



Securing container-based environments with Anchore

Veera Laurikainen

Bachelor's thesis

October 2022

Information and Communication Technologies

Laurikainen, Veera

Securing container-based environments with Anchore

Jyväskylä: JAMK University of Applied Sciences, October 2022, 85 pages.

Technology, Communication and Transport. Degree Programme in Information and Communication Technologies. Bachelor's thesis.

Permission for web publication: Yes

Language of publication: English

Abstract

In software development third-party libraries and open-source components are being used even more. Entities created from these are called artifacts. Utilization of these pieces of software makes development easier and faster for developers but not always safer. Making sure everything is done in a secure way and the importance of security checks in CI/CD pipelines has been growing. Adding security scanners as part of CI/CD pipeline can help mitigating security issues that arise during software development related to used components. This reduces manual work of the team responsible for securing the system and gives more time for reacting to threats.

Research and implementation were done for Qvantel Oy. The objective of the research was to get familiar with the used technologies and deploying scanner tool that monitors security of artifacts as part of the client's QRP production environment. Required feature for scanner was easily readable results and possibility to configure them. Created results are utilized by the security team. During research it was investigated if deployed artifact scanning coverage is broad enough or if manual scanning is needed as well. Another target of evaluation was the most beneficial spot in the CI/CD pipeline for the scanner.

Implementation consisted of two parts: a local implementation that was done to test the used software, compatibility, and to show parts of installing and configuring Jenkins CI/CD tool that could not be done in the production environment due to permission restrictions. Local implementation worked as a basis for the production environment implementation. Implementation consisted of an artifact scanner that is triggered via Jenkins server and a Jenkins plugin that is part of a CI/CD pipeline. Artifact scanner is working on top of Docker in an AWS instance in cloud environment.

Constructive research method was utilized to solve the thesis research problem and the result was a working Docker container image artifact scanning process as part of the existing QRP CI/CD pipeline. Artifact scanning can be triggered from Jenkins CI/CD tool either manually or automatically. Manual version works by giving image information as parameters and automatic version is triggered when new QRP "system-spec" image is pushed into binary repository. Proposal for further development is comparing results of existing Xray scanner and the deployed Anchore Engine to get even broader view of the found vulnerabilities. Anchore policy should be configured further making it more precise depending on what image is scanned.

Keywords/tags (subjects)

Cybersecurity, Anchore Engine, Container, Docker, Automation, Artifact, DevSecOps, Software development, Software Composition Analysis, Jenkins

Miscellaneous (Confidential information)

Laurikainen, Veera

Konttipohjaisten ympäristöjen suojaaminen Anchoren avulla

Jyväskylä: Jyväskylän ammattikorkeakoulu, Lokakuu 2022, 85 sivua.

Tietojenkäsittely ja tietoliikenne. Tieto- ja viestintäteknikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisulupa avoimessa verkossa: Kyllä

Julkaisun kieli: englanti

Tiivistelmä

Ohjelmistotuotannossa käytetään kolmannen osapuolen kirjastoja ja avoimen lähdekoodin komponentteja yhä enemmän. Näistä syntyviä kokonaisuuksia kutsutaan artefakteiksi. Näiden ohjelmiston palasten hyödyntäminen tekee kehittämisestä helpompaa ja nopeampaa kehittäjille, mutta ei aina turvallisempaa. Tietoturvallisen kehittämisen varmistaminen ja turvallisuustarkastuksien tärkeys CI/CD-putkessa ovat kasvussa. Tietoturvakannereiden lisääminen osaksi CI/CD-putkea voi auttaa pienentämään tietoturvaongelmia, jotka nousevat ohjelmistokehityksen aikana liittyen käytettyihin komponentteihin. Tämä pienentää tietoturvasta vastaavan tiimin manuaalista työtä ja antaa enemmän aikaa reagoida uhkiin.

Tutkimus ja toteutus tehtiin Qvantel Oy:lle. Tutkimuksen tavoitteena oli tutustua käytettäviin teknologioihin ja ottaa käyttöön skannerityökalu artefaktien tietoturvan valvontaan osana toimeksiantajan QRP-tuotantoympäristöä. Haluttu ominaisuus skannerille oli helposti luettavat tulokset ja mahdollisuus muokata niitä. Tietoturvatyö hyödyntää luodut tulokset. Työn aikana oli tutkittava, oliko käyttöön otetun artefaktiskannerin kattavuus tarpeeksi laaja vai tarvitaanko lisäksi manuaalista skannausta. Toinen arvioitava kohde oli skannerin parhaiten hyötyä antava sijainti CI/CD-putkessa.

Toteutus koostui kahdesta osasta: lokaali toteutus, jolla pystyttiin testaamaan käytettävät ohjelmat, niiden yhteensopivuus ja näyttämään osa Jenkins CI/CD-työkalun asennuksesta ja konfiguroinnista, jota ei käyttöoikeusrajoitusten takia pystytty tekemään tuotantototeutuksen osalta. Lokaali toteutus toimi pohjana tuotantoympäristön toteutukselle. Toteutus koostui artefaktiskannerista, joka käynnistetään Jenkins serverin ja Jenkins pluginin avulla toimien osana CI/CD-putkea. Artefaktiskanneri toimii Dockerin päällä AWS-instanssissa pilviympäristössä.

Opinnäytetyön tutkimusongelman ratkaisuun käytettiin konstruktiivista tutkimusmenetelmää ja tuloksena saatiin toimiva Docker konttien kuvien artefaktiskannaus prosessi osana olemassa olevaa QRP CI/CD-putkea. Artefaktiskannaus voidaan käynnistää Jenkins CI/CD-työkalusta joko manuaalisesti tai automaattisesti. Manuaalinen versio toimii antamalla kuvan tiedot parametreina ja automaattinen versio käynnistyy, kun uusi QRP "system-spec" kuva lisätään binaariseen säilytyspaikkaan. Jatkokehityksenä ehdotetaan vertailemaan olemassa olevan Xray skannerin tuloksia käyttöön otetun Anchore Engine:n tuloksien kanssa, jotta saataisiin vielä laajempi kuva löydetyistä haavoittuvuuksista. Anchore:n menettelytapaa pitäisi kehittää pidemmälle, joka tekee siitä vielä tarkemman riippuen mitä kuvaa skannataan.

Avainsanat (asiasanat)

Tietoturva, Anchore Engine, Kontti, Docker, Automaatio, Artefakti, DevSecOps, Ohjelmistotuotanto, Ohjelmisto kompositio analyysi, Jenkins

Muut tiedot (salassa pidettävät liitteet)

Contents

Abbreviations	8
1 Introduction	9
1.1 Background and goals	9
1.2 Client	9
2 Research layout	10
2.1 Research problem and questions.....	10
2.2 Research methods.....	11
3 Theory.....	12
3.1 Software Engineering	13
3.1.1 DevOps.....	15
3.1.2 DevSecOps	17
3.1.3 Cloud Computing Platform	17
3.1.4 Version control	18
3.1.5 Concept of Binary repository manager	21
3.1.6 Continuous Integration, Delivery, and Deployment (CI/CD)	22
3.1.7 Concept of artifact in software development	23
3.1.8 Kubernetes - Container Orchestration System.....	24
3.1.9 Docker - Containerization Platform	25
3.1.10 Artifactory - Binary Repository Tool	27
3.1.11 Jenkins - CI/CD Tool	28
3.1.12 Anchore – Artifact Scanner Tool.....	29
4 Current process environment.....	34
4.1 Artifact Scanners as part of pipeline process.....	37
4.1.1 Different scanner types in software development life cycle and current market situation.....	37
4.1.2 What is wanted to achieve with the artifact scanning?	41
4.1.3 Challenges with artifact scanning tools	41
5 Implementation.....	43
5.1 Local implementation	44
5.1.1 Installation of software.....	45
5.1.2 Configuration of environment and software.....	53
5.1.3 Use of implementation and configuring Anchore policy bundle	59
5.2 Production implementation	65
5.2.1 Installing required software	66

5.2.2	Workflow of implementation	67
5.2.3	QRP image policy bundle and results	68
6	Outcome	69
6.1	Research question one - How to deploy artifact scanning?	69
6.2	Research question two - Artifact scanning coverage?	71
6.3	Research question three - Most beneficial spot for Anchore?	71
7	Conclusion	72
7.1	Advantages, deficiencies, and challenges of the implementation	72
7.2	Ethicalness.....	74
7.3	Utilization of outcome.....	75
7.4	Further Development.....	75
	References	76
	Appendices	82
	Appendix 1. Anchore Engine default policy bundle JSON file.....	82
	Appendix 2. Modified Anchore Engine default policy bundle JSON file for local implementation.....	84
	Appendix 3. QRP policy bundle JSON file.....	87

Figures

Figure 1.	Common software engineering sectors and concepts.	13
Figure 2.	DevOps lifecycle (Gunja 2021).	16
Figure 3.	Centralized Version Control System (Chacon & Straub 2014, 10).	19
Figure 4.	Distributed Version Control System (Chacon & Straub 2014, 11).	20
Figure 5.	Binaries in DevOps loop (DevOps: 8 Reasons for DevOps to use a Binary Repository Manager 2021).	22
Figure 6.	Difference between container and VM (What is a Container? n.d).	27
Figure 7.	JFrog Artifactory acting as single source for all artifacts moving in the DevOps pipeline (Jfrog artifactory n.d).	28
Figure 8.	Anchore's analysis and policy check process (Anchore Engine Overview 2020).	30
Figure 9.	Connecting to Anchore Engine via API or CLI (Accessing the Engine 2020).	31
Figure 10.	Anchore image analysis and analyzer module process (Analyzing Images 2020).	32
Figure 11.	Anchore image analysis status workflow (Image Analysis Process 2020).	32
Figure 12.	Anchore policy evaluation workflow (Policy 2020).	33
Figure 13.	Anchore as part of a CI/CD pipeline (CI / CD Integration 2020).	34

Figure 14. Current environment with numbered QRP pipeline phases.	36
Figure 15. Overview of the current market situation for security and compliance tools (CNCF Cloud Native Interactive Landscape n.d).	40
Figure 16. Anchore scanning automatic and manual workflows.	44
Figure 17. Local implementation deployment diagram.	45
Figure 18. Installing “yum-utils” and setting up docker repository.....	46
Figure 19. Installing Docker Engine, containerd, and Docker Compose.....	46
Figure 20. Enabling and starting Docker service.....	46
Figure 21. Starting Jenkins server container.....	47
Figure 22. Checking Jenkins server container logs for Administrator password.....	47
Figure 23. Starting initial configuration of the Jenkins server.	48
Figure 24. Jenkins initial plugin options and installation of plugins.	49
Figure 25. Creating Jenkins administrator account.	50
Figure 26. Checking Jenkins server container IP address and giving “Jenkins URL”.....	50
Figure 27. Fetching Docker Compose YAML and starting services.....	51
Figure 28. Verifying system status of Anchore Engine services.....	51
Figure 29. Adding image to Anchore Engine.....	52
Figure 30. Anchore Engine image vulnerability check.	52
Figure 31. Image policy evaluation check.	53
Figure 32. Checking Anchore Engine network name.....	54
Figure 33. Connecting Jenkins server to “root_default” and disconnecting from “bridge” network.	54
Figure 34. Changing “Jenkins URL” to new one.	55
Figure 35. Installing Anchore scanner plugin.....	55
Figure 36. Configuring Anchore scanner plugin global settings.	56
Figure 37. Jenkins job “Execute shell” build step.	57
Figure 38. Jenkins job “Anchore Container Image Scanner” build step.	58
Figure 39. Jenkins job build results.	58
Figure 40. Jenkins build “Anchore Report” results.	59
Figure 41. “Anchore Report” summary for two wordpress images.	60
Figure 42. Filtering “Anchore Report” results with “Trigger Id”.....	60
Figure 43. Accessing Anchore Engine container with “root” user and listing policies.	61
Figure 44. Fetching Anchore Engine default policy bundle to modify it.	61
Figure 45. Adding modified default policy bundle to Anchore Engine and activating it.....	62
Figure 46. Empty policy bundle JSON.	62

Figure 47. New “Anchore Report” summary after modifications.	64
Figure 48. New “Anchore Report” results after modifications.....	65
Figure 49. Production implementation deployment diagram.....	66
Figure 50. Building Anchore job manually with parameters.	67
Figure 51. Parameterized Anchore echo command.	68
Figure 52. Default policy bundle evaluation against QRP system-spec image.....	68
Figure 53. QRP policy bundle evaluation against QRP system-spec image.....	69

Abbreviations

API	Application Programming Interface
App	Application
AWS	Amazon Web Services
BSS	Business Support System
CD	Continuous Delivery, Continuous Deployment
CI	Continuous Integration
CLI	Command Line Interface
CSP	Communication Service Provider
CTE	Component Test Environment
CVSS	Common Vulnerability Scoring System
DAST	Dynamic application security testing
GB	Gigabyte
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
IAST	Interactive Application Security Testing
IP	Internet Protocol
ITE	Integration Test Environment
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
OS	Operating System
PaaS	Platform as a Service
PTE	Performance Test Environment
QRP	Qvantel Reference Product
RAM	Random Access Memory
REST	Representational State Transfer
SaaS	Software as a Service
SAST	Static Application Security Testing
SCA	Software Composition Analysis
SIT	System Integration Test
SSO	Single sign-on
URL	Uniform Resource Locator
UTE	Upgrade Test Environment
VM	Virtual Machine
VCS	Version Control System
YAML	YAML Ain't Markup Language

1 Introduction

1.1 Background and goals

Nowadays more businesses and services work on the internet, this means there is a lot more information moving from one place to another including business secrets and other sensitive data. New applications are created on a daily basis and the development needs to be fast and agile. Modern applications are a combination of third-party and open-source components that consist of different artifacts (Springett n.d). This means great amount of the applications functions are made by someone else and there might not always be a clear view of what the origin is and what is included.

It is common that the first phase of a network attack includes system scanning for vulnerabilities that can be exploited. It is important to close security gaps related to these. As human errors are prone to happen one way to reduce security risk is implementing a security scanner that finds vulnerabilities in used external components. (What Is a Vulnerability Scan, and Why Is It Important? n.d.)

The purpose of this research is to deploy a new security scanner Anchore to the client's Qvantel Reference Product (QRP) development pipeline and configure it to show selected vulnerabilities and results. There is a need to implement a new scanner with better features, Anchore will be added to enhance the current security scanning and to give results in a more readable way. The scope and functionality of the scanner will be estimated with pros and cons. The current environment will be used as a comparison.

1.2 Client

Qvantel Oy was founded in 2006 and the headquarters is in Helsinki. Revenue for Qvantel Oy in December 2020 was 86 032 000 euros. (Qvantel Oy n.d.) There are around 501 to 1000 people working for Qvantel Oy and there are office premises in over 20 places world-wide. Qvantel provides Digital Business Support System (BSS) for customers. Qvantel Flex BSS is a no-code and cloud-native solution that helps communication service providers (CSPs) to accelerate their movement to digital-first companies. End-to-End solutions contains mobile apps, product catalogs with

efficient management, configurable tool-suites for daily sales and customer care, Business-to-Business sales customer relationship management, Qvantel Flex framework-based order management, and billing management. Current customers include telecom groups, CSPs, mobile virtual network enablers, mobile virtual network operators, and digital-first sub-brands. Qvantel delivers services for over 230 million telecoms customers including mobile, fixed, and TV services. (Qvantel Flex BSS n.d; Qvantel LinkedIn n.d.)

As Qvantel is providing tools and solutions for other customers it is important to ensure that the products are created in a safe environment. Vulnerabilities should be caught early on before the product goes to the customer's environment. Research is done to improve this process by adding Anchore as an additional scanner.

2 Research layout

2.1 Research problem and questions

The thesis research problem consists of how to deploy Anchore security scanner to the current development pipeline in a way it is convenient to use and does not hinder the current software development lifecycle. Scanning is manually triggered or an automated process that is recurring. Another problem is that current scanner JFrogs Xray gives a large number of results making them hard to analyze. Anchore should be configured so that results are easy to analyze. It will be questioned if Anchore gives enough information or if manual scanning is needed in addition. The best position for Anchore will be considered with the thought that it will catch as many vulnerabilities as possible, earliest as possible.

The objective of the thesis is to answer the following research questions, they are all created from the presented research problems. Chosen questions to be answered:

- 1) How to deploy artifact scanning as a part of a CI/CD pipeline?
- 2) Is the artifact scanning coverage enough or is manual scanning needed?
- 3) What is the most beneficial spot for Anchore in a CI/CD pipeline?

There were functional requirements determined for the first research question. All these requirements were discussed with the representative of the security team. Chosen functional requirements are as follows:

FR1) Jenkins has access to Anchore Engine.

