WHAT'S DOCKER MONITORING? HOW TO MONITOR CONTAINERS & MICROSERVICES

Docker Monitoring is the activity of monitoring the performance of microservice containers in Docker environments. Monitoring is the first step towards optimizing and improving performance.

(*This is part of our Docker Guide. Use the right-hand menu to navigate.)*

Setting the stage for Docker Monitoring

Not long ago, most software application systems ran on bare-metal infrastructure hosted in data centers. The hardware—consisting mainly of compute, storage, and networking components—was largely fixed in those environments, and, so was the monitoring infrastructure. It required few updates later once a comprehensive solution was rolled out because changes in production, both hardware and application related, that have impact on monitoring configuration were infrequent.

The virtualization of compute resources didn't change that scenario much. Though it provided lots of flexibility in provisioning non-production environments, the changes that impact monitoring tend to be few, except in situations where an application component running in a cluster is auto-scaled. Such dynamic configurations made sense only when implementing elasticity resulted in cost savings, and, for that, environments have to be on public cloud platforms like AWS where charges are usage-driven.

As the virtual machine (VM) retained the concept of a machine that runs an operating system, the tools and methods used for bare-metal infrastructure could still be useful for VM based environments with occasional tweaks. However, the use of containers to build application

environments has a disruptive impact on traditional monitoring methods because containers don't fit well with the assumptions made by traditional tools and methods that were originally designed for bare-metal machines.

The containers, of which Docker is a popular implementation, are normally brought up and down on demand. They are ephemeral as they are lightweight and can be started up with little system overhead so they could be discarded when not actively in use.

The Dockerization also forced the applications to be redesigned to work as distributed systems with each functional element is run in one more containers. That enabled a container based system to be scaled easily and the available compute resources could be allocated much more efficiently. As a result of inherent architectural change that containerization brought in, the production environments built using containers become highly dynamic and monitoring of such environments became much more important than it used to be before.

Common challenges

The dynamic nature of container-based application infrastructure brings new problems to monitoring tools. Docker also adds another layer of infrastructure and network monitoring requirements to the overall scope.

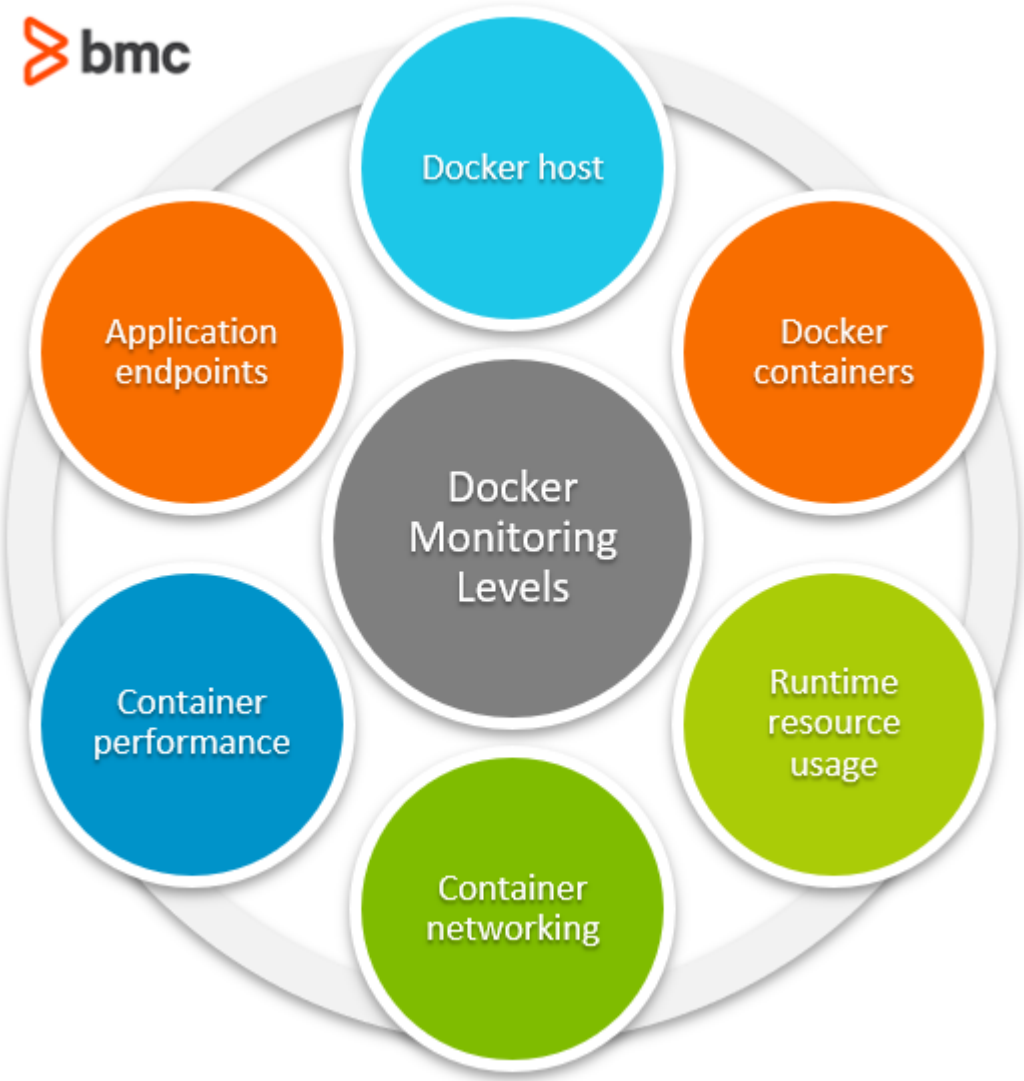
Think of the typical scenario of multiple VMs provisioned on a bare-metal machine and containers come and go on each one of those VMs. The monitoring requirements include checking:

- The health of bare-metal host
- The VMs provisioned on it
- The containers active at a given point of time

Of course, how well these components interact with each other and to the outer world should also be checked from the networking side of monitoring requirements. It can soon become very complex.

