How Well Do You Understand the Costs of Your Kubernetes Workloads?

Find the waste. Tune the spend.
Keep the performance.

splunk>
a CISCO company

Introduction

Kubernetes has changed the way organizations build, ship, and scale applications. It offers flexibility and speed, allowing teams to iterate quickly and respond to demand. But that speed introduces complexity. As environments scale and shift constantly, it gets harder to understand what's running, why it matters, and what it costs. In fact, Gartner forecasts worldwide public cloud end-user spending to total $723 billion in 2025.

Teams collect mountains of telemetry. According to Splunk's The New Rules of Data Management report, 91% of organizations surveyed say their overall spend on data management has increased compared to the previous year. They track system health and uptime, and collect observability data from every corner of the environment. But without context, all that data becomes expensive noise. That context is often missing in Kubernetes environments, where resources are short-lived and scaling happens automatically. As a result, teams are left guessing about resource usage, performance bottlenecks, and cost drivers instead of taking action.

The result? Metrics everywhere, but no clear next move. Teams watch the dashboards but still don't know what to fix or where to start. Observability bridges that gap by correlating telemetry signals with cost and performance data. It ties usage to cost, reveals what's working and what's wasteful, and helps teams focus their efforts where it counts.

Decisions about how to deploy and scale Kubernetes workloads come with tradeoffs. Overprovisioning can keep services stable, but it wastes budget. Underprovisioning cuts costs, but risks downtime and degraded performance. Practitioners need more than gut instinct. They need clear insight into how workload configurations and scaling decisions affect performance, reliability, and cost.

This e-book breaks Kubernetes cost management into three practical steps: **gaining visibility** into what's running, **optimizing high-impact workloads** where performance, cost, and reliability tradeoffs are most significant, and **staying ahead of drift** as things change. Observability enables each of these by tying resource usage to cost, surfacing inefficiencies, and tracking how changes affect performance over time.

For a deeper dive into troubleshooting and remediation strategies in Kubernetes, see our companion guide, Troubleshooting Kubernetes Environments.

Observability's critical role in managing Kubernetes performance and cost tradeoffs allows teams to:

- Understand the health of applications and services
- Identify cost drivers and inefficiencies
- Make informed tradeoffs between performance and spend

The illusion of control

Many teams believe they have a handle on their Kubernetes environments. They can see what their services are doing, monitor uptime, and track resource usage within their own silos. But beneath the surface, operational blind spots are everywhere.

In many organizations, teams work independently with their own tooling, standards, and observability practices. This decentralized model can lead to overlapping resources, missed signals, and uneven performance. And even when teams trust their own view, no one has a clear picture of the entire system.

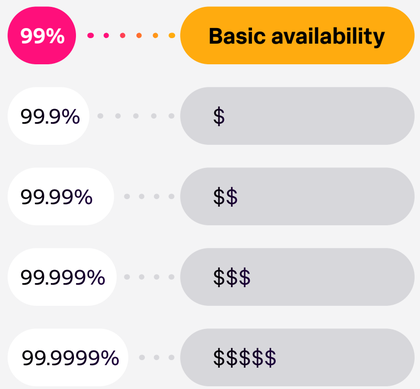
This fragmented visibility leads to what some call **invisible infrastructure**. These are workloads, services, or clusters that quietly consume resources without anyone questioning their efficiency or value. These gaps often go unnoticed until costs spike, performance drops, or an outage forces a deeper investigation.

In the early days of cloud computing, **shadow IT** took hold quickly. Developers didn't wait for approvals. They just swiped a credit card and spun up what they needed. That freedom sped things up, but it also created real problems. Teams ended up with duplicate environments, surprise costs outside the official budget, and security gaps that flew under the radar. With no central visibility, it was hard to know what was running, where, or why. It worked for a while, until it didn't.

Kubernetes makes it easy to deploy and scale fast, but just like with early cloud sprawl, what's running and why often slips through the cracks. Without shared standards or visibility, teams can't track usage or spot inefficiencies until it's too late.

