Thresholds

4 — Thresholds (metas de performance)

Os **Thresholds** são **critérios de aprovação ou reprovação** dos testes de performance.

Eles funcionam como **metas automáticas** que o sistema deve cumprir — se o desempenho estiver abaixo do esperado, o teste **falha automaticamente**, mesmo que as requisições não gerem erros.

Em outras palavras, os *thresholds* garantem que o sistema continue **rápido, estável e confiável**, sem depender de análise manual dos relatórios.

Por que usar limites?

- **Padronização** entre versões.
- **Automatizam a validação** no pipeline CI/CD (ex: Jenkins, GitHub Actions).
- **Detectam regressões** de performance antes que cheguem ao usuário final.
- **Eliminam análises subjetivas**, pois tudo é avaliado por métricas objetivas.

Estrutura básica

```
export const options = {
  thresholds: {
    'nome_da_metrica': ['condição1', 'condição2', ...],
  },
};
```

Cada condição é uma expressão lógica que o K6 avalia ao final do teste.

Se qualquer uma for violada, o teste é marcado como **FAILED** .

Exemplo prático

```
export const options = {
  vus: 10,
  duration: '10s',
  thresholds: {
```

```
    http_req_duration: ['p(95)<500'], // 95% das requisições devem respond
er em menos de 500 ms
    http_req_failed: ['rate<0.01'],   // Menos de 1% das requisições podem fal
har
    checks: ['rate>0.98'],          // Pelo menos 98% das validações devem p
assar
  },
};
```

Se algum threshold for violado, o K6 mostrará um erro no final do relatório:

```
✓ http_req_duration.....: p(95)=420.8 ms < 500
× http_req_failed.....: rate=0.015 > 0.01
✓ checks.....: rate=0.99 > 0.98

ERRO: Threshold falhou — 1.5% das requisições tiveram erro
```

Isso permite **automatizar a análise de performance** e impedir o deploy de versões lentas ou instáveis.

4.1 — Thresholds em detalhes

A opção `thresholds` pode ser usada tanto com **métricas padrão** quanto com **métricas customizadas**.

O K6 oferece flexibilidade para definir regras baseadas em percentis, médias, máximos, contadores ou taxas.

Principais métricas nativas disponíveis

Métrica	Tipo	Descrição
`http_req_duration`	Tendência	Tempo total da requisição (DNS + conexão + envio + espera + resposta)
`http_req_failed`	Avaliar	Taxa de falhas em requisições HTTP
`checks`	Avaliar	Proporção de validações bem-sucedidas feitas com `check()`
`http_reqs`	Contador	Total de requisições HTTP feitas
`vus` / `vus_max`	Medidor	Número atual/máximo de usuários virtuais

Métrica	Tipo	Descrição
`iterations`	Contador	Quantidade total de execuções da função principal

Exemplo 1 — Thresholds básicos

Arquivo: `thresholds_basicos.js`

```
import http from 'k6/http';
import { check } from 'k6';

export const options = {
  thresholds: {
    http_req_duration: ['p(95)<500'], // 95% das requisições < 500ms
    http_req_failed: ['rate<0.01'],   // menos de 1% de falhas
    checks: ['rate>0.98'],            // 98% das validações devem passar
  },
};

export default function () {
  const res = http.get('http://localhost:3000/usuarios');
  check(res, { 'status 200': (r) => r.status === 200 });
}
```

Rodar:

```
k6 run thresholds_basicos.js
```

Exemplo 2 — Thresholds com diferentes percentis

Arquivo: `thresholds_percentis.js`

```
import http from 'k6/http';
import { check } from 'k6';

export const options = {
  thresholds: {
    http_req_duration: [
```

```
      'p(90)<400',   // 90% das requisições < 400ms
      'p(99)<1000',  // 99% das requisições < 1s
      'avg<300',     // média geral < 300ms
      'max<2000',    // nenhuma requisição pode ultrapassar 2s
    ],
  },
};

export default function () {
  const res = http.get('http://localhost:3000/produtos');
  check(res, { 'status 200': (r) => r.status === 200 });
}
```

