Monitoring Kubernetes

CONTENTS

WRITTEN BY STEFAN THORPE
HEAD OF DEVOPS AND SECURITY AT CHERRE AND CTO AT CAYLENT

Kubernetes: A Brief Overview

Kubernetes allows you to deploy and manage cloud-native applications with more flexibility than any container orchestration platform has ever provided before. Kubernetes, affectionately referred to as Kube or K8s for short, is an open-source platform for developers that eliminates many of the manual processes involved in managing, deploying, and scaling containerized applications.

The fault-tolerant and scalable platform can be deployed within almost any infrastructure. Kubernetes will integrate with anything, from public clouds (Google Cloud, Microsoft Azure, Amazon Web Services, and so on) to private or hybrid clouds to server clusters, data centers, or on-premises — or, indeed, any combination of these solutions. The remarkable fact that Kubernetes supports the automatic placement and replication of containers over such a large number of hosts is no doubt one of the many reasons why the platform is becoming the de-facto stage for running microservices infrastructure in the cloud.

THE KUBERNETES ORIGIN STORY

Kubernetes was initially conceived and developed by the engineers at Google. Open-sourced since its launch in mid-2014, K8s is now maintained, upgraded, and managed by a large community of contributors that includes software titans like Microsoft, Red Hat, IBM, and Docker.

As well as being publicly open about how the company's entire infrastructure runs in containers, Google was an early contributor to Linux container technology (the tech that powers all of Google's extensive cloud services). Kubernetes originally came into being through the cloud platform's 15 years of experience running production workloads, and is designed to handle tens, thousands, and even millions of containers. Google itself initiates more than 2 billion container deployments weekly. At one point, it ran everything through its internally developed platform: Borg.

Borg was a large-scale internal cluster management system and Kubernetes' predecessor. The internal program ran hundreds of thousands of jobs from a multitude of different applications across countless clusters, each with a vast number of machines. Much of the education garnered over the years from developing Borg was distilled into the Kubernetes technology we see today.

Alongside the release of Kubernetes v1.0 in July 2015, Google partnered with the Linux Foundation to establish the Cloud Native Computing Foun-

Sensu

Future-proof, multi-cloud monitoring at scale.

Monitor your entire infrastructure,
from Kubernetes to bare metal.

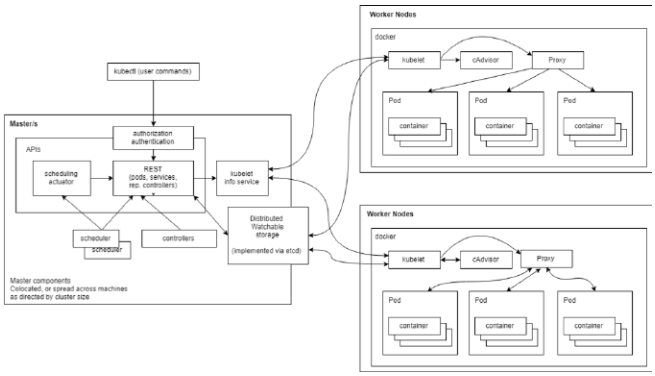
Download Sensu Now

dation (CNCF). The CNCF's purpose is to build sustainable ecosystems while fostering a community around a network of high-quality projects that orchestrate containers within either a monolithic or a microservices architecture. Kubernetes is one of the highest velocity CNCF projects in the history of open source.

HOW KUBERNETES WORKS

Kubernetes, at its fundamental level, is a system for coordinating containerized applications across a cluster of machines. It's designed to allow developers to fully manage the lifecycle of containerized applications and services using features that deliver predictability, scalability, and high availability. Simply put, K8s helps developers pilot container clusters, hence the name, which means helmsman (or pilot) in Greek. It is the most popular option available today because it provides an abstraction to make a cluster of machines act identically to one big machine — a vital feature when running large-scale environments.

By default, Kubernetes optimizes the use of Docker to run images and manage containers. However, K8s can also use other container engines, such as rkt from CoreOS, for example.



This architecture diagram shows how Kubernetes works. It outlines a set of master components, including "pods," that model an application-specific "logical host." A Kubernetes pod envelops an application container (or, sometimes, multiple containers), storage resources, an exclusive network IP address, and details about how the container(s) should run. They are the smallest atomic unit for containers. Theoretically, pods contain one or more tightly coupled applications — with best practice aiming for one container per pod. More on pods later.

