To install k6, we go to official site, and get the K6 binary files

****

Then We Create Basic JS-Script Using K6-modules:

```
import http from 'k6/http'

export default function() {
    http.get('https://quickpizza.grafana.com/')

}
```

****

Then We Run The Script Using: *k6 run script-name-here.js*

- Example: ***k6 run k6-script-01.js***

```
C:\Tests\K6\K6-Beginner-01>k6 run k6-script-01.js

         /\      Grafana   /‾‾/
    /\  /  \     |\  __   /  /
   /  \/    \    | |/ /  /   ‾‾\
  /          \   |   (  |  (‾)  |
 / ‾‾‾‾‾‾‾‾‾ \  |_|\_\  \_____/

     execution: local
        script: k6-script-01.js
        output: -

     scenarios: (100.00%) 1 scenario, 1 max VUs, 10m30s max duration (incl. graceful stop):
              * default: 1 iterations for each of 1 VUs (maxDuration: 10m0s, gracefulStop: 30s)


  TOTAL RESULTS

    HTTP
    http_req_duration.......................................................: avg=223.44ms min=223.44ms med=223.44ms max=223.44ms p(90)=223.44ms p(95)=223.44ms
      { expected_response:true }............................................: avg=223.44ms min=223.44ms med=223.44ms max=223.44ms p(90)=223.44ms p(95)=223.44ms
    http_req_failed.........................................................: 0.00%  0 out of 1
    http_reqs...............................................................: 1      1.434343/s

    EXECUTION
    iteration_duration......................................................: avg=697.18ms min=697.18ms med=697.18ms max=697.18ms p(90)=697.18ms p(95)=697.18ms
    iterations..............................................................: 1      1.434343/s

    NETWORK
    data_received...........................................................: 6.8 kB 9.8 kB/s
    data_sent...............................................................: 543 B  779 B/s


running (00m00.7s), 0/1 VUs, 1 complete and 0 interrupted iterations
default ↳ [======================================] 1 VUs  00m00.7s/10m0s  1/1 iters, 1 per VU
```

****

VUs: In K6 means: ***Virtual Users*** **that are visiting the websites**. Like *Robots OR Scrapers* That Visits The Site.

****

Latency: In Simple Terms; The ***Time Between*** Asking Sites OR APIs To Get My Data And The Time To Response. (The ***Total Time*** Of *Request And Response*)

****

Throughput: How Many Our Servers, Sites, …etc Can ***Handle Requests During Period Of Times***.

- **Ex**: Our Sites Can Handle 100-Requests During 1-Hour.
- We Can Call It The **Website Capacity**.

****

Iterations In K6: *Repeating The Same Action Multiple Times* To Check Our Site Response Measures (**Latency**, **Request Time**, **Response Time**, **Exceptions**, **Failures**).

****

To override the default options of k6:

```
export const options = {
    vus: 10,
    duration: "10s"
}
```

****

When the number of iteration, and number of request, not equal, that means we have extra requests for each vus

```
     http_req_failed.....................................................: 0.00%  0 out of 156
     http_reqs...........................................................: 156    29.070511/s

     EXECUTION
     iteration_duration..................................................: avg=330.17ms min=166.51ms med=263.51ms max=1.2s     p(90)=431
.43ms p(95)=1.16s
     iterations..........................................................: 156    29.070511/s
     vus.................................................................: 10     min=10       max=10
     vus_max.............................................................: 10     min=10       max=10
```

****

If there is time, and vus finish from their jobs that specify in the function, then k6 will start new iteration for vus.