FR2) Anchore works via Jenkins Anchore plugin.

FR3) Anchore Engine can retrieve docker images from Artifactory.

FR4) Anchore policies can be configured.

FR5) Vulnerabilities can be whitelisted with Anchore.

FR6) Anchore gives scanning results as Anchore Report in Jenkins.

FR7) Anchore scanning can be started manually from Jenkins.

FR8) Anchore scanning is started automatically when a new image is pushed to Artifactory.

2.2 Research methods

In the thesis, the research method best suitable to answer the research questions will be chosen. As Kananen (2015) describes, in some studies only one methodology is not sufficient and multimethodological methodologies need to be used. This group includes case study, design-based research, and process-oriented research. These methods have elements from both qualitative and quantitative research methods. (Kananen 2015, 75-79.) As the object of this research is to deploy a security scanner to the client's environment the most suitable multimethodological option was constructive research method.

Constructive research methodology attempts to solve real life problems and provide contribution this way. Main idea is to create construction that can be a software, a plan, or anything that can be created and developed. By developing a construction something new is created. For example, mathematical algorithms are a genuine example of a construction. Constructive research method has five core traits which are the following: focuses on real life problem that is essential to fix, produces innovative construction that fixes the real-life problem and it can be tested in practice, includes close collaboration between client and researcher where experimental learning happens, is connected to the theoretical base of the study, and reflects empirical findings back to the theoretical research. (Lukka 2001.)

Research is based on the Client's need to deploy a new security scanner to the development pipeline. As Toikko and Rantanen (2009) explain, in design-based methodologies reliability aspect means ultimately the feasibility of the end result. The collected information is not only reliable but also needs to be useful. In terms of development results the feasibility means all things created during research and utilizing those results. (Toikko & Rantanen 2009, 121-125.) In this case working security scanner in the client's environment.

Theoretical research material was chosen to support development of the research and to give background information about the environment where the implementation will be done. Material includes used technologies and tools documentations and previously done research about security scanners and how they function. The objective of the theoretical research is to gain extensive knowledge about the technologies used in client's environment and tools that are deployed. Practical research is done to improve safety in the client's current development pipeline.

3 Theory

The theory section includes background for understanding what software development lifecycle and CI/CD pipeline is and what concepts, technologies, and tools they include that are used in the current project environment. It includes the scanner tool Anchore that will be implemented to the environment and its current features. It explains types of scanners in different parts of the development cycle including scanner type of Anchore and the ones that already exist in the environment.

3.1 Software Engineering

Software engineering is a concept that should be explained through the terms it includes to better comprehend how it works. Software engineering is more complicated than it seems. Making software engineering design for an application is not straightforward and usually takes time. To understand how information system creation works, both engineering and software development should be considered. A collection of code, documentation, triggers that perform a certain task, and meets a given criterion is referred to as software. Engineering is the process of creating products and utilizing the most up-to-date techniques, principles, and methodologies. Combining these two makes what software engineering is about as represented in Figure 1. (What is Software Engineering? n.d.)

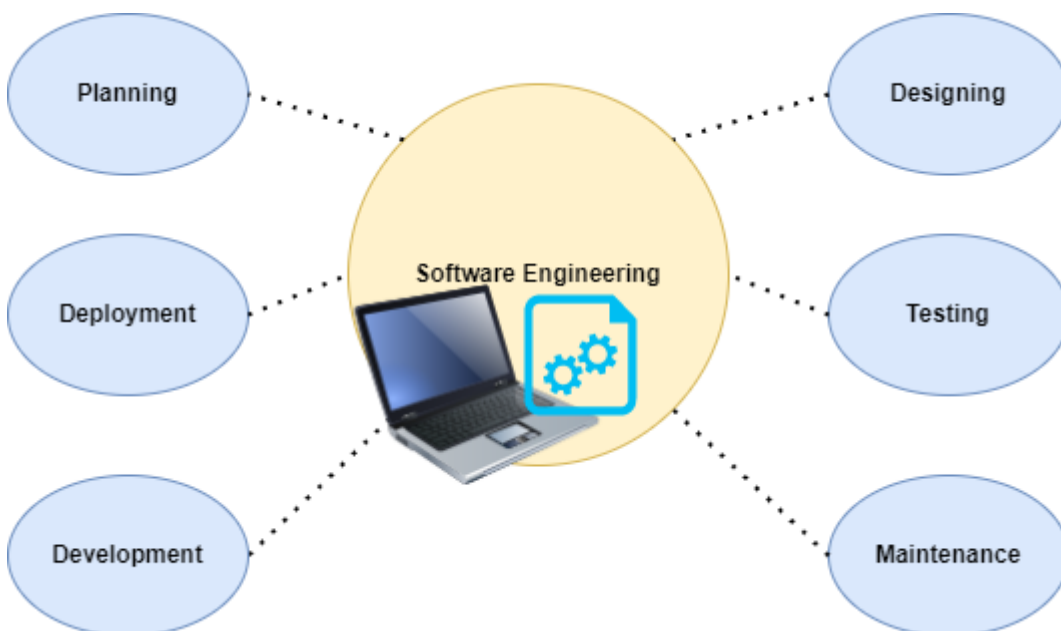


Figure 1. Common software engineering sectors and concepts.

Software engineering is an area that deals with software development. It follows a set of principles, best practices, and procedures that have been fine-tuned through time, evolving as software and technology evolves. As a result, the created product is dependable, efficient, and effective. (What is Software Engineering? n.d.)

The process starts when there is a need for a certain outcome or output from an application. There is a request to create software and to make it work, requirements and steps to follow need

to be prepared. Software engineering tools can help here to make sure that the work is done as required and best practices are being followed. Next a roadmap will be built to break the process into smaller pieces. Making it easier for developers to keep up with the project. After planning has been made it is time to start coding. In many cases, this is the most time-consuming element of the process because the code must be compatible with existing systems and languages. Sadly, these issues are sometimes not discovered until much later in the process, necessitating revision. As the code is being created it should be constantly tested even after it is completed throughout the life cycle. Continuous monitoring and testing can be done with software engineering tools.

(What is Software Engineering? n.d.)

Before the product is even created, the process of software engineering begins. The fundamentals of software engineering state that it should continue long after the "task" has been accomplished. This so-called sustainability can refer to a far larger range of issues, including economic, social, and environmental sustainability. Most important is to understand what is required from the software, what it does, in what kind of environment it must work and all security aspects that need to be reviewed. Security is one of the key elements of development because it is essential to all aspects. Without the help of tools like software engineering it is easy to get lost. (Lago 2019, 61-64; What is Software Engineering? n.d.)

The fundamentals of software engineering design entail writing computer and system instructions. A lot of this will be done by professionals with extensive knowledge at the coding level. Process is not always as linear as this, meaning it needs a lot of confirmation even after it is complete. Some impacts emerge after the product is being used over time. (Lago 2019, 61-64; What is Software Engineering? n.d.)

Not everything needs software engineering but because of the high-risk information companies store and the security concerns they pose, almost every company needs software engineering. Software engineering aids in the development of tailored, individualized software that identifies weaknesses and dangers before they occur. To accomplish this security should be implemented from the very beginning. In a case where a system is created first and corrected after is not how security should be handled. Even if the software engineering policy of safety are not needed, they

can assist in saving money and improve customer satisfaction. (El Rhaffari & Roudies 2013, 255; What is Software Engineering? n.d.)

Software engineering is used at every stage of the software development process. It has different levels, there is operational, transitional and maintenance software engineering. On operational level focus is on how the software interacts with the system, what is the focus security, budget, functionality, or some other aspect. Transitional level focuses on how it will respond when it is moved from one environment to another. The development process usually necessitates some scalability or flexibility. Recurrent software engineering referred to as maintenance verifies how the software functions inside the existing system while being changed. (What is Software Engineering? n.d.)

3.1.1 DevOps

History of DevOps

Software development lifecycle was originally invention in 1970's by developers and is now called Waterfall Methodology. Waterfall methodology uses separate phases, and each comes after another starting from the beginning till the end of a project. There is no way to see how the product will end up as there is no overlapping of working and no time to save on development. The method was not structured to go back to a prior phase, problems were not discovered until late at the end of the so-called waterfall and were highly expensive to fix. This way of working created separate work effort and requires sharing information between teams and joint effort was not a possibility. (Sharma 2018a; What is DevOps? Waterfall to DevOps 2.0 2018.)

As waterfall methodology could not support businesses needs and was inflexible and costly in 2001 Agile Manifesto was written. After that Agile became popularly used. Agile contains steps of analyzing, developing, testing, implementing, and managing software through the lifecycle. Even though the steps have not changed as much, it was important that teams were working more closely together, and steps were no longer one way. Feedback loops were created that allowed information to flow between different teams within the lifecycle. Nevertheless, the software test-

ing and operations teams, who continued to work in their original silos and with their own approaches, were not cooperating with the Agile development teams. (Sharma 2018b; What is DevOps? Waterfall to DevOps 2.0 2018.)

DevOps

When practitioners began to consider how they could convert their businesses to a paradigm that not only enabled but also encouraged strong collaboration between developers and operations team, they came up with DevOps. DevOps is a set of cultural concepts, practices, and technologies that improves businesses capacity to produce fast paced applications and services, allowing it to evolve and improve products at a faster rate than traditional software development and infrastructure management methods. Many firms were enticed to adopt a DevOps culture by the prospect of drastically reduced cycle times. Cycle time refers to the time it takes for a concept to be realized as software in a production environment. In DevOps model developers and operations team are not as divided. These teams are combined into a single unit where the engineers work in all parts of the software lifecycle, from development and testing to deployment and operations, and develop a diverse set of abilities that are not limited to a particular role, as seen in Figure 2. (Dörnenburg 2018, 73-74; What is DevOps? n.d.)

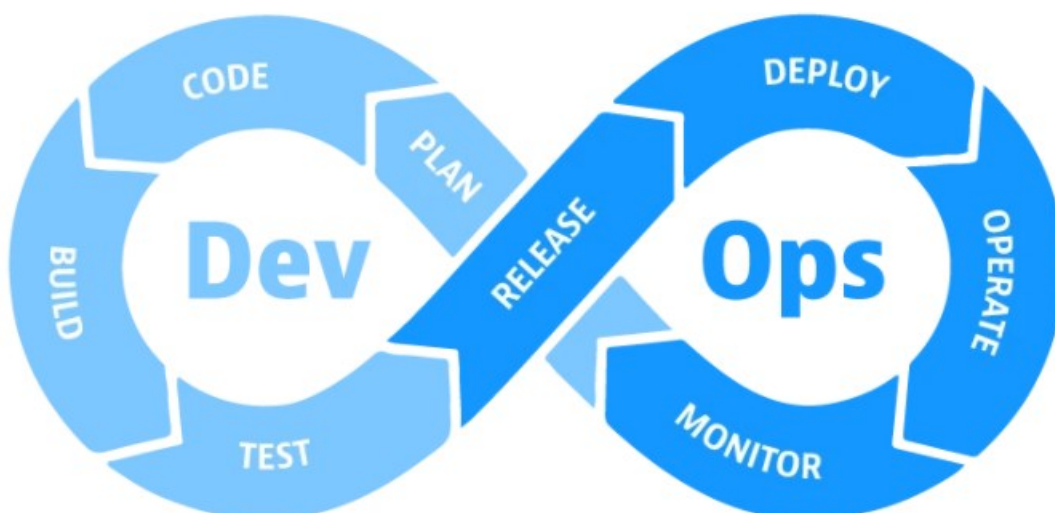


Figure 2. DevOps lifecycle (Gunja 2021).

3.1.2 DevSecOps

When comparing DevOps to DevSecOps in addition to development and operations also security is added as a responsibility across the software development lifecycle hence the name DevSecOps. Before the security role was handled by a separate security team and mostly done at the end of the development cycle. As software development is getting more agile this is not a valid way of handling security anymore. Instead, application and infrastructure security are seamlessly integrated into DevOps processes and tools using DevSecOps. It takes care of security vulnerabilities when they arise, when they are faster, easier, and cost less to repair, and before they are deployed to production environment. (DevSecOps 2020; What is DevSecOps? 2018.)

DevSecOps' two key advantages are speed and security. Development teams produce proper, more secure code in a prompt manner and hence at a lower cost. Securing software starts at the beginning of the development lifecycle when cybersecurity processes are introduced. The software code is checked, scanned, audited, and tested for security vulnerabilities throughout the lifecycle. As scanning and patching is introduced to the development cycle the ability to find and fix common vulnerabilities and chance of exposures to vulnerabilities is decreased. Furthermore, increased collaboration across development, security, and operations teams enhances businesses response time to incidents and problems. (DevSecOps 2020; What is DevSecOps? 2018.)

3.1.3 Cloud Computing Platform

Instead of having a physical data center and servers, cloud provides storage, databases, computing power, networking, software, and analytics via the Internet. Using cloud, you only pay for resources that have been used called pay-as-you-go pricing aiding with lowering operating costs. Resources that cloud computing provides help with running environments more efficiently and with flexible resources. (What is cloud computing? n.d.a; What is cloud computing? n.d.b.) There are many cloud providers like Amazon Web Services, Microsoft Azure, and Google Cloud Platform. In this project Amazon Web Services is used.

Most cloud computing can be incorporated into one of three main varieties: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). IaaS has the most basic features of cloud computing. It provides servers, virtual or hardware machines, networking

features and storage. IaaS allows you to have the most freedom and control over the infrastructure. PaaS eliminates the need to handle the underlying infrastructure, supplying an on-demand environment ready to be developed. This allows developers to be more productive focusing only on deployment, testing, and managing the application. Developers do not have to think about resourcing, capacity, patching, or any other time-consuming task that comes with maintaining an application. SaaS means delivering software via the Internet on demand against a fee. Cloud provider hosts and administers the software and infrastructure and handles upkeep including security patching and upgrading software. Users must only consider how to use the software. (What is cloud computing? n.d.a; What is cloud computing? n.d.b.)

3.1.4 Version control

Version control, which is also called source control, is a system that tracks and manages a file or multiple files in the course of time so that it is possible to access older versions later. Version control helps developers work faster and more reliably. When developing code, it is important to keep every version and modification, version control system helps with this. Code is saved to a chosen database, and it allows you to turn back the time of selected files, taking them back to previous state. It is even possible to revert an entire project like this or compare older changes to newer ones. Systems like this help minimize disruption of other teammates and it can be seen who last modified the code that might be the source of the problem and when this happened. Version control protects you from ruining the source code and losing files. (Chacon & Straub 2014, 8-11; What is version control? n.d.)

There are three different types of version control, local, centralized, and distributed. The local version control system works by copying files into another folder. System like this is very simple and is easily exposed to errors. To help with these problems and improve revision control, developers used a local VCS which means having a simple database storing all modifications to the files. (Chacon & Straub 2014, 8-11.)

Centralized Version Control Systems were developed when people started to encounter problems on how they can collaborate using separate systems. Systems like these work on a single server that stores all the versioned files and clients can take files from that centralized storage like represented in Figure 3. This version has a lot of advantages compared to local version control systems.

There is better knowledge of what everyone is doing on the project. Permissions can be added to control what each person can do instead of having a database on each client locally. The downside is that it has the possibility for a single point of failure if the server goes down nobody can collaborate or save their current work. There is also risk of data corruption and if there is no backup all the code will be lost excluding snapshots that people might have on their local systems. Local version control system has the same problem since it has the project in one place. (Chacon & Straub 2014, 8-11; Version Control Software: An Overview n.d.)

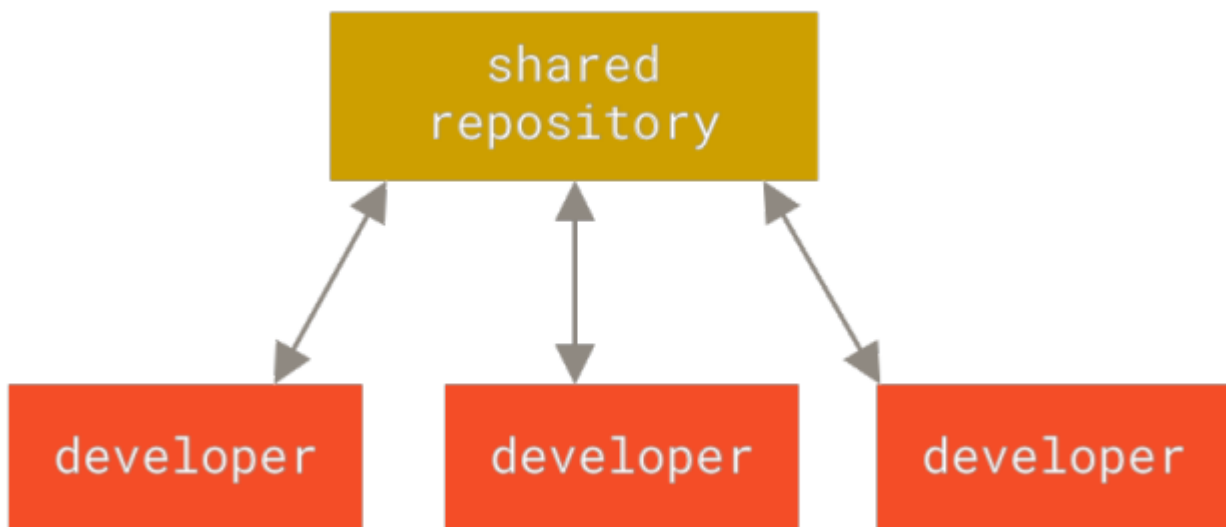


Figure 3. Centralized Version Control System (Chacon & Straub 2014, 10).