Rodar:

```
k6 run thresholds_percentis.js
```

Exemplo 3 — Thresholds com métricas customizadas

Arquivo: `thresholds_metricas_customizadas.js`

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

const tempoRequisicao = new Trend('tempo_requisicao');
const taxaSucesso = new Rate('taxa_sucesso');

export const options = {
  thresholds: {
    'tempo_requisicao': ['p(95)<300'], // 95% dos tempos < 300ms
    'taxa_sucesso': ['rate>0.99'],     // 99% de sucesso esperado
  },
};

export default function () {
  const res = http.get('http://localhost:3000/usuarios');
```

```
  tempoRequisicao.add(res.timings.duration);

  const ok = check(res, { 'status 200': (r) => r.status === 200 });
  taxaSucesso.add(ok);
}
```

Rodar:

```
k6 run thresholds_metricas_customizadas.js
```

Exemplo 4 — Thresholds com contadores

Arquivo: `thresholds_contadores.js`

```
import http from 'k6/http';
import { sleep } from 'k6';

export const options = {
  vus: 5,
  iterations: 100,
  thresholds: {
    http_reqs: ['count>100'],   // Garante que o teste fez pelo menos 100 req
uisições
    iterations: ['count>=50'],  // Garante que o total de execuções é 50 veze
s ou mais
  },
};

export default function () {
  http.get('http://localhost:3000/status');
  sleep(1);
}
```

Rodar:

```
k6 run thresholds_contadores.js
```

Exemplo 5 — Limiares com filtros (subseletores)

Arquivo: `thresholds_filtros.js`

```
import http from 'k6/http';
import { check } from 'k6';

export const options = {
  thresholds: {
    'http_req_duration{expected_response:true}': ['p(95)<6'],
    'http_req_duration{expected_response:false}': ['p(95)<10'],
  },
};

export default function () {
  const res = http.get('http://localhost:3000/usuarios/0uxuPY0cbmQhpEz
1');

  check(res, {
    'status 200 ou 404': (r) => [200, 404].includes(r.status),
  });
}
```

- **Primeira linha:**

  Para todas as requisições **bem-sucedidas** (`expected_response:true`, ou seja, status < 400),

  o *percentil 95* do tempo deve ser **menor que 600 ms**.

- **Segunda linha:**

  Para todas as requisições **com erro** (`expected_response:false`, ou seja, status >= 400),

  o *percentil 95* do tempo deve ser **menor que 1000 ms**.

Rodar:

```
k6 run thresholds_filtros.js
```

Agregações suportadas

Agregação	Descrição	Exemplo
`avg`	Média	`'avg<500'`
`min`	Mínimo	`'min>100'`
`max`	Máximo	`'max<2000'`
`count`	Número total de registros	`'count>50'`
`p(90)`	90º percentil	`'p(90)<300'`
`p(95)`	95º percentil	`'p(95)<500'`
`p(99)`	99º percentil	`'p(99)<1000'`
`rate`	Proporção (0–1)	`'rate>0.98'`

Quando um threshold falha

Quando uma condição não é atendida:

- O K6 marca o teste como **FAILED** ;
- O log exibe ❌ na métrica correspondente;
- O **exit code** do processo é diferente de zero, permitindo que um pipeline CI/CD interrompa o deploy automaticamente.

Conclusão

Neste módulo você aprendeu:

- Como validar respostas HTTP com `check()`.
- Como interpretar as métricas padrão (`http_req_duration`, `http_reqs`, `checks`, etc.).
- Como acessar detalhes de tempo via `res.timings`.
- Como criar métricas personalizadas (`Trend`, `Counter`, `Rate`, `Gauge`).
- Como usar `thresholds` para definir metas de performance automáticas.

Esses conceitos formam a **base da análise de performance com K6**.

A partir deles, você consegue **entender gargalos**, **automatizar validações** e **garantir que o sistema mantenha o desempenho esperado** mesmo sob carga.