Security Assurance of Docker Containers

Stefan Winkle

Security Assurance of Docker Containers

GIAC GCCC Gold Certification

Author: Stefan Winkel, stefan@winkelsnet.com
Advisor: Adam Kliarsky

Accepted: 10/18/2016

Abstract

With recent movements like DevOps and the conversion towards application security as a service, the IT industry is in the middle of a set of substantial changes to how software is developed and deployed. In the infrastructure space, software developers have seen the uptake of lightweight container technology, while application technologies are moving towards distributed micros services. There is a recent explosion in popularity of package managers and distributors like OneGet, NPM, RubyGems, and PyPI. Amid this process, application containers technologies like Docker, LXC, and Rocket, used to compartmentalize software components, are getting immense popular. More and more software development becomes dependent on small, reusable components developed by many different developers and is often distributed by infrastructures outside control of development. As a result, the thread landscape is changing. Because of these changes the risk of introducing vulnerabilities in the development cycle has increased manifold. The Notary project, recently introduced in Docker, is built upon the assumption that the software distribution pipeline can no longer be trusted. Notary attempts to protect against attacks on the software distribution pipeline by association of trust and duty separation to Docker containers. In this paper, the Notary service will be explored with regards to an in-depth look at security testing of Docker containers.

1. Introduction

1.1. DevOps, SecDevOps, and DevSecOps

With recent growth of software and services delivered through cloud computing, information security is playing a catch-up game with rapid continuous development. One of the latest trends is around DevSecOps and/or SecDevOps. Jim Bird, author of 'DevOpsSec: Securing software through continuous delivery', explains in his book the contrasting perspectives of the growth of DevOps when he states "Some people see DevOps as another innovation, the newest thing over-hyped by Silicon Valley and by enterprise vendors trying to stay relevant. Others believe it is an authentically disruptive force that is radically changing the way that we design, deliver, and operate systems." (Bird, 2016) No matter what one believes, one cannot ignore that many companies, from small startups to Fortune 500 companies, including Google, Netflix, Etsy and Amazon, are having real success with DevOps at scale. In 2014, Amazon deployed 50 million changes: that is more than one change every second of every day (Brigham, & Liguori, n.d.). Google, well known for various cloud services like Gmail, Google Maps, etc. has also embraced DevOps technologies. Joe Beda, a senior staff engineer at Google, recently stated at a conference that "Google spins up more than 2 billion containers per week, more than 3,300 containers per second." (Beda, n.d.)

The various recent DevOps technologies that are being developed catalysis the speed of software development even further when are being used in conjunction. Victor Farcic, a senior consultant at CloudBees, explains in his book 'DevOps 2.0 toolkit', the relation between cloud services and containers. "On the first look, continuous deployment (CD), microservices (MS) and containers might seem like three unrelated subjects. After all, DevOps movement does not stipulate that microservices are necessary for continuous deployment, nor microservices need to be packaged into containers" (Farcic, 2016). But he goes on to explain that when these three concepts are bundled together, are very powerful. Combining these concepts, allows for split-second deployments, which decreases the time to market, while at the same time, the combination improves the quality of the services by providing continuous quality feedback loop and hence benefiting both worlds. Farcic explains that "MS are used to create complex systems composed of small and autonomous services that exchange data through their APIs and limit their scope to a specific bounded context." These services provide us with more freedom to make better decisions, faster development and easier scaling of our services. Finally, "containers provide the solution to many deployment problems; in general, and especially when working with microservices. They also increase reliability due to their immutability." (Farcic, 2016). To summarize microservices are abstracting software problems while use of containers solve issues related to deployment scenarios of software updates.

One major benefit of using containers over Virtual Machines (VMs) is that containers have less overhead associated with server density as they are typically 1/10th to 1/00th the size of a similar application packaged within in a VM. A technology called Linux Containers (LXC) achieves this reduced server density. In LXC, a Linux Kernel is shared to manage the underlying Operation System (OS). If, for example, a physical server would be running 4 VMs, this would require 4 OSes in addition to a hypervisor. But with containers, the server could share the same OS, binaries, and libraries as shown in 'Figure 1: VMs and containers resource utilization comparison' below.

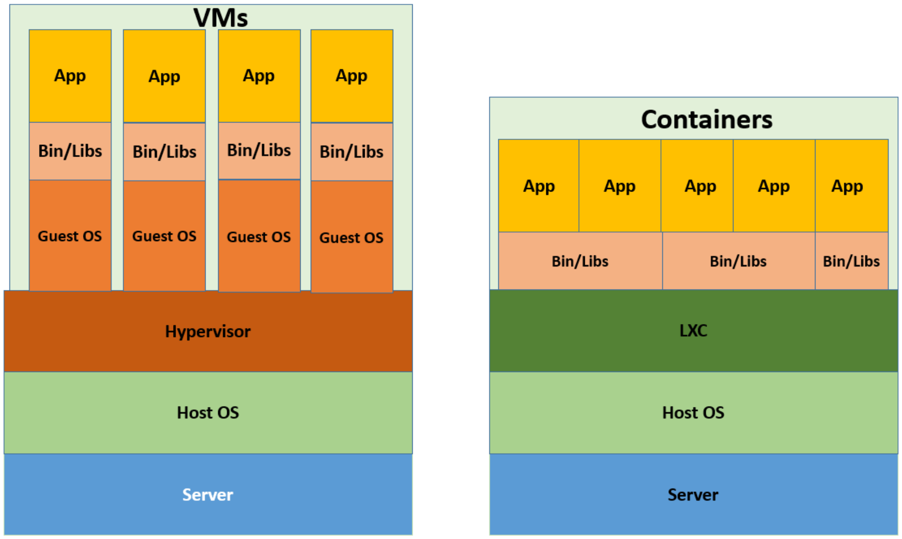


Figure 1: VMs and containers resource utilization comparison.

Though containers share the same Linux kernel, they are platform agnostic, which makes them portable to any environment. Other benefits of using containers include encapsulation and scalability. Encapsulation means to package everything needed by the application (e.g. dependencies, environment variables) within the container. The containers are also scalable which means that they can be dynamically be reduced or expanded in size. "Scalability can be applied to either one or multiple instances through centralized orchestration." (Bird, 2016). These powerful container concepts explain why there has been an immense growth in use container usage in DevOps environments in the past few years.

Furthermore, combining containers with microservices makes it possible to support micro-segmentation; each microservice is running in a separate runtime environment. This is the catalysator for container technologies like LXC, Rocket and especially Docker.

1.2. Introduction to Docker

Docker is a platform that combines applications and all their dependent components (e.g. libraries, tools) into an archive called a Docker Image. A Docker Image can be run on many different platforms like PCs, data centers, VMs or clouds. As a Docker Image compartmentalizes the application(s) and all its dependencies, it provides various benefits over bare metal like portability and scalability. These features, combined with reduced footprint that Docker Images have over Virtual Images, result in deployments of Docker Images in many different environments like data centers and cloud solutions.

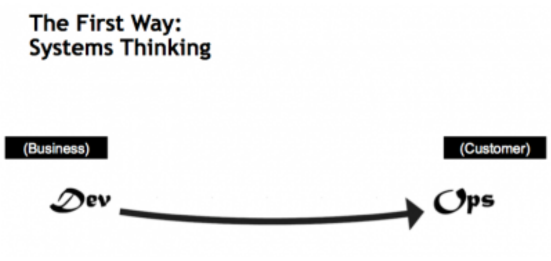
Started in 2013, Docker is an open -source project, and was released under the Apache 2.0 license, which efficiently allows for the creation, shipment, and running of contains within a single Linux instance. Docker was initiated as a project to build single-application Linux Containers (LXC) and introduced numerous improvements to LXC that made containers more flexible and portable to use than LXC, as well as some other older container technologies like FreeBSD Jails and Solaris Zones.

LXC, based on a user-space lightweight virtualization mechanism that implements namespaces and Control Groups (cgroups), manages resource isolation. Chenxi Wang, strategy officer at container security firm Twistlock, describes this isolation when he says, “Namespaces deal with resource isolation for a single process, while cgroups, originally developed by Google, manage resources for a group of processes” (Wang, 2016). Cgroups isolate and limit a given resource over a collection of processes to control performance or security.

Portability is probably amongst the biggest advantage of Docker over LXC (Wang, 2016). Portability allows the container to run on different OS distributions and hardware configurations without any changes to the image itself. This makes it very attractive to be used in multitude of different architectures suitable in cloud environments.

1.2.1. Role of Docker in DevOps

John Willis, an evangelist at Docker, explains the concepts of DevOps discussed above in something that he calls “The Three Ways of DevOps”. The “Three Ways of DevOps” are systems thinking, amplifying and shortening feedback loops, and continuous learning (Willis, 2015). All other DevOps patterns use these three principles. ‘Figure 2: The three ways of DevOps’ below visualizes these development patterns.



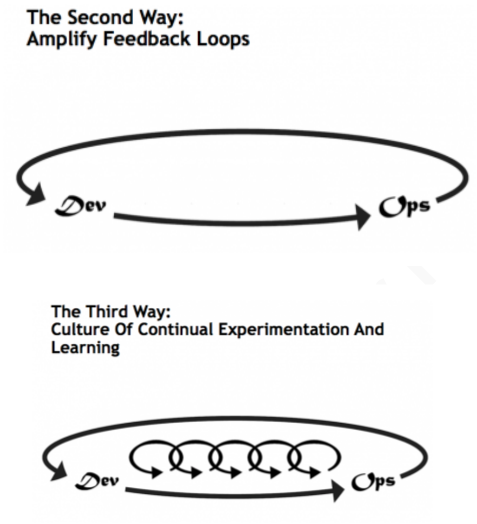


Figure 2: The Three Ways of DevOps

Figure 2 visualizes the continuous feedback loops in the third drawing. In this model, ‘the way of continual experimentation and learning’, development and operations teams adjust production environments on the fly based upon customer feedback. Through features like portability and micro-segmentation, Docker amplifies this third way of continual experimentation and learning, which leads to “faster innovation, higher quality and a feedback loop of continuous learning, advancing to a higher rate of success” (Willis, 2016). The fact that Docker has been embraced by large software powerhouses like Red Hat, IBM, Microsoft, Huawei, Google, and Cisco, who are also the top contributors to the Docker project ("Docker - Updated project statistics · GitHub," n.d.), indicates that Willis might be right when he states that Docker is a great adjunct to the third way of DevOps. The embracement of the software power houses has led to a quick adoption rate and to extensive investments being made in Docker. Jack Dougal, author at Banking.com, confirms this when stating that Docker has been included in the financial industry by firms like Goldman Sachs and Bank of America (Dougal, 2015).

