Framework to Secure Docker Containers

Conference Paper · July 2021

DOI: 10.1109/WorldS451998.2021.9514041

2 authors, including:

Manish Abhishek
Indian Railways
13 PUBLICATIONS **48** CITATIONS

SEE PROFILE

Framework to Secure Docker Containers

1st Manish Kumar Abhishek
Department of CSE
Koneru Lakshmaiah Education Foundation
Vaddeswaram, India
manish.abhishek@gov.in

2nd D. Rajeswara Rao
Department of CSE
Koneru Lakshmaiah Education Foundation
Vaddeswaram, India
rajeshpitam@gmail.com

***Abstract*—Docker is one the key component for application deployment using CI/CD pipelines. Wherever containers are going ot be used, Docker engine is always the first choice but on other hand security of the Docker images using which application is going to be deployed is always a concern. In cloud computing, validation of the Docker images security is a paramount. Containers virtualization which is based on operating system virtualization is not secure as hypervisor virtualization. In this paper we are proposing a framework which uses an architecture including plugins, CI/CD pipeline to deploy the application to ensure the security of application bundled as a Docker image. It is going to be referred from the starting of application development till the deployment including plugin for Docker build, bundling the application in form of images along with required libraries, pushing the images to Docker registry. Jenkin jobs are going to be used for getting the build and then for deployment. For validation, we came up with vulnerable Docker images and validated against our architecture having proposed model to compare the results. In later sections, we have also considered the containers security measures.**

Keywords— Containers, Continuous Integration (CI), Continuous Deployment (CD), Docker, Jenkin, Virtualization

I. INTRODUCTION

In today's containerization world, the application development and deployment using CI/CD pipeline in one of the common requirements that everyone is looking at. Containers are very light weight in nature as they are totally based on Operating System (OS) virtualization and do not need the whole configuration for its dependent binaries. They have been significantly adopted for the whole software lifecycle from deployment to development phases including upgrades, fix packs and much more. They are in trend an making a buzz in entire IT world. Containers have been considered a suitable choice even for High performance computing applications via adopting their dynamic allocation at the infrastructure level in cloud computing [1]. Instead of having virtual machines using hypervisor, containers have been given more weightage to make the environment scalable and suitable for failover use cases. It executes on isolated layer of operating system to avoid the overhead. They are very portable in nature in comparison to virtual machines which are basically using the whole individual operating system including its own libraries, binaries as an individual underlying layer of hardware layer via hypervisor. This is one of the major reasons behind the adoption of containers in virtualization world.

Docker is an open source platform to facilitate applications in packages none other than containers. It consist multiple light weighted containers. Docker consists several modules which allows to bundle the application in form of image along with its dependencies, tagging of images, uploading and downloading of images to/from Docker hub or private registry, their execution and even persisting the running state in terms of updates to use later. Docker commands need to be executed to achieve all these tasks and Docker file is going to be used to bundle all the required dependencies, libraries. For software deployment using Docker, the development community has been created the Docker hub that is used for persist the Docker images as a repository. Now a day for software deployment features are spitted in terms of micro services to achieve the code reusability and independent deployment via containers. Every container is going to represent one individual process holding its own process id during its execution. It is recommended to have one container per micro service holding individual or set of feature/module.

This requirement of using containers to bundle the application in form of Docker image is raising a security concern for its distribution. Software architects are also encouraging the inclusion of risk analysis which considers the distribution pipeline as malicious one. In this paper, the proposed framework is based on the multi-layer security architecture including the private registry to push the images to address the vulnerabilities. For application deployment, it is based on several stages wrapped as a Jenkin job in terms of CI/CD pipeline to evaluate the bundled Docker images that is pushed later in Docker hub which is eliminating the push and continuous reuse of vulnerable Docker images. Images selection along with malicious content has been also analysed and evaluated via using the CI/CD pipeline to figure out the abnormal execution environment followed by security standards and practices with containers considerations. The remaining sections in this paper have been represented in following manner: Section 2 is describing the background details of security analysis and concerns with respect to Docker images. Section 3 is all about the proposed framework model using which the whole architecture has been defined. Section 4 is describing the evaluation methodology, results and the related work. Conclusion is provided in Section 5.

