SPONSORED BY

Twistlock™

GEEK GUIDE

Managing Container Security and Compliance in Docker

LINUX™ JOURNAL

Table of Contents

About the Sponsor 4
Introduction 5
Linux Virtualization 6
Containers 7
Docker 10
Comparing Docker to LXC 11
- Process Management 11
- State Management 12
- Portability 12

Common Uses 12
Orchestration 13
Image Security and Compliance 14
Enter Twistlock 15
Design and Implementation 17
- Runtime and Protection 19
- Vulnerability Management 19
- Continuous Integration 20
- Compliance 20
- Access Control 21
- Analytics 21

Additional Emphasis on Compliance and Security 22
Summary 23

PETROS KOUTOUPIS is currently a senior software developer at **IBM** for its **C**loud **O**bject **S**torage division (formerly **C**leversafe). **H**e is also the creator and maintainer of the **R**apid**D**isk **P**roject (http://www.rapiddisk.org). **P**etros has worked in the data storage industry for more than a decade and has helped pioneer the many technologies unleashed in the wild today.

GEEK GUIDES:
Mission-critical information for the most technical people on the planet.

About the Sponsor

Twistlock

Twistlock protects today's applications from tomorrow's threats with advanced intelligence and machine learning capabilities. Automated policy creation and enforcement along with native integration to leading CI/CD tools provide security that enables innovation by not slowing development. Robust compliance checks and extensibility allow full control over your environment from developer workstations through to production. As the first end-to-end container security solution, Twistlock is purpose-built to deliver modern security.

Managing Container Security and Compliance in Docker

PETROS KOUTOUPIS

Introduction

In recent years, operating system virtualization has taken the industry by storm. It allows for minimizing the over-provisioning of resources and, in turn, re-using them at the end of the virtual server lifecycle. The idea behind it is a noble one. Why invest in allocating more server hardware and not utilize it to its full potential, when instead you can consolidate it all onto one or a few servers and share their

resources? In turn, the costs to acquire new hardware, energy consumption and management are reduced exponentially. To add to this, most commercial hypervisors offer additional migration/recovery and high availability functions (in the event of a failure). It is no surprise that the technology has become commonplace in the modern computing industry.

Linux Virtualization

What exactly is virtualization? Virtualization provides users with a virtualized platform (that is, virtual machine) that behaves much like physical hardware. When enabled, you are able to install or execute an instance of a traditional operating system into this virtual environment, and it typically would operate as it normally would on traditional hardware.

Each virtual instance is managed through a hypervisor. A hypervisor is a piece of computer software that creates, runs and manages one or more instances of these virtual machines. The hypervisor typically is referred to as the *host machine*, while the virtual machines often are called *guest machines*. The hypervisor's primary role is to translate all of the virtual hardware resources requested by the guest machine over to the physical system's hardware resources.

Not all hypervisors are created equal though. Linux supports multiple hypervisors, some of which are commercially developed and supported by software vendors, and others are developed for and by an open-source community. The Linux kernel ships with the KVM and Xen hypervisors. These hypervisors are designed to install and run

Instead of creating a full-fledged virtual machine, LXC enables a virtual environment with its own process and network space.

full operating systems and applications not native to the host machine's environment. It is pretty far removed from bare-metal performance and functionality, and it does introduce a layer of performance degradation as the hypervisor attempts to map virtual addresses to physical ones.

A secondary method of virtualization is called operating-system-level virtualization. This is where the hypervisor executes one or more *isolated* instance of the host machine. In Linux, these are called containers.

Containers

Linux Containers (LXC) is about as close to bare metal that one can get when running virtual machines. It imposes very little to no overhead when hosting virtual instances. First introduced in 2008, LXC adopted much of its functionality from the Solaris Containers (or Solaris Zones) and FreeBSD jails that preceded it. Instead of creating a full-fledged virtual machine, LXC enables a virtual environment with its own process and network space. LXC leverages the kernel's very own cgroups functionality to accomplish this. Control Groups (cgroups) is a kernel feature that limits, accounts for and isolates the resources used by one or more processes and limits its access to the the CPU, memory, disk I/O,

network and so on. Think of this userspace framework as a very advanced form of *chroot*.