1.3. The Faulty Software Distribution Pipeline

As many organizations are starting to integrate Docker into their CI and CD practices to help speed up system provisioning, reduce job time, and improve the overall infrastructure utilization, they are becoming more dependent on small, reusable components developed by many different developers and often distributed by infrastructures outside control of development. Because of these changes the risk of introducing vulnerabilities in the development cycle has increased manifold. The thread landscape is shifting because of these changes. The Notary project, recently introduced in Docker, is built upon this assumption that the software distribution pipeline can no longer be trusted.

1.4. Changed Security Lifecycle

One disadvantage/shortcoming of Docker is the impact of security on the software development cycle. As companies are adopting continuous deployment workflows and implementing microservices and embracing containers, security needs to adapt at this rate of change when there is no time to do pen testing or audits (Bird 2016). Figure 3 shows how in a traditional waterfall development cycle security is often part of the hardening phase, at the end of the release cycle just before putting the code base in production. Bird states that “Security must ‘shift left’ earlier into design and coding and into automated test cycles instead of waiting until the system is designed and built and then trying to fit some security checks just before release” (Bird, 2016). By shifting left, he means that security needs be integrated earlier in the development stage instead of close to production at the end. As the second picture shows, in a pure DevOps environment, security needs to be integrated from the design phase and not be implemented as an afterthought. In other words, security becomes integral part of the SDLC in a pure DevOps environment as can be seen in the second picture. This is aligned with the Third Way of DevOps as explained above, e.g. only when integrated with the development cycle through continuous security, code can secure be deployed in a DevOps world.

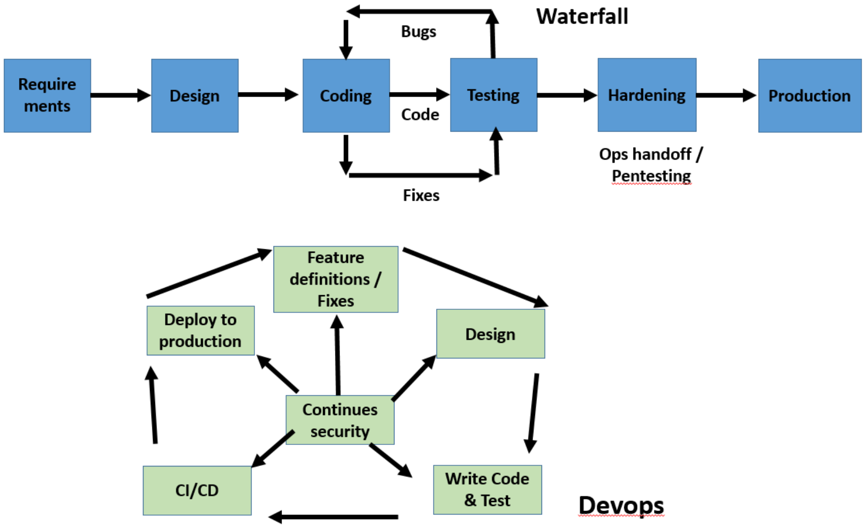


Figure 3: Waterfall versus DevOps development cycle.

1.5. Separation of Duties and other Critical Controls

One of the most difficult challenges in DevOps is Separations of Duties (SoD). Breaking down silos and sharing responsibilities between developers and operations seem to be in direct conflict with SoD (Bird,2016). In the continuous development model the developer cannot hand over code to the next phase as there are continuous adjustments being made. The developer becomes part of the end-product and closer interaction with customers is crucial to streamline efficiency. The role of the developer and the operator are merging. In an interview for ACMQueue, Amazon's CTO Werner Vogels explains why Amazon promotes this development model: 'You build it, you run it' (Vogels, 2006). Similar as to Amazon, John Allspaw, CTO at Etsy, explained why they promote at Flickr to giving developers access, or least limited access, to product environments (Allspaw, 2009). But this also raises concerns. Given developer access to managed systems, even giving them read-only access, raises questions and problems for regulators, compliance, infosec, and customers. To address such concerns, you will need to put strong compensating controls in place (Bird, 2016; Robinson, 2016). Such controls can only come from automation, e.g. security tools.

1.6. Are security tools ready for DevOps?

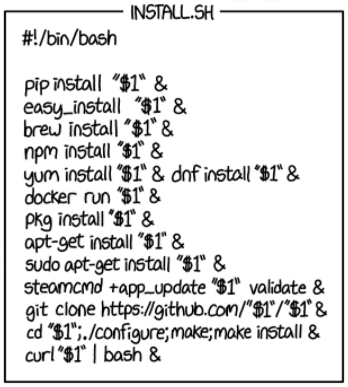
With continuous learning through experimentation, we have seen that the DevOps model not only changes the development phases but we have seen also a shift regarding responsibilities as developers become directly responsible for the end product. Use of security tools in this changed model is key to success. “CIS Critical Security Controls (CSCs) describe a set of specific actions designed to improve an organization’s ability to resist or recover from information security incidents” (“CIS critical security controls,” 2016). Usage of automated tools is an effective way to enforce policies associated with CSCs. Tools will help to continuously measure, test and validate the effectiveness of an organization’s current security measures. Tool usage is even more important in a DevOps environment where the approach to change management is reversed (e.g. optimize small and frequent changes). Robinson concludes in her paper ‘Continuous Security: Implementing the Critical Controls in a DevOps environment’ that, “we can expect increased maturity for new security tools developed for DevOps as the shift towards DevOps continuous” (Robinson, 2016). The question remains if security tools are currently on par with DevOps landscape.

2. Container integrity: Docker Notary

2.1. Who do you trust?

As applications become more dependent on external components, having secure software update systems becomes increasingly important. Diogo Monica, a security architect at Docker, argues that software developers and publishers should start to include in their risk analysis the possibility that considers the distribution infrastructure itself as being actively malicious. He explains, “They should start following best practices concerning role responsibility separation, offline storage of signing keys, and routine rotation of signing keys” (Monica, 2015). So basically, to securely deliver updates check and balances need to be put in place during the delivery phase itself. No longer can one assume that the content can be trusted blindly.

Diogo blames the easy of package installation being a root cause of the distribution infrastructure potentially becoming malicious. “More and more our infrastructures are depending on external sources of content like NPM and package managers such as RPM. The funny part is that these things are all being managed by thousands of developers that we don't know in infrastructures that are totally outside of our control while the number of package managers keeps on increasing.” (Monica (2015). Figure 4, a comic from xkcd.com, shows how a modern install script calls the many different package mangers to install packages from many different locations.



```
#!/bin/bash

pip install "$1" &
easy_install "$1" &
brew install "$1" &
npm install "$1" &
yum install "$1" & dnf install "$1" &
docker run "$1" &
pkg install "$1" &
apt-get install "$1" &
sudo apt-get install "$1" &
steamcmd +app_update "$1" validate &
git clone https://github.com/"$1"/"$1" &
cd "$1";./configure;make;make install &
curl "$1" | bash &
```

Figure 4: Universal install script from http:/xkcd.com/1654/

Another example that the distribution infrastructure might be tainted comes from a recent security analysis of OEM Updates by Duo Security which indicated that “all OEM vendors had at least one vulnerability that could allow for a man-in-the-middle (MITM) attacker to execute arbitrary code as SYSTEM.” (Camp, Czub, & Dadidov, n.d.). Whether it is through different package managers or through automatically applied updates in an OEM environment it is obvious there are many different attack vector on the distribution pipeline.

There are many known attacks on software update systems. From arbitrary software installation and mix-and-match attacks to fast-forward attacks. For an overview of many known attacks on these update systems, see ("tuf/SECURITY.md at develop · theupdateframework/tuf · GitHub," n.d).

Diogo explains in a Docker blog that HTTPS and GPG by itself are not sufficient to trust the content. GPG is not a framework, but a message format, in which one applies a signature to an application and then verifies the signature which leaves the system open to, for example, downgrade attacks. Such attacks mean, if there is a man in the middle, or someone has control over the actual cloud, the adversary can then serve the victim an old (vulnerable) version of the content as there is no revocation scheme.

2.2. The Update Framework (TUF)

Software update systems that do not authenticate updates have received increased scrutiny in recent years. Unfortunately, due to this attention, many of these systems have implemented simple authentication mechanisms that cannot survive key compromise (Docker Inc, 2016). The Update Framework (TUF) is a flexible, comprehensive security framework that is used for securing software update systems that mitigate such attacks. “TUF, allows both new and existing systems to benefit from a design that leverages responsibility separation, multi-signature trust, trust revocation, and low-risk roles” (Docker Inc, 2016). There are many different update systems in use today but TUF is different in the sense that it is built upon a specification and library that can be used universally to secure update systems.

2.3. Notary

To securely publish Docker Images with content that is verifiable, Docker introduced the Notary utility. “Notary is a Docker utility build upon the TUF framework for securely publishing and verifying content, distributed over any insecure network” (Monica, 2015). Notary has a few important objectives:

- Survivable key compromise
- Freshness guarantee
- Configurable trust thresholds
- Signing delegation
- Use of existing distribution
- Untrusted mirrors and transport

The TUF specification outlines these and other implementation directives ("tuf/SECURITY.md at develop · theupdateframework/tuf · GitHub," n.d.).

Notary implements various recommendations from the TUF framework. For example, through signed collections, it supports software to have relations where versions are dependent on other versions. With survivable key compromise and signing delegation, Notary allows for key delegation. Best practices would be to store the most crucial key, (GPG master key), offline. Other keys which are less sensitive and should have a short expiration could live in the cloud. Such keys could for example be keys that sign certain portions of the software development lifecycle.