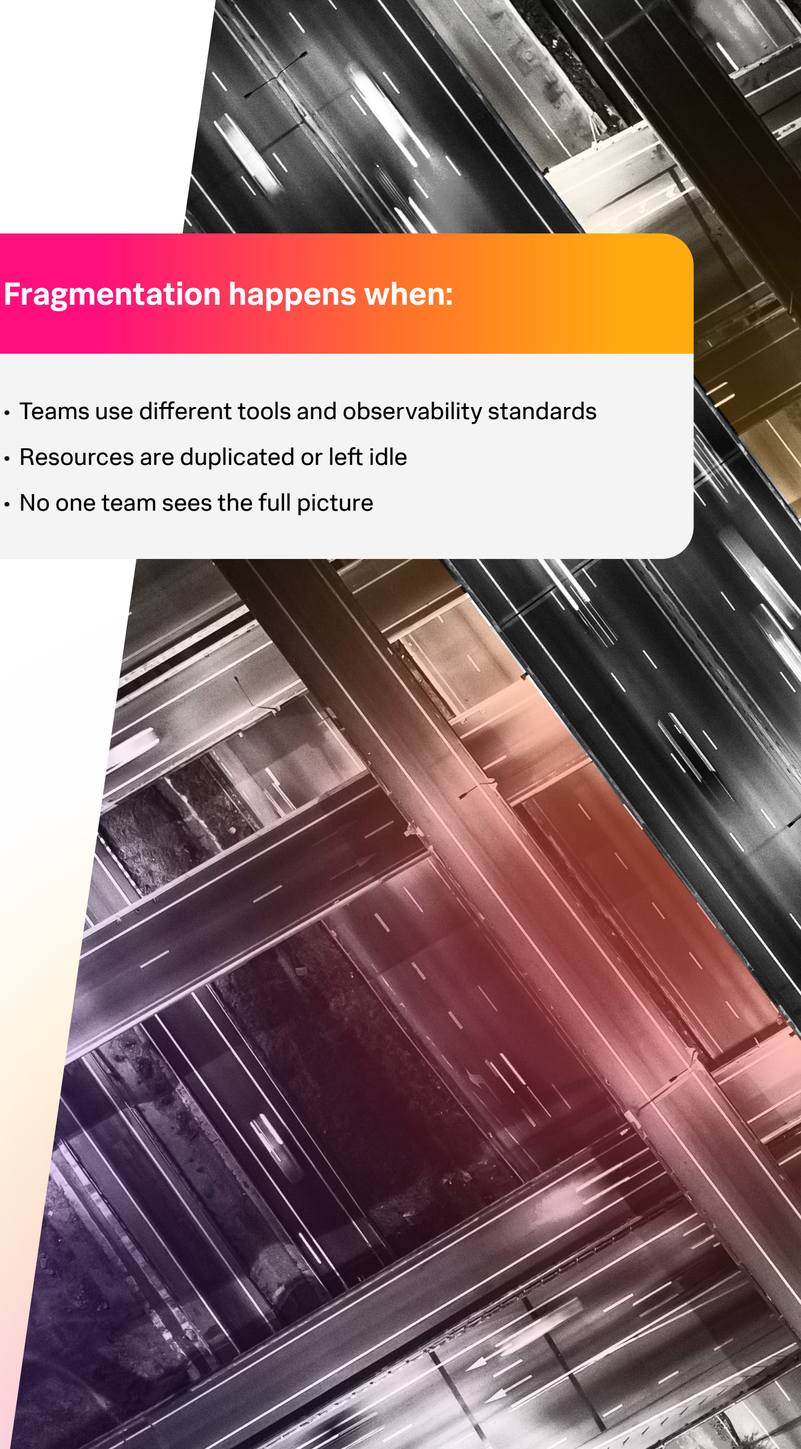
Platform engineering has emerged to help centralize tools, enforce standards, improve the developer experience, and promote cross-team collaboration. Platform teams reduce fragmentation and give engineering leadership a clear view of what's running, where it's happening, and of its impact.

So here's the real question: Do you know what your Kubernetes workloads are actually costing you? Some may be starving for resources. Others are quietly draining budgets more than necessary. **And in any part of your stack, every extra "9" of uptime has a price tag.** The real challenge is knowing when that spend delivers value and when it doesn't.



