Project 02 — Multi-hop Service on Mini-K8s: Cgroups/QoS and End-to-End Tail Latency

Build a small **multi-hop** service and run it on a **local mini-Kubernetes** setup inside your Ubuntu VM. Then make **p99 tail latency** go bad in controlled ways, explain the critical path with OS/K8s evidence, and demonstrate mitigations.

Team size **1–2**, duration **10 weeks**, VM-friendly.

> ***Baseline expectation (this course): MS level.***

Target stack

- **Default:** kind (Kubernetes-in-Docker) on the VM
- Alternative: k3s (if you prefer)

Your report must specify which you used and your VM resource budget.

System you build (minimum)

A 3–4 component deployment, for example:

- `gateway` (HTTP)
- `service-a` (business logic)
- `service-b` (dependency)
- `writer` (background fsync/writeback pressure) or `db` (SQLite/Postgres) or `cache` (Redis/memory)

Requirements:

- At least **two internal hops** on the critical path (not counting the load generator).
- Each hop must expose request latency (at least basic logging + percentiles).

Required K8s/cgroup concepts to demonstrate

You must run experiments that clearly involve:

- **requests/limits → cgroup behavior** (CPU quota throttling and/or memory limits)
- **QoS class effects** (Guaranteed vs Burstable vs BestEffort)
- At least one of:
  - OOM kill / eviction-related behavior
  - throttling-induced latency amplification
  - noisy-neighbor interference that is visible in cgroup stats

Workload + metric

- Define a client workload that produces stable baseline p50/p95/p99.
- Primary metric: **p99 end-to-end latency** (also report p50/p95).
- Secondary metrics: error rate, throughput.

Fault / interference menu (pick at least 2)

Pick two different tail-latency amplifiers, and for each provide a mechanism-level explanation.

Examples:

- CPU throttling via low limits → runqueue delay / application queueing
- memory pressure (tight limits) → major faults / reclaim / OOM kill
- storage writeback + fsync spikes → tail latency bursts
- network impairment using `tc netem` (delay/loss) → retries / queue growth

Evidence standard (MS baseline)

For **each** interference experiment and mitigation, you must provide:

1. **Two independent pieces of evidence (cross-layer observations)**
    - At least one must come from the **application/user space** (end-to-end latency percentiles, error rate, throughput, hop-level latency, etc.), and another from the **OS/resource control/K8s control plane** (cgroup/PSI/pidstat/iostat/kubectl events, etc.).
    - "Independent" means the two pieces should not both come from the same tool or the same type of metric (for example, two top screenshots do not count).
    - Examples:
        - App latency histogram/logs + cgroup v2 cpu.stat / memory.current
        - PSI (/proc/pressure/*) + iostat -x / pidstat
        - K8s events (kubectl describe pod) + cgroup stats
2. **One exclusionary control (controlled variable / counterexample)**
    - For the most likely alternative explanation, provide evidence showing that it is not the primary cause.
    - Examples
        - Stable iostat → not an I/O bottleneck
        - No throttling in cpu.stat → not a CPU quota issue
        - Low CPU PSI / low CPU usage in pidstat → not CPU contention
3. **Before/after percentiles + corresponding mechanism-level metric changes**
    - Report p50/p95/p99 (or at least p50/p99).
    - Also show one mechanism-level metric moved consistently with your explanation (e.g., `throttled_usec`, PSI, `await`, `oom_kill`).

Perf PMU counters may not work in VMs; don't depend on hardware cache events.

Mitigations (required)

Provide and validate **two** mitigations:

1. **resource / OS / K8s-level** mitigation (change requests/limits, adjust QoS/priority, isolate noisy neighbor, etc.)
2. **application-level** mitigation (timeouts, retries/backoff, batching, caching, admission control)

For each: show that the metric moved, and explain why.