splunk>
a CISCO company

Troubleshooting Kubernetes Environments

Shifting Gears on your Journey to Digital Resilience

How observability helps you diagnose, fix, and prevent critical issues

Navigating Kubernetes at high speed

Imagine cruising at 80 mph on a highway when warning lights flash across the dashboard. You know something's wrong, but without real diagnostics, you're left guessing. That's exactly what troubleshooting inside Kubernetes environments feels like without observability — blind, reactive, and full of uncertainty.

Kubernetes is constantly in motion — containers start and stop, workloads shift dynamically, and infrastructure scales horizontally and vertically in response to demand. When something goes wrong, "pulling over" to assess the issue isn't an option. Engineers must diagnose problems in real time and those that lack observability often don't have a clear view of what's happening under the hood.

The growing complexity of Kubernetes environments

Kubernetes has progressed beyond a simple container orchestrator into a comprehensive system for managing dynamic, distributed, and multicloud workloads. While this evolution has unlocked new efficiencies and scalability, it has also introduced unique troubleshooting challenges that traditional monitoring tools struggle to address.

- **Kubernetes workloads are often stateless and ephemeral**, leading to difficulty tracking failures, debugging transient issues, and maintaining persistent context across restarts.
- **Failures can manifest anywhere inside a Kubernetes environment** — infrastructure (nodes, storage, networking), Kubernetes control plane (API server, scheduler, controllers), and workloads (pods, deployments, applications). Each layer presents distinct challenges, making root cause analysis more complex than traditional infrastructure.
- **As Kubernetes environments evolve, new tools and modernization strategies accelerate change but also introduce unforeseen complexity** — siloed solutions, multicloud deployments, and increased operational overhead make troubleshooting more difficult. Without observability, fragmented visibility and inefficiencies further slow resolution, leaving teams struggling to diagnose issues in dynamic environments.
- **The constant change in Kubernetes environments** (with frequent deployments, autoscaling, and shifting infrastructure) generates an overwhelming volume of telemetry data. Identifying key signals and tracking performance over time is increasingly complex, making it harder to detect patterns and diagnose recurring issues.

From blind spots to clear roads: Using observability to troubleshoot Kubernetes environments

Troubleshooting in Kubernetes environments without observability is like driving through an unfamiliar city with no navigation system — you might eventually get where you need to go, but it'll be slow, frustrating, and full of wrong turns. Like a torn map, fragmented logs and manual debugging leave engineers piecing and taping together clues with little to no guidance.

When implemented properly, observability acts as your GPS, helping you quickly pinpoint root causes and navigate the complexities of modern Kubernetes environments by consolidating the entire system view and providing critical insights into its dynamic operations.

Effective troubleshooting in Kubernetes environments requires more than raw data. Engineers need visibility, correlation, and context to make sense of what's happening. An observability solution should provide:

- **Comprehensive visibility across Kubernetes** environments integrates logs, metrics, and traces within a unified observability platform, ensuring engineers can track performance, detect failures, and correlate events across nodes, pods, and services, helping them quickly diagnose and resolve issues.
- **Proactive anomaly detection** identifies irregular workload behavior, unexpected resource contention, and scaling inefficiencies before they escalate into service disruptions.
- **Contextual, real-time insights** provide correlation across Kubernetes events, workload performance, and cluster metrics, accelerating root cause analysis (RCA) and reducing Mean Time to Resolution (MTTR).
- **Forensic troubleshooting and historical insights** enable teams to investigate past incidents, identify root causes, and analyze resource usage patterns to optimize allocations and prevent recurring issues.

Observability provides the insight needed to stay ahead of issues, prevent disruptions, and accelerate troubleshooting during unplanned downtime, keeping systems running at optimal performance. But even with the best observability in place, failures can still occur.

Rapid diagnosis and resolution are just as critical as proactive monitoring to ensure reliable operations at scale. Much like a modern car's dashboard alerts you to low fuel, tire pressure, or engine trouble before they cause problems, observability equips engineers with the insights they need to drive Kubernetes environments smoothly and efficiently.

The Kubernetes visibility crisis: Why legacy tools are falling behind

Don’t operate a high-performance machine with a broken dashboard

Driving an older car with a flickering warning light — or worse, no warning lights at all — leaves you guessing at what’s failing beneath the hood and when catastrophe might hit.

Similarly, many teams operate Kubernetes clusters with incomplete visibility, relying on fragmented tools that leave them blind to critical performance issues. Teams that lack full observability are forced to troubleshoot through trial and error, slowing down resolution. Engineers waste time piecing together logs and traces, only to chase false leads and misdiagnosed root causes.

The problem

Kubernetes environments generate vast amounts of telemetry data — logs, metrics, and traces — but legacy monitoring tools weren't built for the dynamic nature of containerized workloads. Teams have historically relied on separate tools for each data type, leading to non-contextual, inconsistent, and incomplete insights.