Fragmentation happens when:

- Teams use different tools and observability standards
- Resources are duplicated or left idle
- No one team sees the full picture



Step 1

Visibility: See the whole picture

Understanding your Kubernetes environment starts with clear, reliable visibility. That means going beyond system health checks and uptime metrics. It means knowing how resources are being used, who owns them, and how they contribute to cost. Without this level of insight, optimization remains out of reach.

Step one is seeing where spend is actually going. Teams need to identify which workloads are responsible, how costs break down across teams or environments, and whether resource requests match real usage. This kind of clarity often reveals overprovisioned clusters or short-term spikes that lead to long-term overprovisioning. These spikes typically lead to increased costs by requiring teams to maintain excess capacity or by triggering expensive autoscaling actions.

Cost attribution and forecasting make this visibility actionable. Cost attribution ties usage to the teams or services responsible. Forecasting projects future spend based on usage patterns. With both, teams move from reacting to costs to planning with confidence. They can budget more accurately and scale based on actual needs, not guesswork.

Let's say a DevOps team spots an unexpected spike in cloud spend tied to a specific namespace. Using a Kubernetes-aware observability tool, they trace it to a long-forgotten test service that kept scaling. It turns out the workload was never shut down. With cost attribution and efficiency metrics, they can pinpoint the issue, retire the workload, and prevent the overage from repeating.

Cost attribution shows what's running, who owns it, and what it's costing. When teams can trace services back to real owners, it's easier to clean up what's no longer needed. Instead of chasing mystery workloads, they can follow the trail and fix what's out of sync. That might mean tweaking usage, trimming spend, or shutting things down altogether.

This kind of issue is common in environments where developers and engineers have autonomy to scale services independently. Without centralized reporting or budget tracking, unnecessary spend can persist for weeks or even months. Visibility at the workload level, tied to team or application ownership, is key to preventing this.

Effective visibility means that systems and their data are:

- **Accessible:** Everyone can view the data they need
- **Contextual:** Insights are tied to real usage or cost
- **Actionable:** Teams know what to do next

Forecasting helps teams stay ahead of change. By looking at past usage, engineering and operations teams can identify trends, seasonal spikes, and outliers. That makes it easier to set limits, plan capacity, and spot problems early before they become expensive. When you know your infrastructure's baseline behavior, it becomes much easier to detect when something is off.

Signals such as starvation risk, efficiency rate, and idle resource indicators help surface hidden inefficiencies. These signals are especially useful when aggregated and prioritized, helping teams focus on the most urgent or high-impact opportunities for optimization. Some platforms even summarize this at the cluster or namespace level to highlight what needs attention first.

It is not enough to just monitor. **Visibility** should make information accessible, contextual, and actionable. A dense dashboard with hundreds of charts may look comprehensive, but without clear ownership and prioritization, it only adds to the noise.

Visibility also supports accountability. When teams have access to cost breakdowns, efficiency trends, and resource utilization tied to their services, they are better equipped to make informed tradeoffs. A team might realize that their aggressive autoscaling policy is keeping reliability high but at an unsustainable cost. With the right data, they can evaluate whether performance and spend are in balance.

At the heart of all this is shared understanding. Visibility tools should help engineering, operations, and business teams align infrastructure usage with business goals and budget realities. That means having honest conversations about expectations, workload performance, and where optimization efforts will make the biggest difference. When teams rely on the same signals and insights, it's easier to align priorities, reduce waste, and work better together.

Starvation Risk: This signal indicates the likelihood that a workload will run out of CPU or memory resources based on its historical usage patterns, potentially leading to performance degradation, errors, or instability.

Efficiency Rate: This metric measures how much of the resources a workload has requested or been allocated it is actually using. A low efficiency rate means you're paying for resources that the workload isn't consuming, highlighting waste.

Idle Resource Indicators: These point to resources (such as nodes, pods, or specific resource requests) that are provisioned and consuming cost but are not actively being used or are significantly underutilized over a sustained period.