Transparent key rotation is another feature that allows keys to be rotated at different intervals. In case the root key is the source key of trust, the administrator cannot rotate it without taking the system offline. By using trust delegation, the root key can be taken offline. New keys can be signed and send to the user. Adversaries cannot compromise the trust chain as the root key was offline. Trust delegation allows for key rotation multiple times a week/day. One could, for example, rotate CI keys every month.

2.4. Notary Threshold Signing

One of the advantages of Notary from a security assurance perspective is that it allows users to sign packages by multiple keys unlikely to GPG key signing where there is only one key. For example, a software package needs to be built and signed by CI system and then later the security team need to rubber stamp it. A second example could be where different types of the assurance process could get signatures, such as a unit test, integration test, security test, etc. Clients should be able to verify all keys being signed. Packages can get as many signatures as desired. This features also protects against non-technical attacks like subpoenas by nation states. For example, multiple keys could be hosted in different countries (e.g. Russia, China, US). So, US companies would need approval from a security team in China.

2.5. Deploying and testing Docker Notary

The Notary and Registry services have much different scaling and security requirements, so decoupling them has many benefits. Notary has both a server and client component. In the section below a sandbox will be set up to demonstrate trust operations locally without impacting production images. The sandbox will be used to test the Notary service and look at the various security tools for testing Docker images. To use Notary, the user must be familiar with the command-line environment (Gallagher, 2016). Note that this sandbox is just for development purposes. When moving from deployment to production, there are various considerations like high availability, databases and certificates to ensure security and scalability. See the online Docker documentation for how to run the Notary service in production (Docker Inc, 2016).

2.5.1. Setup of a Docker Content Trust Sandbox

In this section, the example shows various containers and how to setup a sandbox to demonstrate the functionality of Docker Notary. A container called Trustsandbox will be generated, which has the latest version of the Docker Engine with some preconfigured certificates. In the example the sandbox is used to test the Docker client to test various trust operations.

The registry server container is a local registry service where Docker images can be stored. The Notary server container is the service that does all the heavy-lifting of managing Trust. The Notary Signer service ensures that the keys are secure while the MySQL container has the database that stores all the trust information. Docker Hub has these components already built-in so one would not need those if working exclusively with the Docker Hub.

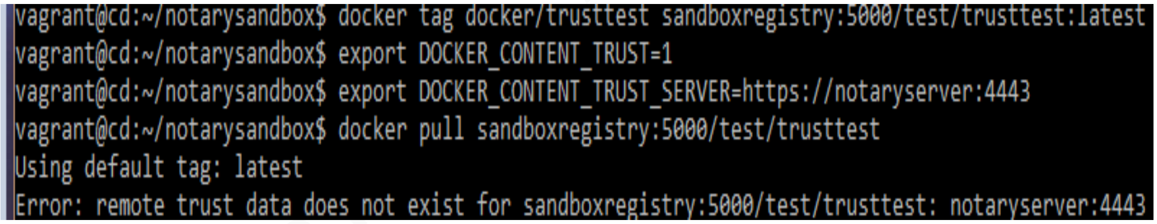
The commands below, with minor modifications, are obtained from Docker's website (Docker Inc., 2016). See Appendix Section A 'Prerequisites Docker Content Trust Sandbox' for prerequisites and Appendix Section B 'Setting up Docker Content Trust SandBox' for setting up the trust sandbox.

2.5.2. Testing Notary Trust Operations

When the content trust sandbox is up and running, various trust operations will be executed to demonstrate the Notary functionality. These operations are as follows:

```
# Test Trust Operations
# Download a Docker test image
$ docker pull docker/trusttest

# Tag it to be pushed to sandbox registry
```

```
$ docker tag docker/trusttest
sandboxregistry:5000/test/trusttest:latest

# Enable content trust
$ export DOCKER_CONTENT_TRUST=1

# Identify the trust server
$ export DOCKER_CONTENT_TRUST_SERVER=https://notaryserver:4443

# Pull the test image
$ docker pull sandboxregistry:5000/test/trusttest
```

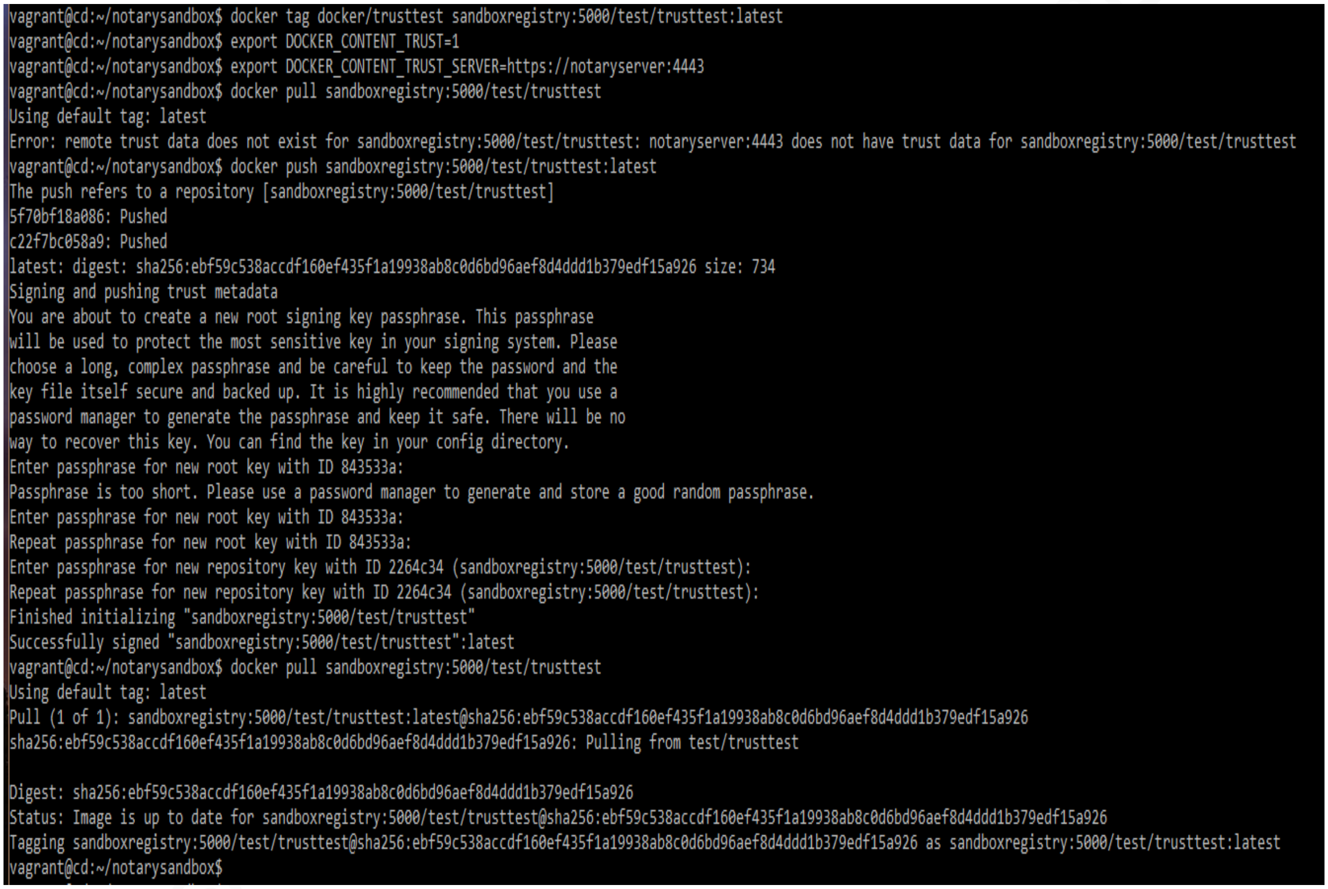


```
vagrant@cd:~/notarysandbox$ docker tag docker/trusttest sandboxregistry:5000/test/trusttest:latest
vagrant@cd:~/notarysandbox$ export DOCKER_CONTENT_TRUST=1
vagrant@cd:~/notarysandbox$ export DOCKER_CONTENT_TRUST_SERVER=https://notaryserver:4443
vagrant@cd:~/notarysandbox$ docker pull sandboxregistry:5000/test/trusttest
Using default tag: latest
Error: remote trust data does not exist for sandboxregistry:5000/test/trusttest: notaryserver:4443
```

Figure 5: Pull trusted Docker container from local Registry fails

You will get an error with the pull command above as the content does not exist in the sandbox Registry yet.

```
# Push and sign the trusted image
$ docker push sandboxregistry:5000/test/trusttest:latest

# Pull the pushed image
$ docker pull sandboxregistry:5000/test/trusttest
```