II. SECURITY ANALYSIS

It is difficult to move the whole monolith application to container in comparison to micro services. With monolith application, scalability and code reusability is always a concern as with software lifecycle, the whole application grows and it's difficult to maintain the legacy code over a time of period. The application running on physical server to a Docker Container, it requires the identification of all the elementary components. Figure 1 shows the abstracted architecture for Docker containers. With MVC architecture, every layer is divided into containers. For example, database layer will be in one container, web app in another and need a separate for component holding the business logic along with its server. It has been analysed and found that file or executions examination is one of the forms of malware analysis to find out the probable cause of security compromises. It is mainly categorized into two parts with

respect to images. One of them is static where second one is at run time or can be called as behavioural analysis.

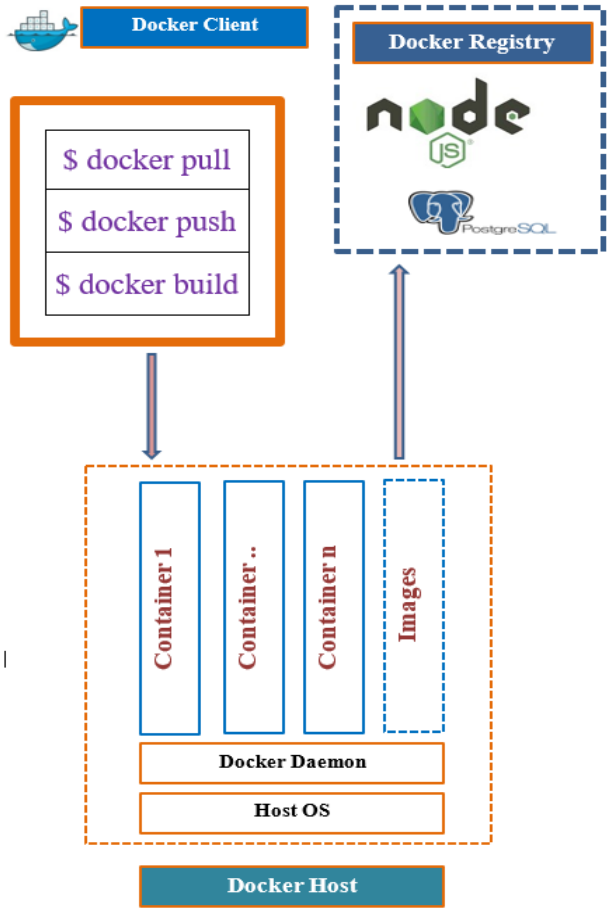

Fig. 1: Docker containers high level architecture.

A. *Static*

During static kind of analysis, the data has been examined prior to its execution. It is holding a set of predefined actions. CVE system is one the example [2]. In past, software vulnerabilities were often computed by scoring metrics. Multiple scoring metrics were computed to find out the vulnerability. The third-party jars used as a part of software development were also getting considered from security issues. Common Vulnerabilities and Exposures (CVE) system is used against the inconsistent results to find out the security issues and vulnerabilities. CVE is internally providing the framework to share these details publicly [3]. The most common option to find out the security issues is using the CVE system to scan the Docker images regularly which will be easily addressed before application deployment. Appropriate actions can be taken against these vulnerabilities. SonarQube is another example which is going to continuous examines the code quality. It will help to find out the unreachable code, memory leak, boundary value violations, code smells, and vulnerabilities, bugs categorized in form of blocker, critical, major and minor. Static analysis is also helpful to find out the malicious files based on hashes, signatures and extension of file. It is mainly going to analyses the security issues at application development level and before its deployment.

