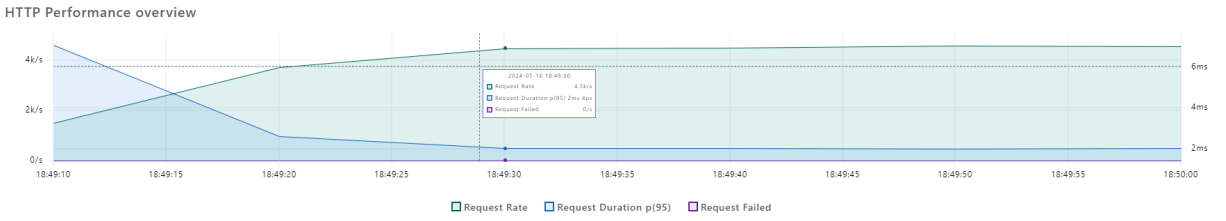
Memoria Práctica 7

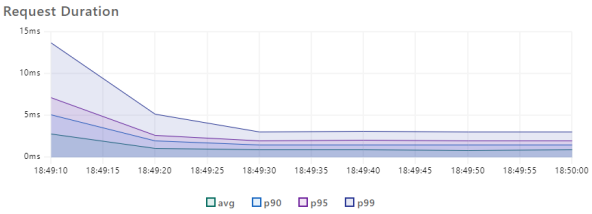
Mantenimiento y Pruebas del Software

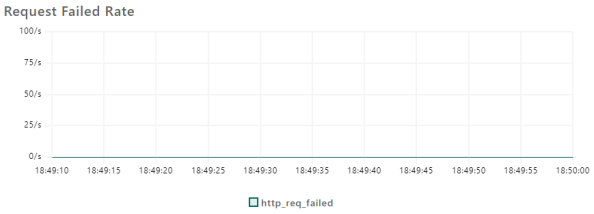
Por Guillermo Alejandro Westerhof Rodríguez y Pedro Rueda Cabrera

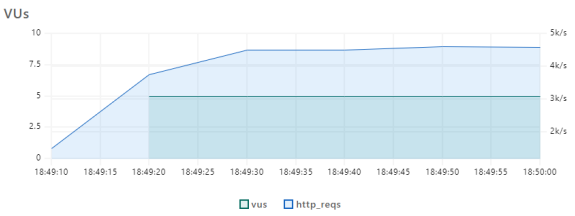
1. Smoke test:

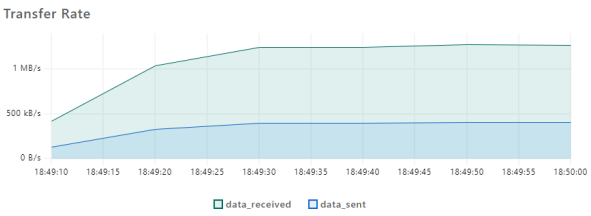
```
JS smoke.js > [] options > thresholds
1  import http from 'k6/http';
2  import { check } from 'k6';
3
4  export default function () {
5      let res = http.get('http://localhost:8080/medico/1');
6      check(res, {
7          'status was 200': (r) => r.status === 200,//Asegurar que las respuestas tengan código 200
8      });
9
10 }
11
12 export const options = {
13     vus: 5, // 5 usuarios virtuales
14     duration: '1m', // Duración de 1 minuto
15     thresholds: {
16         // El promedio de las solicitudes debe ser menor a 100 ms
17         http_req_duration: [{ threshold: 'avg<100', abortOnFail: true }],
18         // Asegurar que el 100% de las respuestas tengan código 200
19         checks: [{ threshold: 'rate==1', abortOnFail: true }],
20         // No debe haber peticiones fallidas
21         http_req_failed: [{ threshold: 'rate==0', abortOnFail: true }]
22     },
```











Summary

This chapter provides a summary of the test run metrics. The tables contains the aggregated values of the metrics for the entire test run.