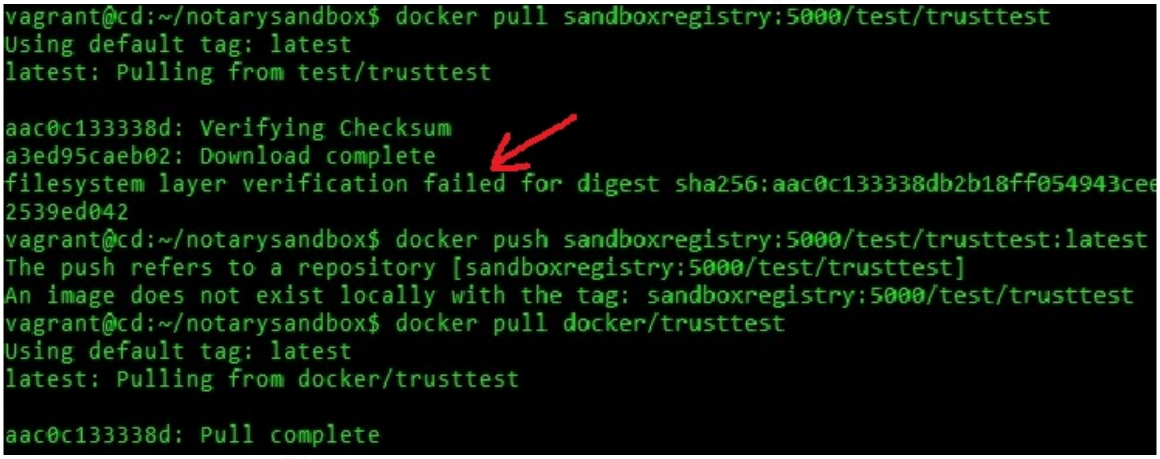
```
vagrant@cd:~/notarysandbox$ docker tag docker/trusttest sandboxregistry:5000/test/trusttest:latest
vagrant@cd:~/notarysandbox$ export DOCKER_CONTENT_TRUST=1
vagrant@cd:~/notarysandbox$ export DOCKER_CONTENT_TRUST_SERVER=https://notaryserver:4443
vagrant@cd:~/notarysandbox$ docker pull sandboxregistry:5000/test/trusttest
Using default tag: latest
Error: remote trust data does not exist for sandboxregistry:5000/test/trusttest: notaryserver:4443 does not have trust data for sandboxregistry:5000/test/trusttest
vagrant@cd:~/notarysandbox$ docker push sandboxregistry:5000/test/trusttest:latest
The push refers to a repository [sandboxregistry:5000/test/trusttest]
5f70bf18a086: Pushed
c22f7bc058a9: Pushed
latest: digest: sha256:ebf59c538accdf160ef435f1a19938ab8c0d6bd96aef8d4ddd1b379edf15a926 size: 734
Signing and pushing trust metadata
You are about to create a new root signing key passphrase. This passphrase
will be used to protect the most sensitive key in your signing system. Please
choose a long, complex passphrase and be careful to keep the password and the
key file itself secure and backed up. It is highly recommended that you use a
password manager to generate the passphrase and keep it safe. There will be no
way to recover this key. You can find the key in your config directory.
Enter passphrase for new root key with ID 843533a:
Passphrase is too short. Please use a password manager to generate and store a good random passphrase.
Enter passphrase for new root key with ID 843533a:
Repeat passphrase for new root key with ID 843533a:
Enter passphrase for new repository key with ID 2264c34 (sandboxregistry:5000/test/trusttest):
Repeat passphrase for new repository key with ID 2264c34 (sandboxregistry:5000/test/trusttest):
Finished initializing "sandboxregistry:5000/test/trusttest"
Successfully signed "sandboxregistry:5000/test/trusttest":latest
vagrant@cd:~/notarysandbox$ docker pull sandboxregistry:5000/test/trusttest
Using default tag: latest
Pull (1 of 1): sandboxregistry:5000/test/trusttest:latest@sha256:ebf59c538accdf160ef435f1a19938ab8c0d6bd96aef8d4ddd1b379edf15a926
sha256:ebf59c538accdf160ef435f1a19938ab8c0d6bd96aef8d4ddd1b379edf15a926: Pulling from test/trusttest

Digest: sha256:ebf59c538accdf160ef435f1a19938ab8c0d6bd96aef8d4ddd1b379edf15a926
Status: Image is up to date for sandboxregistry:5000/test/trusttest@sha256:ebf59c538accdf160ef435f1a19938ab8c0d6bd96aef8d4ddd1b379edf15a926
Tagging sandboxregistry:5000/test/trusttest@sha256:ebf59c538accdf160ef435f1a19938ab8c0d6bd96aef8d4ddd1b379edf15a926 as sandboxregistry:5000/test/trusttest:latest
vagrant@cd:~/notarysandbox$
```

Figure 6: Notary signing and pulling of the signed image

```
# Test with a malicious image
# Open terminal into sandboxregistry
$ docker exec -it sandboxregistry sh

# Change into registry storage
# cd /var/lib/registry/docker/registry/v2/blobs/sha256/aa/<SHA
_received_when_pushing_image>

#Add malicious data to one of the trusted layers
$ echo "Malicious data" > data

#Return to sandbox terminal and list the trusted image
$ docker images | grep trusttest

# Remove the trustiest:latest image
$ docker rmi -f a9539b34a6ab
```

```
#Pull the image again
$ docker pull sandboxregistry:5000/test/trusttest
```

Figure 7 shows that the pull operation did not complete because the trust system could not verify the image. The user will get an error similar as the one listed below. This error validates that the Notary works as expected.



```
vagrant@cd:~/notarysandbox$ docker pull sandboxregistry:5000/test/trusttest
Using default tag: latest
latest: Pulling from test/trusttest

aac0c133338d: Verifying Checksum
a3ed95caeb02: Download complete
filesystem layer verification failed for digest sha256:aac0c133338db2b18ff054943cee
2539ed042
vagrant@cd:~/notarysandbox$ docker push sandboxregistry:5000/test/trusttest:latest
The push refers to a repository [sandboxregistry:5000/test/trusttest]
An image does not exist locally with the tag: sandboxregistry:5000/test/trusttest
vagrant@cd:~/notarysandbox$ docker pull docker/trusttest
Using default tag: latest
latest: Pulling from docker/trusttest

aac0c133338d: Pull complete
```

Figure 7 Docker Pull fails on corrupt image

```
# Bring down services
$ docker-compose down -v
```

This section illustrates how Docker Notary can be used to implement various basic trust operations on Docker containers. By using Notary one can start securing the distribution infrastructure by simple operations as the ones above.

2.6. Notary Integration with 3rd Party Repositories

Section 2.5.2 shows how to use Notary with a private Docker Registry as a repository. Instead of using a private Docker Registry, the same also works with cloud repositories like Docker Hub as well with third party repositories like Nexus and Artifactory. See for example, JFrog’s Artifactory User Guide on how to setup Notary with Artifactory (JFrog, 2016).

2.6.1. Google Container Registry

In early 2015, Google introduced Google Container Registry for managing private Docker images. Its functions are described by the company as follows: “The Google service, which runs on Google's Cloud Platform, stores, shields, encrypts, and controls access to a customer's Docker containers, offering a higher level of security for containers than has been available in the past." (Google Inc, 2015) While the Google Registry has Docker V2 API registry support, it is not clear at the time of this writing if this includes Notary functionality as well. But it shows that containers cannot be trusted as is and a verification service is needed to secure the distribution pipeline.

3. Docker Security Scanning

3.1. Docker Registry in a CI/CD Environment

The Docker Notary service allows a user to assign trust to Docker containers as shown above. This section describes how to validate these Docker images during the CI/CD lifecycle, once trust has been assigned.

At Dockercon 2016, Cem Gürkök gave an overview of SalesForce architecture (see Figure 8) that uses Notary for signing and validation of Docker images in their CI/CD lifecycle.

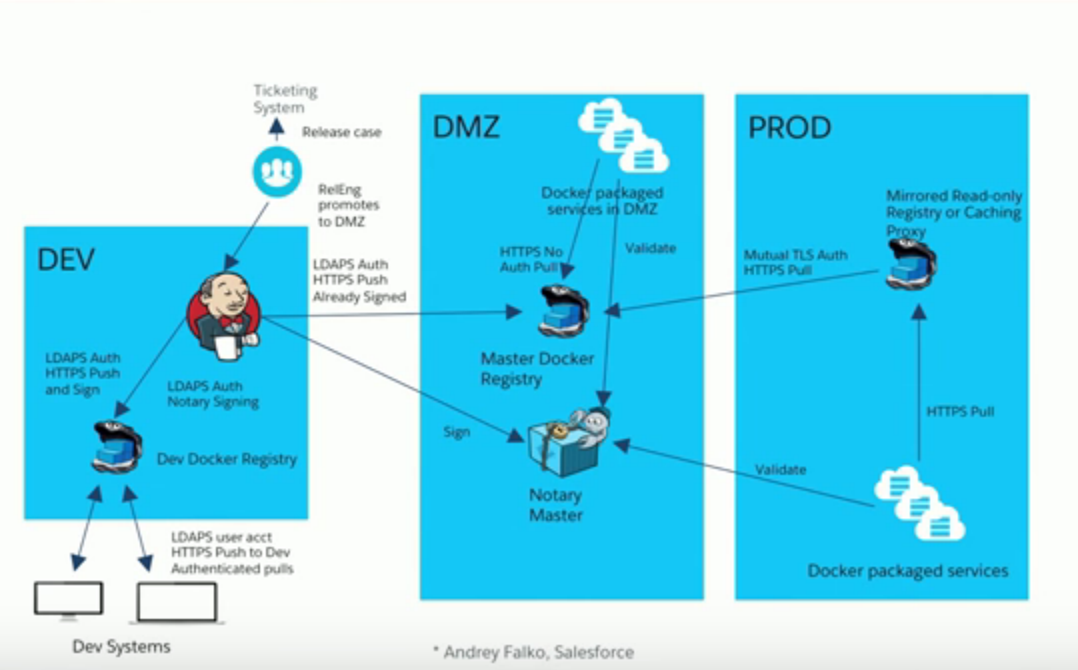
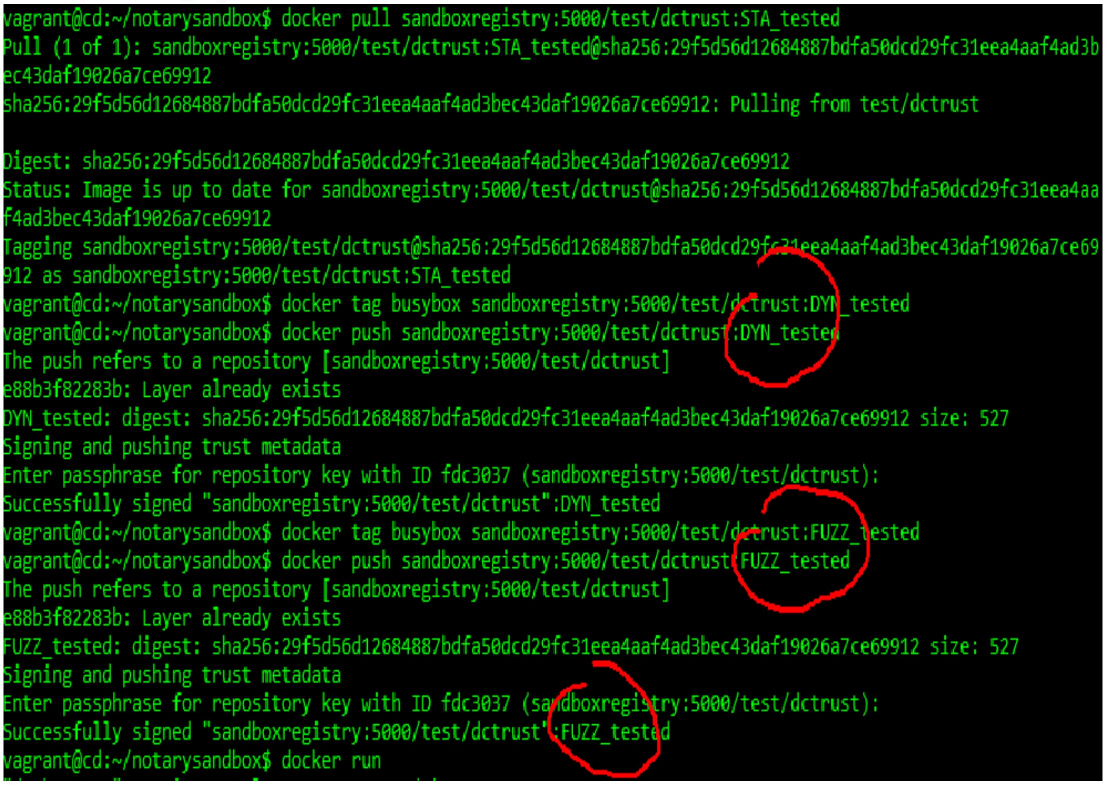

