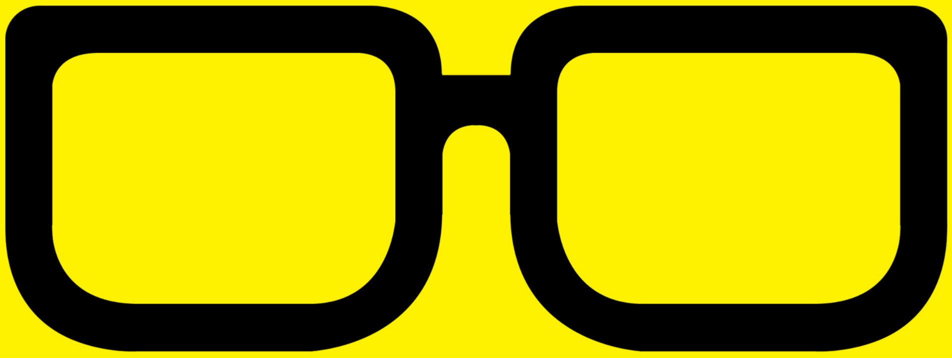
SPONSORED BY

Twistlock™

GEEK GUIDE



Securing Serverless Applications

LINUX™ JOURNAL

Table of Contents

Introduction 5

Introducing Cloud Native Computing 6

Going Serverless 8

The Shrinking Operating System 9

The Evolution of Containers in the Cloud 11

Security on Serverless Systems 13

The Importance of Application Security 15

Twistlock 16

The Many Benefits of Using Twistlock 17

Leveraging Twistlock for a More Secure Cloud Deployment 20

The Cloud Native Computing Foundation 21

Summary 22

PETROS KOUTOUPIS is currently a senior platform architect at IBM for its Cloud Object Storage division (formerly Cleversafe). He is also the creator and maintainer of the RapidDisk Project (http://www.rapiddisk.org). Petros 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.

Securing Serverless Applications

PETROS KOUTOUPIS

Introduction

Access to services and data is what drives this little thing we refer to as the cloud. In the past decade alone, the paradigm shift toward a wider and more accessible network has forced both hardware vendors and service providers to rethink their strategies and cater to a new model of storing and, in turn, accessing information. With that comes the responsibility to develop on service-focused applications and integrate them continuously into a stable and controlled environment. And as more individuals and businesses connect themselves to the greater world, it becomes increasingly necessary to secure the information that travels across our networks. This is where a serverless computing model begins to shine.

Often described as Functions as a Service (FaaS), serverless computing enables the division of system logic into a collection of independent functions or applications. This model adds a level of granularity to the types of services actually running and, in turn, to maintaining an accurate usage count of those services, affecting billing. This is unlike traditional whole server or virtual machine deployments where a user pays for each running instance. And for what? To host a few functions. Economic benefits aside, the serverless platform is designed for better security, fail-over and load-balancing. With all the overhead required to maintain fuller operating environments removed, developers are left with an isolated micro-instance in which they can develop and run their application code quickly.

This ebook introduces the concept of serverless computing as it relates to the cloud, while also highlighting the challenges of securing such an environment. Read on to learn what tools are available to deploy your applications securely.

Introducing Cloud Native Computing

Cloud native computing, a more focused term for serverless computing, is not only the latest trending buzzword in the data center, but it also offers a new way of hosting applications. The idea challenges what traditionally has been the norm and puts more power into the application itself while abstracting away everything underneath it.

Far too often, the idea of the "cloud" is confused with the internet in general. Although it's true that various components that make up the cloud can be accessible via the internet, they're not one and the same. In its most general terms, cloud computing enables companies, service providers and individuals to provision dynamically the appropriate amount of computing resources (for example, compute nodes, block or object storage and so on) for their application needs. These application services are accessed over a network and not necessarily a public network either.

There are three distinct types of cloud deployments: public, private and a hybrid of both.

The public cloud differentiates itself from the private cloud in that the private cloud typically is deployed in the data center and under the proprietary network using its cloud computing technologies—that is, it is developed for and maintained by the organization it serves. Resources for private cloud deployments are acquired via normal hardware purchasing means and through traditional hardware sales channels. This is not the case for the public cloud. Resources for the public cloud are provisioned dynamically to its user as they are requested, and they may be offered under a pay-per-usage model or for free.