Trends

metric	avg	max	med	min	p90	p95	p99
http_req_blocked	18µs	35ms	0ms	0ms	0ms	0ms	640µs
http_req_connecting	10µs	14ms	0ms	0ms	0ms	0ms	0ms
http_req_duration	1ms	86ms	760µs	0ms	1ms	2ms	5ms
http_req_receiving	157µs	80ms	0ms	0ms	537µs	866µs	1ms
http_req_sending	27µs	44ms	0ms	0ms	0ms	0ms	869µs
http_req_tls_handshaking	0ms	0ms	0ms	0ms	0ms	0ms	0ms
http_req_waiting	840µs	70ms	572µs	0ms	1ms	2ms	4ms
iteration_duration	1ms	87ms	997µs	0ms	2ms	2ms	5ms

Counters

metric	count	rate
data_received	65 MB	1.07 MB/s
data_sent	20.6 MB	341 kB/s
http_reqs	234.5k	3.87k/s
iterations	234.5k	3.87k/s

Rates

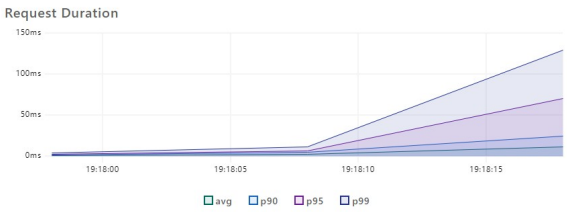
metric	rate
checks	1/s
http_req_failed	0/s

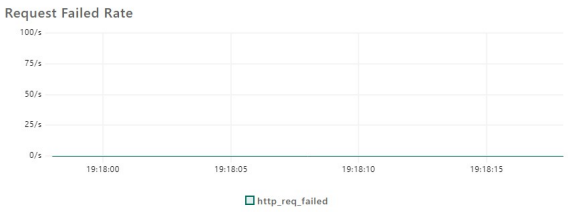
Gauges

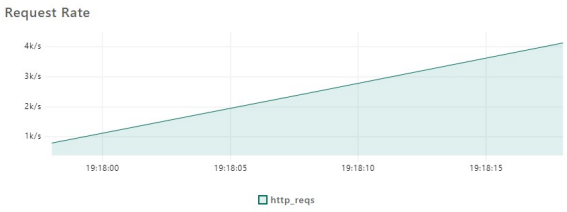
metric	value
vus	3
vus_max	5

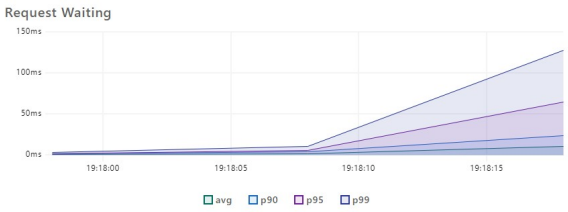
2. Breakpoint test(vus máximos :1015):

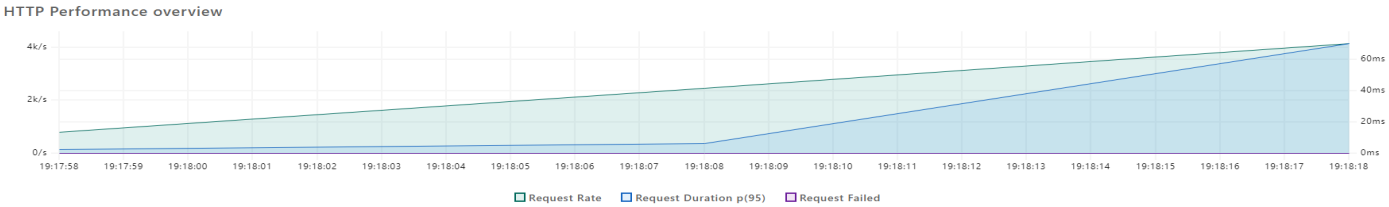
```
JS breakpoint.js > options > thresholds > http_req_failed > threshold
1   import http from 'k6/http';
2   import { check } from 'k6';
3
4   export default function () {
5       let res = http.get('http://localhost:8080/medico/1');
6       check(res, {
7           'status was 200': (r) => r.status === 200,
8       });
9   }
10
11
12  export const options = {
13      scenarios: {
14          breakpoint: {
15              executor: 'ramping-arrival-rate',
16              preAllocatedVUs: 1000, // VUs alocados inicialmente
17              maxVUs: 100000, // VUs máximo (al menos 100,000)
18              stages: [
19                  { duration: '10m', target: 100000 }, // Aumentar hasta 100,000 usuarios en 10 minutos
20              ],
21          },
22      },
23      thresholds: {
24          http_req_failed: [{ threshold: 'rate<=0.01', abortOnFail: true }], // Abortar si más del 5% de las solicitudes fallan
25      },
```

