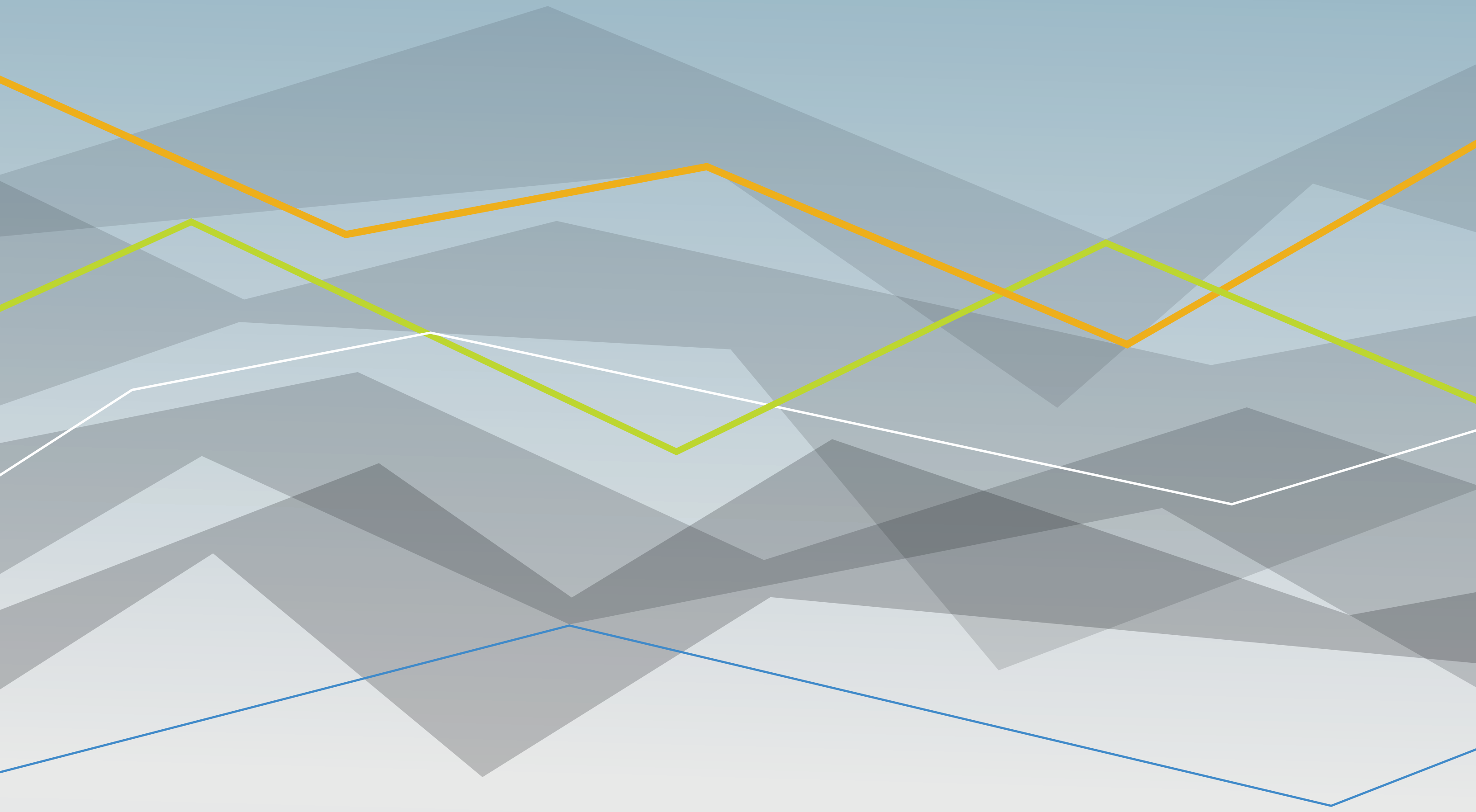
SignalFx



THE DEFINITIVE GUIDE: CONTAINER MONITORING and OBSERVABILITY

Introduction

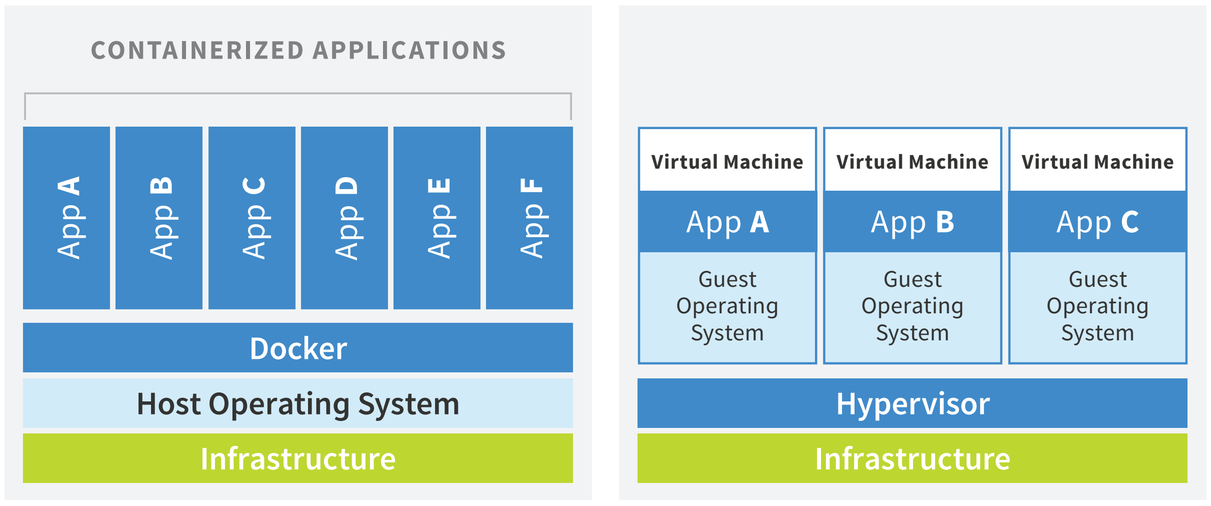
With more than 50 percent of the world now connected to the Internet via mobile phones, developing new applications has become a core requirement for any digital business. The advent of this core requirement has brought about a paradigm shift in the infrastructure space. As a part of their cloud journey companies are adopting cloud-native technologies to increase the speed of developing these new applications. Containers have become one of the most popular cloud-native infrastructure components to adopt allowing for faster speed of development and testing of new applications. Alongside container technologies like Docker, DevOps teams are also leveraging container orchestration technologies such as Kubernetes to manage and operate them at scale.

Containers 101

WHAT ARE CONTAINERS

Containers enable a new agile way to develop software that empowers developers to have application portability and the ability to run multiple apps on the same OS without sharing dependencies.

As you can see below, each container shares the host OS kernel and its binaries and libraries, making them extremely light (megabytes). With this level of abstraction, containers can be spun up and spun down in seconds, unlocking the ability to run immutable infrastructure.



Formats and Open Standards

In 2013, Docker launched as an open-source container runtime project — thus beginning the container adoption sprawl. Since then, a multitude of open standards has arisen around the technology. Some are complementary, others are competitive, but a few have become the cream of the crop. Here are some of the more popular open standards:

- **OCI** (Open Container Initiative) Managed by the Linux Foundation, OCI Standards are supported by many vendors and govern image and runtime specifications.
- **CNI** (Container Network Interface) A CNCF (Cloud Native Computing Foundation) project consisting of specifications and libraries for writing plugins to configure network interfaces in Linux containers.
- **Kubernetes CRI** (Container Runtime Initiative) While Docker is the most popular container runtime, the space continues to evolve. CRI enables developers to use a wide variety of container runtimes.

