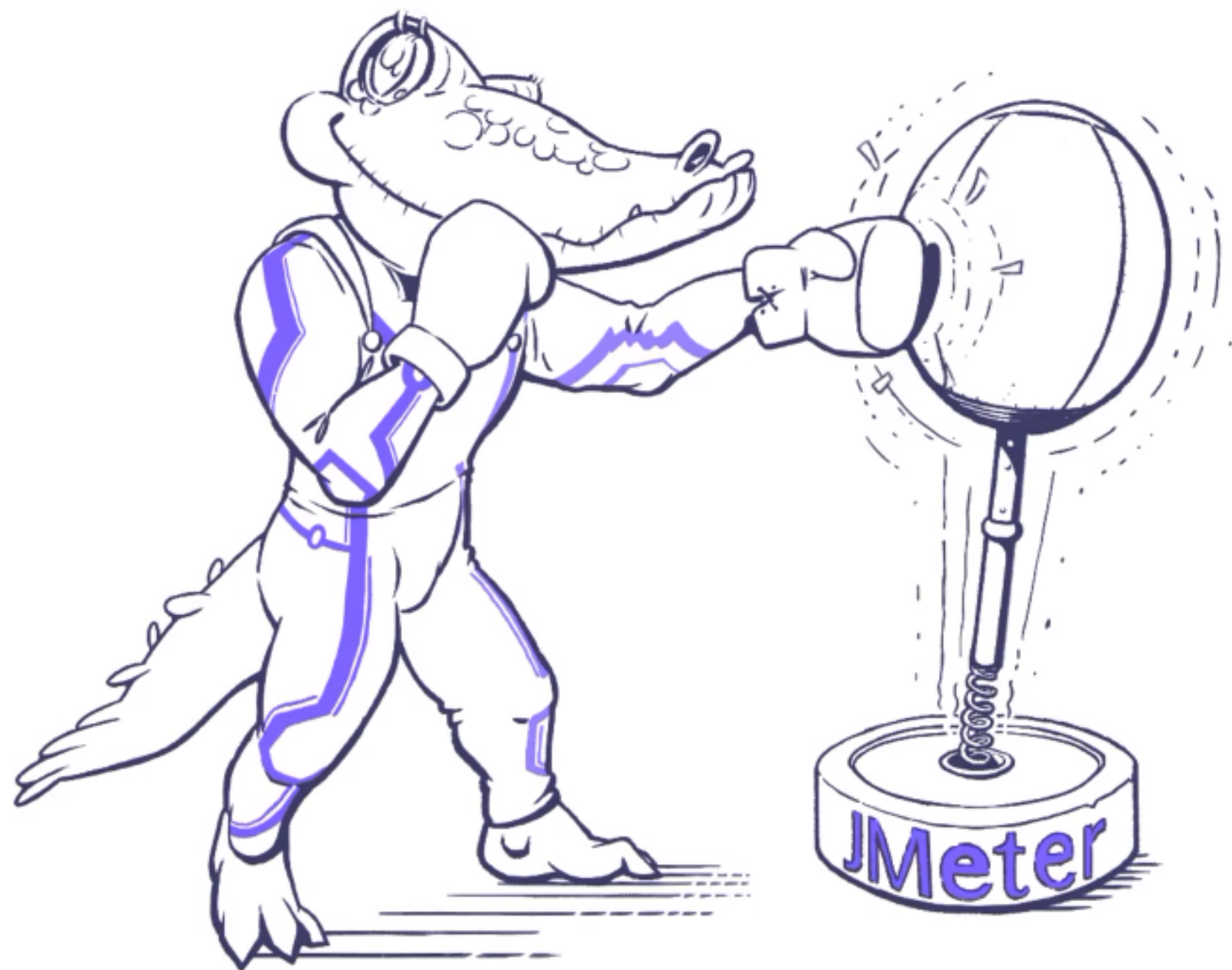


CONHECENDO O K6

PARA TESTES DE PERFORMANCE





Ainda preso com JMeter?