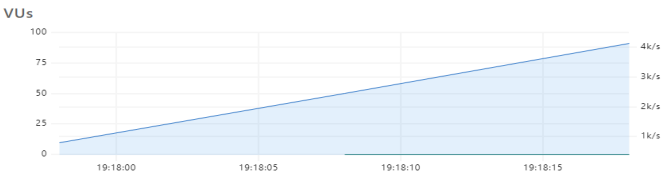


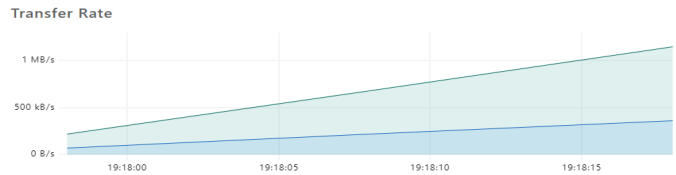












Trends

metric	avg	max	med	min	p90	p95	p99
http_req_blocked	88µs	377ms	0ms	0ms	0ms	0ms	998µs
http_req_connecting	35µs	77ms	0ms	0ms	0ms	0ms	506µs
http_req_duration	15ms	2s	2ms	0ms	30ms	72ms	176ms
http_req_receiving	1ms	364ms	0ms	0ms	998µs	1ms	10ms
http_req_sending	373µs	374ms	0ms	0ms	0ms	989µs	2ms
http_req_tls_handshaking	0ms	0ms	0ms	0ms	0ms	0ms	0ms
http_req_waiting	13ms	2s	1ms	0ms	28ms	66ms	136ms
iteration_duration	19ms	2s	2ms	0ms	39ms	84ms	211ms

Counters

metric	count	rate
data_received	29.2 MB	748 kB/s
data_sent	9.28 MB	238 kB/s
dropped_iterations	398	10.19/s
http_reqs	106.5k	2.73k/s
iterations	105.9k	2.71k/s

Rates

metric	rate
checks	1/s
http_req_failed	0.01/s

Gauges

metric	value
vus	737
vus_max	1k

Breakpoint con stages(vus máximos :1000)

breakpoint_stages.js > default

```js
import http from 'k6/http';
import { check } from 'k6';

export const options = {
    stages: [
        { duration: '10m', target: 100000 }, // Aumentar hasta 100,000 usuarios en 10 minutos
    ],
    thresholds: {
        http_req_failed: [{ threshold: 'rate<=0.01', abortOnFail: true }], // Abortar si más del 1% de las solicitudes fallan
    },
};

export default function () {
    let res = http.get('http://localhost:8080/medico/1');
    check(res, {
        'status was 200': (r) => r.status === 200,
    });
}
```

Summary

This chapter provides a summary of the test run metrics. The tables contains the aggregated values of the metrics for the entire test run.

Trends

metric	avg	max	med	min	p90	p95	p99
http_req_blocked	106ms	4s	0ms	0ms	7ms	709ms	3s
http_req_connecting	20ms	2s	0ms	0ms	2ms	14ms	1s
http_req_duration	384ms	3s	0ms	0ms	2s	3s	3s
http_req_receiving	299µs	129ms	0ms	0ms	0ms	0ms	997µs
http_req_sending	948µs	43ms	0ms	0ms	0ms	1ms	30ms
http_req_tls_handshaking	0ms	0ms	0ms	0ms	0ms	0ms	0ms
http_req_waiting	382ms	3s	0ms	0ms	2s	3s	3s
iteration_duration	3s	6s	4s	6ms	4s	5s	5s

Counters

metric	count	rate
data_received	140 kB	1.85 kB/s
data_sent	59 kB	777 B/s
http_reqs	1.4k	18.6/s
iterations	405	5.33/s

