WHITE PAPER

The Five Principles of Monitoring Microservices

1 August 2016

sysdig

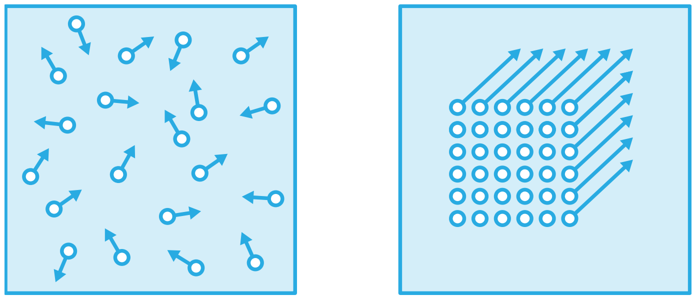
The need for microservices can be summed up in just one word: **speed.**

The need to deliver more functionality and reliability faster has revolutionized the way developers create software. Not surprisingly, this change has caused ripple effects within software management, including monitoring systems. In this post we'll focus on the radical changes required to efficiently monitor your microservices in production. We'll lay out five guiding principles for adapting your monitoring approach for this new software architecture.

On Human Civilization, Trigger Events, and the Rise of Microservices

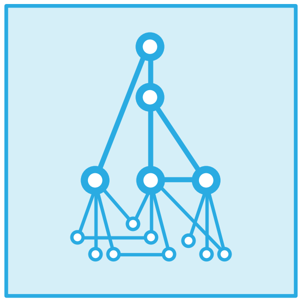
Here's a surprising fact: the rise of Microservices replicates the evolution of human societies. Societies, made of individual units called humans, have adopted various organizational structures over time, which impact how the people involved in them operate. Humans grown from independent units with their own agenda into coordinated teams that have a goal or purpose. Each of those humans is incredibly complex at an atomic scale, but as you scale up from those atoms the action of the team becomes more coherent. Does this all sound familiar?

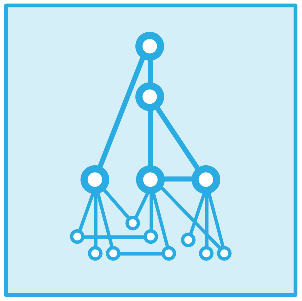
It should. Microservices has a similar set of characteristics. The basic unit is a container, and within each container is an arbitrarily complex and unique set of actions called your code. But these containers are then coordinated into a coherent purpose--a service--designed to enable an even larger society called your application.



As civilizations grew more complex, a simple “command and control” hierarchical model didn’t deliver the speed or flexibility to improve the lives of the whole. This caused change. Everything from the globalized economy and transportation systems, to the instabilities of dictatorships, communism and corporate hierarchies demonstrates change in how the global “system” of humanity works and adapts.

Again, it’s similar to software development. As Adrian Cockcroft, former Cloud Architect at Netflix, points out speed became the primary means to deliver new functionality within an increasingly complex software system: “...everything basically is subservient to the need to be able to make decisions, and build things, faster than anyone else...”.


Hierarchy


Hybrid

To increase speed within a software system you can do things such as 1) allow different parts of the whole to be re-architected without debilitating the overall application, and 2) allow different services to have related but distinctly different roadmaps. So as a result of the desire to move faster in, more complex software architectural models arose, such as microservices.

As societies and your applications evolve, so do management systems. To put it bluntly, it wouldn't be too productive to use the same management structure that the military uses for ground campaigns on, say, a group of developers building the next-generation ecommerce system.

The same is true for the management systems delivering microservices; whether it's a CI/CD system, an orchestration system, or a monitoring system, they must all adapt to the new world of microservices to unlock the power of this new architectural model.

Monitoring is a critical piece of the control systems of microservices, as the more complex your software gets, the harder it is to understand its performance and troubleshoot problems. Given the dramatic changes to software delivery, however, monitoring needs an overhaul to perform well in a microservice environment. The rest of this article presents the five principles of monitoring microservices, as follows:

- **Monitor containers and what's inside them**
- **Alert on service performance, not container performance**
- **Monitor services that are elastic and multi-location**
- **Monitor APIs**
- **Map your monitoring to your organizational structure**

Leveraging these five principles will allow you to establish more effective monitoring as you make your way towards microservices. These principles will allow you to address both the technological changes associated with microservices, in addition to the organizational changes related to them.