With Distributed Version Control Systems, the whole code repository is copied to client machines with the full history instead of checking out the latest snapshot of files as seen in Figure 4. Now if the server breaks and collaboration was done via this server any of the client machines can restore the repository back to the server. All the client machines have a full backup of the data. It is also possible to collaborate on multiple groups and projects simultaneously since this version control system works well with several remote repositories. Several workflows can be set that are not possible in centralized systems, for instance hierarchical models. (Chacon & Straub 2014, 8-11; Version Control Software: An Overview n.d.)

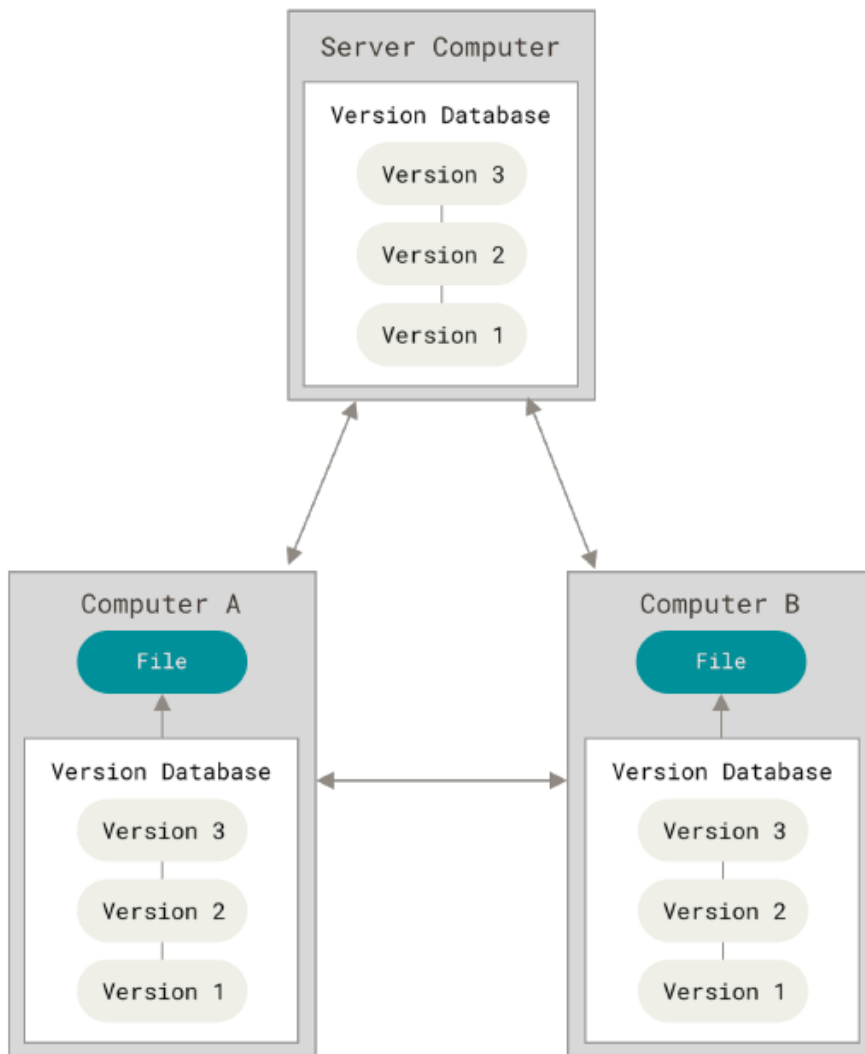


Figure 4. Distributed Version Control System (Chacon & Straub 2014, 11).

Version control has many benefits it gives to a project. As there are multiple people editing code it is important to have insight into the history of the project and changes that have happened in every file. It aids in finding bugs and conflicts between developers. It can help fixing problems in older versions of the application or code can be reverted to previous version if the new one is not working. Developers can work concurrently with the use of branches and merging. Branching means keeping multiple streams of work separated from each other and with merging you can bring things back together. Each branch can for example have one feature and by merging the features can be brought back to the main working stream one by one. These properties make projects very traceable and easy to follow. (Version Control Software: An Overview n.d; What is version control? n.d.)

It is good to understand that version control system is different from a hosting service such as Bitbucket or GitHub. The version control system is on your computer and the hosting service is a place to store all the project source code in some server or cloud. These 3rd party hosting solutions can offer extensions and other services. These extensions can help with task tracking and documentation that are connected to the VCS hosting service. Performance can be tracked giving insight to the speed and efficiency of the project development. Automation can be constructed through these external integrations like automated builds and tests. (Version Control Software: An Overview n.d.)

3.1.5 Concept of Binary repository manager

Binary repositories store binary artifacts and work similarly to source code repositories. A binary repository can be represented as a library. One central place, a server that distributes, fetch, and stores files, these files can be called artifacts. Artifacts can be thought of as binaries and dependencies and for instance, a code library can be called one type of artifact. (DevOps: 8 Reasons for DevOps to use a Binary Repository Manager 2021; OBrien 2010.)

When creating software, you frequently rely on third-party libraries. The main work of a binary repository manager is to retrieve and cache artifacts from third-party repositories. In a more intricate program, there might be hundreds of libraries from external sources. When the library is used it will be queried from the local repository manager and if it is not found it will be fetched from an external source. Running an application is a combination of many binaries from many sources and the quality of those binaries determines the quality of the end product that the client uses. As seen from Figure 5 source code is important, but The DevOps cycle is focused on the binaries that are built, tested, deployed, and run rather than on the code itself. Examples of binary repository managers are JFrogs Artifactory and Sonatypes Nexus. (DevOps: 8 Reasons for DevOps to use a Binary Repository Manager 2021; OBrien 2010.)

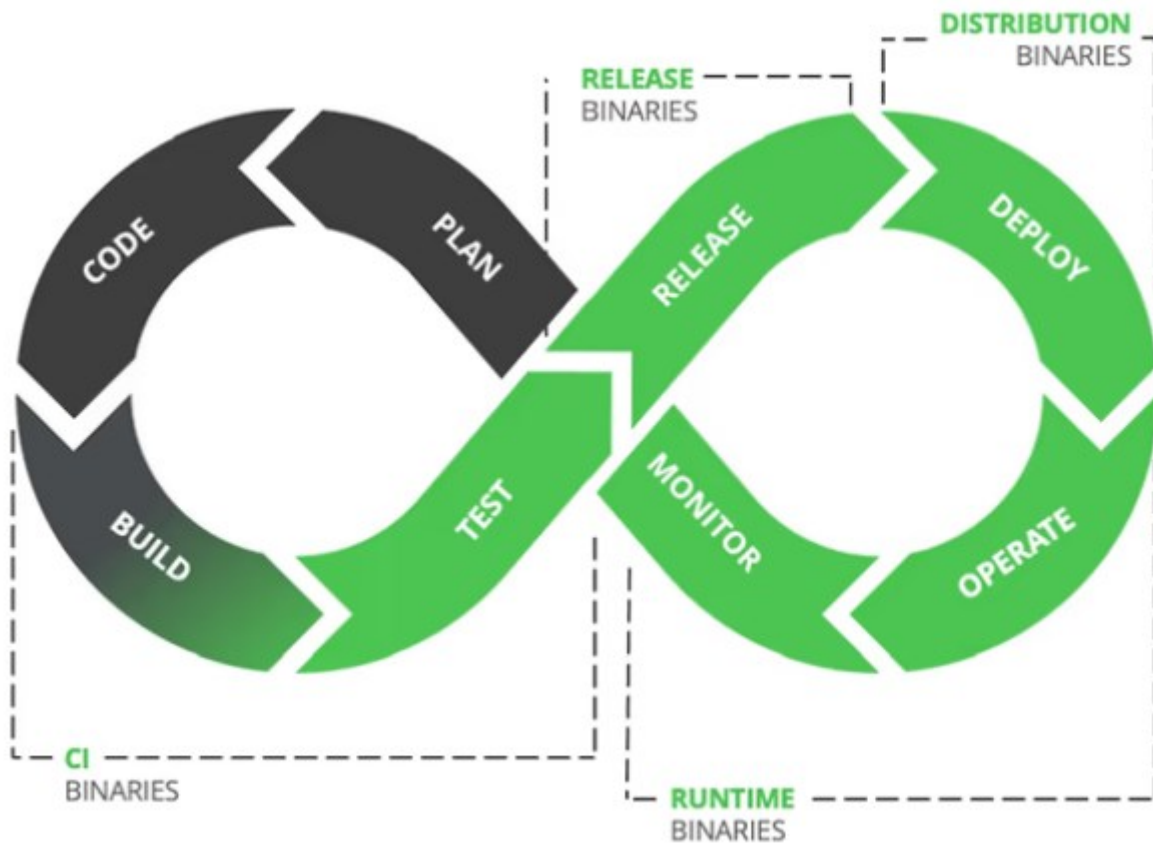


Figure 5. Binaries in DevOps loop (DevOps: 8 Reasons for DevOps to use a Binary Repository Manager 2021).

3.1.6 Continuous Integration, Delivery, and Deployment (CI/CD)

Continuous integration, delivery, and deployment are strategies that aim to minimize feedback loops and automate repetitive operations to make the software release process faster. The agile philosophy of continuously delivering valuable, properly functioning software to the end user is made possible by these techniques. (Continuous Integration vs. Delivery vs. Deployment n.d.) Changing from manual to highly automated development process can take a long time, nevertheless it almost always exceeds the challenges that arise on the way. It is important to make the solution while keeping in mind the business goals that are unique for every situation. (Labourdy 2021, 111-121.) The finesse of constructing a CI/CD pipeline is that it can be represented in individual steps, each building on the previous (Continuous Integration vs. Delivery vs. Deployment n.d.).

Continuous integration (CI) is the process of maintaining a centralized and shared code repository and routing all modifications and features via a sophisticated pipeline that confirms the new changes build correctly and passes all tests before leading them into the central repository. The practical CI process is a combination of the use of source control, committing code on a regular basis, building after every commit, and automating tests. Combining all of these gives regular reliable feedback on the process of the code. (Continuous Integration vs. Delivery vs. Deployment n.d; Labouardy 2021, 80-88.)

Continuous delivery (CD) is the next step from continuous integration. Each successful build that passes through all phases and tests of your CI pipeline is deployed to your preproduction environment automatically. By utilizing CD there is no need to decide what will be deployed. At this stage there can be additional tests done by security and QA team. Continuous deployment is the last step of the pipeline and means the build has successfully passed all the previous steps of the pipeline and is deployed to the production environment. This means that any modification to the program gets distributed to your end users promptly and only after it passes all tests. (Continuous Integration vs. Delivery vs. Deployment n.d; Labouardy 2021, 99-106.)

3.1.7 Concept of artifact in software development

When building software, it is not only source code but also consists of a combination of different artifacts. Fundamentally, artifacts are the skeleton of any piece of software. Artifacts related to software development consist of source code, dependencies, documentation, container images, the compiled application, and anything that is a by-product of a software. When thinking about software development, artifacts have a broad meaning. Most software requires multiple artifacts to be executed. Artifacts can be divided into ones that describe how the software is supposed to work and ones that enable it to run. (DevOps: 8 Reasons for DevOps to use a Binary Repository Manager 2021; What Is an Artifact? Everything You Need to Know 2020.)

The metadata you collect about your artifacts is critical for reusing code and enhancing your build process. They aid other programmers in understanding the thought process that goes into creating software. Furthermore, it helps programmers make decisions and comprehend what is the best way to proceed. (Ben-Zvi 2021; What Is an Artifact? Everything You Need to Know 2020.)

Usually before starting the software development process, the team will be listing artifacts required by the software before any coding. Source code, risk assessment, blueprints, and use cases are examples of these. From the beginning it is important aspects of the software development life cycle to gather all artifacts. The development team can begin developing and constructing the actual program once the basic artifacts have been gathered. Throughout the process artifacts might be developed and pieces like end-user agreements come after the software is finished. Before compiling and sending the program to end users these artifacts can be added in. Typically, software artifacts are kept in an artifact repository like JFrogs Artifactory. (Ben-Zvi 2021; What Is an Artifact? Everything You Need to Know 2020.)

3.1.8 Kubernetes - Container Orchestration System

Kubernetes is an open-source container orchestration platform for managing configuration and automation of containerized services and workloads. Kubernetes can automate various manual processes that are part of deploying, administering, and scaling containerized applications. Kubernetes groups containers that as a combination make an application and organize them into comprehensible units for easier administering and discovery. (Kubernetes n.d; What is Kubernetes? 2020; What is Kubernetes? 2021.)

Containers in Kubernetes are placed into pods to be able to run them on nodes. Kubernetes nodes are made of virtual or physical machines. Every node is administered by the control plane. Control planes include services needed to run pods. (Nodes 2022.) Control plane is a container orchestration layer that controls the lifecycle of containers used via API and different interfaces (Glossary 2021).

Kubernetes cluster is made of nodes. Packaging applications with all dependencies and needed services makes them lightweight and resilient compared to virtual machines. Kubernetes clusters make it possible to develop, move, and administer applications with ease. Clusters can be run across various environments like physical, virtual, and cloud based. Kubernetes clusters have one master node and several worker nodes. Master node controls the state of the worker nodes, for instance managing which applications are running and their equivalent container images. The master node is responsible for all task assignments, for example, administering the state of the cluster,

scheduling applications, and implementing updates. For Kubernetes cluster to be operational at least one master node and worked node is needed. (What is a Kubernetes cluster? n.d.)

Kubernetes is practical for managing containers and ensures there is no downtime. Starting and stopping containers can be handled automatically via the system that Kubernetes provides. It handles scaling and failover for your application, as well as providing deployment patterns and other features. (What is Kubernetes? 2021.)

3.1.9 Docker - Containerization Platform

Docker is an open-source virtualization technology. Its main purpose is to make developing, sharing and running applications easier and faster. It gives developers the ability to package applications and separate them from infrastructure, making delivering software effortless. All this is possible due to containers that are a combination of application source code with needed operating system libraries and dependencies making a standard unit of software. Containers can be created without Docker but using it as a toolkit enables developers to build, run and stop containers using simple commands. Ultimately, time saving automation and easy to command through one API. Nowadays app development entails far more than just creating code. Docker streamlines and speeds up workflow from writing code to running it in a production environment. Developers have the chance to use different architectures, languages, and frameworks unrestricted without fear of causing problems in interfaces between applications and their lifecycle stage stays simpler. Use of containers has risen enormously when moving towards cloud-native development. (Docker 2021; Why Docker? n.d.)

Docker containers and images

Containers are executable instances of images. A Docker container image is a lightweight, standalone, read-only template with commands for forming a container. Images can be based on other images, most used is taking a ready operating system image like Ubuntu and installing other needed components to run your application, for example a web server and suitable configurations. Images can be created from the beginning or using published ones from a registry. Registry is made for storing docker images. Docker Hub is a public registry provided by Docker, but making a private registry is also an option. Docker images can be pulled and pushed to the reg-

istry making it easy to modify and share. (Docker overview n.d.) On runtime docker images become docker containers, docker uses Docker Engine to do this. Docker Engine enables containerized apps to run consistently anywhere in any environment. (What is a Container? n.d.)

Dockerfile

Dockerfiles are used to create docker images. You create your own image by writing a Dockerfile with a simple syntax for outlining the procedures required to create and execute the image. Each Dockerfile instruction forms a layer in the image. Only the layers that have changed are rebuilt when you edit the Dockerfile and rebuild the image. When compared to other virtualization technologies, this is part of what makes images so light, small, and quick. (Docker overview n.d.)

Difference between container and VM

Virtualization is the process of using software to create an abstraction layer on top of computer hardware that enables using computers hardware to be divided into several virtual machines (Containers vs. Virtual Machines (VMs): What's the Difference? 2021). Both containers and virtual machines can isolate resources and are good for allocating. The main difference is that containers virtualize the operating system and VM's virtualize hardware. (What is a Container? n.d.)

