Brought to you by:

Sysdig

Running Containers in Production

for dummies

A Wiley Brand

Understand containers and orchestrators

Monitor and secure containers in production

Set up a CI/CD/CS delivery pipeline

Jorge Salamero Sanz
Eric Carter
Knox Anderson

Sysdig
Special Edition

Running Containers in Production

Sysdig Special Edition

by Jorge Salamero Sanz, Eric Carter, and Knox Anderson

for dummies®
A Wiley Brand

Running Containers in Production For Dummies®, Sysdig Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN 978-1-119-52110-5 (pbk); ISBN 978-1-119-52105-1 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

We're proud of this book and of the people who worked on it. Some of the people who helped bring this book to market include the following:

Contributing Writer: Emily Freeman

Development Editor: Scott Lowe

Project Editor: Martin V. Minner

Executive Editor: Steve Hayes

Editorial Manager: Rev Mengle

Business Development Representative: Karen Hattan

Production Editor: Mohammed Zafar Ali

Table of Contents

Introduction

Many people consider containers a hot new technology that they need to jump on. In reality, however, containers aren't new at all. They're actually pretty old and have their origins dating back as far as the introduction of the chroot system call in 1979.

Over the years, there have been other attempts to implement container-like constructs. Docker finally hit the mark in 2013 with the introduction of a standardized container format. Since then, containers have enjoyed widespread adoption because of the robust Docker ecosystem and the popularity of Kubernetes.

Although many have embraced containers, others have struggled with transitioning mature codebases and enterprise products into the new ecosystem. Many organizations that have moved them into production have done so haphazardly and in ways that don't enable best practices-based security.

About This Book

Running Containers in Production For Dummies, Sysdig Special Edition, explores the benefits of containers and the challenges of deploying containerized applications in

production. It provides an overview of the container ecosystem as well as the development processes and architectures that complement it. This book covers

- **Building and managing container environments**
- **Creating a CI/CD/CS pipeline**
- **Monitoring containers**
- **Securing containers**

Icons Used in This Book

This book uses the following icons to call your attention to can't-miss information.


REMEMBER

Paragraphs marked with the Remember icon are particularly important for you to keep in mind.


TIP

Don't miss the information marked with the Tip icon — it can make your life easier.


TECHNICAL STUFF

We use this icon to introduce something that's particularly technical in nature.

IN THIS CHAPTER

- Moving to microservices
- Comparing orchestration platforms

Chapter 1
Understanding Containers and Orchestration Platforms

Containers may be all the rage today, but they aren't a new development. They've actually been around since the late 1970s. It wasn't until Docker debuted its container platform in 2013 that users found the

technology mature enough to run applications for production workloads.

Docker did something that hadn't been done before. The company's approach enabled a containerized application to run across different operating systems by packaging dependencies — the software and libraries necessary to operate a workload — to give developers a way to code applications and easily move them from a development laptop into a test environment and, finally, into production.

WHAT DO CONTAINERS DO?

Containers are a type of operating system virtualization that isolates resources — CPU, memory, disk, or network — while allowing isolated workloads to run on the same host. They hold software binaries and libraries — everything required to run an application. You can think of containers as lightweight virtual machines without the overhead of a full operating system getting in your way, meaning that they can be far more agile and enable far more workload density than traditional approaches to virtualization.

Moving to Microservices

Before containers came on the scene, most enterprise applications were unmoving monoliths that were being crushed under their own weight. These applications consisted of one massive code base that contained all of the functionality required to make the application do the company's bidding.

REMEMBER

The monolithic approach had some downsides, most notably around how to grow a workload environment as business needs demanded. Sure, you could scale your hardware resources in a never-ending fashion, but easily scaling the application was sometimes a tall order.

Enter microservices.

In a microservices world, applications are no longer maniacal, monolithic monsters devouring hardware resources. Instead, applications are smashed apart and decomposed into loosely coupled, independently deployable application services that talk to each other, typically using standard protocols such as HTTP or GRPC. Instead of one massive application to deal with, you have tons of little, tiny ones that are designed to work in harmony and meet complex business needs.

