Managing security in the container image

Natan Teferi Asegehegn: ntas@kth.se

April 2021

Abstract

As DevOps becomes increasingly more adopted in software development, the need to incorporate security becomes more and more apparent. Containers are a very powerful technology that solves many of the issues that software teams ones faced. Their isolated nature makes them a perfect tool to bring together software developers and IT operations. However, they come also with their flaws when it comes to security. A mistake in the parameter settings can leave a free gateway for attackers to take control of the host machine. Unverified third party images can have malicious software that can leave containers vulnerable to attacks. The intricate software dependencies that lie within images leave a wide surface area for attackers to take advantage of.

Reviewing the parameters settings before deployment can be a preemptive solution to make sure containers only have the privileges they require to operate and nothing else. Static and dynamic analysis of images to detect malicious code can help secure the infrastructure. Finally, automatic update of an image when a package it uses has received a patch can improve security drastically.

To minimize breaches, security should be taken into consideration during every step of the deployment of containers. From managing secrets properly to configuring good logging infrastructure, these are steps that will eventually pay off in the long run.

1 Background

Containerization is a technique that allows the encapsulation of an application with its own operating environment. Unlike the traditional virtualization via virtual machine, which partition the hardware resources and require us to install an operative system for each virtual machine, containerization uses the same underlying operative system. This significantly reduces the overhead when running an application [1]. Figure 2.1 below shows the differences in overhead between traditional virtualization via virtual machine and virtualization via containers.

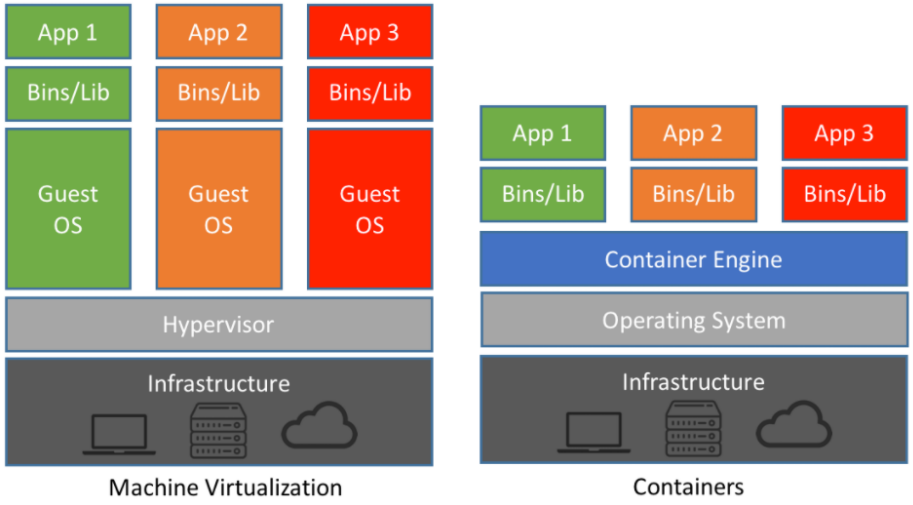

Figure 1.1: Overhead of virtualization via virtual machines vs containerization [2]

Docker is the most commonly used tool to package and run applications in containers. Containers are simply units of software that packages up code and all its dependencies so that an application can run smoothly on any underlying operating system [3]. Since containers have all the software dependencies an application requires, there is no need to rely on what is currently installed on the host machine.[4] This creates a high level of isolation without the need of multiple virtual machines. Figure 2.2 shows the architecture behind Docker.

In order to create a container, a template containing the relevant software dependencies is required. This template is called an image. An image is read-only and it contains instructions for the creation of the Docker container. Oftentimes, images are built on top of other images. For instance, one can create an image on top of the Ubuntu image, but that installs other packages and dependencies that are not found on the base Ubuntu image. [4]

