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

Constant arrival rate

With the `constant-arrival-rate` executor, k6 starts a fixed number of iterations over a specified period of time. It is an open-model executor, meaning iterations start independently of system response (for details, read Open and Closed models).

This executor continues to start iterations at the given rate as long as VUs are available. The time to execute an iteration can vary with test logic or the system-under-test response time. To compensate for this, the executor starts a varied number of VUs to meet the configured iteration rate. For explanations of how allocation works, read Arrival-rate VU allocation.

> **NOTE**
>
> **Iteration starts are spaced fractionally.** Iterations **do not** start at exactly the same time. At a `rate` of `10` with a `timeUnit` of `1s`, each iteration starts about every tenth of a second (that is, each 100ms).

Options

Besides the common configuration options, this executor has the following options:

Expand table

Option	Type	Description	Default
duration(required)	string	Total scenario duration (excluding `gracefulStop`).	-
rate(required)	integer	Number of iterations to start during each `timeUnit` period.	-
preAllocatedVUs(required)	integer	Number of VUs to pre-allocate before test start to preserve runtime resources.	-
timeUnit	string	Period of time to apply the `rate` value.	`"1s"`
maxVUs	integer	Maximum number of VUs to allow during the test run.	If unset, same as `preAllocatedVUs`

When to use

When you want iterations to remain constant, independent of the performance of the system under test. This approach is useful for a more accurate representation of RPS, for example.

> **NOTE**
>
> **Don't put sleep at the end of an iteration.**
>
> The arrival-rate executors already pace the iteration rate through the `rate` and `timeUnit` properties. So it's unnecessary to use a `sleep()` function at the end of the VU code.

Example

This example schedules a constant rate of 30 iterations per second for 30 seconds. It pre-allocates 2 VUs, and allows k6 to dynamically schedule up to 50 VUs as needed.

JavaScript — Copy

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

export const options = {
  discardResponseBodies: true,
  scenarios: {
    contacts: {
      executor: 'constant-arrival-rate',

      // How long the test lasts
      duration: '30s',

      // How many iterations per timeUnit
      rate: 30,
```

```
// Start `rate` iterations per second
timeUnit: '1s',

// Pre-allocate 2 VUs before starting the test
preAllocatedVUs: 2,
```

Expand code

Observations

The following graph depicts the performance of the example script:

Based upon our test scenario inputs and results:

- The desired rate of 30 iterations started every 1 second is achieved and maintained for the majority of the test.
- The test scenario runs for the specified 30 second duration.
- Having started with 2 VUs (as specified by the `preAllocatedVUs` option), k6 automatically adjusts the number of VUs to achieve the desired rate, up to the `maxVUs`. For this test, this ended up as 17 VUs.
- The number of VUs to achieve the desired rate varies depending on how long each iteration takes to execute. For this test definition, if it would take exactly 1 second, then 30 VUs would be needed. However, as it takes less than 1 second, then less VUs are needed.

Using too low of a `preAllocatedVUs` setting will reduce the test duration at the desired rate, as resources need to continually be allocated to achieve the rate.

Related documentation

Community Support

API load testing

Performance testing with Grafana Cloud k6

Use the test builder

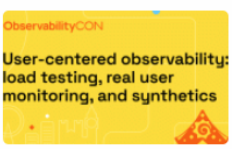
Related resources from Grafana Labs

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

Sitemap Legal and Security Terms of Service Privacy Policy Trademark Policy