No longer are you at the mercy of the monolith. Your shiny new microservices model enables a "scale-out" architecture where additional process instances can be started to keep pace with load — all of which interact with each other over the network.

But, what does this microservices approach have to do with containers, anyway? Containers happen to be ideally suited to these loosely coupled services because containers simplify scaling and ongoing deployments.

Comparing Orchestration and Management Tools

Automating the operation of hundreds, if not thousands, of containers in a cluster across multiple hosts requires an *orchestration tool*. As the name suggests, orchestration tools assume the role of an orchestra conductor by managing and coordinating all of the services that comprise the environment. This means managing how hosts, containers, and services are created, started, stopped, upgraded, connected, and made available.

Many orchestration solutions are available, with options ranging from homegrown systems, to open source packages, to commercial products. This section introduces five popular container orchestration and management tools.

Kubernetes

Originally created by Google as a mechanism for deploying, maintaining, and scaling applications, Kubernetes — K8s or Kube, for short — was donated to the Cloud Native Computing Foundation in 2015 and is now available as an open source project.

REMEMBER

Kubernetes simplifies the orchestration of containers across multiple hosts by managing the scale and health of nodes. It organizes containers into groups referred to as *pods* to streamline workload scheduling. Auto-placement, auto-restart, and auto-replication enable your applications to self-heal to ensure uptime for the services they control. Kubernetes ably performs *rolling upgrades* of an entire cluster without application or service downtime.

Best of all, as a popular open source project, Kubernetes shows no sign of slowing down. With a large community of project contributors, Kubernetes sits at the very heart of a fast-growing ecosystem.

OpenShift

OpenShift is not a stand-alone orchestration platform. Rather, OpenShift is Red Hat's enterprise container application platform. It's built around Kubernetes and extends that platform's capabilities. OpenShift inherits

all the upstream capabilities of Kubernetes but also enhances the enterprise user experience by adding features to enable rapid application development, easy deployment, and lifecycle maintenance.

If your team is looking for a fully featured orchestration platform for developers that includes additional enterprise-specific features such as support, OpenShift is a good choice.

Docker Swarm/Docker Enterprise

Docker Swarm is Docker's native host clustering and container scheduling tool. Docker includes Swarm with Docker Engine and, unlike other solutions, doesn't require additional components to operate.

The initial setup is straightforward. Given its simplicity, Docker Swarm works great in small environments but has also been successfully scaled for large environments with up to 30,000 containers. With Docker Enterprise Edition, you can deploy apps using both Docker Swarm and Kubernetes within the same platform.

Amazon ECS, Fargate, EKS

Amazon Web Services (AWS) provides several services that orchestrate containerized applications. Amazon Elastic Container Service (ECS) allows you to run and scale Docker-based applications on AWS EC2 instances.

ECS also supports *Fargate*, which allows you to run containers without having to manage the servers that comprise the cluster. With ECS, a built-in scheduler can be used to trigger container deployment based on resource availability and demand.


TIP

Amazon ECS is ideal for small teams that rely on AWS to manage their infrastructure and require tight integration with other AWS services.

Amazon also offers a Kubernetes-specific service, Elastic Container Service for Kubernetes (EKS), which lets you run Kubernetes on AWS without needing to install and manage clusters. This solution is best for those who want to use the Kubernetes features but prefer a managed service.

Apache Mesos, Mesosphere DC/OS, and Marathon

Apache Mesos is both a cluster manager tool and a host operating system. Mesos has its own container format but also supports Docker containers.

Mesosphere developed its DC/OS (Datacenter Operating System) on top of Mesos as an open source project with commercial offering for enterprises that need additional capabilities and support. DC/OS provides a ready to use platform with a management interface, scheduling,

network, and more. With Mesos alone, you need to set up components individually.

Marathon is a framework for container and services orchestration typically used with Mesos and DC/OS. Kubernetes can also be used for orchestration on top of DC/OS.

TIP

Mesos and DC/OS are typically used in large-scale clusters with nodes counts exceeding 10,000, with the clusters running hundreds of thousands of containers. This is a great option in organizations that need massive distributed system scale and use big data applications such as Hadoop, Spark, or Cassandra.