So, what are containers? The short answer is that containers decouple software applications from the operating system, giving users a clean and minimal Linux environment while running everything else in one or more isolated "containers".

The primary purpose for enabling a container is to launch a limited set of applications or services (often referred to as microservices) and have them run within their own sandboxed environment. This isolation prevents processes running within a given container from monitoring or affecting processes running in another container. Also, these containerized services do not influence or disturb the host machine. The idea of being able to consolidate many services scattered across multiple physical servers into one is one of the many reasons data centers have chosen to adopt the technology.

Container features include:

- Security: network services can be run in a container, thus limiting the damage caused by a security breach or violation. An intruder who successfully exploits a security hole on one of the applications running in that container is restricted to the set of actions possible within that container.

- Isolation: containers allow the deployment of one or more applications on the same physical machine, even if those applications must operate under different

domains, each requiring exclusive access to its respective resources. For instance, multiple applications running in different containers can bind to the same physical network interface by using distinct IP addresses associated with each container.

- Virtualization and transparency: containers provide the system with a virtualized environment that can hide or limit the visibility of the physical devices or system's configuration underneath it. The general principle behind a container is to avoid changing the environment in which applications are running with the exception of addressing security or isolation issues.

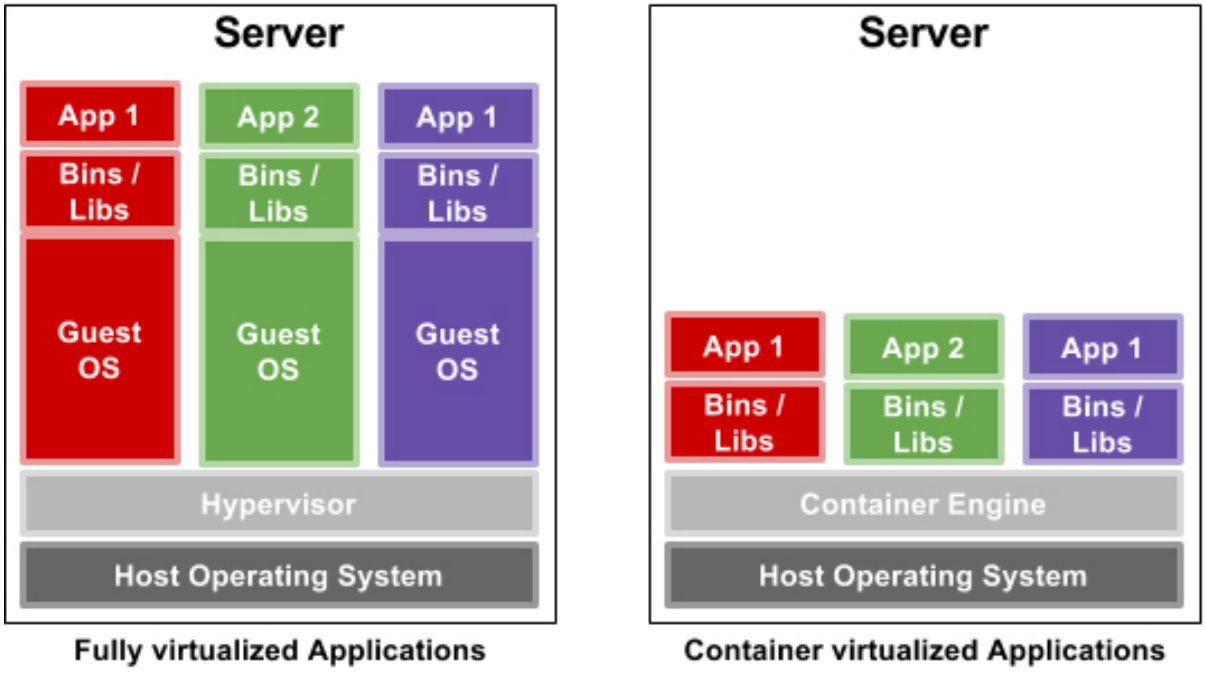

FIGURE 1. A Comparison of Traditional Virtualization to Containers

Docker

In recent years, the term Docker has become quite commonplace within the data center. But, what is it really? In the world of technology, the word "docker" can mean several things—from the company of the same name to a collection of software tools, to the project those tools are built around and the community of users and developers supporting it. However, for the purpose of this guide, Docker is an Apache-licensed open-source containerization technology designed to automate the repetitive task of creating and deploying microservices inside containers. Docker treats containers as if they were extremely lightweight and modular virtual machines.

Initially, Docker was built on top of LXC, but it has since moved away from that dependency, thus resulting in a better developer and user experience. Much like LXC, Docker continues to make use of the kernel cgroup subsystem. The technology is more than just running containers; it also eases the process of creating containers, building images, sharing those built images and versioning them.

Some of the features that Docker brings into the equation include:

- Portability: Docker provides an image-based deployment model. This type of portability allows for an easier way to share an application or set of services (with all of their dependencies) across multiple environments.

- Version control: a single Docker image is made up of a series of combined layers. A new layer is created