Essentially, the process includes an API server, a scheduler, and controllers. The API server exposes the Kubernetes API and processes REST operations as well as subsequent updates. The scheduler attaches the undeployed pods with a suitable VM or physical machine. If none are available, the pod stays unscheduled until it can identify an appropriate node. The master runs the other cluster-level functions through multiple embedded controllers to achieve endpoint creation, node discovery, and replication control, among other operations. Because this controller design is flexible and extendable, Kube admins can build their own controllers. Kube tracks the shared state of the Kubernetes cluster via the API server and implements changes to make the current state and the desired state of the cluster correspond.

As well as being truly container runtime agnostic, Kubernetes' features include everything necessary to centrally automate, control, and scale containerized applications. Here are the highlights:

- Enterprise-scale container deployments
- Rollout management
- Built-in service discovery
- Self-healing
- Autoscaling
- Persistent storage management
- High-availability features
- Interconnected clusters
- Resource bin packing
- Secrets

These highlighted features are why Kubernetes has become widespread for leveraging different application architectures from monolithic or batch-driven applications to highly distributed microservice applications.

Monitoring Kubernetes: The Challenges

Traditionally, servers were treated like pets, with "all hands on deck" being called to fix one that went down. With containerization, servers are numbered as cattle in a herd; when one goes down, it's simply replaced. Since its release in 2014, Kubernetes has revolutionized container technology and become a critical tool for quickly launching applications at scale.

However, challenges arrive hand-in-hand with innovation and speed, and they are not easy to overcome from the control panel. Containers produce a significant level of complexity when it comes to orchestration.

While Kubernetes has made it dramatically more accessible for everyone to implement and manage containers — including everything from scheduling and provisioning to automatically maintaining the desired state — there are key obstacles to overcome in terms of monitoring both Kubernetes itself and the applications running on it. Among these challenges are:

- **Interconnected applications distributed across multiple vendors:** At its core, Kubernetes serves as a universal platform that companies can use to deploy applications wherever they wish them to run. Typically, companies adopt a multi-cloud solution to remain cloud-agnostic and avoid vendor lock-in. Multiple cloud-hosted environments also come with their own inherent value of mitigating downtime and data loss, which is why it's so common. However, with assets, software, and applications distributed across multi-cloud solutions comes monitoring issues for extracting real-time data about the health of all widely dispersed resources.

- **Constantly moving applications on dynamic infrastructure:** With constant movement of multiple applications, it can be difficult to achieve complete visibility across every platform and protocol at any given moment. This can result in problems with isolating any bottlenecks or constraints in the system. As many established companies have multiple applications within their

infrastructure, this challenge is practically unavoidable. Without a robust monitoring system to capture the detailed picture, companies are vulnerable to visibility and buried problems that can burn through money and resources.

- **Numerous complex components to observe:** Akin to migrating from a monolithic application to a microservices architecture, deploying and monitoring even a small cluster on K8s can present a challenge. Kubernetes is a complex machine with a lot of components operating together, so adopting the platform means monitoring all of the following (and then some):
    - The capacity and resource utilization of clusters including:
        - Nodes: To ensure the state of all K8s nodes, track the CPU, memory, and disk utilization.
        - Deployments/pods: Confirm all implemented pods in a deployment are healthy.
        - Containers: Watch the CPU and memory consumption in relation to configured limits.
    - Applications: Track the performance and availability of all applications running in the cluster through request rates, throughput, error rate, and so on.
    - End-user experience: Supervise mobile application and browser performance to respond to insights and errors. Ensure customer satisfaction by following and improving load time and availability.
    - Supporting infrastructure: As mentioned above, it's vital to track Kubernetes clusters running across different platforms and multiple vendors.
- **Monitoring the minutiae of operational complexity:** All the Kubernetes core elements (i.e., the kubelet, API server, Kube controller manager, and Kube scheduler) involve numerous flags that determine how the cluster operates and performs. The default values may be fine for smaller clusters at the beginning, but as deployments scale up, it's important to make and monitor adjustments. It also becomes crucial to keep track of Kubernetes details such as labels and annotations.

Ironically, monitoring solutions can be responsible for bringing down clusters by pulling extreme amounts of data from the Kubernetes APIs. Therefore, it is a good idea to determine a key baseline for monitoring, and dial it up when necessary to further diagnose issues that require additional attention.