IN THIS CHAPTER

- Building containerized applications
- Shipping container images
- Implementing CI/CD/CS

Chapter 2
Building and Deploying Containers

Every DevOps organization strives to have a supply chain that can consistently develop, package, and get applications into production faster. The technology to accomplish these goals has not always been available. Today, however, containers make this possible.

The first step for getting your supply chain pipeline up and running is to create processes to streamline building,

packaging, shipping, deployment, and operation of your services.

Building Your Images

Your images are the core of your environment and must be treated with care. Keep these four key items in mind as you undertake the image creation process:

- **Use a trusted base image.** Building containers from a standard base image ensures that you have better knowledge of how and when your images are updated so you can keep things consistent.
- **Restrict libraries and dependencies.** Containers are often used to help decouple components of an application as a part of creating a microservices-based architecture. Doing so helps make security and operations easier by reducing the contents of an image and more strictly focusing the functionality of that image.
- **Restrict access.** Security is paramount. As a part of the image creation process, you need to develop a security approach. For example, avoid running processes as the root user and restrict access to storage volumes and other resources in containers.
- **Scan images for known vulnerabilities.** Use an image scanner to identify vulnerabilities within

your image and track components and versions included on each one.


TIP

Once you have a process in place, many good tools such as Jenkins, CircleCI, Bamboo, CodeFresh, and GoCD can automate the tasks required to build, package, and test software. Whatever your team decides, make sure everyone is comfortable with the software and make it a mandatory step for containers to pass through your build tool before being added to a registry.

Shipping Container Images

After an image is built it should be pushed into a registry such as DockerHub, Quay, Google, Amazon, Azure Container Registry, or your self-hosted registry, running on a tool such as Docker Registry, Portus, or Harbor.

Your trusted registry must require authentication so images are not available to just anyone and to ensure that only images that have been scanned and are ready to be deployed are available.


TIP

As a part of this process, you should enable the content trust feature. This provides the ability to add digital signatures to images. These signatures allow clients to verify the integrity and publisher of images.

Implementing CI/CD/CS

A complete container supply chain process typically covers integration (CI), deployment (CD), and security (CS) as a continuous, ongoing process. Here's an overview of the steps necessary for a CI/CD/CS pipeline.

Continuous integration

Continuous integration is a process by which code is automatically tested each time a change is committed to version control. Implementing continuous integration is accomplished through a series of activities, including

- **Writing automated tests for every feature:** This prevents bugs moving forward in the software supply chain process.
- **Creating a defined CI process:** A CI server monitors the code repository for changes and triggers the automated tests when new commits are pushed.
- **Maintaining a strong testing culture:** Testing should be seen as a core part of the development process, including unit tests as well as functional and packaging testing.

Continuous delivery

Continuous delivery is a concept that enables organizations to automatically build, test, and otherwise prepare

software for deployment into the production environment. Here are several important considerations to keep in mind when implementing continuous delivery:

- **Automated deployment to the Docker registry.** Images should be automatically pushed by the CI server into an image repository known as the registry.
- **Infrastructure, configuration, and security should be handled as code.** Changes to infrastructure, service configuration, and security should be part of the same change tracking and management process used with changes in code.
- **Functional testing should be in place.** Automated functional tests for all layers should be run in staging environment before production. This ensures that nothing was missed before code is pushed into production.

Continuous deployment

At first, continuous delivery and continuous deployment may appear to be the same thing, but there are some nuances. Continuous delivery has an end result that a particular update is safely *deployable*. The update isn't automatically deployed into production. That's where

continuous deployment comes in. Here are a few of its features:

- **Implementation of automated deployments:** Deployments can be fully automated, or at least a one-step process triggered by a human, in concert with integration with the orchestration platform.
- **Rollback ready:** This ensures that the organization can go back to a previous software version if issues are found after the deployment goes into production.
- **Integration of feature flags.** A *feature flag* is a toggle that enables or disables a specific feature. This allows developers to begin including code for new features, even if those new features aren't quite ready for prime time. This ensures that users are not affected by incomplete features.

Continuous security

Don't miss security as part of the pipeline. A movement known as DevSecOps advocates making security part of this process.

