How do you analyze thread activity, lock contention, or deadlocks in Dynatrace JVM monitoring?

🔍 Dynatrace provides **deep JVM observability**, and it's incredibly effective for analyzing **thread activity**, detecting **lock contention**, and identifying **deadlocks** — even in **live production systems** without manual thread dump collection.

Let's break this into a step-by-step method to analyze:

- 🔸 **Thread states**
- 🔸 **Lock contention**
- 🔸 **Deadlocks**
- 🔸 **Stack traces**
- 🔸 How I typically use it in real scenarios

✅ 1. Navigate to JVM Process for the Java Service

🧭 **Steps:**

1. **Go to Dynatrace → Hosts → Your Java-based host**
2. Click **"Processes" tab**
3. Select the **JVM process** (e.g., Tomcat, Spring Boot, JBoss, etc.)
4. Click **"View process details" → "JVM metrics"**

From here, you can analyze thread activity, heap memory, GC, and CPU usage.

🧵 2. Analyze Thread Activity

📈 **Dynatrace automatically tracks:**

Metric	**What it shows**
Live threads	Current active threads
Daemon threads	Threads in background services
Peak thread count	Useful to correlate with spikes
Thread state breakdown	RUNNABLE, BLOCKED, WAITING, TIMED_WAITING

📊 **Visual View:**

Dynatrace plots this over time. You can correlate:

- Spikes in threads
- Against GC pauses or CPU usage
- With transaction latencies

🔒 3. Detect Lock Contention

Dynatrace tracks **threads in BLOCKED state** (waiting for a monitor/lock).

💡 **How to Identify:**

1. Navigate to the **"Threads" section** of the JVM process
2. Filter by **Thread state = BLOCKED**
3. View the **thread name**, owning class, and **what lock is being waited on**

🧠 **You'll see:**

Thread-83

State: BLOCKED

Waiting to lock: java.util.HashMap@3e0b5eb

Owned by: Thread-44

✓ This helps you catch **which threads are bottlenecked** and **what resource** is causing it.

🚨 4. Deadlock Detection (via Dynatrace AI)

Dynatrace automatically detects **Java-level deadlocks** using thread metadata.

🧠 **Davis AI triggers a Problem card when:**

- Two or more threads are blocked and waiting on each other's locks (circular wait)
- This is correlated with **CPU starvation**, **latency spikes**, or **application freeze**

You'll get:

- Problem ID
- Threads involved
- Locked objects

- Stack traces of both threads
- Suggested root cause

📌 Unlike traditional thread dump parsing, this is **real-time and auto-detected** — no manual steps needed.

🔍 5. Stack Trace Analysis of Individual Threads

For **each thread**, you can:

- View current **stack trace**
- Identify stuck loops, blocking I/O, or long-running operations

Example:

"Thread-121" (RUNNABLE)

→ at com.example.UserService.getUserDetails()

→ at java.sql.Connection.prepareStatement()

→ waiting on jdbc:mysql://...

This helps correlate:

- **Latency** to **code-level hot spots**
- See if threads are doing active work or waiting/blocking

⚙️ 6. Advanced: Thread Diagnostics Snapshot (on-demand)

Dynatrace also allows you to **trigger a thread diagnostics snapshot**:

📸 **Use case:**

- When you detect a CPU spike or thread lock issue
- Capture live thread states without restarting or attaching jstack

Steps:

1. Navigate to the JVM process
2. Click "... More Actions" → **Thread dump**
3. Capture thread snapshot
4. Download or analyze directly in Dynatrace UI

📄 Exports in Thread Dump format similar to jstack

📈 7. Correlate with Metrics and PurePaths

🔄 Combine thread insight with:

Insight	Tool
🚦 **Latency spikes**	PurePath or service dashboards
⚙️ **High CPU + few RUNNABLE threads**	Possible thread starvation
⛔ **Lots of BLOCKED threads**	Lock contention on shared resources
💥 **App freeze with cyclic lock wait**	Deadlock (Problem card generated)

🧪 Real Example: Lock Contention in Production

🎯 Problem:

- Latency spike in invoice-service
- Davis AI flagged high CPU + blocked threads

🧩 Findings:

- 200+ threads in BLOCKED state
- All waiting on ConcurrentHashMap access
- Stack traces pointed to InvoiceCache.computeIfAbsent()

🛠 Fix:

- Replaced hotspot logic with memoized lock granularity
- Validated with thread view post-deployment: **No BLOCKED threads**

✅ Summary Cheat Sheet

Feature	Purpose
🧵 Thread state graph	Visualizes thread growth over time
🔒 BLOCKED state detection	Shows lock wait points
🧠 Deadlock detection	Auto-detected with thread relationship analysis

🪵 Stack trace viewer	Pinpoints where thread is blocked or spinning
📸 Thread snapshot	On-demand full state export for offline analysis
🔄 PurePath + Threads	Correlate code-level tracing with thread state