"O esforço de engenharia de escrever testes para apenas um de nossos microsserviços seria de mais de 2 semanas de engenharia no JMeter, em comparação com o ½ dia que levamos no k6!"

Alex Hibbitt, engenheiro de confiabilidade do site.

O QUE É O K6

CARGA - IMERSÃO - PICO - STRESS

- k6 é otimizado para consumo mínimo de recursos e projetado para executar testes de alta carga (testes de pico , estresse , absorção).
- Com o k6, você pode executar testes com uma pequena quantidade de carga para validar continuamente o desempenho e a disponibilidade do seu ambiente de produção.
- k6 fornece uma arquitetura extensível. Você pode usar o k6 para simular o tráfego como parte de seus experimentos de caos ou acioná-los a partir de seus testes do k6.



Obs*: Não é executado nativamente em um navegador. Por padrão, o k6 não renderiza páginas da web da mesma forma que um navegador. Os navegadores podem consumir recursos significativos do sistema. Ignorar o navegador permite executar mais carga em uma única máquina. No entanto, com xk6-browser , você pode interagir com navegadores reais e coletar métricas de front-end como parte de seus testes k6
[<https://www.youtube.com/watch?v=y04wavsZxSs>]

INSTALANDO O K6

K6 TEM PACOTES PARA LINUX, MAC E WINDOWS. COMO ALTERNATIVA, VOCÊ PODE USAR UM CONTÊINER DO DOCKER OU UM BINÁRIO AUTÔNOMO.

Linux

Debian/Ubuntu

```
sudo gpg --no-default-keyring --keyring /usr/share/keyrings/k6-archive-keyring.gpg --keyserver  
echo "deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg] https://dl.k6.io/deb stable ma  
sudo apt-get update  
sudo apt-get install k6
```

Fedora/CentOS

Usando `dnf` (ou `yum` em versões mais antigas):

```
sudo dnf install https://dl.k6.io/rpm/repo.rpm  
sudo dnf install k6
```

Docker

```
docker pull grafana/k6
```

MacOS

Using [Homebrew](#):

```
brew install k6
```

Windows

If you use the [Chocolatey package manager](#) you can install the unofficial k6 package with:

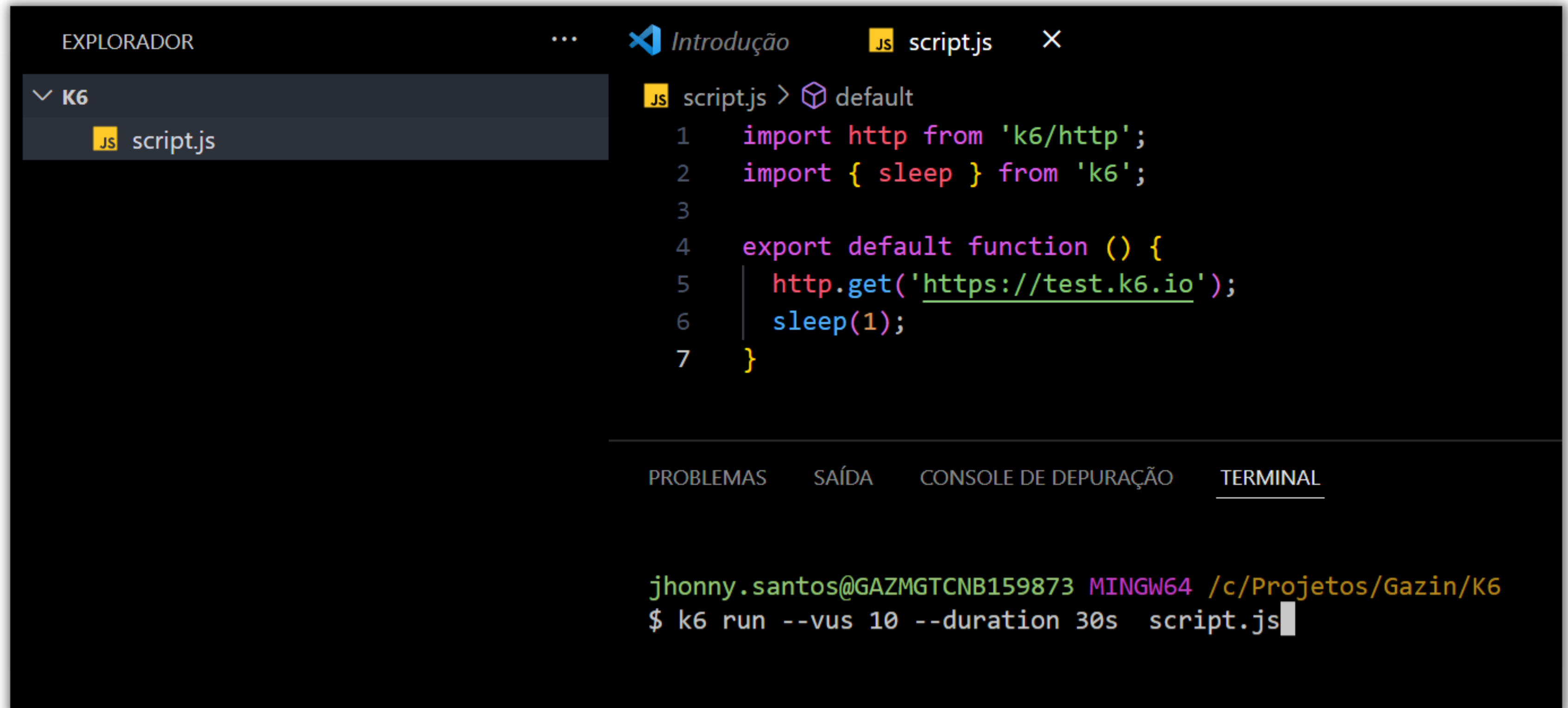
```
choco install k6
```

If you use the [Windows Package Manager](#), install the official packages from the k6 manifests ([created by the community](#)):

```
winget install k6
```

Alternatively, you can download and run [the latest official installer](#).

PRIMEIRO TESTE



The image shows a Visual Studio Code editor window with a dark theme. On the left, the 'EXPLORADOR' (Explorer) sidebar shows a folder named 'K6' containing a file 'script.js'. The main editor area has two tabs: 'Introdução' and 'script.js'. The 'script.js' tab is active, displaying a JavaScript script for K6. The script imports 'http' from 'k6/http' and 'sleep' from 'k6', then exports a default function that performs an HTTP GET request to 'https://test.k6.io' and sleeps for 1 second. Below the editor, the 'TERMINAL' panel is open, showing the command prompt for 'jhonny.santos@GAZMGTCNB159873' in a 'MINGW64' shell, with the command '\$ k6 run --vus 10 --duration 30s script.js' entered.

```
EXPLORADOR
```

▼ K6

- Js script.js

```
Js script.js > default
1  import http from 'k6/http';
2  import { sleep } from 'k6';
3
4  export default function () {
5      http.get('https://test.k6.io');
6      sleep(1);
7  }
```

PROBLEMAS SAÍDA CONSOLE DE DEPURAÇÃO TERMINAL

```
jhonny.santos@GAZMGTCNB159873 MINGW64 /c/Projetos/Gazin/K6
$ k6 run --vus 10 --duration 30s script.js
```

RUSULTADOS EXPORTADOS PARA ONDE QUISER

- Resultados Locais
- AWS CloudWatch / k6 Cloud
- CSV / JSON
- Datadog / Prometheus
- Grafana Cloud
- Outros



execution: local
script: lscript.js
output: -

scenarios: (100.00%) 1 scenario, 1 max VUs, 10m30s max duration (incl. graceful stop):
* default: 1 iterations for each of 1 VUs (maxDuration: 10m0s, gracefulStop: 30s)