REMEMBER

In addition to the security best practices you should consider while building and deploying containers, closing the gap requires implementing new security layers during the production phase of the container lifecycle. Chapter 4 covers this in more detail.

IN THIS CHAPTER

- Collecting and managing metrics
- Monitoring the environment
- Troubleshooting: Going beyond monitoring

Chapter 3
Monitoring Containers

Over the past few years, the infrastructure and ecosystem evolution with microservices and containers has made many existing monitoring tools and techniques no longer relevant. Instead, developers need solutions that can adapt to the short-lived and isolated nature of containers and application services.

Understanding Container Visibility Challenges

On one hand, containers provide flexibility and portability, but on the other hand, they complicate monitoring and troubleshooting. Why is that? As isolated "black boxes," containers make it difficult for traditional tools to penetrate their shells in an effort to observe processes and performance metrics.

Here's why.

Monitoring agent shouldn't add a second service

A best practice for using containers is to isolate workloads by running only a single process per container. Placing a monitoring agent — which amounts to a second process or service — in each container to get visibility risks destroying a key value of containers: simplicity.

Scalable and dynamic infrastructure with microservices

In static environments, it was simple to get monitoring agents running on a host and pointed at relevant applications. You installed the agent and then moved on.

With containers, microservices come and go. They move around, scaling up and down as demand shifts. The dynamic nature of containers makes manual configuration to collect relevant metrics impossible. Instead, monitoring must focus a bit differently. You want to know how your service — the one that is comprised of multiple containers — behaves overall but also how each contributing container individually performs in its role.

Metric data volume is voluminous

Modern application infrastructure is increasingly focused on large-scale deployments. Host, container, network, and orchestration metrics — the elements required to understand performance and health — grow exponentially. Storing millions of data points for long-term trending and analysis requires horizontal scaling and new types of databases to support storing of metrics for analysis. Because these metrics come from different sources, the metric cardinality grows exponentially.

Collecting Metrics

Monitoring containers is not simply about visibility into container processes. To understand the whole picture, teams must monitor containers, services, and the infrastructure on which these run — and do so with minimal

impact. A number of methods exist for instrumenting and collecting data.

Container monitoring requires collection of a broad range of metrics and event data to reflect true service response times and resource utilization while also capturing overall infrastructure state and health. Several approaches are available. Understanding each will help you choose what's best for your environment:

- **Monitoring process inside a container:** As previously mentioned, this method is generally considered undesirable because of the bloat it can impose on containers. This approach adds monitoring software to containers to collect and export metrics.
- **Sidecar containers:** This approach attaches a monitoring agent container to each application container deployed — like a motorcycle sidecar. This enables a monitoring agent to run as an isolated process; however, this method comes with the downside of contributing to container sprawl.

TIP

When you use sidecar containers, you're essentially doubling the number of containers in production. This introduces significant overhead into your application environment.

- **Agent-per-pod:** This approach attaches a monitoring agent to a group of containers like Kubernetes pods — containers that share a namespace. This method is easy to set up, but resource consumption is high per-agent because of the volume of metrics flowing through it.
- **Syscalls:** This model utilizes a monitoring agent for each host that collects metrics by observing all system calls traversing the OS kernel. This method drastically reduces the monitoring agent's resource consumption and per-container instrumentation. It allows you to see what's happening inside a container from the outside. This approach also enables you to collect in-depth data for containers, short-lived processes, orchestration tools, and underlying infrastructure with little overhead.

Aggregating and Segmenting Metrics

To understand actual performance with microservices and containers, it becomes critical to view metrics broken down by logical service rather than physical infrastructure. Container-native solutions leverage orchestration metadata to aggregate container and application data on a per-service basis.

This allows you to drill into metrics at different layers of a hierarchy. For example, in Kubernetes, you have namespaces, services, deployments, pods, and containers. Segmenting metrics by these layers lets you see aggregated performance, which is essential for logical troubleshooting: a drill-down process where you identify the application with an issue; the microservice where the issue comes from; the specific pod, container, and process with the issue; and the host where it is running at that moment.

Monitoring the Environment Layers

When monitoring containers, it is important to have visibility into a wide range of environment components. Getting a complete picture depends on availability of the right data in the right format. When evaluating monitoring tools, look for availability of the following metrics and information.