B. *Dynamic*

It is applicable after application development and referring the container's behaviour. This analysis includes containers monitoring including ports scan, consumption of resources in terms of CPU, memory, registry keep tracking, network activity and firewall rules monitoring. As it is related to containers and performed at execution level, results can be more impactful. Even dynamic allocation of computing resources with respect to containers always helps us for their monitoring [4]. Docker engine is going to handle and take care of containers provisioning where each container is well defined with its own computing resources [5] and not going to impact other containers. Every container is going to hold its own unique process id. Containers can impact the host if it is going to be executed by the admin or root privileges. On other hand if multiple containers are executed by the same user, they can also impact each other. If every container is going to have its own network bridge with respect to host physical network, single container can consume the whole network and result into network denial of service attack. One of the major issues is to trust the Docker image downloaded from the Docker hub that is available publicly for all [6].

III. PROPOSED DESIGN

Here, we are describing the designed framework architecture to address the analysed security issue or vulnerabilities via scanning the images and third-party jars regularly fitted inside a model. CI/CD pipeline has been created to track the whole workflow from development to deployment. Using this proposed architecture of our framework, it will minimize the cost as well as security analyses fix faster and transient to user. Figure 2 shows the proposed architecture to secure Docker containers.

A. *Continuous Integration (CI) Pipeline*

Continuous Integration pipeline is going to be responsible for building the Docker image and pushing it to private registry followed by multiple stages itself in pipeline. Once the code changes have been pushed to the code base repository, this pipeline will start. First stage of pipeline will be checking out the code changes implemented by developer, and then it is going to compile it followed by running test cases in case of any and generate the SonarQube report. This report is now be the static analysis to figure out the code smells, bugs, security vulnerabilities. This report is having the details of code coverage, new code changes code coverage, issues, measures of reliability, security, maintainability and other application details. Issues type will be bugs, vulnerability, code smells and security hotspot having the unsafe arguments details. Using this report having the details of static analysis of application code, we can easily address the security vulnerabilities even before creation of our Docker image. Once issues have been addressed, we can go ahead with later stages of pipeline i.e. Docker build, Image Tagging, Pushing the images to registry and updated the image versions for build track. Instead of using the public registry, every organization can have a private registry to keep images of their application which will make it more secure.

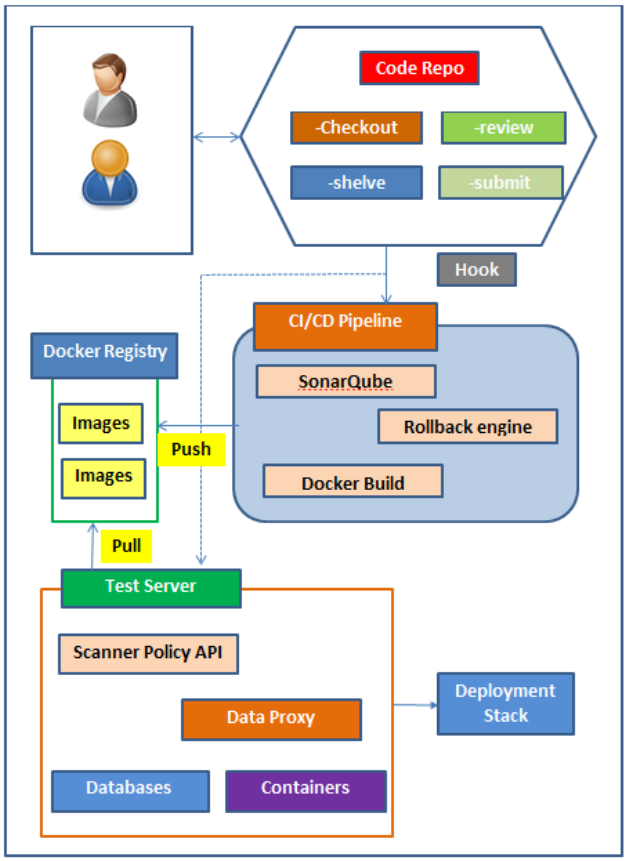

Fig. 2: Proposed model to secure Docker containers.

