Assume you are expert in k6 for performance testing and know very well in understanding out then provide a good guide for a beginner to understand very well about " execution: local script: jsfile.js output: -

scenarios: (100.00%) 1 scenario, 10 max VUs, 40s max duration (incl. graceful stop):

* default: 10 looping VUs for 10s (gracefulStop: 30s)

TOTAL RESULTS

checks_total.....: 31 0.774965/s

checks_succeeded.....: 100.00% 31 out of 31

checks_failed.....: 0.00% 0 out of 31

✓ Status is 200

HTTP

http_req_duration.....: avg=1.04s min=225.34ms med=675.72ms max=3.94s p(90)=2.4s p(95)=2.69s

{ expected_response:true
}.....: avg=1.04s
min=225.34ms med=675.72ms max=3.94s
p(90)=2.4s p(95)=2.69s

http_req_failed.....:
0.00% 0 out of 31

http_reqs.....: 31
0.774965/s

EXECUTION

iteration_duration.....:
avg=2.94s min=725.98ms med=1.17s max=14.77s
p(90)=6.66s p(95)=10.15s

iterations.....: 31
0.774965/s

vus.....: 2
min=2 max=10

vus_max.....:
10 min=10 max=10

NETWORK

data_received.....:
396 kB 9.9 kB/s

data_sent.....:
6.5 kB 163 B/s

running (40.0s), 00/10 VUs, 31 complete and 2 interrupted iterations
 default ✓
[==]
10 VUs 10s "

Absolutely! Let’s break this down like a true beginner-friendly guide to K6 performance testing — focusing on your shared output. We'll cover:

1. **What is K6?**
2. **Understanding the test configuration (`execution`, `scenarios`)**
3. **Decoding the test result summary (`TOTAL RESULTS`)**
4. **What to look for in performance testing results**
5. **How to improve based on this**

🧪 1. What is K6?

K6 is an open-source performance testing tool written in Go, but you write your scripts in JavaScript (`.js`). It’s popular for load testing APIs, websites, and services. The tool helps you understand how your system performs under different user loads.

⚙️ 2. Understanding Test Configuration

You provided this configuration part:

execution: local

script: jsfile.js
output: -

scenarios: (100.00%) 1 scenario, 10 max VUs, 40s max duration (incl. graceful stop):
        * default: 10 looping VUs for 10s (gracefulStop: 30s)

Explanation:

Term	Meaning
`execution: local`	Test was run locally (on your machine), not cloud/distributed.
`script: jsfile.js`	The test logic is in this JavaScript file.
`output: -`	No output file was written (default is console).
`scenarios:`	Load pattern of the test.
`default: 10 looping VUs for 10s`	You ran 10 virtual users (VUs), all executing the script simultaneously in a loop for 10 seconds.
`gracefulStop: 30s`	After the 10s, any VUs still running get up to 30s to finish cleanly.
`max duration: 40s`	Total time test will run including the graceful stop.

📊 3. Understanding the TOTAL RESULTS Section

Let's decode the output step-by-step.

✅ CHECKS

Checks are **assertions** you make in your script.

check(res, { "Status is 200": (r) => r.status === 200 });

Result:

checks_total.....: 31
checks_succeeded.....: 100.00%

checks_failed.....: 0.00%
✓ Status is 200

- `31 checks` ran — these were likely from each HTTP request.
- All succeeded ✅

🌐 HTTP METRICS

http_req_duration: avg=1.04s min=225.34ms med=675.72ms max=3.94s p(90)=2.4s p(95)=2.69s

Term	Meaning
`avg`	Average response time: 1.04 seconds
`min`	Fastest request took 225.34 ms
`med`	Median (middle) was 675.72 ms
`max`	One request took nearly 4 seconds (slow!)
`p(90)`	90% of requests were under 2.4s
`p(95)`	95% of requests were under 2.69s

http_reqs

http_reqs: 31

- 31 HTTP requests were made.

⏱️ EXECUTION METRICS

iteration_duration: avg=2.94s min=725.98ms med=1.17s max=14.77s

iterations: 31
vus: 2 min=2 max=10
vus_max: 10

Metric	Meaning
`iteration_duration`	How long one VU took to run a full iteration of your script. Some took as long as 14 seconds!
`iterations`	Total of 31 iterations (each user-loop)
`vus`	You had between 2 and 10 active VUs at a time
`vus_max`	Maximum number of VUs allowed (10 in your config)

📡 NETWORK

data_received: 396 kB
data_sent: 6.5 kB

Just tracking how much data was exchanged.

📌 4. What to Look for as a Beginner

Checkpoint	Goal
✅ Checks	Should have high success rate (100% is great!)
📈 HTTP duration	Look for response time spikes (e.g. max = 3.94s is quite slow)
🔁 Iteration duration	If your script has loops, track how long each takes. 14.77s is high — optimize this
🧍 VUs	Know how many users your system can handle. This test used only 10 VUs — try increasing and observing changes
💡 Percentiles	p(90)/p(95) show worst-case performance — important for SLA/SLO planning

---

🛠️ 5. Tips to Improve and Practice

✅ Basic Improvements

- Use more VUs and duration to simulate real-world traffic.
- Add `thresholds` in your script to fail tests if performance degrades.

```
export let options = {
  thresholds: {
    http_req_duration: ['p(95)<1000'], // 95% requests should be <1s
  },
};
```

🔁 Test Script Sample (`jsfile.js`)

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

export const options = {
  vus: 10,
  duration: '10s',
};

export default function () {
  const res = http.get('https://example.com');
  check(res, {
    'Status is 200': (r) => r.status === 200,
  });
  sleep(1); // simulate user thinking time
}
```

🎓 Summary

Section	Key Takeaway
Config	10 VUs, 10s run time, graceful exit after
Results	31 successful iterations & requests

Performance	Most responses <2.4s, but some were slow (up to 3.9s)
Learning Path	Experiment with thresholds, load types, more VUs, and longer durations

---

If you’d like, I can create a **K6 practice project** for you step-by-step. Want to do that next?