Case Study: Modern services at scale

A provider of real-time customer engagement tools wanted to achieve an order of magnitude improvement in the flexibility and scalability of their software architecture.

They made the shift over to Kubernetes-based Docker deployments, leveraging Jenkins as the CI/CD tool. They now run Kubernetes in multiple regions on about 65 large VMs and increasing to over 100 in the next couple months. They currently process 800 million requests per day with the plan to process over 2 billion requests per day in the coming months.

After 3 months into the transition, they were deploying software 10 times per week. After 4 months, 40 per week. Their goal in the coming months is to achieve 100 deployments per week based on this new infrastructure.

The Principles of Microservice Monitoring

1. Monitor Containers & What's Running Inside Them

Containers gained prominence as the building blocks of microservices. The speed, portability, and isolation of containers made it easy for developers to embrace a micro-service model. There's been a lot written on the benefits of containers, so we won't recount it all here.

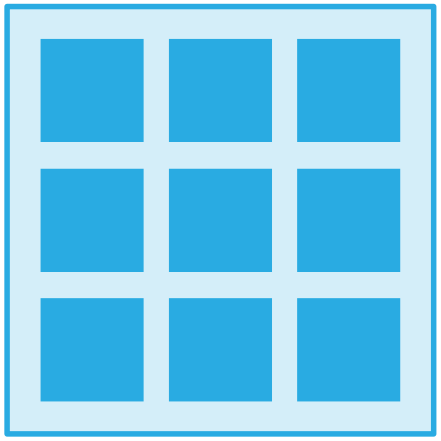
Containers are black boxes to most systems that live around them. That's incredibly useful for development, enabling a high level of portability from Dev through Prod, from developer laptop to cloud. But when it comes to operating, monitoring, and troubleshooting a service, black boxes make common activities harder, leading us to wonder: what's running in the container? How is the application / code performing? Is it spitting out important custom metrics? From the devops perspective, you need deep visibility inside containers rather than just knowing that some containers exist.

The typical process for instrumentation in a non-containerized environment--an agent that lives in the user space of a host or VM--doesn't doesn't work particularly well for containers. That's because containers benefit from being small, isolated processes with as few dependencies as possible. And, at scale, running thousands of monitoring agents for even a modestly-sized deployment is an expensive use of resources.

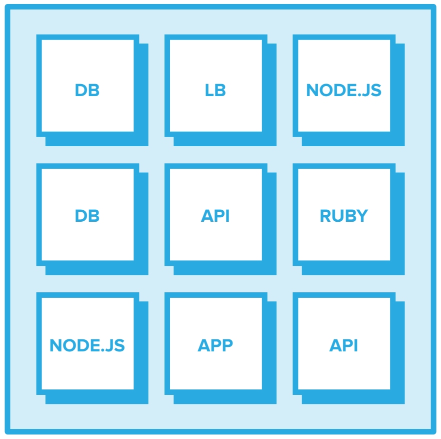
Two potential solutions arise for containers:

1) Ask your developers to instrument their code directly
2) Leverage kernel-level instrumentation to see all application and container activity on your hosts

We won't go into depth here, but each method has pros and cons.



Great for operations



Great for development

The Principles of Microservice Monitoring

2. Leverage Orchestration Systems to Alert on Service Performance

CMaking sense of operational data in a containerized environment is a new challenge. The metrics of a single container have a much lower marginal value than the aggregate information from all the containers that make up a function or a service.

This particularly applies to application-level information like which queries have the slowest response times or which URLs are seeing the most errors, but also applies to infrastructure-level monitoring, like which services' containers are using the most resources beyond their allocated CPU shares.

Increasingly, software deployment requires an orchestration system to "translate" a logical application blueprint into physical containers. Teams use an orchestration system to (1) define your microservices and (2) understand the current state of each service in deployment. You could argue that the orchestration system is even more important than the containers. The actual containers are ephemeral--they matter only for the short time that they exist--while your services matter for the life of their usefulness.