B. *Continuous Deployment (CD) Pipeline*

Continuous deployment pipeline is going to be responsible for the deployment of application build i.e. the result of CI pipeline. In the first stage of this pipeline, it will first check the registry details, download the latest version image automatically and extract the archived images. In next stage, it is going to deploy it. If already exists, will do uninstall/upgrade accordingly. Here we have also introduced our own automated API which will take care of identifying the security issues with Docker images. If we use public Docker registry, this API will help us to find out the issues. It is mainly going to execute that image to find out the traffic, network changes, firewall rules and other security scans before deploying it to production environment. It is a kind of staging environment execution that will help us to find out the issues in labs only before releasing it for usage and deployment. We have analysed and found that mostly the images which include the bash scripts responsible to create the secure shell (SSH) tunnels during container execution are the malicious one. These images tend to download the non-defined binaries and installing the shell code. To address this issue, we are proposing a small service which will examine the Docker image and removed thee kind of binaries. We have used sandbox in one of our cloud services to provide the isolation.

IV. EVALUATION AND RESULTS

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the Here, we are evaluating our proposed architecture for determining the security vulnerabilities and hotspots related to Docker containers including both static as well as dynamic resources allocation, usage of free computing resources for application, performance, profiling, and queue throughput. Table 1 shows the details of Docker images and its version details. We have used the PostgreSQL server for persistency as a database, CentOS 7 as operating system, AdoptOpenJDK 11 for application development basically maven build, Docker version 18.09.7, Nginx 1.10.3 and Jenkin Server with 2.164.1 as a version.

TABLE I. DOCKER IMAGES AND THEIR VERSION USED FOR EVALUATION

Images	Version
CentOS	7.0
PostgreSQL	12.0
Docker	18.09.07
AdoptOpenJDK	11
Nginx	1.10.3
Jenkin	2.164.1

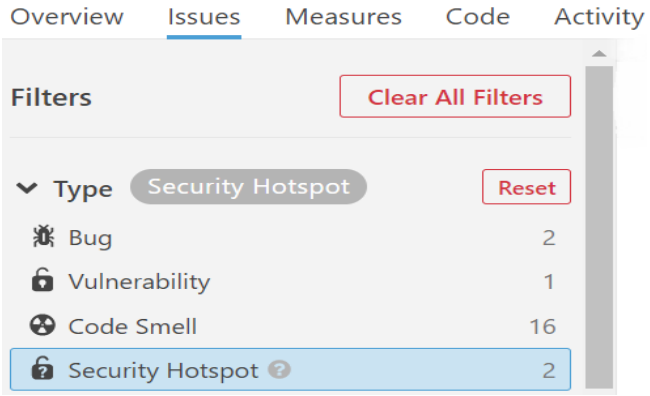

Fig. 3: SonarQube report details.

Figure 3 shows the sonar report that has been generated during CI pipeline as one of the stages to find out the security hotspots and vulnerabilities. Even before checking the code, one round of examine has been done by developers using local server for analysis. For our application, we addressed mostly issue at development time only but still found 2 security hotspots and 1 vulnerability due to the usage of old jars. For CD pipeline, we have deployed our application having REST Server where API has been exposed as an API gateway using Nginx. Swagger documentation is provided to invoke the REST APIs respective to modules. As an infra we have deployed the PostgreSQL which is more secure. For CentOS images, we have the defined rule which will make sure that after having a count of below than 16 minor vulnerabilities use, the quality gate will be marked as true and we are good to use this image. As a result of our quality gate, we found that PostgreSQL image is secure and Nginx using our application has been passed even with 120 as a total count of security vulnerabilities as they were not falling into category of critical and blocker. SonarQube repost has been also passed even after having 90 minor issues. Docker file has been analysed using the Achore Engine's security policy. PostgreSQL image has cleared all the analysis and found suitable to push to our private registry. For Nginx based application we were unable to push image as found vulnerable checks for it.

We have also used the VirusTotal as an antivirus engine to scan the images and found the malicious content. Figure 4 shows the scanning results performed on Docker images to find out the malicious content. One of the root causes was found that whenever we are using the old version of few jars. Wherever possible always try to use latest stable version of third-party jars. Third party jars need to be upgraded regularly per release. Figure shows the results performed using VirusTotal i.e. antivirus engine. As CentOS, Java, Nginx and node images are default supported by Docker registry but available publically, they have been scanned before pushing into our private registry. The application images including all dependencies/libraries found malicious mainly due to start pod, probe scripts and old jars.