- **Logs, metrics, and traces are spread across multiple tools**, making correlation difficult while increasing spend due to overlapping functionality.
- **Teams lack visibility across multicloud deployments**, leaving critical blind spots, increasing operational costs and impacting SLAs.
- **Teams monitor Kubernetes clusters in isolation**, without visibility into the broader application stack. This leads to blind spots in diagnosing issues that originate outside Kubernetes, such as databases, external APIs, or front-end services.
- **Vendor lock-in limits flexibility and increases toil** when adapting telemetry pipelines for compliance, cost efficiency, or evolving best practices — highlighting the need for open standards like OpenTelemetry.
- **Manual dependency mapping can't keep pace with Kubernetes environments**, where dynamic service relationships shift constantly, and static diagrams quickly become outdated.

Inconsistent troubleshooting methods slow down issue resolution, increasing MTTR and putting service reliability at risk.

Why it's challenging

- **Monitoring only individual clusters** apart from the larger IT ecosystem leads to missed dependencies and incomplete root cause analysis.
- **Fragmented tools force engineers into "swivel chair" troubleshooting**, manually stitching together logs, metrics, and traces — slowing decision-making, increasing MTTR, and creating excessive alert noise.
- **Lack of intelligent or adaptive thresholding** (like anomaly detection) leaves teams reacting to failures rather than preventing them.
- **The rapid pace of change** in Kubernetes environments makes static dashboards and manual instrumentation outdated almost immediately.
- **Legacy monitoring tools lack full coverage across metrics, events, logs, and traces (MELT)**, making troubleshooting issues in dynamic, multicloud Kubernetes environments difficult.

Observability insights and best practices

Modern Kubernetes environments require more than just traditional monitoring — they demand a comprehensive approach to observability. When teams lack a unified strategy, they waste time treating symptoms rather than solving underlying problems. For example, a failing pod in one cluster might not trigger alerts in another, but it could still affect service performance across clusters.

- **Unify observability across metrics, events, logs, and traces** to eliminate data silos, improve real-time correlation, and provide visibility across infrastructure, applications, and multicloud environments.
- **Monitor dependencies across clusters and applications** to identify bottlenecks in distributed services, ensuring troubleshooting includes Application Performance Monitoring (APM) as well as tight integrations with Digital Experience Monitoring (DEM) tools, including Real User Monitoring (RUM) and synthetic testing solutions.
- **Use AI-driven anomaly detection and intelligent thresholding** based on historical baselines to proactively surface issues, reduce alert noise, and improve troubleshooting accuracy.
- **Utilize distributed tracing** to pinpoint the exact source of latency issues and accelerate root cause analysis.
- **Leverage OpenTelemetry** to avoid vendor lock-in, standardize telemetry collection, and enhance cross-platform troubleshooting.
- **Collect and analyze RED metrics (Rate, Errors, Duration) to Kubernetes** workloads to systematically measure request throughput, failure rates, and response latency — ensuring effective performance monitoring and troubleshooting.
- **Embrace AI assistants** that access your system data to accelerate analysis, identify patterns and anomalies, and guide troubleshooting — enabling users of all skill levels to operate with expert insight.

Troubleshooting Kubernetes without full visibility is like driving at night with your headlights off — limited sight means slower reactions and higher risk. Observability cuts through the darkness, giving engineers a clear path to diagnose issues, optimize performance, and ensure workloads run efficiently at scale.

ON-RAMP TO LEARN MORE:

Kubernetes Monitoring: The Ultimate Guide | Splunk

Key kubernetes metrics

Category	Metric	Description and Details
Cluster Health	Cluster API Server Availability, Latency, and Errors	Monitors the Kubernetes API server's health, ensuring it's available and responsive. High latency or frequent errors can indicate control plane issues affecting cluster management.
	Node Status	Tracks the health and status of cluster nodes. Detecting unavailable or unhealthy nodes early helps prevent workload disruptions.
Pods and Container Health	Status	Indicates the current state of running pods, showing if they are ready, pending, or in an error state.
	Desired vs. Actual Counts	Compares the expected number of running pods to the actual count. Discrepancies can signal scheduling failures or resource constraints.
	Restarts	Tracks pod restarts due to crashes or failures. Frequent restarts may indicate misconfigurations, insufficient resources, or application errors.
Network and Service Performance	Cross-cluster Network Latency and Failures	Measures latency and failures in cross-cluster communication. High latency can lead to degraded service performance and timeout issues.
	DNS Resolution Time and Failures	Tracks DNS resolution times and failures within the cluster. Slow resolution or failures can cause service disruptions and connectivity issues.
	Service Discovery Errors	Identifies service discovery errors that can impact inter-service communication. Monitoring helps detect misconfigurations and broken dependencies.
	Request Rates, Latency, Throughput	Observes request rates, latency, and throughput for services. High latency or degraded throughput can indicate bottlenecks or overloaded services.
Resource Utilization and Capacity	CPU/Memory/Disk Utilization and Capacity	Measures resource usage, consumption, and available capacity across nodes, pods, and containers. Monitoring trends helps prevent resource exhaustion and improve workload scheduling.