Ticketing System
Release case
RelEng promotes to DMZ
DEV
LDAPS Auth HTTPS Push Already Signed
LDAPS Auth HTTPS Push and Sign
LDAPS Auth Notary Signing
Dev Docker Registry
LDAPS user acct HTTPS Push to Dev Authenticated pulls
Dev Systems
DMZ
Docker packaged services in DMZ
HTTPS No Auth Pull
Validate
Master Docker Registry
Sign
Notary Master
PROD
Mirrored Read-only Registry or Caching Proxy
Mutual TLS Auth HTTPS Pull
HTTPS Pull
Validate
Docker packaged services
* Andrey Falko, Salesforce

Figure 8 SalesForce usage of Notary.(Gurkok & Falko, 2016)

Gürkök categorizes the various aspects of Docker security into a few categories:

- Hardening
- Patching
- Monitoring

Hardening includes both host as well as Container hardening. The NCC Group published an excellent white paper 'Understanding and hardening Linux containers' on different aspects of Container hardening (Grattafiori, 2016). Tools like Docker Bench are available to test many best practices around Docker containers in production. Vulnerability management uses image scans to validate the OS, application source code and their dependent libraries and network scans for traditional vulnerability scanning (discovery and exposed services). Furthermore, there are of course manual, and automated source code audits. Each of these categories theoretically could mean running a Security Tool and the Notary client signing/tagging the Docker image upon successful verification. Figure 9 shows an example of Notary client signing a Docker image with different keys.



```
vagrant@cd:~/notarysandbox$ docker pull sandboxregistry:5000/test/dctrust:STA_tested
Pull (1 of 1): sandboxregistry:5000/test/dctrust:STA_tested@sha256:29f5d56d12684887bdfa50dcd29fc31eea4aaf4ad3b
ec43daf19026a7ce69912
sha256:29f5d56d12684887bdfa50dcd29fc31eea4aaf4ad3bec43daf19026a7ce69912: Pulling from test/dctrust

Digest: sha256:29f5d56d12684887bdfa50dcd29fc31eea4aaf4ad3bec43daf19026a7ce69912
Status: Image is up to date for sandboxregistry:5000/test/dctrust@sha256:29f5d56d12684887bdfa50dcd29fc31eea4aa
f4ad3bec43daf19026a7ce69912
Tagging sandboxregistry:5000/test/dctrust@sha256:29f5d56d12684887bdfa50dcd29fc31eea4aaf4ad3bec43daf19026a7ce69
912 as sandboxregistry:5000/test/dctrust:STA_tested
vagrant@cd:~/notarysandbox$ docker tag busybox sandboxregistry:5000/test/dctrust:DYN_tested
vagrant@cd:~/notarysandbox$ docker push sandboxregistry:5000/test/dctrust:DYN_tested
The push refers to a repository [sandboxregistry:5000/test/dctrust]
e88b3f82283b: Layer already exists
DYN_tested: digest: sha256:29f5d56d12684887bdfa50dcd29fc31eea4aaf4ad3bec43daf19026a7ce69912 size: 527
Signing and pushing trust metadata
Enter passphrase for repository key with ID fdc3037 (sandboxregistry:5000/test/dctrust):
Successfully signed "sandboxregistry:5000/test/dctrust":DYN_tested
vagrant@cd:~/notarysandbox$ docker tag busybox sandboxregistry:5000/test/dctrust:FUZZ_tested
vagrant@cd:~/notarysandbox$ docker push sandboxregistry:5000/test/dctrust:FUZZ_tested
The push refers to a repository [sandboxregistry:5000/test/dctrust]
e88b3f82283b: Layer already exists
FUZZ_tested: digest: sha256:29f5d56d12684887bdfa50dcd29fc31eea4aaf4ad3bec43daf19026a7ce69912 size: 527
Signing and pushing trust metadata
Enter passphrase for repository key with ID fdc3037 (sandboxregistry:5000/test/dctrust):
Successfully signed "sandboxregistry:5000/test/dctrust":FUZZ_tested
vagrant@cd:~/notarysandbox$ docker run
```

Figure 9: Signing Docker image after different Security Testing Phases (Static, Dynamic, and Fuzz testing)

This shows that Docker Notary can be used to assign and validate trust by different signatures as the Docker Image moves to different stages in the CD/CI pipeline. For example, after successfully running a web application scan with a tool like Burp Suite, the Docker Image could be signed to indicate it successfully completed the dynamic test phase.

4. Docker Container Testing as a Service (DCTaaS)

This chapter will explore externalizing some of the security testing of Docker Containers. Stephen DeVries, CTO of Continuum Security, states that security tests are often not included as part of the quality controls. "Automated unit, integration and acceptance tests are essential quality controls in running a reliable continuous integration or continuous delivery pipeline. Too often, security tests are left out of this process because of the erroneous belief that security testing is solely the domain of security experts." (DeVries, 2015). This is important as the traditional separation of duties no longer apply in a CD/CI environment as discussed earlier.

4.1. Security Testing in Different CI/CD Stages

In the continual experimentation and learning model, described earlier as the Third Way, and often referred to as the automated deployment pipeline, it is important to include different security tests and the results. As discussed above this could mean assigning a signature by Docker Notary after Burp Suite successfully ran. Hooks need to build into the automated process to validate these security tests. "The automated deployment pipeline provides a mechanism which requires that defenses like static analysis, web app scanning and code review are executed before putting software in production" (Cole & Terala, 2015).

Jenkins, Hudson, and most other popular deployment tools provide support via plugins both for static analysis as well as for requiring code reviews as part of the pipeline (Robinson, 2016). Humble & Farley explain, "These acceptance tests should be designed to complete quickly and can be run before deploying new code to the integration/staging environment as part of the full commit stage" (Humble & Farley, 2011). In integration/staging, vulnerability and application security scans, as well as other security tests, can then be run alongside other acceptance tests (De Vries, 2015). DeVries distinguishes between quick inline acceptance tests who should be acting as a gate to code submissions and longevity tests, which could be run in parallel before production deployment.

Keeping the production deployment separate from the infrastructure that pushes code to non-production environments like load or staging can help to provide the access needed for automation and debugging while still enforcing separation of duties and restricted access to the production environment (Smith, 2014). To summarize security tests will need to build into the pipeline some of which can be run inline and some of the m separately on non-production environments.

4.1.1. Notary Trust Delegation

To externalize some of these security tests it becomes important to delegate some of the security test validation. Docker supports targets/releases delegation as an approved source of a trusted image tag. Using trust delegation, it allows one to collaborate with different publishers without sharing repository keys. A collaborator can keep his or her delegation key private (Docker, 2016). This could allow a central or external entity for example to sign content after static analysis has been successfully completed.

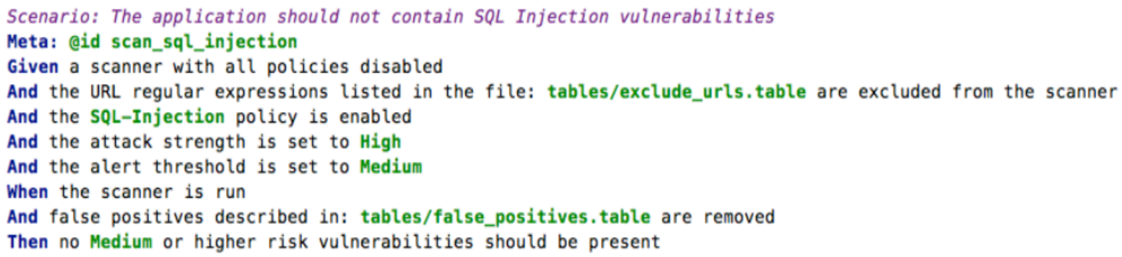
4.1.2. Categorizing Security Testing

To implement the Security Testing of Docker Images, it is important to group different tests. There are many ways of classifying Security Testing. One way as described by DeVries, which extends itself to work in CI/CD environment, is the following:

- Functional Security Tests
- Specific non-functional tests against known weaknesses
- Security Scanning of application and infrastructure
- Security Testing Application Logic

Some of these categories require an additional step when integrating into a CI/CD environment, to identify clearly passing and failing criteria. Defining these benchmarks is with an automated scanning tool a bit more complicated as it could involve False Positives (FPs), which in a manual process involves investigating and removing FPs. To automate this process one can wrap the security operation (FPs) in a test, like the open source security testing framework BDD-Security does, that uses Behavior Driven Development concepts (De Vries, 2015).

