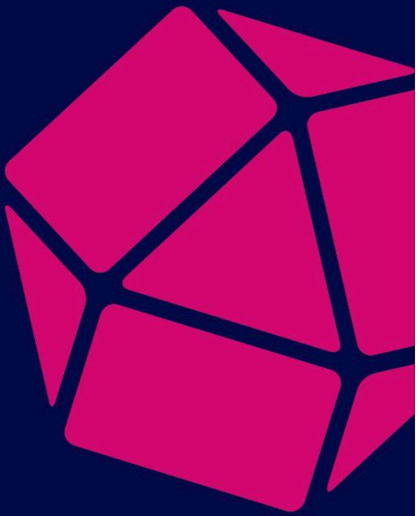
influxdb™

AN INFLUXDATA TECHNICAL PAPER

Kubernetes Monitoring With InfluxDB



APRIL 2024 | REV 1

This tech paper provides an overview of some key things to keep in mind when monitoring Kubernetes and how to use InfluxDB and Telegraf to set up a scalable Kubernetes monitoring solution.

An introduction to Kubernetes monitoring

Monitoring is integral to Kubernetes operations because Kubernetes orchestration acts based on monitored cluster resources. However, from a monitoring perspective, the focus of Kubernetes is relatively narrow. It collects data to make a point decision and moves on to the next sample interval without storing the previous data long-term. Because Kubernetes' main goal is to comply with the declared desired state recorded in etcd, it only retains the most up-to-date value. As a consequence, there is no native monitoring of historical records or behavior trends included in Kubernetes.

The growth of Kubernetes adoption is accompanied by the need for holistic monitoring of containerized application environments—from the observation of resource utilization and workloads running inside containers to the communication networks/service meshes connecting them. All these elements must maintain end-to-end performance and contribute positively to critical business indicators.

Fortunately, the Kubernetes community is ecosystem-friendly and didn't overlook the need for holistic monitoring. They understood that there are aspects of modern application environment monitoring beyond Kubernetes' orchestration scope and that would be better handled by other platforms.

Core metrics pipeline and monitoring pipelines

Kubernetes generates plenty of metrics and logs. Although it stays focused on its core functionalities, consuming only fresh data, it does provide mechanisms to facilitate monitoring via other solutions. For instance, Kubernetes adopted a standard way to expose metrics, the Metrics API, which uses Prometheus format implemented in all its components.

From a metrics pipeline perspective, Kubernetes collects metrics it consumes, monitors (including resource and Kubernetes state metrics), and core metrics. But that is still limited to K8's orchestration scope and should not be confused with a comprehensive monitoring pipeline provided by third-party solutions to monitor the whole Kubernetes application environment. See below how they differentiate from each other:

Core metrics serve metrics that Kubernetes understands and consumes to perform the functions of its internal components. For example, core metrics are used for scheduling which encompasses resource estimation, initial resources/vertical autoscaling, cluster autoscaling, and horizontal pod autoscaling.

Resource and Kube state metrics are used to monitor containers, pods, services, jobs, and the status of the Kubernetes Objects in alignment with the orchestration of the overall cluster. Kubernetes provides information regarding resource usage via Resource Pipeline and information about the status of Kubernetes Objects via Kube State Metrics Pipeline. See the below mechanisms by which data is exposed for collection:

Resource Metric Pipeline provides the amount of resources currently used by a given node or pod and a limited set of metrics related to cluster components such as the HorizontalPodAutoscaler controller, as well as the kubectl top utility.

These metrics are collected by metrics-server and exposed via the Metrics API (metrics.k8s.io API). Metrics-server discovers all nodes on the cluster and queries each node's Kubelet for CPU and memory usage. The Kubelet fetches the data from cAdvisor and aggregates it on the metrics-server (lightweight short-term in-memory store).

In summary, metrics-server is a cluster-level aggregator component that periodically scrapes metrics from all Kubernetes nodes served by Kubelet through Summary API (exposed by cAdvisor). The metrics are aggregated, stored in memory, and exposed in Metrics API format (Prometheus format). The metric-server stores the latest values only.