Containers work on the app layer packaging code and dependencies needed to run an application or microservice. Containers do not use hypervisor which makes them faster, there is no need to run an entire guest OS in every instance. This allows them to run effortlessly in almost any environment. One machine can contain multiple running containers because operating systems kernel can be shared, and each container can be executed as a separate process in user space. Docker uses two Linux kernel technologies for separating processes and resources, namespaces for processes and control groups for resource isolation. (What is a Container? n.d.)

When using virtual machines, it is possible to turn computers hardware into multiple servers. This is possible by using hypervisor software which abstracts computer software from its hardware and handles requests between virtual and physical resources. Virtual machines must virtualize the entire operating system, taking up a lot of space. Instead of using minimum resources, virtual ma-

chines provide the whole runtime environment of the app. Figure 6 illustrates the difference between the abstraction of containers and VMs. (What is a Container? n.d; What is a Hypervisor? n.d.)

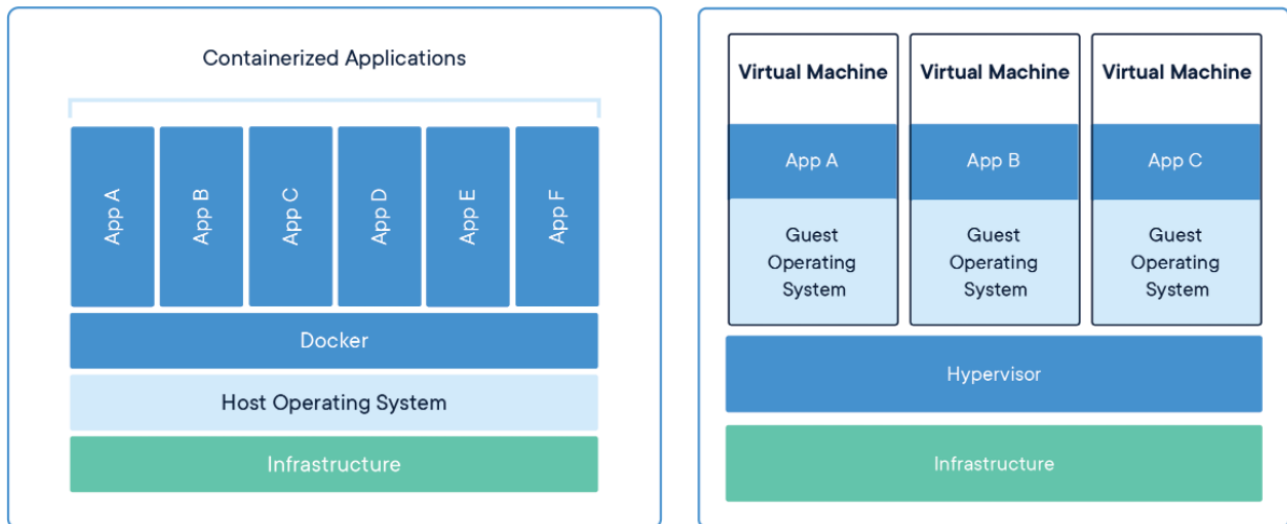


Figure 6. Difference between container and VM (What is a Container? n.d).

3.1.10 Artifactory - Binary Repository Tool

JFrog Artifactory is an artifact repository providing end-to-end artifact lifecycle as seen in Figure 7 that improves maintaining consistency in your CI/CD workflow supporting various software package management systems, all notable CI/CD platforms, and DevOps tools. It is both a source for artifacts needed for a build and a destination for artifacts generated during the build process. Artifactory comes with fully customizable CLI and REST APIs for the ecosystem. (Atzmony 2021a; Masarwa 2021.)

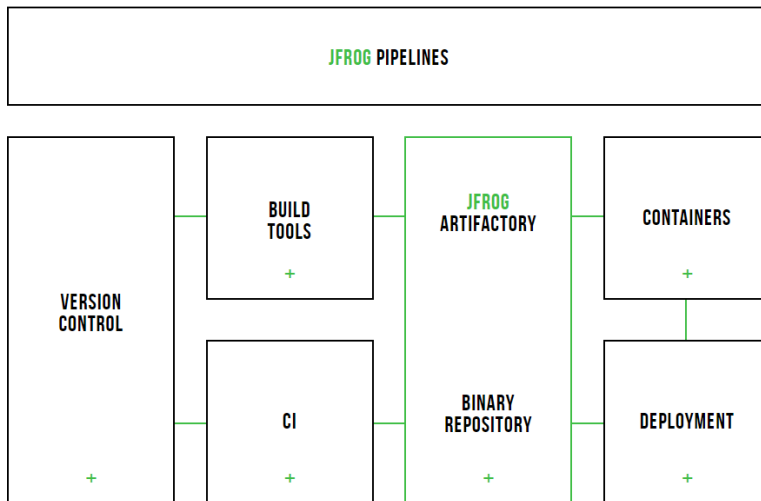


Figure 7. JFrog Artifactory acting as single source for all artifacts moving in the DevOps pipeline (Jfrog artifactory n.d).

Artifacts and third-party components from many sources are frequently used by multiple developers from various sites. This can result in testing issues, slowing down the pace of your releases. For rational, effective software development, a DevOps artifact repository is essential. (Masarwa 2021.)

Artifactory gives multiple benefits for a project. Fully traceable builds that go through CI server coupled with detailed build environment information recorded before deployment, resulting in fully reproducible builds. Searching for artifacts is easy as Artifactory always keeps an eye on your repository's current state. Artifactory lets you move, copy, and delete artifacts, and the accompanying metadata descriptors are promptly and automatically updated to reflect these modifications, allowing sustaining linear and uniform repositories with package clients. (Masarwa 2021.)

3.1.11 Jenkins - CI/CD Tool

Jenkins is an open-source, self-contained, and Java-based automation server. Jenkins can be a trivial CI server or used as a central center for project's continuous delivery. By using Jenkins various tasks related to building, static code analysis, testing, and delivering or deploying software can be

accelerated by automating it. Jenkins is a tool that manages and controls software delivery processes across the whole lifecycle. Jenkins may be configured to monitor any code changes in version control tools like Bitbucket and do an automatic build using build automation tools, for example Gradle. (Jenkins n.d; Jenkins User Documentation n.d; What is Jenkins? n.d.) Jenkins helps developers work in a smooth manner by discarding the pipeline when a problem occurs and that way informing about errors and bugs at an early stage of the building (Kshitiz 2021).

Jenkins has hundreds of plugins which give it a wide range of environments to work in and can be integrated with almost every tool that is used in continuous integration and continuous delivery chain. It is also possible to code your own plugins and use them if no plugin is available. Jenkins is a platform-independent application that runs on practically any operating system because it is written in Java. Jenkins can be easily accessed and configured over a web interface. (Jenkins n.d; Kshitiz 2021.) Container technologies like Docker can be utilized (What is Jenkins? n.d).

3.1.12 Anchore – Artifact Scanner Tool

Anchore Engine is a tool for scanning docker images including static analysis and policy-based checks defined by the user automating inspection, analyzing and evaluation on the given rules. Anchore ensures workload content fulfills the needed criteria, allowing for high trust in container deployments. Policy evaluation checks each image and results in pass or fail depending on the policies defined. This works as an audit mechanism allowing container images properties and content attributes to be evaluated at any chosen moment. (Anchore Engine Overview 2020.)

There are two options of versions of Anchore which are Anchore Engine and Anchore Enterprise. Anchore Engine is an open-source project and Anchore Enterprise is a proprietary commercial product and needs to be paid for. Main difference between these two is that Anchore Enterprise adds more functionality to already existing Anchore Engine. Enterprise version provides graphical user interface, authentication systems like SSO or LDAP, better vulnerability feed data, functions to run Anchore without connection to Internet, reporting service, and integrations to workflow tools like Jira. Anchore engine will be used in this project. (Frequently Asked Questions 2020.)

Anchore works by fetching an image, for instance from a repository, after fetching the image is extracted but not executed. Analyzing of the image starts by extracting as much metadata as possible and results are saved in the database. Policy evaluation is the next step, and this means checking policies against the vulnerabilities found in the artifacts uncovered from the image. Image analysis results and external data for policy evaluations are updated with the latest discoveries, if any changes are found the user will be informed via notice. Updates are checked at designated intervals and the user is informed about changes, ensuring the latest information and evaluations are delivered. Result of the image analysis can be either pass, fail or a warning after all checks like shown in Figure 8. (Anchore Engine Overview 2020.)

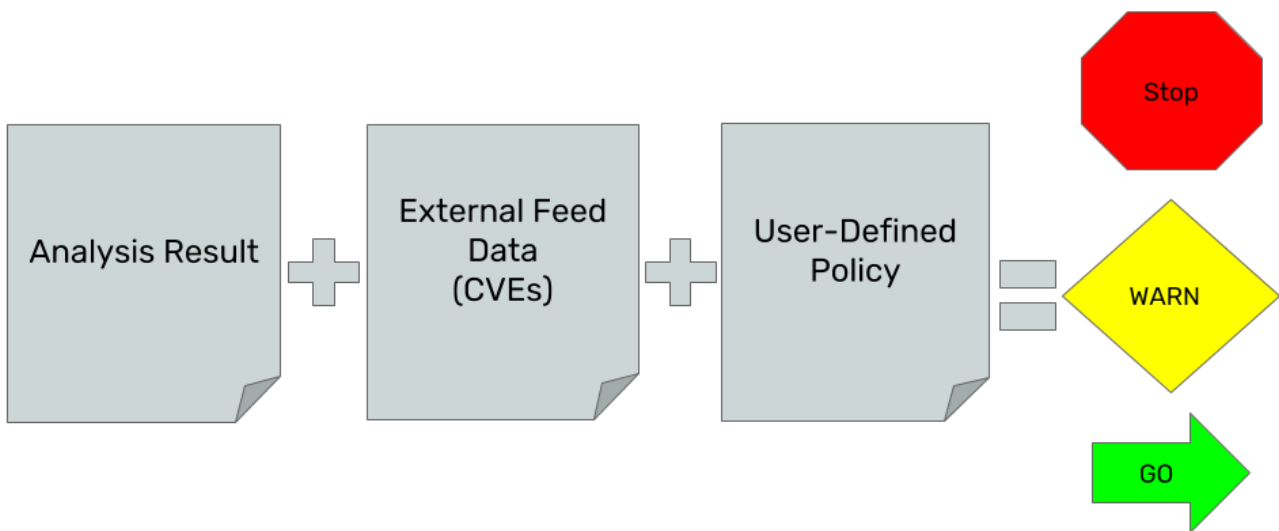


Figure 8. Anchore's analysis and policy check process (Anchore Engine Overview 2020).

Anchore engine is available for use as a Docker container, image is fetched from Docker Hub. Anchore also requires a PostgreSQL database for analyzing operations, results, and unpacking images. Anchore is a combination of six micro-services and can be either used as a single container or divided for handling bigger loads. (Anchore Engine Installation 2020.) Anchore Engine version 1.0.0 and newer is fully integrated with Gype for vulnerability scanning (Gype Integration 2021). For developer to control Anchore Engine Anchore CLI or Anchore Engine REST API is used as seen in Figure 9. With Anchore CLI commands you can pull images from registries, store them in database, and perform scanning and policy evaluations. (Using the Anchore CLI 2020.) Using Anchore API or CLI images can be downloaded from hosted registries like Docker Hub and on-premises registries

like JFrog Artifactory, if registry requires authentication credentials will have to be defined. Images can be analyzed from any Docker V2 compatible registry. (Accessing Registries 2020.)

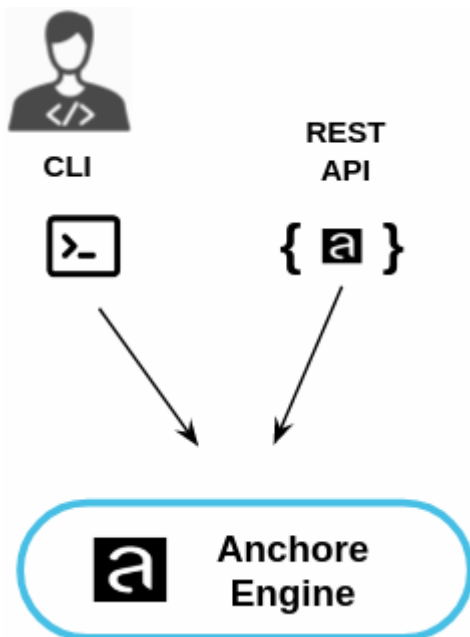


Figure 9. Connecting to Anchore Engine via API or CLI (Accessing the Engine 2020).

Anchore image analysis

During Anchore image analysis every package, library, and files are checked and stored in database. Anchore has multiple analyzer modules that collect data from the image, including image metadata, image layers, file data, python packages, and others as seen in Figure 10. (Analyzing Images 2020.) Image analysis is carried out as a separate, asynchronous, and scheduled process, with analyzer workers polling queues on a regular basis. A minor state-machine is present in image recordings, as seen in Figure 11. (Image Analysis Process 2020.) Tags can be added to Anchore Engine and repository's tagged items will be observed for updates (Analyzing Images 2020). When new updates become available, they are automatically delivered to the analyzers to be inspected via internal queue (Image and Tag Watchers 2020). It is also possible configure Anchore Engine to emit webhooks when changes happen in images and tags. These are called subscriptions and can be used to trigger when new tag analyze, image update, vulnerability update, or change in policy status update happens. (Subscriptions 2020.)

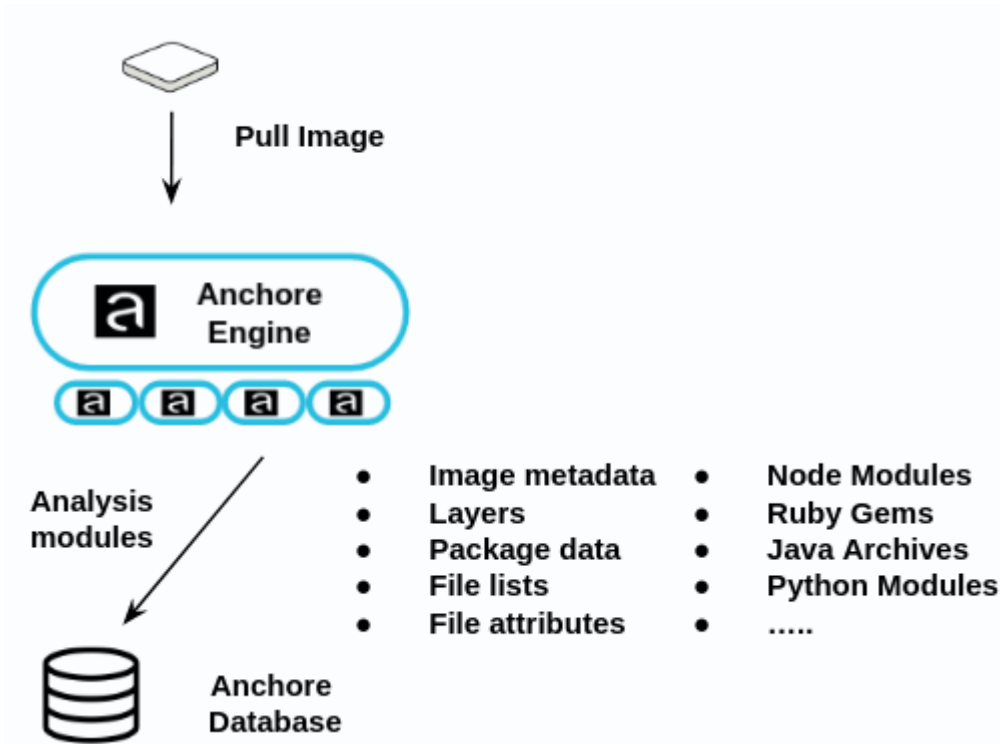


Figure 10. Anchore image analysis and analyzer module process (Analyzing Images 2020).

analysis_status Transitions

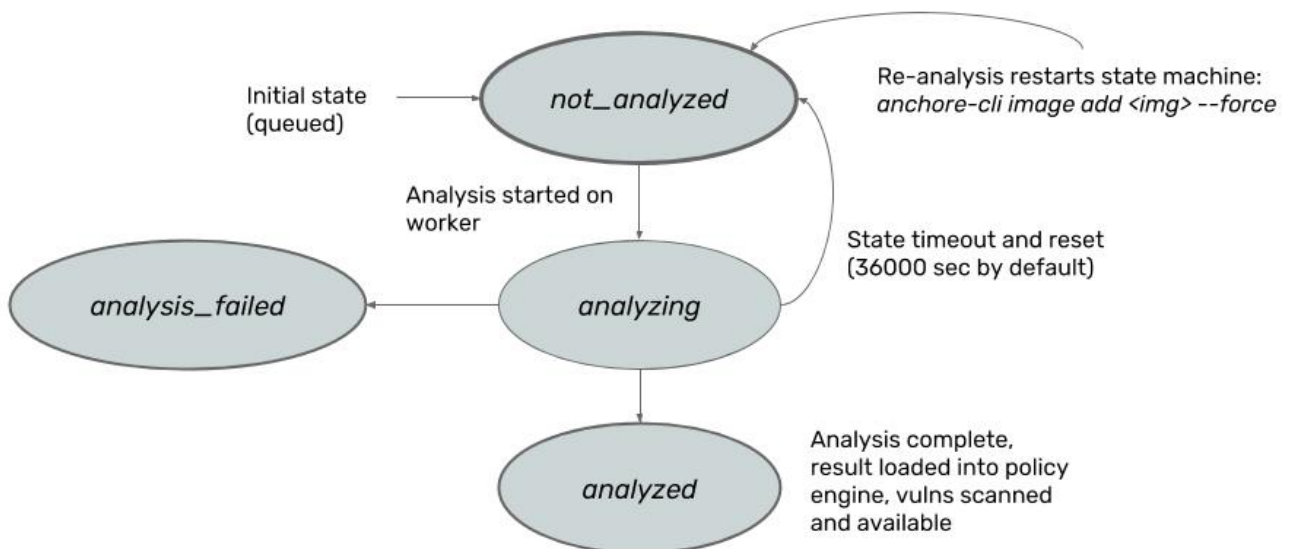


Figure 11. Anchore image analysis status workflow (Image Analysis Process 2020).

