Preparação do Ambiente para Comparação de Desempenho entre Arquiteturas *Serverless* e *Server-based* no Contexto Educacional

Matheus Lucca Alves

FACOM - Universidade Federal do Mato Grosso do Sul (UFMS)

SUMÁRIO

1 - Códigos

1.1 Código 1: Criptografia e Descompressão de Texto no Ambiente *Server-based*

```
import zlib
import awsgi
from flask import (
    Flask,
    request,
)
from cryptography.fernet import Fernet
import time
import random

app = Flask(__name__)
app.config['TIMEOUT'] = 600
app.config['PROPAGATE_EXCEPTIONS'] = True
app.config['PRESERVE_CONTEXT_ON_EXCEPTION'] = True



chaves = [Fernet.generate_key() for _ in range(10)]
ciphers = [Fernet(chave) for chave in chaves]

@app.route('/criptografar_descriptografar', methods=['GET'])
def criptografar_descriptografar_texto():
    texto = request.args.get('texto')

    if not texto:
        return 'Por favor, forneça um texto.', 400

    try:
        # Criptografar múltiplas vezes para estressar a CPU
        texto_criptografado = texto.encode('utf-8')
        for _ in range(22):
                for i in range(10):
                texto_comprimido = zlib.compress(texto_criptografado)
                texto_criptografado = ciphers[i].encrypt(texto_comprimido)

        texto_criptografado_final = texto_criptografado

        for _ in range(22):  # Aumentar o número de iterações
            for i in range(9, -1, -1):
                texto_descriptografado = ciphers[i].decrypt(texto_criptografado)
                texto_criptografado = zlib.decompress(texto_descriptografado)

        texto_descomprimido_final = texto_criptografado.decode('utf-8')

        return {
            'texto_original': texto,
            'texto_criptografado': texto_criptografado_final.decode('utf-8'),
            'texto_descomprimido': texto_descomprimido_final,
        }, 200
    except Exception as e:
        return {'erro': 'Erro ao criptografar/descriptografar o texto: {}'.format(e)}, 500
```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

1.2 Código 2: Alocação de Memória e Cálculos para Estresse da CPU e Memória no Ambiente Server-based

```
import logging
from flask import Flask, request
import time
import math

app = Flask(__name__)
app.config['TIMEOUT'] = 600
app.config['PROPAGATE_EXCEPTIONS'] = True
app.config['PRESERVE_CONTEXT_ON_EXCEPTION'] = True

# Configuração de logging
logging.basicConfig(filename='/tmp/flask_app.log', level=logging.DEBUG)

@app.route('/stress_memory_and_cpu', methods=['GET'])
def stress_memory_and_cpu():
    try:
        size = int(request.args.get('size', '3000'))  # Tamanho da lista em MB
        duration = int(request.args.get('duration', '25'))  # Duração do estresse da CPU em segundos

        # Estressar memória
        large_list = ['x' * 1024 * 1024] * size  # Aloca uma lista de strings grandes

        # Estressar CPU
        end_time = time.time() + duration
        count = 0
        while time.time() < end_time:
            count += 1
            _ = math.sqrt(count)

        return {
            'status': 'success',
            'allocated_size_mb': size,
            'cpu_iterations': count
        }, 200
    except Exception as e:
        logging.error('Erro ao estressar memória e CPU: %s', e)
        return {'erro': 'Erro ao estressar memória e CPU: {}'.format(e)}, 500

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

1.3 Código 1: Criptografia e Descompressão de texto no Ambiente Serverless:

```
import logging
import zlib
from flask import Flask, request
from cryptography.fernet import Fernet

app = Flask(__name__)
app.config['TIMEOUT'] = 600
app.config['PROPAGATE_EXCEPTIONS'] = True
app.config['PRESERVE_CONTEXT_ON_EXCEPTION'] = True

logging.basicConfig(filename='/tmp/flask_app.log', level=logging.INFO)

chaves = [Fernet.generate_key() for _ in range(10)]
ciphers = [Fernet(chave) for chave in chaves]

def lambda_handler(event, context):
    texto = ''
    if 'queryStringParameters' in event:
        if 'texto' in event['queryStringParameters']:
            texto = event['queryStringParameters']['texto']

