🎓 Approach for Handling Sudden Increase in Errors: A Deep Dive

---

🚨 Step 1️⃣: Immediate Response – Triage & Containment

🛡️ **Primary Objective:**
Minimize impact while you investigate.

🔷 **Actions:**

- **Pause/Throttle Load**:
  - If in performance test → pause test in JMeter.
  - If in production → engage circuit breakers, rate limiters, or temporarily redirect traffic.
- **Acknowledge Alerts**:
  - Triggered by error thresholds (e.g., 5xx > 2% TPS).
  - Note the **exact time of occurrence**.
- **Check Blast Radius**:
  - Is it **global** (affecting all services) or **localized** (specific to a few APIs/microservices)?
  - What **error codes** are spiking? (4xx vs 5xx vs network errors)
- **Snapshot System State** (Forensic Capture):
  - Take **thread dumps**.
  - Capture **top**, **vmstat**, **iostat**, **netstat** outputs.
  - Download recent **logs**.
  - Save **GC logs**, **DB slow query logs**, **connection pool metrics**.

🧭 Step 2: Metric Correlation & Pattern Identification

🔷 Analyze System Metrics (from CloudWatch, Prometheus, top, or custom tools):

Metric	What to Check	Tools/Commands
CPU Utilization	Spikes indicating thread pool exhaustion, GC storms, or high compute tasks.	top, htop, CloudWatch, Prometheus
Memory Usage	Signs of memory leaks, OOM killers.	free -m, vmstat, GC logs
Thread Pool Usage	Reaching max active threads? Stuck threads?	jstack, Tomcat metrics (maxThreads, activeThreads)
Connection Pool	Hitting limits? Long waits?	HikariCP, DB metrics, logs
GC Activity	Full GCs? Promotion failures?	jstat -gc, GC logs
Disk I/O	Latency spikes, IOPS bottlenecks.	iostat -dx 1, CloudWatch
Network I/O	Dropped packets, high latency.	netstat -s, ss, VPC Flow Logs

---

🔬 Step 3: Log Analysis & Error Classification

🔷 Classify Errors:

Error Type	Typical Cause	Example
5xx (500/502/503)	App crashes, resource exhaustion, downstream failures	500 Internal Server Error: SQLTimeout
4xx (400/401/403/404)	Invalid client input, auth errors, missing resources	401 Unauthorized due to expired JWT
Network Errors (Timeouts, Resets)	DNS issues, API latency, load balancer timeouts	ReadTimeoutException: 30000 ms
Custom App Errors	Business logic failures, null pointer exceptions	NullPointerException in OrderService

🏗 Step 4️⃣: Deep-Dive Service/Code Analysis

🔷 For 5xx Errors:

1️⃣ **Thread Dump Analysis** (jstack, ps -eLf):

- Look for threads stuck on:
  - DB calls: java.sql.*
  - HTTP clients: org.apache.http.client
  - Locks: java.util.concurrent

2️⃣ **GC Logs**:

- Full GC pauses?
- Old Gen fill-up rate?
- jstat -gcutil <pid> 1s

3️⃣ **DB Bottlenecks**:

- Slow queries? (EXPLAIN, slow query logs)
- Connection pool exhausted? (Check maxConnections, timeouts)
- Locks/Deadlocks? (SHOW ENGINE INNODB STATUS)

4️⃣ **External Dependencies**:

- API timeouts? Rate limiting? (curl, Postman)
- Circuit breaker/fallback logic missing?

5️⃣ **Infra Limits**:

- Container CPU/Mem limits? (docker stats, ECS/Fargate task settings)
- Load balancer unhealthy targets?

🧱 Step 5️⃣: Dependency & Release Analysis

🔷 **Check Recent Changes:**

- **Code Deployments**:
  - Any **recent merges/releases** that could impact flow?
  - Rollback if necessary.
- **Infrastructure Events**:
  - **Autoscaling events**, **pod restarts**, **node replacements**.
- **Config Changes**:
  - Timeouts, connection limits, retries, feature flags.
- **Dependency Failures**:
  - API version changes?
  - DB migrations?
  - Cache invalidation?

🧪 Step 6️⃣: Reproduction & Controlled Load Testing

🔷 **Validate Error Threshold:**

- Simulate failing scenarios:
  - Use **curl** with same headers, payloads.
  - Use **JMeter/LoadRunner** to replay transactions.
- Gradually ramp up load:
  - Identify the **breakpoint** where errors begin.
- Vary scenarios:
  - Different user flows, payloads, auth tokens, regions.

🔄 Step 7: Fixes & Mitigation

Fix Area	Action	Examples
DB Bottleneck	Indexing, query optimization, connection pool tuning	Add index on order_id, status; increase HikariCP maxPoolSize
App Bottleneck	Thread pool sizing, code optimization, async patterns	Increase maxThreads; use non-blocking I/O
Infra Bottleneck	Scale-out, increase resources	Add ECS tasks; increase EC2 size
Resilience	Add circuit breakers, retries, fallbacks	Use Resilience4j for downstream APIs
Configs	Timeouts, retries, rate limits	Tune HTTP client timeouts to < LB idle timeout
Observability	Add metrics, alerts, logs	Expose custom metrics (e.g., DB pool usage)

📊 Step 8: Document RCA & Action Plan

Aspect	Details
Root Cause	DB pool exhaustion due to inefficient query & under-provisioned infra.
Contributing Factors	Missing index, low pool size, no circuit breaker, under-scaled ECS.
Impact	20% 5xx errors at 250+ users.
Fixes Applied	Index added, pool size increased, ECS scaled, circuit breaker added.
Lessons Learned	Add observability, pre-load testing for infra limits, query tuning.
Next Steps	Monitor proactively, integrate metrics into CI/CD gates, chaos testing.

📋 Final Technical Checklist

✅ Error classification by type & endpoint
✅ Metric correlation (CPU, Memory, GC, DB, threads)
✅ Thread dump & GC analysis
✅ Logs deep dive (stack traces, exceptions, latency spikes)
✅ DB connection pool & query performance review
✅ Recent deployment/configuration changes
✅ External dependencies health
✅ Reproduction under controlled conditions
✅ Fixes: Query optimization, infra scaling, thread pool tuning
✅ RCA documentation
✅ Monitoring & alerting setup post-mortem