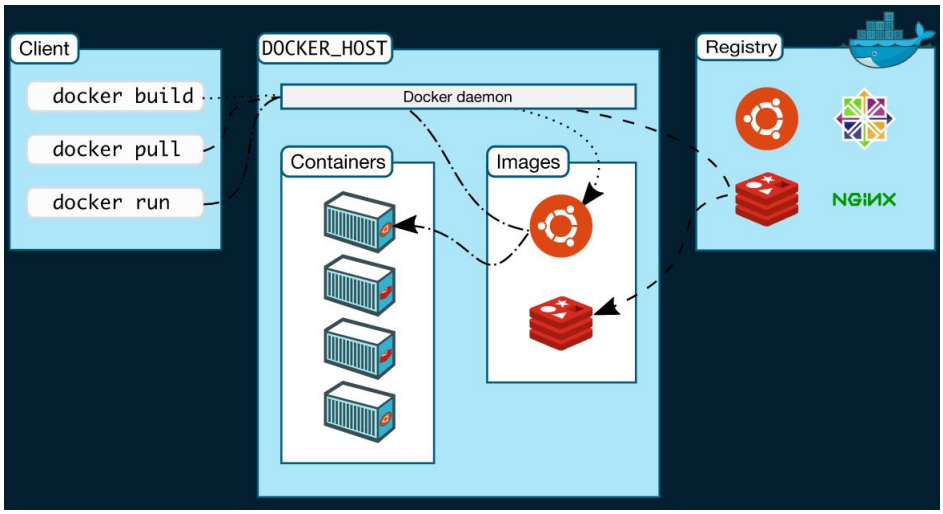

Figure 1.2: Overview of the communication flow between the different process behind Docker [4]

2 Security weaknesses and how to mitigate them

This section describes the security vulnerabilities that may arise when using containers as well as how developers can overcome them. Since Docker [5] is the most commonly used tool to deploy and run applications in containers, it will be the primary focus for the vulnerabilities described here. However, these vulnerabilities may also expand to other engines.

2.1 Faulty parameter settings

Docker Hub is the most popular registry of image repositories. [6] These images can be used by users in their own configuration instead of needing to create their own custom image every time they want to run containers. When running a Docker container, a user need to execute the run-command. This command specifies the image and the parameters that should be used when starting the container. A potential threat to security are parameters that are given to the run-command. These parameters can affect the behaviour of the containers that are running. For instance, if given the flag "–privileged" to the run command, the container will have root access on the host machine. It is clear here that if certain operational parameters are miss-used or miss-configured, it may lead to disastrous consequences. [7]

A way to stop users from running containers with sensitive parameter settings is to make sure there is a clear and thorough description of what a parameter/flag does and what consequences it can lead to. Often times users simply run the

commands that are displayed in the repository documentation. A lot of damage could be mitigated by simply adding warnings on what certain parameters do when used. This could be done by adding some sort of text analysis, which could run when there is a new entry on Docker Hub. [7][8] From the user's side, it is important to analyze and understand what each command does before using it.

2.2 Malicious container images

Another vulnerability to the use of containers for DevOps are malicious container images. These images may reside on Docker Hub and plague the user ones they are pulled. An example of the consequences of malicious images occurred in 2019, when electronic coin miners were used in certain images on Docker Hub, earning the attackers a profit of $90000 [9]. This is also a result of the lack of thorough controls when users upload an image on Docker Hub.

In order to detect images containing malicious software, it is important to do an analysis of the relevant software packages. This can be done with static or dynamic analysis (e.g. signature based). Static analysis refers to the manual check of software contained in an image, while dynamic analysis refers to the trace of system calls and API calls at run-time. However, the sheer size and complexity of images can make it extremely time consuming to do such analysis. Therefore a more a pragmatic approach would be to remove unnecessary software components before doing such analysis. If this removal cannot be done for some reason or another, then a heuristic approach (analyzing related images for example) could offer a workaround.[7] Generally, when incorporating images supplied by third parties, it is important to ensure that they are from a trusted source and that the integrity of the image is verified. [8]

2.3 Breaches in software dependencies

When creating a Docker image, there are many packages and software dependencies that come into play. A single security flaw in any of the dependencies means that the whole image could possibly be compromised. The larger and more intricate the dependencies between software packages are, the harder it gets to detect vulnerabilities. Furthermore, Docker software programs are often duplicated from the original ones. This means that a bug that is fixed quickly in a software packages might not necessarily be fixed as quickly in the duplicate that is used in the Docker program. There is also a lack of general incentive for Docker developers to fix issues in the duplicates in a timely fashion.[7] As a result, the security flaws found in software packages are elevated when said packages are used with Docker.

A solution to this problem is to automatically update packages as soon as a patch has been released. [7] In order to avoid breaking an image with automatic updates, a solution might be to rebuild the image frequently; at least as frequently as security updates are released. To make the rebuilding as smooth and time

efficient as possible, a continues integration and continuous deployment pipeline can be set up. When updates become available, the pipeline automatically tests and redeploys the newly built image to the operational environments. [8]

2.4 Good habits to improve security

Managing security in a container is not a binary operation. As discussed above, there are several attack vectors that come into play. Adopting good habits will come a long way into making sure that any security breaches do as minimum damage as possible.

