Ramping arrival rate

SUGGEST EDITS

With the `ramping-arrival-rate` executor, k6 starts iterations at a variable rate. It is an open-model executor, meaning iterations start independently of system response (for details, read Open and Closed models).

This executor has *stages* that configure target number of iterations and the time k6 takes to reach or stay at this target. Unlike the ramping VUs executor, which configures VUs, this executor dynamically changes the number of iterations to start, and starts these iterations as long as the test has enough allocated VUs. To learn how allocation works, read Arrival-rate VU allocation.

Similar to the constant arrival rate model, the ramping arrival rate model is considered open because it allows for dynamic changes in the arrival rate without imposing strict limits on the system's capacity or the number of incoming requests.

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:

OPTION	TYPE	DESCRIPTION	DEFAULT
stages(required)	array	Array of objects that specify the target number of iterations to ramp up or down to.	`[]`
preAllocatedVUs(required)	integer	Number of VUs to pre-allocate before test start to preserve runtime resources.	-
startRate	integer	Number of iterations to execute each `timeUnit` period at test start.	`0`
timeUnit	string	Period of time to apply the `startRate` to the `stages` ' `target` value. Its value is constant for the whole duration of the scenario, it is not possible to change it for a specific stage.	`"1s"`
maxVUs	integer	Maximum number of VUs to allow during the test run.	If unset, same as `preAllocatedVUs`

When to use

If you need start iterations independent of system-under-test performance, and want to ramp the number of iterations up or down during specific periods of time.

> **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. It's unnecessary to use a `sleep()` function at the end of the VU code.

Example

This is an example of a four-stage test.

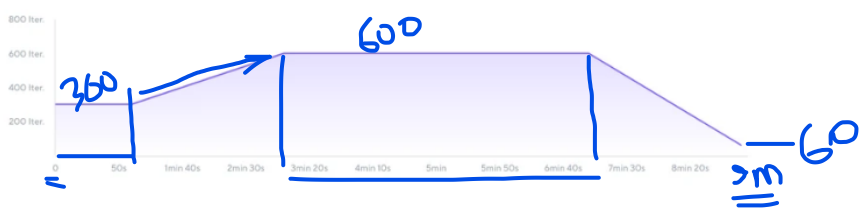
It starts at the defined `startRate`, 300 iterations per minute over a one minute period. After one minute, the iteration rate ramps to 600 iterations started per minute over the next two minutes, and stays at this rate for four more minutes. In the last two minutes, it ramps down to a target of 60 iterations per minute.

ramping-arr-rate.js

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

export const options = {
  discardResponseBodies: true,

  scenarios: {
    contacts: {
      executor: 'ramping-arrival-rate',

      // Start iterations per `timeUnit`
      startRate: 300,

      // Start `startRate` iterations per minute
      timeUnit: '1m',

      // Pre-allocate necessary VUs.
      preAllocatedVUs: 50,

      stages: [
        // Start 300 iterations per `timeUnit` for the first minute.
        { target: 300, duration: '1m' },

        // Linearly ramp-up to starting 600 iterations per `timeUnit` ove
        { target: 600, duration: '2m' },

        // Continue starting 600 iterations per `timeUnit` for the follow
        { target: 600, duration: '4m' },

        // Linearly ramp-down to starting 60 iterations per `timeUnit` ov
        { target: 60, duration: '2m' },
      ],
    },
  },
};

export default function () {
  http.get('https://test.k6.io/contacts.php');
}
```

Observations

The following graph depicts the performance of the example script:



Based upon our test scenario inputs and results:

- The configuration defines 4 stages for a total test duration of 9 minutes.
- Stage 1 maintains the `startRate` iteration rate at 300 iterations started per minute for 1 minute.
- Stage 2 ramps *up* the iteration rate linearly from the *stage 1* of 300 iterations

started per minute, to the target of 600 iterations started per minute over a 2-minute duration.

- Stage 3 maintains the *stage 2* iteration rate at 600 iterations started per minute over a 4-minute duration.
- Stage 4 ramps *down* the iteration rate linearly to the target rate of 60 iterations started per minute over the last two minutes duration.
- Changes to the iteration rate are performed by k6, adjusting the number of VUs
- The script waits for a period of time (defined by the `gracefulStop` option) for iterations to finish. It doesn't start new iterations during the `gracefulStop` period.
- Our example performed, 4020 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 arrival rate

NEXT
Externally controlled