Running containers at an enterprise scale requires decisions about which container runtime to use, as well as which container engine, orchestrator, storage, and network solutions should be adopted.

Orchestration Considerations

Container orchestrators handle the deployment, maintenance and scaling of containerized workloads that assist companies in operating containers. Kubernetes (K8s), born two years after Docker launched, has taken the community by storm to become the dominant container orchestration technology. K8s was spun out of Google, and was built by engineers who were tasked with managing more than two billion containers that were spun up and down each week.

K8s is by far the most popular container orchestration technology. In Heptio's State of K8s Survey, 68% of developers surveyed said they chose K8s as their orchestration layer of choice when using containers in production. That said, you may want to consider different orchestrators depending on your specific environment and needs. Here's our view of your options.

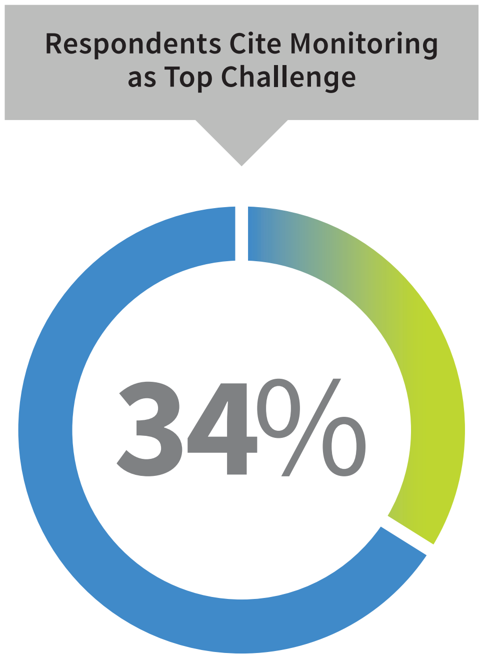
Orchestrator	Limitations	Strengths	Recommended Use
Kubernetes	• Challenging cluster setup and configuration • Load balancing must be configured manually (not too difficult)	• Market leader with 'production-ready' toolset • Feature-rich • Built for highly-scalable container environments • Able to operate newer technologies like serverless (Knative, Cloud Run, etc.) as well as containers	• On-prem and hybrid/ multi-cloud implementations • Leverageable for any size container environment • Stateless and composable workloads
HashiCorp Nomad	• Doesn't have a huge open source community yet • Limited feature set for Docker users	• Great to use for legacy and windows (stateful) workloads alongside containers • Simplistic cluster setup and configuration • Seamless use across multi-cloud and hybrid cloud infrastructure • General purpose and leverageable for various container runtimes	• May want to use if you are already using HashiCorp Consul and vault • If you are heavily leveraging container and virtualized technology outside of Docker
Docker Swarm	• No Auto-scaling capability • Limited admin controls • Relatively smaller OSS community than K8s	• Ease of cluster setup and configuration • Automatic load balancing	• Highly compatible with other Docker tools, so if you are all in with Docker Enterprise you may want to use Swarm
Marathon	• Challenging cluster setup and configuration • GUI no longer maintained by the open source community • Limited OSS community	• Great for long-running apps/ services • Persistent containers	• Good if you use Mesosphere DC/OS or Apache Mesos • Looking to run stateful containers

As you can see there are a fair amount of orchestrators to choose from. However, given the popularity of K8s, many of the cloud platform providers have created managed offerings of K8s. From AWS EKS, Azure AKS, Google GKE, and Redhat OpenShift, there are good alternatives if you're looking for a cloud provider that will deal with upgrading and patching K8s in the background for you.

When choosing a managed K8s solution, here are some things to consider: (1: what cloud provider you are currently on or plan to run on, (2: whether you plan to run workloads in a hybrid or multi-cloud fashion, and (3: pricing. While most managed K8s providers are releasing hybrid versions of their solutions, multi-cloud support remains negligible, driving many to continue to adopt Kubernetes as a standalone they manage themselves.