Note: in this way may we have delay between iteration and to handle it we need sleep-function.

```
export default function() {
    http.get('https://quickpizza.grafana.com/')
}
```

****

The sleep function make the vu sleep between the requests of our previous function *(delay between requests of each vu)*

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

export default function() {
    http.get('https://quickpizza.grafana.com/')
    sleep(1)
}
```

****

After set the sleep function

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS    POSTMAN CONSOLE                cmd

     HTTP
     http_req_duration.......................................................: avg=250.03ms min=169.11ms med=247.91ms max=624.73ms p(90)=335
.71ms p(95)=346.02ms
       { expected_response:true }............................................: avg=250.03ms min=169.11ms med=247.91ms max=624.73ms p(90)=335
.71ms p(95)=346.02ms
     http_req_failed.........................................................: 0.00%  0 out of 40
     http_reqs...............................................................: 40     6.771836/s

     EXECUTION
     iteration_duration......................................................: avg=1.39s    min=1.17s    med=1.25s    max=1.93s    p(90)=1.8
5s    p(95)=1.92s
     iterations..............................................................: 40     6.771836/s
     vus.....................................................................: 10     min=10      max=10
     vus_max.................................................................: 10     min=10      max=10

     NETWORK
     data_received...........................................................: 154 kB 26 kB/s
     data_sent...............................................................: 7.1 kB 1.2 kB/s      Activate Windows
```

****

The http_req_duration means: the total time of our request.

****

The p means percent; that means p(95) that 95% of request are faster *p(95)=346.02ms (The value means the biggest one, so the vu faster)*

****

```
  TOTAL RESULTS

     HTTP
     http_req_duration.......................................................: avg=250.03ms min=169.11ms med=247.91ms max=624.73ms p(90)=335
.71ms p(95)=346.02ms
       { expected_response:true }............................................: avg=250.03ms min=169.11ms med=247.91ms max=624.73ms p(90)=335
.71ms p(95)=346.02ms
     http_req_failed.........................................................: 0.00%  0 out of 40
     http_reqs...............................................................: 40     6.771836/s
```

****

Service-Level Objective (SLO)

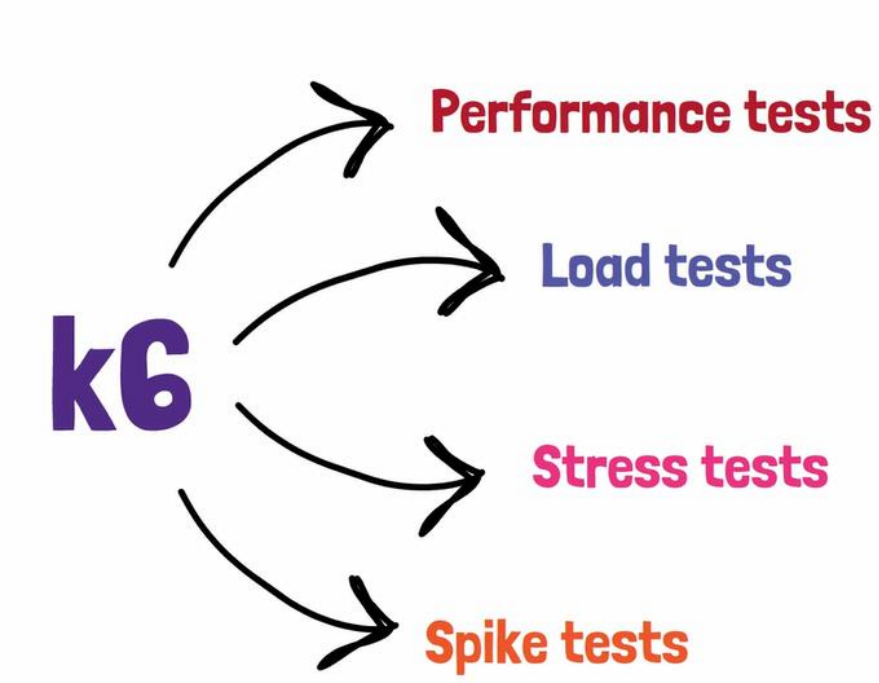
Availability:

The application will be available 99.8% of the time

Response Time:

90% of responses are within 0.5 seconds of receiving a request
95% of responses are within 0.9 seconds of receiving a request
99% of responses are within 2.5 seconds of receiving a request