Rates

metric	rate
checks	0.53/s
http_req_failed	0.85/s

Gauges

metric	value
vus	1.4k
vus_max	100k

Con Executor:

- Permite un control preciso de la tasa de llegada de usuarios virtuales (VUs).
- Más flexible para definir cómo se incrementa o disminuye la carga.
- Necesita una asignación inicial de VUs

Sin Executor (usando stages)

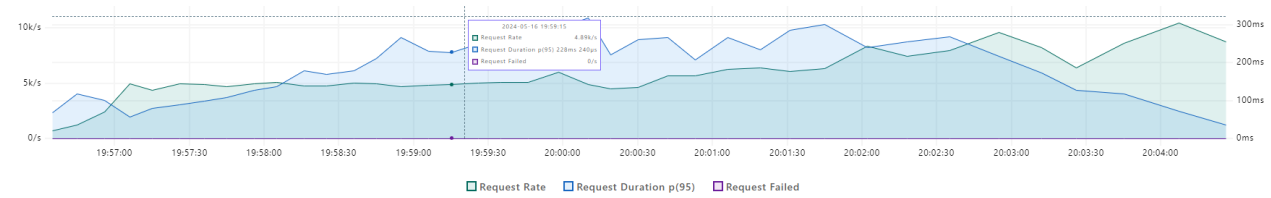
- Control simplificado. Se puede definir exactamente cómo aumenta y disminuye (en los stages)
- Mas sencillo y facil de leer.
- No necesita una asignación inicial de VUs

Conclusiones: Con una "rampa" podriamos llegar a mas VUs.

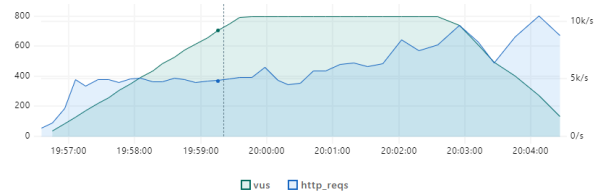
3. Stress test:

```
JS stress.js > default
1   import http from 'k6/http';
2   import { check } from 'k6';
3
4   export const options = {
5       stages: [
6           { duration: '3m', target: 800 }, // Subimos a 800 VUs en 3 minutos
7           { duration: '3m', target: 800 }, // Mantenemos 800 VUs durante 3 minutos
8           { duration: '2m', target: 0 }, // Bajamos a 0 VUs en 2 minutos
9       ],
10      thresholds: {
11          http_req_failed: [{ threshold: 'rate<=0.01', abortOnFail: true }], // Las peticiones fallidas deben ser menores al 1%
12          http_req_duration: [{ threshold: 'avg<1000', abortOnFail: true }], // El promedio de la duración de las peticiones debe ser menor a 1000 ms
13      },
14  };
15
16  export default function () {
17      let res = http.get('http://localhost:8080/medico/1');
18      check(res, {
19          'status was 200': (r) => r.status === 200,
20      });
21  }
```