Some of the world's leading public cloud offerings platforms include:

- Amazon Web Services (AWS).
- Microsoft Azure.
- Google Cloud Platform.
- IBM Cloud (formerly, SoftLayer).

This approach moves applications away from physical hardware and operating system dependency and into their own self-contained and sandboxed environment that can run transparently and seamlessly anywhere within the data center.

As the name implies, the hybrid model allows for seamless access and transitioning between both public and private deployments, all managed under a single framework.

In short, the cloud simplifies operating system/application deployment and the management of those allocated resources. It also reduces capital expenses and optimizes budgets by reducing the needs to acquire, configure and maintain data-center hardware. The best part of the cloud is that it is designed to scale to consumer needs or requirements.

Going Serverless

Cloud native computing is a more recent term describing the modern trend of deploying and managing applications. The idea is pretty straightforward. Each application or process is packaged into its own container, which in turn is orchestrated dynamically (that is, scheduled and managed) across a cluster of nodes. This approach moves applications away from physical hardware and operating system dependency and into their own self-contained and sandboxed environment that can run transparently and seamlessly anywhere within the data center. The cloud native approach is about separating the various components of application delivery. And with the entire operating system removed, it aids developers in writing cleaner, more robust and easier to debug code.

The Shrinking Operating System Once upon a time, data-center administrators deployed entire operating systems, occupying entire hardware servers, to host a few applications each. It was a lot of overhead with a lot to manage. Now, scale that across multiple server hosts, and it increasingly become more difficult to maintain. This was a problem and a problem that wasn't easily solved. It would take time for the technological evolution to bring us to where we are able to shrink the operating system to the point of running a type of virtualized technology commonly referred to as containers, but 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". It is about as close to bare metal that one can get when running a virtual instance, and the technology imposes very little to no overhead. Using namespaces to enforce process isolation and leveraging the kernel's very own Control Groups (cgroups) functionality, the feature limits, accounts for and isolates the CPU, memory, disk I/O and network usage of one or more processes.

The primary purpose for enabling a container is to

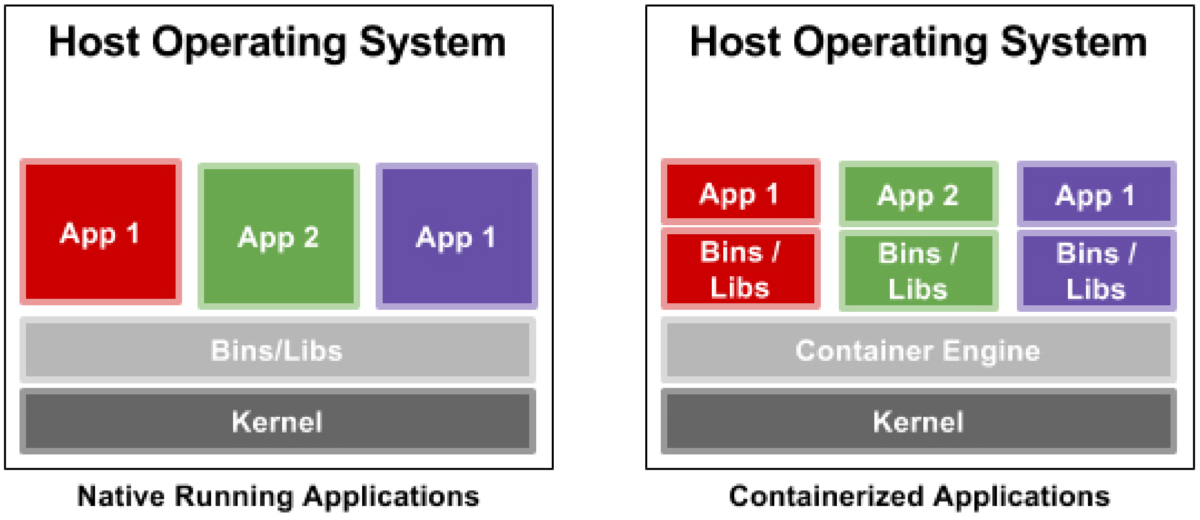


FIGURE 1. **A Comparison of Applications Running in a Traditional Environment to Containers**

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. It is a more secure way to host your applications. 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.