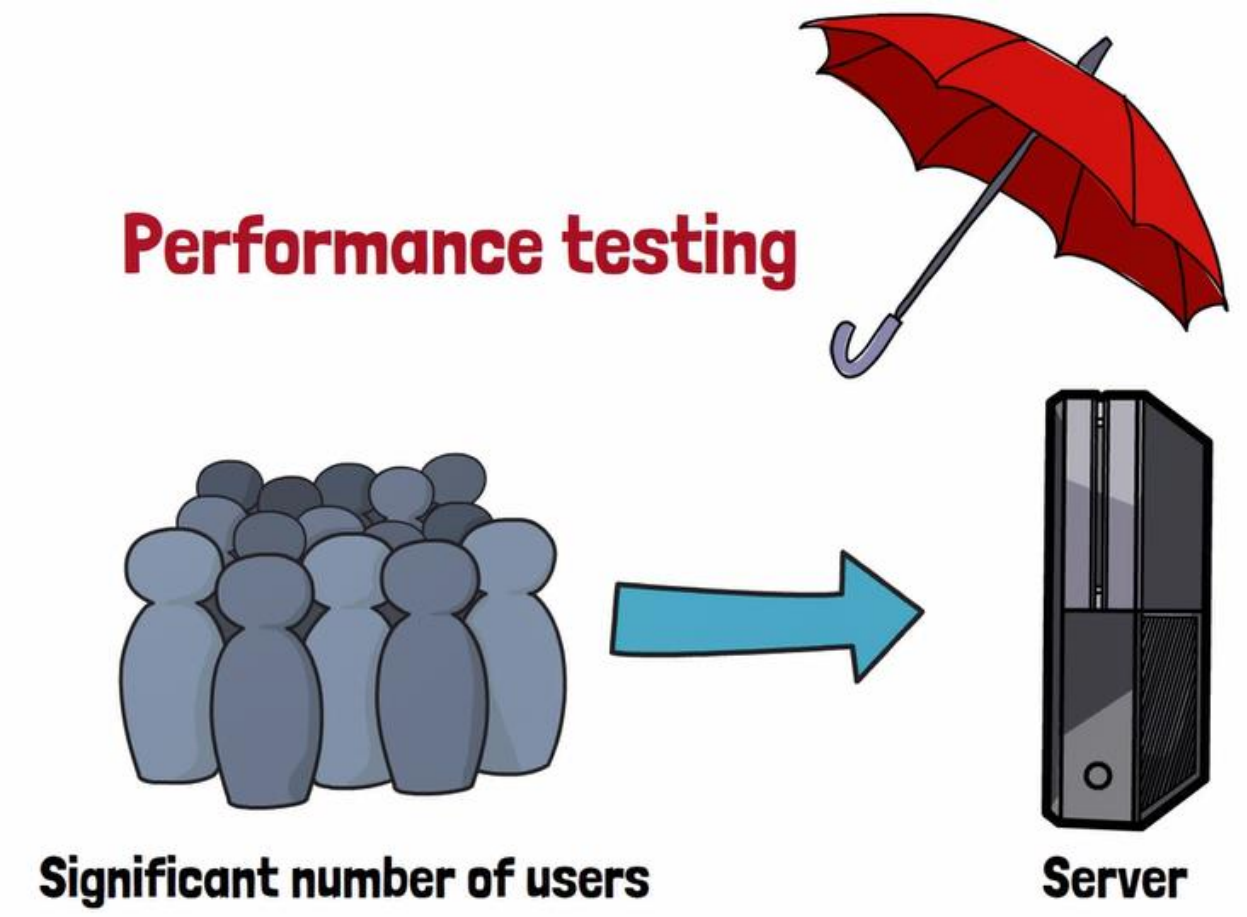
****



****

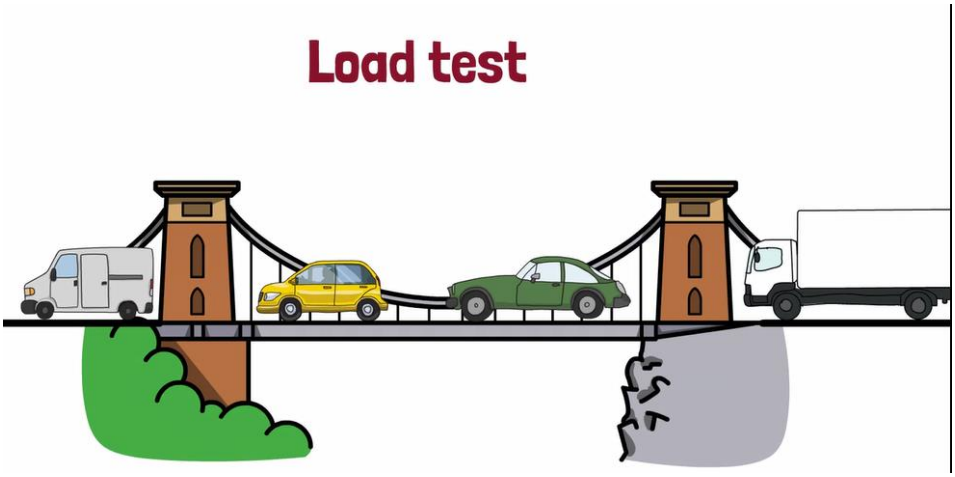
The number of users that want to use our service/site/app at once

Performance testing is the umbrella for all tests.



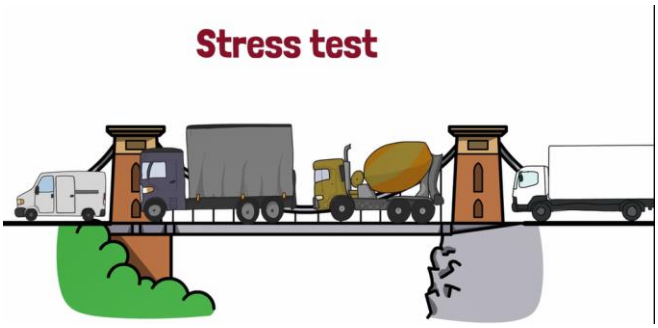
****

Load Testing: Simulate number of users access our site in different times. (Not at once)