CHALLENGE 2

Diagnosing scaling and performance bottlenecks

Pressing the gas but not speeding up

Have you ever had a car where you step on the gas and there's a hesitation before it accelerates? That's like Kubernetes autoscaling delays — by the time it reacts, demand has already shifted.

Not anticipating demand and load as well as misconfigured scaling can leave workloads starving for resources exactly when you need them the most (like during a traffic spike) or wasting capacity — just like a turbocharged engine might experience turbo lag before delivering full power. Engineers need real-time insights into scaling behavior to ensure their Kubernetes workloads respond efficiently, keeping applications running smoothly under fluctuating demand.

The problem

Kubernetes pod autoscaling mechanisms, including some popular approaches such as Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA), are designed to adjust resources dynamically, but improper configurations or failing to use these autoscaling approaches altogether can lead to severe performance bottlenecks:

- **Kubernetes scaling inefficiencies have up- and down-stream impacts** — poor scaling decisions can lead to increased latency, service slowdowns, and cascading failures across microservices, databases, and APIs.
- **Setting CPU budgets too low** (sometimes called under-requesting) can lead to throttling, while **misconfigured memory budgets** can cause Out of Memory (OOM) kills, leading to pod restarts and degraded performance.
- **Node resource saturation** causes pod evictions, resource contention, and even cluster instability.
- **Cold start delays** prevent new pods from responding to increased demand quickly, especially in autoscaled environments where workloads with large images, JVM-based applications, or database dependencies require extended initialization times.
- **Over- or under-provisioning of resources** not only inflates cloud costs but also leads to throttling, slow response times, or failures under load, especially in environments without visibility into real usage patterns.
- **Complexity in mixed workloads** (e.g., hybrid or multicloud environments) makes optimizing scaling strategies even more difficult, especially when workloads must communicate across clusters or clouds with different autoscaling policies.

Cloud-native autoscaling often relies on cloud provider-specific mechanisms, such as AWS Auto Scaling, Google Cloud's Cluster Autoscaler, or Azure Kubernetes Service scaling. While these tools can help with dynamic workload adjustments, they are often disjointed, leading to inconsistencies across multicloud or hybrid environments. Teams that lack an observability strategy face challenges diagnosing scaling issues in real-time, leading to higher MTTR and lower application reliability.

Why it's challenging

- **Fragmented observability makes it hard to connect scaling events to business impact**, leaving teams blind to user-facing consequences.
- **Without real-time visibility into scaling behavior**, teams are prone to misconfiguring resource requests — leading to wasted capacity or performance bottlenecks.
- **Ignoring pod evictions and resource contention**, resulting in unexpected application failures.
- **Autoscaling tuning is an art form.** HPA utilizes multiple occurrences of an observed metric to balance temporary deviations and avoid "flapping" (frequent scaling in/out).
- **HPA and VPA can conflict in dynamic environments.** HPA scales pod count based on resource utilization, while VPA adjusts pod resource requests, potentially causing restarts. Coordination is needed to prevent conflicts.
- **JVM-based workloads add tuning complexity** (e.g., cold starts, garbage collection, memory management), often interfering with autoscaling and potentially contributing to performance bottlenecks.
- **Kubernetes issues can cause widespread disruption**, as the platform now powers many of the most critical business services.

Observability insights and best practices

Not thinking about scaling in advance leads teams to reactively adjust limits, modify thresholds, and deploy resources without fully grasping the consequences. These guesswork-driven changes can either lead to unnecessary costs, ongoing performance bottlenecks, or even worse, eroded brand reputation.

To maintain optimal Kubernetes performance, teams need observability-driven insights that provide clarity into autoscaling behavior and resource utilization:

- **Unify your observability data** to capture the full picture of Kubernetes performance, ensuring that scaling and tuning decisions are informed by end-to-end insights — from infrastructure to customer impact.
- **Correlate autoscaling events with external service dependencies** to identify latency bottlenecks caused by uneven scaling across microservices or databases.
- **Monitor real-time scaling behavior** to detect and resolve delays before they impact users.
- **Track key resource and performance metrics** (like CPU, memory, and RED signals) to spot bottlenecks early and prevent degraded user experiences.
- **Leverage baselines, forecasts, and business demand** to fine tune autoscaling configurations, optimizing efficiency over time.
- **Track node-level resource saturation** to detect pod evictions early and ensure workloads are scheduled efficiently across the cluster.
- **Analyze pod startup times** and correlate with autoscaling events to fine-tune scaling triggers, especially for latency-sensitive applications.
- **Ensure balanced resource allocation** by leveraging observability to avoid over-provisioning (wasting compute) or under-provisioning (causing service degradation) capacity.
- **Stay current on evolving autoscaling strategies beyond pods**, and use observability to track how node, cluster, and workload scaling impacts overall performance.