running (00m01.7s), 0/1 VUs, 1 complete and 0 interrupted iterations
default ✓ [=====] 1 VUs 00m01.7s/10m0s 1/1 iters, 1 per VU

```
data_received.....: 17 kB 9.6 kB/s
data_sent.....: 442 B 254 B/s
http_req_blocked.....: avg=624.52ms min=624.52ms med=624.52ms max=624.52ms p(90)=624.52ms p(95)=624.52ms
http_req_connecting.....: avg=112.66ms min=112.66ms med=112.66ms max=112.66ms p(90)=112.66ms p(95)=112.66ms
http_req_duration.....: avg=110.51ms min=110.51ms med=110.51ms max=110.51ms p(90)=110.51ms p(95)=110.51ms
{ expected_response:true }...: avg=110.51ms min=110.51ms med=110.51ms max=110.51ms p(90)=110.51ms p(95)=110.51ms
http_req_failed.....: 0.00% ✓ 0 × 1
http_req_receiving.....: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
http_req_sending.....: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
http_req_tls_handshaking.....: avg=151.08ms min=151.08ms med=151.08ms max=151.08ms p(90)=151.08ms p(95)=151.08ms
http_req_waiting.....: avg=110.51ms min=110.51ms med=110.51ms max=110.51ms p(90)=110.51ms p(95)=110.51ms
http_reqs.....: 1 0.573656/s
iteration_duration.....: avg=1.74s min=1.74s med=1.74s max=1.74s p(90)=1.74s p(95)=1.74s
iterations.....: 1 0.573656/s
vus.....: 1 min=1 max=1
vus_max.....: 1 min=1 max=1
```


INDICADORES - K6

- **VUS**

Quantidade ativa e executada de usuários virtuais;

- **VUS_MAX**

A quantidade máxima atingida de usuários virtuais ;

- **Iterations**

Quantas vezes, os nossos usuários virtuais executaram os script de testes;

- **Iterations_duration**

Quanto tempo nossos usuários virtuais, levaram para executar, cada vez que eles executaram o script inteiro;

- **Dropped_interations**

Quantas iterações foram dropadas, por problemas no teste ou por problemas na aplicação por tempo de resposta;

- **Data_received**

Quantidade de dados, qual tamanho das requisições totais que nossos usuários reportaram;

- **Data_sent**

Quantidade de dados, qual tamanho das requisições totais que nossos usuários enviaram para a aplicação;

- **Checks**

Quantidade de checks com segurança por nosso teste



INDICADORES - HTTP

- **http_reqs**

Quantidade de requisições geradas no total;

- **http_reqs_blocked**

Quanto tempo nossos usuários virtuais, passaram esperando (Largura de banda);

- **http_reqs_connecting**

Quanto tempo nosso usuário virtual, levou em média para estabelecer uma conexão com a aplicação (latência);

- **http_reqs_TLS_handshaking**

quanto tempo o certificado ssl, leva para ser entregue e assinado pelos clientes;

- **http_reqs_sending**

Quanto tempo levamos enviando dados, para a aplicação;

- **http_reqs_waiting**

quanto tempo fica-se esperando pela resposta da aplicação depois de uma ação (percepção do usuário na página);

- **http_reqs_receiving**

quanto tempo ficamos esperando a resposta da aplicação, até que seja concluída;

- **http_reqs_duration**

duração total do tempo da requisição (sending + waiting + receiving);

- **http_reqs_failed (>=0.31)**

quantidade de requisições que falharam;





BOAS
PRÁTICAS

O QUE QUER TESTAR ?

- **O que faz essa aplicação ?**

11 módulos, que em conjunto entregam uma experiência de marketplace;

- **Quem acessa essa aplicação? De onde acessa? Como acessa?**

70% via PC, 30% mobile (todos no Brasil);

- **Quantos acessos por dia ? Quantos costumam acontecer em simultâneo ?**

2000 usuários por dia. Em pico de vendas 60 requisições por minuto em períodos contínuos médios de 2h constante;

- **Qual é a stack tecnológica? Quais são dependências externas ?**

Múltiplas stacks, como dependência de serviços internos (microserviços) e externos como cálculo de frete e autenticação;

- **Onde está hospedada a aplicação ?**

Google Cloud, no Brasil;



PERCEPÇÃO PARA OTIMIZAR O DESEMPENHO.