Step 2

Optimization: Find the balance

Once teams achieve clear visibility into their Kubernetes environments, the next challenge is turning insights into smart, measurable improvements. Rather than reacting only to cost spikes or performance issues, teams can create a repeatable process to fine-tune workloads with purpose.

Think of this tension as a see-saw. On one end is **performance**. This includes responsiveness, availability, and elements of resilience, often measured in levels of uptime or service expectations. On the other is **cost control**. Overprovisioning keeps services stable, but wastes compute and memory. Underprovisioning saves money, but risks latency, degraded performance, or outages.

The only way to strike the right balance is with context: How important is this workload? Who depends on it? What level of performance is acceptable, and what are you willing to spend to maintain it?

In organizations without clear optimization signals, decisions about scaling and tuning are often driven by habit, fear, or pressure to avoid outages. Teams might default to doubling CPU and memory allocations rather than risk being underpowered. But that approach scales poorly, especially in large environments where small inefficiencies add up quickly.

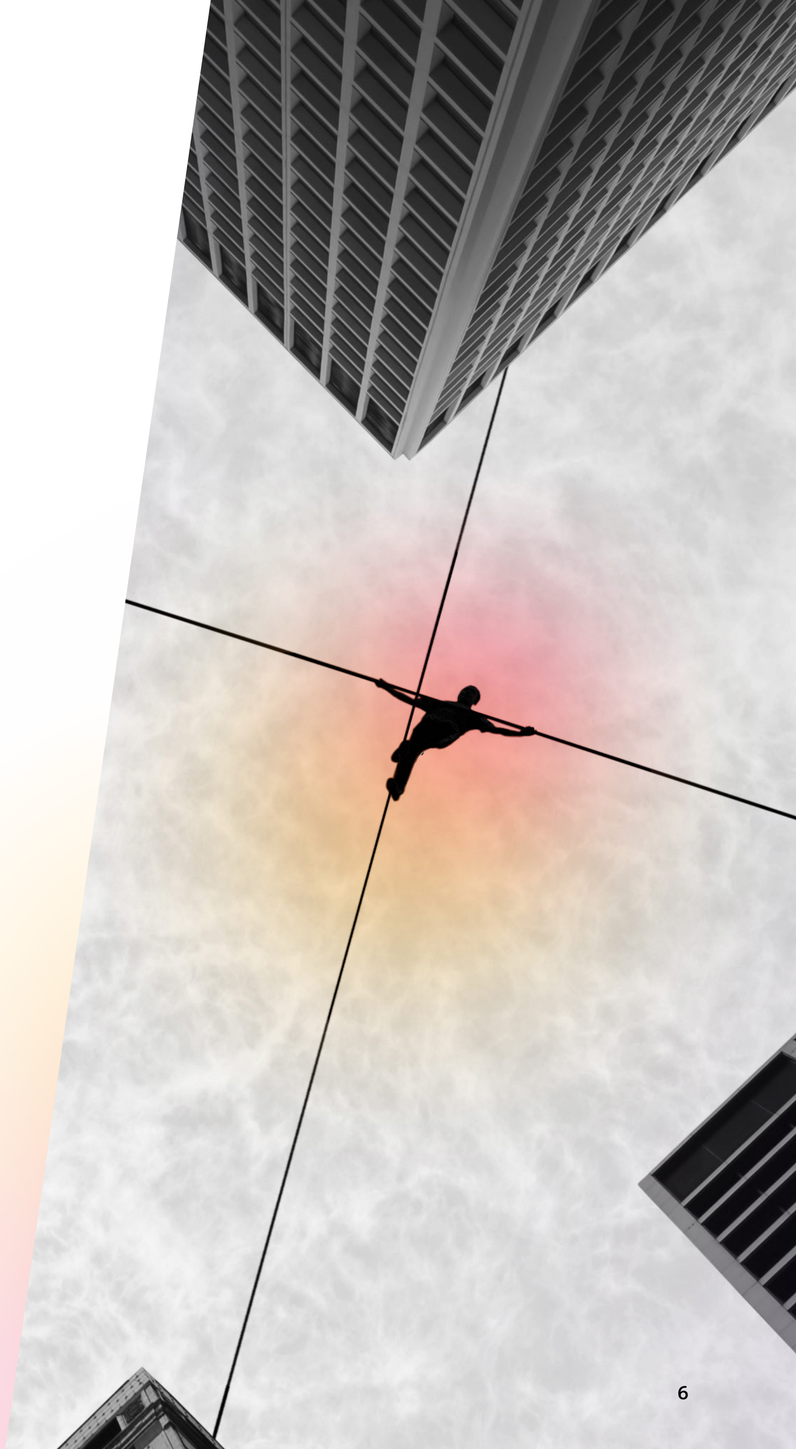
Historical workload behavior provides critical context for better decision-making. Teams can analyze usage patterns, peak load times, and past incidents to shape more thoughtful scaling policies. Platforms that incorporate this kind of data can provide recommendations or risk indicators that help prioritize optimization efforts.

