When contemplating a move to shared CPU instances in the cloud, engineers often ask: "What metrics and observations should I collect to gauge success and catch hidden costs?" The answer is—not surprisingly—nuanced. Shared CPU offerings differ widely between AWS, Azure, and other providers, and your typical reliance on averages or simple utilization can miss the real story.

In this post, I'll walk through the key recording and measurement strategies that have worked in my 12+ years running cloud cost reviews and infrastructure migrations. I cover why **always-on small services** tend to hide subtle cloud waste, how different shared CPU models impact performance, and what precise measurements you should use—especially regarding *latency metrics*, *queue depth*, and *error rates*. Plus, we'll lean on tools like AWS Compute Optimizer and Azure Advisor to build a solid baseline and validate post-pilot results.

Why Small, Always-On Services Often Hide Cloud Waste

It's tempting for organizations to overlook small internal tools, development staging environments, or lightweight background workers when hunting for cost savings. These "always-on small services" frequently run on over-provisioned or unnecessarily dedicated CPUs that rack up cloud spending silently.

Problems arise because:

- **Low average CPU doesn't mean they are idle:** Many of these services have unpredictable spikes requiring capacity bursts which customarily remain under the radar when only average CPU utilization is considered.
- **Shared CPU instances can be ideal here:** With proper measurement, the cost-performance sweet spot is often a burstable or baseline-shared vCPU model, but only after validating latency and error impacts during bursts.
- **Always-on nature ensures continuous consumption:** Even small waste across hundreds of such services aggregates to significant monthly cloud spend.

A solid pilot and measurement plan is your key to unlocking these savings confidently.

Understanding Shared CPU Definitions Across Cloud Providers

Before touching any instance types, know how *shared CPU is defined* by your cloud provider. Most people just see "2 vCPUs" and assume uniform performance guarantees — this is where many get tripped up.

Provider Common Shared CPU Type CPU Allocation Model Performance Guarantees AWS T4g / T3 / T2 (Burstable) Baseline CPU + CPU Credits; bursts until credits exhaust No sustained CPU guarantee, bursting up to full vCPU core Azure B-Series Credit-based baseline CPU and bursting; CPU credits accumulate when idle Baseline guaranteed with ability to burst; credit exhaustion leads to throttling GCP E2 Shared-core Fractional vCPU shared physical core; no bursting credits, contention-based Best-effort CPU available; no guaranteed CPU performance

Key takeaway: Do not vary instance sizing and performance expectations based solely on nominal vCPU counts. Interpret shared CPU differently based on baseline https://computingforgeeks.com/shared-cpu-cloud-waste-migration-guide/ and bursting models or physical sharing with contention.

Measure Peaks Using the Right Observation Window

The cardinal sin when evaluating shared CPU pilots is to trust average CPU or resource statistics over a long window. Average metrics mask peak bursts responsible for latency spikes and errors.

To properly measure the impact on your services, adopt the following best practices:

- **Use fine-grained intervals:** CPU and latency data collected at least every 1 minute or down to 10s if possible to detect spikes.
- **Match windows to workload burst patterns:** For queue workers and HTTP services, observe over 15m, 30m, and 1h windows to capture variability due to load fluctuations.
- **Focus on peak windows:** Align CPU usage peaks with latency and queue build-up during those peak times.

When running your pilot, confirm that your monitoring back-end supports querying at the desired granularity and that no data smoothing eliminates crucial spike information.

Use Percentiles and Spike Duration — Not Averages

Average latency and CPU only tell half the story. We need to quantify worst-case scenarios and their frequency since your users and SLIs care deeply about tail latency and error bursts.

Here's how to approach summarizing your metrics before and after the pilot:

1. **Compute P95 and P99 latency metrics:** These percentile latencies highlight tail behavior where shared CPU contention most hurts.
2. **Track error rates during spikes:** Look for correlation between rising CPU peaks and elevated HTTP 5xx or queue processing errors.
3. **Measure queue depth dynamics:** Growing backlog length during CPU saturation shows visible degradation beyond raw latency.
4. **Determine spike durations:** Monitor how long latency and errors stay high, e.g., spikes > 30 seconds or sustained CPU throttle windows.

