Grafana Labs documentation

Search docs

Product

Grafana k6

Viewing: v0.51.x (latest)

Find another version

Grafana k6 documentation

- Get started
- Set up
- Using k6
  - HTTP Requests
  - Metrics
  - Checks
  - Thresholds
  - Options
  - Test lifecycle

Open source

Open and closed models

Different k6 executors have different ways of scheduling VUs. Some executors use the *closed model*, while the arrival-rate executors use the *open model*.

In short, in the closed model, VU iterations start only when the last iteration finishes. In the open model, on the other hand, VUs arrive independently of iteration completion. Different models suit different test aims.

Closed Model

In a closed model, the execution time of each iteration dictates the number of iterations executed in your test. The next iteration doesn't start until the previous one finishes.

Thus, in a closed model, the start or arrival rate of new VU iterations is tightly coupled with the iteration duration (that is, time from start to finish of the VU's `exec` function, by default the `export default function`):

JavaScript — Copy

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

export const options = {
  scenarios: {
    closed_model: {
      executor: 'constant-vus',
      vus: 1,
      duration: '1m',
    },
  },
};

export default function () {
  // The following request will take roughly 6s to complete,
  // resulting in an iteration duration of 6s.

  // As a result, New VU iterations will start at a rate of 1 per 6s,
  // and we can expect to get 10 iterations completed in total for the
  // full 1m test duration.
```

1 minute duration

Duration = iteration duration * iterations
= 6.31s * 10
= 63.1s ~ 1m

Expand code

Running this script would result in something like:

bash — Copy

```
running (1m01.5s), 0/1 VUs, 10 complete and 0 interrupted iterations
closed_model ✓ [======================================] 1 VUs  1m0s
```

Drawbacks of using the closed model

When the duration of the VU iteration is tightly coupled to the start of new VU iterations, the target system's response time can influence the throughput of the test. Slower response times means longer iterations and a lower arrival rate of new iterations—and vice versa for faster response times. In some testing literature, this problem is known as *coordinated omission*.

In other words, when the target system is stressed and starts to respond more slowly, a closed model load test will wait, resulting in increased iteration durations and a tapering off of the arrival rate of new VU iterations.

This effect is not ideal when the goal is to simulate a certain arrival rate of new VUs, or more generally throughput (e.g. requests per second).

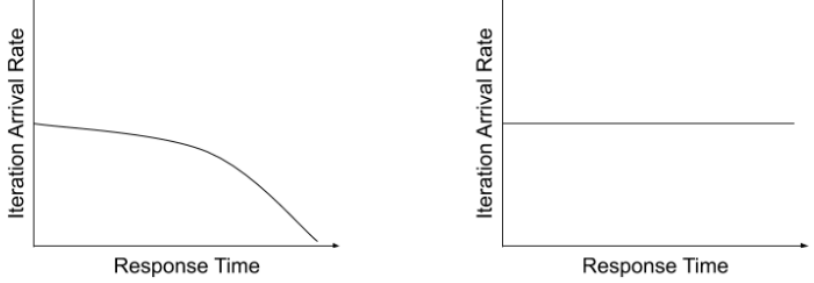
Open model

Compared to the closed model, the open model decouples VU iterations from the iteration duration. The response times of the target system no longer influence the load on the target system.

To fix this problem of coordination, you can use an open model, which decouples the start of new VU iterations from the iteration duration. This reduces the influence of the target system's response time.

Closed Model

Open Model



k6 implements the open model with two *arrival rate* executors: constant-arrival-rate and ramping-arrival-rate :

JavaScript Copy

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

export const options = {
  scenarios: {
    open_model: {
      executor: 'constant-arrival-rate',
      rate: 1,
      timeUnit: '1s',
      duration: '1m',
      preAllocatedVUs: 20,
    },
  },
};

export default function () {
  // With the open model arrival rate executor config above,
  // new VU iterations will start at a rate of 1 every second,
  // and we can thus expect to get 60 iterations completed
  // for the full 1m test duration.
```

Expand code

Running this script would result in something like:

bash Copy

```
running (1m09.3s), 000/011 VUs, 60 complete and 0 interrupted iterations
open_model ✓ [======================================] 011/011 VUs  1m0s  1 iters/s
```

Was this page helpful?

Suggest an edit Contribute to docs Report a problem

Related documentation

Community Support

Performance testing with Grafana Cloud k6

Average-load testing

Use the test builder

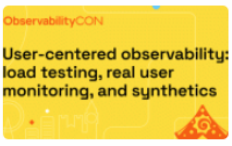
Related resources from Grafana Labs

Additional helpful documentation, links, and articles:

Video

Performance testing and observability in Grafana Cloud

In this webinar, learn how Grafana Cloud k6 offers you the best developer experience for performance testing.



User-centered observability: load testing, real user monitoring, and synthetics

Learn how to use load testing, synthetic monitoring, and real user monitoring (RUM) to understand end users'

experience of your apps. Watch on demand.

Sign up for Grafana stack updates

Email

Subscribe

Note: By signing up, you agree to be emailed related product-level information.

Grafana Labs

Grafana

- Overview
- Deployment options
- Plugins
- Dashboards

Products

- Grafana Cloud
  - Grafana Cloud Status
- Grafana Enterprise Stack
- Grafana Cloud Application Observability
- Grafana Cloud Frontend Observability
- Grafana Cloud IRM
- Grafana Cloud k6
- Grafana Cloud Logs
- Grafana Cloud Metrics
- Grafana Cloud Profiles
- Grafana Cloud Synthetic Monitoring
- Grafana SLO

Open Source

- Grafana
- Grafana Loki
- Grafana Mimir
- Grafana OnCall
- Grafana Tempo
- Grafana Agent
- Grafana Alloy
- Grafana k6
- Prometheus
- Grafana Faro
- Grafana Pyroscope
- Grafana Beyla
- OpenTelemetry
- Grafana Tanka
- Graphite
- GitHub

Learn

- Grafana Labs blog
- Documentation
- Downloads
- Community
- Community forums
- Community Slack
- Grafana Champions
- Community organizers
- Grafana ObservabilityCON
- GrafanaCON 2024
- The Golden Grot Awards
- Successes
- Workshops
- Videos
- OSS vs Cloud
- Load testing

Company

- The team
- Press
- Careers
- Events
- Partnerships
- Contact
- Getting help
- Merch

Grafana Cloud Status

Legal and Security | Terms of Service | Privacy Policy | Trademark Policy