# Problem 3: Diagnose Me Doctor

Below is a quick-reference grid for the two most common causes:

| Root Cause | Impacts | Recovery Actions |
|---|---|---|
| Misconfiguration / Memory Leak | Gradual climb to 99% memory, OOM kills, dropped requests. | - Restart NGINX<br>- Adjust or remove problematic modules<br>- Tune workers, caching<br>- Patch/upgrade NGINX |
| Excessive / Malicious Traffic | Rapid or sustained spike in memory usage, request timeouts, and potential downtime. | - Block or throttle suspicious IPs<br>- Scale horizontally (more load balancers)<br>- Harden configuration with rate-limiting<br>- Use DDoS mitigation service |

## 1. Verify and Quantify the Issue

1. **Check Monitoring Dashboards**
    - Confirm that memory usage is indeed at 99% and not a false alarm.
    - Compare current usage with historical trends to see if this is a one-time spike or a recurring issue.
2. **Inspect Memory on the VM**
    - Run `free -h` or `vmstat` to verify total memory, used memory, and any swap usage.
    - Use `top` / `htop` to see real-time memory consumption by process.
3. **Check Processes Sorted by Memory**
    - `ps aux --sort=-%mem | head -n 10` to identify the top memory-consuming processes.
    - Confirm that NGINX (or one of its worker processes) is the culprit.

---

## 2. Gather Logs and System Information

1. **System Logs**
    - Review `/var/log/syslog` or `journalctl -xe` for any OOM (Out-Of-Memory) kill events or errors pointing to resource exhaustion.
    - Look at `dmesg` for kernel messages that might show memory allocation or OOM kill details.

2. **NGINX Logs**
    - Check the NGINX error log (commonly `/var/log/nginx/error.log`) for memory allocation issues, repeated “worker process exited” messages or warnings about timeouts/overloads.
    - Examine the access log (`/var/log/nginx/access.log`) for unusual spikes in requests, and patterns that might indicate a flood of traffic or DDoS attempts.
3. **Configuration Files**
    - Open `/etc/nginx/nginx.conf` to review the main configuration:
        - `worker_processes`, `worker_connections`, and any module-specific settings.
    - Look at additional config files in `/etc/nginx/conf.d` or `/etc/nginx/sites-enabled` for caching, Lua modules, or other features that might consume extra memory.

---

# 3. Narrow Down the Root Cause

## 3.1 Confirm Whether It’s a Misconfiguration or Memory Leak

1. **Inspect NGINX Worker Processes**
    - Run `ps aux | grep nginx` to see each worker process’s memory usage.
    - If one or more workers are continuously growing in memory usage, suspect a memory leak or misconfiguration (e.g., excessive caching).
2. **Audit Custom Modules**
    - Check if you’re using any third-party modules (e.g., Lua, Redis, or advanced caching).
    - Temporarily disable or revert to a vanilla NGINX configuration to see if the memory usage improves.
3. **Tweak NGINX Worker Settings**
    - If `worker_processes` is manually set too high (e.g., 16 workers on a small VM), reduce it to align with available CPU cores or use `worker_processes auto;`.
    - Review `worker_connections` to ensure it isn’t excessively high for your traffic patterns.
4. **Look for Known Bugs**
    - Check your NGINX version and any modules against known issues in official or community forums.
    - Update or patch if there’s a reported memory leak in a specific version or module.

**Potential Outcomes**

- **Misconfiguration**: Memory usage drops after correcting worker_processes or disabling a problematic module.

- **Memory Leak in a Module**: Memory usage continues to climb over time. A software upgrade or patch may be required.

---

### 3.2 Check for Excessive or Malicious Traffic

1. **Traffic Analysis**
   - Inspect access logs to identify a sudden surge in traffic or repetitive requests from the same IP ranges.
   - Use `netstat -antp` or `ss -tunap` to see the number of active connections and their states (ESTABLISHED, TIME_WAIT, etc.).
2. **Rate of Requests**
   - If your normal traffic is, say, 1K requests/min, but logs show 5x that amount, suspect a DDoS or heavy legitimate spike (e.g., a marketing campaign).
   - Compare request patterns with standard load to confirm abnormal spikes.
3. **Check Upstream Errors**
   - If you have health checks or an upstream service that is failing, NGINX might be retrying connections excessively, leading to ballooning memory usage.

**Potential Outcomes**

- **Legitimate Traffic Spike**: High usage caused by real demand (e.g., a product launch).
- **DDoS or Malicious Traffic**: Abnormal IP patterns, large numbers of small requests, or suspicious user agents flooding the service.

---

## 4. Immediate Remediation Steps

1. **Restart NGINX**
   - `sudo systemctl restart nginx` or `sudo service nginx restart` can temporarily free memory used by worker processes.
   - Monitor to see if memory usage quickly ramps up again (indicating a persistent leak or recurring spike in traffic).
2. **Enable or Increase Swap** (Short-Term Relief)
   - If you're running critically low on RAM, adding swap (`sudo fallocate -l 2G /swapfile`, then `sudo mkswap /swapfile && sudo swapon /swapfile`) can prevent immediate OOM kills.
   - Note that swap is slower than RAM—this is a band-aid, not a permanent solution.
3. **Block or Rate-Limit Malicious IPs**
   - If logs indicate malicious traffic from specific IP ranges, block them temporarily with firewall rules (e.g., `iptables` or your cloud's security groups).

    - Consider using `limit_req_zone` or `limit_conn_zone` in NGINX to throttle suspicious traffic.
4. **Scale Up Temporarily**
    - If you're on a cloud platform, increase the VM size (RAM) or add more load balancer nodes to handle the surge until you can implement a more permanent fix.

---

# 5. Long-Term Fixes

1. **Optimize NGINX Configuration**
    - Right-size `worker_processes` and `worker_connections`.
    - Tune keep-alive settings (`keepalive_timeout`) to ensure connections aren't held open unnecessarily.
    - If caching, configure cache size limits and eviction policies carefully.
2. **Upgrade or Patch NGINX / Modules**
    - If a memory leak is identified in a specific version, upgrade to the latest stable release.
    - Remove or replace buggy third-party modules.
3. **Set Up Autoscaling**
    - Use a managed load balancer service or create an autoscaling group that spins up additional instances when traffic spikes.
    - Implement a DNS-based round-robin or cloud load-balancing solution for better high-availability.
4. **Harden Against DDoS**
    - Enable DDoS protection at the cloud or CDN level (e.g., AWS Shield, Cloudflare, etc.).
    - Configure rate-limiting rules, IP reputation checks, and WAF (Web Application Firewall) to filter malicious traffic.
5. **Improve Observability**
    - Implement metrics collection and analysis (e.g., Prometheus + Grafana).
    - Create alerts on memory usage, connection counts, 5xx error rates, etc., to catch issues early.