Anchore policies

After an image has been examined and its content has been inspected, categorized, and processed, the results can be compared to a user-defined set of criteria to determine a final pass/fail judgment. Users describe which evaluation to run on certain images and how the findings should be dealt with using Anchore Engine policies. A policy is represented as a policy bundle, which consists of a set of rules that are used to evaluate a container image. Checks against an image can be defined with rules like configuration file contents, security vulnerabilities, whitelists and blacklists, exposed ports. The evaluation process is portrayed in Figure 12. These checks are described as Gates that contain Triggers that run specific checks and emit matched outcomes, and they define what the engine may analyze and deliver a judgment regarding automatically. (Policy 2020.) For example, whitelisting can be used to ignore CVE matches that have been identified as false positives or ignore CVE matching on specified packages (Policy Bundles 2020). These rules can be enforced globally or tailored to specific images or application categories. Policy checks returns passed or failed indicating if the image complies with defined policies or not. (Policy 2020.)

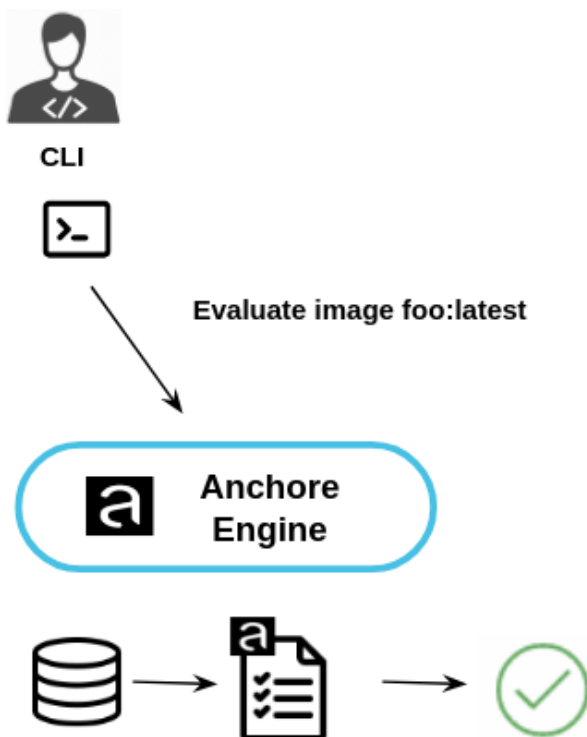


Figure 12. Anchore policy evaluation workflow (Policy 2020).

Anchore CI/CD integration

Anchore can be integrated into CI/CD pipelines like Jenkins to secure the pipeline by adding image scanning including CVE based security scanning and policy scanning that makes it more comprehensive. Images are given to the Anchore Engine for processing as part of the CI/CD pipeline. The build can be defined to fail if an image fails to pass the policy checks and if the image passes it can be transferred to a registry. It may also be defined that when a build fails it gives a warning and informs developers but is delivered to the registry. The process flow can be something like demonstrated in Figure 13. For example, Jenkins supports a plugin that allows creating Pipeline jobs and Freestyle jobs utilizing Anchore. Another way of using CI/CD systems is making API or CLI calls to the engine, first delivering an image for analysis and then retrieving policy status. (CI / CD Integration 2020.)

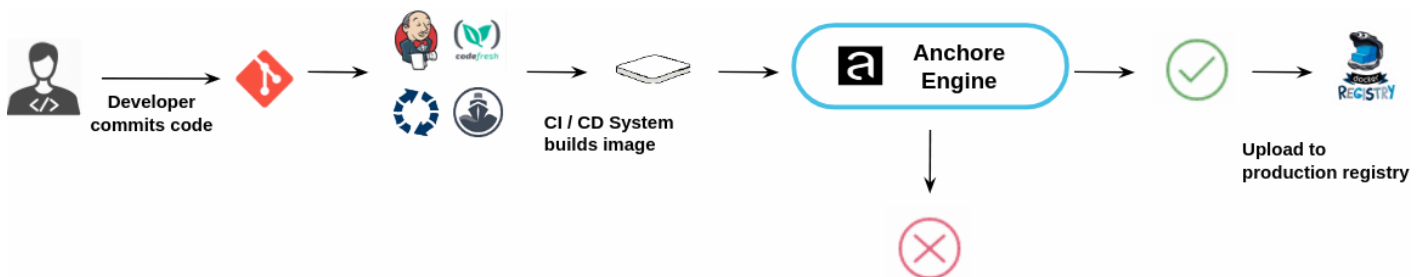


Figure 13. Anchore as part of a CI/CD pipeline (CI / CD Integration 2020).

4 Current process environment

In this project Qvantel Reference Product (QRP) environment is utilized and its software development lifecycle can be seen from Figure 14. QRP and Customer specific pipelines share same principles, but customer pipeline uses QRP release candidate as an input. The current situation and used technologies will be explained from QRP pipeline's perspective. Main goal of QRP and its processes is to provide verified system releases on a regular interval, currently every second week, to customer programs and support a maximum of two old releases with hot fixes. This environment is running in AWS cloud and is following a strategy called AWS Well-Architected Framework by using five pillars of operational excellence, security, reliability, performance efficiency, and cost optimization (AWS Well-Architected n.d). This assists in producing stable and efficient systems. Anchore will be deployed into QRP pipeline.

Current software development lifecycle

Used technologies, such as Jenkins, Artifactory, Docker, and Kubernetes are included in different phases of the pipeline. The first phase being QRP Component Development, the second QRP system-spec development, and the third QRP System verification. These phases are marked with numbers in Figure 14 making them easier to comprehend. In the first phase, before code and artifacts are pushed to Component Test Environment (CTE), the components are built, and unit tested using Jenkins worker. Resulting docker image and artifacts are uploaded to Artifactory and scanned for vulnerabilities. If the scan result is compliant with defined security policy, such as "no critical vulnerabilities allowed", the docker image and other artifacts are tagged accordingly. After that, the build process can continue to CTE phase. Components are then tested in CTE and after successful results components are tagged and moved forward to the second phase. Kubernetes manages the containers and clusters in all the different testing environments. Kubernetes fetches the docker images from Artifactory during the deployment phase.

In the second phase, the system-spec, which describes the whole contents of the deployment, is created. The contents of the new build include all new committed components that have passed the tests in CTE system-spec building phase. This phase is executed using Jenkins. Static verification is performed on the system-spec and the resulting docker image and artifacts are uploaded to Artifactory and scanned for vulnerabilities. After successful results, the system-spec is tagged, and the build process can move forward to phase three. System-spec is a versioned entity of the system, and it lists, for example, included versions of the component and more.

Phase three of the build process includes various testing stages, starting with the Integration Test Environment (ITE). The ITE includes integration and smoke tests on full system-spec. After successful checks of the criteria, the tag is added, and the system-spec is moved to the System Integration Test (SIT) environment. During SIT phase, the functional tests are run by using the use cases against API and GUI. Finally, the Upgrade Test Environment (UTE) and Performance Test Environment (PTE) UTE verify the upgrades from the last released version to current release. The PTE runs a set of dedicated performance tests to measure the system and component performance. Both

UTE and PTE give their own tags to the system-spec. The system-spec gets tagged from each environment starting from the second phase. After success, the released version of the system is allowed to be deployed to pre-production and production environments.

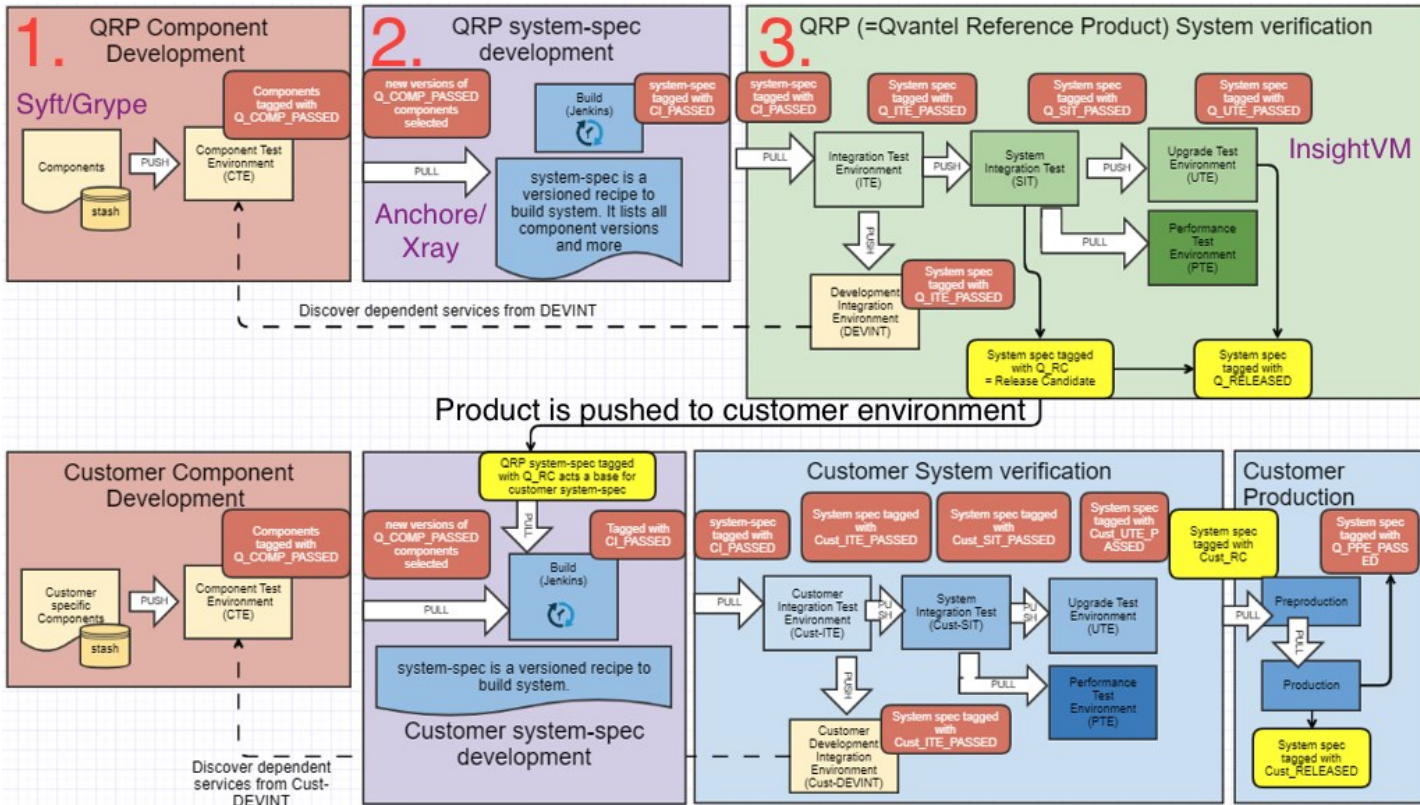


Figure 14. Current environment with numbered QRP pipeline phases.

Current security scanners in QRP pipeline

QRP pipeline includes multiple security scanners in different phases of the software development lifecycle, see Figure 14 for numbered phases. First scans are executed by the developers with Syft and Grype. This is done before they commit the new code. This gives them the ability to do preliminary security checks of their work locally. Syft is a tool that creates Bill of Materials of the contents of a Docker image and Grype is a vulnerability scanner. Both Syft and Grype are implemented and provided by Anchore, Inc.

In the second phase of development, the Xray tool by Jfrog scans the chosen repositories in Artifactory. Xray has watches that check the given policies. These watches are set to alert for violations that are triggered by the chosen policies. The results are available from Xray's GUI. Two separate policies are used. The first policy includes the Common Vulnerability Scoring System (CVSS) vulnerabilities from range 7 to 8.9, and the second, CVSS from range 9 to 10. The main purpose of Xray is to scan and identify the known vulnerabilities of all the artifacts in the selected repositories.

The third phase of vulnerability scans uses InsightVM by Rapid 7. The purpose of InsightVM is to scan the known vulnerabilities through the whole environment. All the selected environments contain a scan agent, which has the capability to check both the software and the operating system environment. Scan type is credentialed scan, and the scanner agent is run inside the operating system with elevated privileges. The agent can, for example, access and check the resources on the OS level, such as services, ports, running applications, and OS utility programs. Every environment includes assets that are machines from where the information is being collected. Every asset is scanned using the chosen scan profile. InsightVM creates reports from these scans that can be seen from the GUI.

4.1 Artifact Scanners as part of pipeline process

Subdivisions of this chapter go through the meaning of artifact scanning in pipeline process. This includes scanner types used in various phases of a development cycle. Purpose of artifact scanning overall and in current environment. Finishing with challenges found in artifact scanning tools.

4.1.1 Different scanner types in software development life cycle and current market situation

In the past, security testing was mostly done in a separate team and at the end of the development cycle, it was soon noticed that it was time consuming and costly. Today many organizations take advantage of DevSecOps model, which considers security straight from the beginning of the development cycle giving possibility to get feedback instantly enabling faster and cheaper fixes. (Chen et al. 2022, 313-314; Sengupta 2021.)

It is recommended for software teams to have comprehensive visibility over the vulnerabilities from the initial stages of pushing source code to repository given how complex modern applications are. Automated testing processes and tools are a huge help in this regard. Aside from assuring security, automated security testing improves agility by detecting and resolving security vulnerabilities early. (Chen et al. 2022, 313-314; Sengupta 2021.)

One of the most widely advised best practices is to make sure that vulnerabilities found in application are never overlooked during the development process. Organizations should consider vulnerabilities in all stages of development and deployment even if the program is not yet available for public use, to prevent sensitive data exposure and the risk of system breach. To achieve optimum accuracy in identifying threats and vulnerabilities, a thorough application security testing procedure can be performed through numerous mechanisms that target distinct phases of a software development life cycle. (Chen et al. 2022, 313-314; Sengupta 2021.)

Software Composition Analysis (SCA) is a software-only subset of component analysis that has a limited scope. These tools are used for inspecting open-source libraries and third-party components. SCA scan can be done in the very beginning when code has been committed to confirm that development has been done safely. Later in the testing phase SCA tool can keep developers updated if vulnerabilities in the used code and artifacts appear and maintain track of a program's dependencies and generate a warning if the application contains publicly reported vulnerabilities. These scanners compare open-source components for creating risk profiles and offering patches or other solutions to mitigate the risks. In this project Anchore will be used as a SCA scanner for scanning docker images and included artifacts. Another example of a SCA scanner is JFrogs Xray. (Chen et al. 2022, 314; Open Source Vulnerability Scanning: Methods and Top 5 Tools n.d; Springett n.d.)

Static Application Security Testing (SAST) is like SCA as it happens in the beginning and testing phase inspecting source code or compiled versions of the code. SAST entails examining source code and speculating on security issues, as well as recognizing design and construction flaws that could lead to a security risk. This type of scanner does not need the program to be run in production and its value lies in detecting flaws during software development, providing developers with

real-time feedback as they write code. (Dynamic Application Security Testing (DAST) 2014; Sengupta 2021; Wichers et al. n.d.)