The most popular container technology is Docker. It is a userspace and lightweight virtualization platform that, again, utilizes cgroups and namespaces to manage resource isolation. Docker stood out from the rest by adding better portability for rapid image-based deployments and image version control.

The Evolution of Containers in the Cloud As one would expect, container technology has helped accelerate cloud adoption. Think about it. You have these persistent containerized application images that within seconds are spun up or down as needed and balanced across multiple nodes or data-center locations to achieve the best in quality of service (QoS). Even the big-time public cloud providers discussed earlier make use of the same container technologies and for the same reason: rapid application deployment. For instance, Amazon, Microsoft and Google provide their container services with Docker. And as it applies to the greater serverless ecosystem, the applications hosted in these containers are stateless and event-triggered. This means that a third-party component will manage access to this application, as it is needed and invoked.

Now when I think of a true serverless solution, one of the first things that comes to mind is Amazon's AWS Lambda. Amazon takes serverless to the next level with Lambda, by spinning up a container to host the applications you need, ensuring access and availability for your business or service. Under this model, there is no need to provision or manage physical or virtual servers. Assuming it is in a stable or production state, you just deploy your code and you are done. With Lambda, you do not manage the container (further reducing your overhead). Your code is just deployed within an isolated containerized environment. It is pretty straightforward. AWS Lambda enables user-defined code functions to be triggered directly via a user-defined HTTPS request. The way Lambda differs from traditional containerized deployment is that Amazon has provided a framework for developers to upload their event-driven application code (written in Node.js, Python, Java or C#) and respond to events, such as website clicks, within milliseconds. All libraries and dependencies to run the bulk of your code are provided for within the container. Lambda will scale automatically to support the exact needs of your application.

As for the types of events, labeled an *event source*, on which to trigger your application, or code *handlers*, Amazon has made it so you can trigger on website visits or clicks, a REST HTTP request to its API gateway, a sensor reading on your Internet-of-Things (IoT) device, or even an upload of a photograph to an S3 bucket. This API gateway forms the bridge that connects all parts of AWS Lambda. For example, a developer can write a *handler* to trigger on HTTPS request *events*.

Let’s say you need to enable a level of granularity to your code. Lambda accommodates this by allowing developers to write modular handlers. For instance, you can write one handler to trigger for each API method. And each handler can be invoked, updated and altered independent of the others.

Lambda allows you to combine all required dependencies (that is, libraries, native binaries or even external web services) to your function into a single package, giving a handler the freedom to reach out to any of those dependencies as it needs them.

By adopting a serverless paradigm for building deployable host applications, the operational responsibilities are delegated to the cloud provider.

Now, how does this compare to an Amazon AWS Elastic Cloud Computing (EC2) instance? Well, the short answer is that it's a lot more simplified, and by simplified, I mean, there is zero to no overhead on configuring or maintaining your operating environment. If you need more out of your environment that requires access to a full-blown operating system or container, you will spin up an EC2 virtual instance. EC2 provides users the flexibility to customize their virtual machine with both the hardware and software it will host. If you only need to host a function or special-purpose application, that is where Lambda becomes the better choice. With Lambda, there isn't much to customize. And sometimes, less is good.

Security on Serverless Systems

By adopting a serverless paradigm for building deployable host applications, the operational responsibilities are delegated to the cloud provider. There is no need for the application developer to patch the operating system or install an assortment of services to collect system logs and metrics. And while the serverless model does address a large portion of today's security concerns by pushing those concerns to the platform provider, it still isn't entirely immune to attacks. If anything, hackers or attackers will shift their focus away from the entire server and to the application itself (such as cross-site scripting and SQL injection). Even data at rest (that is, contents stored in your database) can become accessible to an attacker. Restricted access and crypto algorithms often are used to deter attackers from the latter.