If a high level of monitoring is justified, it may be mindful to deploy more masters and nodes (which you'll need to do for HA anyways). Another practical technique, especially when it comes to large-scale implementations, is to implement a lone cluster for storing Kubernetes events. This way the performance of the main instances won't be affected by any volatility in event creation and event retrieval for monitoring purposes.

Monitoring Kubernetes from the Sources

Like most container orchestration platforms, Kubernetes comes with a set of elementary tools that monitor servers, though they aren't all exactly "built-in." With some additional minor tinkering, it's possible to leverage these base additional components to gain more visibility into Kubernetes from the get-go. These rudimentary options include:

- The Kubernetes Dashboard: Not deployed by default, this add-on is worth running for providing the basic visualization of the resources running on each deployed K8s cluster. It also delivers a primary means of managing and interacting with those resources as well as the environment.
- Container probes: These are diagnostics which actively monitor the health of a container. Each probe results in either one of three outcomes:
    - Success: The container passed the diagnostic probe.
    - Failure: If the probe determines that a container is no longer healthy, the probe will restart it.
    - Unknown: The diagnostic failed, so no action is necessary.
- Kubelet: The kubelet is a primary node agent that runs on every node in the cluster and ensures the containers are working. It's also the mechanism through which the master communicates with the nodes. The kubelet exposes many individual container usage metrics straight from cAdvisor.
- cAdvisor: This Kubernetes component is an open-source container resource usage and performance analysis agent that is integrated with the kubelet. It is purpose-built for containers, identifying all containers running in a machine to collect statistics about memory, network usage, filesystem, and CPU. cAdvisor operates per node, and also analyzes overall machine usage by assessing the "root" container on the machine. It is simple to use, but also limited:
    - cAdvisor only covers basic resource utilization; it doesn't have the capacity to analyze how applications are actually performing.
    - cAdvisor also doesn't provide any trending, long-term storage or analysis facilities.
- Kube-state-metrics: Another add-on service that listens to the Kubernetes API and translates information about Kubernetes constructs into metrics. In general, the model interrogates the Kubernetes API server and recovers data about the states of Kubernetes objects.
- Metrics server: The metrics server collects metrics from the Summary API, exposed by the kubelet on each node. It is a cluster-wide aggregator of resource usage data. As of Kubernetes v1.8, it's deployed by default in clusters created by *kube-up.sh* script as a deployment object.

The path of the core monitoring pipeline is as follows:

1. As cAdvisor is installed by default on all cluster nodes, it collects data metrics about those containers and nodes.
2. The kubelet exposes these metrics through the kubelet APIs. (The default is a one-minute resolution.)
3. The metrics server identifies all available nodes and requests the kubelet API to report all container and node resource usage.
4. The metrics server then exposes these metrics through the Kubernetes aggregation API.

Fundamentally, this pipeline outlines a sense of the initial steps necessary to incorporate reasonable monitoring for Kubernetes environments. While we still don't have detailed application monitoring through these rudimentary components, we can at least observe and monitor the underlying hosts and Kubernetes nodes, and receive some metrics about any Kubernetes abstractions.

Customarily, the average K8s cluster administrator is interested in monitoring from a holistic point of view, while application developers who build services tend to lean towards monitoring from an app-centric point of view. Recently, however, these lines have blurred. Regardless of the perspective required, all teams want to minimize the amount of input required to monitor their systems and manage data collection. We'll now take a look at two viable monitoring options that bring in full visualization and system-level metrics, allowing developers to track, troubleshoot, and receive alerts about the most crucial parts of K8s clusters.

Prometheus

OVERVIEW OF PROMETHEUS

The second major project to graduate CNCF's process from incubation to maturity, Prometheus is an open-source monitoring and alerting toolkit designed specifically for containers and microservices. Renowned for being a top systems and service monitoring system, Prometheus offers robust features that benefit all stakeholders involved in the development pipeline, including cloud administrators and developers. It aggregates metrics from configured targets at specified intervals, assesses rule expressions, presents the results, and can provoke alerts if certain conditions are observed to be true.

Prometheus delivers support for instrument application in multiple languages. The tool also utilizes exporters to collect and expose telemetry for services that can't be instrumented with the Prometheus instrumentation client or library (PostgreSQL, MySQL, AWS CloudWatch, ETCD, and Kubernetes itself).

Prometheus does more than simply monitor predefined metrics. It is capable of implementing dimensional data models for truly in-depth analysis, creating relational insights based on multiple metrics. In other words, developers and administrators can get a clear view of Ops from different angles.

