



# The Complete Container Observability Checklist: What IT Leaders Need to Look For

## What IT Leaders Need to Look For

Mastering Kubernetes observability requires understanding a sophisticated system where traditional monitoring approaches fall short. Whether you just started learning about containers or are a hardened kubectl expert, it's clear that there are a lot of moving pieces, layers, dependencies, and abstractions in the land of Kubernetes. Coupling this with the ephemerality of containers and the complexity of microservices gives you environments where traditional monitoring approaches are ineffective, at best.

Migrating from traditional infrastructure to containers? Expanding your container footprint? Enhancing your container observability strategy?

This checklist serves as a guide for IT leaders and practitioners evaluating Kubernetes observability solutions, helping to understand key requirements for a platform that will ensure you have reliable, performant, optimized, and cost-efficient container environments.

### Core Observability Capabilities

#### Complete Topology Visibility

**What to look for:** A solution that provides both technology stack dependencies (vertical topology) and service interactions (horizontal topology) in a unified view.

**Why it's important:** In container environments, understanding both the infrastructure stack and the service-to-service communications is critical. Containers are ephemeral by nature, making it difficult to track dependencies without proper visualization tools. Complete topology visibility helps teams quickly understand complex relationships, significantly reducing troubleshooting time when issues occur.

#### Topology Generation Without Requiring Code Instrumentation

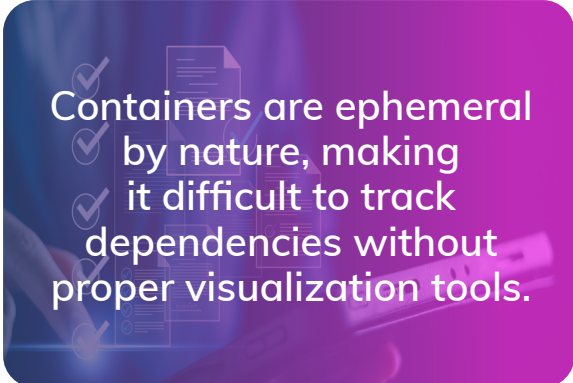
**What to look for:** The ability to generate accurate service maps without requiring deep code-level instrumentation.

**Why it's important:** Traditional topology from APM tools typically requires extensive code instrumentation, which creates maintenance overhead and may not be possible for all applications. In diverse container environments, the ability to automatically generate topology maps without modifying application code ensures comprehensive visibility across your entire estate, including third-party and legacy applications.

#### Code Level Visibility

**What to look for:** A platform that provides auto-discovered transaction visibility along with each transaction's logging context, and the ability to drill down into individual methods and functions to understand latency, error rates, and health of your business functions.

**Why it's important:** While the previous section talks about topology without code-level instrumentation, the ability to analyze code-level data is still a key requirement of any complete observability solution. Deep transaction analysis at the code level allows engineers and developers to quickly find offending lines of code in misbehaving or unhealthy requests, and the accompanying log data allows for a richer environmental context when diagnosing.



Containers are ephemeral by nature, making it difficult to track dependencies without proper visualization tools.

**Comprehensive Telemetry Integration**

**What to look for:** Unified collection and deep, relationship-and-dependency-based correlation of metrics, logs, configurations and traces from all container ecosystem and supporting infrastructure components.

**Why it's important:** Containers and their infrastructure generate massive amounts of telemetry data across different formats. Without a platform that can collect and correlate metrics, logs, configuration, and distributed traces in one place, teams waste valuable time switching between tools and manually connecting data points. Comprehensive telemetry integration provides the context needed to quickly understand and resolve issues.



**Kubernetes-Specific Insights**

**What to look for:** Purpose-built monitoring for Kubernetes that natively understands different workload types (DaemonSets, Jobs, Deployments) at the cluster, node, and workload levels, with the ability to visualize these relationships in systematic graphs.

**Why it's important:** Kubernetes introduces its own complex abstractions and failure modes that general-purpose monitoring tools don't adequately address. Each workload type (DaemonSets, Jobs, Deployments) has unique characteristics and potential failure modes. A solution that natively understands these differences can provide context-aware observability data and visualization, helping teams quickly understand how orchestration decisions impact application performance and provide the insights needed to optimize both the application and the platform.

**Problem Detection and Resolution Capabilities**

**Automated Anomaly Detection**

**What to look for:** Multiple detection approaches, including AI, machine learning, and knowledge-based patterns to identify issues before they impact users.

**Why it's important:** Container environments generate too many metrics and events for traditional monitoring. Automated anomaly detection can identify unusual patterns across thousands of containers, their workloads, and supporting infrastructure services, alerting teams to potential issues before they cause outages. This proactive approach is essential when managing large-scale containerized applications.

**Advanced Noise Reduction**

**What to look for:** Intelligent alert correlation and filtering that reduces alert fatigue by grouping related issues, suppressing redundant notifications, and prioritizing alerts based on business impact.