CADA TIPO DE TESTE É PROJETADO PARA FORNECER INFORMAÇÕES DIFERENTES SOBRE SEU SISTEMA.

- Teste de Fumaça (Smoke Test)
- Teste de Carga (Load Test)
- Teste de Estresse
- Teste de Pico (Spike Test)
- Teste de Imersão (Soak testing)

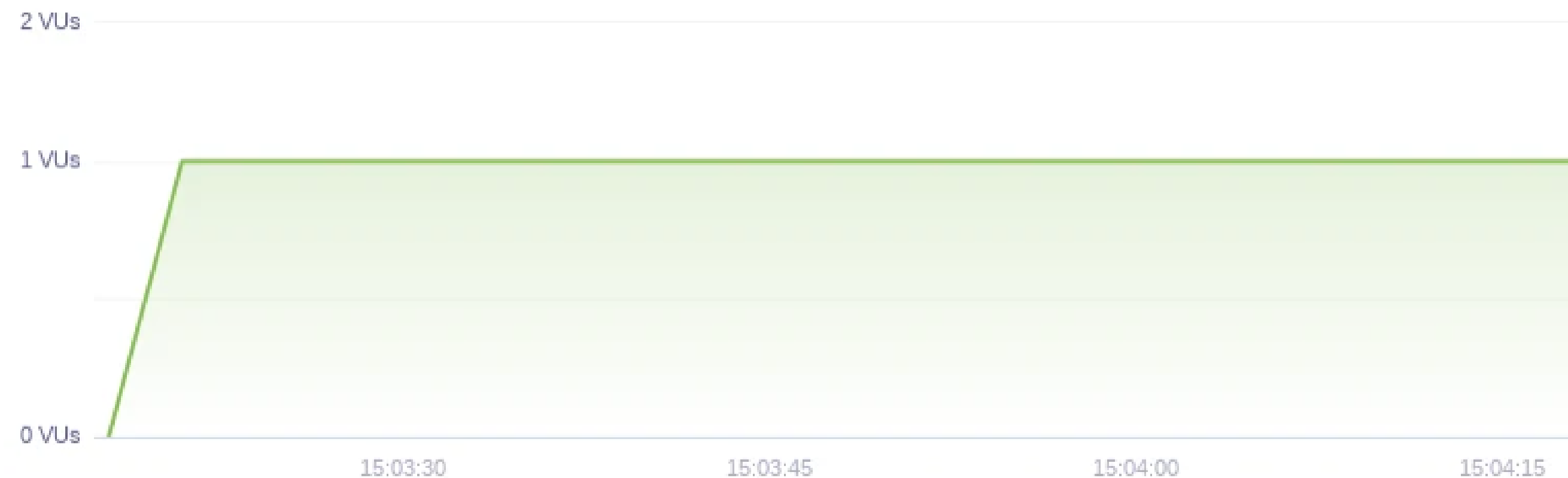


SMOKE TEST

Teste de fumaça é um teste de carga regular, configurado para carga mínima. Você deseja executar um teste de fumaça como uma verificação de sanidade toda vez que escrever um novo script ou modificar um script existente.

Você deseja executar um teste de fumaça para:

- Verifique se seu script de teste não contém erros.
- Verifique se o seu sistema não gera nenhum erro quando está sob carga mínima.