Container Monitoring and Observability Strategy Considerations

Choosing the right container monitoring and observability strategy can be as daunting as choosing the right runtime and orchestration technologies. It is one of the most critical components of getting your container architecture production-ready. If you're pushing new containers into production without a fool-proof monitoring and observability strategy in place, you're going to have problems. Monitoring and observability has become one of the highest cited concerns for DevOps teams that are adopting new cloud-native architectures, according to a CNCF survey of nearly 2,400 CNCF members.



Source: https://www.cncf.io/blog/2018/08/29/cncf-survey-use-of-cloud-native-technologies-in-production-has-grown-over-200-percent/

Below are considerations you should take into account when planning and executing your strategy.

Consideration #1: Integrations

There are many container technologies to choose from, just as there are many varieties of cloud-native technologies. Knowing how well your container technology integrates within your broader cloud-native stack is critical. For example, if your ElasticSearch data is in monitoring system A and your container telemetry data is in monitoring system B, there's not much chance of success. To ensure you have all of the data to make effective decisions, you need a monitoring platform that has all the integrations you need to capture and correlate data, with a single source of truth for observability.

RECOMMENDED STRATEGY

Either opt into a monitoring vendor that is highly integrated into the cloud-native ecosystem, or leverage open source options to help incorporate these into your homegrown system (e.g. Prometheus Exporters).

Consideration #2: <u>Discovery and Ingest</u>

Since containers can be extremely short-lived (seconds/minutes), it's paramount to discover new containers as they are spun up and down, and associate them with the correct host (EC2, etc.)–and microservice for faster MTTR. Containers are lightweight, and as such it doesn't make sense to run heavyweight agents on them that are used by traditional monitoring vendors.

RECOMMENDED STRATEGY

Choose a vendor or open source tool that employs a lightweight agent to auto-discover new containers and services. Select a solution based on open standards rather than one that makes use of its heavyweight and/or proprietary agents. This alleviates the concern of vendor lock-in and reaps the benefits of being ecosystem friendly.

Consideration #3: <u>Scale</u>

The proliferation of microservices and containers has created an explosion of data points to analyze. Current trends will continue to increase the number of data points: Bin-packing is a common strategy that increases the number of containers per host which means more data per host. Teams are doing this to help with cost savings on cloud compute. Also, right-sizing of compute instances means there are many more instances than the traditional large hosts.

Granular observability use cases (such as monitoring/alerting on a per customer or category basis) creates performance degradation of traditional monitoring tools. High data volume prevents important use cases (e.g., Searching across 1,000 containers over the course of a year translates to 31 billion data points), capacity planning, and even slows root-cause analysis during outages.

RECOMMENDED STRATEGY

Building your own solution to handle scale presents serious tradeoff considerations, and ultimately might not be worth the headache. Remember, as you continue to scale your containers in the future, data volumes will continue to grow rapidly. A home-grown system must not only handle your data scale today, but continue to perform well in the future. It will require serious investments, care and feeding. You'll not only have to scale data storage so it doesn't need to be fragmented among multiple siloed databases, but queries must continue to respond quickly at higher data volumes. Finally, data burstiness means peak loads are much higher than normal — and provisioning for peak capacity wastes a lot of money during off-peak hours.

Open source options are available. Prometheus is often the TSDB (Time Series Database) of choice when leveraging K8s and containers for a lot of monitoring teams. However, at some point you will feel the pain of the issues discussed above. It may be a better use of your time, developers, and money to focus on your core business and simply leverage a managed monitoring solution that is built with containers in mind.

Consideration #4: Latency

Traditional batch monitoring systems that run the full query and load the full dataset every time an alert is evaluated are not adequate for modern needs. These systems are unable to refresh queries more often than once every few minutes, which means end users can't track applications with fine granularity. They'll also miss important alerts and analysis on containers due to their ephemeral (short-lived) nature.