Observability acts as a performance tuning tool, much like a vehicle's onboard diagnostics system that continuously tracks performance over time. Just as modern cars use telemetry data to optimize engine efficiency and anticipate issues, Kubernetes teams can analyze historical metrics to fine-tune autoscaling strategies. This approach ensures Kubernetes scaling decisions are as precise as a finely tuned engine — delivering the right resources at the right time with no unnecessary delays.

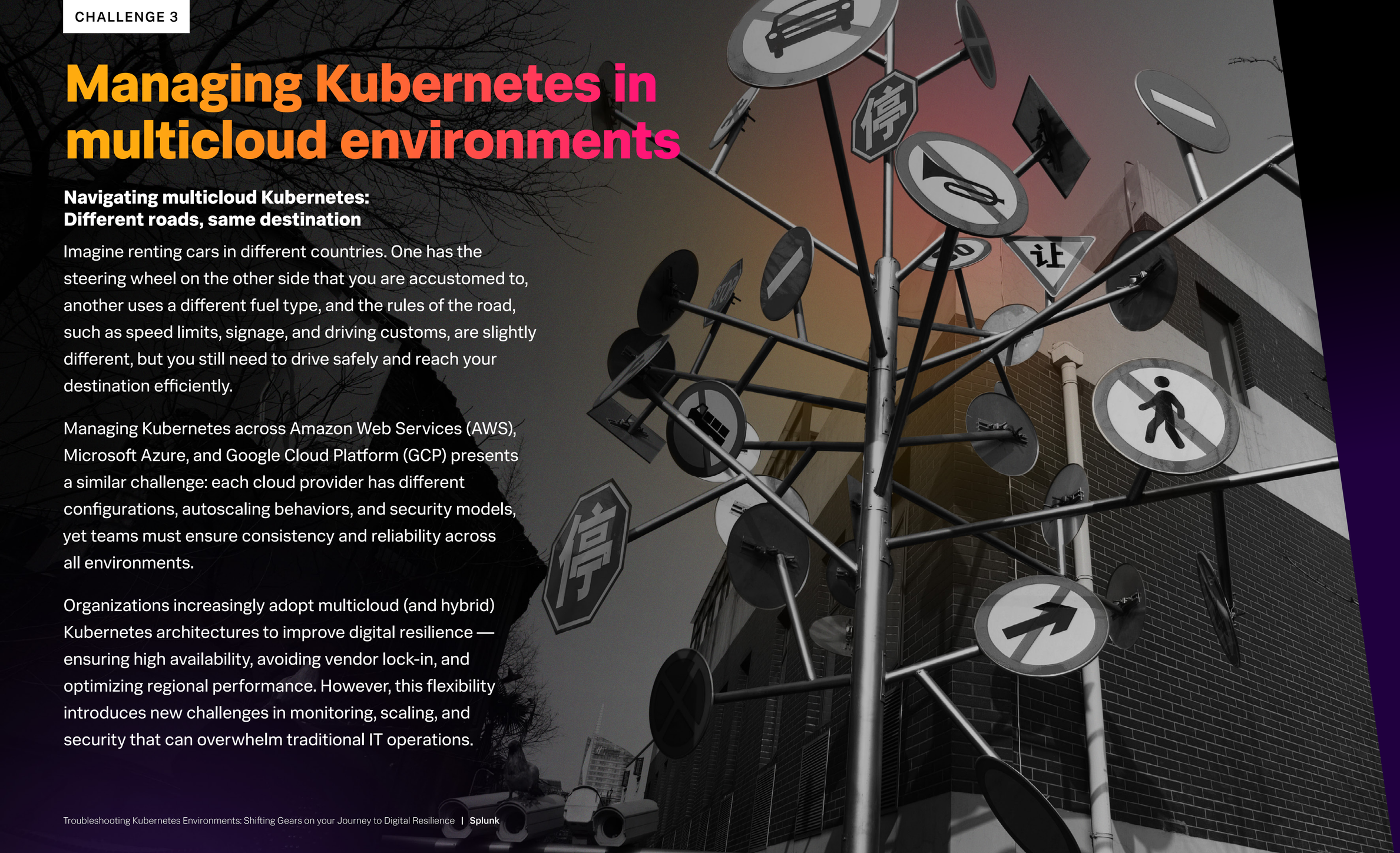
ON-RAMP TO LEARN MORE:

Kubernetes Horizontal Pod Autoscaling

Key autoscaling metrics

Category	Metric	Description and Details
Horizontal and Vertical Scaling	Desired Replicas vs. Actual Replicas (HPA)	Compares the number of desired pod replicas to the actual running pods. Large discrepancies may indicate HPA misconfigurations, a lack of resources in the cluster, or delays starting applications.
	Cluster Autoscaler Activity and Node Count	Monitors autoscaler decisions and node count changes. Helps detect delayed scaling events, resource saturation, and cluster imbalance.
	Cluster Autoscaler Event Logs	Logs scaling decisions made by the Cluster Autoscaler. Analyzing events helps identify scaling inefficiencies and bottlenecks in resource provisioning.
Pod and Node Health	CPU/Memory Requests vs. Limits	Evaluates requested vs. allocated CPU and memory resources. Disparities or throttling can lead to resource waste (over-provisioning) or performance bottlenecks (under-provisioning).
	Pod Evictions and Restart Counts OOM (Out of Memory)	Counts pod evictions and restarts caused by out-of-memory (OOM) conditions, resource constraints, or policy violations. Frequent occurrences indicate excessive resource contention or poor memory allocation.
Traffic and load balancing	Request Rate per Pod	Measures the number of requests per pod to evaluate traffic distribution. Uneven request rates may signal load-balancing inefficiencies.
	Load Balancer Request Distribution	Analyzes how requests are distributed across load balancers. Skewed distribution can indicate misconfigurations, affecting service responsiveness.
	Network I/O	Monitors network throughput and data transfer rates across nodes and services. Helps identify potential network bottlenecks impacting scalability and performance.

Managing Kubernetes in multicloud environments



Navigating multicloud Kubernetes: Different roads, same destination

Imagine renting cars in different countries. One has the steering wheel on the other side that you are accustomed to, another uses a different fuel type, and the rules of the road, such as speed limits, signage, and driving customs, are slightly different, but you still need to drive safely and reach your destination efficiently.

Managing Kubernetes across Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) presents a similar challenge: each cloud provider has different configurations, autoscaling behaviors, and security models, yet teams must ensure consistency and reliability across all environments.

Organizations increasingly adopt multicloud (and hybrid) Kubernetes architectures to improve digital resilience — ensuring high availability, avoiding vendor lock-in, and optimizing regional performance. However, this flexibility introduces new challenges in monitoring, scaling, and security that can overwhelm traditional IT operations.

The problem

Running Kubernetes in a multicloud environment introduces multiple layers of complexity that single-cloud and/or non-orchestrated clustered deployments don't face. Teams must navigate differences in networking, security models, and autoscaling logic while ensuring workloads remain portable and resilient. Without a unified observability strategy, maintaining visibility and operational consistency across disparate cloud platforms becomes an ongoing challenge.

The key is balancing consistency with cloud-specific insight. Teams need a standardized way to monitor Kubernetes — while still exposing provider-level differences that influence workload behavior, performance, and reliability.

- **Unified visibility must extend beyond cloud providers** to include distributed services within a single cloud provider, such as cross-account and cross-tenant observability.
- **Cloud-specific configurations** vary across providers, including security enforcement, API behaviors, and autoscaling policies, requiring teams to adapt workload tuning and governance strategies accordingly.
- **Cloud provider monitoring tools** (AWS CloudWatch, Azure Monitor, GCP Operations Suite) **generate observability telemetry but remain siloed**, making it difficult to correlate deployment interdependencies across clouds.
- **Legacy vendor-centric agents and solutions** require engineering teams to learn multiple tools and UIs, increasing toil and technical debt and reducing operational efficiency.
- **Inconsistent telemetry formats and naming conventions** across cloud providers complicate log parsing, metric correlation, and root cause analysis — especially when relying on open-source tools or homegrown solutions.

Teams running separate Kubernetes clusters in multiple clouds struggle with governance, security, and compliance inconsistencies, requiring additional overhead to maintain operational standards. Inconsistent monitoring and observability practices create troubleshooting delays, unreliable performance metrics, and governance challenges.

Why it's challenging

- **Kubernetes observability must be contextually integrated** with the broader business application ecosystem to ensure infrastructure health is correlated with application performance and user experience. Multicloud deployments introduce additional context complexity.
- **Cloud-native monitoring tools are often vendor-specific and siloed**, making it difficult to maintain a unified view — whether for a single application spanning clouds or separate clusters operating in different environments.
- **Disconnected monitoring solutions slow down troubleshooting**, increasing Mean Time to Resolution (MTTR) and operational inefficiencies.
- **Security and compliance enforcement varies across cloud providers**, making consistent governance and regulatory adherence difficult.
- **Different configurations and security models** across AWS Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE), and Azure Kubernetes Service (AKS) create challenges for teams managing either a single distributed application or independent clusters in each cloud.
- **Teams running multicloud Kubernetes must manage different autoscaling behaviors**, whether balancing workloads across clouds or optimizing independent clusters to meet performance demands.
- **Different cloud providers have varying machine types and pricing**, affecting instance selection, resource allocation, and performance limits.