whenever the image is altered. For instance, a new layer is created every time a user specifies a command, such as *run* or *copy*. Docker will reuse these layers for new container builds. Layering to Docker is its very own method of version control.

- Rollback: again, every Docker image has layers. If you do not wish to use the current running layer, you can roll back to a previous version. This type of agility makes it easier for software developers to integrate and deploy their software technology continuously.

- Rapid deployment: provisioning new hardware often can take days. And, the amount of effort and overhead to get it installed and configured is quite burdensome. With Docker, you can avoid all of that by reducing the time it takes to get an image up and running to a matter of seconds. When you are done with a container, you can destroy it just as easily.

Comparing Docker to LXC

Fundamentally, both Docker and LXC are very similar. They both are userspace and lightweight virtualization platforms that implement cgroups and namespaces to manage resource isolation. However, there are a number of distinct differences between the two.

Process Management Docker restricts containers to run as a single process. If your application consists of X number of concurrent processes, Docker will want you to run X number of containers, each with its own distinct process.

This is not the case with LXC, which runs a container with a conventional init process and, in turn, can host multiple processes inside that same container. For example, if you want to host a LAMP (Linux + Apache + MySQL + PHP) server, each process for each application will need to span across multiple Docker containers.

State Management Docker is designed to be stateless, meaning it does not support persistent storage. There are ways around this but, again, only necessarily when the process requires it. When a Docker image is created, it will consist of read-only layers. This will not change. During runtime, if the process of the container makes any changes to its internal state, a `diff` between the internal state and the current state of the image will be maintained until either a *commit* is made to the Docker image (creating a new layer) or until the container is deleted, resulting in that `diff` to disappear.

Portability This word tends to be overused when discussing Docker—that's because it is the single most important advantage Docker has over LXC. Docker does a much better job of abstracting away the networking, storage and operating system details from the application. This results in a truly configuration-independent application, guaranteeing that the environment for the application always will remain the same, regardless of the machine on which it is enabled.

Common Uses

Docker is designed to benefit both developers and system administrators. It has made itself an integral part of many

DevOps (developers + operations) toolchains.

Developers can focus on writing code without having to worry about the system ultimately hosting it. With Docker, there is no need to install and configure complex databases or worry about switching between incompatible language toolchain versions. Docker gives the operations staff flexibility, often reducing the number of physical systems needed to host some of the smaller and more basic applications. Docker streamlines software delivery. New features and bug/security fixes reach the customer quickly without any hassle, surprises or downtime.

