Docker: Version and Security Upgrades with Containerized Applications

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Johannes Cornelis Hollebrandse
Fall 2022

Briana Morrison, Department of Computer Science

Docker: Version and Security Upgrades with Containerized Applications

CS4991 Capstone Report, 2022

Corneel Hollebrandse
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
jh7jss@virginia.edu

Abstract
Progeny Systems utilized outmoded and unsecure versions of its CI/CD (continuous integration and continuous delivery tool) and static code analysis tools, leaving them without modern features and vulnerable to security exploits. Containerizing these tools simplifies the time-consuming manual processing of updating and upgrading security features. A team consisting of myself and one other intern developed Docker containers that pulled the base images of these tools from supported repositories, then mounted volumes and, finally, created a Docker network. This network utilized an Nginx reverse proxy to secure web traffic to the front-facing interfaces of the CI/CD and static code analysis tools with https. The effort made both tools more maintainable for future upgrades and more secure, and led to new static analysis on 50,000 lines of code within the code base. Future work could include further upgrades to the static analysis tool to scan shell Progeny scripts.

1. Introduction
United States Navy submarines carry up to 70% of the nation's nuclear arsenal [1]. The primary defense contracts for Progeny Systems are submarine related. I interned with the IA (Information Assurance) team, which focuses on securing communications with the submarine fleet. If a malicious agent were to access the internal Progeny network, they could have attacked the CI/CD and static code analysis tools in order to steal information about the code that keeps communication with the submarine fleet secure.

2. Related Works
The Raddatz and Martinez (2022) paper on submarine nuclear capability was shared on a company email during the internship to remind us of the importance of our work. It clearly explains the importance of upgrading security features for the tools the IA team uses.

Bellairs (2020) explains the importance of static code analysis. It provides background on the importance of SonarQube being extended to more languages.

Stoneman (2020) explains the conceptual importance of Docker and provides tutorials to teach Docker principles and how to properly write Docker files. My manager gave me a physical copy on the first day of the internship and I used it extensively to implement containers.

3. Process Design
Implementing upgrades to the CI/CD and static code analysis tools requires a greater depth of understanding of the security issues affecting both systems, docker as a means to

fix them, and specifically how docker was tuned to solve this problem.

3.1 Security Issues
The CI/CD and static code analysis tools the IA team uses had major security issues. The CI/CD tool the IA team uses is Jenkins, and it was running a version released in 2015, causing the Jenkins web interface to list a page and a half of security issues associated with running that outmoded version. The static code analysis tool the IA team uses is SonarQube, which was running a version released in 2021 that had only been configured to scan java code. Although around 70% of the IA code base was written in java, the remaining 30%, consisting of shell scripts, python, and C code, was unscanned by SonarQube, lowering the quality and security of non-java code because static analysis reveals security hotspots, bugs, code smells and general syntax inconsistencies [2].

3.2 Docker
Applications often rely on external packages and libraries. An application that works on one operating system will likely not work in another operating system. Docker solves this problem by packaging the application with all the packages and libraries needed into a container. This container will then operate the same way on every machine. Without docker, a large operating system would have to be installed and configured in a precise way to enable the application to work. Docker provides the consistency of application function while utilizing substantially less computer storage. Another advantage of Docker is that if the application image a container is dependent on is changed (updated security features, for example), the container simply needs to be restarted to automatically pull the updated image [3].

3.3 Implementation
To containerize Jenkins, I created a docker-compose file that functions as a script of command line docker instructions. These instructions mounted an external volume that stored ssh keys for connecting to the various software builds that Jenkins needed to perform CI/CD on.

Additionally, they started an Nginx container that ran on ports 80 and 443 for http and https traffic respectively. Finally, the docker-compose commands started the Jenkins container on port 8085 and pulled its base image from the official Jenkins host service every time the container started. Port 8085 was blocked on the firewall such that Nginx was the only service that could access that port, and by extension Jenkins. Taken together, this means docker will always pull the most secure supported version of Jenkins each time the container is restarted and ensure that those who try to connect to the Jenkins web interface in http will be redirected by the Nginx reverse proxy to connect to the interface securely with https.