Kube-state-metrics service listens to the Kubernetes API server and generates metrics about the state of the objects.It doesn't focus on system health or resource metrics but on the status of Kubernetes Objects, such as deployments, nodes, pods, etc. Kube-state-metrics holds an entire snapshot of Kubernetes state in memory and continuously generates new metrics based on it. Kube-state-metrics are also exposed via Prometheus format ("/metrics" endpoints). See below for a list of monitored objects:

- CronJob
- DaemonSet
- Deployment
- Job
- LimitRange
- Node
- PersistentVolume
- PersistentVolumeClaim
- Pod
- ReplicaSet
- ReplicationController
- ResourceQuota
- Service
- StatefulSet
- Namespace

- Horizontal Pod Autoscaler
- Endpoint
- Secret
- ConfigMap

Monitoring pipeline is provided by third-party solutions to augment monitoring scope beyond what is covered with Kubernetes pipelines:

- **Additional monitoring targets:** include master node, Kubernetes components (scheduler, controller, etcd, kubelet, kube-proxy, coredns, and other add-ons) and container workloads (applications, services, jobs, etc.).
- **Additional sources of data (high-fidelity data):** include systems, network, APM, and logging collection.
- **Additional methods:** include Prometheus pull method and support for pushed data (events).
- **Additional data types:** besides numeric metrics, Prometheus format (float64) also supports integer, string, and boolean types—particularly important for monitoring services—Error code, and true/false output.
- **Higher precision:** Prometheus supports milliseconds, but for certain types of applications (in finance and gaming markets or forensics, for instance), higher precision could be required.

You can implement a third-party monitoring pipeline in multiple ways in Kubernetes environments:

    ○ **As a local monitoring agent** on each Kubernetes cluster node, deployed as a DaemonSet—the agent can directly collect node, system, and container data and push it to a monitoring solution or expose it for polling (polling is associated with a repetitive interval when you pull data).
    ○ **As a sidecar agent**, deployed and paired with the containerized application in a pod encapsulation. They share volume and network namespace. This is particularly useful for custom instrumentation data (application needs unique plugins/monitoring agent configuration). It's also important to isolate the application monitoring impact on overall monitoring (a large amount of metrics exposure, for instance). High volume/intense communication to pull metrics can be prohibitive to the performance of monitoring solutions and resource consumption. As an additional benefit, sidecar deployment promotes accountability in monitoring.
    ○ **As a Prometheus scraper/central monitoring agent** to discover and scrape cluster-wide exposed metrics on Prometheus (/metrics) endpoints. For instance, a Prometheus server deployed in the cluster or a central monitoring agent. However, particular to scraping-type monitoring, cluster-wide monitoring is subject to /metrics cardinality impact, so one may have to consider partitioning Prometheus-type monitoring into separate sets to cope with

the /metrics endpoint growth. In general, it would be better to deploy local sidecar agents to isolate applications with intense communications (constant flow of pulled or pushed metrics) and a central monitoring agent as an aggregator to pipeline all customer monitoring data to the central repository.

The HA central repository—for visualization of all monitored data in dashboards—and retention of historic data for as long as needed are fundamental supports for an effective monitoring pipeline.

One scalable platform, full coverage, HA, and cost-effectiveness

Monitoring Prometheus endpoints is straightforward: you scrape API server for the list of exposed "/metrics" endpoints, pull these exposed metrics, and store them in a time series database. The time series database of choice could be Prometheus itself or another TSDB that is Prometheus compatible, like InfluxDB. InfluxDB is particularly interesting if scalability, high availability, multiple collection methods (pull and push), as well as types of monitored data (metrics, events, and logs) and analytics across them, are required.

Usually, Prometheus /metrics endpoints are not the only type of monitoring needed because applications provide data in multiple ways and formats. Furthermore, the nature of the data to be captured also varies. For instance, an event (error, user action, specific output of interest, etc.) is best monitored if pushed to the monitoring system when it occurs.