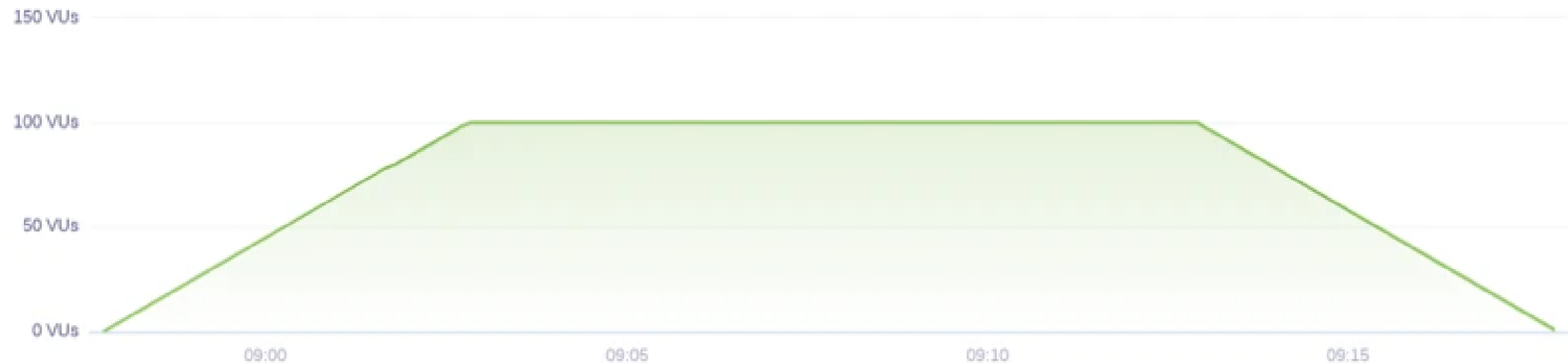


TESTE DE CARGA

O Load Testing se preocupa principalmente com a avaliação do desempenho atual do seu sistema em termos de usuários ou solicitações simultâneas por segundo.

Quando você quiser entender se seu sistema está atendendo às metas de desempenho, esse é o tipo de teste que você executará.

- Avalie o desempenho atual do seu sistema sob carga típica e de pico.
- Certifique-se de continuar atendendo aos padrões de desempenho ao fazer alterações em seu sistema (código e infraestrutura).



TESTE DE ESTRESSE

Teste de estresse é um tipo de teste de carga usado para determinar os limites do sistema. O objetivo deste teste é verificar a estabilidade e confiabilidade do sistema sob condições extremas .

- Como seu sistema se comportará sob condições extremas.
- Qual é a capacidade máxima do seu sistema em termos de usuários ou taxa de transferência.
- O ponto de ruptura do seu sistema e seu modo de falha.
- Se o seu sistema se recuperar sem intervenção manual após o término do teste de estresse.



TESTE DE PICO

O teste de pico é uma variação de um teste de estresse, mas não aumenta gradualmente a carga. Em vez disso, atinge uma carga extrema em um período muito curto de tempo.

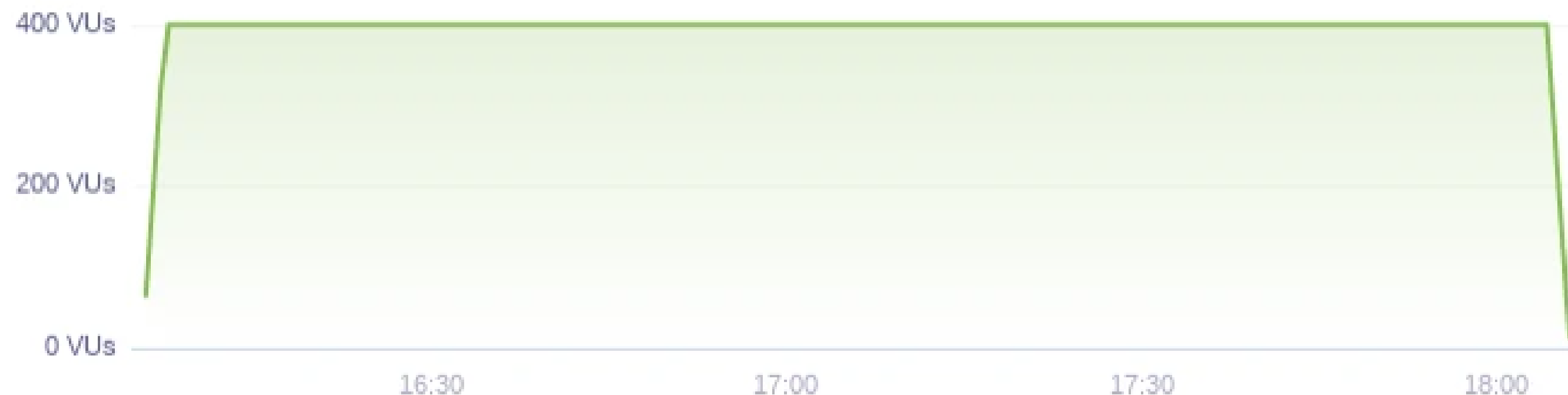
- Como seu sistema funcionará sob um aumento repentino de tráfego.
- Se o seu sistema irá se recuperar assim que o tráfego diminuir.



