

Technically elaborate guide to checking for deadlocks or stuck threads in a Java application, especially useful in performance engineering, production debugging, and JVM tuning contexts.

Goal:

Detect, confirm, and analyze deadlocks or stuck threads using thread dumps, monitoring tools, and logs.

Tools & Prereqs:

Tool/Resource	Use Case
jstack, kill -3	Capture thread dumps on Unix/Linux
VisualVM / JConsole	GUI-based live thread inspection
TDA / FastThread.io	Analyze thread dumps visually
Splunk / Dynatrace	Observability, traces, thread states
App logs	Check for hung operations, long waits

Step-by-Step Troubleshooting Guide

Step 1: Capture Thread Dumps


➤ Option 1: CLI – jstack

```
jstack <PID> > threadDump_$(date +%F_%H-%M-%S).log
```

➤ Option 2: kill -3 (sends SIGQUIT)

```
kill -3 <PID>
```

Output goes to stdout or catalina.out (Tomcat), server.log (JBoss)

 Capture 3–5 dumps at 10-second intervals to confirm "stuck" threads

Step 2: Look for Deadlock Patterns

Search for this header:

Found one Java-level deadlock:

=====

Followed by:

"Thread-1": waiting to lock monitor 0x00000000xxx (object A), which is held by "Thread-2"

"Thread-2": waiting to lock monitor 0x00000000yyy (object B), which is held by "Thread-1"

 **Deadlock = Cyclic wait** for resources/locks.

Step 3: Identify Stuck Threads (Without Deadlock)

Search for:

- BLOCKED state with waiting to lock
- WAITING or TIMED_WAITING state for long duration
- Threads with same stack trace across multiple dumps

Sample indicators:

"Thread-23" #45 prio=5 os_prio=0 tid=0x00007f8b0c1b1800 nid=0x75d3 waiting on condition [0x00007f8b0b7f7000]

java.lang.Thread.State: WAITING (parking)

at sun.misc.Unsafe.park(Native Method)

at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)

at

java.util.concurrent.locks.AbstractQueuedSynchronizer.parkAndCheckInterrupt(AbstractQueuedSynchronizer.java:836)

at

java.util.concurrent.locks.AbstractQueuedSynchronizer.acquireQueued(AbstractQueuedSynchronizer.java:870)

at

java.util.concurrent.locks.AbstractQueuedSynchronizer.acquire(AbstractQueuedSynchronizer.java:1199)

at java.util.concurrent.locks.ReentrantLock.lock(ReentrantLock.java:267)

Step 4: Look for Stacktrace Repeats

Use `uniq -c` to find identical thread stack traces in dump:

```
grep -A20 "java.lang.Thread.State" threadDump.log | sort | uniq -c | sort -nr | head
```

 **High repeat count = same stack → thread stuck there repeatedly**

Step 5: Check for Monitor Locks

Look for locked `<0x...>` and waiting to lock `<0x...>` entries.

- locked `<0x00000000f0a12f10>` (a `java.util.concurrent.locks.ReentrantLock$NonfairSync`)

- waiting to lock `<0x00000000f0a12f10>` (a `java.util.concurrent.locks.ReentrantLock$NonfairSync`)

 Investigate which threads own the lock vs which are blocked.

Step 6: Use Visual Tools for Better Insight


✓ [TDA (Thread Dump Analyzer)]

- Load .log file
- Shows grouped threads
- Visualizes waiting/blocking relationships

✓ fastthread.io

- Upload thread dump → get:
 - CPU-consuming threads
 - Stuck threads
 - Lock hierarchy
 - Root cause summary
-

Additional Tips:

 Scenario	Possible Root Cause Example
Threads in WAITING on Future.get()	Async task never completes
Many threads blocked on same lock	Contention on synchronized block or DB connection pool
Threads stuck in I/O methods	Disk/Network slowness or dead remote call
Threads with JDBC calls	DB connection leakage or unclosed ResultSet
Long GC pause + thread park	GC thrashing causing all threads to pause

Sample Automation – Cron Thread Dump Capture

```
#!/bin/bash
PID=$(jps | grep MyApp | awk '{print $1}')
while true; do
    jstack $PID > threadDump_$(date +%F_%H-%M-%S).log
    sleep 10
done
```

Metrics to Correlate (from APM or JVM):

Metric	Meaning
Thread count spikes	More threads → potential contention
Blocked threads	Waited for lock too long
GC pause time + count	GC may pause all app threads
CPU usage + no progress	CPU bound or thread stuck
DB pool active connections = max	All threads waiting for DB

Final Diagnosis Strategy

Confirm Stuck Threads:

- Appears in 3+ dumps
- Same call stack
- State is WAITING, BLOCKED, or RUNNABLE doing nothing

Confirm Deadlock:

- JVM clearly logs Java-level deadlock
- Involves mutual lock hold/wait

Fix Strategies

Issue Type	Fix Ideas
Deadlocks	Use tryLock(), re-architect lock order, avoid nested locks
Long Waits	Set timeouts on get(), join(), etc.
Lock Contention	Use concurrent structures, reduce lock granularity
DB thread pool full	Increase pool size, fix slow queries
GC pauses	GC tuning, increase heap, fix memory leaks

Real-World Case Study Snapshot

Problem: All API requests hung intermittently.

Findings:

- Thread dump showed 87 threads stuck in:
at org.apache.commons.dbcp2.PoolingDataSource.getConnection()
- DB connection pool exhausted due to:
 - Unclosed connections
 - High query execution time

Fix:

- Set connection timeout
 - Ensure try-with-resources on JDBC
 - Optimized long-running queries
-