RECOMMENDED STRATEGY

Streaming analytics technology, when properly implemented, is superior to a batch approach when it comes to monitoring. Not only can streaming analytics support bigger data volumes and concurrent users/queries, but it can also trigger updates and alerts in real-time. We recommend that you build or buy a monitoring solution that streams data so you can alert on and analyze container data within seconds. This delivers the ability to act on problems faster and even trigger alerts via webhooks to perform automated rollbacks and other operations to remediate issues at machine time.

Consideration #5: Churn

Churn is the most underestimated, and potentially the most painful problem in monitoring. From a technical standpoint, churn occurs when a source of data is replaced by another equivalent one. This churning of containers creates an explosion of metadata because the identities of the data sources keep changing. Some use cases that cause this churn are CI/CD deployments (some organizations blue-green their entire environment), auto-scaling, short-lived spot instance in the cloud, and large scale tagging that affects many metrics (e.g., adding a new tag like 'customer_id' to all your metrics). Not only does churn cause accumulation of metadata over time, but sometimes it also happens in bursts (e.g., while blue-greening an entire application or service). Many monitoring systems are unable to handle this sudden volume of metadata and cannot process or index it quickly enough.

RECOMMENDED STRATEGY

Churn can be dangerous for your monitoring system, and the effects can become progressively worse over time. If you're pushing code daily, you'll have to handle 365x more metadata after a year, and it's much more challenging with containers.

If you've already built or plan to build a system, here are some of the things you will want to consider are your plans for provisioning capacity for handling bursts or slow down your code pushes to smoothen bursts if historically querying of your data is an important use case for you, you'll probably end up implementing some sort of 'pre-aggregation' or 'pre-compute' scheme on a per-service or app basis to deal with this. However, that will introduce the trade-off between timeliness and accuracy of your alerts. Most open source tools won't let you have both.

Churn issues are also widely present in today's monitoring vendors. They might work great in a small POC (Proof of Concept), but performance will degrade with time and scale of your environments. If you plan to go the vendor route, make sure your chosen vendor has built its system with the problem of churn in mind.

Consideration #6: Visibility

Finding the root cause of an issue is difficult in today's distributed, containerized environments. Having a solution that can help you visualize your data across your stack is important in teasing the signal from the noise when incidents pop up.

RECOMMENDED STRATEGY

Leveraging Grafana for general visualization or SysDig to get deep into the visualization of your containers (down to the kernel level) are good choices. But they're only one piece to the observability puzzle. Be careful you don't stack up too many point tools. Instead, consider consolidating your tooling as much as possible for cost savings. You may want to choose a platform rather than cobbled together point tools that can cover the breadth of your full-stack, preferably with pre-built dashboards for fast time to value.

Consideration #7: Troubleshooting

It's important to correlate your infrastructure to the applications that infrastructure is servicing. As some companies adopt containers, they're doing so with the mindset to break up their monolith app into microservices or start anew with a distributed services architecture for their greenfield app. This makes it critical to also be able to capture the transaction data as it traverses these services your containers support.

RECOMMENDED STRATEGY

Build or buy a tail-based distributed tracing solution that observes all of the transactions that traverse your application layer and store the p90 and p99 traces for historical analysis. Also, make sure your solution will help your team troubleshoot issues in a guided fashion. The distributed nature of today's systems create interdependencies between your services that are critical to observe and require the ability to tease out where the bulk of the latency is in the communication between your services. You also need to quickly answer whether the underlying infrastructure is contributing to increased latency or errors, so you need a solution that ties your infrastructure metrics and application traces together.

SignalFx as the Core of Your Observability Strategy for Containers

Founded the same year Docker launched, SignalFx engineers built best-of-breed Infrastructure Monitoring and Microservices APM products with containers in mind. As you can see below, our technology is uniquely positioned to meet the new monitoring criteria of the cloud-native stack, and help you take centralized control over your observability strategy.