Dynamic application security testing (DAST) is a method of evaluating an application while it is operating in an environment. This type of testing is beneficial for meeting industry standards and providing broad security measures for projects that are still in a development environment before going to production. DAST scanning is meant for operational testing rather than looking into source code or components that are included in the application. DAST is a black box testing method that assesses the application's security posture from the perspective of an attacker. DAST scanners look for security vulnerabilities like path traversal, insecure server configuration and command injections. InsightVM is a DAST scanning tool and used in this project environment. (Chen et al. 2022, 314; Dynamic Application Security Testing (DAST) 2014; Sengupta 2021; Vulnerability Scanning Tools n.d.)

Another commonly mentioned testing method is Interactive Application Security Testing (IAST) which is a combination of SAST and DAST. It allows security checks at different stages of development and deployment. IAST tool scans code for security flaws when the app is being tested by an automated test, manual tester, or any other activity that interacts with the application's functionality. (Chen et al. 2022, 314; Sengupta 2021.)

Current market situation

There are currently a lot of security and compliance tools on the market as can be seen from Figure 15. Companies and tools presented in Figure 15 do not include every tool on the market but gives a good overview of the number of options available. In this project SCA tool Anchore will be implemented and there is DAST tool InsightVM in the environment so these two will be used as examples. As an example, SCA tool providers are Snyk, WhiteSource, and Anchore. A few not represented in Figure 15 are JFrog Xray and Sonatype Nexus. For DAST tool providers there are StackHawk in Figure 15 and others for instance Veracode, Rapid7 InsightVM, Crashtest Security, and Invicti.

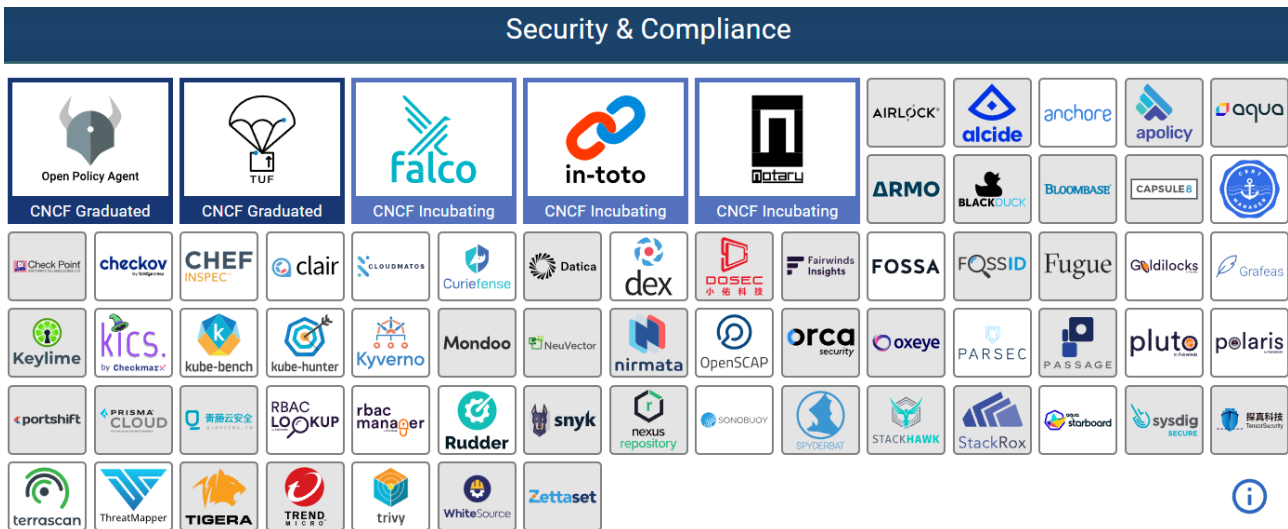


Figure 15. Overview of the current market situation for security and compliance tools (CNCF Cloud Native Interactive Landscape n.d).

As an example, for one type of SCA tool, Xray can find license violations and vulnerabilities in declaration stage and stop builds that include security issues. It can detect artifacts and their dependencies with recursive scan for more precise scan including smaller binary components. Xray analyzes how different components affect others providing continuous impact analysis. Being natively integrated with Frog Artifactory gives it direct access to metadata Artifactory stores. Xray has its own vulnerability database from where it gets information. Xray is controlled through REST API that allows customization of analysis for different components. Using Xray binary scan a specific binary can be pointed out and scanned from a file system. (Atzmony 2021b.)

As an example, for one type of DAST tool, InsightVM has a feature for assigning agents that automatically collect data from all available endpoints additionally checking remote workers and sensitive assets which cannot be actively scanned. InsightVM has an active dashboard that can be interacted with and can be customized. It also has a Real Risk Score view that collects information about likeliness of a specific attack depending on the vulnerability including threat feeds and business context, threat feeds also have their own view that shows the most relevant threats to the current environment. Feature called Remediation Projects allows security teams to designate and follow remediation duties in real time giving an opportunity to see how issues are fixed, this can be integrated with a ticketing system. InsightVM can periodically scan the public internet to access

information about global exposure to generic vulnerabilities using Project Sonar. It can check container images while they are in the build process. Security goals and common compliance requirements can be set with pre-built scan templates also Custom Policy Builder allows modifying currently active benchmarks or creating new ones. InsightVM gives the possibility to automate patching by gathering key information, collecting fixes for detected vulnerabilities, and finally implementing the patches and automating containment by applying provisional or permanent substitute controls. (InsightVM Features n.d.)

4.1.2 What is wanted to achieve with the artifact scanning?

Software is a product of third-party components and open-source code. To deliver the needed functionality, modern software is created utilizing open-source components in different ways integrating with the original code. Using open source has many benefits giving required building blocks for businesses to provide value, increase quality, minimize risk, and shorten time of delivering new features. Problems can arise when nobody knows the security of these components and organizations take the risk by using code they did not develop. (Paine 2022; Springett n.d.) All these components include pieces of artifacts that should be thoroughly scanned minimizing potential risks.

Artifact scanning should provide vulnerabilities found from components and evaluate risk by scoring or taking information from a vulnerability database. The scanning process should be fully automated or launched manually using automated processes improving continuity and frequency done against used artifact repository. Scanner tool can provide mitigation steps to fixing the found flaw.

Every environment is different, and it is important that the artifact scanner used can utilize policy checks etc. For finding false positives excluding irrelevant vulnerabilities and finding the ones that matter. Overall, the goal is to keep artifact repository safe from vulnerable components and catching possible flaws before they get to the production environment.

4.1.3 Challenges with artifact scanning tools

Understanding of actual risks can be a harder task than it seems. Artifact scanning tools can produce an extensive list of vulnerabilities including insignificant ones which add to system noise and postpone fixing. Manual results review is frequently required and consumes valuable time that

could be used to concentrate on the genuine issues. It is important to make practices that enhance the review of results and apply policies that alert about relevant issues making assessment of the findings simpler. (Berman 2021; Software Composition Analysis (SCA): What You Should Know n.d.)

Technical debt and hindering development

Initial scans may reveal a significant amount of technical debt if you have a vast codebase, and artifact scanning has not been done before. Some of the technical debt will be accumulated by the used open-source components and deprecated libraries. Development teams are usually responsible for addressing any defects, weaknesses, or vulnerabilities in the component. As a result, there might be need to spend more time and effort fixing open-source libraries that are vital to your applications, or you may have to modify your apps to work without the deprecated or at-risk library. In future this slowing process can be relieved by educating the development team on the significance of researching open-source components before use. (Berman 2021; Software Composition Analysis (SCA): What You Should Know n.d.)

Scanning should be performed in a way that it does not slow software development lifecycle. Instead of placing an unnecessary amount of security checks in different phases planning should be done carefully. Responsibility of security could be moved partly to the development teams to minimize interruption of workflows. (Berman 2021; Software Composition Analysis (SCA): What You Should Know n.d.)

Data accuracy and scanning coverage

It is possible that artifact scanners will not find all open-source components from chosen target or repository. The majority of artifact scanning tools keep databases of known open-source code and security vulnerabilities. Maintaining the accuracy of the data is a constant task that has a direct impact on the security of the deployed software. Scanner tool vulnerability databases may also be missing information on specific libraries purchased from smaller dealers or not well known open-source projects. Artifact scanning tool providers are under constant pressure to keep up with the increasing number of package managers, programming languages, and build systems. For most

cases, some amount of manual discovery or tracking is needed. Artifact scanning does not replace all forms of application security testing and is there only to complement the big picture. (Overview of SCA Tools: Core Features and Benefits of Deployment 2021; Software Composition Analysis (SCA): What You Should Know n.d.)

5 Implementation

In this project two implementations were created, one in a local environment and the second one in the production environment. The reason for this was to ensure that the components in use are compatible with each other as many of the examples found were two to three years old using older versions of Anchore Engine. Another reason was because I could not do all the setting up in the production environment because of the permissions and those changes were made through requests. On the local implementation it can be seen how all the components of the implementation can be set although taking into account that the local setup is not as complex as the production environment and is done on one virtual machine. Setting up that I could not do in the production environment was creating the AWS instance, Jenkins configurations except the Anchore job configurations, networking, and port openings.

The first step of the implementation was to research different types of implementation possibilities and choose the most suitable one. In the end the official Anchore Jenkins plugin was chosen as it could be easily set up as part of the existing QRP pipeline. The implementation works in the following way. Anchore Engine is running in a docker container waiting for requests to analyze an image. Anchore Jenkins plugin is set up in a Jenkins job in the Jenkins server and is sending requests to add and analyze image(s) to Anchore Engine via Anchore CLI commands. Jenkins build is started manually or automatically after a commit has been pushed to repository. Anchore Engine fetches, analyzes, saves results, evaluates image(s), and sends evaluation results to Jenkins and finally Jenkins job creates “Anchore Report” that contains all the results from the image evaluation. Jenkins job build is either “pass” or “fail” depending on the result of the Anchore policy evaluation and if wanted this can affect the whole pipeline or can be set as a separate check, see sequence diagrams (Figure 16) about the workflows.

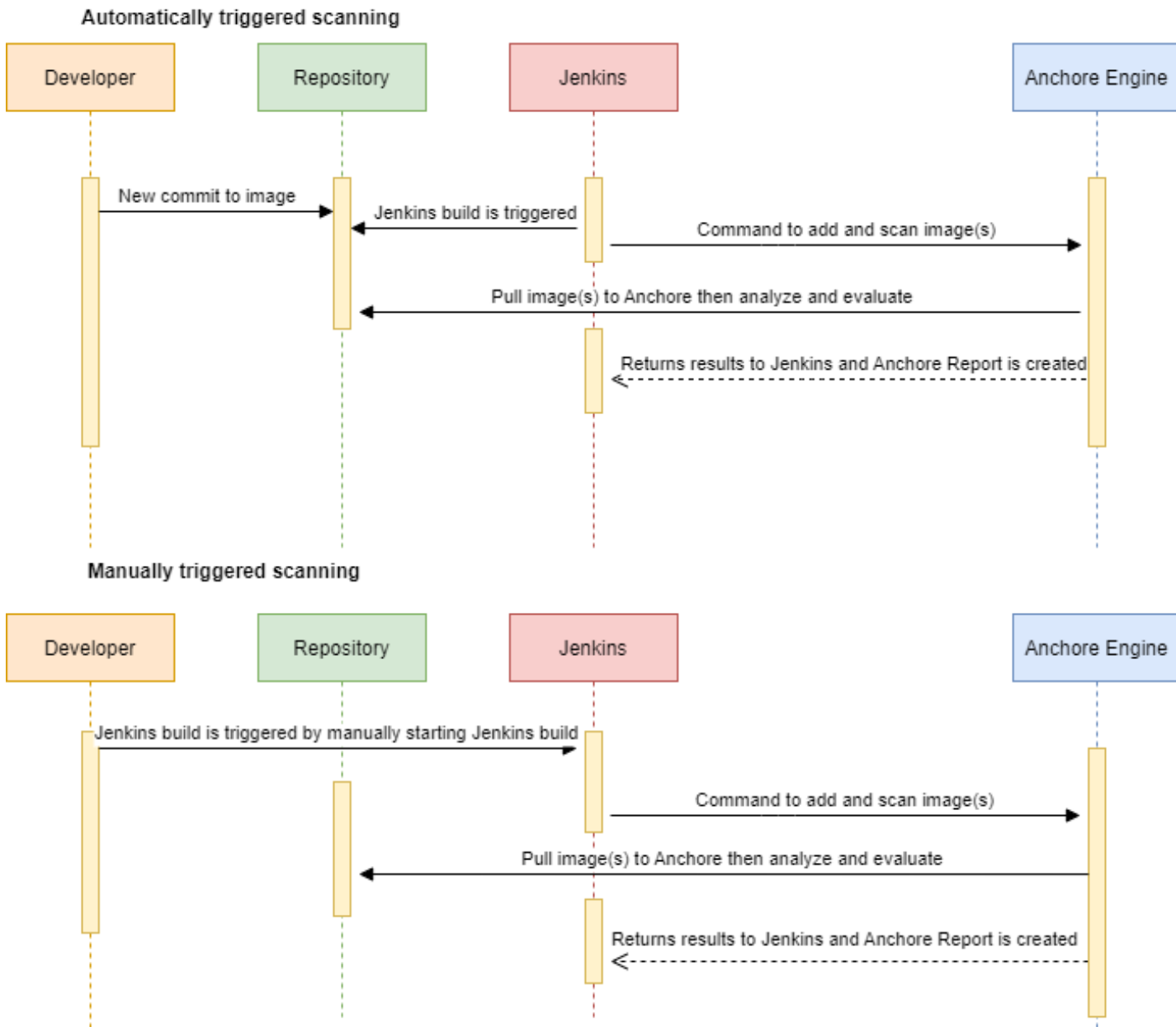


Figure 16. Anchore scanning automatic and manual workflows.

5.1 Local implementation

Local implementation was set up on a single virtual machine using Oracle VM VirtualBox see deployment diagram Figure 17. Anchore's documentation has instructions on what is required to set up a stand-alone installation and those were followed. This virtual machine had 4GB of RAM and enough disk space to handle larger container images, in this case 30GB was sufficient. To make the

machine as similar as the production instance it was given 2 processors and the OS used was CentOS 7. All the installing and configuring was done as root user. This implementation was set so that the Anchore scanning is manually triggered by starting the Jenkins job build (Figure 16).

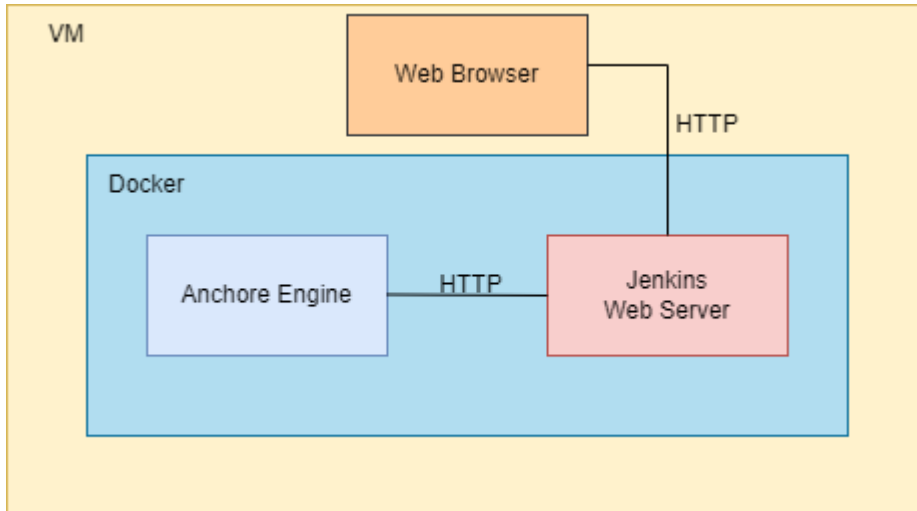


Figure 17. Local implementation deployment diagram.

5.1.1 Installation of software

In addition to system requirements Anchore Engine is working from a docker container and build with docker compose as it is a set of micro-services. Docker Engine is installed for this purpose and to make the networking simpler also Jenkins server was installed as a container. The whole setup was run on top of docker as containers.

Docker installation

First “yum-utils” package that includes “yum-config-manager” needs to be installed to be able to set up the docker repository as seen in Figure 18. Command “yum-config-manager” enables adding, enabling, and disabling repositories.

```
[root@localhost ~]# sudo yum install -y yum-utils
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: mirror.zetup.net
 * extras: mirror.zetup.net
 * updates: mirror.zetup.net
Package yum-utils-1.1.31-54.el7_8.noarch already installed and latest version
Nothing to do
[root@localhost ~]# sudo yum-config-manager \
> --add-repo \
> https://download.docker.com/linux/centos/docker-ce.repo
Loaded plugins: fastestmirror, langpacks
adding repo from: https://download.docker.com/linux/centos/docker-ce.repo
grabbing file https://download.docker.com/linux/centos/docker-ce.repo to /etc
/yum.repos.d/docker-ce.repo
```

Figure 18. Installing “yum-utils” and setting up docker repository.

