Executors

SUGGEST EDITS

All executors

Executors control how k6 schedules VUs and iterations. The executor that you choose depends on the goals of your test and the type of traffic you want to model.

Define the executor in `executor` key of the scenario object. The value is the executor name separated by hyphens.

```
export const options = {
  scenarios: {
    arbitrary_scenario_name: {
      //Name of executor
  },
};
```

All executors

The following table lists all k6 executors and links to their documentation.

NAME	VALUE	DESCRIPTION
Shared iterations	`shared-iterations`	A fixed amount of iterations are shared between a number of VUs.
Per VU iterations	`per-vu-iterations`	Each VU executes an exact number of iterations.
Constant VUs	`constant-vus`	A fixed number of VUs execute as many iterations as possible for a specified amount of time.
Ramping VUs	`ramping-vus`	A variable number of VUs execute as many iterations as possible for a specified amount of time.
Constant Arrival Rate	`constant-arrival-rate`	A fixed number of iterations are executed in a specified period of time.
Ramping Arrival Rate	`ramping-arrival-rate`	A variable number of iterations are executed in a specified period of time.
Externally Controlled	`externally-controlled`	Control and scale execution at runtime via k6's REST API or the CLI.

```
executor: 'shared-iterations',
vus: 10,
iterations: 200,
maxDuration: '30s',
```
Total iterations = 200

```
executor: 'per-vu-iterations',
vus: 10,
iterations: 20,
maxDuration: '30s',
```
Total iterations = 10 * 20 = 200

```
executor: 'constant-vus',
vus: 10,
duration: '30s',
```
Total iterations = 900 iterations are produced roughly within 30s

```
executor: 'ramping-vus',
startVUs: 0,
stages: [
  { duration: '20s', target: 10 },
  { duration: '10s', target: 0 },
],
gracefulRampDown: '0s',
```

```
executor: 'constant-arrival-rate',
duration: '30s', // How long the test lasts
rate: 30, // How many iterations per timeUnit
timeUnit: '1s', // Start `rate` iterations per second
preAllocatedVUs: 50,  // Pre-allocate VUs
```

```
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` over the following two minutes.
      { target: 600, duration: '2m' },
      // Continue starting 600 iterations per `timeUnit` for the following four minutes.
      { target: 600, duration: '4m' },
      // Linearly ramp-down to starting 60 iterations per `timeUnit` over the last two minutes.
      { target: 60, duration: '2m' },
    ]
```

```
executor: 'externally-controlled',
vus: 10,
maxVUs: 50,
duration: '10m',
```

Shared iterations
Per VU iterations
Constant VUs
Ramping VUs
Contant arrival rate
Ramping arrival rate
Extremely controlled

Open model

VUS MIGHT NOT DISTRIBUTE UNIFORMELY OVER ITERATIONS

For any given scenario, you can't guarantee that a specific VU can run a specific iteration.

With `SharedArray` and **execution context variables**, you can map a specific VU to a specific value in your test data. So the tenth VU could use the tenth item in your array (or the sixth iteration to the sixth item).

But, you *cannot* reliably map, for example, the tenth VU to the tenth iteration.

PREVIOUS
Dropped iterations

NEXT
Shared iterations