SERVERLESS WITH AWS USING PYTHON

PYCON INDIA Chennai 2019

WHAT IS SERVERLESS?

- No servers to provision, scale or manage
- Flexible scaling
- Build-in availability and fault tolerance

AWS LAMBDA

- Run code without thinking about servers
- Pay for only the compute time you consume
- Lambda takes care of everything required to run and scale your code with High Availability

AWS LAMBDA FUNCTION

```python
def handler(event, context):
    return event['foo'].upper()
```

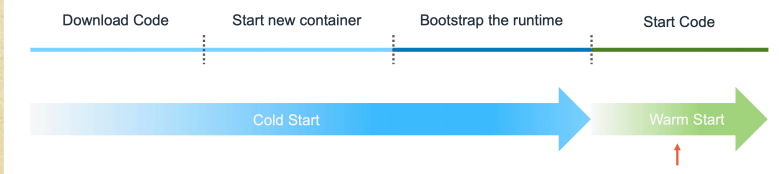
app.py

1. Define python function
2. Event - data about what triggered invocation
   Context - runtime information about your function
3. Return value sent back as response

```
$ python
>>> import app
>>> app.handler(event={"foo":"bar"}, context=None)
'BAR'
```

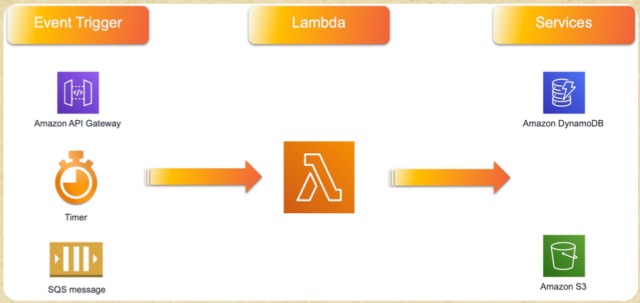
```
1 $ zip -r app.zip app.py

2 $ aws iam create-role \
    --role-name MyApp \
    --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}'

3 $ aws lambda create-function \
    --function-name hello \
    --handler app.handler \
    --runtime python3.6 \
    --role arn:aws:iam::635100884080:role/MyApp \
    --zip "fileb://./app.zip"
```

```
$ aws lambda invoke --function-name hello --payload '{"foo": "bar"}' output
```

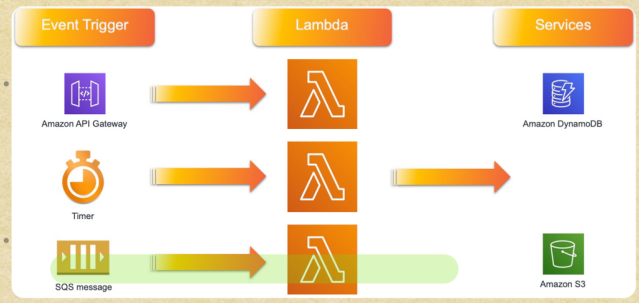



```
$ aws lambda invoke --function-name hello --payload '{"foo": "baz"}' output
```

