🔍 Understanding & Diagnosing API Latency Spikes with Splunk

🎯 Focus: Full-stack observability, log correlation, SPL deep dive, service tracing, RCA, and real examples

---

🧱 1. Architecture Context: Why Latency Spikes Are Hard to Trace

Modern microservices = asynchronous + distributed + multi-hop.
One user request can span 5–15 services, sometimes across:

- AWS Lambda, EC2, K8s pods
- Managed DBs (Aurora, MongoDB Atlas)
- Queues (Kafka, SQS)
- External APIs (Stripe, Twilio)

🚨 Problem: **Latency in one hop** (say, Stripe call or thread lock in one container) causes **end-to-end spikes**.
Splunk becomes your microscope when every millisecond matters.

🧠 2. Log Design Best Practices for Splunk-Based Tracing

To trace latency:

📎 Use Consistent Structure for Logs

Log Format (JSON, consistent, indexed fields):

```
{
 "timestamp": "2025-04-18T11:45:02Z",
 "level": "INFO",
 "service": "order-service",
 "span_id": "a1b2c3",
 "parent_span_id": "root",
```

```
 "trace_id": "TXN-001234",
 "duration_ms": 1450,
 "operation": "placeOrder",
 "status": "success",
 "host": "order-node-1",
 "message": "Order placed successfully"
}
```

✅ Include:

- trace_id: Unique per request
- span_id: Per service operation
- duration_ms: Local processing time
- host, container_id, or pod_id for infra mapping
- Optional: http_status, db_time, external_api_time

🧪 3. Step-by-Step Workflow for Diagnosing Latency

🔍 Step 1: Detect latency spikes using percentile SPLs

```
index=prod_logs earliest=-15m
| stats perc50(duration_ms), perc90(duration_ms), perc99(duration_ms) by service
```

➡️ Check which service has **P90/P99 spike**, e.g., payment-service has perc90 > 1400ms.

📈 Step 2: Identify spike pattern over time

```
index=prod_logs service=payment-service
| timechart span=1m perc90(duration_ms)
```

🧠 You now know:

- When latency started
- Whether it's constant, bursty, or increasing

🔬 Step 3: Drill down to affected transactions

index=prod_logs service=payment-service

| where duration_ms > 1000

| stats count, avg(duration_ms), max(duration_ms) by trace_id

| sort -max(duration_ms)

👆 Output shows **trace IDs** where high latency happened.

🧵 Step 4: Trace full call chain of a single trace_id

Let's say trace_id=TXN-001234

index=prod_logs trace_id="TXN-001234"

| sort _time

| table _time, service, operation, duration_ms, host, message

🔍 Example Output:

_time	service	operation	duration_ms	host	message
11:45:01.101	gateway	route	30	gw-1	Routed to checkout
11:45:01.134	auth-service	validateToken	50	auth-1	Auth success
11:45:01.200	cart-service	getCart	90	cart-2	Cart fetched
11:45:01.300	inventory	checkItems	100	inv-2	Inventory confirmed
11:45:01.500	payment	chargeCard	1300	pay-1	Stripe API delay: 1200ms ⚠

11:45:02.900	notify-service	sendSMS	60	sms-1	SMS sent

🚨 Spike is in payment during Stripe API call.

🔄 Step 5: Check Deployment Events Around Spike

index=deployments service=payment-service

| table _time, version, deployed_by, commit_id

Correlate deploy time to spike time.

📟 Step 6: Overlay Infra Metrics

Use metrics logs ingested into Splunk (from Telegraf, CloudWatch, Prometheus exporters).

Overlay CPU & Memory of affected host

index=infra_metrics host="pay-1"

| timechart span=1m avg(cpu_utilization), avg(memory_used)

➡️ High CPU or memory pressure may cause GC pauses, IO delays.

📊 Step 7: Heatmap of Latency Distribution

index=prod_logs service=payment-service

| bucket duration_ms span=100

| stats count by duration_ms

| sort duration_ms

➡️ Helps build latency histogram.

🎯 Step 8: Correlate with Error, GC, Thread Logs

🔥 Find Full GC Events

index=gc_logs host="pay-1" message="Full GC"

| table _time, message, duration

Correlate Full GC (1.2s) with your trace latency!

🧵 Detect Thread Blocking

index=thread_logs host="pay-1" message="blocked"

| stats count by thread_id

🔁 Step 9: Automate RCA with transaction or stats

index=prod_logs service=payment-service

| transaction trace_id startswith="Payment request started" endswith="Payment complete"

| eval total_time = duration * 1000

| where total_time > 1000

| table trace_id, total_time, host

Or alternatively:

index=prod_logs service=payment-service

| stats sum(duration_ms) as total_duration by trace_id

| where total_duration > 1000

🔍 Step 10: Look for Common API/DB Bottlenecks

🔗 External API (Stripe, Razorpay, etc.)

index=prod_logs service=payment-service message="Calling Stripe"

| stats avg(duration_ms), count by api_url

📄 DB Query Analysis

```
index=prod_logs service=payment-service message="Executing query"
| stats avg(duration_ms), count by sql_query
| sort -avg(duration_ms)
```

📌 RCA Matrix Cheat Sheet

🔍 Suspected Cause	SPL Example / Log Signal
External API delay	message="Stripe API" + duration > 1000
Thread Pool Saturation	message="ThreadPoolExhausted"
GC Pause	index=gc_logs Full GC
N+1 Query Pattern	High # of similar SQLs per trace
Load Balancer Delay	Logs from gateway show high queue_time_ms
Network I/O delay	message="HTTP timeout"
CPU pressure	cpu_utilization > 85%
Memory leak / GC churn	Rising GC duration + heap_used across traces

---

🧠 Final Dashboards You Must Have

Dashboard Panel	SPL Summary
P90 Latency Over Time	timechart span=1m perc90(duration_ms)
Top 5 Slow Services	`stats avg(duration_ms) by service
Slowest Transactions (trace_id)	where duration_ms > 1000 + stats by trace_id
GC Events Timeline	`index=gc_logs Full GC
External API Call Latency	message="Stripe API" + stats by api_url
Heatmap of duration buckets	`bucket duration_ms span=100
Infra overlays (CPU/Memory)	timechart avg(cpu_utilization), avg(memory_used)

Errors vs Duration Correlation	stats count by http_status, duration_ms

---

🔔 **Advanced Use Case: Automated Anomaly Detection**

Use **Splunk Machine Learning Toolkit** or SPL thresholds:

index=prod_logs service=payment-service

| timechart span=1m avg(duration_ms) as duration

| anomalydetection duration

➡️ Setup **alerts** when P90 > 1000ms or Stripe API > 1200ms using savedsearch.

✅ **Summary – What to Do When You See Latency Spikes**

1. **Check percentiles** (p90, p99) across services
2. **Drill down by trace ID** using duration_ms > threshold
3. **Trace full call path** → locate spike
4. **Correlate with deployments, infra logs, GC, thread logs**
5. **Use SPL to find common patterns** in external APIs or DB
6. **Automate dashboards and alerts** for future detection