Orchestration

On its own, Docker is extremely simple to use, and running a few images simultaneously also is just as easy. Now, scale that out to hundreds, if not thousands, of images. How do you manage that? Eventually, you need to step back and rely on one of the few orchestration frameworks specifically designed to handle this problem. Enter Kubernetes and Swarm.

Kubernetes originally was developed by Google. It is an open-source platform that also automates container operations. Google was an early adopter and contributor to the Linux Container technology. In fact, it is Linux Containers that powers Google's very own Cloud services. Anyway, Kubernetes eliminates all of the manual processes involved in the deployment and scaling of containerized applications. It is capable of clustering together groups of servers hosting Linux Containers while also allowing the administrator to manage those

clusters easily and efficiently.

Docker's very own version of this platform is called Swarm. It accomplishes much of the same tasks and boasts a lot of the same features. The primary difference between the two though is that Swarm is centralized around the use of Docker, while Kubernetes tends to adopt a more generalized container support model.

Sometimes production applications will span across multiple containers, and those containers may be deployed across multiple physical server machines. Both Kubernetes and Swarm give you the orchestration and management capabilities required to deploy and scale those containers to accommodate the always changing workload requirements.

Image Security and Compliance

Although Docker brings security to running applications in a shared environment, containers alone are not an alternative to taking proper security measures. Unlike traditional hypervisors, a container can have a more direct path to the host operating system's kernel, which is why it is standard procedure to drop privileges as quickly as possible and run all the services as non-root wherever possible. Also, whenever a containerized process requires access to the underlying filesystem, you should make it a good habit of mounting that filesystem as read-only.

One of the biggest concerns with Docker is that it has become quite easy to download and run any random Docker image found on the internet. Not all of those sources should be trusted. When running unknown

or unofficial images, you increase the risk of running vulnerable or buggy code in your environment. And if your container is hosting a privileged process, any attack exposing a potential vulnerability eventually could own the entire host system. This is why you need to download those images from a trusted repository or build and maintain your very own. These trusted sources typically will apply security updates to patch vulnerabilities. They also will have a team of software engineers working tirelessly not only to integrate security enhancements but also to manage and maintain the packages within that image. I can't stress this enough: *run Docker images from trusted parties only*.

There also exists the potential for a Docker container to run system binaries that it probably shouldn't be touching in the first place, at least without your knowledge. Another similar scenario is when a rogue application or attacker gains container access through an application vulnerability and replaces some of the underlying system binaries with one that does not belong or was not intended to run in that Docker image—all of which will continue to run during the life of that container. This can result in additional system and network compromises or worse.

Enter Twistlock

Hope is not lost. There are solutions intended to prevent such things from ever occurring. Twistlock is one such software vendor providing that type of solution. Twistlock develops and distributes a product of the same name

Twistlock offers the first end-to-end security solution built for containerized environments.

focusing on securing cloud-native apps and container environments. I also appreciate the play on words here: a "twistlock" is a rotating connector used for securing shipping containers.

Twistlock offers the first end-to-end security solution built for containerized environments. It protects against software exploits, malware and active threats by automatically creating predictive models that associate every image with an intent. These models are deduced using a combination of static analysis and machine learning capabilities. Twistlock then plugs into each phase of the container lifecycle, starting with development, through to deployment, and into runtime. Twistlock's early lifecycle tools are important because it makes security a priority, rather than an afterthought during the development process. After a container is deployed, Twistlock ensures your containers remain secure, so that even as time passes, and new threats emerge, they remain resistant to attack.

Twistlock sources more than 30 vulnerability and threat feeds, combining it with its proprietary research. This ensures that Twistlock's customers are kept updated, in real time, on all known application CVEs (Common Vulnerabilities and Exposures), exploits and threats.

Design and Implementation

Think of Twistlock as a tool to both harden your images in development and protect them against runtime threats. Twistlock is built as a Docker image and runs as a privileged container image on top of the Docker Engine. The idea is to run a single instance of this Twistlock image on every physical or virtual machine hosting Docker containers.

