Load Test Script

Follow Kind_Cluster Script first then do following steps :

7. Create the k6 spike-load script

Create s1_spike.js:

```
cat > s1_spike.js <<'EOF'

import http from 'k6/http';

import { check, sleep } from 'k6';

import { randomSeed } from 'k6';

randomSeed(42);

export const options = {

 summaryTrendStats: ['avg','min','med','p(90)','p(95)','max'],

 scenarios: {

  spike_all: {

   executor: 'ramping-arrival-rate',

   startRate: 10,          // warm start

   timeUnit: '1s',

   preAllocatedVUs: 200,

   maxVUs: 400,

   stages: [

    { target: 30,  duration: '60s' }, // warm up

    { target: 400, duration: '20s' }, // sudden spike

    { target: 400, duration: '2m'  }, // hold

    { target: 60,  duration: '60s' }, // ramp down

    { target: 0,   duration: '30s' }, // drain

   ],

  },
 },
 thresholds: {
   'http_req_failed': ['rate<0.02'],       // <2% failures
   'http_req_duration{app:sock}': ['p(95)<1000'],
   'http_req_duration{app:book}': ['p(95)<1000'],
   'http_req_duration{app:tea}':  ['p(95)<1500'],
 },
};
// Default to hostPort 80 on the kind VM
const BASE = __ENV.BASE_URL || 'http://127.0.0.1';

const TARGETS = [
 { path: '/sock',          tag: 'sock', weight: 0.55 },
 { path: '/book?u=normal', tag: 'book', weight: 0.20 },
 { path: '/book?u=test',   tag: 'book', weight: 0.05 },
 { path: '/tea',           tag: 'tea',  weight: 0.20 },
];
const totalW = TARGETS.reduce((a, b) => a + b.weight, 0);
function pick() {
 let r = Math.random() * totalW;
 for (const t of TARGETS) {
   r -= t.weight;
   if (r <= 0) return t;
 }
 return TARGETS[0];
}
```

```
export default function () {
  const t = pick();
  const res = http.get(`${BASE}${t.path}`, { tags: { app: t.tag } });
  check(res, { 'status 2xx/3xx': r => r.status >= 200 && r.status < 400 });
  sleep(Math.random() * 0.8);
}
EOF
```

Quick check:

```
ls -l s1_spike.js
head -20 s1_spike.js
```

---

8. Run the spike test

You can either rely on the default BASE (http://127.0.0.1) or set it explicitly.

```
export BASE_URL="http://localhost"
k6 run s1_spike.js
```

k6 will show:

- checks (success %)
- http_req_duration (avg, p95, etc.), also split by app tag
- http_req_failed rate

Start with lower numbers (e.g. reduce target peaks to ~150–200 req/s) if your VM is small, then increase until you find the “knee” where latency or error rates blow up.

Cluster_Load_script_updated

---

9. What to monitor during the spike

From the same VM:

kubectl top pods -A

kubectl -n istio-system logs -f deploy/istio-ingressgateway

kubectl -n sock-shop get pods -o wide

kubectl -n bookinfo get pods -o wide

kubectl -n teastore get pods -o wide

If you enabled the optional addons:

- **Kiali**: service graph, p95 latency, error rates.
- kubectl -n istio-system port-forward svc/kiali 20001:20001
- # then open http://localhost:20001
- **Grafana**: detailed Prometheus dashboards.
- kubectl -n istio-system port-forward svc/grafana 3000:3000
- # then open http://localhost:3000

You can also inspect traces in Jaeger (if installed) for long-tail requests during the spike.

---

10. Saving and comparing runs

Export results to JSON so you can compare before/after experiments (e.g. with and without HPA, different resource limits, etc.):

k6 run --summary-export spike_summary.json s1_spike.js

Repeat with different configs and compare p95 latency, error rates, and throughput over time.