2.4.1 Protecting the container host

The first step in protecting the integrity of containers is to ensure the security of the host machine. Ideally, a container host should not be used for any other purpose. By doing so, the number of users who need access to said host is minimized. This also minimized the attack surface by reducing the number of software packages that are installed on the host machine.[8] Containers should also adopt the principle of least privilege [10]. A container should only have the privileges it needs to execute its task and nothing more.

2.4.2 Managing secrets

The storage of credentials and private keys should never be done inside containers. The immutable nature of image layers make them unsafe to store secrets. Even if the files containing the secrets were to be deleted, they could still be retrieved from earlier image layers. Therefore, the safest way to manage secrets is to use the management system provided by the container orchestrator and load them at run-time. [8]

2.4.3 Limiting resource use

There should be resource limit configurations for any given container. If there is no such limit, an attacker that is unable to escalate the privileges from an infected host can opt for a Denial of Service (DoS) [11] attack instead. To protect from such attacks and to protect an infected container from impacting other containers on the same host, flags provided by the container engine should be used to limit CPU, memory and other resource usage. Docker provides a ”–memory” flag to specify memory usage by a container and a ”–cpus” flag to specify the how much of the available CPU resources a container should use.[12]

2.4.4 Configure a logging system

Concise and descriptive logs are crucial in order to be able to swiftly resolve errors as well as respond to possible attacks to the system. The nature of containers, however, can sometimes complicates the process of collecting logs [13]. A solution to this problem is to configure the containers to send the logs

to a centralized server that fetches them and process them. Developers are then able to easily access the logs in case of a security breach. [8]

3 Conclusion

As the need for more rapid deployment of software becomes increasingly predominant, containers occupy a more crucial role in the development pipeline. The addition of new tools, however, also means a wider surface area for attackers to take advantage of. Considering security on every step during the deployment of containers will eventually bear its fruits in the long run.

References

[1] *Virtualization via Containers*. en. URL: https://insights.sei.cmu.edu/blog/virtualization-via-containers/ (visited on 04/08/2021).

[2] URL: https://courses.engr.illinois.edu/cs398acc/sp2018/slides/Lecture%2012.pdf (visited on 04/08/2021).

[3] *2What is a Container? — App Containerization — Docker*. en. URL: https://www.docker.com/resources/what-container (visited on 04/17/2021).

[4] *Docker overview*. en. Apr. 2021. URL: https://docs.docker.com/get-started/overview/ (visited on 04/17/2021).

[5] *Empowering App Development for Developers — Docker*. en. URL: https://www.docker.com/ (visited on 04/08/2021).

[6] *Large-scale-analysis*. URL: https://people.cs.vt.edu/~butta/docs/cluster2019-dockerhub.pdf (visited on 04/08/2021).

[7] *Understanding the Security Risks of Docker Hub*. URL: https://www-users.cs.umn.edu/~kjlu/papers/docker.pdf (visited on 04/08/2021).

[8] *7 Quick Steps to Using Containers Securely*. en. URL: https://insights.sei.cmu.edu/blog/7-quick-steps-to-using-containers-securely/ (visited on 04/17/2021).

[9] *Malicious Docker Containers Earn Cryptomining Criminals $90K*. en. URL: https://threatpost.com/malicious-docker-containers-earn-crypto-miners-90000/132816/ (visited on 04/08/2021).

[10] Fred B. Schneider. *Least Privilege and More*. URL: http://www.cs.cornell.edu/fbs/publications/leastPrivNeedham.pdf (visited on 04/18/2021).

[11] Jeeva Chelladhurai, Pethuru Raj Chelliah, and Sathish Kumar. "Securing Docker Containers from Denial of Service (DoS) Attacks". In: June 2016, pp. 856–859. DOI: 10.1109/SCC.2016.123.

[12] *Runtime options with Memory, CPUs, and GPUs*. en. Apr. 2021. URL: https://docs.docker.com/config/containers/resource_constraints/ (visited on 04/18/2021).

[13] Mubin Ul Haque. *Challenges in Docker Development: A Large-scale Study Using Stack Overflow*. URL: http://arxiv-export-lb.library.cornell.edu/pdf/2008.04467 (visited on 04/19/2021).