****

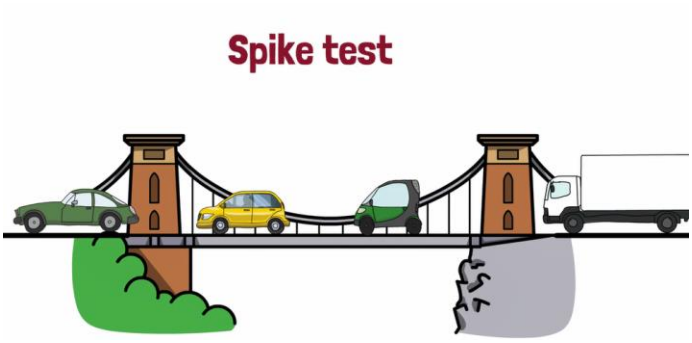
Stress Test: push the system to the limit, simulate high traffic at the same time or during different frames of time.



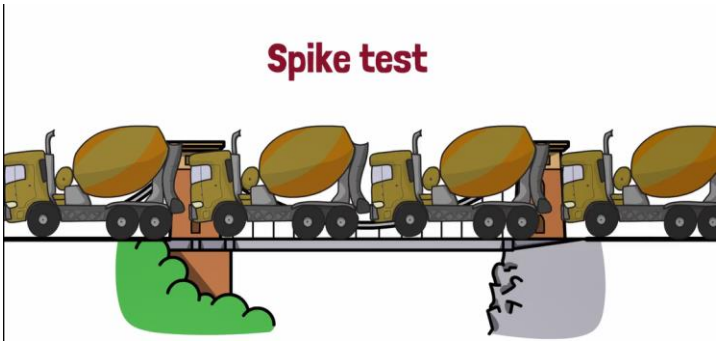
****

Spike Test: at any time we change the number of requests to the server (May increase or decrease the number of requests to server at the same time).