To containerize SonarQube, I also created a docker-compose file that applied many of the same commands the docker-compose file for Jenkins did. The only major difference was that SonarQube by default runs on port 9000, so instead of 8085 the SonarQube container ran on this default port. Otherwise, Nginx was set up the same way to redirect unsecured traffic to https. To add multi-language support to SonarQube, I mounted a docker volume that would save configured settings between restarts of the container. After achieving persistent storage, I configured SonarQube to support static code analysis of shell, python, and C code via SonarQube's web interface.

4. Results

Containerizing the IA team's Jenkins and SonarQube services saves the IA team from expending costly labor hours manually updating them. This is evident in the SonarQube service, which had taken a junior developer one week to upgrade from 8.9 to version 9.1 in early 2021. Containerization trivializes this process by pulling the most recent image from the internet whenever the container is restarted. Effectively, what was previously a week-long task now takes seconds.

Additionally, because this updating happens every time the container is restarted, the containerized services are now much more likely to run a secure version of the software instead of running an outmoded, insecure version for several years before being updated.

Finally, adding multi-language support to the SonarQube service allowed an additional 50,000 lines of code to be scanned from the existing IA team code base. This resulted in the discovery of 53 new bugs, over 700 new code smells, and uncovering one new security hotspot.

5. Conclusion

At the beginning of my internship with Progeny Systems, their CI/CD and static code analysis tools were outmoded and unsecured. By using docker to upgrade these tools, they are now secured and future updates to them are trivialized to simply restarting the containers the tools are hosted on. This helps secure Progeny System's vital national defense work and saves future labor hours manually upgrading these tools. Additionally, the upgrades increased the efficacy of finding bugs and security issues within the codebase itself.

6. Future Work

Future work for upgrading the CI/CD and static code analysis tools could include writing a script that sends failed builds to the Progeny Microsoft Teams in real-time and upgrades the SonarQube license to scan more lines of code in the codebase.

7. UVA Evaluation

Three UVA courses stood out as particularly valuable for this internship: CS4501, CS3710, and CS4414. *CS4501 Spec Topics: Intro to Software Analysis* taught me about static vs dynamic code analysis which was extraordinarily helpful when upgrading SonarQube. *CS 3710 Intro to Cybersecurity* made me comfortable with the Linux command line and taught me about securing http traffic with https as well as other general cybersecurity principles that I applied when constructing my docker files to make them safer. *CS 4414 Operating Systems* gave me the knowledge to debug issues with my machine as well as improve my conceptual understanding of the importance of docker as a replacement for an operating system.

The IA team was shocked to learn that I had no experience with docker at UVA and although UVA offers it in cloud computing, the class fills up immediately every semester, which meant I never had a chance to take it. My only suggestion to improve the UVA Computer Science Program would be to increase the number of seats available in cloud computing courses.

8. Acknowledgments

I would like to acknowledge my manager at Progeny Systems, Kevin Sullivan, for his technical support and leadership during the internship.

References

Bellairs, R. (2020, February 10). *What is static analysis? Static Code Analysis Overview*. Perforce Software.

Retrieved September 20, 2022, from https://www.perforce.com/blog/sca/what-static-analysis

Raddatz, M., & Martinez, L. (2022, May 19). *Inside the Submarine Capable of Launching Nuclear Missiles*. ABC News. Retrieved September 20, 2022, from https://abcnews.go.com/Politics/inside-submarine-capable-launching-nuclear-missiles/story?id=84832056#:~:text=Each%20missile%20is%20capable%20of,by%20the%20New%20START%20Treaty.

Stoneman, E. (2020). *Learn Docker in a month of lunches*. Manning Publications Co.