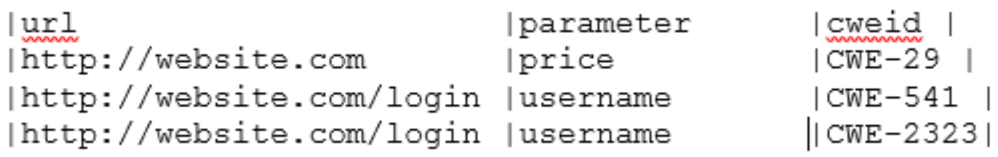
The following BDD-Security sample performs an automated scan for SQL injection vulnerabilities using the following test:



```
Scenario: The application should not contain SQL Injection vulnerabilities
Meta: @id scan_sql_injection
Given a scanner with all policies disabled
And the URL regular expressions listed in the file: tables/exclude_urls.table are excluded from the scanner
And the SQL-Injection policy is enabled
And the attack strength is set to High
And the alert threshold is set to Medium
When the scanner is run
And false positives described in: tables/false_positives.table are removed
Then no Medium or higher risk vulnerabilities should be present
```

Figure 10: Sample BDD-Security test scenario

This example stores the false positives in the tables/false_positive.table. Content could include:



```
|url                         |parameter       |cweid |
|http://website.com          |price           |CWE-29 |
|http://website.com/login    |username        |CWE-541 |
|http://website.com/login    |username       ||CWE-2323|
```

Figure 11: BDD-Security, example of tables/false_posible.table

4.2. Security Software Testing as a Service (SSTaaS)

Due to a wide range of applications of Cloud Computing, Software Security Testing as a Service has become very popular in the current era of the computing (Virdi, Kalyan, & Kaur, 2015). Virdi, Kalyan and Kaur conclude in their paper "Software Testing as a Service using Cloud Computing" that, while STaaS reduces effort and development costs of software development, there are many challenges, including lack of overall system testing, as subject matter experts in a particular field perform the testing.

Besides an explosion in STaaS services, there is uprising related to security testing services as well. For example, Microsoft recently released a new "cloud-based service for developers that will allow them to test application binaries for vulnerabilities before deploying." (Microsoft Corporation, 2016). This cloud-based fuzz testing, called project Springfield, is an example of how to use externalized SSTaaS to test different aspects of the security testing.

There are different aspects of security testing that can be externalized, outsourced, or both to 3rd parties in the form of services. Using the security testing categorization described above, security scanning of application and infrastructure is probably an effective example that lends itself for Security Testing as a Service as it requires limited knowledge of the application, business logic, or both. The decision regarding which aspects of security testing to outsource is business-related and beyond the scope of this paper.

4.3. Open-source Docker Tools

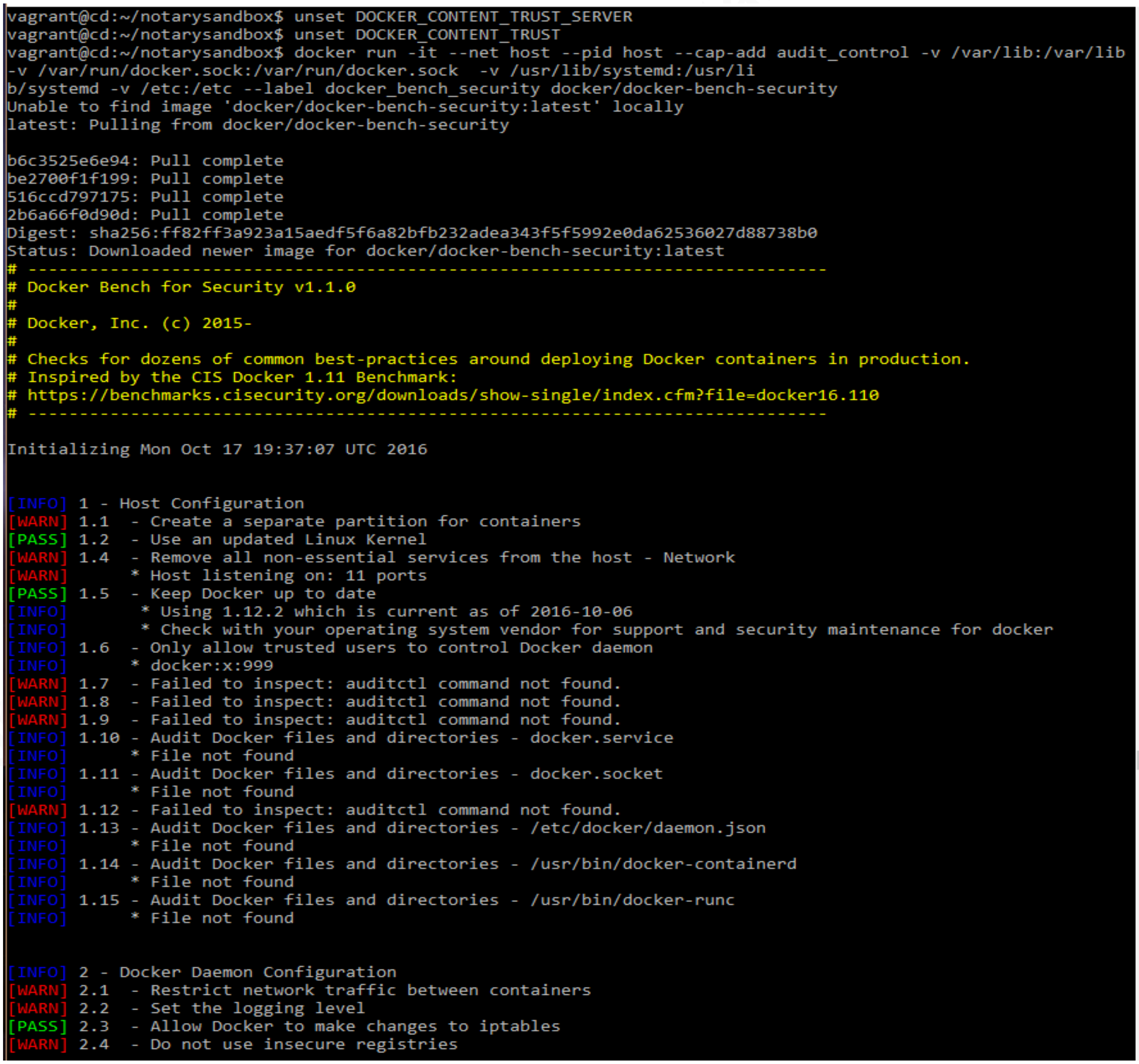
Many open-source and commercial security tools are being developed because of the explosive growth of Docker. Below includes an in-depth look at some of them:

4.3.1. Docker Bench for Security

Docker Bench is an open- source tool that validates configuration based upon CIS benchmark recommendations (Center for Internet Security, 2015). It can be utilized to scan Docker environments as well as start the host level and inspect all the aspects of the host. Other features include testing the Docker daemon and its configuration, validating the containers running on the Docker host, and reviewing the Docker security operations. The tool also can give recommendations across the board of a threat or concern (Gallagher, 2016).

To run Docker Bench, execute the following on the Docker host:

```
$ docker run -it --net host --pid host --cap-add audit_control \
-v /var/lib:/var/lib \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /usr/lib/systemd:/usr/lib/systemd \
-v /etc:/etc --label docker_bench_security \
docker/docker-bench-security
```



```
vagrant@cd:~/notarysandbox$ unset DOCKER_CONTENT_TRUST_SERVER
vagrant@cd:~/notarysandbox$ unset DOCKER_CONTENT_TRUST
vagrant@cd:~/notarysandbox$ docker run -it --net host --pid host --cap-add audit_control -v /var/lib:/var/lib
-v /var/run/docker.sock:/var/run/docker.sock  -v /usr/lib/systemd:/usr/li
b/systemd -v /etc:/etc --label docker_bench_security docker/docker-bench-security
Unable to find image 'docker/docker-bench-security:latest' locally
latest: Pulling from docker/docker-bench-security

b6c3525e6e94: Pull complete
be2700f1f199: Pull complete
516ccd797175: Pull complete
2b6a66f0d90d: Pull complete
Digest: sha256:ff82ff3a923a15aedf5f6a82bfb232adea343f5f5992e0da62536027d88738b0
Status: Downloaded newer image for docker/docker-bench-security:latest
# ------------------------------------------------------------------------------
# Docker Bench for Security v1.1.0
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker containers in production.
# Inspired by the CIS Docker 1.11 Benchmark:
# https://benchmarks.cisecurity.org/downloads/show-single/index.cfm?file=docker16.110
# ------------------------------------------------------------------------------

Initializing Mon Oct 17 19:37:07 UTC 2016


[INFO] 1 - Host Configuration
[WARN] 1.1  - Create a separate partition for containers
[PASS] 1.2  - Use an updated Linux Kernel
[WARN] 1.4  - Remove all non-essential services from the host - Network
[WARN]      * Host listening on: 11 ports
[PASS] 1.5  - Keep Docker up to date
[INFO]       * Using 1.12.2 which is current as of 2016-10-06
[INFO]       * Check with your operating system vendor for support and security maintenance for docker
[INFO] 1.6  - Only allow trusted users to control Docker daemon
[INFO]      * docker:x:999
[WARN] 1.7  - Failed to inspect: auditctl command not found.
[WARN] 1.8  - Failed to inspect: auditctl command not found.
[WARN] 1.9  - Failed to inspect: auditctl command not found.
[INFO] 1.10 - Audit Docker files and directories - docker.service
[INFO]      * File not found
[INFO] 1.11 - Audit Docker files and directories - docker.socket
[INFO]      * File not found
[WARN] 1.12 - Failed to inspect: auditctl command not found.
[INFO] 1.13 - Audit Docker files and directories - /etc/docker/daemon.json
[INFO]      * File not found
[INFO] 1.14 - Audit Docker files and directories - /usr/bin/docker-containerd
[INFO]      * File not found
[INFO] 1.15 - Audit Docker files and directories - /usr/bin/docker-runc
[INFO]      * File not found


[INFO] 2 - Docker Daemon Configuration
[WARN] 2.1  - Restrict network traffic between containers
[WARN] 2.2  - Set the logging level
[PASS] 2.3  - Allow Docker to make changes to iptables
[WARN] 2.4  - Do not use insecure registries
```

Figure 12: Example of Docker Bench execution

4.3.2. OpenSCAP Container Compliance

OpenSCAP is a tool that can assess vulnerabilities (CVEs) or security compliance (CCEs) of running Docker containers or cold Docker images based on the same philosophy as the parent OpenSCAP project which supports CVE scan, multiple report formats and customized policies (Red Hat Corporation, 2016). Below are some samples of how to start the different type of OpenSCAP scans of Docker Images.

```
# Install oscap-docker
wget https://raw.githubusercontent.com/OpenSCAP/container-
compliance/master/oscap-docker  && chmod 755 oscap-docker

# Offline compliance scan
./oscap-docker image docker.io/richxsl/rhel6.2
xccdf eval --profile xccdf_org.ssgproject.content_profile_rht-ccp \
/usr/share/xml/scap/ssg/content/ssg-rhel6
-ds.xml

#Offline vulnerability scan
./oscap-docker image-cve docker.io/richxsl/rhel6.2 --results
/var/www/html/image-oval.xml --report /var/www/html/
image-rhel62.html

#Online compliance scan
./oscap-docker container-cve docker.io/richxsl/rhel6.2 --results
/var/www/html/container-oval.xml --report /var/
www/html/container
-rhel62.htm
```

The examples above show that OpenSCAP project for Docker is similar to the well-known generic OpenSCAP project and hence has a small learning curve to start scanning Docker Images.

