🔍 GC Metrics to Monitor in Production (Advanced GC Observability)

1. 🚀 Allocation Rate (objects/sec or MB/sec)

🔧 What it is:

The **Allocation Rate** represents how fast your application is **allocating objects in the heap**, typically measured in **MB/sec** or **objects/sec**.

🔬 Why it's important:

- High allocation rates can **trigger frequent Young GCs**, increasing CPU usage.
- It helps you understand if object creation is aligned with heap size and GC frequency.
- Indicates **object churn**—the volume of short-lived objects being created and collected.

📈 Observability Impact:

- **Young GC frequency ↑** if allocation rate is high.
- Could cause **CPU pressure** even if GC pause is low.
- Correlates with **application throughput** (e.g., request volume).

✅ Best Practice:

- Tune GC frequency or heap size based on allocation trends.
- Optimize code to **reuse objects** (e.g., buffer reuse, object pooling).

📌 Metrics Example:

- jvm.memory.heap.allocated.rate: 850 MB/sec

- jvm.gc.memory.allocated: Prometheus counter reset at GC

---

2. ⏱️ GC Pause Time (milliseconds per GC event)

🔧 What it is:

Pause Time is the duration during which the **application threads (STW - Stop-The-World)** are paused while the GC runs.

🔬 **Why it's important:**

- **Directly impacts latency-sensitive workloads**.
- Prolonged pauses may cause **request timeouts**, **lag spikes**, and **throughput drops**.
- Reflects **GC algorithm efficiency** (G1GC vs CMS vs ZGC).

📈 **Observability Impact:**

- Latency spike in APM tools (e.g., NewRelic, Dynatrace).
- High tail latencies (P95/P99) often correlate with GC pauses.

✅ **Best Practice:**

- Aim for **consistent low pause times**.
- Tune GC flags: -XX:MaxGCPauseMillis=200 (for G1).
- Avoid **Full GC** at runtime by controlling promotion (see below).

📌 **Metrics Example:**

- jvm.gc.pause.time: Histogram of all GC pauses (mean, 95th, 99th percentile)

- jvm.gc.pause.count: Counter per GC event

---

3. 🔄 Promotion Rate (MB/sec or objects/sec)

🔧 **What it is:**

The **Promotion Rate** measures how much memory (or how many objects) are being **moved from Young Generation to Old Generation** during GC.

🔬 **Why it's important:**

- High promotion rate → rapid Old Gen fill-up → **increased risk of Full GC**.
- Indicates objects surviving longer in Young Gen (might be a **leak symptom**).
- Helps **forecast memory pressure** on Old Gen.

📈 **Observability Impact:**

- **Spikes in promotion rate** often precede **Full GC events**.
- Memory leaks show consistent upward trend in promotion without reclaim.

✅ **Best Practice:**

- Use -XX:MaxTenuringThreshold to control survivor age.
- Analyze object lifetimes using **heap dumps** or **GC logs**.

📌 **Metrics Example:**

- jvm.gc.promotion.rate: 200 MB/sec to Old Gen

- GC log line sample: "Promoted 25MB to Old"

Additional GC Metrics (Bonus)

Metric	Description	Monitoring Insight
🧠 **Old Gen Utilization**	Memory used in Old Generation space	Early warning for Full GC
📦 **Live Set Size**	Amount of heap used after GC	Indicates memory retention trend
📉 **Heap Fragmentation**	Gaps in Old Gen that GC can't compact	Increases allocation failure risk
🚨 **Full GC Count & Time**	# of Full GC events & duration	SLA-critical, must be near-zero in prod
💀 **Finalizer Queue Length**	Finalizable object backlog	High values → memory leak or GC pressure

---

📊 **Sample GC Dashboard Panels (Grafana / Prometheus)**

1. 🔁 **Young GC Frequency vs Allocation Rate**
2. ⏳ **GC Pause Time Distribution (P50, P95, P99)**
3. 🧠 **Old Gen Usage vs Promotion Rate Trend**
4. 🔥 **GC Event Heatmap by Type (Young, Old, Full)**

🧠 Real-World Observability Scenario

Problem: Latency spikes every 3 minutes in production API.

Investigation using GC Metrics:

- Allocation rate: 1200 MB/sec
- Promotion rate: 250 MB/sec
- Old Gen near 80% filled every 3 mins
- GC logs show frequent **Full GC**

Fixes:

- Increased heap size from 8GB → 12GB
- Tuned G1GC with -XX:MaxTenuringThreshold=5
- Refactored service to avoid large object allocation per request

Result: GC pause dropped from 800ms → 120ms, P99 latency improved.

✅ Summary – What to Track Daily

Metric	Goal / Threshold
Allocation Rate	< 800 MB/sec (for medium traffic apps)
GC Pause Time (P95)	< 200ms (for APIs), < 500ms (batch jobs)
Promotion Rate	< 200 MB/sec; avoid sudden spikes
Full GC Frequency	0 or near-zero
Old Gen Occupancy	< 70% sustained usage
Survivor Promotion Fail	0 ideally; if present → tune survivor space sizes

---