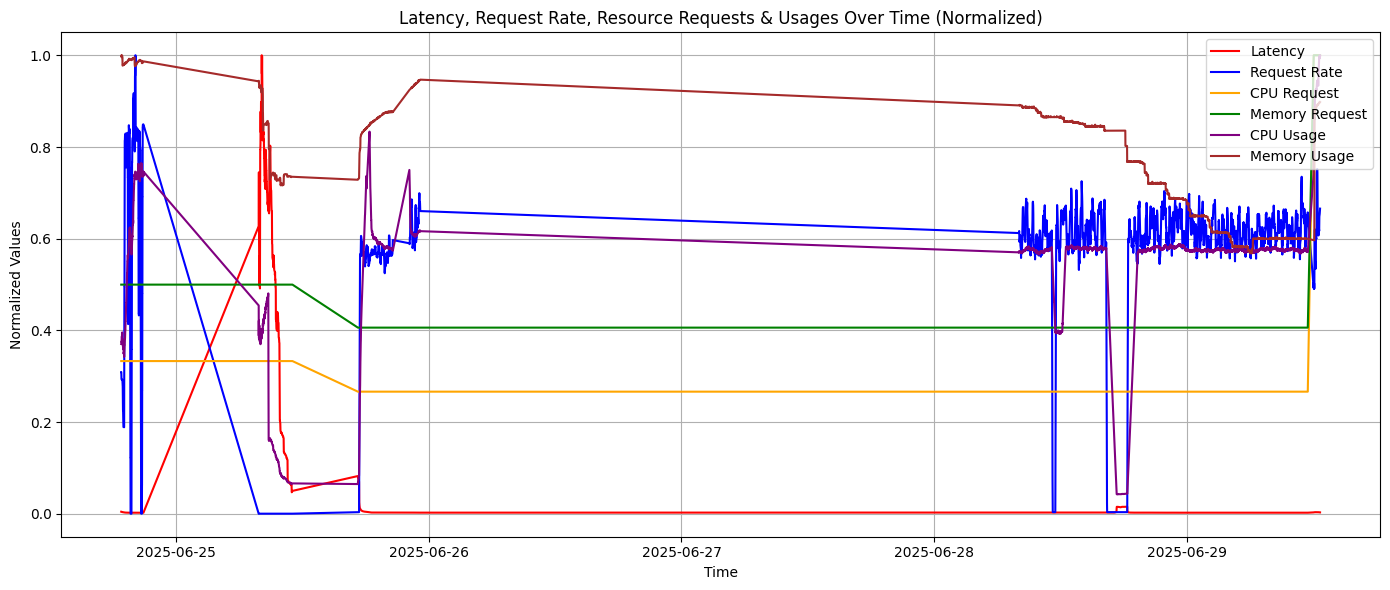
# Service 1



## 1. Observation

Latency:

* June 25: High volatility and sharp latency spikes.
* June 26–29: Latency sharply drops and remains close to zero.

## 2. Interpretation – In-Depth Breakdown

### A. Sharp Latency Spikes on June 25

* These spikes indicate high response times for the application.
* Latency values may be breaching the SLO (Service Level Objective) thresholds intermittently.

|  |  |
| --- | --- |
| Root Cause | Explanation |
| Resource Reduction | CPU/memory request have been reduced, causing queuing or throttling. Check cpu\_throttled\_seconds\_total and memory OOM metrics. |
| Cold Starts or Pod Initialization Delays | New pods were created on demand (e.g., in HPA or VPA scenarios), latency spikes reflect startup delay. |
| Garbage Collection or Go GC Delays | The service is CPU-intensive, latency spikes arise from GC pauses. |
| Reactivity Delay | Scaling process have not reacted quickly enough to load, creating a temporary resource–demand mismatch. |

### B. Latency Drops to ~0 from June 26 to June 29

**Possibilities:**

1. System was Idle

* Ifrequest\_rate ≈ 0, the service was not receiving external load.
* In such cases, latency measurements fall near-zero since:
* No requests are being processed.
* Some systems report default low values or do not record at all.

1. Service Failure or Unavailability

* Latency being near-zero can falsely appear healthy when in fact:
  + The service may have crashed.
  + No pods running, hence no metrics captured.
  + Prometheus scrapes default to zero if no target is available.

1. Aggressive Resource Overprovisioning

* CPU/memory limits may have been increased excessively by an autoscaler or manually.
* This can lead to underutilization and very fast request processing.
* If latency is near-zero with a stable non-zero request rate, this is plausible.

1. Caching or Optimization Mechanism Kicked In

* Caching layers (e.g., Redis, in-memory caches) may have reduced workload drastically.

1. Service is lightweight

* The service can be very lightweight & so the latency will be almost near zero.

|  |  |  |
| --- | --- | --- |
| Behaviour | Most Likely Explanation | Additional Notes |
| Spikes | Resource contention, delayed autoscaling | Indicates stress; tune autoscaler or initial allocation |
| Drop near zero | Lightweight service |  |

### Insights

If we're building a latency-aware autoscaler:

* We must not react to low latency alone — always combine it with:
  + request\_rate
  + cpu\_usage\_pct and memory\_usage\_pct
  + Service health metrics (up, restarts)
* Introduce latency bounds:
  + A latency close to zero with zero usage should trigger scale-down or no-op, not mislead the system into thinking resources are sufficient for future high loads.
* Modeling tip:  
   If we're using this dataset for ML, filter out near-zero request periods or label them as low activity windows so they don’t bias our model.

## Request Rate

* Significant initial spike — a surge in incoming requests.
* Indicates active use or a sudden test/load event.
* Near-zero request rate throughout most of the day.
* Implies the system was light weight, idle, possibly turned off, or not receiving any external/internal requests.
* Frequent, fluctuating activity resumes.
* Pattern resembles non-continuous, possibly batch, or time-triggered request bursts.

### **Layer-by-Layer Technical Breakdown**

**High Initial Request Rate**

|  |  |
| --- | --- |
| Cause | Description |
| Load Testing Initiated | We or a process (e.g., curl, JMeter, Python client) started a high-throughput workload for performance testing. |
| CronJob Executed | A Kubernetes CronJob or scheduled script have kicked off a burst of requests at a specific time. |
| Autoscaling Test or Experiment Triggered | Controlled experiment have started on that day to test latency impact under load. |
| User Traffic or Synthetic Load Generator | External clients or a synthetic generator like Locust, or a Python requests thread pool. |