For example, a team managing a batch processing workload might realize that their job runs daily with significant idle time in between. Instead of maintaining fixed resources around the clock, they can configure the service to scale down after execution and re-provision only when needed. These decisions rely on a blend of observability, historical trends, and confidence in automation.

Some platforms now provide **optimization recommendations** based on actual usage over time. These recommendations suggest right-sizing workloads to align requested CPU and memory with real-world demand. They also help flag workloads with high starvation risk, where services are likely to struggle to get the resources they need, or those that are simply oversized.

Ask yourself:

- Is this workload mission-critical?
- Are we paying for performance that's unnecessary?
- Can we consolidate or retire underused services?



Prioritization is key. In large Kubernetes environments, teams cannot optimize everything at once. Observability tools that identify the biggest opportunities, such as workloads that are significantly underutilized or contributing disproportionately to spend, help focus engineering time where it matters. In smaller environments, teams may be able to address optimization more directly, but strong signals still help them move faster and make better decisions.

Optimization means adjusting based on how infrastructure is actually being used. The goal is to make sure every workload is worth what it costs. Teams should ask themselves: Are we paying for performance that users notice? Can we maintain reliability with fewer resources? Are there services we can consolidate or shut down?

Balancing reliability and efficiency requires collaboration. SREs, developers, and platform engineers need a common view into workload behavior and tradeoffs. Without shared context, tuning efforts can become misaligned or stall altogether.

Application tiering can also help guide optimization efforts. Not every service can be treated as mission critical. Teams can categorize workloads by business impact, from essential systems that require high availability to best-effort services that can tolerate occasional disruption. This allows for smarter tradeoffs based on criticality. High-priority applications might justify higher spend for performance and redundancy, while lower-tier workloads could be controlled more aggressively.

It is equally important to recognize when optimization is not necessary. There's only so much time in your day, so it makes sense to focus on optimizing the most expensive, the most business-critical, or the most impactful to users. For example, a background reporting job that costs very little to run and has minimal user impact may not warrant immediate attention. Instead, teams can focus on services that are high-volume, customer-facing, or disproportionately expensive to operate.

Balance doesn't happen. It's built between what you need and what you can afford. It takes human judgment, clear signals from your environment, and the nerve to tune what others won't touch.

Step 3

Operation: Keep efficiency on track

Optimization is not a one-time effort. Once workloads are tuned and resources are right-sized, teams need to ensure those improvements hold up over time. Kubernetes environments don't sit still. They are shaped by constant updates, shifting usage, and evolving team priorities. These changes may seem incremental, but over time they can quietly undo the benefits of earlier tuning efforts if not tracked and managed.

The third phase of operational maturity is about **sustaining efficiency**. Without consistent attention, even well-tuned workloads gradually drift toward inefficiency. That drift might result from evolving usage patterns, added features, or infrastructure changes elsewhere in the environment. Sustaining gains requires a combination of intelligent alerting, historical context, and ways to spot issues before performance drops or costs become unpredictable.

Anomaly detection plays a critical role here. When costs spike or resource usage changes unexpectedly, teams need early signals. This might include alerts when a service suddenly consumes more memory than its baseline or when spend across a namespace exceeds projected thresholds. These warnings can prompt timely reviews before issues grow into bigger problems.