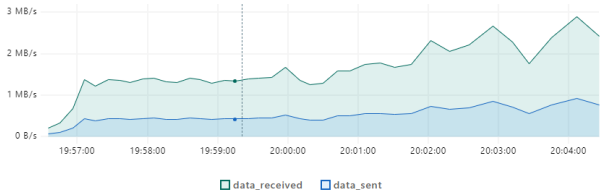
HTTP Performance overview



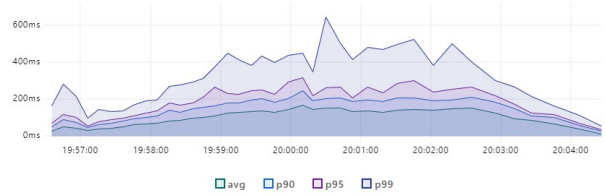
VUs



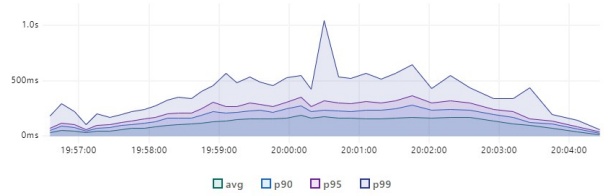
Transfer Rate



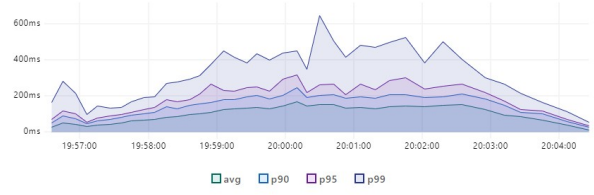
HTTP Request Duration



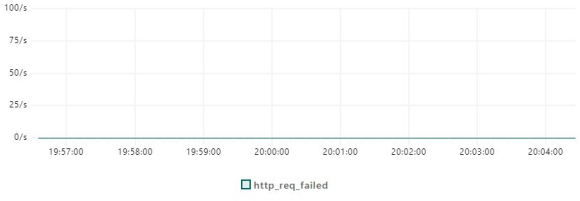
Iteration Duration



Request Duration



Request Failed Rate



Request Rate

Request Waiting

Trends

metric	avg	max	med	min	p90	p95	p99
http_req_blocked	903µs	1s	0ms	0ms	0ms	0ms	4ms
http_req_connecting	869µs	694ms	0ms	0ms	0ms	0ms	998µs
http_req_duration	103ms	1s	95ms	0ms	176ms	210ms	371ms
http_req_receiving	3ms	1s	0ms	0ms	997µs	1ms	138ms
http_req_sending	279µs	1s	0ms	0ms	0ms	0ms	1ms
http_req_tls_handshaking	0ms	0ms	0ms	0ms	0ms	0ms	0ms
http_req_waiting	99ms	1s	94ms	0ms	172ms	200ms	309ms
iteration_duration	118ms	2s	111ms	0ms	198ms	253ms	428ms

Counters

metric	count	rate
data_received	612 MB	1.26 MB/s
data_sent	194 MB	401 kB/s
http_reqs	2.2m	4.56k/s
iterations	2.2m	4.56k/s

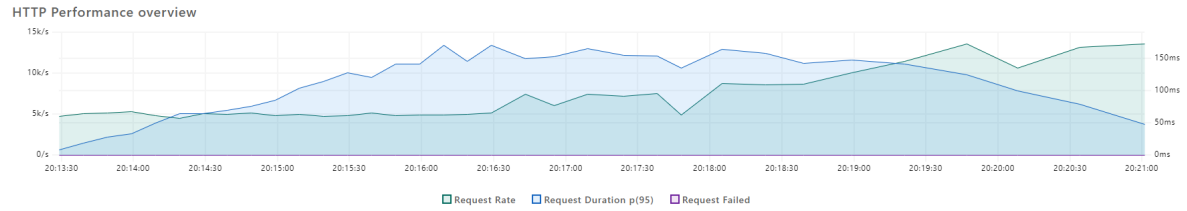
Rates

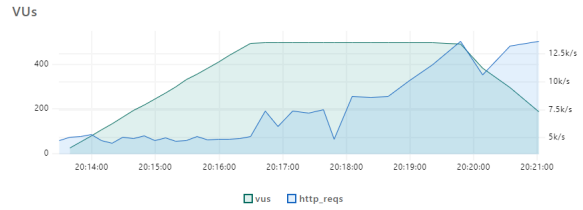
metric	rate
checks	1/s
http_req_failed	0/s