Infrastructure

Infrastructure-level monitoring, from host resources to storage and networking, provides information that helps determine the root cause of certain container issues. For example, host CPU metrics are an important factor when trying to understand which containers are using the most computing resources.

Services

Service metrics provide views into the performance of each of the services, such as load balancing or web endpoints, that comprise your application. If there's an application slowdown, being able to see the relative performance of each microservice helps you pinpoint problems.

Applications

Application metrics such as the number of connections, current response time, and reported errors, are focused on the health and performance of your application as a whole from a user perspective. Having data at this level takes much of the guesswork out of understanding the user experience with your solution.

Custom metrics

Custom metrics are those that are uniquely defined within an application or by a developer for tracking specific information. Custom metrics are typically of high value, put in place to reveal important details about application behavior and events:

- **JMX:** Java Management Extensions (JMX) are used to expose run-time metrics for monitoring Java applications.

PROMETHEUS MONITORING

Prometheus is an incubating project of the Cloud Native Computing Foundation (CNCF). The CNCF fosters a community around open source technologies that orchestrate containers as part of microservices architectures. Prometheus is one of the fastest growing projects, providing real-time monitoring, alerting, and time-series database capabilities for cloud native applications. It is used to generate and collect metrics from monitored targets and integrates with many popular open source and commercial tools for data import/export. Prometheus client libraries enable developers to instrument application code. Its PromQL query language lets users select and aggregate time series data. Many of the orchestration and enterprise container application platforms, including Kubernetes, OpenShift, and Mesosphere DC/OS, have embraced the solution and export Prometheus metrics by default.

» **StatsD:** StatsD is a lightweight protocol for custom metrics. Libraries for implementing StatsD are available in most popular languages.

Despite being widely used, because of its lack of metrics labels support, StatsD's popularity is decreasing in favor of Prometheus metrics.

- **Prometheus:** Prometheus is an open source monitoring project that introduced a metric format known as Prometheus metrics. Client libraries also exist for multiple languages. This format features a generic label functionality that has multiple benefits on microservice-based applications.

Orchestration: The new layer

As the chief coordinator of containerized services, orchestration tools provide a wealth of information about a microservices environment. Tapping into this resource is critical to gaining a true understanding of your infrastructure. Tags and labels from the orchestration allow you to represent metrics by the logical organization of your architecture versus a physical orientation. This means you can take an application-centric view of your environment and gain critical insights into a number of elements, including

- **Health metrics:** Health metrics for orchestration provide performance data like CPU, memory, disk, or response time. When aggregated for logical/application entities such as namespaces, deployments, or pods in Kubernetes, these metrics give

you a better understanding of your application's behavior.

- **State metrics:** State metrics, such as those provided by Kubernetes kube-state-metrics, are about the status or count of orchestration objects. For example, state metrics can provide information about the number of containers ready versus the desired number for a given service, or let you know if a container is restarting in a loop so that you can take appropriate action.
- **Internal services:** Orchestration tools are built from multiple components and services. To guarantee health and performance of the platform, you also need to monitor them. In Kubernetes, services that require monitoring include etcd, API server, and kubelet.

Designing a Monitoring Process

Containerized applications running in production must be constantly monitored for availability, errors, and service response times. Achieving this requires collection of a wide range of telemetry and event data. Sources and approaches for collecting and presenting this information can be varied.

Metrics

By collecting and correlating container metrics with infrastructure and orchestration data, you can monitor the performance, health, and state of your containerized applications and maximize availability.

Two widespread solutions for container monitoring use different methods:

- **Sysdig Monitor:** Sysdig auto-discovers container metrics via system calls. It collects a wide range of data such as host and container performance metrics, custom metrics, state metrics, application metrics, and more. This includes integration with Prometheus to ingest metrics from instrumented apps and enable advanced queries using the Prometheus query language (PromQL). Sysdig tags all metrics with all the available metadata and tags to support exploring, aggregating, segmenting, and drilling down. All system call activity can be recorded for container troubleshooting. Sysdig Monitor backend has outstanding scalability and serves well in scenarios that support thousands of nodes.
- **Prometheus:** Prometheus lets you define and expose metrics via an HTTP endpoint through sidecar containers known as *Exporters*. Prometheus will scrape these endpoints periodically and will store the metrics in its database. Additional

software like Grafana or AlertManager is required to build a complete monitoring system. Prometheus metrics is the format used internally for Kubernetes orchestration state metrics.

