Engenharia de Desempenho (Performance Testing)

1. Introdução e Contexto

Nesta atividade, vocês receberão uma API de *Checkout de E-commerce* (simulada) já pronta. Seu trabalho não é alterar o código da API, mas sim atuar como um Engenheiro de Performance para criar uma suíte de testes que revele os limites dessa aplicação.

2. Objetivos de Aprendizagem

- Configurar e executar testes de performance utilizando k6.
- Diferenciar na prática Carga, Estresse e Resistência.
- Analisar métricas de Latência (p95, p99) vs. Throughput.
- Identificar o ponto de ruptura (Breaking Point) de uma aplicação.

3. O Projeto Base (SUT)

Vocês utilizarão o projeto ecommerce-checkout-api. Ele possui endpoints propositalmente desenhados com comportamentos distintos:

- `GET /health`: Resposta rápida.
- `POST /checkout/simple`: Simula uma operação leve de I/O (banco de dados).
- `POST /checkout/crypto`: Simula uma operação pesada de CPU (cálculo de hash).

4. Etapas do Trabalho

Link do Repositório: https://github.com/CleitonSilvaT/teste-de-desempenho

Etapa 1: Configuração e Smoke Test

1. Clone o repositório fornecido e inicie a API (`npm install && npm start`).
2. Instale o k6 na sua máquina.
3. Crie o script `tests/smoke.js`.
    - Objetivo: Verificar se a API está de pé antes de iniciar testes pesados.
    - Config: 1 usuário (VUser) por 30 segundos acessando `/health`.
    - Critério de Sucesso: 100% de sucesso nas requisições.

Etapa 2: Teste de Carga (Load Testing)

O Marketing anunciou uma promoção e esperamos um pico de 50 usuários simultâneos.

1. Crie o script `tests/load.js`.
2. Alvo: Endpoint `/checkout/simple`.
3. Cenário (Stages):
    - Ramp-up: 0 a 50 usuários em 1 minuto.
    - Platô: Manter 50 usuários por 2 minutos.

    - Ramp-down: 50 a 0 usuários em 30 segundos.
4. SLA (Thresholds): Defina que o p95 da latência deve ser menor que 500ms e erros abaixo de 1%.

Etapa 3: Teste de Estresse (Stress Testing)

Queremos saber: "Quantos usuários fazendo cálculos de criptografia (CPU Heavy) derrubam o servidor?"

1. Crie o script `tests/stress.js`.
2. Alvo: Endpoint `/checkout/crypto`.
3. Cenário: Aumente a carga agressivamente.
    - 0 a 200 usuários em 2 minutos.
    - 200 a 500 usuários em 2 minutos.
    - 500 a 1000 usuários em 2 minutos.
4. Análise: Observe no terminal o momento exato em que os tempos de resposta começam a subir exponencialmente ou ocorrem Timeouts.

Etapa 4: Teste de Pico (Spike Testing)

Simule um comportamento de "Flash Sale" (ex: abertura de venda de ingressos).

1. Crie o script `tests/spike.js`.
2. Alvo: Endpoint `/checkout/simple`.
3. Cenário:
    - Carga baixa (10 usuários) por 30s.
    - Salto imediato para 300 usuários em 10s.
    - Manter por 1 minuto.
    - Queda imediata para 10 usuários.

5. O Que Entregar

1. Link do seu repositório contendo a pasta `tests/` com os 4 scripts.
2. Relatório Técnico (PDF) (1 página):
    - Resumo Executivo: Qual é a capacidade máxima de usuários que a API suporta no cenário de CPU (`/crypto`) e no de I/O (`/simple`)?
    - Evidências: Prints dos resumos de execução do k6 para cada teste.
    - Análise de Estresse: Indique em que ponto (número de VUsers) a aplicação começou a falhar no teste de estresse.