Métricas customizadas

3 — Criando métricas customizadas

As **métricas customizadas** permitem medir **qualquer coisa** dentro do seu teste — tempos específicos, contadores, taxas de sucesso, valores de resposta, etc.

> Elas complementam as métricas padrão (http_req_duration, http_reqs, etc.) e te dão total controle sobre o que medir.

Tipos de métricas customizadas

Cada tipo de métrica serve para **analisar um aspecto diferente do comportamento da aplicação**.

Escolher o tipo certo garante que suas medições sejam úteis e representem o que realmente importa.

Tipo	Classe (no código)	Ideal para medir	Como funciona	Exemplo prático
Tendência	`new Trend('nome')`	Tempos, durações ou valores que variam a cada requisição	Guarda **todos os valores registrados** e calcula automaticamente estatísticas como **média**, **min**, **max**, **p90**, **p95**, etc.	Medir o tempo de resposta da rota `/produtos` : → `tempoRequisicao.add(res.timings.duration)`
Contador	`new Counter('nome')`	Quantidade total de eventos	Apenas **soma valores** (normalmente 1 a cada evento). Não mede tempo nem proporção.	Contar quantas requisições retornaram erro: → `totalErros.add(1)`
Taxa	`new Rate('nome')`	Proporção de sucesso/falha (entre 0 e 1)	Registra **1 para sucesso** e **0 para falha**, permitindo calcular a **porcentagem de acerto**.	Calcular taxa de respostas com status 200: → `taxaSucesso.add(res.status === 200)`
Medidor	`new Gauge('nome')`	Valor atual (não média nem total)	Guarda **apenas o último valor adicionado**, substituindo o anterior.	Armazenar o último tempo de resposta medido: → `ultimoTempo.add(res.timings.duration)`

Exemplo prático

```
import http from 'k6/http';
import { check } from 'k6';
import { Trend, Counter, Rate, Gauge } from 'k6/metrics';

// Criando métricas customizadas
const tempoRequisicao = new Trend('tempo_requisicao'); // média, p90, p95...
const totalErros = new Counter('total_erros');         // soma total de erros
const taxaSucesso = new Rate('taxa_sucesso');          // % de respostas com sucesso
const ultimoTempo = new Gauge('ultimo_tempo');         // guarda o último tempo medido

export const options = {
  vus: 5,
  iterations: 10,
};

export default function () {
  const res = http.get('https://test.k6.io/');

  // Adiciona o tempo da requisição em milissegundos
  tempoRequisicao.add(res.timings.duration);
  ultimoTempo.add(res.timings.duration);

  // Verifica se o status retornou 200
  const sucesso = check(res, { 'status é 200': (r) => r.status === 200 });

  // Registra nas métricas
  taxaSucesso.add(sucesso);
  if (!sucesso) totalErros.add(1);
}
```

Saída esperada no console

Após rodar:

```
k6 run script.js
```

Você verá algo como:

```
     checks_total.......: 10      743.638175/s
     checks_succeeded...: 100.00% 10 out of 10
     checks_failed......: 0.00%   0 out of 10

   ✓ status é 200

   CUSTOM
   taxa_sucesso.....................: 100.00% 10 out of 10
   tempo_requisicao.................: avg=3.41509 min=0.9519  med=3.32675 max=6.1433  p(90)=5.5
```

```
943  p(95)=5.8688
    ultimo_tempo.....: 0.9519  min=0.9519   max=6.1433

    HTTP
    http_req_duration.....: avg=3.41ms  min=951.9µs med=3.32ms  max=6.14ms  p(90)=5.59
ms  p(95)=5.86ms
      { expected_response:true }...: avg=3.41ms  min=951.9µs med=3.32ms  max=6.14ms  p(90)=
5.59ms  p(95)=5.86ms
    http_req_failed.....: 0.00%   0 out of 10
    http_reqs.....: 10      743.638175/s

    EXECUTION
    iteration_duration.....: avg=6.59ms  min=951.9µs med=6.5ms   max=12.49ms p(90)=11.94
ms p(95)=12.22ms
    iterations.....: 10      743.638175/s

    NETWORK
    data_received.....: 11 kB   797 kB/s
    data_sent.....: 780 B   58 kB/s
```

Entendendo o resultado

Métrica	O que mede	Como interpretar
`tempo_requisicao`	tempo médio das respostas	se o p95 subir muito → instabilidade
`total_erros`	número total de falhas	deve ser 0 (ideal)
`taxa_sucesso`	taxa de sucesso (1 = 100%)	abaixo de 0.95 → problema
`ultimo_tempo`	último tempo medido	útil em execuções contínuas (monitoramento)