Additionally, some applications don't expose metrics, only push them. Prometheus server has a pull-only model, and therefore, doesn't support monitoring of events or pushed metrics. Besides leaving a whole category of monitoring data without a mechanism to collect, Prometheus also doesn't provide the flexibility to balance between regular pulling interval periods and pushing data, which can be instrumental in managing the impact of monitoring on your resources.

When you are left with only one method, hard compromises must be made. On the one hand, if the polling interval is very fine to capture changes in real-time, a heavy toll on system and storage resources is claimed. On the other hand, large intervals could leave critical gaps in data collection. Therefore, a monitoring solution must support both polling of regular metric data and pushed data to be effective and practical.

Other important aspects to consider are the volume of data ingested, analytics, and query performance. To keep up with the fragmentation and complexity of container environments, large volumes of monitoring data are generated and correlated for detection and fast diagnosis. You can expect at least an order of magnitude increase in monitoring data volume with containerized applications.

As a result, monitoring not only consumes more resources but also requires performance at scale. That is where a purpose-built time series database makes a significant difference—both as an enabler of monitoring at high ingest and cost-effective storage. InfluxDB stands out as a preferred choice for a time series database to balance cost and performance.

What makes InfluxDB stand out, even among monitoring solutions with support for high-fidelity data (metrics, events, traces, and logs), includes:

- Support for real-time monitoring use cases
- Integration with tools for data processing and visualization
- Data persistence using object storage to allow for cost effective historical data storage and analysis

These capabilities make InfluxDB a great choice for organizations looking to remain at the forefront of intelligent alerting, trend analysis for anomaly detection, and automation. Furthermore, as InfluxDB scales horizontally, it accommodates a centralized monitoring approach, ingesting data from multiple locations and cloud deployments.

In addition to these benefits, InfluxDB, like Kubernetes and Prometheus, is an open source project and can be used as a time series database and data source for many other solutions and open source projects, such as Grafana.

InfluxDB adds to Kubernetes the means to detect in real-time the impact of workloads while learning and predicting from historical data.

Features to look for in a Kubernetes monitoring solution

While migrating applications to Kubernetes, one must think beyond the benefits to account for what else a successful migration requires. While continuously working to keep the clusters in the desired state, Kubernetes may be masking dangerous trends. For instance, its effectiveness in fixing undesirable situations makes continued and intermittent faulty behavior invisible. Kubernetes scheduling logic also may not consider all the information required to assess the effectiveness of deploying or removing pods.

Monitoring solutions need to go through the same considerations on key aspects of monitoring production environments, ensuring they are suitable for:

- **Scalability:** horizontally scalable to handle the required data granularity from the whole Kubernetes cluster
- **High availability:** able to deploy in fully replicated instances
- **Long-term retention of monitored data:** durability of data stored for as long as needed

- **Real-time detection and quick diagnosis of issues:** support event and fine-grained monitoring, as well as data analytics and correlation for root-cause analysis
- **Multiple data types support:** integers, floats, strings, and booleans are especially important for ingestion of desired data without pre-processing to reformat it
- **Multiple-source, high-fidelity data and business KPIs:** support for monitoring and cross-analytics of data from multiple sources (systems, network, APM, logs) as well as composed business indicators
- **Security:** support for secured and authenticated monitoring

In terms of monitoring goals, refer to the following considerations:

Kubernetes core services and cluster metadata (kubelet API and kube inventory via API Server): in general terms, the health and status of a Kubernetes cluster

- **Make sure that Kubernetes components are functional and performant**, including API server, Metrics server, etcd, scheduler, controller, kubelet, kube-proxy, CoreDSN, and other add-ons in use.
- **Make sure that Kubernetes platform is effective**—monitor orchestration events such as pod destruction and creation, autoscaling, service and namespace creation, etc.
- **Examples of what to monitor:**
  - Deployments status: desired, current, up-to-date, available, age
  - StatefulSets status
  - CronJobs execution stats
  - Kubernetes components health checks
  - Kubernetes events
  - API Server requests
  - Etcd stats

Node-level, pod-level, and container-level metrics: standard telemetry on Kubernetes hosts and services

- **Make sure that resources are available and at adequate capacity utilization:** master node, nodes, pods, and containers.
- **Examples of what to monitor:**
  - Resource utilization per node (CPU, memory, disk, network bandwidth)
  - Node status (ready, not ready, etc.)