To make this process manageable, teams need ways to filter noise and focus on meaningful trends. Alert fatigue is real, especially in complex systems. Platforms that highlight high-impact signals, like declining workload efficiency or rising starvation risk, make it easier for teams to focus on what needs attention first.

Budgets and cost thresholds can also help reinforce **accountability**. By assigning budgets to teams or projects, organizations create natural checkpoints. If a team exceeds its expected spend, it prompts a conversation; not as punishment, but as a trigger to re-evaluate deployment decisions or usage policies.

Some platforms now assign **scores** or ratings to workloads, namespaces, or clusters based on resource usage, risk signals, and alignment with efficiency targets. These scores help teams understand where to focus next, especially in large environments with limited bandwidth. In addition, tracking by label or namespace makes it easier to assign ownership, spot trends, and ensure usage aligns with organizational priorities. This structure enables broader conversations about policy enforcement, automation, and resource governance.

Sustaining optimization requires:

- Ongoing alerting and monitoring
- Periodic reassessment of workloads
- Cross-team collaboration and ownership

Policy-driven frameworks make it easier to maintain consistency. For example, if a platform detects that a new workload is exceeding standard CPU requests, it can flag the deployment for review or, in stricter environments, prevent it from launching. These checks help ensure teams follow shared guidelines, identifying potential issues early to maintain efficiency and support development velocity.

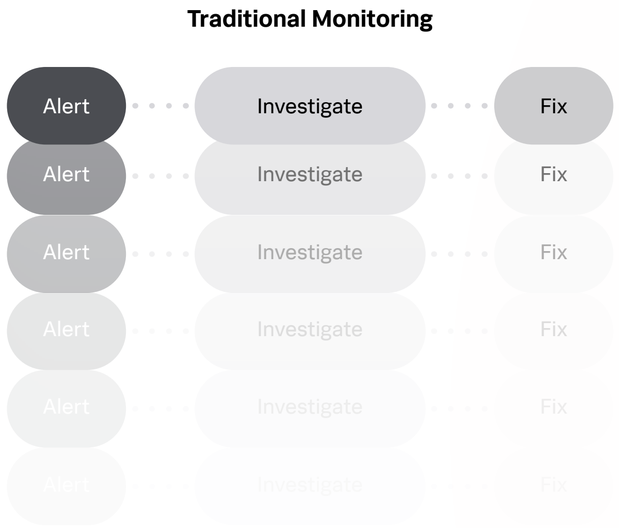
Efficiency over time depends on **continuous feedback**. That feedback should be timely, targeted, and tied to the metrics that matter. Teams need clear, consistent indicators to understand how well they are maintaining efficiency. The system should help guide that awareness through summaries, comparisons, and insights.

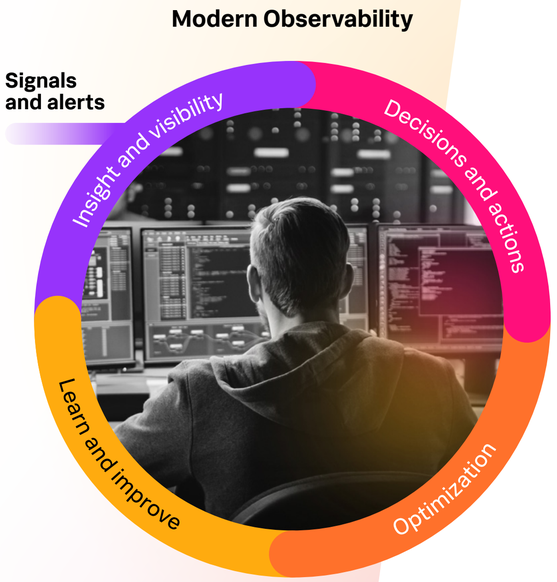
Let's consider this scenario: a team optimized its most expensive service last quarter. Without continued visibility, that progress can start to erode. Perhaps a team adds a new feature, adding load. And another team updates dependencies or removes autoscaling. Each change seems small, but the impact builds. Sustaining efficiency means spotting these shifts early and responding with the same focus as the original optimization.

