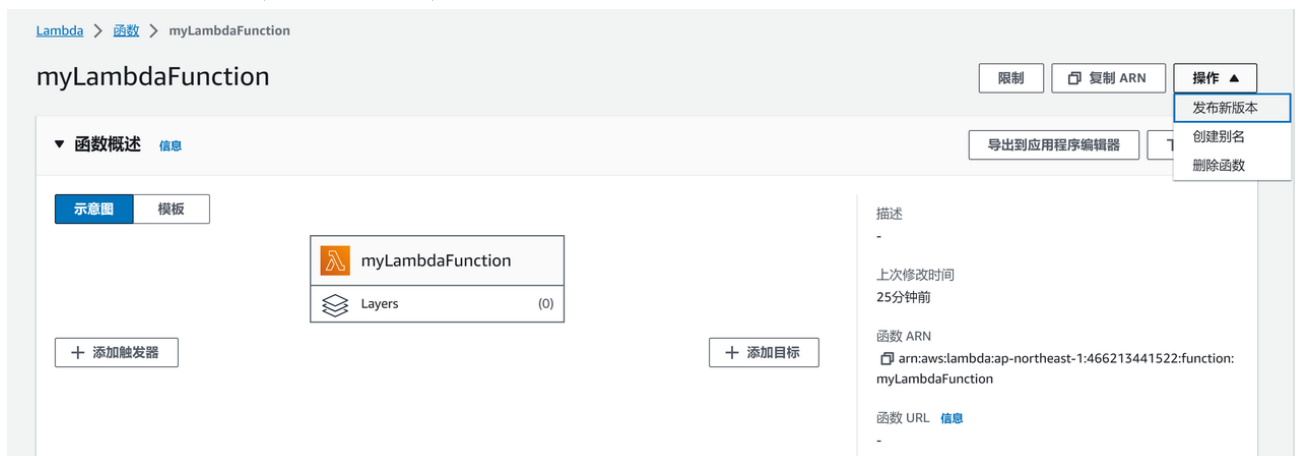
Function Logs
START RequestId: fce291e3-dfcf-4cd6-92ad-f09036ff5290 Version: \$LATEST
The area is 42
[INFO] 2023-11-24T00:36:02.152Z fce291e3-dfcf-4cd6-92ad-f09036ff5290 CloudWatch logs group: /aws/lambda/myLambdaFunction
END RequestId: fce291e3-dfcf-4cd6-92ad-f09036ff5290
REPORT RequestId: fce291e3-dfcf-4cd6-92ad-f09036ff5290 Duration: 1.60 ms Billed Duration: 2 ms Memory Size: 128 MB Max Memory Used: 40 MB Init Duration: 109.

Request ID
fce291e3-dfcf-4cd6-92ad-f09036ff5290

4. 可以在CloudWatch Logs [Log groups page](#) 中，选择对应的函数查看函数的调用记录



5. 使用结束后，在控制台依次选择**Actions**（操作）和**Delete**（删除）。在**Delete function**（删除函数）对话框中，输入`delete`，然后选择**Delete**（删除）



二、使用容器镜像部署PythonLambda函数

使用Python的AWS基本映像，以Linux-X86操作系统为例，使用命令行创建

官方文档：[使用容器镜像部署 Python Lambda 函数 - AWS Lambda \(amazon.com\)](#)

1. 安装AWS CLI的最新版本

a. 下载安装文件

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
```

b. 解压缩安装程序

```
unzip awscliv2.zip
```

c. 运行安装程序

```
sudo ./aws/install -i /usr/local/aws-cli -b /usr/local/bin
```

d. 确认安装

```
aws --version
```

```
aws-cli/2.13.38 Python/3.11.6 Linux/5.15.0-86-generic exe/x86_64.ubuntu.22 prompt/off
```

2. 安装docker

```
curl -fsSL https://get.docker.com | sudo sh -s
```

3. 从基本映像创建映像

a. 创建一个目录，创建名为 `lambda_function.py` 的新文件。代码示例：

```
import sys
def handler(event, context):
    return 'Hello from AWS Lambda using Python' + sys.version + '!'
```

b. 创建名为 `requirements.txt` 的新文件，列出每个必需的库，如boto3等。上述示例代码无需依赖库，留空即可

c. 创建Dockerfile，将 `FROM` 属性设置为基本映像的URI。然后，使用COPY命令将函数代码和运行时系统依赖项复制到 `{LAMBDA_TASK_ROOT}`，将 `CMD` 参数设置为Lambda函数处理程序。示例：

```
FROM public.ecr.aws/lambda/python:3.11

# Copy requirements.txt
COPY requirements.txt ${LAMBDA_TASK_ROOT}

# Install the specified packages
RUN pip install -r requirements.txt

# Copy function code
COPY lambda_function.py ${LAMBDA_TASK_ROOT}

# Set the CMD to your handler (could also be done as a parameter override
outside of the Dockerfile)
CMD [ "lambda_function.handler" ]
```

d. 构建Docker映像