With serverless deployments, you are most vulnerable in your code and the dependencies (which are quickly outdated) accessed by or bundled with your code. Monitoring for such flaws or inappropriate access becomes increasingly difficult. It is nearly impossible to track who is using your functions after it has been deployed. To minimize your exposure, it is recommended to disable or remove unused and stale code from your production environment. Delete unused functions. And if you know something has open vulnerabilities, either patch it or remove it. Also, make it a habit to update those dependencies. These dependencies can provide would-be attackers the entry they need to compromise your application.

Another way to minimize your exposure is to keep your functions simple and small, and in turn, restrict permissions to it. This also makes both your code and the environment in which it is executed easier to maintain. For instance, cloud platforms such as AWS enable Identity and Access Management (IAM) services to control access to AWS resources securely (including your Lambda deployments) for users. If it is an option, be sure to use it.

The Importance of Application Security

Not everything can and will run in this serverless environment. There will be a need to communicate with applications and services hosted on other systems (physical, virtual or in containers). And although container technology brings with it an added layer of security for running applications in an isolated 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, note that whenever a containerized process requires access to the underlying filesystem, you should make it a good habit of mounting that filesystem as read-only.

The state of a container image also raises concern—which is why it is improper to run containers (be it Docker or anything else) from an untrusted source. When deploying an unknown or unofficial image, you increase the risk of running vulnerable or buggy code in your environment. And if that container is configured to host a privileged process, any attack exposing a potential vulnerability eventually could cost the data center an entire host system.

There also exists the potential for a 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

Twistlock offers the first end-to-end security solution built for container, serverless and other cloud native environments.

that container 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.

Twistlock

Twistlock offers the first end-to-end security solution built for container, serverless and other cloud native environments. It protects against software exploits, malware and active threats through its advanced intelligence and machine-learning capabilities. Twistlock automatically profiles expected application behavior and enforces this as a whitelist-based security model—reducing manual overhead and delivering stronger security than traditional blacklist-based technologies. By integrating much earlier into the application lifecycle, Twistlock helps developers address security gaps and CVEs during the build phase—before code is ever deployed to production—whether the code is part of a serverless function, container image or other modern application.

Twistlock sources more than 30 vulnerability and threat intelligence feeds, combining them with its proprietary research from Twistlock Labs. This ensures that Twistlock customers are kept updated, in real time, on all known application CVEs (Common Vulnerabilities and Exposures), exploits and threats—while significantly reducing false positives.

The Many Benefits of Using Twistlock The Twistlock platform both addresses risks before deploying code to production and defends running applications against threats. Twistlock is built as a Docker image and runs as a container on each Docker engine to protect your running applications. An additional container provides vulnerability and compliance scanning, and a centralized management console.

Let's look at Twistlock's main benefits:

- Runtime protection: Twistlock 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, preventing disruption to any and all other containers across the Kubernetes cluster. Twistlock can identify when a container does something that it shouldn't be doing. For instance, if a container running nginx suddenly invokes netstat and netstat isn't a whitelisted process for that image, Twistlock will detect it.

- Vulnerability management: Twistlock Vulnerability Explorer provides in-depth details on the state of known CVEs impacting your serverless applications, including risk scoring, to help you identify risk and prioritize remediation. The Twistlock Intelligence Stream, which includes vulnerability data for serverless-specific threats and vulnerabilities, powers this vulnerability management capability. Twistlock provides users with granular control when managing the types of vulnerabilities beyond their severity ratings. For example, you can block individual CVEs explicitly while ignoring or alerting upon identifying others.

- Continuous integration: Twistlock integrates 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 and CIS Kubernetes Benchmarks provide guidance for establishing a secure configuration of a Docker container. In short, this benchmark provides the best security practices for deploying Docker. Twistlock has developed 150+ 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.

- Firewalls: Twistlock offers both Cloud Native Application Firewalls (CNAF) and Cloud Native Network Firewalls (CNNF) that provide added security for your production containers. Twistlock automatically maps, identifies and whitelists valid traffic flows in your environment based on proximity to your applications and knowledge of how they behave. Twistlock dynamically creates filters that automatically allow valid connections and drop suspicious connections, regardless of where your containers are running in the cluster.

- 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 allows you to 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.

Leveraging Twistlock for a More Secure Cloud Deployment Now that you understand a bit about the problem—that is, security in the cloud—it goes without saying that Twistlock tackles that problem head on. It will complement your current infrastructure while adding that additional layer of security and container image compliance across your entire cluster of nodes (local and remote).