TESTE DE IMERSÃO

O teste de imersão está preocupado com a confiabilidade por um longo período de tempo.

- Verifique se o seu sistema não sofre de bugs ou estouro de memória, que resultam em travamento ou reinicialização após várias horas de operação.
- Verifique se as reinicializações esperadas do aplicativo não perdem solicitações.
- Encontre bugs relacionados a condições de corrida que aparecem esporadicamente.
- Certifique-se de que seu banco de dados não esgote o espaço de armazenamento alocado e pare.
- Certifique-se de que seus logs não esgotem o armazenamento em disco alocado.
- Certifique-se de que os serviços externos dos quais você depende não parem de funcionar após a execução de uma determinada quantidade de solicitações.



CENÁRIOS

Os cenários nos permitem fazer configurações detalhadas de como as VUs e as iterações são agendadas. Isso possibilita modelar diversos padrões de tráfego em testes de carga. Os benefícios do uso de cenários incluem:

- Vários cenários podem ser declarados no mesmo script e cada um pode executar independentemente uma função JavaScript diferente, o que torna a organização de testes mais fácil e flexível.
- Cada cenário pode usar uma VU distinta e um padrão de agendamento de iteração, alimentado por um executor específico. Isso permite a modelagem de padrões de execução avançados que podem simular melhor o tráfego do mundo real.
- Os cenários são independentes uns dos outros e executados em paralelo, embora possam parecer sequenciais, definindo o horário de início e o fim de cada um com cuidado.
- Diferentes variáveis de ambiente e tags de métrica podem ser definidas por cenário.



EXECUTORES

Os executores são "o diferencial" do k6. Cada um programa VUs e iterações de forma diferente, e você escolherá um dependendo do tipo de tráfego que deseja modelar para testar seus serviços.

- Os cenários são independentes uns dos outros e executados em paralelo, embora possam parecer sequenciais.
- Diferentes variáveis de ambiente e tags de métrica podem ser definidas por cenário.

| NAME | VALUE | DESCRIÇÃO |
|-----------------------|-----------------------|--|
| Shared iterations | shared-iterations | Uma quantidade fixa de iterações é "compartilhada" entre várias VUs. |
| Per VU iterations | per-vu-iterations | Cada VU executa um número exato de iterações. |
| Constant VUs | constant-vus | Um número fixo de VUs executa tantas iterações quanto possível por um período de tempo especificado. |
| Ramping VUs | ramping-vus | Um número variável de VUs executa tantas iterações quanto possível por um período de tempo especificado. |
| Constant Arrival Rate | constant-arrival-rate | Um número fixo de iterações é executado em um período de tempo especificado. |
| Ramping Arrival Rate | ramping-arrival-rate | Um número variável de iterações é executado em um período de tempo especificado. |
| Externally Controlled | externally-controlled | Controle e dimensione a execução em tempo de execução por meio da API REST do k6 ou da CLI . |

MATERIAL DE SUPORTE



https://github.com/jhonnyrobert/Projeto_k6_Grafana

<https://k6.io/docs/>

<https://www.udemy.com/course/performance-test-primeiros-passos-com-o-k6/>

<https://www.youtube.com/watch?v=gqvpc08uF6E>

CODE TESTING

