Ramping VUs

SUGGEST EDITS

With the `ramping-vus` executor, a variable number of VUs executes as many iterations as possible for a specified amount of time.

For a shortcut to this executor, use the **stages** option.

Options

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

OPTION	TYPE	DESCRIPTION	DEFAULT
stages(required)	array	Array of objects that specify the target number of VUs to ramp up or down to.	`[]`
startVUs	integer	Number of VUs to run at test start.	`1`
gracefulRampDown	string	Time to wait for an already started iteration to finish before stopping it during a ramp down.	`"30s"`

When to use

This executor is a good fit if you need VUs to ramp up or down during specific periods of time.

Example

This example schedules a two-stage test, ramping up from 0 to 10 VUs over 20 seconds, then down to 0 VUs over 10 seconds.

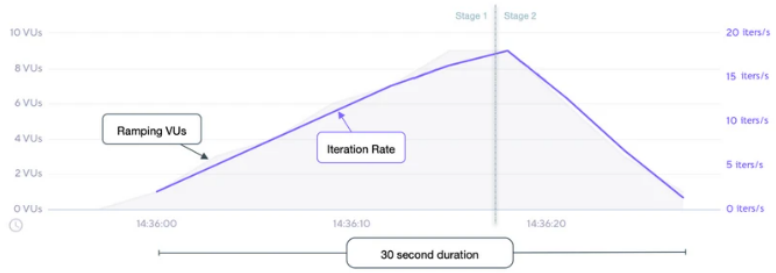
ramping-vus.js

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

export const options = {
  discardResponseBodies: true,
  scenarios: {
    contacts: {
      executor: 'ramping-vus',
      startVUs: 0,
      stages: [
        { duration: '20s', target: 10 },
        { duration: '10s', target: 0 },
      ],
      gracefulRampDown: '0s',
    },
  },
};

export default function () {
  http.get('https://test.k6.io/contacts.php');
  // Injecting sleep
  // Sleep time is 500ms. Total iteration time is sleep + time to finish
  sleep(0.5);
}
```

> **NOTE**
>
> With `gracefulRampDown` set to 0 seconds, some iterations might be interrupted during the ramp down stage.

Observations

The following graph depicts the performance of the example script:



Based upon our test scenario inputs and results:

- The configuration defines 2 stages for a total test duration of 30 seconds;
- Stage 1 ramps *up* VUs linearly from the `startVUs` of 0 to the target of 10 over a 20 second duration;
- From the 10 VUs at the end of stage 1, stage 2 then ramps *down* VUs linearly to the target of 0 over a 10 second duration;
- Each *iteration* of the `default` function is expected to be roughly 515ms, or ~2/s;
- As the number of VUs changes, the iteration rate directly correlates; each addition of a VU increases the rate by about 2 iters/s, whereas each subtraction of a VU reduces by about 2 iters/s;
- The example performed ~300 iterations over the course of the test.

Get the stage index

To get the current running stage index, use the `getCurrentStageIndex` helper function from the k6-jslib-utils library. It returns a zero-based number equal to the position in the shortcut `stages` array or in the executor's `stages` array.

```
import { getCurrentStageIndex } from 'https://jslib.k6.io/k6-utils/1.3.0/ind

export const options = {
  stages: [
    { target: 10, duration: '30s' },
    { target: 50, duration: '1m' },
    { target: 10, duration: '30s' },
  ],
};

export default function () {
  if (getCurrentStageIndex() === 1) {
    console.log('Running the second stage where the expected target is 50');
  }
}
```

Using this feature, it is possible to automatically tag using the current running stage. Check the Tagging stages section for more details.

PREVIOUS
Constant VUs

NEXT
Constant arrival rate

Subscribe to our newsletter!
Product developments and news from the k6 community.

Email

SUBSCRIBE NOW