Now how do all these components come together? It begins with users accessing your service through a website or via an application on their mobile devices. The web server

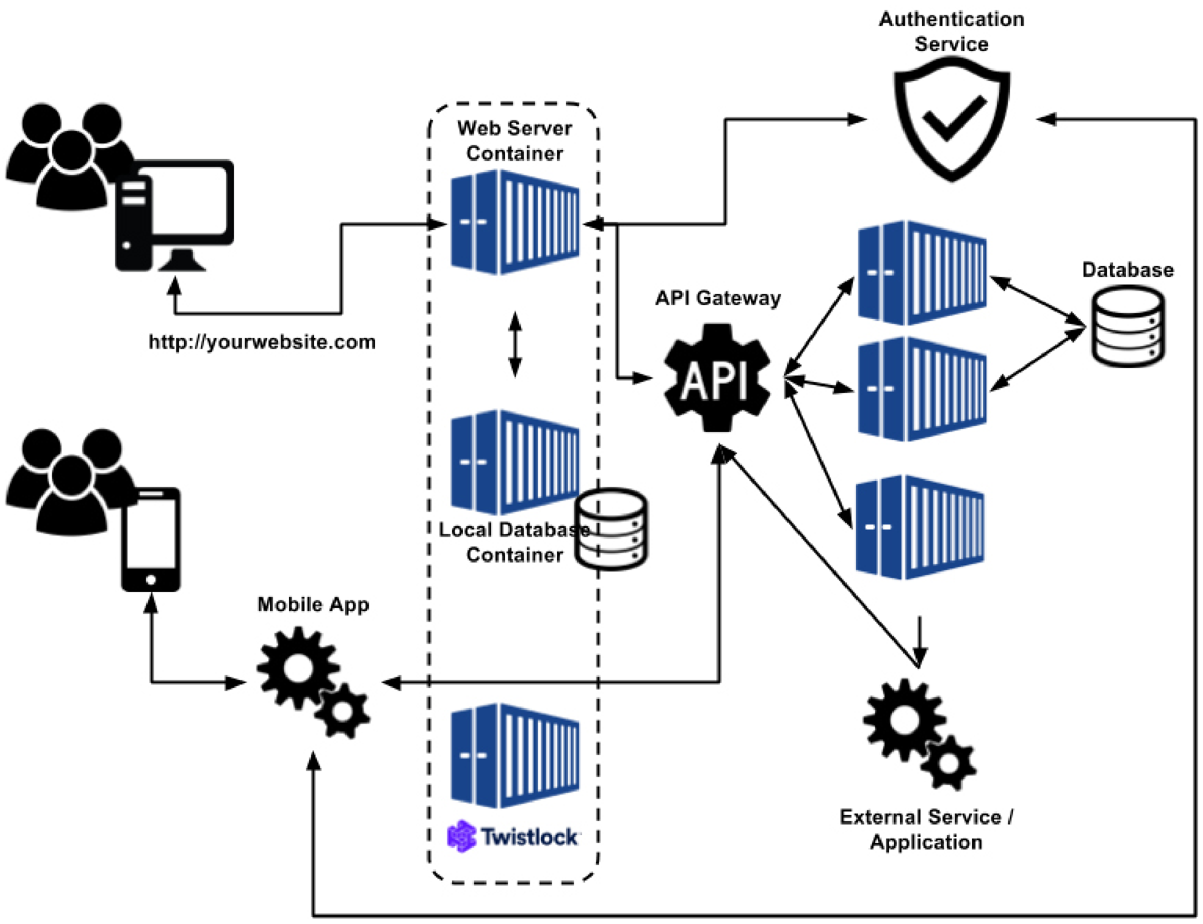


FIGURE 2. Putting All of the Pieces Together

and the various components on which it relies may be hosted from locally managed containers. Those containers are secured and monitored by Twistlock. If a particular function is required by either the web server container or the mobile application, it will reach out to a third-party authentication service, such as IAM, to gain the proper credentials for accessing the serverless functions hosted beyond the API gateway. When triggered, these functions will perform the necessary actions and return with whatever the web server or mobile application requested.