    - Number of pods running per node
    - Container health, i.e., status, restarts
    - Container resource utilization (CPU, memory, disk, network bandwidth)
    - Pods health, i.e., instances ready, status, restarts, eviction
    - Pod resource utilization (CPU and memory)
    - Mounted volumes stats

Application metrics from applications "contained" in pods: monitor the container workload

- **Make sure that applications running in containers are running without issues or performance degradation:** monitor exposed metrics, pushed events and logging files.
- **What to monitor depends on the application**
    - Health/up/down check
    - HTTP requests (queue, latency, response code, etc.)
    - Outgoing connections (e.g., database connections)
    - Number of threads

High-fidelity data: this is collected in relation to full application stack (system, network, APM). log, and Kubernetes cluster metadata

- **Make sure that no trend of harmful events goes undetected, benefiting from Kubernetes automation:** for instance, correlate system and application metrics with Kubernetes object's state events.
- **Make sure that issues are quickly diagnosed and remediated:** collect log data correlated with performance degradation.

What makes automation reliable is a comprehensive monitoring approach that keeps visibility over all layers of the application environment, observes behavior over time, and, in case of issues, provides quick access to data for expedited diagnosis and time to recovery. From that perspective, a more effective solution would integrate monitoring data from infrastructure, network monitoring, application performance, and logs in one central platform, where you can observe a holistic view of the Kubernetes environment.

InfluxDB Kubernetes monitoring

As noted in previous sections of this paper, InfluxData's time series platform, InfluxDB, supports pull and push of metrics, events, and logs from multiple sources, including all Prometheus /metrics endpoints. Additionally, it provides real-time stream analytics for efficient ingestion and effective alerting.

Telegraf is InfluxData's open source, plugin-based collecting agent with more than 200 plugins covering most common systems and applications found in IT infrastructure. Telegraf can be configured in listening and scraping modes, supporting both push and pull methods. This makes Telegraf a versatile agent.

Some plugins are specially crafted for Kubernetes monitoring:

- Telegraf Kubernetes Input Plugin: `[[inputs.kubernetes]]` collects data from Kubelet API (node, pod, and container monitoring)
- Telegraf Kubernetes Inventory plugin: `[[inputs.kube_inventory]]` collects kube state metrics (nodes, namespaces, deployments, replica sets, pods, etc.).
- Telegraf Prometheus Input Plugin: `[[inputs.prometheus]]` reads metrics from one or many Prometheus clients (array of urls or services to scrape metrics from /metrics endpoints). It scrapes Kubernetes pods for Prometheus annotations to discover exposed endpoints.

Here are some baseline plugins that complement or supplement monitoring breadth:

- Telegraf Internal plugin: `[[inputs.internal]]` collects its own systems metrics (cpu, disk, mem, etc.) and aggregates agent stats on all Telegraf plugins (input and output). By doing so, Telegraf provides visibility in the monitoring pipeline itself, such as:
  - Gather_errors
  - Metrics_dropped
  - Metrics_gathered
  - Metrics_written
- Telegraf Docker plugin: `[[inputs.docker]]` collects Docker container run-time events. It can be used as an alternative to cAdvisor /summary API.
- Telegraf System plugin: The System input plugin gathers general stats on system load, uptime, and number of users logged in. It is basically equivalent to the UNIX uptime command.

Here are some commonly used plugins for listening to event and log messages, as well as retrieving/parsing data from files:

- Telegraf HTTP Listener: `[[inputs.http_listener]]` listens for messages (in InfluxDB line-protocol) sent via HTTP POST. The plugin allows Telegraf to serve as a proxy/router for the /write endpoint of the InfluxDB HTTP API. It can be configured for HTTPS (TLS Certs, TLS Keys, and MTLS).
- Telegraf Syslog plugin: `[[inputs.syslog]]` listens for syslog messages transmitted over UDP,

TCP, or TLS.