Observability insights and best practices

To ensure reliability across multicloud Kubernetes environments, teams need a unified observability approach that standardizes monitoring and troubleshooting while enabling contextual correlation of all components within a business application ecosystem. Observability must abstract the complexity of multicloud and multi-tenant Kubernetes deployments — providing unified visibility across environments — while enabling seamless correlation of logs, metrics, and traces independent of the cloud provider.

- **Unify observability across cloud providers to correlate telemetry** and quickly pinpoint cross-cloud issues like application bugs, security-related problems, or networking failures.
- **Embrace and leverage OpenTelemetry** to collect, process, and standardize metrics, logs, and traces across multicloud environments, enabling seamless data export and vendor-agnostic analysis.
- **Use service dependency mapping and distributed tracing** to track latency, detect cross-cloud bottlenecks, and maintain consistent application performance across providers.
- **Integrate cloud provider telemetry data** (e.g., AWS CloudWatch, Azure Monitor, GPC Operations Suite) into your observability strategy to ensure complete visibility when leveraging these Kubernetes services.
- **Standardize metric names and adopt tagging and annotations across cloud providers** to enrich observability telemetry with business context.
- **Continuously monitor workload performance variations** between cloud providers to optimize resource efficiency, balancing costs, latency, and compute power across environments.
- **Analyze deployment failures and rollback trends** to detect API mismatches, configuration drift, and cloud-specific performance regressions — leveraging CI/CD tools for automated resolution.
- **Monitor autoscaling efficiency across cloud providers** to detect under- or over-provisioning, ensuring optimal resource usage and cost efficiency while aligning with Kubernetes HPA/VPA and cloud-native scaling policies.

Unified observability equips teams with the insights to operate Kubernetes smoothly across any cloud environment — just like an experienced driver can confidently navigate any car in any country, seamlessly convert MPH to KPH, and understand the essential controls and regulations.

Whether managing workloads across multiple providers or optimizing independent clusters, a well-structured unified observability solution ensures teams maintain visibility, efficiency, and performance independent of the underlying provider.

ON-RAMP TO LEARN MORE:

Monitoring Amazon Elastic Kubernetes Service (EKS)

Key multicloud monitoring metrics

Category	Metric	Description and Details
Cross-cloud Performance	Service Mesh Latency	Tracks the latency introduced by service mesh implementations across clouds. Increased latency can signal inefficient routing, misconfigurations, or network congestion.
	API Server Request Latency across Clusters	Monitors the response time of API server requests across different clusters. High latency can indicate communication inefficiencies, version mismatches, network bottlenecks, or authentication issues.
Resource and Configuration Monitoring	Node Availability and Health per Cloud Provider	Evaluates node availability and health within each cloud provider's environment. Helps ensure clusters are running optimally and detect cloud-specific outages or capacity constraints.
	Storage, IOPS and Throughput	Measures storage, input/output operations, and data transfer rates across cloud platforms. Ensures storage throughput, IOPS, and data transfer rates are performing adequately.
Security and Audit Monitoring	Audit logs and security events	Captures security-related events and audit logs across cloud environments. Helps identify suspicious activity, compliance violations, or configuration drift.
	API request failures and unauthorized access attempts	Tracks API request failures and unauthorized access attempts across clusters. Helps detect potential security breaches, misconfigurations, or failed authentication policies.

Detecting and resolving resource pressure in Kubernetes clusters

The overheating engine problem

Your car's cooling system keeps engine temperatures stable, preventing breakdowns. But what happens if it fails? An overheated engine can cause cascading failures — damaging the radiator, warping the cylinder heads, and even leading to complete engine failure. Similarly, unchecked CPU and memory pressure in Kubernetes can trigger pod evictions, resource starvation, and severe performance degradation, which can disrupt workloads and impact system stability.

Proactive monitoring allows engineers to detect early warning signs, preventing degraded performance and instability. In Kubernetes environments, where workloads shift constantly, even small inefficiencies in resource requests and limits can create ripple effects. Without timely insight into how CPU and memory are used, teams often struggle to maintain balance — some applications may be over-provisioned, while others are left starved for resources. These mismatches can lead to unnecessary scaling, increased costs, and performance issues that are difficult to trace until it's too late.

The problem

Teams build Kubernetes clusters to manage workloads dynamically, but resource pressure remains a persistent challenge. Inefficient resource allocation, often caused by misconfigured requests and limits or unbalanced scaling policies, can lead to escalating contention, affecting multiple applications running in the same cluster. As CPU and memory thresholds are exceeded, workloads may experience throttling, evictions, or degraded performance, especially under peak demand.