Case Study: Dynamic service monitoring

A provider of drone-based services built a complex backend consisting of about 30 microservices. Fifteen of these are proprietary while the others are integrated third party services, scaling on-demand.

For the operations team, the challenge was how they could allow their monitoring services to dynamically adapt to changes in the application environment.

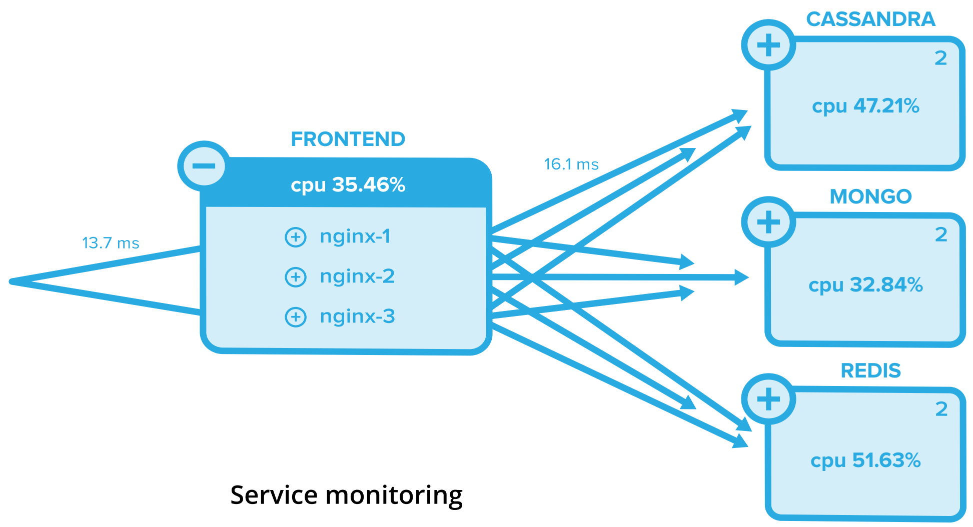
Their strategy leveraged two main philosophies:

- Automatically instrument containers without human intervention or per-container agents
- Combine information from their orchestration system to aggregate container data into service data.

"This approach allows us to first monitor services and their performance. Then, only if we need to, we can drop down into infrastructure metrics in order to troubleshoot a problem."

Devops teams should redefine alerts to focus on characteristics that get as close to monitoring the experience of the service as possible. These alerts are the first line of defense in assessing if something is impacting the application. But getting to these alerts is challenging, if not impossible, unless your monitoring system is container-native.

Container-native solutions leverage orchestration metadata to dynamically aggregate container and application data to calculate monitoring metrics on a per-service basis. Depending on your orchestration tool, you might have different layers of a hierarchy that you'd like to drill into. For example, in Kubernetes, you typically have a Namespace, ReplicaSets, Pods, and some containers. Aggregating at these various layers is essential for logical troubleshooting, regardless of the physical deployment of the containers that make up the service.



Service monitoring

The Principles of Microservice Monitoring

3. Be Prepared for Services that are Elastic and Multi-Location

Elastic services are certainly not a new concept, but the velocity of change is much faster in container-native environments than virtualized environments. Rapidly changing environments can wreak havoc on brittle monitoring systems.

Frequently monitoring legacy systems required manual tuning of metrics and checks based on individual deployments of software. This tuning can be as specific as defining the individual metrics to be captured, or configuring collection based on what application is operating in a particular container. While that may be acceptable on a small scale (think tens of containers), it would be unbearable in anything larger. Microservice focused monitoring must be able to comfortably grow and shrink in step with elastic services, without human intervention.

For example, if the DevOps team must manually define what service a container is included in for monitoring purposes, they no doubt drop the ball as Kubernetes or Mesos spins up new containers regularly throughout the day. Similarly, if ops was required to install a custom stats endpoint when new code is built and pushed into production, challenges may arise as developers pull base images from a Docker registry.