    if texto == '':
        return {
            "statusCode": 200,
            "body": "{'Test': 'Test'}",
            "headers": {
                'Content-Type': 'text/html',
            }
        }
    try:
        texto_criptografado = texto.encode('utf-8')
        for _ in range(22):
            for i in range(10):
                texto_comprimido = zlib.compress(texto_criptografado)
                texto_criptografado = ciphers[i].encrypt(texto_comprimido)


        texto_criptografado_final = texto_criptografado


        for _ in range(22):  # Aumentar o número de iterações
            for i in range(9, -1, -1):
                texto_descriptografado = ciphers[i].decrypt(texto_criptografado)
                texto_criptografado = zlib.decompress(texto_descriptografado)

        texto_descomprimido_final = texto_criptografado.decode('utf-8')

        return {
            'statusCode': 200,
            'headers': {'Content-Type': 'application/json'},
            'body': {
                'texto_original': texto,
                'texto_criptografado': texto_criptografado_final.decode('utf-8'),
                'texto_descomprimido': texto_descomprimido_final,
            }
```

```
    }
except Exception as e:

    logging.error('Erro ao criptografar/descriptografar o texto: %s', e)
    return {
        'statusCode': 500,
        'headers': {'Content-Type': 'application/json'},
        'body': {
            'erro': 'Erro ao criptografar/descriptografar o texto: {}'.format(e)
        }
    }
```

1.4 Código 2: **Alocação de Memória e Cálculos para Estresse da *CPU* e Memória no Ambiente Serverless:**

```
import json
import time
import math

def lambda_handler(event, context):
    try:
        # Obter parâmetros do evento
        size = int(event.get('queryStringParameters', {}).get('size', '3000'))
        duration = int(event.get('queryStringParameters', {}).get('duration', '25'))

        # Estressar memória
        large_list = ['x' * 1024 * 1024] * size  # Aloca uma lista de strings grandes

        # Estressar CPU
        end_time = time.time() + duration
        count = 0
        while time.time() < end_time:
            count += 1
            _ = math.sqrt(count)

        return {
            'statusCode': 200,
            'body': json.dumps({
                'status': 'success',
                'allocated_size_mb': size,
                'cpu_iterations': count
            }),
            'headers': {
                'Content-Type': 'application/json',
            }
        }
    except Exception as e:
        return {
            'statusCode': 500,
            'body': json.dumps({
                'erro': f'Erro ao estressar memória e CPU: {e}'
            }),
            'headers': {
                'Content-Type': 'application/json',
            }
        }
```

2 - Resultados

2.1 Código 1: Criptografia e Descompressão de texto;

Métrica	*EC2 t2.micro* (1 vCPU - 1 GiB)	*EC2 t2.medium* (2 vCPUs - 4 GiB	*Lambda* 128mb	*Lambda* 256mb
Latência média	56376 ms	4140 ms	7236 ms	3965 ms
Latência mínima	1164 ms	1009 ms	6045 ms	3242 ms
Latência máxima	113978 ms	33355 ms	10762 ms	5763 ms
Desvio padrão	25556.21ms	7697.58 ms	376.26 ms	261.89 ms
Throughput	2.5/sec	4.2/sec	4.1/sec	4.2/sec
Tempo execução	33,38 min	20,03 min	20,21 min	19,49 min

2.2 Código 2: Alocação de Memória e Cálculos para Estresse da CPU e Memória;

Métrica	*EC2 t2.micro* (1 vCPU - 1 GiB)	*EC2 t2.medium* (2 vCPUs - 4 GiB	*Lambda* 128mb	*Lambda* 256mb
Latência média	62833 ms	48351 ms	25527 ms	25515 ms
Latência mínima	25353 ms	25472 ms	25462 ms	25464 ms
Latência máxima	351454 ms	78543 ms	26856 ms	26861 ms
Desvio padrão	60408.67 ms	14248.41 ms	99.65 ms	81.68 ms
Throughput	1.3/sec	1.4/sec	1.7/sec	1.7/sec
Tempo execução	52,50 min	46,45 min	39,09 min	39,08 min