Each instance of Twistlock can be managed from the Twistlock Console. Through this very same console, you can create/remove security policies, establish image compliance and also monitor the security state of each running container. If that container surpasses the defined threshold of vulnerabilities or does not comply to the parameters you have set, Twistlock either will

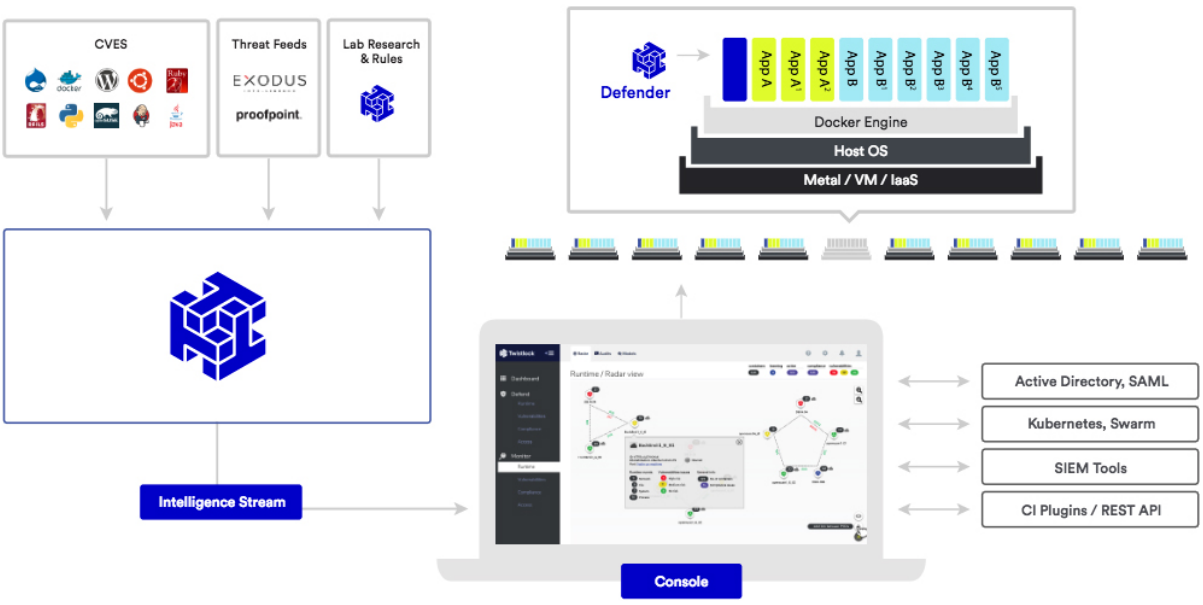

FIGURE 2. A General Overview of Twistlock

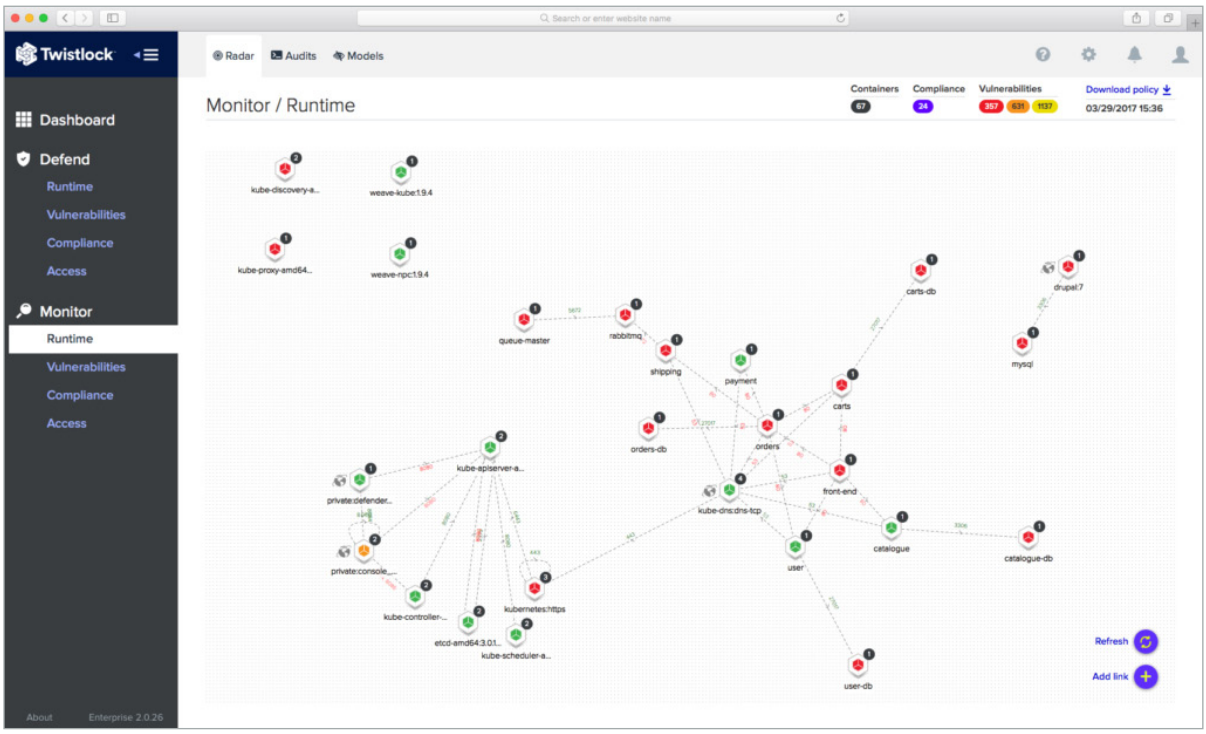

FIGURE 3. The Twistlock Management Console

prevent that container from running or disconnect it completely from the network.

The Twistlock Intelligence Stream is a live feed that is delivered to your installation. It is updated several times a day and contains the latest threat data (CVEs, malware checksums, malicious endpoints and zero-day defects).

An additional feature supported by Twistlock is its ability to scan through all the running containers' images and build a database storing the checksums of each packaged execution binary and library. If Twistlock detects that a file has been altered, it will flag that container and take the proper course of action to prevent it from disrupting other services or from

wreaking havoc on the system.

Much like any other Docker image, a Twistlock Docker image can be managed and deployed through the same orchestration frameworks discussed earlier: Kubernetes and Swarm. Twistlock also features a RESTful Application Program Interface (API) to manage various aspects of the Twistlock ecosystem from another framework or an external application.

Twistlock is broken down to handle the following actions.

Runtime Protection Remember, Twistlock's runtime defense protects your containers against detected exploits, compromises, application flaws and configuration errors. It actively monitors container activities and detects policy violations. Twistlock will report all anomalous behaviors while also taking the appropriate actions to disconnect or isolate them.

Twistlock automatically creates predictive models that associate every image with an intent. These models are built using a combination of static analysis and machine learning. With these models, Twistlock can identify when a container does something that it shouldn't be doing. Sensors enforce the model and take action when the model is breached. For example, the process sensor can detect when a container that runs nginx suddenly runs netstat. The model says that netstat isn't a whitelisted process in an image that is built to load-balance web traffic, and the anomaly could indicate an emerging threat.

Vulnerability Management Twistlock is constantly scanning container images in registries, workstations and servers for known vulnerabilities and misconfigurations.

I mentioned earlier that Twistlock aggregates this information from multiple external and internal sources. All detected vulnerabilities are reported and extend across Linux distributions (Debian, Ubuntu, Fedora), application frameworks (Node.js, Python, Java) and even your custom application packages. Twistlock breaks the Docker image apart and parses each individual layer, specifically searching for these threats. Twistlock can and will take remediation actions based on the severity of the vulnerability during runtime.