In production, build monitoring toward a sophisticated deployment that spans multiple data centers or multiple clouds. Leveraging, for example, AWS CloudWatch will only get you so far if your services span your private data center as well as AWS. That leads back to implementing a monitoring system that can span these different locations as well as operate in dynamic, container-native environments.

The Principles of Microservice Monitoring

4. Monitor APIs

In microservice environments APIs are the lingua franca. They are essentially the only elements of a service that are exposed to other teams. In fact, response and consistency of the API may be the “internal SLA” even if there isn’t a formal SLA defined.

As a result API monitoring is essential. API monitoring can take many forms, but clearly must go beyond binary up/down checks. For instance, it’s valuable to understand the most frequently used endpoints as a function of time. This allows teams to see if anything noticable has changed in the usage of services, whether it be due to a design change or a user change.

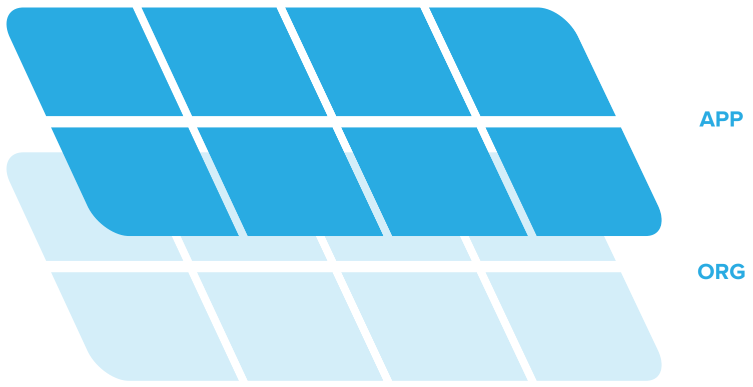
You can also consider the slowest endpoints of your service, as these can reveal significant problems, or, at the very least, point to areas that need the most optimization in your system.

Finally, the ability to trace service calls through your system represents another critical capability. While typically used by developers, this type of profiling will help you understand the overall user experience while breaking information down into infrastructure and application-based views of your environment.

5. Map Monitoring to Your Organizational Structure

While most of this post has been focused on the technological shift in microservices and monitoring, like any technology story this is as much about people as it is about software bits.

For those of you familiar with Conway's law, he reminds us that the design of systems are defined by the organizational structure of the teams building them. The allure of creating faster, more agile software has pushed teams to think about restructuring their development organization and the rules that govern it.



Map monitoring

The organization must therefore mirror microservices themselves. That means smaller teams, loosely coupled, that can choose their own direction as long as it still meets the needs of the whole. Within each team there is more control than ever over languages used, how bugs are handled, or even operational responsibilities.

Not surprisingly, the monitoring approach must also embrace this change. Development teams responsible for creating and delivering micro-services should be able to isolate the monitoring data, views, and alerts associated with their service. They need the freedom to define how their team looks at that data both for general monitoring purposes as well as functional improvements to the application and usage patterns.

DevOps teams can enable a monitoring platform that does exactly this: allows each micro-service team to isolate their alerts, metrics, and dashboards, while still giving operations a view into the global system.

Case Study: Building an internal PaaS

One of the largest media companies in the world was creating a new platform as a service (PaaS) to better assist developers in getting their software to market faster.

“For the developer, a container-based PaaS is simpler and faster to get going with,” noted the devops lead. “And for the ops team, we have a slightly different agenda: this approach makes it easier for us to provide consistent monitoring and troubleshooting capabilities.”

The devops lead continued, “With our new approach, we can now give each development team isolated views into the operations of their own software. At the same time the devops team can see it all to ensure overall platform performance.”

sysdig

Conclusion

There's one, clear trigger event that precipitated the move to microservices: speed.

Organizations wanted to deliver more capabilities to their customers in less time. Once this happened, technology stepped in: the architectural move to micro-services and the underlying shift to containers make speed happen. Anything that gets in the way of this progress train is going to get run over on the tracks.

As a result, the fundamental principles of monitoring need to adapt to the underlying technology and organizational changes that accompany microservices. Operations teams that recognize this shift can adapt to micro-services earlier and easier.