HOW PROMETHEUS WORKS

Prometheus is implemented as an additional layer into your Kubernetes environment. When first installing Prometheus, it's important to define a few parameters that allow the system to scrape and collect data, starting with the scrape interval. Prometheus can be configured to provide either real-time monitoring or interval-based analysis of Kubernetes nodes.

By assigning external labels and attaching them to time-series alerts, it's also possible to use Prometheus as an alert system for node failures and other issues. The rest of the system is defined through scrape configurations.

THE PROMETHEUS OPERATOR

The difference between controllers and operators often confuses many users. To deconstruct the two: a Kubernetes *Operator* is the name of a pattern that involves a *Controller* adding new objects to the Kubernetes API to configure and manage applications such as Prometheus. Put simply, an operator is a domain-specific controller.

"The Prometheus Operator serves to make running Prometheus on top of Kubernetes as easy as possible, while preserving Kubernetes-native configuration options." ("Prometheus Operator", 2019)

Using the Prometheus Operator allows for easy monitoring of K8s services and deployments. It is viable to run Prometheus using a predetermined configuration .yml file, but this can later be automated according to how Kubernetes is set up. The Prometheus Operator creates, configures, and manages all monitoring instances atop Kubernetes. When deploying a new version of an app, K8s creates a new pod (container). Once the pod is ready, Kube destroys the old one. Prometheus is on constant watch, tracking the API; when it detects any differentials, it creates a new Prometheus configuration based on the services or pods changes.

CORE COMPONENTS

Prometheus is a robust monitoring tool that uses a pull-based architecture — as opposed to *pushing* metrics to the monitoring tool, it *pulls* metrics from services. Prometheus also has a push gateway, meaning it does support "push" for certain metrics when the pull model doesn't work (though this method is discouraged).

In addition, Prometheus allows you to connect time series with metric name and key-value pairs, simplifying monitoring in a complex, multi-node cloud environment. As well as being able to see the big picture, it's also possible to dig deep into an individual microservice easily with Prometheus.

The tool also comes with PromQL, for easy processing of time-series data. You can use queries to manipulate data and generate insights relevant to your situation. You can also use PromQL to create graphs, visualize datasets, create tables, and generate alerts based on specific parameters.

Prometheus' web-based console lets you manage all features and tools available inside Prometheus. You can use regular expressions and advanced PromQL queries for creating datasets and alerts, and it's accessible from the outside (when the cloud infrastructure is set up to enable remote access).

BENEFITS OF THE PROMETHEUS APPROACH

Simplicity and flexibility are two of Prometheus' strongest suits. When using Prometheus as a monitoring framework on Kubernetes, it's possible to implement a highly dimensional data model for monitoring. Prometheus is also available as a Docker image, which means it is easy to set up the monitoring framework as part of a containerized cluster. The tool also integrates well with Grafana for increased monitoring visualization.

When used with Kubernetes, Prometheus can also collect node, pods, and services metrics using the native service discovery configurations of Kubernetes. There is no need to jump through hoops just to get a comprehensive monitoring system up and running; users can jump straight to defining expressions and creating alerts.

There is also Prometheus' simple and fluent nature. The scraping capability of Prometheus can be easily integrated with Kube and other compatible tools, including Docker and StatsD. In more complex environments, Prometheus works well with *kubernetes-service-endpoints* and API servers. Even better, you can use the web GUI to configure alerts and graphs on the fly.

As with any pure telemetry monitoring solution, however, you miss out on context with Prometheus' constrained data model. The fact that Prometheus' data collection model defaults to time series presents a double-edged sword. On one hand, you can easily standardize collecting data in a particular way, which helps simplify things; on the other, you've now limited yourself to a constrained data model that might be missing some vital context.

The pull-based model also gives you limited granularity, as exporters only provide summarized data that's only scraped periodically. This model also requires that you punch holes in the firewalls of traditional networks, and therefore is not suited for monitoring legacy or multi-generational infrastructure. Because Prometheus uses a set of discovery libraries to stay up to date with the platforms it monitors (like Kube), a degree of lag is introduced as resources come and go, which in turn is exaggerated by metric scraping intervals. Finally, Prometheus collection is unauthenticated and unencrypted — anything that has access to the network can observe your telemetry data. (This includes metrics labels — you add labels for context, but then that context would be out in the open.)