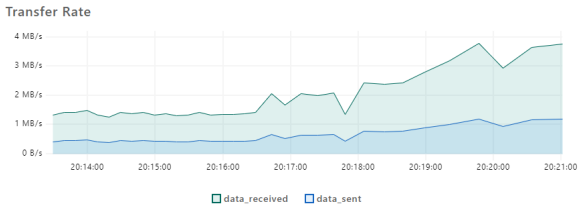
Gauges

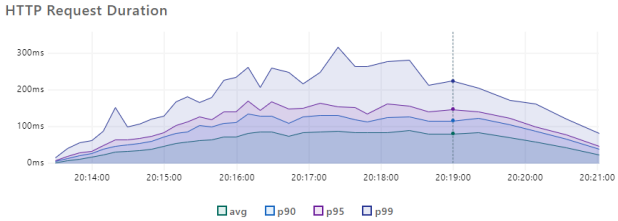
metric	value
vus	1
vus_max	800

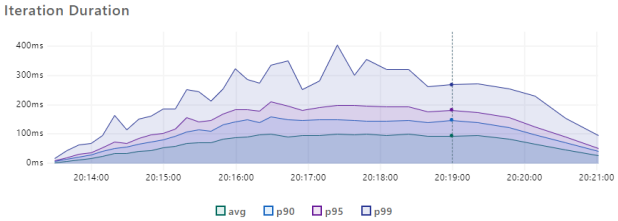
4. Average load test:

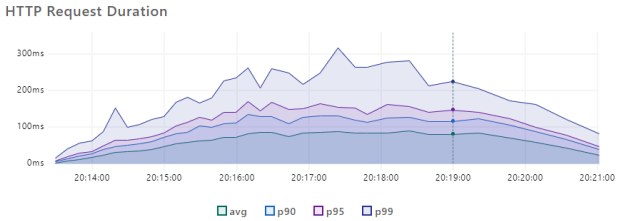


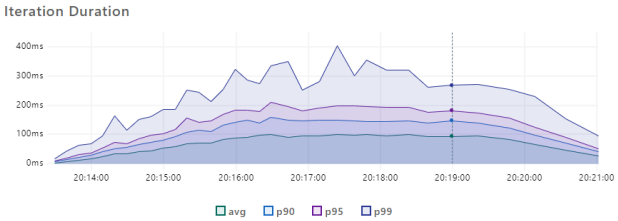












Trends

metric	avg	max	med	min	p90	p95	p99
http_req_blocked	527µs	752ms	0ms	0ms	0ms	0ms	2ms
http_req_connecting	500µs	509ms	0ms	0ms	0ms	0ms	0ms
http_req_duration	59ms	1s	55ms	0ms	106ms	127ms	203ms
http_req_receiving	1ms	817ms	0ms	0ms	997µs	1ms	51ms
http_req_sending	187µs	1s	0ms	0ms	0ms	0ms	1ms
http_req_tls_handshaking	0ms	0ms	0ms	0ms	0ms	0ms	0ms
http_req_waiting	57ms	984ms	54ms	0ms	103ms	122ms	189ms
iteration_duration	67ms	1s	62ms	0ms	120ms	154ms	258ms

Counters

metric	count	rate
data_received	670 MB	1.38 MB/s
data_sent	213 MB	439 kB/s
http_reqs	2.4m	4.99k/s
iterations	2.4m	4.99k/s

Rates

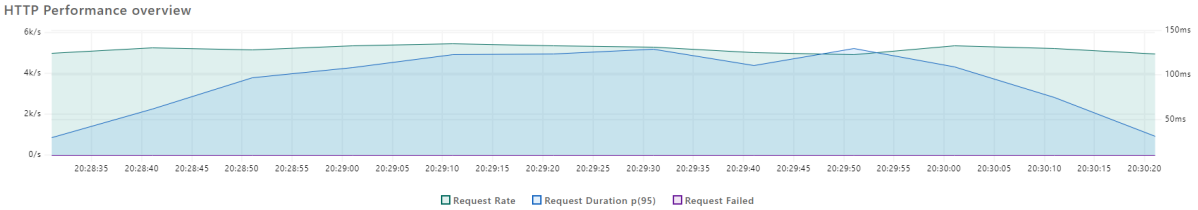
metric	rate
checks	1/s
http_req_failed	0/s