- Telegraf StatsD plugin: `[[inputs.statsd]]` is a special type of plugin that operates a background StatsD listener service while Telegraf is running. StatsD messages are formatted as described in the original Etsy StatsD implementation.
- Telegraf Kafka Consumer plugin: `[[inputs.kafka_consumer]]` reads from Kafka and creates metrics using one of the supported input data formats*. Kafka can be a suitable choice for event sourcing microservices that keep track of a sequence of events.
- Telegraf Tail plugin: `[[inputs.tail]]` "tails" a logfile and parses each log message. The plugin expects messages in one of the Telegraf Input Data Formats*.
- Telegraf File plugin: `[[inputs.file]]` updates a list of files every interval and parses the contents using the selected input data format. Files are always read in entirety; if you wish to tail/follow a file, use the tail input plugin instead.
- Telegraf Multifile plugin: `[[inputs.multifile]]` allows Telegraf to combine data from multiple files into a single metric, creating one field or tag per file. This is often useful for creating custom metrics from the /sys or /proc filesystems.
- Telegraf Logparser plugin: `[[inputs.logparser]]` streams and parses the given logfiles. Currently, it can parse "grok" patterns from logfiles, which also supports regex patterns.
- Telegraf Socket Listener plugin: `[[inputs.socket_listener]]` service input plugin that listens for messages from streaming (tcp, unix) or datagram (udp, unixgram) protocols. The plugin expects messages in the Telegraf input data formats*.

*** Telegraf input data formats:**

  - InfluxDB Line Protocol
  - Collectd
  - CSV
  - Dropwizard
  - Graphite
  - Grok
  - JSON
  - Logfmt
  - Nagios
  - Value, i.e.: 45 or "booyah"
  - Wavefront

Telegraf monitoring ecosystem

IT Ops can fully benefit from a vast list of Telegraf plugins (200+) and clients to meet their needs for a full-stack application environment and cloud monitoring solution. Below are some of the Telegraf plugins available for monitoring:

- **Network:** Ping plugin, HTTP Response plugin, SNMP plugin, JTI OpenConfig
- **System:** CPU plugin, Disk plugin, DiskIO plugin, Mem plugin, Net plugin, Netstat plugin
- **Application:** Apache HTTP Server plugin, Apache Tomcat plugin, Postgres plugin, MySQL plugin
- **Message Queue/Bus:** Kafka consumer plugin, AMQP
- **Cloud monitoring:** Google Cloud Pub/Sub plugin, Amazon Kinesis Consumer

Also available from InfluxData is a list of plugins that collect metrics and events from tools, such as Nagios and Icinga. There are also plugins for a unified network monitoring solution comprising network availability, performance, traffic-flow, and application performance monitoring (APM). Collected data from multiple sources can be sent to Kapacitor for alert processing and data transformation and visualized via Chronograf or Grafana dashboards.

Telegraf agent deployment options

In Kubernetes cluster environments, Telegraf can be deployed as a DaemonSet in every node, an application sidecar in pods, or a central agent.

Telegraf deployed as DaemonSet agent

Telegraf can be installed as a DaemonSet in every Kubernetes node in the same way that Prometheus Node_Exporter is. The only difference is that Telegraf doesn't expose collected node metrics for posterior scraping, it pushes the data to InfluxDB.

All agents deployed as a DaemonSet are configured from the same ConfigMap (same active plugins and configuration). The Daemonset will run a Telegraf Pod on every node in a cluster, collecting node, system, and logging data.

Telegraf deployed as a scraper or central agent

Telegraf can run on a cluster node in a non-DaemonSet deployment. It will be in its own pod. In this configuration, we call Telegraf a "scraper" or central agent. It could discover and collect all exposed /metrics endpoints in the Kubernetes environment. Telegraf in a scraper deployment has its own configuration file (ConfigMap) with Prometheus plugin. The Prometheus plugin can be configured to scrape

Pod annotations written into deployment configurations in Kubernetes to discover pods that expose data. At intervals, Telegraf will grab metrics from the “/metrics” endpoints discovered, providing the same auto-discovery that Prometheus offered. As discussed previously, a central scraper may face performance issues as the number of /metrics endpoints increases with the metrics exposed by applications. Consider a sidecar approach for more accountability and to isolate impact from applications that generate a lot of monitoring data.