Sensu

OVERVIEW OF SENSU

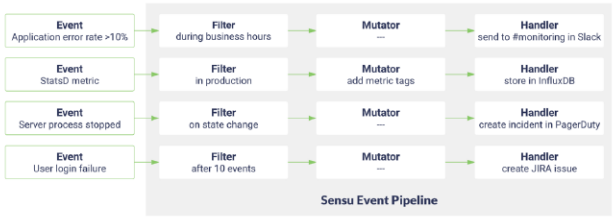
Sensu is a monitoring solution designed for multi-cloud and containerized infrastructure. One of the benefits that Sensu brings to the table is a composable monitoring event pipeline with an event-driven architecture. The Sensu agent is a cross-platform event producer, enabling you to execute service checks to monitor system and service health as well as collect and analyze unique metrics. Custom workflows enable you to go beyond alert or incident management (e.g., auto remediation), and monitoring APIs, client libraries, and plugins written in any scripting or programming language (including Nagios). Such customization offers a range of capabilities.

While Prometheus uses pull architecture for actively scraping data from various sources, Sensu leverages a hybrid of push/pull with its publish/subscribe (Pub/Sub) model — because Sensu uses a message bus for communication, you can *publish* messages to a specific topic and consumers can *subscribe* to one or more topics. This difference in approach makes Sensu equally as robust, but for different reasons. Both Prometheus and Sensu allow you to collect the same metrics, although Sensu's service health checking abilities (like the ability to monitor external resources with proxy requests and entities) can help fill gaps in a purely telemetry-focused approach. A better way forward, therefore, would be to run them in unison.

HOW SENSU WORKS

Sensu uses native collection plugins to collect data for analysis from a wide range of mainstream utilities (such as StatsD libraries, Prometheus exporters, Nagios plugins, SNMP traps, CollectD metrics, and so on). It differentiates itself from other tools, including Prometheus, by supporting multi-cloud environments out of the box and providing high availability immediately from when users begin configuring the framework.

Sensu provides building blocks such as event filters, mutators, and handlers for operators to use to model workflows and automate them. Via this method, data can be consolidated from external monitoring tools with existing operational ones to establish an event-based approach to monitoring. (Sensu Go)



The Sensu Go open-source project is governed by an MIT license, with source code that users are able to download and compile from github.com/sensu/sensu-go. Supported (built and tested) packages are available for download on the Sensu website, and are free to use up to 1,000 entities. Organizations that require monitoring at scale with enterprise-level integrations will need an enterprise license.

CORE COMPONENTS

For Kubernetes containers, Sensu optimizes its auto-discovery capabilities. It's easy to configure monitoring checks and collectors for the Kubernetes components, and you can do the same in other containerized environments, including Docker. It's also easy to configure a handful of checks for monitoring all Kubernetes components and the applications running on them.

Similar to Prometheus, Sensu also comes with native support for integration and plugins. It works with logging tools and plays well with Prometheus itself. It's possible to use the two frameworks running alongside each other and have them work together in processing different datasets. For example, Sensu can collect StatsD metrics and write them to Prometheus.

Sensu can also adapt to its Kubernetes environment. Let's say Sensu is deployed inside a container, and the decision is made to move the entire

app to another containerized environment. Complete the app migration, and Sensu will still work in a different environment. With its auto-native support, the Sensu agent will run at the new location and be discovered by the Sensu system. You don't need another integration — it has its own auto-discovery mechanism.

KUBERNETES MONITORING WITH SENSU

Sensu is a highly scalable monitoring framework for Kubernetes, one that will grow alongside deployed apps and cloud environments and provide detailed functional monitoring. There is no upper limit to using Sensu, even in terms of the level of complexity needed for monitoring requirements.

Sensu is a well-established, extensible solution with an active community of expert operators.

Sensu and Prometheus

Using Sensu and Prometheus together can improve operational visibility through monitoring. Both Sensu and Prometheus can run in tandem, enhancing the other's capabilities and filling in the gaps.

HOW IT WORKS

The Sensu Prometheus Collector is a Sensu Check Plugin that aggregates scraped data from a Prometheus exporter or the Prometheus query API. The collected metrics are issued to STDOUT in one of three formats:

- Influx (the default)
- Graphite
- JSON

The Sensu Prometheus Collector combines the workflow automation capabilities of Sensu and the power of Prometheus data scraping into a Prometheus metric poller. You can develop your own preferences for your instrumented code and for when you receive alerts. Sensu will also deliver the collected metrics to the external time-series database of your choice, such as InfluxDB, Graphite, or Prometheus.