**Why it's important:** Container environments can generate thousands of alerts daily, overwhelming teams with notification noise. Advanced noise reduction capabilities ensure that teams focus only on actionable information rather than drowning in redundant alerts. This targeted approach dramatically improves operational efficiency, allowing engineers to focus on true issues rather than chasing false positives or duplicate notifications for the same root cause.

**Advanced Root Cause Analysis**

**What to look for:** Automated investigation capabilities that can determine the source of issues across complex container dependencies.

**Why it's important:** In containerized environments, a single issue can generate alerts across dozens of entities. Without automated root cause analysis, teams waste hours in war rooms trying to correlate symptoms with causes. Advanced RCA dramatically reduces mean time to resolution by pinpointing the exact source of problems, even when the triggering container has already been replaced.



**Historical Analysis Capabilities**

**What to look for:** The ability to view past configurations and states, even for containers and infrastructure resources that no longer exist.

**Why it's important:** By the time an issue is reported in container environments, the problematic container (and even the node!) may have already been terminated and replaced. Time travel capabilities enable teams to investigate what happened in the past, understand the conditions that led to failures, and prevent similar issues in the future—even when the original resources are long gone.





## Deployment and Integration

### Deployment Flexibility

**What to look for:** Support for SaaS, on-premises, and private cloud deployment models.

**Why it's important:** Organizations have different requirements regarding data residency, security, and operational models. Deployment flexibility ensures that your observability solution can adapt to your specific needs, whether you're all-in on the cloud, maintaining on-premises infrastructure, or operating in a hybrid model.



### Open-Source Integration

**What to look for:** First-class support for open-source telemetry collectors and observability standards.

**Why it's important:** Many organizations have already invested in open-source observability tools like Prometheus, Grafana, or OpenTelemetry. A platform that treats these data sources and their collection mechanisms as first-class citizens rather than afterthoughts allows you to leverage existing investments while extending capabilities. This approach prevents vendor lock-in and provides more freedom to evolve your observability strategy over time.

### Rapid Deployment and CI/CD Integration

**What to look for:** The ability to be up and running in a cluster in as little as one minute, with seamless integration into your organization's CI/CD pipeline tools like ArgoCD, Terraform, Jenkins, or GitLab.

**Why it's important:** Traditional monitoring tools often require complex setup processes that can take days or weeks. In fast-moving container environments, this delay is unacceptable. Modern organizations deploy workloads through CI/CD pipelines using tools like ArgoCD, and your observability solution should honor the same practices. This ensures that monitoring is deployed alongside applications automatically, maintaining consistency between environments and eliminating manual configuration steps that can lead to gaps in visibility.



## Advanced Capabilities

### Resource Utilization Optimization

**What to look for:** Identification of under/over-allocated resources and right-sizing recommendations.

**Why it's important:** Container environments can quickly become inefficient without proper oversight. Resources that are over-allocated waste money, while under-allocated resources risk application performance and stability. Optimization tools help teams balance cost and performance by identifying precisely where adjustments are needed across their entire container estate.

### AIOps Capabilities

**What to look for:** The ability to ingest and correlate data from third-party tools without duplicating data storage.

**Why it's important:** Most organizations have existing investments in other monitoring tools. The ability to integrate with these platforms for additional context while avoiding data duplication reduces costs and complexity. This approach provides a unified view of your environment while respecting your existing technology choices.

### Architecture Validation

**What to look for:** Runtime validation that confirms applications are behaving as architected.

**Why it's important:** With container environments constantly and perpetually changing, implementations often drift from the intended architecture. This can lead to security risks, performance issues, and maintenance challenges. Architecture validation ensures that what's running in production matches what was designed, helping teams maintain control over increasingly complex systems.



### Simple, Transparent Pricing

**What to look for:** Pricing based on entity counts rather than data volume.

**Why it's important:** Container environments generate enormous amounts of telemetry data, making volume-based pricing models unpredictable and potentially very expensive. Entity-based pricing provides cost predictability, allowing teams to instrument everything without fear of surprise bills. This approach aligns costs with the value received rather than penalizing you for collecting more data.

## Getting Started

Selecting the right container observability platform is critical for maintaining the reliability, performance, and efficiency of modern containerized environments. By evaluating potential solutions against this focused checklist, you can ensure you're choosing a platform that provides the essential capabilities needed for effective container monitoring and management.

The ideal platform should help you solve immediate issues and provide the insights needed to continuously improve your container infrastructure and applications. Look for solutions that combine deep technical capabilities with ease of use and clear business value.

At Virtana, we've built our container observability platform with these criteria in mind, leveraging the best of open source along with advanced capabilities to deliver a solution that meets the needs of modern containerized environments. We invite you to see how our approach can help your organization achieve better reliability, faster problem resolution, and optimized resource utilization.