Telegraf central agent can also be used as an aggregator for more efficient writing to the database. It could receive data from other Telegraf instances and batch it to InfluxDB.

Telegraf deployed as a sidecar agent

Telegraf can be installed as an application sidecar in Kubernetes pod deployments. As a sidecar, Telegraf is in the same network as the containers inside the pod and will be sharing storage resources. This setup provides a much faster, safer, and less resource-impacting means to collect (pushed or pulled) monitoring data from Kubernetes workloads. It is a more controlled, modular, and composable setup. For instance, instead of exposing a lot of /metrics endpoints for scraping, the application running in the container could send data straight to Telegraf sidecar instance by writing events inside a socket (Telegraf Socket Listener plugin) or a file (Telegraf Tail plugin).

Another benefit of this deployment option is to be specific to the monitoring needs of the application running on the containers inside the pod. Sidecar deployment is also a good option for collecting logging data (from files) and monitoring other types of solutions deployed as sidecars, such as service mesh proxies.

See below an overall Kubernetes monitoring diagram with InfluxDB and various options for Telegraf deployment.

How InfluxData facilitates working with Kubernetes

InfluxData has added Kubernetes-specific capabilities to make it easier for its users to work with Kubernetes:

- **Telegraf Kubernetes input plugins** - Telegraf Kubernetes Input Plugin, Telegraf Kubernetes Inventory plugin, Telegraf Prometheus Input Plugin.
- **Telegraf plugin for Prometheus endpoint scrapping and service discovery** - Telegraf Prometheus Input Plugin discovers and gathers metrics from HTTP servers exposing metrics in Prometheus format (/metrics endpoints).
- **High availability (HA) and scalability of monitored data** - A large volume of metrics and events

can be preserved in InfluxDB storage clusters, allowing long-term policy retention together with high data granularity and high series cardinality.

- **Integration with Prometheus Server for long-term storage and HA** - Kubernetes native monitoring is based on Prometheus local storage. Nonetheless, scalability and durability are beyond Prometheus's scope. The Prometheus project includes a set of interfaces that integrate with remote storage systems, providing scalability and durability assurances while preserving existing Prometheus PromQL/Grafana dashboards. InfluxDB integration with Kubernetes Prometheus monitoring is supported in three ways:
- **Remote Write API:** Prometheus can write samples and ingest them to InfluxDB in a standardized format.
- **Remote Read API:** Prometheus can read (back) sample data from InfluxDB in a standardized format.

Kubernetes embraces the openness, agility, and extensibility required to keep up with the demands of developers, operations engineers, managers, and users of applications. However, to deliver its value, Kubernetes requires an equally open, agile, and extensible monitoring platform that can keep up with new challenges:

- Keep track in real-time of performance metrics and events of short-lived, ephemeral workloads
- Facilitate diagnosis and root-cause identification of issues and performance degradation in high-complexity fragmented environments
- Keep performance at scale while cost-effectively supporting the exponential increase of monitoring data volume
- Provide complete visibility from resource utilization to application and business performance indicators
- Provide scalability and high availability of monitoring data for long-term or permanent retention

InfluxData—monitor more than just Kubernetes

What and how you collect and monitor shouldn't be limited by the tools you use. On the contrary, tools must create opportunities to expand visibility horizons in your infrastructure, networks, and applications. InfluxData's platform gives you total flexibility for instrumentation, analysis, and visualization of your entire environment. At its core is InfluxDB, which is purpose-built for time series to empower developers and enterprises to accomplish more—and with more efficiency.

With a motivated global developer community, InfluxData is continuously pushing the edge of time series processing and storage, embracing the cloud's demands and challenges for applications. InfluxData delivers a solution that meets the requirements of even the most sensitive and demanding sectors like finance and service providers. The combination of Kubernetes with InfluxDB's time series platform provides the monitoring foundation to support organizations on their journey to growth and modernization.

