When planning a cloud migration or cost optimization initiative, a critical early decision is defining the **pilot scope**. Should you start with a single service or test a whole fleet of applications? While it might be tempting to begin small and simple, choosing the right scale for your pilot can make a significant difference in uncovering hidden inefficiencies, avoiding costly surprises, and enabling a *controlled migration*.

In this post, I'll share lessons learned from dozens of migration pilots across AWS, Azure, and Google Cloud environments. We'll cover:

- Why **always-on small services** can mask cloud waste
- How **shared CPU definitions differ by cloud provider** and why that matters
- The importance of measuring **peaks with the right observation window**
- Using **percentiles and spike duration**, not averages, to set instance sizes
- Leveraging native tools like **AWS Compute Optimizer** and **Azure Advisor** effectively

The Pitfall of Always-On Small Services in Cost Optimization

One of the most common blind spots I encounter during cost reviews is the influence of small, always-on infrastructure components—internal tooling, APIs, or microservices running 24/7 but serving very low throughput. These may be a handful of tasks or services that don't register as significant CPU or memory users on average, but their impact compounds when multiplied across many teams or environments.

These small services are usually provisioned conservatively with general-purpose instances or shared CPU offerings and often overlooked during downsizing proposals. The problem? Because they rarely generate sustained high CPU, their average utilization looks "reasonable," but their *P95* and *P99* CPU usage can reveal periodic bursts that require more overhead than expected.

Considering just one service in isolation—especially if it's small and barely loaded—can hide this cloud waste. If your pilot exclusively covers such a service, optimizations will look marginal and may not generalize well to your fleet. On the other hand, a **representative workload** that includes both always-on infrastructure and bursty, high-throughput services provides clearer signal for sizing and migration decisions.

Shared CPU Definitions Differ by Provider—Know Your Baseline

A major source of confusion in cloud migrations comes from interpreting vCPU counts across AWS, Azure, and Google Cloud. The temptation is to treat vCPU as a fixed unit of guaranteed CPU, but this analogy doesn't hold up:

Cloud Provider vCPU Definition Shared CPU Instance Type Examples Implications for Pilots AWS One vCPU = one hyperthread of a dedicated physical core (in most cases) T3, T4g (burstable instances with CPU credits) vCPU quota can burst; sustained usage depends on CPU credits. Average CPU usage can mask real limitations during spikes. Azure vCPU is a logical processor on a dedicated core, but bursting exists in some series (B-series) B-series (burstable), D-series (dedicated) Bursts allowed on B-series but require understanding of accumulated credits and burst duration. Google Cloud Each vCPU is a core assigned by the hypervisor E2 instances (shared CPU), N1 high-CPU (dedicated) Shared CPU instances may contend with other tenants; performance more variable especially during bursts.

When running a pilot, be wary of equating "shared CPU" with inherently bad performance or uptime. Instead, interpret CPU usage in the context of burst credit availability, contention, and your application's tolerance for

short-term throttling.

https://computingforgeeks.com/shared-cpu-cloud-waste-migration-guide/

Measure Peaks with the Right Observation Window

A flawed assumption often derails sizing decisions: basing them on average CPU utilization across a long observation window. The error here is equating average load with actual performance demand.

Imagine a worker queue: it sits idle for most of the day, jumping to 90% CPU when a batch job runs every hour for a few minutes. The day's average CPU usage might be 10%, but the spikes require an instance capable of handling 90% load without throttling.

How to Choose the Right Observation Window

- Capture at least a full representative workload cycle. For scheduled batch jobs, this may be 24 hours or longer.

- Consider peak windows in the last 7-14 days for seasonal or weekly batch jobs.
- If possible, validate with synthetic load or traffic-replay tools to force peak behavior.

In other words, don't base your pilot sizing only on a week's average usage. Identify and analyze *spike durations and magnitudes* relevant to your workload.

Use Percentiles and Spike Duration, Not Averages

For engineering teams, the key question is: *how high does CPU usage get, for how long, and with what frequency?* This is what percentile metrics reveal:

- **P95 CPU usage:** The CPU baseline you can expect during 95% of your time—so 5% of the time it spikes beyond.
- **P99 CPU usage:** The CPU level during the rare but impactful spikes.

When setting instance types or resource requests, plan for steady-state capacity near P95 levels and burst capacity to handle P99 spikes without slowdowns. Also, measure *spike duration*—there's a big difference between a 1-minute spike and a 10-minute sustained peak.

Long sustained peaks require dedicated capacity, while short spikes can be tolerated by burstable instances or auto-scaling mechanisms.

Leveraging AWS Compute Optimizer and Azure Advisor in Your Pilot

Tools such as **AWS Compute Optimizer** and **Azure Advisor** offer insightful recommendations but should not be your only source for sizing decisions.

AWS Compute Optimizer

- Analyzes multiple metrics like CPU, memory, disk I/O, and network to recommend right-sizing.
- Supports viewing recommendations based on different time windows (1 day, 3 days, 14 days).
- Provides instance refresh fleet recommendations to optimize cost savings.

The key is to review the Compute Optimizer's *observation window* configuration and confirm it aligns with your business cycles and spike durations.

Azure Advisor

- Analyzes resource configurations, utilization, and best practices.
- Offers cost optimization tips including resizing VMs, removing idle resources, and improving networking.
- Includes integration with Azure Monitor for detailed metrics analysis.

For both tools:

- Use them as a starting point, not the final word.
- Complement suggestions with manual percentile-based metric analysis.
- Validate recommendations via pilots that simulate production load spikes.

Best Practices for Defining Pilot Scope

1. **Include a representative workload across your service types:**

- Always-on small services (internal tools, APIs)
- Batch worker queues with known spike patterns
- High-throughput front-end services

This avoids bias from outliers and hidden cloud waste.

2. **Define clear rollback criteria before starting:**
   - Acceptable P95 and P99 latency and error rates
   - Thresholds for CPU throttling and resource contention
   - Cost thresholds including storage and egress—not just compute
3. **Use detailed observation windows aligned with workload cycles:**
   - Match monitoring granularity to spike duration
   - Trigger additional monitoring or synthetic tests during expected high load
4. **Leverage multiple data sources:**
   - Cloud provider tooling + external monitoring + logs
   - Percentiles and spike detection are essential
5. **Iterate and scale pilot scope carefully:**
   - Start with a subset but plan to add adjacent services incrementally
   - Test fleet-wide migrations or optimizations only after pilot success

Summary

In defining your migration or optimization pilot scope, avoid the temptation to start too small or to chase average utilization numbers. Always include a **representative workload** that captures subtle but costly cloud waste from always-on small services, understands the nuances of **shared CPU in your cloud provider**, and leverages percentile-based CPU utilization and spike duration metrics to set reliable capacity.

Tools like AWS Compute Optimizer and Azure Advisor can guide your sizing but only when paired with a detailed analysis of peak workloads and a carefully designed observation window. Finally, define rollback criteria upfront and view the pilot as a controlled migration that informs fleet-wide decisions with confidence.

Following these best practices will help your team build a cost-effective, reliable, and scalable cloud environment instead of chasing endless surprises after a “pilot” that wasn’t representative.