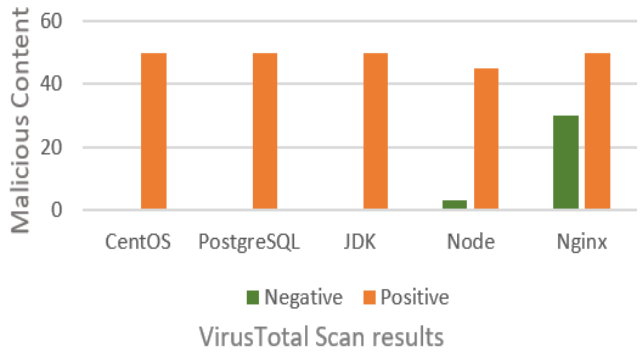

Fig. 4: Docker image scan results for malicious content.

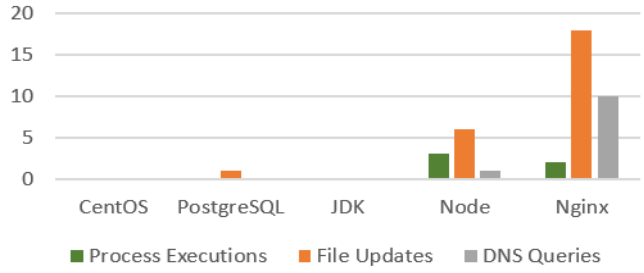

Fig. 5: Dynamic analysis for Docker images at run time.

Figure 5 shows the results of dynamic analysis API including the process executions, file updates followed by 25 seconds of runtime. File system is getting modified with respect to the image functionality and performed actions. We have examined the system and computing resources, for example CPU metadata. We found the major issue once images started the SSH daemon. For network traffic we have uploaded a pcap during image execution that found abnormal in nature. The observed finding was around DNS request and found the image as malicious one as it was trying to attempt cryptocurrency.

A. *Related Work*

For CVEs, there are multiple tools that can be used. For example: OpenSCAP6. It examines and based on data available as per the National Vulnerability Database7 determines the security vulnerabilities and policies violations. To scan the Docker images, generally oscap-Docker tool can be used. Many tools are open source and can be easily integrated with CI/CD pipeline for static and dynamic analysis. Even Docker itself officially offers trusted registries for security scan as per CVE database but we need to pay for that. CI/CD pipeline is helpful in case of Docker containers security checks. Adethyaa and Jernigan [7] demoed a CI/CD pipeline for Docker images which uses AWS resources. Valance [8] also demonstrated the same with the help of Anchore Engine to perform security analysis on Docker images. We observed a limitation with their process i.e. the manual provisioning of AWS services and an inability to define custom security policies. With Valance's approach there is a lack of source which really initiates the whole CI/CD pipeline. If we compare these with our proposed framework, it will be comparable that proposed architecture is automated, scalable and simplify the complex workflows of application release/upgrades and flexible in nature.

V. CONCLUSION

For application development and deployment, Docker containers are the best choice but on another hand, these are vulnerable and lacks of appropriate tools are effectively quantifying the risk. Major tools are either needs enterprise license or time consuming. On the top that there is always a risk as they are using third party jars. Our proposed framework can be easily adopted in any organization as its core architecture to find out the bugs, security hotspots, vulnerabilities before pushing the Docker images to registry and also scanning the images to avoid the issues at runtime. Malicious images will always try to download and execute the non-defined dependencies and can be vulnerable. With our static and dynamic analysis approach, developers can easily examine the security defects and push non-malicious Docker images. The complete automated flow is ensuring the security concerns without any manual interventions.

ACKNOWLEDGMENT *(Heading 5)*

A special thanks to the Koneru Lakshmaiah Education Foundation for facilitating me required infrastructure and my guide helpful nature as well as other staff members who helped me to accomplish this research work.