The Cloud Native Computing Foundation

Formed in 2015, the Cloud Native Computing Foundation (CNCF) was assembled to help standardize these recent paradigm shifts in hosting cloud services—that is, to unify and define the cloud native era. The primary goal of the foundation is to be the best place to host cloud native software projects. The foundation is home to many cloud-centric projects, including the Kubernetes orchestration framework.

To help standardize this new trend of computing, the foundation has divided the entire architecture to a set of subsystems, each with its own set of standardized APIs for inter-component communication. Subsystems include orchestration, resource scheduling and distributed systems services.

As a proud member of the foundation, Twistlock is committed to providing its customers with the most stable and secure serverless computing environments.

You can learn more about the foundation by visiting the

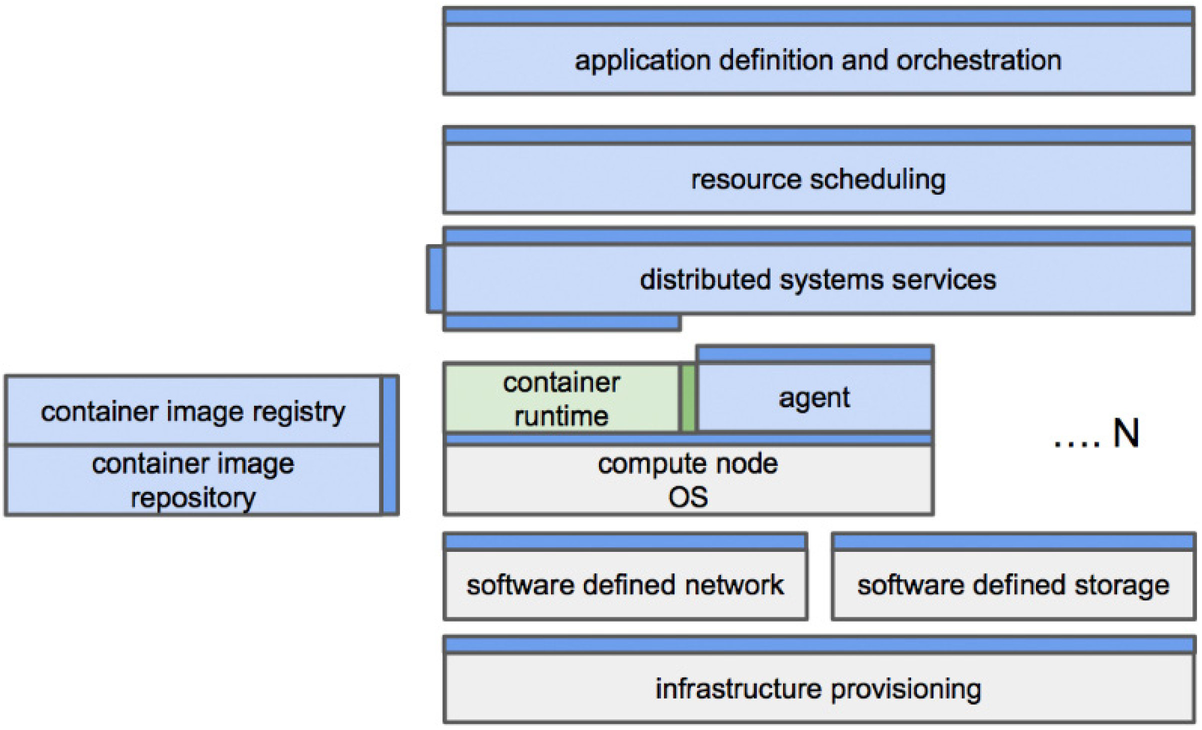


FIGURE 3. This diagram illustrates the subsystems as defined by the CNCF (from https://www.cncf.io).

foundation's official website: https://www.cncf.io.

Summary

DevOps and end users are in constant need of an environment where new or updated code can be deployed instantly. The cloud native computing model allows for this, and it also reduces the overhead and complexity in managing and maintaining the environment hosting the application. It is simple: focus on your code and nothing more.

With Twistlock in the picture, you can not only focus solely on your code, but you can do so with the assurance that it will be running in a secured environment.■