The last step is to install Docker Engine, containerd, and Docker Compose (Figure 19). After installing docker needs to be started and optionally added to start on boot. Docker can be started and enabled on boot by using “systemctl” command that controls service manager (Figure 20). Now Docker is ready, and the actual software can be installed on top of it.

```
[root@localhost ~]# sudo yum install docker-ce docker-ce-cli containerd.io docker-compose-plugin
Loaded plugins: fastestmirror, langpacks
```

Figure 19. Installing Docker Engine, containerd, and Docker Compose.

```
[root@localhost anchore-user]# systemctl enable docker
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr
/lib/systemd/system/docker.service.
[root@localhost ~]# sudo systemctl start docker
```

Figure 20. Enabling and starting Docker service.

Jenkins server installation

Jenkins server will be run as a Docker container, and the image is taken from Docker Hub. Chosen image is from Docker repository “jenkins”, image name is “jenkins”, and tag “lts-jdk11” meaning it

is long-term supported and uses version 11 of Java SE Platform. This image will be updated with each LTS release and can change over time. Jenkins container will be started with various arguments as seen in Figure 21. Argument “-d” indicates that the container is started in detached mode and will not be attached to terminal but will be running in the background. Argument “-p” will be mapping local port to a port that Docker container exposes. The first port number is the local port and after colon the container port. Port 8080 is for the webserver and enables opening Jenkins on local machine Docker also directs traffic for the server that is in the container. Port 50000 is for the Jenkins agent. Argument “-v” is for creating volumes in this case volume called “jenkins_home” is created if it does not already exist and mounted under the path given after colon “/var/jenkins_home” in the container.

```
[root@localhost ~]# docker run -d -p 8080:8080 -p 50000:50000 -v jenkins_home:/var/jenkins_home jenkins/jenkins:lts-jdk11
Unable to find image 'jenkins/jenkins:lts-jdk11' locally
lts-jdk11: Pulling from jenkins/jenkins
```

Figure 21. Starting Jenkins server container.

To check that Jenkins server is up running docker containers can be listed with “docker ps”. Container logs should also contain line “Jenkins is fully up and running” indicating the server is ready. This text can be found with the “docker logs <container id>” command and the Administrator password can also be found from the logs that is needed to proceed with the installation as seen in Figure 22. The Administrator password can be optionally retrieved from location “/var/jenkins_home/secrets/initialAdminPassword”.

```
[root@localhost ~]# docker logs 1e35
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:
83ebbc63107a4f9496d4a782f48064c1
This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
```

Figure 22. Checking Jenkins server container logs for Administrator password.

Browsing to URL “http://localhost:8080” will open the Jenkins server GUI. The server needs to be configured when it is taken into use for the first time. Getting started screen will be opened to unlock Jenkins. Administrator password is given here to get started with the initial configuration as seen in Figure 23.

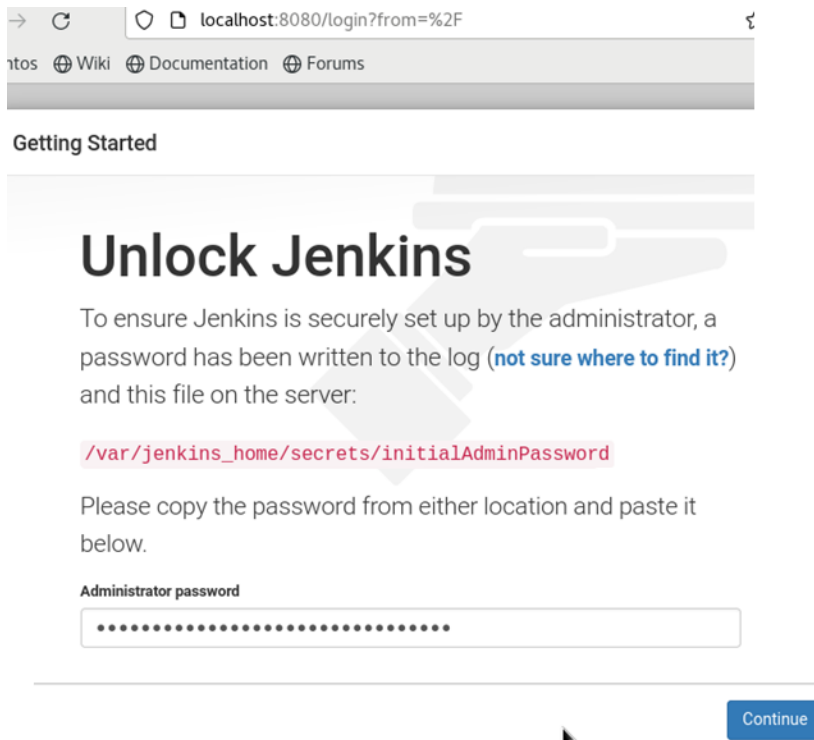


Figure 23. Starting initial configuration of the Jenkins server.

To ease the initial set up “Install suggested plugins” is selected as it includes a list of common plugins that are generally useful. Installing the plugins will take a few minutes. Plugin options and installation can be seen from Figure 24.

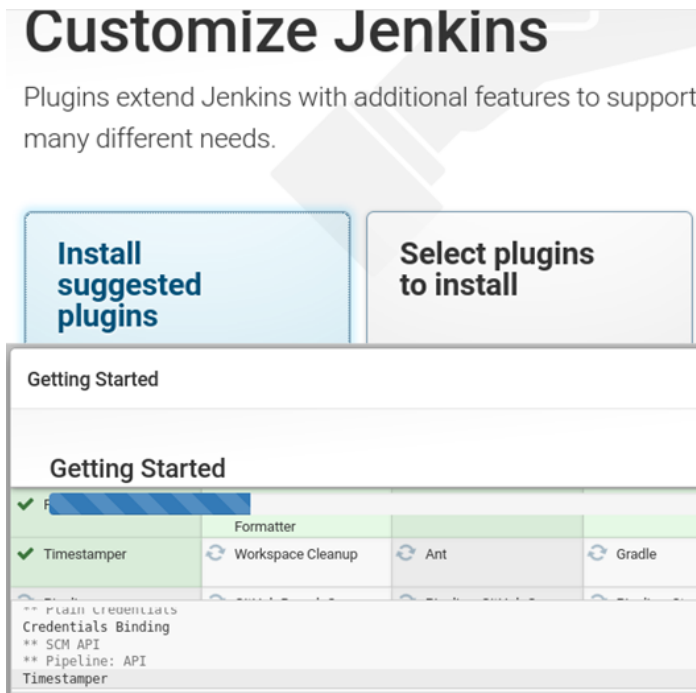


Figure 24. Jenkins initial plugin options and installation of plugins.

Next step is to create the first admin user, Jenkins administrator account. Give credentials and click “Save and Continue” (Figure 25). The final step in the initial setup is to provide the “Jenkins URL” and this can be the IP of the Jenkins server. Docker container IP address can be checked with “docker inspect <container id>” command and found from the network section. Checking IP address and giving “Jenkins URL” can be seen from Figure 26. The “Jenkins URL” can be later changed from the Jenkins global settings. Jenkins is now ready, and everything has been configured. Click “Start using Jenkins”. Jenkins server can be found with URL “<http://localhost:8080>” or “<container-ip-address>:port” on the host machine.

Create First Admin User

Username:

Password:

Confirm password:

Full name:

E-mail address:

3 [Skip and continue as admin](#) [Save and Continue](#)

Figure 25. Creating Jenkins administrator account.

```
[root@localhost ~]# docker inspect 1e35
```

```

  "Gateway": "172.17.0.1",
  "IPAddress": "172.17.0.2",
  "IPPrefixLen": 16

```

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build

[Not now](#) [Save and Finish](#)

Figure 26. Checking Jenkins server container IP address and giving “Jenkins URL”.

Anchore Engine installation

Anchore Engine Docker Compose YAML file is fetched that includes all micro-services needed to run it. When fetch is done services are started with “docker compose” command in detached mode see Figure 27. After a few seconds Anchore Engine should be up and running, this can be

checked with “docker compose ps” command that lists running containers. Included services are engine API, engine catalog, engine simpleq, policy engine, engine analyzer, and database. System status of Anchore Engine services can be verified via “docker compose exec” and Anchore CLI command (Figure 28). Anchore Engine is now ready for use.

```
[root@localhost ~]# curl -O https://engine.anchore.io/docs/quickstart/docker-compose.yaml
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 4331    0 4331    0     0  11266      0  --:--:--  --:--:--  --:--:-- 11249
[root@localhost ~]# ls
anaconda-ks.cfg  docker-compose.yaml
[root@localhost ~]# docker compose up -d
[+] Running 0/14
```

Figure 27. Fetching Docker Compose YAML and starting services.

```
[root@localhost ~]# docker compose exec api anchore-cli system status
Service simplequeue (anchore-quickstart, http://queue:8228): up
Service analyzer (anchore-quickstart, http://analyzer:8228): up
Service catalog (anchore-quickstart, http://catalog:8228): up
Service policy_engine (anchore-quickstart, http://policy-engine:8228): up
Service apiext (anchore-quickstart, http://api:8228): up

Engine DB Version: 0.0.15
Engine Code Version: 1.0.0
```

Figure 28. Verifying system status of Anchore Engine services.

When Anchore Engine is first started it will take some time for it to synchronize all vulnerability data to the engine. Status of the feed sync can be checked by using the Anchore CLI “docker compose exec api anchore-cli system feeds list”. Listing shows “RecordCount” for different vulnerability groups as soon as none is 0 the sync should be ready.

Docker images can be scanned via the Anchore Engine by adding an image (Figure 29). In this example Debian 7 image from Docker Hub is used. Status of the analysis and waiting for the analysis to be completed can be done with command “docker compose exec api anchore-cli image wait docker.io/library/debian:7”. When the analysis is done vulnerability scan results for the image can be checked. Anchore-engine will list vulnerability ID, package name, severity, fix if there is one,

CVE reference, URL with more details about the vulnerability, type of the artifact, and feed group as seen in Figure 30.

```
[root@localhost ~]# docker compose exec api anchore-cli image add docker.io/library/debian:7
Image Digest: sha256:81e88820a7759038ffa61cff59dfcc12d3772c3a2e75b7cfe963c952da2ad264
Parent Digest: sha256:2259b099d947443e44bbd1c94967c785361af8fd22df48a08a3942e2d5630849
Analysis Status: not_analyzed
Image Type: docker
Analyzed At: None
Image ID: 10fcec6d95c4a29f49fa388ed39cded37e63a1532a081ae2386193942fc12e21
Dockerfile Mode: None
Distro: None
Distro Version: None
Size: None
Architecture: None
Layer Count: None

Full Tag: docker.io/library/debian:7
Tag Detected At: 2022-08-18T05:01:55Z
```

Figure 29. Adding image to Anchore Engine.

```
[root@localhost ~]# docker compose exec api anchore-cli image vuln docker.io/library/debian:7 all
Vulnerability ID      Package              Severity            Fix
  CVE Refs              Type                Feed Group          Package Path
CVE-2005-2541         tar-1.26+dfsg-0.1+deb7u1      Negligible          None
  CVE-2005-2541         dpkg                 debian:7             pkgdb
  https://security-tracker.debian.org/tracker/CVE-2005-2541
CVE-2007-5686         login-1:4.1.5.1-1+deb7u1      Negligible          None
  CVE-2007-5686         dpkg                 debian:7             pkgdb
  https://security-tracker.debian.org/tracker/CVE-2007-5686
CVE-2007-5686         passwd-1:4.1.5.1-1+deb7u1     Negligible          None
  CVE-2007-5686         dpkg                 debian:7             pkgdb
  https://security-tracker.debian.org/tracker/CVE-2007-5686
```

Figure 30. Anchore Engine image vulnerability check.

The last step is checking policy evaluation against the analyzed image (Figure 31). With these default policies the result was “pass” and in a real-life case the image could have been passed forward in the development lifecycle. If the evaluation result had been “fail” the image would have been sent back for fixing.

```
[root@localhost ~]# docker compose exec api anchore-cli evaluate check docker.io/library/debian:7
Image Digest: sha256:81e88820a7759038ffa61cff59dfcc12d3772c3a2e75b7cfe963c952da2ad264
Full Tag: docker.io/library/debian:7
Status: pass
Last Eval: 2022-08-18T05:35:08Z
Policy ID: 2c53a13c-1765-11e8-82ef-23527761d060
```

Figure 31. Image policy evaluation check.

5.1.2 Configuration of environment and software

In this chapter networks and Jenkins are configured so images can be scanned. Installation process and configuration of Jenkins plugin is shown. Jenkins job is created and configured for triggering the Anchore scanning on specific image(s).

Starting Jenkins in same network as Anchore Engine

Jenkins server container will be connected to the same network as Anchore Engine for Jenkins to be able to access the engine. Network of Anchore containers can be checked with “docker inspect <container id>” command from the “Networks” section. All the Anchore Engine services are in the same network by default, and the network name is checked from the root-api-1 container (Figure 32). When using Docker Compose, network name will be created based on the name of the folder it is started from, and in this case compose file in root home folder will give the network name “root_default” (Networking in Compose n.d).

```

"Networks": {
  → "root_default": {
    "IPAMConfig": null,
    "Links": null,
    "Aliases": [
      "root-api-1",
      "api",
      "a657b2b8b900"
    ],
    "NetworkID": "3d85a8a8faaa",
    "EndpointID": "e236d45f21d",
    "Gateway": "172.18.0.1",
    "IPAddress": "172.18.0.6",
    "IPPrefixLen": 16,
  },
}

```

Figure 32. Checking Anchore Engine network name.

As for Jenkins container it will be connected to the “bridge” network by default. It needs to be connected to “root_default” and it is good to disconnect the container from the “bridge” network, so the Jenkins container does not have multiple networks see Figure 33. Container id for Jenkins begins with “1e35” and that is used in the commands. Now both Anchore Engine and Jenkins server are in network “172.18.0.X”.

```

[root@localhost ~]# docker network connect root_default 1e35
[root@localhost ~]# docker network disconnect bridge 1e35

```

Figure 33. Connecting Jenkins server to “root_default” and disconnecting from “bridge” network.

Jenkins server in “root_default” network has a new IP address. Jenkins webserver will now be found behind the new IP address URL in port 8080. “Jenkins URL” will be updated with the new one, as it was first set in the initial configuration. Changes can be done in the global settings of Jenkins. The path to changing “Jenkins URL” is the following: Open “Manage Jenkins” there is “Configure System” and find “Jenkins Location” section header. URL is changed to contain current IP address as seen in Figure 34.

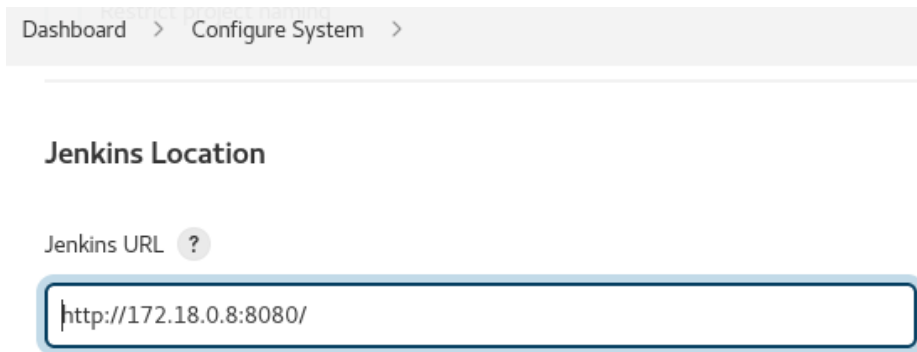


Figure 34. Changing "Jenkins URL" to new one.

Installation of Jenkins Anchore plugin

To scan images with Anchore Engine via Jenkins, plugin called "Anchore Container Image Scanner" needs to be installed. In "Plugin Manager" searching for "anchore" in available plugins will give the latest version as seen in Figure 35. "Install without restart" option is chosen. "Plugin Manager" can be found from under "Manage Jenkins" that includes various configurations. The version of the plugin in this local environment is 1.0.24.