- **Nodes under resource pressure** experience CPU throttling and pod evictions. At the same time, pods with misconfigured resource requests or limits can suffer from resource starvation or trigger inefficient autoscaling behavior.
- **Dynamic workload shifts**, such as scaling events or node rescheduling, can lead to imbalanced resource consumption. This not only degrades application performance but also increases unnecessary cloud costs.
- **Complex Kubernetes configurations** may result in excessive resource use and misallocations.
- **Lack of real-time visibility** into usage and capacity thresholds forces teams to react to failures instead of preventing them.
- **Identifying bottlenecks in complex environments** is difficult, especially when multiple workloads compete for shared resources like CPU, memory, and network bandwidth.

Without clear visibility into what's driving resource pressure — whether it's misconfigured requests, workload contention, or node-level constraints — teams are often forced to make blind adjustments to resource limits or autoscaling configurations. These reactive changes may temporarily ease symptoms but rarely address the root cause, leading to recurring performance bottlenecks that could have been prevented through better data and signal-driven insights.

Why it's challenging

- **Failure to proactively monitor cluster health and capacity limits** leads to sudden failures.
- **Ignoring early signs of resource exhaustion** causes reactive scaling instead of strategic allocation.
- **Misconfigured requests and limits** that result in inefficient resource distribution.
- **The "Noisy Neighbor" effect** comes into play where some workloads consume excessive resources, starving others.
- **Failing external services** can trigger excessive retries or stalled processes within Kubernetes workloads, increasing CPU and memory usage and contributing to overall cluster resource strain.

Observability insights and best practices

Observability gives engineers real-time insights into resource consumption, workload contention, and early indicators of performance issues before they escalate into failures. Without this visibility, teams are left responding to symptoms instead of resolving root causes, often resulting in recurring performance or scaling inefficiencies that drain time, budget, and team focus.

By leveraging observability, engineers can fine-tune resource allocations, optimize autoscaling strategies, and prevent the Kubernetes equivalent of an overheating radiator where unchecked pressure builds until it causes cascading failures across workloads and services.

- **Leverage real-time cluster observability** to detect and resolve resource bottlenecks before workloads fail.
- **Monitor eviction metrics and correlate them with CPU/memory saturation** to identify imbalanced resource allocation.
- **Track the impact of failing external dependencies** by monitoring increases in CPU, memory, or wait times within affected workloads, helping identify when resource pressure stems from outside the cluster.
- **Understand the correlation of resource utilization to end-user impact** to avoid overreacting to high usage and instead focus on the moments that affect customer experience.
- **Analyze failed autoscaling events** to optimize Horizontal Pod Autoscaler (HPA) thresholds and prevent resource gaps.
- **Implement dynamic resource allocation strategies**, adjusting pod requests based on historical usage trends.
- **Correlate pod restarts, CPU throttling, and eviction patterns** to diagnose whether resource pressure is caused by workload contention, misconfigured requests, or underlying node constraints.

Just as a car's cooling system prevents overheating by regulating engine temperature, Kubernetes observability provides engineers with tools to detect and address resource pressure before workloads overheat. By continuously monitoring CPU and memory allocation, engineers can fine-tune resource distribution, optimize autoscaling strategies, and keep clusters running smoothly under fluctuating demand — preventing failures and their associated brand and financial costs before they cause major disruptions.

ON-RAMP TO LEARN MORE:

Detect and Resolve Issues in a Kubernetes Environment

Key resource pressure metrics

Category	Metric	Description and Details
Pod and Container Level	Pod Evictions and Pending Pod Counts	Tracks the number of pods evicted due to resource pressure and those still pending scheduling. High counts indicate insufficient capacity or workload contention.
	Restart Counts	Measures how frequently pods restart due to crashes or resource exhaustion. Persistent restarts signal instability or misconfigured resource limits.
	Containers withoutResource Limits	Identifies containers running without defined CPU/memory limits. Lack of constraints can lead to excessive resource consumption and 'noisy neighbor' issues.
Node Level	Node CPU/Memory/Disk Pressure	Measures pressure on CPU, memory, and disk at the node level. Persistent high usage may lead to degraded performance and potential node failures.
Cluster Level	Resource Quotas	Defines the maximum allowable resource allocation for namespaces or workloads. Breaching quotas may cause pod failures and service disruptions.
	Cluster Autoscaler Event Logs	Logs scaling decisions made by the Cluster Autoscaler. Analyzing events helps identify scaling inefficiencies and bottlenecks in resource provisioning.
	Daemonsets at Deficit/ Restart Counts	Tracks whether daemonsets (essential system-level services) have enough available nodes to run properly and how frequently they restart. A deficit or frequent restarts indicate constraints, node instability, misconfigurations, or resource exhaustion.
Network	Latency	Monitors delays in processing requests due to resource bottlenecks. Increased latency often signals overloaded nodes or insufficient scaling.
	Throughput	Measures the rate of successful requests per second. Low throughput can indicate resource constraints affecting service responsiveness.
	Errors	Tracks failed network requests, connection timeouts, and dropped packets. Persistent errors may indicate network congestion, misconfigured routing, or service communication failures.
	Network utilization	Tracks overall network traffic and bandwidth usage. High network utilization can impact communication between services and degrade application responsiveness.