- *Low traffic*

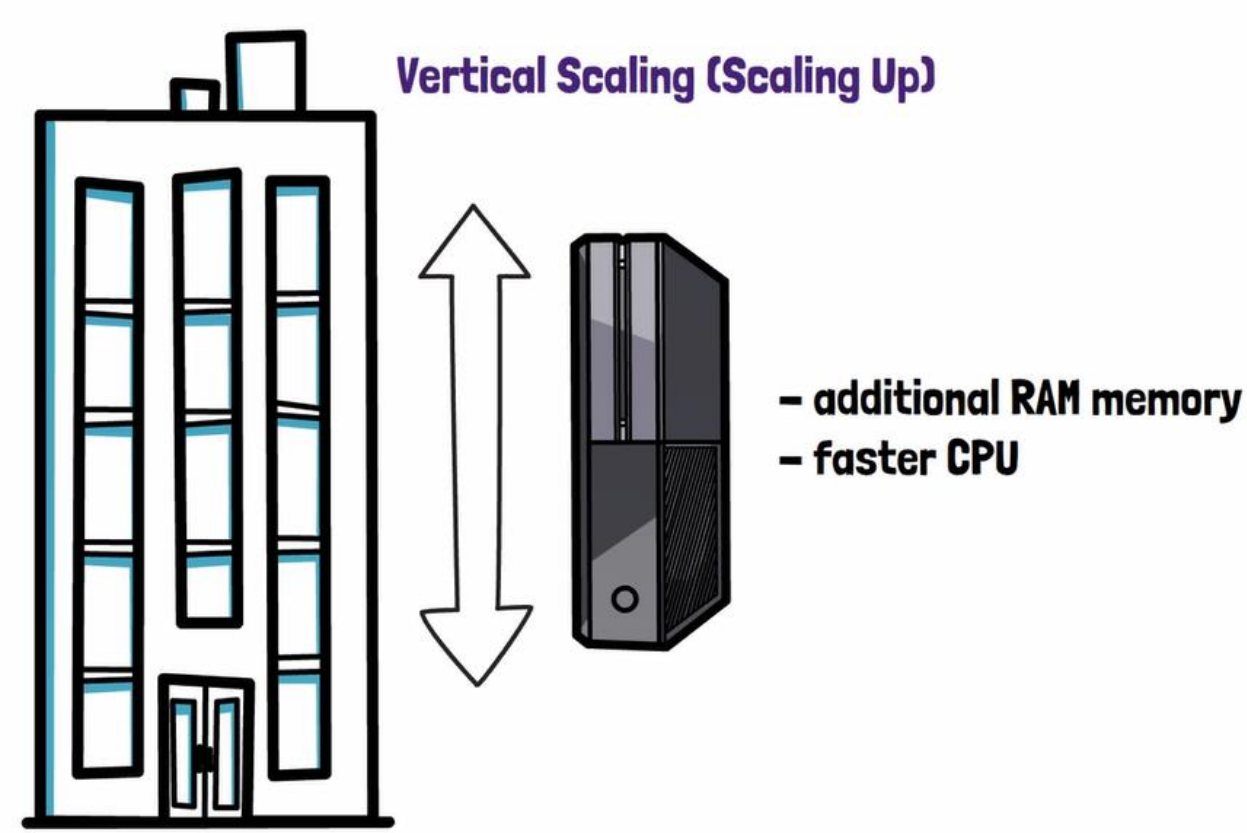


- *High traffic*



****

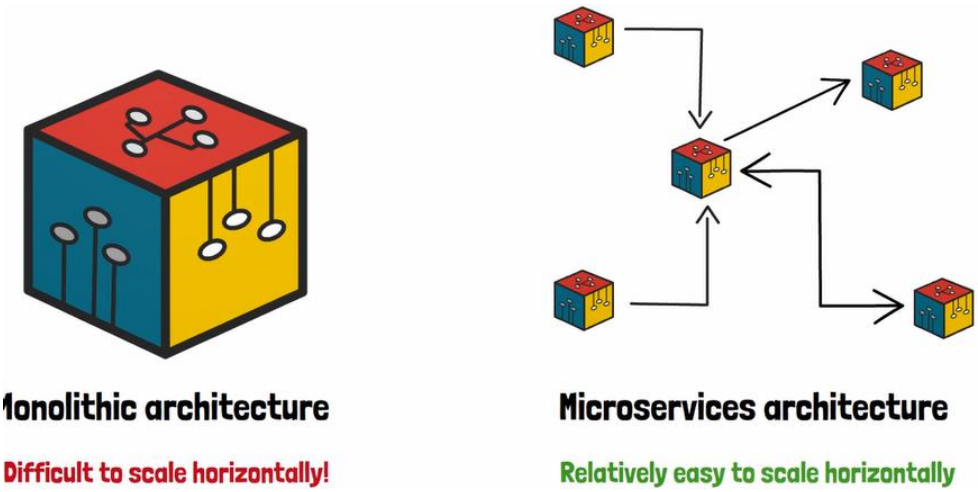