REFERENCES

[1] Abhishek, Manish. (2020). Containerization for shipping Scientific Workloads in Cloud. International Journal of Advanced Trends in Computer Science and Engineering. 9. 5327. 10.30534/ijatcse/2020/166942020.

[2] P. Mell, K. Scarfone, and S. Romanosky, "The Common Vulnerability Scoring System (CVSS) and Its Applicability to Federal Agency Systems," National Institute of Standards and Technology, Tech. Rep. Interagency Report 7435, August 2007.

[3] Abhishek, Manish. (2020). Dynamic Allocation of High-Performance Computing Resources. International Journal of Advanced Trends in Computer Science and Engineering. 9. 3538-3543. 10.30534/ijatcse/2020/159932020.

[4] R. A. Martin, "Managing vulnerabilities in networked systems," in Computer, vol. 34, no. 11, pp. 32-38, Nov. 2001, doi: 10.1109/2.963441.

[5] Abhishek, Manish. (2020). High Performance Computing using Containers in Cloud. International Journal of Advanced Trends in Computer Science and Engineering. 9. 5686. 10.30534/ijatcse/2020/220942020.

[6] D. Goodin, "Backdoored images downloaded 5 million times finally removed from Docker Hub," [https://arstechnica.com/informationtechnology/ 2018/06/backdoored-images-downloaded-5-million-timesfinally- removed-from-Docker-hub/, June 2018].

[7] V. Adethyaa and T. Jernigan, "Scanning Docker Images for Vulnerabilities using Clair, Amazon ECS, ECR, and AWS CodePipeline," AWS Compute Blog, November2018[https://aws.amazon.com/blogs/compute/scanning-Docker-images-forvulnerabilities- using-clair-amazon-ecs-ecr-aws-codepipeline/.].

[8] J. Valance, “Using Anchore Policies to Help Achieve the CIS Docker Benchmark,” Anchore Blog, May 2019, [https://anchore.com/cisDocker- benchmark/]

[9] K. Brady, S. Moon, T. Nguyen and J. Coffman, "Docker Container Security in Cloud Computing," 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2020, pp. 0975-0980, doi: 10.1109/CCWC47524.2020.9031195.

[10] J. Blackthorne, A. Bulazel, A. Fasano, P. Biernat, and B. Yener, “AVLeak: Fingerprinting Antivirus Emulators through Black-Box Testing,” in 10th USENIX Workshop on Offensive Technologies. Austin, TX: USENIX Association, Aug. 2016.

[11] V. Rastogi, C. Niddodi, S. Mohan, and S. Jha, “New directions for container debloating,” in Proceedings of the 2017 Workshop on Forming an Ecosystem Around Software Transformation, ser. FEAST ’17. New York, NY, USA: ACM, November 2017.

[12] Mullinix, Samuel & Konomi, Erikton & Townsend, Renee & Parizi, Reza. (2020). On Security Measures for Containerized Applications Imaged with Docker.

[13] D. Huang, H. Cui, S. Wen and C. Huang, "Security Analysis and Threats Detection Techniques on Docker Container," 2019 IEEE 5th International Conference on Computer and Communications (ICCC), Chengdu, China,2019, pp. 1214-1220, doi: 10.1109/ICCC47050.2019.9064441.

[14] A. R. Manu, J. K. Patel, S. Akhtar, V. K. Agrawal and K. N. B. Subramanya Murthy, "A study, analysis and deep dive on cloud PAAS security in terms of Docker container security," 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT), Nagercoil, 2016, pp. 1-13, doi: 10.1109/ICCPCT.2016.7530284.

[15] 15. P. P. W. Pathirathna, V. A. I. Ayesha, W. A. T. Imihira, W. M. J. C. Wasala, N. Kodagoda and E. A. T. D. Edirisinghe, "Security testing as a service with Docker containerization," 2017 11th International Conference on Software, Knowledge, Information Management and Applications (SKIMA), Malabe, 2017, pp. 1-7, doi: 10.1109/SKIMA.2017.8294109.