From dashboard warnings to true diagnostics

Driving a car is, honestly, quite a challenge. You must pay attention to literally everything. From your dashboard displaying urgent lights or no warning lights at all to an overheating engine to learning the laws of the road, you definitely have a lot to think about. It all boils down to getting as much intelligent, contextual, and accurate information as possible when it truly matters.

Apart from relying on experience, you might not know exactly **why** the engine is failing, just that something's wrong. If you want real answers, you need **a diagnostic tool that goes beyond "it's broken"** — pinpointing exactly which part or system is failing and why.

Kubernetes environments are the same way. Deployments emit **vast amounts of telemetry data** and you definitely do not want to be repeatedly looking for the same needle in the needle stack, over and over. You need effective **observability** to trace issues across clusters, correlate performance signals, and provide actionable insights — ideally positioning your team to fix problems before they impact users.

Throughout this e-book, we've explored the complexities of troubleshooting Kubernetes environments, from diagnosing scaling issues and resource constraints to managing multicloud deployments. Each of these challenges reinforces a central theme: **a strong observability strategy is critical to troubleshooting Kubernetes environments successfully**.

To provide effective Kubernetes observability, you'll need to address several key priorities: multicloud, correlation of telemetry to business health, AI insights, and more.

Observability must **support multicloud Kubernetes deployments and distributions** while maintaining vendor-agnostic flexibility. Solutions like **OpenTelemetry** help eliminate instrumentation toil by consolidating metrics, logs, and traces into a single framework — simplifying telemetry collection and enabling consistent observability across Kubernetes environments. By **correlating real-time telemetry with business application health**, teams can perform end-to-end troubleshooting, optimizing both infrastructure performance and user experience.

AI-guided insights surface anomalies, identifying patterns, and accelerating root cause analysis across your specific Kubernetes environments. **Context-rich logging and telemetry correlation** help engineers quickly pinpoint root causes — maintaining critical links across application traces (the path a request takes through service), Kubernetes components, and infrastructure layers to enable true end-to-end troubleshooting. Additionally, **observability must scale with your environment**, offering visibility across clusters, cloud providers, and hybrid deployments.

High-level troubleshooting guidance

To efficiently troubleshoot Kubernetes environments, teams need a structured approach that addresses the entire technology stack, from nodes to applications:

- **Deploy comprehensive observability solutions** that integrate infrastructure monitoring, APM, and logging capabilities.
- **Identify the problem scope** (node, pod, service layer, network) to narrow down root causes quickly.
- **Train teams on observability tools** to improve troubleshooting speed and accuracy.
- **Detect failure trends before they impact production** by continuously analyzing key performance indicators.
- **Conduct proactive resource allocation** using historical resource utilization trends to prevent resource exhaustion, optimize scaling strategies, and mitigate performance bottlenecks.
- **Surface and analyze anomalies in key metrics** to detect early warning signs of instability, performance degradation, or resource contention before they impact service reliability.
- **Check for misconfigurations and dependencies** that may be contributing to performance degradation.
- **Utilize cloud-native observability tools** designed for microservices environments to correlate logs, metrics, and traces, enabling effective root cause analysis across distributed Kubernetes workloads.
- **Use AI-driven insights and machine learning-powered anomaly detection** to minimize reliance on tribal knowledge, surface hidden patterns, and accelerate root cause analysis.
- **Implement vendor-neutral instrumentation with OpenTelemetry** to unify metrics, logs, and traces across clusters and clouds, ensuring consistent visibility and preventing vendor lock-in.
- **Ensure end-to-end observability in multicloud Kubernetes** (independent of where you deploy your Kubernetes workloads) by correlating performance data, detecting cross-cloud issues, and streamlining troubleshooting.

Observability transforms troubleshooting in Kubernetes environments from a reactive, time-consuming process to a strategy that ensures performance, efficiency, and ultimately, digital resiliency.

Just as modern car diagnostics go beyond simple dashboard warnings, observability equips teams with real-time, actionable insights to understand what's happening under the hood and address issues with precision. **Observability replaces guesswork with clarity, enabling engineers to troubleshoot faster and keep Kubernetes environments running at peak performance**.

Learn More About Kubernetes Observability

Embark on your Kubernetes Road Trip

- Explore Splunk Kubernetes Monitoring
- Download the Splunk Kubernetes Monitoring Data Sheet
- Join the Splunk Community for Kubernetes
- Accelerate data onboarding in Kubernetes

splunk> a CISCO company

25_CMP_ebook_troubleshooting-kubernetes-environments_v9