May Cause Some Core OR Ram To Be Idle At All Time (No Working).



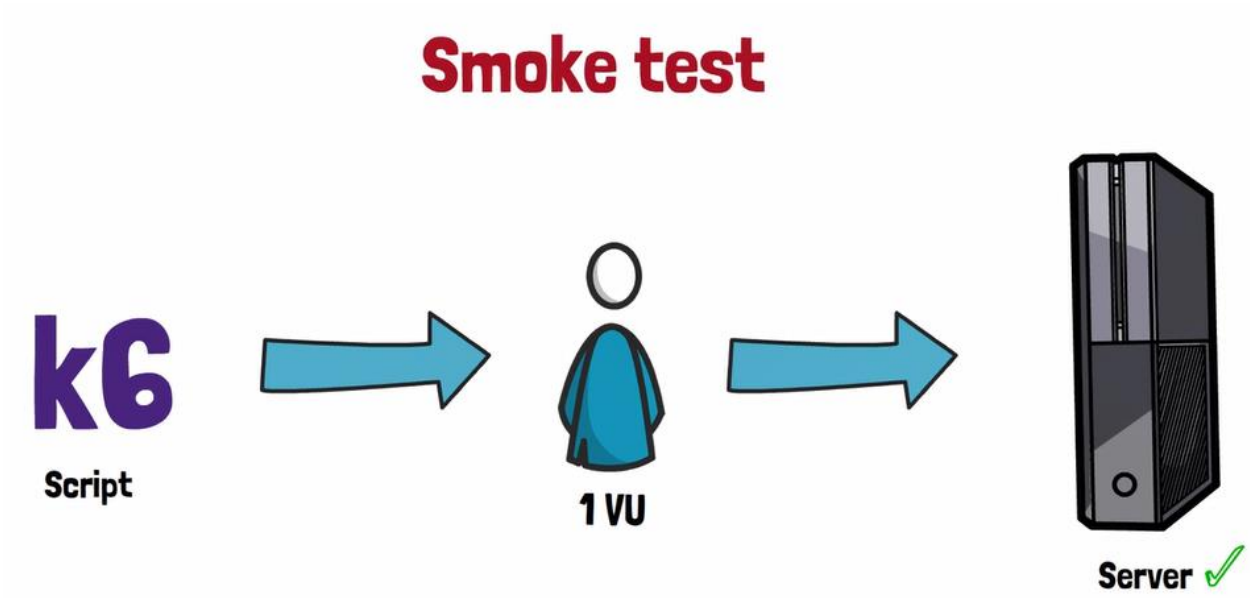
****

Cloud computing services: make it simple to achieve.