Before running any instance type switch, ask: " *What do my P95 and P99 latency curves look like at peak CPU and queue depth times?*" Without this, you can hit a hidden cost in team justifiably frustrated by sporadic slowdowns.

Recommended Metrics to Record Before and After Your Shared CPU Pilot

Here is a comprehensive checklist of metrics to capture to evaluate the quality and safety of switching a workload to shared CPU instances.

Metric Category Metric Name Notes Example Tools CPU Usage CPU Utilization P95 / P99 Use fine-grained intervals (1m or less). Identify sustained bursts. AWS CloudWatch, Azure Monitor CPU Credits / Burst Balance Credit balance (until exhaustion) Only applies to burstable (AWS T, Azure B series). Ensure credits sustain bursts. AWS CloudWatch, Azure Monitor Latency P95, P99 Latency (End-to-end request or processing latency) Tail latencies indicate impact of CPU contention. Application telemetry (OpenTelemetry, X-Ray), Datadog APM Queue Depth Length of worker or request queue Key for backend services with async workloads; growing queue = insufficient CPU. Custom metrics, CloudWatch, Azure Monitor Error Rates HTTP 5xx / processing error rate over time Correlate error spikes with CPU contention. Application logs, Datadog, Azure Monitor Compute Optimizer / Advisor Recommendations Instance sizing and cost recommendations Compare recommended right-sizing before and after pilot. AWS Compute Optimizer, Azure Advisor

How to Leverage AWS Compute Optimizer and Azure Advisor for Your Pilot

Both AWS and Azure offer automated tooling to audit your fleets and suggest right-sizing opportunities. Use them both as a starting point and after your pilot concludes to gauge impact.

AWS Compute Optimizer

This service analyzes instance metrics including CPU, memory, and network to recommend ideal instance types. While useful, it often oversimplifies burstable instances because it relies heavily on average CPU.

- **Before pilot:** Run Compute Optimizer to flag clearly oversized instances and evaluate burstable eligibility.
- **Pay attention to:** CPU credit exhaustion alarms combined with the recommendations.
- **After pilot:** Check if your pilot usage reduces average consumption and cost while maintaining service quality.

Azure Advisor

Azure Advisor similarly analyzes VM performance metrics and recommends resizing, including suggestions for B-series burstable types.

- **Before pilot:** Review Azure Advisor recommendations for your workloads focusing on CPU and IOPS insights.
- **Check:** Whether your workload has consistent baseline utilization patterns or highly variable burst patterns.
- **After pilot:** Confirm that moving to burstable instances hasn't increased throttling or error rates.

Defining Rollback Criteria Before You Run Your Pilot

Before deploying the first shared CPU instance in production or staging, define concrete rollback triggers based on your latency and error targets. Examples include:

- P99 latency increase > 20% sustained for more than 5 minutes during peak load
- Error rate spike > 1% lasting longer than 15 minutes
- Queue depth doubling compared to baseline under steady load
- Explicit CPU credit exhaustion more than 3 times per day (if burstable model)

If any of these criteria are met, automate alerts and roll back to dedicated or higher-class CPU instances before impacting end-users.

Summary and Recommendations

Switching to shared CPU instances offers a compelling path to reducing your cloud spend on many small, low-utilization but always-on services. However, success depends on a disciplined measurement approach that respects the nuances of shared CPU models:

- **Capture fine-grained, time-aligned CPU utilization with latency and error metrics.**
- **Analyze tail latencies (P95/P99) and queue depths over short windows to detect peak stress.**
- **Understand how your provider's shared CPU model impacts bursting and throttling.**
- **Leverage AWS Compute Optimizer and Azure Advisor reports as starting points—not blind directives.**
- **Define clear rollback criteria upfront and monitor during pilot runs to prevent regressions.**

By measuring the right things, at the right resolution, before and after your pilot, you'll avoid common pitfalls and build a scalable, cost-efficient cloud footprint without surprising your users or ops teams.