Changing instance p99.5 types is a critical lever in cloud cost optimization and performance tuning. But going in blind can lead to under-provisioning, application instability, or worse, wasted spend. Before you tweak your AWS EC2 or Azure VM families, you need to understand your workload's behavior at a granular level.

In this post, we'll unpack the essential telemetry you need to collect and analyze before modifying instance types. Drawing on tools like AWS Compute Optimizer and Azure Advisor, we'll also cover the nuances in how providers define CPU, memory, and network resources — and why simple averages don't cut it.

Why Telemetry Matters: The Hidden Cost of Assumptions

When teams think about instance sizing changes, it's often based on simplistic metrics—like average CPU utilization or a vague feeling that "there's too much idle time." This is especially problematic for always-on small services that quietly consume resources 24/7. Small inefficiencies multiply quickly across global fleets.

Additionally, cloud providers don't define CPU or memory equivalently, especially when it comes to burstable or shared CPU instances. Without good telemetry, you might interpret a "2 vCPU" Azure VM as equivalent to a certain AWS instance—but the performance profiles can differ widely.

From Averages to Distributions

Average utilization tells you so little about performance spikes or idle phases. What you really want to know is how utilization behaves at the 95th or 99th percentile — those spikes that might require higher baseline capacity to avoid throttling or latency issues.

Let's dive into what metrics you should collect, and how to make sense of them.

Core Metrics to Collect Before Touching Instance Types

Before making any instance family or size changes, collect a multi-dimensional space of telemetry:

- **CPU distribution**
- **Memory usage patterns**
- **Network utilization**
- **Spike duration and frequency**

CPU Distribution

CPU demand fluctuates wildly during normal operations. Instead of "average CPU = 30%," you need to capture the entire distribution and observe:

- **P95 and P99 usage:** What does the CPU utilization look like at the top 5% and 1%? Are there brief spikes that your app cannot tolerate?
- **Spike duration:** Is high CPU lasting seconds, minutes, or hours? Very short spikes might be tolerable in burstable instances but long-lasting spikes may cause throttling.
- **Baseline with idle periods:** What percentage of time is the instance near idle? This predicts waste potential.

Using top or htop and metrics from CloudWatch (AWS) or Azure Monitor can give you high-resolution CPU usage metrics. But be sure you're collecting enough data points per minute to see sharp spikes.

Memory Usage Patterns

Memory can often be overlooked in cost reviews, but it's equally important:

- **Peak vs average:** An app might run at 50% average memory but hit spikes that cause swapping or OOM errors.
- **Working set stability:** Does memory usage fluctuate, or is it mostly stable at a high watermark?
- **Cache and buffer usage:** Are caches holding dynamically sized data that influences memory demands?

Tools like vmstat or free, plus cloud-native memory metrics, can give insight here. Azure Advisor also analyzes memory pressure patterns to recommend right-sized VMs.

Network Utilization

Network bandwidth and throughput can become chokepoints or cost centers:

- **Data ingress vs egress:** Many cost estimates ignore egress charges, which can be substantial.
- **Peak network usage:** Does your service require sudden bandwidth spikes that smaller instances can't sustain?
- **Packet loss, latency, errors:** High network errors might indicate the need for a different instance with better NIC performance.

Metrics can come from cloud provider monitoring or embedded network monitoring tools (e.g., VPC flow logs in AWS).

The Importance of the Right Observation Window

Collecting telemetry over too short a period gives an incomplete picture, while over long a period may hide important transient behavior.

Here's what I recommend:

1. **Start with at least 7 days of continuous monitoring.** This captures weekday/weekend patterns, backup windows, and batch jobs.

2. **Granularity of 1-minute or finer.** Longer intervals mask spikes and burst behavior.
3. **Cross-reference deployment events.** Identify whether performance anomalies align with deployments or code paths.

Only with the right observation window can you confidently map CPU, memory, and network peaks to real workload demands.

What AWS Compute Optimizer and Azure Advisor Tell You (and What They Don't)

Both **AWS Compute Optimizer** and **Azure Advisor** provide automated instance sizing recommendations by analyzing historical utilization data. They are excellent starting points but have caveats:

Feature AWS Compute Optimizer Azure Advisor Metric Inputs CPU, memory (optional), EBS throughput, network CPU, memory, network, disk IOPS Observation Period Last 14 days by default at 5-minute granularity Last 7 days at 1-minute granularity (varies) Output Right sizing recommendations at instance family/size level Right sizing, overprovisioned/underprovisioned flags Customizable Thresholds Limited Basic customization

But a key pitfall is reliance on averages or CPU utilization only. Compute Optimizer's "recommendation" might downgrade instance size even if your P99 CPU spikes are very high but brief. Likewise, Azure Advisor might miss workload-specific latency criteria.

Always Cross-check with Raw Telemetry

Before acting on these automated recommendations, export the raw utilization data and plot:

- CPU and memory percentiles over time
- Spikes and spike durations
- Network throughput peaks and error rates

Use dashboards in CloudWatch, Azure Monitor, or third-party tools like Grafana and Kibana for this purpose.

Shared CPU Instances are Not Created Equal

The definition of shared or burstable CPU varies widely across cloud providers:

- **AWS T-series:** Uses CPU credits to burst beyond baseline CPU; sustained high CPU leads to throttling.
- **Azure B-series:** Accumulates CPU credits but with different measuring intervals and burst policies.
- **Google Cloud E2 micro:** Shared CPU cores with no guaranteed baseline, can cause unpredictable latencies.

Ignoring these differences results in wrong assumptions. A "2 vCPU burstable" instance on AWS doesn't equal "2 vCPU" on Azure in terms of burst capacity or baseline throughput.

Collect CPU usage distributions and spike duration to understand if your workload can safely operate on shared CPU instances or it demands dedicated or "fixed" CPU allocations.

Putting It All Together: A Checklist Before Changing Instance Types

1. **Collect 7+ days of telemetry at 1-minute granularity, capturing CPU, memory, and network usage.**
2. **Analyze CPU utilization distributions (P50, P95, P99) and spike durations. Don't depend on average CPU alone.**

3. **Validate memory peak usage patterns and swap activity, especially for workloads prone to out-of-memory kills.**
4. **Understand network throughput peaks, egress costs, and latency metrics.**
5. **Use AWS Compute Optimizer and Azure Advisor as baseline recommendations, but cross-check with raw telemetry.**
6. **Accounting for provider-specific definitions of CPU and sharing models before downgrading or switching instance families.**
7. **Define rollback criteria based on error rates, latency spikes, or above-threshold CPU usage during pilot tests.**

Summary

Cloud instance type changes are powerful yet risky moves in optimizing infrastructure performance and cost. Hand-wavy or average-based telemetry leads to mistakes that cause downtime or hidden waste.

Focus on observing distributions, spike durations, and cross-dimensional telemetry — CPU, memory, and network — over a sufficiently long window. Understand how your cloud provider's CPU definitions and burst mechanisms impact your workload behavior.

Use automated tools like AWS Compute Optimizer and Azure Advisor as guides, but always validate their recommendations against detailed telemetry before hitting "resize."

In short: *measure before you resize*.