Tracing

Tracing is designed to log and track transaction flows as requests propagate throughout an application. This allows administrators to observe latency for each microservice and identify bottlenecks that affect performance. Tracing is accomplished through two primary toolsets:

- **Application Performance Monitoring:** Typically used by programmers for debugging in staging environments with request-level visibility, Application Performance Management (APM) tools can be as useful for containerized apps as they are for traditional apps.
- **OpenTracing:** OpenTracing is a new, open distributed tracing standard for applications. It allows developers of applications to instrument code for transaction tracing without binding to any particular tracing vendor.


TIP

Tracing has a significant performance impact, which is why it's generally limited to troubleshooting. Often developers want specific metrics on their applications performance and

run-time behavior. Custom metric frameworks such as JMX, StatsD, and Prometheus provide the required information without performance drawbacks.

Events, logging, and backtracing

There's a lot more behind the scenes in a container-centric architecture, though. Three additional elements help to complement the observability of the infrastructure, applications, and services:

- **Event monitoring** is the process of collecting and signaling event occurrences to administrators. For containers this might include events such as image pull, start, kill, or out of memory. Administrators may want to be notified when these events take place.
- **Logging** involves the collection of computer-generated errors or informational messages for analysis of system behavior. Each event in your infrastructure — from containers to operating systems and applications designed to do so — generates data that is recorded in logs.
- **Backtracing** provides information about what happened when an application crash occurred. By examining the backtrace, you can try to determine the cause of an application crash and pinpoint where you should focus your troubleshooting efforts.

Dashboarding and Exploring

A key component of any container monitoring solution is the visual display of the metrics and events. Graphical dashboards, as well as other more dynamic visualizations such as topology maps or hierarchical explore views, simplify the task of understanding your environment and identifying anomalies.

Alerting

Just like in all areas of IT, alerting in a container environment is a critical activity when it comes to identifying potential problems and events that can hinder application performance and availability. Keeping track of what's happening, especially in a large, dynamic environment, requires automation.

However, unlike in traditional environments, a process dying or a container being killed doesn't necessarily mean that there is a problem. Orchestration tools are generally designed to handle exactly these situations and can often self-heal and bring things back into working order without administrator intervention. Still, administrators may need to be notified if a problem will affect the platform or users. That's why alerting must be implemented and why it must have connections to let it understand how the entire environment operates.

Troubleshooting: Going Beyond Monitoring

A key challenge with troubleshooting containers is that they may no longer exist after a problem occurs.

REMEMBER

This is by design. Containers are often described as *ephemeral* constructs that last for only as long as they're needed. When their single task is complete, the container is stopped, destroyed, or otherwise disposed of while the application moves on to the next step in its process.

Orchestration tools schedule and reschedule containers as the environment changes. Comprehensive container monitoring solutions should include the ability to automatically record all of the activity on a system that takes place surrounding an event. Capturing information such as commands, process details, network activity, and file system activity allows after-the-fact investigation, even after containers are gone and outside the production environment.

Choosing a platform

Monitoring systems that collect, analyze, dashboard, and alert on containers come in different forms. Open source options that you need to build, maintain, and scale on your own are available. The industry also provides software-as-a-service (SaaS) solutions in a fully managed and fully supported cloud service with no maintenance.

Moreover, on-premises solutions are available that you can deploy as software in a private cloud for greater security and isolation. Some solutions provide a vertically scalable, single-server backend solution for moderate monitoring workloads, and others follow a horizontally scalable distributed systems model to enable flexible scaling to better accommodate growth over time to large-scale monitoring with long-term data retention.

Which platform you choose should take into account expected growth, security and compliance requirements, and historical retention.