There is also value in periodically **revalidating assumptions**. For example, workloads that were previously optimized for peak usage may now serve a different role or face different load patterns. Revisiting these decisions on a quarterly or biannual basis can help teams catch misalignments early and avoid costly oversights.

Collaboration plays a role in this phase as well. While teams are often autonomous, operational efficiency is a shared responsibility. A platform team might surface broad trends across clusters, while individual service owners bring the context needed to interpret and act. The tools and processes supporting efficiency should encourage coordination, not just visibility.

This phase of the cycle naturally leads back to the beginning. Each insight, alert, and decision feeds the next round of visibility. Observability is the thread that ties each phase of this journey together, from initial insight to long-term improvement. The teams that succeed in the long run are the ones that treat optimization not as a finish line, but as part of their daily rhythm.





Conclusion

Kubernetes has opened the door to fast, flexible, and scalable application development. But as systems grow and evolve, so does the complexity of managing them. Keeping environments efficient, reliable, and cost-aware requires more than visibility. It depends on context, operational discipline, and a commitment to continuous improvement.

This e-book has walked through that cycle, broken into three essential phases: **visibility**, **optimization**, and **ongoing operation**. Together, they form the foundation for responsible Kubernetes operations. These phases are not steps to cross off. They build on each other, reinforcing a cycle that becomes more effective as teams apply what they've learned and adapt to changing conditions.

Visibility is the first step, and arguably the most important. It enables teams to understand how resources are used, where costs originate, and who is responsible. Observability supports this by delivering the data, context, and signals needed to make sense of complex Kubernetes environments. Without it, decisions rely on guesswork.

Optimization follows with purpose. Teams tune workloads, adjust configurations, and align infrastructure to demand. In this phase, teams evaluate performance against cost in relation to the value those workloads provide.

Ongoing operation builds resilience and keeps efficiency sustainable. With policy-driven practices, anomaly detection, and shared accountability, teams can preserve optimization gains and prevent drift over time.

This cycle works because efficiency requires ongoing attention. It's a mindset rooted in continuous evaluation and adjustment, not a one-time fix. Observability, when thoughtfully implemented, creates the conditions for teams to revisit, revalidate, and re-optimize based on real usage and shifting business needs. It also supports broader challenges like budgeting across IT stacks, reducing workload silos, and making cost-optimization a continuous priority.

This approach matters now more than ever. As cloud costs continue to climb and engineering teams are asked to do more with less, knowing how your infrastructure performs and what it costs becomes a leadership issue. Whether you manage a platform, a budget, or a business outcome, you need confidence that your systems are operating with purpose and efficiency.

The organizations that lead in this space tend to share a few traits. They regularly assess workload efficiency and focus improvements on areas that drive the greatest business impact. They understand the tradeoffs at play, particularly the ongoing tension between performance and cost. Like a see-saw, that balance shifts. But with the right visibility into usage, cost, and performance, teams can manage that balance more effectively and with greater confidence.

Tooling is only part of the solution. The real value comes when decisions about infrastructure are made with full awareness of the tradeoffs among performance, reliability, and cost. Empowering teams to act on that knowledge means they can respond faster, scale smarter, and maintain control as complexity grows.

So now the question is yours to answer. The data is available, but the real challenge lies in transforming it into actionable understanding. Do you have the visibility and operational discipline to truly know what your Kubernetes workloads are costing and why, or are you allowing complexity to obscure hidden inefficiencies that silently drain your budget?

The Kubernetes cost-performance loop:

- **Visibility:** Understand cost and usage in context
- **Optimization:** Right-size based on performance and impact
- **Operation:** Maintain gains through alerting, policy, and accountability

Keep your Kubernetes journey going

You've learned how to balance cost and performance. Now dig into the details.

- Download Troubleshooting Kubernetes Environments to learn how observability helps diagnose, fix, and prevent critical issues
- Explore Splunk Kubernetes Monitoring

Start Your Free Trial of Splunk Observability Cloud. Gain deep visibility into your Kubernetes costs and performance, and identify optimization opportunities.

splunk> a CISCO company

25_CMP_ebook_How-well-do-you-understand-your-kubernetes-workloads_v6