"CpuShares": 0,

```
        "NanoCpus": 700000000,
        "CpuPeriod": 0,
        "CpuQuota": 0,
        "CpuRealtimePeriod": 0,
        "CpuRealtimeRuntime": 0,
        "CpusetCpus": "",
        "CpusetMems": "",
        "CpuCount": 0,
        "CpuPercent": 0,
            "/sys/devices/system/cpu/cpu0/thermal_throttle",
            "/sys/devices/system/cpu/cpu1/thermal_throttle",
            "/sys/devices/system/cpu/cpu2/thermal_throttle",
            "/sys/devices/system/cpu/cpu3/thermal_throttle",
            "/sys/devices/system/cpu/cpu4/thermal_throttle",
            "/sys/devices/system/cpu/cpu5/thermal_throttle",
            "/sys/devices/system/cpu/cpu6/thermal_throttle",
            "/sys/devices/system/cpu/cpu7/thermal_throttle"
```

Based on your `docker inspect` output, I can see that the CPU limit **is being applied correctly**:

- `"NanoCpus": 700000000` confirms your 0.7 CPU limit is set (700 million nanoseconds = 0.7 CPUs)
- However, `"CpuPeriod": 0, "CpuQuota": 0` shows you're using the less reliable `cpus` parameter rather than the more granular period/quota approach

The fact that you're still seeing 116% CPU usage despite the correctly applied limit indicates a **Docker CPU enforcement bug** or measurement issue.

Why This Happens

1. Docker CPU Limit Enforcement Issues

Docker's CPU limits using `NanoCpus` are known to have enforcement problems where containers can temporarily exceed their limits, especially during short bursts. The enforcement is not always strict in real-time. [1] [2]

2. Monitoring vs. Docker Accounting

Your monitoring tool and Docker's internal CPU accounting may calculate usage differently:

- **Docker stats** shows percentage based on available CPU time over periods
- **System monitoring** might show instantaneous usage peaks
- **cgroups accounting** can have measurement delays

3. CPU Burst Behavior

Even with limits set, Linux's CFS (Completely Fair Scheduler) allows short bursts above the limit before throttling kicks in.[2] [3]

Solutions to Fix This

1. Switch to CPU Period/Quota (Most Reliable)

Replace your current `cpus: "0.7"` with the more granular approach that has better enforcement:

```
virtual_device_3:
  container_name: device_3
  build:
    dockerfile: ${UBUNTU_DOCKERFILE}
    context: .
  image: ${UBUNTU_DEVICE_IMG}
  hostname: device3
  # Remove: cpus: "0.7"
  # Remove: mem_limit: "800m"
  cpu_period: 100000    # 100ms period
  cpu_quota: 70000      # 70ms out of 100ms = 0.7 CPU max
  memory: 800m
  networks:
    - devices_network
  volumes:
    - /etc/timezone:/etc/timezone:ro
    - /etc/localtime:/etc/localtime:ro
    - ./scripts/load_coordinator.sh:/usr/local/bin/load_coordinator.sh:ro
    - load_coordinator:/var/run/load_coordinator
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:9273/metrics"]
    interval: 30s
    timeout: 10s
    retries: 3
```

2. Add CPU Core Restriction

Limit the container to specific CPU cores to prevent it from spreading load across all 8 cores:

```
virtual_device_3:
  # ... other config
```

```
  cpuset: "0"           # Restrict to only CPU core 0
  cpu_period: 100000
  cpu_quota: 70000
  memory: 800m
```

3. Verify the Fix

After applying the changes, check that the new limits are set:

```
docker-compose down && docker-compose up -d
docker inspect device_3 | grep -E "(CpuPeriod|CpuQuota|CpusetCpus)"
```

You should see:

```
"CpuPeriod": 100000,
"CpuQuota": 70000,
"CpusetCpus": "0",
```

4. Monitor Over Longer Periods

CPU limits work over time periods, not instantaneously. Monitor with longer intervals:

```
# Monitor over 10-second intervals instead of real-time
docker stats --format "table {{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}" device_3
```

Why Your Current Setup Allows Bursts

The `NanoCpus` approach you're currently using allows for **CPU bursting** - short periods where the container can exceed its limit before being throttled. The `cpu_period`/`cpu_quota` method provides **stricter enforcement** by defining exact time windows where CPU usage is measured and limited.

Complete Updated Configuration

Here's your `virtual_device_3` with the strictest possible CPU limiting:

```
virtual_device_3:
  container_name: device_3
  build:
    dockerfile: ${UBUNTU_DOCKERFILE}
    context: .
  image: ${UBUNTU_DEVICE_IMG}
  hostname: device3
  cpu_period: 100000      # 100ms measurement period
  cpu_quota: 70000        # Max 70ms CPU time per 100ms period
  cpuset: "0"             # Restrict to CPU core 0 only
  memory: 800m            # Replaces mem_limit
  networks:
```

```
    - devices_network
  volumes:
    - /etc/timezone:/etc/timezone:ro
    - /etc/localtime:/etc/localtime:ro
    - ./scripts/load_coordinator.sh:/usr/local/bin/load_coordinator.sh:ro
    - load_coordinator:/var/run/load_coordinator
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:9273/metrics"]
    interval: 30s
    timeout: 10s
    retries: 3
```

This combination should provide the strictest CPU enforcement possible. If you still see brief spikes above 70%, they should be much shorter and less frequent than with your current configuration.

⁂

1. https://www.reddit.com/r/docker/comments/1h19s19/new_docker_compose_resource_reservations_and/
2. https://overcast.blog/docker-resource-limits-a-guide-5461355171b2
3. https://docs.docker.com/engine/containers/resource_constraints/
4. docker-compose.yml