4.3.3. Docker Security Scanning (aka Project Nautilus)

Docker Security Scanning is Docker's image scanning capability. It delivers secure content by providing deep insights into Docker images along with a security overview of its components. The tool has been in use since the end of last year by various repositories to distribute updated containers signed with Content Trust. Security Scanning is available in Docker cloud and can be integrated to scan an image tag every time one pushes. (Docker, Inc, n.d.).

4.4. Commercial solutions

4.4.1. Banyanops

Banyanops received quite a bit of publicity after it produced a report last year stating that more than 30% of images in the official Docker Hub repositories are highly susceptible to a variety of security attacks (e.g. Shellshock, Heartbleed, Poodle, etc.) (Gummaraju, Desikan, & Turner, 2015). The company used an open- source component, Banyan Collector, a framework for static analysis of Docker container images, and a service called Banyan Insights to produce the data.

4.4.2. AquaSec

The Aqua Container Security Platform delivers an advanced security solution for containerized environments, supporting Docker on both Windows and Linux. The solution is available for on-premise deployment or on Azure, AWS, and Google clouds. The platform provides development-to-production container lifecycle protection by combining smart default security profiles, behavioral analytics, and in-house research to create a comprehensive security and compliance report. The company received in October 2016 a significant funding, led by Microsoft Ventures.

4.4.3. TwistLock

TwistLock scans container images in registries, on workstations like developer systems, and or on production servers. The tool detects and reports vulnerabilities in the Linux distribution layer, app frameworks, and in customer application packages. Users can configure TwistLock with open-source threat feeds as well as commercial threat feeds. It also offers access control to actions based on users and groups and a Runtime defense that   monitors and subsequently responds to malicious actions.

4.4.4. IBM BlueMix – Vulnerability Advisor

IBM has the Vulnerability Advisor (VA) for inspecting Docker images. Its capabilities include: the automatic inventory of packages installed on the Docker image, which then compares them against vulnerability databases, a report of packages that have vulnerabilities, and the scanning of Ubuntu security notices. Currently, VA is only available in the BlueMix- hosted VMs.

4.5. Integration of SSTaaS and Notary

The tools described above provide a sample of numerous tools available for security testing of Docker Images. Alfresco's website provides a decent overview of Docker tools available for just Docker auditing and vulnerability assessment (Alfresco, 2015). Some of these tools are still very new. It is hard to keep up as Docker is growing rapidly and new versions are being released frequently. As there is currently no complete third-party Docker Security Testing solution available, security architects will need to integrate any partial SSTaaS results into the overall Docker Security Assurance lifecycle through Notary and trust delegation to provide a complete security testing solution.

Conclusion

Many tools related to security testing Docker containers are new with just months or weeks since release. As Docker is growing rapidly and releasing versions with many new features (including security improvements) almost weekly, it is a challenging race to keep any of these tools up to date. An initial goal should be to address audit and vulnerability assessments of the container ecosystem regardless of whether it is a development, staging, test or production environment. But it should not stop there. Beyond that, security practitioners should set an overall goal to have security tests integrated as part of the CD process and block software delivery if tests fail. Use of Docker Notary to accomplish this integration seem to be the correct approach. Complete automation is unfortunately not realistic for many at this point with the state of Docker development. In the interim, tests can be run parallel to the build with supervision by the security team. It is then the responsibility of the security team to manually block delivery if test failures indicate the presence of unacceptable risk.

References

Alfresco. (2015, December 3). Docker Security Tools: Audit and Vulnerability Assessment | Alfresco DevOps Blog [Web log post]. Retrieved from https://www.alfresco.com/blogs/devops/2015/12/03/docker-security-tools-audit-and-vulnerability-assessment/

Allspaw, J. (2009, June). *10+ deploys per day: dev and ops cooperation at Flickr*. Paper presented at Velocity, San Jose, CA. Retrieved from http://www.kitchensoap.com/2009/06/23/slides-for-velocity-talk-2009/

Beda, J. (n.d.). *Containers at scale*. Paper presented at GlueCon 2014, Denver, Colorado.

Bird, J. (2016). *DevOpsSec, Securing Software through Continuous Delivery*.

Brigham, R., & Liguori, C. (n.d.). *AWS re:Invent 2015 | (DVO202) DevOps at Amazon: A Look at Our Tools and Processes* [Video file]. Retrieved from https://www.youtube.com/watch?v=esEFaY0FDKc

Camp, D., Czub, C., & Dadidov, M. (n.d.). Out of box exploitation - A security analysis of OEM updaters. Retrieved from https://duo.com/assets/pdf/out-of-box-exploitation_oem-updaters.pdf

Cappos, J. (2008). *A look in the mirror: attacks on package manager* (Doctoral dissertation, University of Arizona). Retrieved from https://isis.poly.edu/~jcappos/papers/cappos_mirror_ccs_08.pdf

Cappos, J., Samuel, J., Baker, S., & Hartman, J. H. (2008). A look in the mirror. *Proceedings of the 15th ACM conference on Computer and communications security - CCS '08*. doi:10.1145/1455770.1455841

Center for Internet Security. (2015, April 22). CIS Docker 1.6 benchmark. Retrieved from https://benchmarks.cisecurity.org/tools2/docker/cis_docker_1.6_benchmark_v1.0.0.pdf

Center for Internet Security. (2016, April 12). CIS Docker 1.11.0 Benchmark. Retrieved from https://benchmarks.cisecurity.org/tools2/docker/CIS_Docker_1.11.0_Benchmark_v1.0.0.pdf

CIS Critical Security Controls. (n.d.). Retrieved October 1, 2016, from https://www.cisecurity.org/critical-controls.cfm

Cole, E., & Tarala, J. (2016). *Implementing & Auditing the Critical Security Controls – In Depth* . The SANS Institute.

Conjur, inc. (2016, June 18). Securing Docker With Secrets and Dynamic Traffic Authorization [Web log post]. Retrieved from https://blog.conjur.net/securing-docker-with-secrets-and-dynamic-traffic-authorization

Docker - Updated project statistics · GitHub. (n.d.). Retrieved October 2, 2016, from https://gist.github.com/icecrime/18d72202f4569a0cab1ee60f7583425f

Docker Inc. (2016). Play in a content trust sandbox. Retrieved October 15, 2016, from http://54.71.194.30:4111/engine/security/trust/trust_sandbox/

Docker Inc. (2016, October). Content trust in Docker. Retrieved October 18, 2016, from http://docs.master.dockerproject.org/engine/security/trust/content_trust/

Dougal, J. (2015, December 19). The Container Factor | Banking.com [Web log post]. Retrieved from http://www.banking.com/2015/12/29/the-container-factor/#.WAaCuK33CUl

Farcic, V. (2016). *The DevOps 2.0 Toolkit: Automating the Continuous Deployment Pipeline with Containerized Microservices*. CreateSpace Independent Publishing Platform.

Gallagher, S. (2016). *Securing Docker: Learn how to secure your Docker environment and keep your environments secure irrespective of the threats out there*.

Google Inc. (2015, January 1). Google Cloud Platform Blog: Secure hosting of private Docker repositories in Google Cloud Platform [Web log post]. Retrieved from https://cloudplatform.googleblog.com/2015/01/secure-hosting-of-private-Docker-repositories-in-Google-Cloud-Platform.html

Grattafiori, A. (2016). Understanding and hardening Linux containers. Retrieved from https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/2016/april/ncc_group_understanding_hardening_linux_containers-1-1pdf/

Gummaraju, J., Desikan, T., & Turner, Y. (2015, May 1). BanyanOps analyzing DockerHub. Retrieved from https://banyanops.com/pdf/BanyanOps-AnalyzingDockerHub-WhitePaper.pdf

Gurkok, C., & Falko, A. (2016, June 27). Salesforce usage of Notary [Digital image]. Retrieved November 3, 2016, from http://image.slidesharecdn.com/dockercon-2016-cg-securingthecontainerpipelineatsalesforce-sf-6-23-2016-public-160627171137/95/securing-the-container-pipeline-at-salesforce-by-cem-gurkok-14-638.jpg?cb=1467047655

Humble, J., & Farley, D. (2011). *Continuous delivery*. Upper Saddle River, NJ: Addison-Wesley.

JFrog. (2016, October 5). Working with Docker Content Trust: JFrog Artifactory User Guide [Web log post]. Retrieved from https://www.jfrog.com/confluence/display/RTF/Working+with+Docker+Content+Trust

Microsoft Corporation. (2016, September 26). Microsoft previews Project Springfield, a cloud-based bug detector - Next at Microsoft [Web log post]. Retrieved from https://blogs.microsoft.com/next/2016/09/26/microsoft-previews-project-springfield-cloud-based-bug-detector/

Monica, D. (2015, August 1). Introducing Docker Content Trust - Docker Blog. Retrieved from https://blog.docker.com/2015/08/content-trust-docker-1-8/

Red Hat Corporation. (2016, June). *A security state of mind: container security*. Paper presented at Usenix, Austin, Texas.