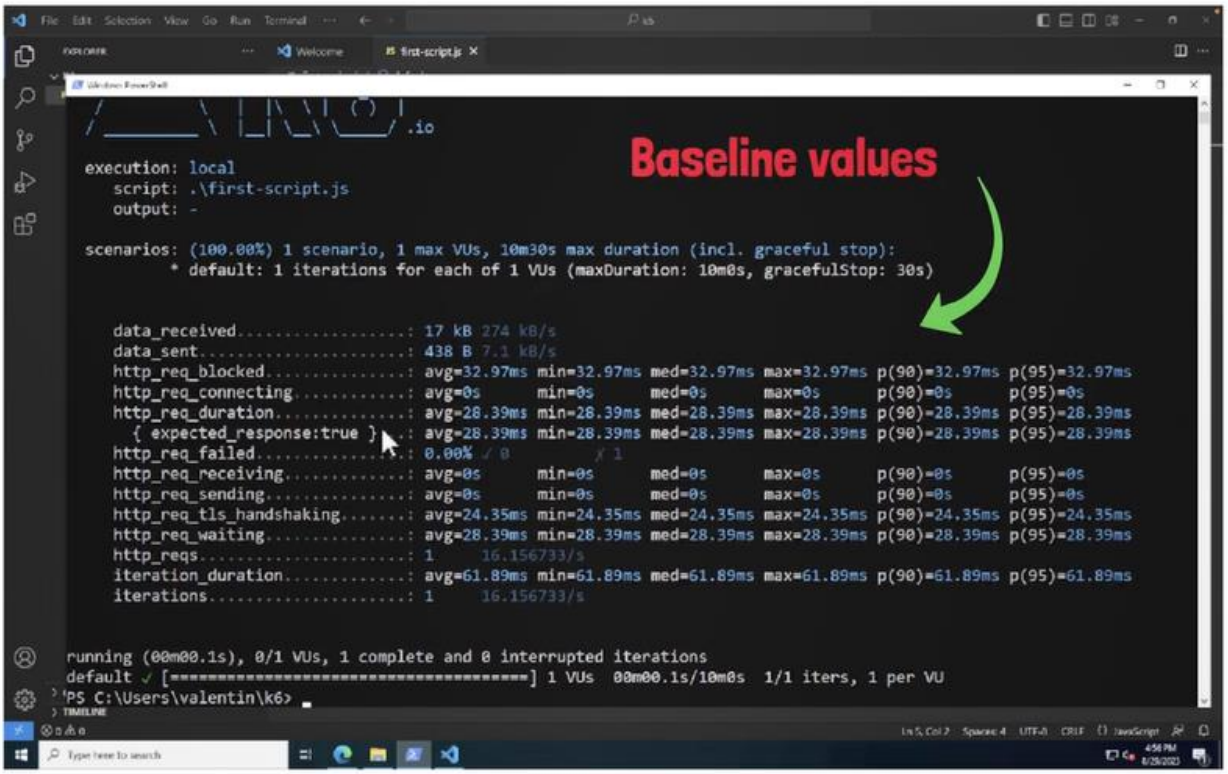
****


Monolithic architecture
Difficult to scale horizontally!
Microservices architecture
Relatively easy to scale horizontally