IN THIS CHAPTER

- Identifying common container threats
- Handling configuration and compliance
- Ensuring run-time security

Chapter 4
Securing Containers

Maintaining secure practices is a never-ending cyclical battle. Even as you implement brand new security measures and install patches, you've arrived just in time for the next vulnerability to be exposed. And so it goes. . . .

A well-designed container architecture can be used to make your environment more secure. Even without additional services being added, containers can provide additional levels of security. Containers are isolated, have fewer dependencies, and should be immutable. However, establishing secure processes in your organization can be challenging because there are so many moving pieces.

Identifying Common Threats

A number of security issues can plague containers. The following sections describe some of the security considerations you need to account for in your container environment.

Container resource abuse

Thanks to their lightweight and single-process nature, containers typically far outnumber virtual machines. Their nature makes it possible for you to spawn big clusters of them on modest hardware. Because of this volume, software bugs, design miscalculations, or a deliberate malware attack can easily cause a denial-of-service.

By implementing container limits, you can prevent your containers from consuming too many resources. If containers don't have limits, they can easily affect the host and deny critical resources to other workloads.


TIP

With container resource limits, you gain the ability to set CPU and memory limits on a container. Over time, you should implement monitoring to compare these constraints against actual consumption.

Outdated container images

The longer you run software without updating it, the more likely it is to eventually be exploited. Your service or application might pull a hardcoded image from a repository, or that image might not be rebuilt as the base image is updated with patches or other new software. This means you could easily have older, vulnerable software running in production.

TIP

You can take several steps to avoid this situation. First, you should continually monitor how long containers have been running in production and make sure they stay current. Second, try to avoid running different versions of the same image in production. This helps to avoid confusion and inconsistencies that can lead to insecurity. Finally, make sure you use a vulnerability scanner to stay up to date on any potential security issues in the software you use.

Secrets management

Software needs sensitive information to run. This includes user password hashes, server-side certificates, encryption keys, and more. This sensitive information should be handled independently from application code, container images, and configuration, and should be stored in a secure location such as a secrets vault. Most

container orchestration platforms have this functionality built in through a feature called *secrets management.*

As with most things, you should follow best practices with regard to secrets management. One common practice is to use environment variables for secrets, although this is an insecure practice that you should avoid. Further, avoid embedding secrets inside container images.


TIP

Consider the use of a Docker credentials management system, but do not attempt to create your own unless you know exactly what you are doing. Doing otherwise could create a major security issue.

Container image authenticity

Plenty of Docker images and repositories doing all kinds of awesome and useful stuff are available on the Internet, but if you are pulling images without using any trust and authenticity mechanism, you are running arbitrary software on your systems. Here are some important questions you should ask:

- Where did the image come from?
- Do you trust the image creator?
- When was the image last updated?
- Which security policies are they using?

- Do you have objective cryptographic proof that the author is actually that person?
- How do you know nobody has tampered with the image after you pulled it?


TIP

Common sense should prevail, along with some additional steps. First, regular Internet common sense applies: Do not run unverified software and do not run software from sources you don't explicitly trust. Next, deploy a container-centric trust server using some of the Docker registry servers available. Finally, enforce mandatory signature verification for any image that is going to be pulled or running. By doing so, you can make sure that you're always running approved software.

Shared kernel architecture

If an attacker compromises your host system, the container isolation and security safeguards won't make much of a difference. The host system is a shared component that, once compromised, results in the compromise of underlying containers. Take appropriate steps to restrict access to unnecessary kernel functionality on containers and host OS services.

One potential issue with containers is referred to as *container breakout*. This phenomenon occurs when a container has bypassed isolation checks and accesses

sensitive information from the host or gained additional privileges. In order to prevent this, it's important to reduce the default container privileges, which can be accomplished via various means.


TIP

Most importantly, services and users should only have access to services that they need. Drop capabilities — fine-grained access control beyond the root all or nothing — that are not required. As you add services, avoid running containers as a privileged (root) user, privileged containers, and sensitive mount points. Lastly, make sure your policies enforce mandatory access control to prevent undesired operations — both on the host and on the containers — at the kernel level.

