Trabalhando com `options`

O `options` é um objeto especial exportado no seu script K6.

Ele define **como o teste será executado**: número de usuários, duração, iterações, metas, thresholds, etc.

Exemplo simples:

```
export const options = {
  vus: 5,
  duration: '10s',
};
```

Neste caso, o teste roda com **5 usuários virtuais (VUs)** durante **10 segundos**.

Iterações e Duração

No K6, cada VU executa a função `default()` várias vezes — cada execução é chamada de **iteração**.

- `duration`: quanto tempo o teste vai durar.
- `iterations`: quantas vezes o teste será executado (no total).
- `vus`: quantos usuários virtuais vão executar o teste.

Exemplo com número fixo de iterações:

```
export const options = {
  vus: 3,
  iterations: 9,
};
```

Cada VU executará 3 iterações (totalizando 9 execuções).

Se o número de iterações **não dividir igualmente entre os VUs**, o K6 distribui o máximo possível.

Configuração com Cenários (`scenarios`)

A propriedade `scenarios` permite criar **múltiplas configurações de execução** dentro do mesmo teste.

```
export const options = {
  scenarios: {
    cenario1: {
      executor: 'constant-vus',
      vus: 5,
      duration: '15s',
    },
  },
};
```

Assim, é possível nomear cada cenário e personalizar seus parâmetros.

O que é o `executor`

O **executor** define **como os VUs serão criados e mantidos durante o teste**.

Cada tipo de executor serve para **um tipo de simulação diferente**.

Executor	Descrição
`constant-vus`	Mantém o mesmo número de usuários durante todo o teste
`ramping-vus`	Aumenta e diminui gradualmente os usuários
`shared-iterations`	Divide um número fixo de iterações entre os VUs
`constant-arrival-rate`	Mantém uma taxa fixa de requisições por segundo
`ramping-arrival-rate`	Aumenta gradualmente a taxa de requisições

Exemplos práticos com diferentes `executors`

Os *executors* do k6 definem **como e com que ritmo os usuários virtuais (VUs) ou requisições são geradas**.

Cada tipo tem um propósito específico, de acordo com o tipo de carga que você quer simular.

`constant-vus` — carga constante

Usado para testes simples e lineares, onde você quer manter o mesmo número de usuários durante todo o período.

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

export const options = {
  scenarios: {
    carga_constante: {
      executor: 'constant-vus',
      vus: 5,           // 5 usuários virtuais
      duration: '15s',   // durante 15 segundos
    },
  },
};

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

Interpretação:

- Mantém **5 usuários simultâneos** ativos por **15 segundos**.
- Ideal para **testes de estabilidade**, como uma verificação rápida de endpoints.

`ramping-vus` — aumento e redução de carga

Esse tipo de executor simula o **crescimento gradual de usuários**, sendo muito usado em **testes de stress ou endurance**.

```
import http from 'k6/http';

export const options = {
  scenarios: {
    aumento_gradual: {
      executor: 'ramping-vus',
```

```
      startVUs: 1, // começa com 1 usuário
      stages: [
        { duration: '10s', target: 5 },  // sobe para 5 VUs em 10s
        { duration: '10s', target: 10 }, // sobe para 10 VUs em 10s
        { duration: '10s', target: 0 },  // reduz até 0 VUs
      ],
    },
  },
};

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

Interpretação:

- Aumenta gradualmente de **1 até 10 usuários**, depois encerra.
- Serve para observar **como o sistema responde ao aumento de carga**.
- Usado em testes que simulam **aumento orgânico de acessos** (ex: início de expediente em um sistema corporativo).

`shared-iterations` — iterações compartilhadas

Esse executor é ótimo para testes rápidos ou quando o objetivo é executar um número **fixo de iterações**, independentemente do tempo.

```
import http from 'k6/http';

export const options = {
  scenarios: {
    iteracoes_compartilhadas: {
      executor: 'shared-iterations',
      vus: 4,          // 4 usuários virtuais
      iterations: 12,  // 12 iterações no total
    },
  },
};
```

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

Interpretação:

- O total de **12 iterações** será **dividido entre 4 VUs**.

  → Cada VU executará em média **3 iterações**.
- Ideal para **testes de validação funcional rápida**, **smoke tests** ou para scripts de **pré-aquecimento** do ambiente.

`constant-arrival-rate` — taxa fixa de requisições

Esse executor controla **a taxa de requisições por tempo**, e não o número de usuários.

O k6 ajusta automaticamente a quantidade de VUs necessários para manter essa taxa.