****


Smoke test
k6
Script
1 VU
Server

****

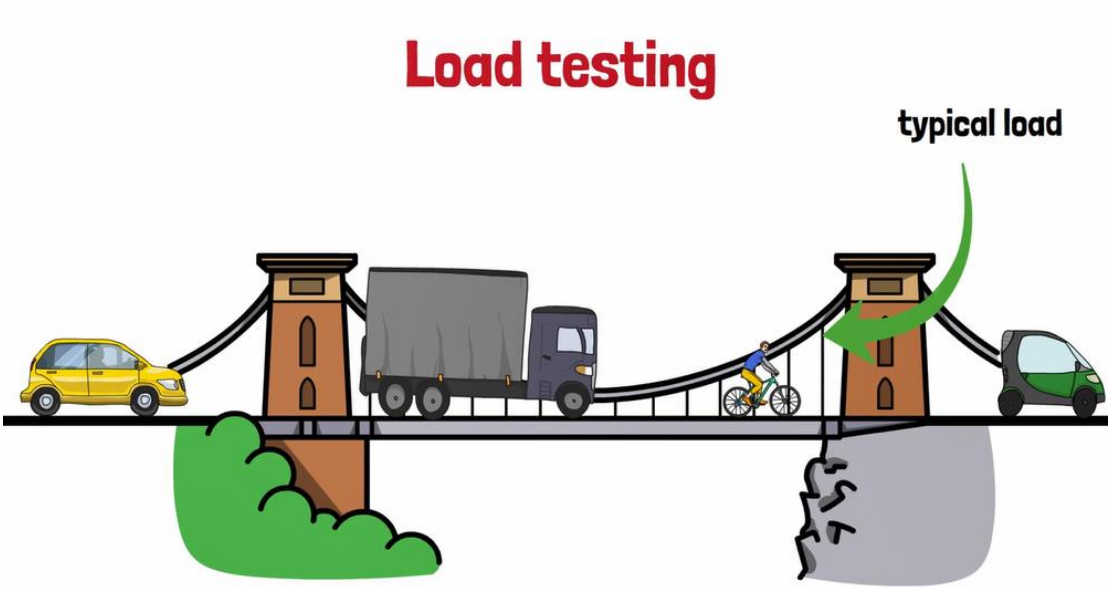


****

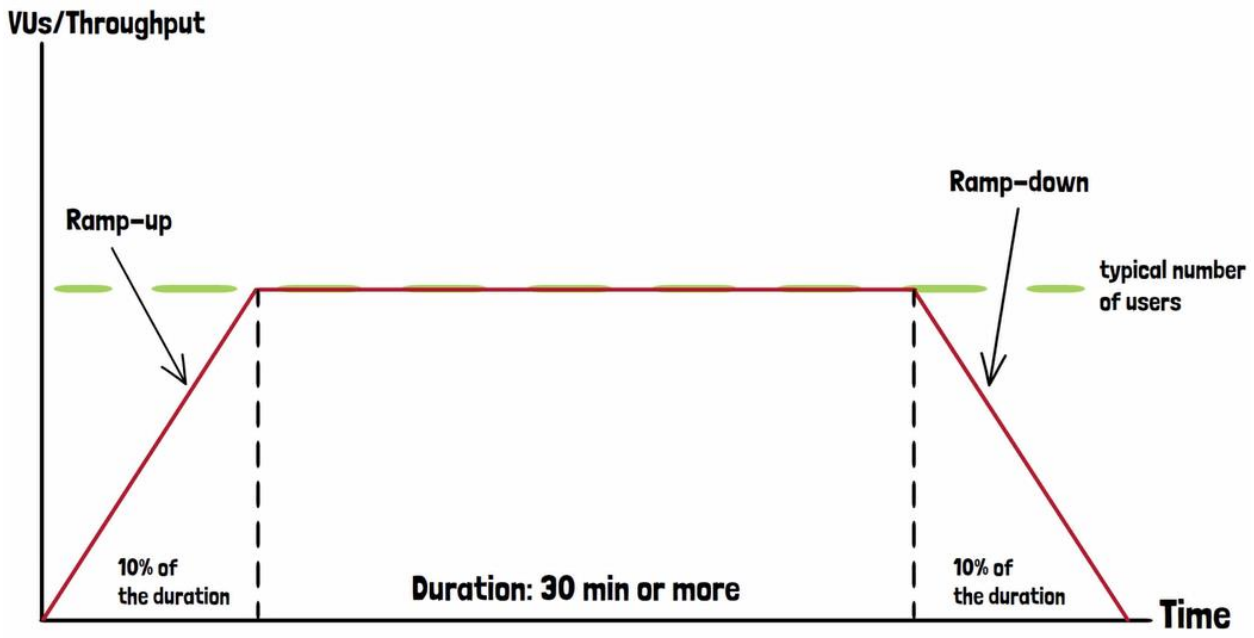
We can use *empirical data* if we have to number of users/throughput of requests during load test.

Note 1: We may run the tests only for few minutes.

Note 2: We can also increase the number of uses during times/tests.



****


VUs/Throughput
Ramp-up
Ramp-down
typical number
of users
10% of
the duration
Duration: 30 min or more
10% of
the duration
Time