Considerations	SignalFx	Open Source	Datadog
Integrations	Hundreds of integrations into the most popular cloud-native technologies (including Prometheus)	Most open source tools lack pre-built integrations, except for Prometheus, which has some	Hundreds of integrations into the most popular cloud-native technologies
Discovery and Ingest	Smart Agent auto-discovery and configuration for Docker, K8s, and other popular cloud-native technologies.	Most open source tools lack auto-discovery and auto-configuration capabilities. However, Prometheus does provide some limited auto-discovery capabilities.	Limited agent auto-discovery and configuration - only for containers and K8s
Scale	Streaming architecture handles 100,000s of components seamlessly	Open source TSDBs are typically not built for horizontal scalability and see performance degradation around 5k components	Batch architecture limits scalability to < 10k components
Latency	Event-driven architecture enables alerts to fire and charts to update in seconds when new containers are spun up or down	All open source tools have pull-based (batch) architectures that poll data every few minutes to update charts and fire alerts missing critical container data sets	Batch architecture also limits alerts to fire and charts to update in minutes, and historical querying takes on the order of minutes to hours. Limiting key use case
Churn	SignalFx has a purpose-built metadata store(separate from the TSDB) built to handle high churn and large metadata volumes. This allows SignalFx to support high-cardinality tagging and analytics	Open source TSDBs only have one database for time series and metadata and often limit the number of tags you can apply to metrics (< 1,000)	Discourages the use of more than 1,000 tags on metrics due to degradation of alerting and charting performance

CONTINUED ▼

Considerations	SignalFx	Open Source	Datadog
Visibility	• Full-stack observability (metrics, logs, traces). Native support for metrics, distributed traces, and deep-linking functionality into logs • Mirrored dashboard functionality cuts down on dashboard sprawl and promotes best practice content across teams and organizations	Limited visibility into metrics only hinders full observability	• Full-stack visibility across metrics and logs • Limited visibility into traces with random sampling hinders full observability
Troubleshooting	• Outlier Analyzer™ uses data science to quickly analyze related traces to identify contributing factors to high latency. This significantly lowers MTTR • Highly integrated metrics and traces platform enables rapid problem isolation. Metrics are created for all transac-tions/traces as well as segments/spans. These metrics present historical aggregates for each span and trace-path, and include RED metrics (request rate, error rate, and duration) as well as percentiles	• Open source tracing tools such as Zipkin and Jaeger lack user-friendly powerful UIs. They often force trace by trace analysis which is time-consuming and drives down MTTR • Random sampling of transactions (to reduce data volumes to manageable levels) misses key outlier and anomalous traces • Siloed tools with APM only. Users are not able to correlate transaction performance with infra-structure health	• Similar to OSS tools, Datadog performs random sampling and requires trace by trace analysis which drives down MTTR • Random sampling misses key outlier and anomalous traces • Captures metrics for a sampling of traces–thus hindering the troubleshoot-ing process

Summary

While new cloud-native technologies like containers are allowing faster innovation with more nimble and resilient application development, these new technologies are also increasing the level of complexity for monitoring and troubleshooting use cases. It's critical to implement the correct monitoring, troubleshooting and overall observability strategy as you adopt containers.

SignalFx is uniquely positioned to help you with a platform that was built for containers and the broader modern, cloud-native stack. If you'd like to learn more about the SignalFx architecture download this **white-paper**, and **start monitoring your containers today with a 14 day free trial**.

About SignalFx

SignalFx, the only real-time cloud monitoring platform for infrastructure, microservices, and applications, collects and analyzes metrics and traces across every component in your cloud environment. Built on a massively scalable streaming architecture, SignalFx applies advanced analytics and data-science-directed troubleshooting to let operators find the root cause of issues in seconds. SignalFx is trusted by leading enterprises across most every industry sector.