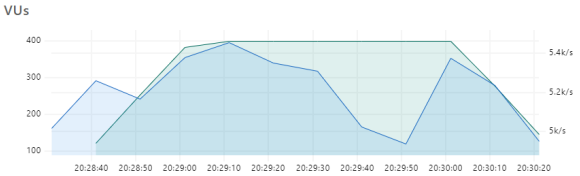
Gauges

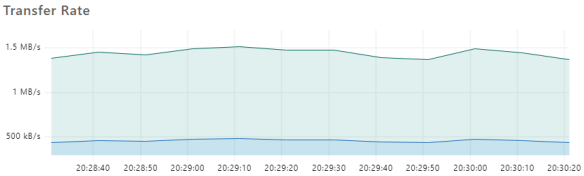
metric	value
vus	1
vus_max	500

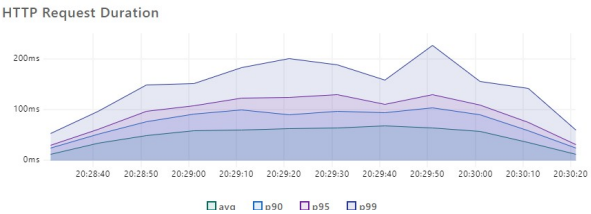
5. Spike test:

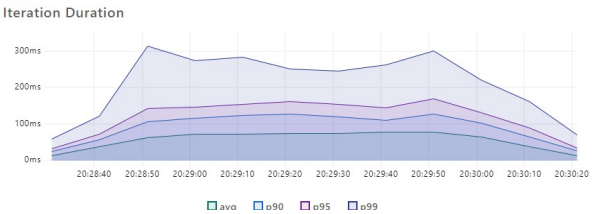
Si el sistema es capaz de gestionar sin ningun problema el 40% de la carga.

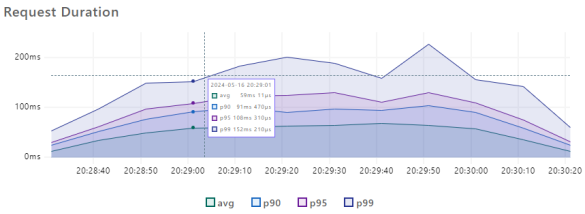


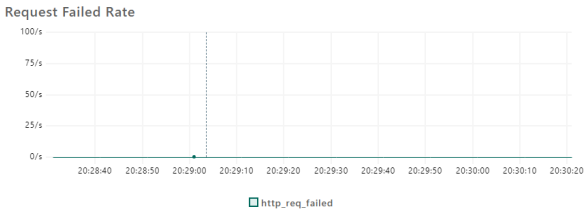


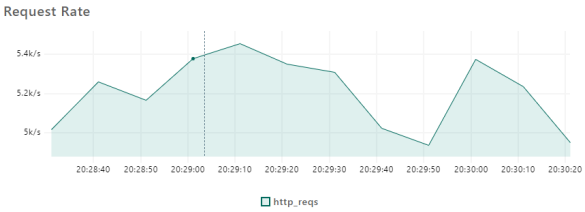


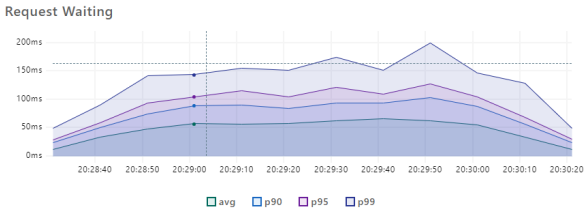












Trends

metric	avg	max	med	min	p90	p95	p99
http_req_blocked	470µs	618ms	0ms	0ms	0ms	0ms	3ms
http_req_connecting	429µs	618ms	0ms	0ms	0ms	0ms	997µs
http_req_duration	48ms	1s	43ms	0ms	84ms	104ms	163ms
http_req_receiving	1ms	1s	0ms	0ms	997µs	1ms	48ms
http_req_sending	219µs	794ms	0ms	0ms	0ms	510µs	1ms
http_req_tls_handshaking	0ms	0ms	0ms	0ms	0ms	0ms	0ms
http_req_waiting	46ms	540ms	43ms	0ms	81ms	99ms	149ms
iteration_duration	57ms	1s	51ms	0ms	100ms	134ms	233ms

Counters

metric	count	rate
data_received	173 MB	1.43 MB/s
data_sent	55 MB	453 kB/s
http_reqs	624.7k	5.15k/s
iterations	624.7k	5.15k/s

Rates

metric	rate
checks	1/s
http_req_failed	0/s

Gauges

metric	value
vus	1
vus_max	400