SERVERLESS ARCHITECTURE

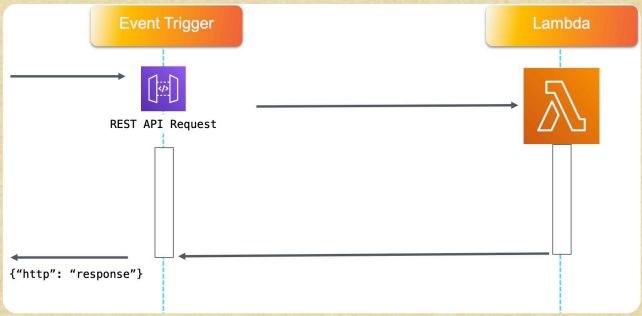


THINK IN TERMS OF EVENTS



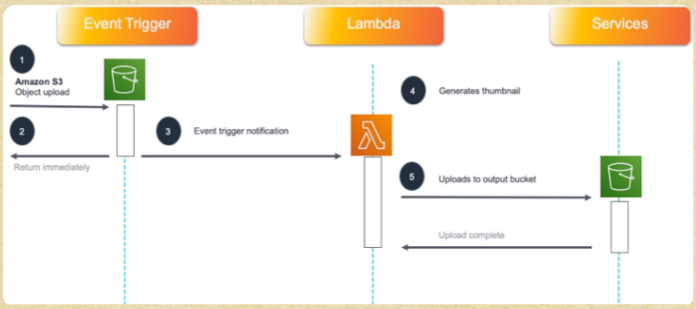
DECOMPOSE CODES INTO SEPARATE FUNCTIONS

SYNCHRONOUS INVOCATION



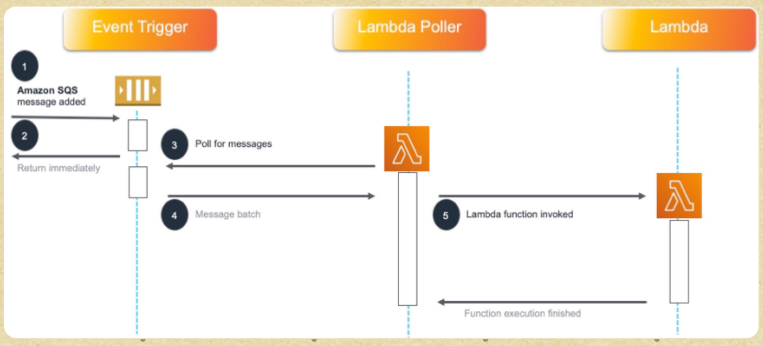
BLOCKS UNTIL RESPONSE IS RETURNED FROM LAMBDA FUNCTION

ASYNC INVOCATION



LAMBDA FUNCTION TRIGGERED IN RESPONSE TO AN EVENT WHICH IS DECOUPLED FROM AN ACTUAL EVENT TRIGGER

STREAM/POLL INVOCATION

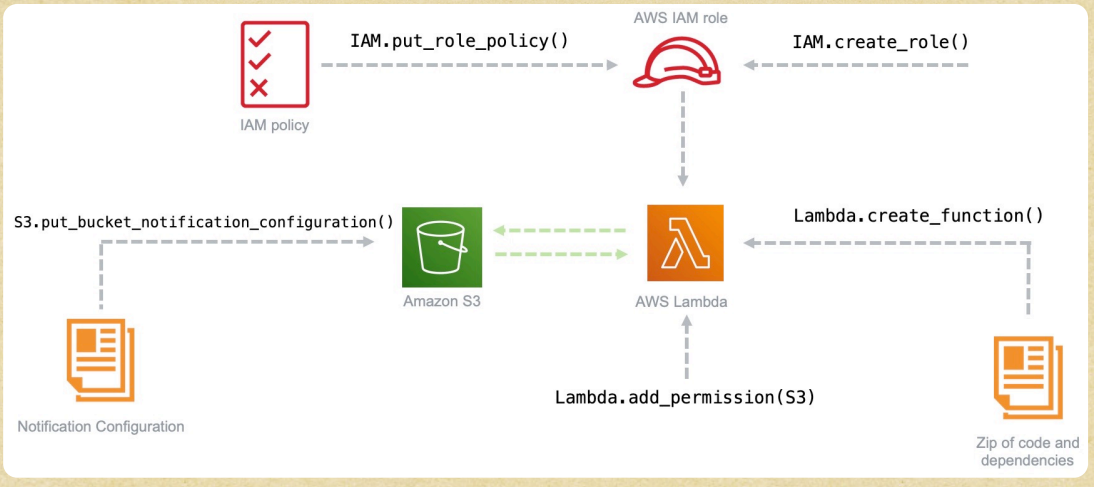


LAMBDA SERVICE ROLLS FOR CHANGES ON A STREAM, INVOKES LAMBDA FUNCTION WITH BATCH Of MESSAGES

IMAGE THUMBNAIL GENERATOR

IAM Policies

S3 Notification configuration



Third party packages

AWS CHALICE

PYTHON SERVERLESS MICROFRAMEWORK

CREATE & DEPLOY SERVERLESS APPLICATION

INTEGRATED WITH MANY OTHER AWS SERVICES

without CHALICE

Lambda

handler.py

AWS Lambda

```
def handler(event, context):
    return 'hello world!'
```

with CHALICE

Lambda

chalice

app.py

AWS Lambda

```
from chalice import Chalice

app = Chalice(app_name='helloworld')

@app.lambda_function()
def hello_world(event, context):
    return 'hello world!'
```

```
$ chalice deploy

Creating deployment package.
Updating policy for IAM role: helloworld-dev
Creating lambda function: helloworld-dev-hello_world
Deleting Rest API: 56mbn4ke87
Deleting function: arn:aws:lambda:us-west-2:635100884080:function:helloworld-dev
Resources deployed:
  - Lambda ARN: arn:aws:lambda:us-west-2:635100884080:function:helloworld-dev-hello_world
```

```
$ chalice invoke -n hello_world

"hello world!"
```

app.py

```
from chalice import Chalice

app = Chalice(app_name='helloworld')

@app.lambda_function()
def hello_world(event, context):
    return 'hello world!'
```

IMAGE THUMBNAIL GENERATOR

```
import tempfile
import boto3
from PIL import Image
from chalice import Chalice

app = Chalice(app_name='chalice_image_thumbnails', debug=True)
s3_client = boto3.client('s3')

@app.on_s3_event(bucket='mydemoimgbucket',
                 suffix='.jpg')
def resize_image(event):
    app.log.debug("Resizing the image from s3://%s/%s",
                  event.bucket, event.key)
    with tempfile.NamedTemporaryFile('w') as f:
        s3_client.download_file(event.bucket, event.key, f.name)
        resized_file = f.name + '.thumbnail.jpg'
        with Image.open(f.name) as image:
            image.thumbnail((256,256))
            image.save(resized_file)
        s3_client.upload_file(
            Filename=resized_file,
            Bucket='mydemothumbnailbucket',
            Key=resized_file.rsplit("/", 1)[-1]
        )
```

other CHALICE decorators

```
@app.on_s3_event('mybucket')
def resize_image(event, context):
    pass

@app.schedule('rate(10 minutes)')
def rate_handler(event):
    pass

@app.on_sns_message(topic='mytopic')
def handler(event):
    pass

@app.on_sqs_message(queue='myqueue')
def handler(event):
    pass
```

```
@app.lambda_function()
def handler(event, context):
    pass

@app.route('/resource/{value}', methods=['PUT'])
def resource(value):
    pass

@app.authorizer(ttl_seconds=60)
def auth(auth_request):
    pass
```

/_sumand /suman-d