Robinson, A. (2016, December 20). Continuous Security: Implementing the Critical Controls in a DevOps Environment. Retrieved from https://www.sans.org/reading-room/whitepapers/critical/continuous-security-implementing-critical-controls-devops-environment-36552

tuf/SECURITY.md at develop · theupdateframework/tuf · GitHub. (n.d.). Retrieved October 4, 2016, from https://github.com/theupdateframework/tuf/blob/develop/SECURITY.md

tuf/tuf-spec.txt at develop · theupdateframework/tuf · GitHub. (n.d.). Retrieved from https://github.com/theupdateframework/tuf/blob/develop/docs/tuf-spec.txt

Virdi, K., Kalyan, R., & Kaur, N. (2015). Software testing as a service (STaaS) using cloud computing. *IJECS*, *4*(1), 7. Retrieved from http://docplayer.net/12543741-Software-testing-as-a-service-staas-using-cloud-computing.html

Vliet, J, & Paganelli, F. (2011). *Programming Amazon EC2*. Sebastopol, CA: O'Reilly Media.

Vogels, W. (2006, June 30). Learning from the Amazon technology platform. *Association for Computing Machinery (ACM)*, *4*(4), 8. Retrieved from http://delivery.acm.org/10.1145/1150000/1142065/p14-o_hanlon.pdf

Wang, C. (2016, May 6). Containers 101: Linux containers and Docker explained | InfoWorld. Retrieved from http://www.infoworld.com/article/3072929/linux/containers-101-linux-containers-and-docker-explained.html

Willis, J. (2015, July 31). Docker and the three ways of devOps. Retrieved from https://www.docker.com/sites/default/files/WP_Docker%20and%20the%203%20ways%20devops_07.31.2015%20(1).pdf

Appendix section A
Prerequisites Docker Content Trust Sandbox

To install the Docker Sandbox, docker-compose, docker-engine and docker need to be installed on the host. The following snippets help to install these prerequisites. It uses the DevOps2 toolkit to install the Docker dependencies.

```
# Kali 20616.1 Prerequisites
# Install Vagrant
wget
https://releases.hashicorp.com/vagrant/1.8.6/vagrant_1.8.6_x86_64.deb
dpkg -i ./ vagrant_1.8.6_x86_64.deb
vagrant plugin install vagrant-cachier
# Install VirtualBox
wget http://download.virtualbox.org/virtualbox/5.1.6/virtualbox-
5.1_5.1.6-110634~Ubuntu~trusty_amd64.deb
dpkg -i ./ http://download.virtualbox.org/virtualbox/5.1.6/virtualbox-
5.1_5.1.6-110634~Ubuntu~trusty_amd64.deb

REM Windows10 Prerequisites
REM Install Chocolatey
REM Install Vagrant
cinst vagrant -yf --AllowEmtpyChecksums
vagrant plugin install vagrant-cachier
REM Install GIT
cinst git -yf

# Clone DevOps2 toolkit
# Prereqs: This requires Vagrant and Git to be available on the host
git clone https://github.com/vfarcic/ms-lifecycle.git
cd ms-lifecycle
# Add the following line to the Dockerfile:
d.vm.box_version="20160801.0.0"
# Start the boxes
vagrant up cd prod
# SSH into CD box
vagrant ssh cd
```

Appendix section B
Setting up Docker Content Trust Sandbox

#Add an entry for the notaryserver to /etc/hosts:

```
$ sudo sh -c 'echo "127.0.0.1 notaryserver" >> /etc/hosts'
```

#Add an entry for the sandboxregistry to /etc/hosts

```
$ sudo sh -c 'echo "127.0.0.1 sandboxregistry" >> /etc/hosts'
```

#Make the notarysandbox/notarytest directory structure

```
$ mkdir notarysandbox && cd notarysandbox && mkdir notarytest && cd
notarytest
```

#Create a Dockerfile with the following content:

```
      FROM debian:jessie

      ADD https://master.dockerproject.org/linux/amd64/docker
      /usr/bin/docker
      RUN chmod +x /usr/bin/docker \
        && apt-get update \
        && apt-get install -y \
        tree \
        vim \
        git \
        ca-certificates \
        --no-install-recommends

      WORKDIR /root
      RUN git clone -b trust-sandbox
      https://github.com/docker/notary.git
      RUN cp /root/notary/fixtures/root-ca.crt /usr/local/share/ca-
      certificates/root-ca.crt
      RUN update-ca-certificates

      ENTRYPOINT ["bash"]
```

# Build the test container

```
$ docker build -t notarysandbox .

# Change to back to the root of your Notarysandbox directory
$ cd ../../notarysandbox

# Clone the Notary project
$ git clone -b trust-sandbox https://github.com/docker/notary.git

# Clone the distribution project.
$ git clone https://github.com/docker/distribution.git

# Change to the Notary project directory.
$ cd notary

# Build the server image and run service on the local box
# mkdir notary2
$ git clone https://github.com/docker/notary.git
$ cd notary
$ docker-compose up -d

# Setup a local version of the Docker Registry v2
# Change to the notarysandbox/distribution directory.
$ cd ../../../notarysandbox/distribution

# Build the sandboxregistry server
$ docker build -t sandboxregistry .

#Start the sandboxregistry server
$ docker run -p 5005:5005 --name sandboxregistry sandboxregistry &

# Start the notarysandbox and link it to the running
notary_notaryserver_1 and  sandboxregistry containers. The links allow
communication among the containers.
$ docker run -it -v /var/run/docker.sock:/var/run/docker.sock --link
notary_notaryserver_1:notaryserver --link
sandboxregistry:sandboxregistry notarysandbox
```

Last Updated: February 26th, 2020

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS Secure Japan 2020	**Tokyo, JP**	**Mar 02, 2020 - Mar 14, 2020**	**Live Event**
ICS Security Summit & Training 2020	**Orlando, FLUS**	**Mar 02, 2020 - Mar 09, 2020**	**Live Event**
SANS Northern VA - Reston Spring 2020	**Reston, VAUS**	**Mar 02, 2020 - Mar 07, 2020**	**Live Event**
SANS Munich March 2020	**Munich, DE**	**Mar 02, 2020 - Mar 07, 2020**	**Live Event**
SANS Jeddah March 2020	**Jeddah, SA**	**Mar 07, 2020 - Mar 12, 2020**	**Live Event**
SANS St. Louis 2020	**St. Louis, MOUS**	**Mar 08, 2020 - Mar 13, 2020**	**Live Event**
SANS Paris March 2020	**Paris, FR**	**Mar 09, 2020 - Mar 14, 2020**	**Live Event**
SANS Dallas 2020	**Dallas, TXUS**	**Mar 09, 2020 - Mar 14, 2020**	**Live Event**
SANS Prague March 2020	**Prague, CZ**	**Mar 09, 2020 - Mar 14, 2020**	**Live Event**
Wild West Hackin Fest 2020	**San Diego, CAUS**	**Mar 10, 2020 - Mar 11, 2020**	**Live Event**
SANS Doha March 2020	**Doha, QA**	**Mar 14, 2020 - Mar 19, 2020**	**Live Event**
SANS Norfolk 2020	**Norfolk, VAUS**	**Mar 16, 2020 - Mar 21, 2020**	**Live Event**
SANS Secure Singapore 2020	**Singapore, SG**	**Mar 16, 2020 - Mar 28, 2020**	**Live Event**
SANS SEC504 Nantes March 2020 (in French)	**Nantes, FR**	**Mar 16, 2020 - Mar 21, 2020**	**Live Event**
SANS San Francisco Spring 2020	**San Francisco, CAUS**	**Mar 16, 2020 - Mar 27, 2020**	**Live Event**
SANS London March 2020	**London, GB**	**Mar 16, 2020 - Mar 21, 2020**	**Live Event**
SANS Kuwait March 2020	**Salmiya, KW**	**Mar 21, 2020 - Mar 26, 2020**	**Live Event**
SANS Secure Canberra 2020	**Canberra, AU**	**Mar 23, 2020 - Mar 28, 2020**	**Live Event**
SANS Seattle Spring 2020	**Seattle, WAUS**	**Mar 23, 2020 - Mar 28, 2020**	**Live Event**
SANS Madrid March 2020	**Madrid, ES**	**Mar 23, 2020 - Mar 28, 2020**	**Live Event**
SANS Oslo March 2020	**Oslo, NO**	**Mar 23, 2020 - Mar 28, 2020**	**Live Event**
SANS SEC560 Lyon March 2020 (In French)	**Lyon, FR**	**Mar 23, 2020 - Mar 28, 2020**	**Live Event**
SEC588 Beta One 2020	**Arlington, VAUS**	**Mar 23, 2020 - Mar 28, 2020**	**Live Event**
SANS Abu Dhabi March 2020	**Abu Dhabi, AE**	**Mar 28, 2020 - Apr 02, 2020**	**Live Event**
SANS FOR585 Rome March 2020 (In Italian)	**Rome, IT**	**Mar 30, 2020 - Apr 04, 2020**	**Live Event**
SANS Frankfurt March 2020	**Frankfurt, DE**	**Mar 30, 2020 - Apr 04, 2020**	**Live Event**
SANS Philadelphia 2020	**Philadelphia, PAUS**	**Mar 30, 2020 - Apr 04, 2020**	**Live Event**
SANS 2020	**Orlando, FLUS**	**Apr 03, 2020 - Apr 10, 2020**	**Live Event**
SANS Muscat April 2020	**Muscat, OM**	**Apr 04, 2020 - Apr 09, 2020**	**Live Event**
SANS Riyadh April 2020	**Riyadh, SA**	**Apr 04, 2020 - Apr 16, 2020**	**Live Event**
SANS Bethesda 2020	**Bethesda, MDUS**	**Apr 14, 2020 - Apr 19, 2020**	**Live Event**
SANS Minneapolis 2020	**Minneapolis, MNUS**	**Apr 14, 2020 - Apr 19, 2020**	**Live Event**
Blue Team Summit & Training 2020	**OnlineKYUS**	**Mar 02, 2020 - Mar 09, 2020**	**Live Event**
SANS OnDemand	**Books & MP3s OnlyUS**	**Anytime**	**Self Paced**