Twistlock provides users with granular control when managing the types of vulnerabilities beyond their severity ratings. You can block individual CVEs explicitly while ignoring others.

Continuous Integration Twistlock was written to integrate directly into your Continuous Integration (CI) process (such as Jenkins). This way, it can find and report problems before they ever make it out into production. In some cases, when a package with an open CVE is reported, Twistlock also will report the package version that has the fix. Developers are given clear insight into the vulnerabilities present in every build. These plugins allow you to define and enforce your vulnerability policies at build time. For instance, you can set a policy requiring that one build job must not have any vulnerability, or you can flag specific CVEs while ignoring the rest.

Compliance The Center for Internet Security (CIS) Docker Benchmark provides guidance for establishing a secure configuration of a Docker container. In short, this benchmark provides the best security practices for

Using Twistlock, you can define and enforce policies governing user access to both Docker and Kubernetes resources, limiting specific users to individual functions or APIs.

deploying Docker. Twistlock has developed 80+ built-in checks to validate the recommended practices from this benchmark. In parallel to this, Twistlock includes an extensive list of configuration checks for the host machine, Docker dæmon, Docker files and directories.

Organizations using Twistlock will be able to enforce *Trusted Registries* (containing images approved by Twistlock) and *Trusted Images*. When configured, Twistlock can enforce that the images from these trusted lists are the only ones deployed onto production servers.

Access Control Using Twistlock, you can define and enforce policies governing user access to both Docker and Kubernetes resources, limiting specific users to individual functions or APIs. Out of the box, Twistlock supports enterprise identity directories that include Active Directory, OpenLDAP and SAML providers. This way, you can specify access policies to container resources without the need to create new identities and groups. You can monitor detailed user access audit trails, action types, services requested and more from the console.

Analytics Twistlock's built-in analytics allow you to visualize all relevant data and enable you to enforce standard

configurations, container best practices and recommend deployment templates. This way, your containers will remain compliant to industry or company policies.

All data (audit events) are made available via open formats (CSV) and APIs (JSON). This makes it easier for you to import Twistlock data into the log analytics tools you may be currently using (such as IBM's QRadar, HP's ArcSight, Datadog, Sumo Logic and Splunk).

Additional Emphasis on Compliance and Security

The National Institute of Standards and Technology (NIST) is a nonregulatory agency of the United States Department of Commerce. NIST is a measurement standards laboratory created with the sole mission of promoting both innovation and industrial competitiveness. One such standard defined by NIST is the Health Insurance Portability and Accountability Act (HIPAA). The HIPAA Security Rule establishes national standards to protect an individual's electronic personal health information that is created, received, used or maintained by a covered entity. It requires that appropriate administrative, physical and technical safeguards are in place to ensure the confidentiality, integrity and security of that individual's electronic-protected health information.

There is also the Payment Card Industry Data Security Standard (PCI DSS) administered by the Payment Card Industry Security Standards Council. This standard is intended for organizations handling branded credit cards

from the major card schemes including Visa, MasterCard, American Express and Discover, and it was created to increase controls around cardholder data to reduce credit-card fraud.

This is yet another area where Twistlock truly shines by providing its users with a complete guide to configure the Docker container images in compliance with the HIPAA Security Rule and PCI Security Standards. Twistlock is also compliant with the recommendations from the recently published NIST Application Container Security Guide (http://csrc.nist.gov/publications/drafts/800-190/sp800-190-draft.pdf). This guide details the security benefits and concerns associated with container technologies.

Summary

Although containers themselves provide you with a more secure model in their implementation of application isolation, it is often not enough to rid yourself of the common security concerns plaguing all published software unleashed in the wild. Containers allow teams to move faster and deliver more innovation, but it is a Herculean task to keep up with the pace and continually update security policies and check for exposure to the latest vulnerabilities. Twistlock eliminates the manual processes and will let you know which applications and which containers are affected by those vulnerabilities while also ensuring that misbehaving containers, unauthorized images, unauthorized users and rogue processes are always kept at bay.■