Run-time security monitoring

But what if, despite all these precautions, the image has been compromised during run-time and starts to demonstrate suspicious activity? What if your own in-house application has a vulnerability you didn't know about? Or, what if you discover that attackers are using a zero-day exploit not detected by your scanning services? Run-time security can be compared to Windows anti-virus scanning. It lets you detect and prevent an existing breach from further penetration deeper into the system.

Handling Configuration and Compliance

Half of the battle in security is making sure your teams follow secure configuration and compliance practices. Luckily, containers have many default run-time security features, some described in this section, and their portability eases the burden on developers.

Compliance checks

The Center for Internet Security (CIS) has published general security recommendations for Docker and Kubernetes. These recommendations can be tested against your infrastructure with some scripts known as benchmarks, docker-bench, and kube-bench projects. Run these periodically against your infrastructure to see if it meets best practices.

Kubernetes security features

Kubernetes has many security features baked into it. Before evaluating external vendors, check out what Kubernetes offers first. These are a few of Kubernetes's security capabilities:

- **Role-based access control (RBAC):** RBAC specifies the authorization and access control specifications that define the actions allowed over Kubernetes entities.

- **Pod security policy:** Using security policies, you can restrict the pods that will be allowed to run on your cluster. For example, you can configure resources, privileges, and sensitive configuration items.
- **Network policy:** A network policy is a specification of how groups of pods are allowed to communicate with each other and other network endpoints.

Ensuring Run-Time Security

Once a container is deployed, you need to put security measures in place to detect violations of expected activity or prevent certain system calls, processes, or network connections from occurring that could be detrimental.

You have several ways to secure your containers at run-time, including these:

- **Seccomp:** This method allows you to restrict the system calls available within a container so that the container stays within its boundaries.
- **SELinux and AppArmor:** These are traditional run-time Linux enforcement modules that can also be used to improve the security of containers.
- **Falco:** This is a behavioral activity monitor that gives visibility into the behavior of your containers

and applications. Falco lets you drill down to details such as system, network, and file activity.

- **Sysdig Secure:** This is a powerful run-time security and forensics solution for your containers and microservices. Sysdig Secure also provides vulnerability management and image scanning as well as compliance and audit services.

Working with Forensics and Incident Response

Containers are ephemeral and bring challenges to collecting the data that is needed to properly investigate an event and comply with the granular forensic requirements dictated by many regulation regimes.

With VMs, you have the ability to connect remotely to the system using something like SSH. However, with containers, chances are good that the container might have disappeared after the security incident — that's just a fact in the world of containers. Logs have limited information, making it difficult to understand why something went awry.

Fortunately, open source options such as Sysdig Inspect enable the recording of all system activity. With more information available, you increase the ability to undertake deep forensics and post-mortem analysis.

IN THIS CHAPTER

- Reviewing key points about containers
- Deploying containers effective and securely

Chapter 5
Ten Container Takeaways

As containers continue to make their way into a dominant role for large-scale application deployments, you should keep these critical considerations in mind:

- **Choose and implement your orchestration tool early on to support your containers in production.**
- **Take a security-centric approach as you build your environment.**

- **Trust and then verify via ongoing monitoring.**
- **Implement a CI/CD/CS pipeline.**
- **Embrace and manage increased complexity.**
- **Understand your monitoring and instrumentation options and the pros and cons of each approach.**
- **Monitor everything.**
- **Make sure you can measure and correlate incident responses.**
- **Use a secrets management framework.**
- **Look beyond deployment.**

Securely deploy containerized applications

Container adoption has spiked since the introduction of the Docker ecosystem and its security improvements. This book provides you with an overview of the containers landscape as well as the evolving development processes and architectures. Learn how to capitalize on the opportunities while navigating the obstacles of deploying containerized applications in production.

Inside…

- Run applications in containers
- Compare orchestrators
- Set up a delivery pipeline
- Automate testing and integrate events
- Gain runtime visibility
- Respond to incidents
- Evaluate risk

Sysdig

Jorge Salamero Sanz is Technical Marketing Manager at Sysdig. **Eric Carter** is Director of Product Marketing at Sysdig. **Knox Anderson** is Senior Product Manager at Sysdig.

ISBN: 978-1-119-52110-5
Not For Resale

for dummies®
A Wiley Brand

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.