```
import http from 'k6/http';

export const options = {
  scenarios: {
    taxa_constante: {
      executor: 'constant-arrival-rate',
      rate: 20,            // 20 requisições por segundo
      timeUnit: '1s',      // a cada segundo
      duration: '15s',     // durante 15 segundos
      preAllocatedVUs: 5,  // VUs pré-alocados
    },
  },
};

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

Interpretação detalhada:

Campo	Significado	Explicação prática
executor	Tipo de execução	Controla o ritmo de requisições.
rate	Taxa de chegada	Define **quantas requisições por segundo** serão enviadas.
timeUnit	Unidade de tempo	`'1s'` indica que a taxa será avaliada por segundo.
duration	Duração do teste	Tempo total de execução.
preAllocatedVUs	Usuários reservados	Quantidade mínima de VUs criados antes do início do teste.

Quando usar:

- Quando você precisa garantir **um volume fixo de requisições** (ex: 50 req/s).
- Ideal para medir **capacidade de throughput (TPS)** e **latência média** sob carga constante.
- Usado em **testes de desempenho sustentado**.

`ramping-arrival-rate` — taxa variável de requisições

Esse executor é usado quando você quer simular **picos de tráfego**, aumentando ou diminuindo **a taxa de requisições** com o tempo.

```
import http from 'k6/http';

export const options = {
  scenarios: {
    taxa_variavel: {
      executor: 'ramping-arrival-rate',
      startRate: 5,        // começa com 5 requisições por segundo
      timeUnit: '1s',      // intervalo base de medição
      preAllocatedVUs: 10,  // VUs pré-criados
      stages: [
        { duration: '10s', target: 10 }, // sobe até 10 req/s
        { duration: '10s', target: 20 }, // sobe até 20 req/s
        { duration: '10s', target: 0 },  // reduz até parar
      ],
```

```
    },
  },
};

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

Explicação completa:

Campo	Significado	Explicação prática
executor	Tipo de execução	`ramping-arrival-rate` muda a taxa de requisições conforme as etapas (stages).
startRate	Taxa inicial	Quantas requisições por segundo o teste começa enviando.
timeUnit	Unidade de tempo	Base de cálculo para o `startRate` e os `stages`.
stages	Etapas de variação	Cada etapa define **quanto tempo dura** e **para qual taxa** a carga deve ir.
preAllocatedVUs	VUs iniciais	Quantos usuários o k6 reserva para manter as taxas.

Comportamento do teste:

1. Começa enviando **5 req/s**.
2. Sobe gradualmente até **10 req/s**, depois **20 req/s**.
3. Finaliza reduzindo até **0 req/s**.

Quando usar:

- Testes que precisam **simular aumento rápido de acessos**, como eventos sazonais (ex: Black Friday).
- Ideal para avaliar **resiliência**, **elasticidade** e **capacidade de escalabilidade** da aplicação.
- Ajuda a identificar o ponto em que o sistema **não consegue mais responder no tempo esperado**.

Cenário com múltiplos executors

Também é possível combinar diferentes estratégias no mesmo teste, executando **cenários paralelos** com comportamentos distintos.

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

export const options = {
  scenarios: {
    cenario1: {
      executor: 'constant-vus',
      vus: 5,
      duration: '15s',
      startTime: '0s',
    },
    cenario2: {
      executor: 'shared-iterations',
      vus: 3,
      iterations: 9,
      startTime: '15s',
    },
  },
};

export function setup() {
  console.log('Iniciando o teste...');
  return { baseUrl: 'http://localhost:3000' };
}

export default function (data) {
  http.get(`${data.baseUrl}/produtos`);
  sleep(1);
}

export function teardown() {
  console.log('Encerrando o teste.');
}
```

Explicação:

- O `cenario1` começa imediatamente e roda **5 VUs fixos** por **15 segundos**.
- O `cenario2` inicia **15 segundos depois**, executando **9 iterações** compartilhadas por **3 VUs**.
- Ideal para simular **diferentes tipos de carga simultânea** — por exemplo, usuários navegando no site e outros realizando compras.

Comparação rápida dos executors

Executor	Ideal para...	Exemplo prático
`constant-vus`	Testes simples e estáveis	10 usuários fixos por 30s
`ramping-vus`	Simular aumento gradual de usuários	Subir de 1 → 20 VUs em 2 min
`shared-iterations`	Execuções controladas e rápidas	100 requisições divididas entre 5 usuários
`constant-arrival-rate`	Garantir volume constante de tráfego	50 req/s durante 5 min
`ramping-arrival-rate`	Simular picos e quedas de tráfego	Crescer 10 → 100 req/s e reduzir novamente