Levels of Docker monitoring

As mentioned earlier, the container is an extra layer for infrastructure monitoring purposes. For addressing monitoring requirements of a container based application environment systematically, the monitoring should be implemented at various levels of the infrastructure and application.



Docker host

Docker containers are run on a cluster of large bare-metal or virtual machines. Monitoring of these machines for their availability and performance is important. This falls into the traditional infrastructure monitoring.

Typically, CPU, memory and storage usages are tracked and alerted based on the thresholds setup for those metrics. Implementing those are relatively easy: any monitoring tool would support it as part of core features.

Docker containers

The Docker containers are run on a cluster of hosts and a specific Docker instance could be running on any one of those hosts depending on the scheduling and scaling strategies set in the container orchestration system used, such as:

- Docker Swarm
- Kubernetes
- Apache Mesos
- Hashicorp Nomad

(Read our comparison of Docker Swarm and Kubernetes.)

Ideally, there is no need to track where the containers are running. But, things are rarely ideal in production (and that's why you need monitoring in the first place) and you may want to look at a specific container instance. Tracking information on the up and running containers would be handy in such situations and also to make sure that scheduling and scaling rules are actually enforced.

Runtime resource usage

As with bare-metal and virtual machines, CPU, memory and storage metrics are tracked for Docker containers as well. Container specific metrics related to CPU throttling, a situation when CPU cycles are allocated based on priorities set when there would be competition for available CPU, can also be tracked.

Tracking of these system performance metrics would help to determine whether resources on bare-metal and virtual machines, the container hosting infra, need to be upgraded. It would also provide insights to finetune the resources allocated to a Docker image so its future container instances will be started up with adequate runtime resources.

The native Docker command "docker stats" returns some of these metrics but a tool like TrueSight is needed to capture these statistics system wide, for getting notified on potential issues and resolving those proactively.

Container networking

Checking on container level networks is one of the most important aspect of Docker monitoring. A container is supposed to run a lightweight component of a distributed application system. Communication between these components has to be reliable and predictable, especially when there is high dynamicity to a container-based environment in which the instances come and go.

Docker provides a container level network layer and also there are third-party tools to expose the services running on containers. Other components in the system can access a specific service using a supported method like REST API.

In a highly distributed environment, Docker networking configuration would soon become complex and it is important to monitor various network access requirements and proxy settings for the whole system to work.

Container performance

Just like it happens on a bare-metal or virtual machine, the runtime requirements would impact the overall performance of container and in turn the service running on it. Gathering performance data from containers is important to fine tune those.

Application endpoints

A container-based environment would be running a large, highly distributed application with each service running on one or more containers. The application checks could be done both at three levels:

- Container level
- Pod level (A pod is a group of containers that offers a service.)

- System-wide level

Usually REST API endpoints would be available to perform such checks that could easily be plugged into any modern monitoring system to check the availability of related services.

Benefits of Docker monitoring

The benefits of Docker monitoring are not different from traditional monitoring. These are the main points:

- Monitoring helps to identify issues proactively that would help to avoid system outages.
- The monitoring time-series data provide insights to fine-tune applications for better performance and robustness.
- With full monitoring in place, changes could be rolled out safely as issues will be caught early on and be resolved quickly before that transforms into root-cause for an outage.
- The changes are inherent in container based environments and impact of that too gets monitored indirectly.

The last point makes it clear that monitoring is an essential part of Docker based environments due to their dynamicity and availability of application services has to be checked constantly.

Getting started with Docker Monitoring

A group of tools are needed for fully monitoring an application system running in production. Typically, they will cover:

- Infrastructure
- Network
- Application features and performance
- Last-mile monitoring and log analytics (Last-mile monitoring refers to checking on user experience.)

The requirement of Docker monitoring dictates that the monitoring tools selected should cover container level monitoring also. Or add extra tools, like those discussed already, to take care of that aspect.

Tracking ephemeral containers

The traditional monitoring is host0centric. The tools from that category assume that an application environment is made of devices with a unique IP address assigned to each one of them. That approach always poses problems beyond simple infrastructure monitoring because requirements such as checking an application feature is not just tied to one or more hosts.

The containers come and go and it would be better if those are not tracked individually. The best method is to tag the containers with keywords. That way time series data from same type of containers could be looked up for monitoring and operational insights, irrespective of their lifecycle status.

Containers add complexity

Usage of containers adds to the operational complexity of an application system and so the monitoring requirements. Most of the popular monitoring tools are not equipped to monitor Docker containers though it is not hard to extend them to support containers. New generation monitoring tools, both open source and licensed, support Docker monitoring out of the box.

The challenges in rolling out a good Docker monitoring system remain the same as with any generic monitoring systems:

- Selecting a set of tools that are most suitable from a wide range of product offerings
- Identifying the most important monitoring requirements
- Mapping those to the features of selected tools
- Making customizations to fill any gaps that are not covered by the tools, especially in the area of application monitoring
- Setting up an alerting and response strategy that will not overwhelm the operations staff

Related reading

- BMC DevOps Blog
- Docker Commands: A Cheat Sheet
- Docker vs Kubernetes: What's The Difference?
- How To Run MongoDB as a Docker Container
- The State of Containers Today: A Report Roundup
- Managing Containers & Code for DevOps