```
docker build --platform linux/amd64 -t docker-image:test .
```

4. 部署映像

a. 配置aws configure

运行 `aws configure` 命令

配置相应的ID、密钥、默认地区以及默认输出格式等信息

```
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

如果没有密钥，则在 [用户 | IAM | Global \(amazon.com\)](#) 页面进行用户创建

IAM > 用户 > 创建用户

步骤 1
指定用户详细信息

指定用户详细信息

用户详细信息

用户名
test
用户名最多可包含 64 个字符。有效字符包括: A-Z、a-z、0-9 和 +, -, @, _ (连字符)

向用户提供 AWS 管理控制台的访问权限 - 可选
如果您向个人提供控制台访问权限, 则 [最佳实践](#) 在 IAM Identity Center 中管理其访问权限。

① 如果您通过访问密钥或者 AWS CodeCommit 或 Amazon Keyspaces 的服务特定凭证创建编程访问权限, 则可以在创建此 IAM 用户后生成这些访问权限。
[了解更多](#)

取消 **下一步**

创建完成后在用户页面安全凭证中点击**创建访问密钥**

权限 | 组 | 标签 | **安全凭证** | 访问顾问

控制台登录

[启用控制台访问权限](#)

控制台登录链接
<https://466213441522.signin.aws.amazon.com/console>

控制台密码
未启用

多重身份验证(MFA) (0)

[删除](#) [重新同步](#) [分配 MFA 设备](#)

使用 MFA 提高您的 AWS 环境的安全性。使用 MFA 登录需要来自 MFA 设备的身份验证码。每位用户最多可分配 8 台 MFA 设备。 [了解更多](#)

设备类型	标识符	认证	创建于
没有 MFA 设备。分配 MFA 设备以提高 AWS 环境的安全性			

[分配 MFA 设备](#)

访问密钥 (0)

[创建访问密钥](#)

使用访问密钥从 AWS CLI、AWS Tools for PowerShell、AWS 软件开发工具包以编程方式调用 AWS, 或者直接进行 AWS API 调用。您一次最多可拥有两个访问密钥(活跃或非活跃)。 [了解更多](#)

没有访问密钥。最佳实践是避免使用长期凭证, 例如访问密钥。请使用提供短期凭证的工具代替。 [了解更多](#)

[创建访问密钥](#)

创建成功后**访问密钥**和**秘密访问密钥**即为 `AWS Access Key ID` 和 `AWS Secret Access Key` 注意将密钥保存好

IAM > 用户 > test > 创建访问密钥

步骤 1
访问密钥最佳实践和替代方案

步骤 2 - 可选
设置描述标签

步骤 3
检索访问密钥

检索访问密钥 信息

访问密钥
如果您丢失或遗忘了秘密访问密钥，将无法找回它。您只能创建一个新的访问密钥并使旧密钥处于非活跃状态。

访问密钥	秘密访问密钥
<input type="text" value="AKIAWZDD4R7ZMRU2MSNE"/>	<input type="text" value="BwXzDKBxYRw7FQn5QyK2M9gzhQfPbjzIGqq6Dxrz"/> <small>隐藏</small>

访问密钥的最佳实践

- 切勿以纯文本、代码存储库或代码形式存储访问密钥。
- 不再需要时请禁用或删除访问密钥。
- 启用最低权限。
- 定期轮换访问密钥。

有关管理访问密钥的更多详细信息，请参阅[管理 AWS 访问密钥的最佳实践](#)。

[下载 .csv 文件](#) [已完成](#)

- b. 运行 `get-login-password` 命令，进行 Docker CLI 身份验证。将 `--region` 值设置为要在其中创建 Amazon ECR 存储库的 AWS 区域。将 `111122223333` 替换为 AWS 账户 ID。

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

- c. 使用 `create-repository` 命令在 Amazon ECR 中创建存储库。

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

创建成功会有如下输出：

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
  }
}
```

```
"createdAt": "2023-03-09T10:39:01+00:00",
"imageTagMutability": "MUTABLE",
"imageScanningConfiguration": {
  "scanOnPush": true
},
"encryptionConfiguration": {
  "encryptionType": "AES256"
}
}
}
```

从中复制 `repositoryUri`

d. 运行 `docker tag` 命令，将本地映像作为最新版本标记到Amazon ECR存储库中。

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

在此命令中：将 `docker-image:test` 替换为Docker映像的名称和标签。将 `<ECRrepositoryUri>` 替换为复制的 `repositoryUri`。确保URI末尾包含 `:latest`。示例：

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

e. 运行 `docker push`，将本地映像部署到Amazon ECR存储库。确保存储库URI末尾包含 `:latest`

f. 创建Lambda函数。对于 `ImageUri`，指定之前的存储库URI。确保URI末尾包含 `:latest`。

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

5. 调用Lambda函数

```
aws lambda invoke --function-name hello-world response.json
```

上述示例中应出现如下响应：

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

至此使用容器镜像部署Lambda完成