> *"One company we work with had over 50 tools just for monitoring their systems and services, implemented in different ways in different regions. So a red alert in one area didn't mean the same thing as the red alert in another area. Within all that noise, there's no real way to figure out how your systems are performing, alerts are often missed or lost, and there's no way to re-determine when a failure is approaching."*
>
> ***Sean Mack**, CEO, XOps*

Reasons to choose InfluxDB over SaaS monitoring services

1. SaaS solutions do not provide Continuous Intelligence – The challenges that businesses face to stay ahead of their competitors is to ensure their services can stand up to the high availability requirements demanded by their customers. Gartner recently published research on a topic they call Continuous Intelligence, where they encourage businesses to combine event stream processing, real-time data analytics, and artificial intelligence (AI) to get the real-time situational awareness required to keep services available. This is where SaaS monitoring solutions fail to deliver as they lack the extensibility capabilities to integrate effectively with AI/ML solutions to deliver business insights.

> *"Low-level monitoring cannot reflect business logic processes in an adequate way because there are interdependencies between various component services, especially with third-party services, where low-level monitoring will not give you information about business processes. Furthermore, we found that by mapping all product, clients, sites, with organizational details (who is responsible for a product, which engineer created the feature etc) with the metric pipeline from our production times series db, we can show Business Intelligence data in near real time."*
>
> ***Aleksandr Tavgen**, Technical Architect, Playtech*

2. Vendor lock-in risk — SaaS monitoring solutions are able to ingest the data for monitoring and dashboarding. However, they often have restrictions on retention policies or do not have the means to easily pull out your data. And this will be important to your organization as you look to use this data for trend analysis and forecasting. Restricted access means limits on what data you can get, how many times you can get it and even paying to get access to your own data. Even then, there is no guarantee that the SaaS solution will be able to provide you all the data that was ingested and stored.

> *"We had restrictions on how much data we could retain to do trend analysis. We had our data locked into data storage with certain commercial vendors, and we were not able to extract the data to do analysis."*
>
> ***Sanket Naik**, VP Cloud Operations & Security, Coupa Software*

3. Lack of flexibility – SaaS solutions are not flexible, requiring users to follow a prescribed path which can be time-consuming. Some SaaS solutions, for example, require users to tag every metric which users find restrictive and time-intensive. Additionally, SaaS solutions do not support the use cases which require monitoring information such as dashboards to be embedded inside another application. They provide limited capabilities to embed their dashboards into other applications. Lastly, when it comes to connectivity to other apps, SaaS solutions offer pre-defined connectivity to specific apps and it becomes a challenge to monitor solutions they do not connect to. This often leads to situations where parts of the infrastructure simply cannot be monitored as the SaaS vendor does not have an agent for them.

> *"We have a lot of custom data sources that we want to be confident that we have the power and flexibility to use a tool rather than getting another locked down proprietary SaaS system like we have used in the past. In addition, they didn't give us the power of having access to the database itself, it was nowhere near as fully featured, and it also did not scale anywhere near as well, particularly when it comes to cost."*
>
> ***Jack Tench**, Senior Software Engineer, NewVoiceMedia*

About InfluxData

InfluxData is the creator of InfluxDB, the open source time series database. Our technology is purpose-built to handle the massive volumes of time-stamped data produced by IoT devices, applications, networks, containers, and computers. We are on a mission to help developers and organizations, such as Cisco, IBM, PayPal, and Tesla, store and analyze real-time data, empowering them to build transformative monitoring, analytics, and IoT applications quicker and to scale. InfluxData is headquartered in San Francisco, with a workforce distributed throughout the U.S. and across Europe. Learn more.

InfluxDB documentation, downloads & guides

Get InfluxDB

Try InfluxDB Cloud for Free

Get documentation

Additional tech papers

Join the InfluxDB community

influxdata®

Try InfluxDB

Contact Sales

Contact us for a personalized demo
influxdata.com/contact-sales