INSTALL THE SENSU PROMETHEUS COLLECTOR

You can discover, download, and share assets using Bonsai, the Sensu asset index. To use the Sensu Prometheus Collector, select the Download button on the asset page in Bonsai to download the asset definition for your Sensu backend platform and architecture. Asset definitions tell Sensu how to download and verify the asset when required by a check, filter, mutator, or handler.

Once you've downloaded the asset definition, you can register the asset with Sensu using sensuctl, the command line tool for managing resources within Sensu, and create your own monitoring workflows.

For example, here's the Prometheus collector asset definition and associated sensuctl command for Linux:

```
sensu-sensu-prometheus-collector-1.1.5-linux-amd64.yml
---
type: Asset
api_version: core/v2
metadata:
  name: sensu-prometheus-collector
  namespace: default
  labels: {}
  annotations: {}
spec:
  url: https://github.com/sensu/sensu-prometheus-
collector/releases/download/1.1.5/sensu-prometheus-
collector_1.1.5_linux_amd64.tar.gz
  sha512:
b183e1262f13f427f8846758b339844812d6a802d5b98ef
28ba7959290aefdd4a62060bee867a3faffc9bad1427997
c4b50a69e1680b1cda205062b8d8f3be9
  filters:
  - entity.system.os == linux
  - entity.system.arch == amd64
```

```
sensuctl create -f  sensu-sensu-prometheus-collector-
1.1.5-linux-amd64.yml
```

And here's an example of a check using the Prometheus Collector:

```
example-app-metrics.yml
---
type: CheckConfig
api_version: core/v2
metadata:
  name: example-app-metrics
  namespace: default
spec:
  command: sensu-prometheus-collector -exporter-url
http://localhost:8080/metrics
  interval: 10
  output_metric_format: influxdb_line
  output_metric_handlers:
  - influxdb
  publish: true
  runtime_assets:
  - sensu-prometheus-collector
  subscriptions:
  - metrics
```

```
sensuctl create -f  example-app-metrics.yml
```

For the most up-to-date asset definition and usage examples, please check Bonsai: bonsai.sensu.io/assets/sensu/sensu-prometheus-collector

BENEFITS

The advantages of using Sensu and Prometheus in parallel through the Sensu Prometheus Collector include:

- As well as monitoring the health of your Kubernetes clusters, the two will dynamically monitor and collect service health checks and metrics for the surrounding infrastructure.
- Being able to go beyond problem detection for self-healing and streamline the monitoring process through workflow automation.
- With the two together, you get more context (what/where the event is from) through context-rich notifications and reporting.

- The ability to achieve more granularity and flexibility to cover the finer aspects of data scraping and analysis with more details.
- Sensu supports and uses standard cryptography for communication. Sensu's model allows for a single agent to collect and transmit data securely without having to compromise firewalls.
- Being able to easily manage and configure your monitoring setup.
- The capability to monitor your entire infrastructure from legacy to cloud-based infrastructure.

Conclusion

Essentially, leveraging Sensu can allow you to monitor your entire infrastructure — from Kubernetes to bare metal — with increased customization and context. Prometheus' robust model is focused on scraping flat telemetry data (at truly comprehensive levels), while Sensu offers visibility into your entire stack, working with industry standard technologies and formats (like Nagios and StatsD). You can use Sensu to complement Prometheus' extensive features for additional context from events, health checks (i.e., not just metrics), more comprehensive capabilities around processing monitoring data as a workflow, and a wholly secure solution.

Given how software-dependent we are today, availability is crucial for a business' survival, and downtime can prove expensive as well as damaging to an organization's reputation. Maintaining visibility into our systems is therefore essential to overcoming these challenges, and monitoring system infrastructure and applications has become integral to business operations. In order to fully reap the rewards that Kubernetes has to offer, implementing a unified, future-proof monitoring solution (i.e., one that bridges the gap between past, current, *and* future technologies) is critical.

References

- Prometheus Operator. (2019). Retrieved from coreos.com/operators/prometheus/docs/latest/user-guides/getting-started.html
- Sensu Go. (n.d.). Sensu Go 5.5. Retrieved from docs.sensu.io/sensu-go/latest/

Written by **Stefan Thorpe,** *Head of DevOps and Security at Cherre and CTO at Caylent*
Stefan is an IT professional with 20+ years management and hands-on experience providing technical and DevOps solutions to support strategic business objectives.

Devada, Inc.
600 Park Offices Drive
Suite 150
Research Triangle Park, NC

888.678.0399 919.678.0300