Chaos Injection Library for AWS Lambda

Release 0.3

Adrian Hornsby

Aug 08, 2023

CONTENTS

`chaos_lambda` is a small library injecting chaos into AWS Lambda. It offers simple python decorators to do *delay*, *exception* and *statusCode* injection for your AWS Lambda function. This allows to conduct small chaos engineering experiments for your serverless application in the AWS Cloud.

- Support for Latency injection using `fault_type: latency`
- Support for Exception injection using `fault_type: exception`
- Support for HTTP Error status code injection using `fault_type: status_code`
- Using for SSM Parameter Store to control the experiment using `is_enabled: true`
- Support for adding rate of failure using `rate`. (Default rate = 1)
- Per Lambda function injection control using Environment variable (`CHAOS_PARAM`)

CHAPTER

ONE

INSTALL

```
pip install chaos-lambda
```

CHAPTER

TWO

EXAMPLE

```
# function.py

import os
from chaos_lambda import inject_fault

# this should be set as a Lambda environment variable
os.environ['CHAOS_PARAM'] = 'chaoslambda.config'

@inject_fault
def handler(event, context):
    return {
        'statusCode': 200,
        'body': 'Hello from Lambda!'
    }
```

Considering a configuration as follows:

```
{
    "fault_type": "exception",
    "delay": 400,
    "is_enabled": true,
    "error_code": 404,
    "exception_msg": "This is chaos",
    "rate": 1
}
```

When excecuted, the Lambda function, e.g `handler('foo', 'bar')`, will produce the following result:

```
exception_msg from config chaos with a rate of 1
corrupting now
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "/.../chaos_lambda.py", line 199, in wrapper
    raise Exception(exception_msg)
Exception: This is chaos
```

CHAPTER

THREE

CONFIGURATION

The configuration for the failure injection is stored in the AWS SSM Parameter Store and accessed at runtime by the `get_config()` function:

```
{
    "fault_type": "exception",
    "delay": 400,
    "is_enabled": true,
    "error_code": 404,
    "exception_msg": "This is chaos",
    "rate": 1
}
```

To store the above configuration into SSM using the AWS CLI do the following:

```
aws ssm put-parameter --name chaoslambda.config --type String --overwrite --value "{
↪"delay": 400, "is_enabled": true, "error_code": 404, "exception_msg": "This is chaos
↪", "rate": 1, "fault_type": "exception"}" --region eu-west-1
```

AWS Lambda will need to have IAM access to SSM.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:DescribeParameters"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters",
                "ssm:GetParameter"
            ],
            "Resource": "arn:aws:ssm:eu-north-1:12345678910:parameter/chaoslambda.
↪config"
        }
    ]
}
```

CHAPTER

FOUR

SUPPORTED FAULTS:

`chaos_lambda` currently supports the following faults:

- *latency* - Add latency in the AWS Lambda execution
- *exception* - Raise an exception during the AWS Lambda execution
- *status_code* - force AWS Lambda to return a specific HTTP error code

CHAPTER

FIVE

MORE INFORMATION:

chaos_lambda.**get_config**()

Retrieve the full configuration from the SSM parameter store The config returns a dictionary value: requested configuration

How to use:

```
>>> import os
>>> from chaos_lambda import get_config
>>> os.environ['CHAOS_PARAM'] = 'chaoslambda.config'
>>> get_config()
{'delay': 500, 'is_enabled': True, 'error_code': 404, 'exception_msg': 'chaos',
↪'rate': 1, 'fault_type': 'latency'}
```

chaos_lambda.**inject_fault**(*func*)

Add failure to the lambda function based on the value of 'fault_type' present in the config returned from the SSM parameter store

Given SSM Configuration:

```
{
    "fault_type": "latency",
    "delay": 400,
    "is_enabled": true,
    "error_code": 404,
    "exception_msg": "chaos",
    "rate": 1
}
```

Usage:

```
>>> @inject_fault
... def handler(event, context):
...     return {
...         'statusCode': 200,
...         'body': 'Hello from Lambda!'
...     }
>>> handler('foo', 'bar')
Injecting 400 of delay with a rate of 1
Added 402.20ms to handler
{'statusCode': 200, 'body': 'Hello from Lambda!'}
```

Given SSM Configuration:

```
{
    "fault_type": "exception",
    "delay": 400,
    "is_enabled": true,
    "error_code": 404,
    "exception_msg": "chaos",
    "rate": 1
}
```

Usage:

```
>>> @inject_fault
  ... def handler(event, context):
  ...     return {
  ...         'statusCode': 200,
  ...         'body': 'Hello from Lambda!'
  ...     }
  >>> handler('foo', 'bar')
  Injecting exception_type <class "Exception"> with message chaos a rate of 1
  corrupting now
  Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
      File "/.../chaos_lambda.py", line 316, in wrapper
          raise _exception_type(_exception_msg)
  Exception: chaos
```

Given SSM Configuration:

```
{
    "fault_type": "status_code",
    "delay": 400,
    "is_enabled": true,
    "error_code": 404,
    "exception_msg": "chaos",
    "rate": 1
}
```

Usage:

```
>>> @inject_fault
... def handler(event, context):
...     return {
...         'statusCode': 200,
...         'body': 'Hello from Lambda!'
...     }
>>> handler('foo', 'bar')
Injecting Error 404 at a rate of 1
corrupting now
{'statusCode': 404, 'body': 'Hello from Lambda!'}
```

INDEX