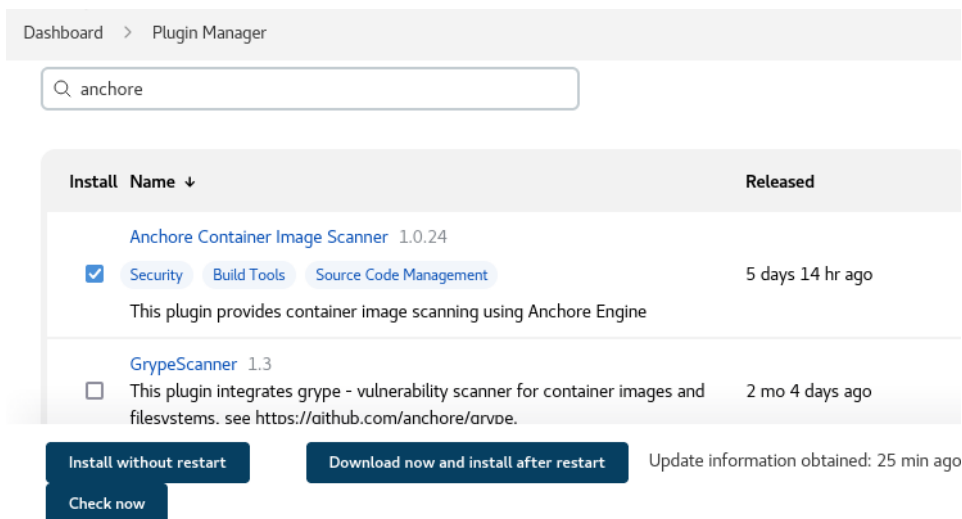
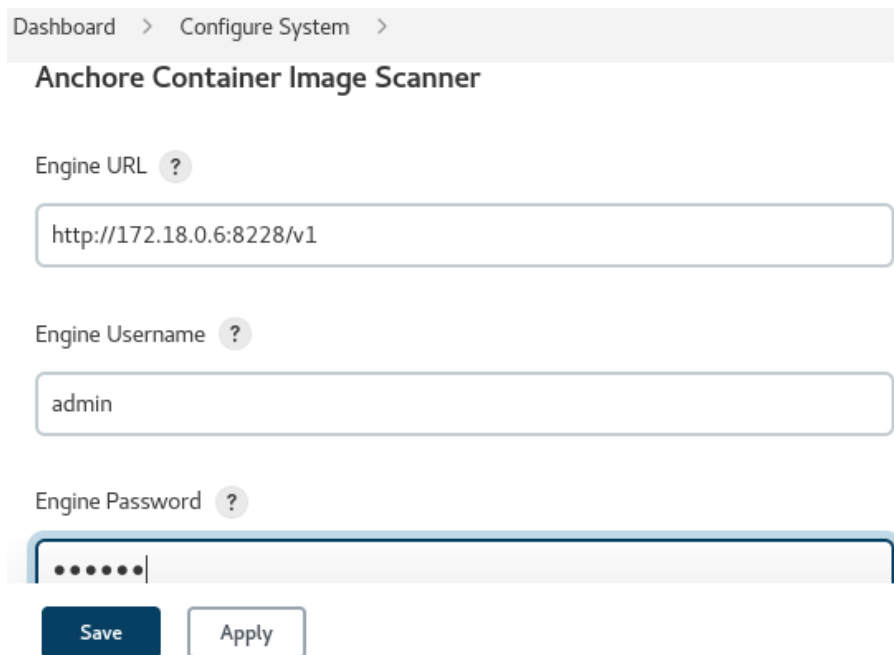


Figure 35. Installing Anchore scanner plugin.

After installing plugin, Jenkins global settings need to be configured so that our Jenkins server can connect and send requests to the Anchore Engine. Global settings can be found from under “Manage Jenkins” then “Configure System” and finding header of section “Anchore Container Image Scanner”. “Engine URL” is the Anchore Engine API URL <http://<root-api-1-container-ip>:8228/v1>. Anchore Engine by default is behind port 8228 if not changed in the compose file. Username and password as default are “admin” and “foobar” it is highly recommended to change at least the password but in this local test they are kept as default, see Figure 36.



The screenshot shows the Jenkins configuration page for the 'Anchore Container Image Scanner' plugin. The breadcrumb navigation at the top reads 'Dashboard > Configure System > Anchore Container Image Scanner'. There are three input fields, each with a help icon (question mark):

- Engine URL**: The input field contains the text 'http://172.18.0.6:8228/v1'.
- Engine Username**: The input field contains the text 'admin'.
- Engine Password**: The input field is masked with seven dots.

At the bottom of the form, there are two buttons: a dark blue 'Save' button and a white 'Apply' button with a grey border.

Figure 36. Configuring Anchore scanner plugin global settings.

Creating Jenkins job for Anchore

Jenkins “Freestyle project” is selected and configured to create the Anchore Jenkins job. Two build steps are needed for the Anchore scanning. First “Execute shell” step is for giving path to the wanted images for Anchore Engine and second “Anchore Container Image Scanner” step is the actual command for Anchore Engine to scan the given images. In this example image(s) will be taken from Docker Hub and path is echoed to Jenkins job workspace file called “anchore_images”. From this file Anchore Engine will retrieve the information of images to be scanned. “Execute shell” step can be seen from Figure 37 and full command used is ‘echo “docker.io/library/wordpress:6.0” > anchore_images’ with this “wordpress” image with tag “6.0” will be scanned.

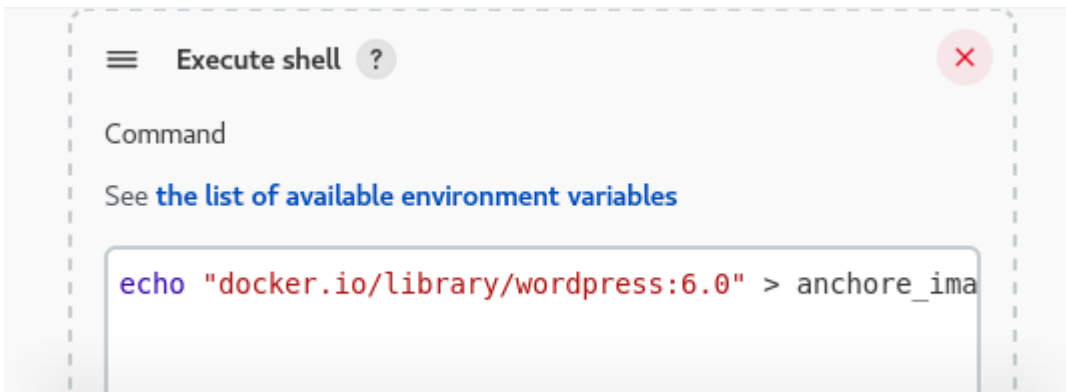


Figure 37. Jenkins job “Execute shell” build step.

Continuing to the scanning step where Anchore plugin will direct Anchore Engine on what to analyze. “Image list file” tells what file includes list of images to be analyzed, in this case ‘anchore_images’ file. As default it is chosen to fail the Jenkins build if policy evaluation results in “FAIL”. All the settings can be left as default, and nothing was changed as seen in Figure 38. During build Anchore will constantly poll status of the analysis until it is finished or if the “Anchore Engine operation retries” value is hit. If this happens the build results in fail. In the “Anchore Container Image Scanner” build step it is also possible to override global configurations, but this is not necessary as the details were already set in the global settings.

The screenshot shows the configuration for the 'Anchore Container Image Scanner' build step. It includes the following fields and options:

- Image list file:** A text input field containing 'anchore_images'.
- Fail build on policy evaluation FAIL result:** A checked checkbox.
- Fail build on critical plugin error:** A checked checkbox.
- Anchore Engine operation retries:** A text input field containing '300'.
- Anchore Engine policy bundle ID:** An empty text input field.
- Anchore Engine image annotations:** A section with an 'Add annotation' button.
- Anchore Engine auto-subscribe tag updates:** A checked checkbox.
- Anchore Engine force image analysis:** An unchecked checkbox.

Figure 38. Jenkins job “Anchore Container Image Scanner” build step.

When running build and scanning WordPress image, as result four JSON files are created including used Anchore gates, security, evaluation, and found vulnerabilities. To see things in a clearer format “Anchore Report” can be opened which the Anchore Jenkins build creates as seen in Figure 39.

The screenshot shows the Jenkins build results for 'Build #2 (Aug 18, 2022, 10:40:49 AM)'. The build status is 'Failed' (indicated by a red 'X' icon). The build was started 1 hr 38 min ago and took 2 min 16 sec. The build artifacts section lists four JSON files:

File Name	Size	Action
anchore_gates.json	12.38 KB	view
anchore_security.json	84.18 KB	view
anchoreengine-api-response-evaluation-1.json	33.34 KB	view
anchoreengine-api-response-vulnerabilities-1.json	316.77 KB	view

Below the artifacts, there is a 'No changes' message, a 'Started by user Admin' message, and an 'Anchore Report (FAIL)' link.

Figure 39. Jenkins job build results.

When opening the Anchore Report, the first header is a summary of found vulnerabilities and severities. Under summary, policy evaluation report lists found vulnerabilities with more details see Figure 40. From the report can be seen the image id, repository tag, what vulnerability triggered this action, used gate, trigger type that is package, vulnerability severity and link to see more details of the vulnerability, gate action, is this vulnerability whitelisted or not, and used policy id. Our Jenkins build resulted as fail as it was configured in the “Anchore Container Image Scanner” build step that the build will fail if the policy evaluation results as fail. This is because the WordPress image contained critical or high vulnerabilities. The result of the evaluation can be changed by configuring Anchore Engine policy bundle or by whitelisting vulnerabilities in the policy bundle.

Dashboard > anchore_scan > #2 > Anchore Report (FAIL)

Policy Security

Anchore Policy Evaluation Summary

Show 10 entries Search:

Repo Tag	Stop Actions	Warn Actions	Go Actions	Final Action
docker.io/library/wordpress:6.0	15	19	0	STOP

Showing 1 to 1 of 1 entries Previous 1 Next

Anchore Policy Evaluation Report

Show 10 entries Search:

Image Id	Repo Tag	Trigger Id	Gate	Trigger	Check Output	Gate Action	Whitelisted	Policy Id
66b89e8b083b68f0	docker.io/library/w	CVE-2022-1882+linux	vulnerabilities	package	HIGH Vulnerability found in os	STOP	false	48e6f7d6-1765-11e8-b5f9-8b6f228548b6

Figure 40. Jenkins build “Anchore Report” results.

5.1.3 Use of implementation and configuring Anchore policy bundle

In this example Anchore Engine policy bundle will be configured to alter Jenkins job “Anchore Report” results. “Execute Shell” build step is modified to take two images. More than one image will be used so policy mapping can be utilized. Mapping will force policy evaluation on any image that matches the given criterion. Images added to Anchore scanning job are “wordpress:6.0” and “wordpress:php8.1”. Image paths must be echoed as multiple lines into “anchore_images” file. “Execute Shell” step is changed to be as follows: ‘echo “docker.io/library/wordpress:6.0\ndocker.io/library/wordpress:php8.1” > anchore_images’. Multiple images can be

scanned with this method. After running the configured job, the following results can be seen (Figure 41). Both images have the same amount of stop and warn actions.

Dashboard > anchore_scan > #9 > Anchore Report (FAIL)

Show 10 entries Search:

Repo Tag	Stop Actions	Warn Actions	Go Actions	Final Action
docker.io/library/wordpress:6.0	18	18	0	STOP
docker.io/library/wordpress:php8.1	18	18	0	STOP

Showing 1 to 2 of 2 entries

Previous 1 Next

Figure 41. “Anchore Report” summary for two wordpress images.

Results can be filtered with “Trigger Id”, this will be used to find same CVE from both images so it can be whitelisted from the other image. As seen in Figure 42, both images have same triggering factor “CVE-2021-30474+libaom0” this will be taken as an example and whitelisted from the other image removing gate action “STOP”.

Image Id	Repo Tag	Trigger Id	Gate	Trigger	Check Output	Gate Action	Whitelisted	Policy Id
66b89e8b083b68f0bc8c80a4190bc16a72368a1b44e04b1ed625d954a854c9ea	docker.io/library/wordpress:6.0	CVE-2021-30474+libaom0	vulnerabilities	package	CRITICAL Vulnerability found in os package type (dpkg) - libaom0 (CVE-2021-30474 - https://security-tracker.debian.org/tracker/CVE-2021-30474)	STOP	false	48e6f7d6-1765-11e8-b5f9-8b6f228548b6
6bf5dbd42db9c69bdd042d541ef376a6ddc748f4d6425c9440dee9f6c23a8ebb	docker.io/library/wordpress:php8.1	CVE-2021-30474+libaom0	vulnerabilities	package	CRITICAL Vulnerability found in os package type (dpkg) - libaom0 (CVE-2021-30474 - https://security-tracker.debian.org/tracker/CVE-2021-30474)	STOP	false	48e6f7d6-1765-11e8-b5f9-8b6f228548b6

Figure 42. Filtering “Anchore Report” results with “Trigger Id”.

All actions from here on are happening inside the “root-api-1” container. To fetch Anchore Engine policy bundle “root-api-1” container is accessed with “docker exec” command and policies are listed with Anchore CLI command as seen in Figure 43. Container will be accessed as “root” user because the default Anchore user will not have permissions to download the policy bundle. The policies can be created from scratch, but in this case the default policy will be modified.

```
[root@localhost ~]# docker exec -u root -it root-api-1 bash
[root@e9b7a406b880 anchore-engine]# anchore-cli policy list
Policy ID                Active    Created                U
pdated
2c53a13c-1765-11e8-82ef-23527761d060    True     2022-08-18T04:53:06Z    2
022-08-18T04:53:06Z
```

Figure 43. Accessing Anchore Engine container with “root” user and listing policies.

The default policy bundle is downloaded as JSON with Anchore CLI command so that it can be modified (Figure 44). Default policy’s “mappings”, “policies” and “whitelists” sections are modified, and new ones created to set the vulnerability “CVE-2021-30474+libaom0” whitelisted only for the “wordpress:php8.1” image. Modified policy bundle JSON will be uploaded back to the Anchore Engine, when listing policies, the policy is not yet active. To activate the policy Anchore CLI policy activation command is used, see Figure 45.

```
[root@e9b7a406b880 anchore-engine]# anchore-cli policy get 2c53a13c-1765-11e8-82ef-2352
7761d060 --detail > policybundle.json
[root@e9b7a406b880 anchore-engine]# ls
policybundle.json
```

Figure 44. Fetching Anchore Engine default policy bundle to modify it.

```

[root@e9b7a406b880 anchore-engine]# anchore-cli policy add policybundle.json
Policy ID: 2c53a13c-1765-11e8-82ef-23527761d060
Active: False
Source: true
Created: 2022-08-18T04:53:06Z
Updated: 2022-08-21T19:21:28Z

[root@e9b7a406b880 anchore-engine]# anchore-cli policy list
Policy ID          Active      Created      U
pdated
2c53a13c-1765-11e8-82ef-23527761d060  False      2022-08-18T04:53:06Z  2
022-08-21T19:21:28Z
[root@e9b7a406b880 anchore-engine]# anchore-cli policy activate 2c53a13c-1765-11e8-82ef-23527761d060
Success: 2c53a13c-1765-11e8-82ef-23527761d060 activated
[root@e9b7a406b880 anchore-engine]# anchore-cli policy list
Policy ID          Active      Created      U
pdated
2c53a13c-1765-11e8-82ef-23527761d060  True       2022-08-18T04:53:06Z  2
022-08-21T19:24:05Z

```

Figure 45. Adding modified default policy bundle to Anchore Engine and activating it.

Anchore Engine default policy bundle (Appendix 1) was modified in the following way: Two different “mappings” for both images, two different “whitelists” for both images, and new policy for “wordpress6.0” and other images as default policy was modified for “wordpress:php8.1” image. All configurations in the policy bundle JSON are expressed as “key: value” pairs. Each policy bundle must contain the following sections see Figure 46 at least “comment”, “whitelisted_images”, and “blacklisted_images” can be left as empty.

```

{
  "id": "policy_id",
  "version": "1_0",
  "name": "Policy name",
  "comment": "Some comment",
  "whitelisted_images": [],
  "blacklisted_images": [],
  "mappings": [],
  "whitelists": [],
  "policies": []
}

```

Figure 46. Empty policy bundle JSON.

“Mappings” array needs to have at least the following values: Registry, Repository, policy id, whitelist id and image detail that is matched with “type” and “value”. With this information the mapping rule will know which registries, what repository names, and what images it applies to. In

addition, mappings have defined policies and whitelists that apply to matched images. As “mappings” are evaluated in order causes it to halt on the first matching rule, this means the order of the “mappings” is important and will affect the report results. As “mappings” were created for “wordpress:php8.1” and other images, type “tag” was selected and value for it “php8.1”. Other images mapping was left to include all other images with type “tag” and value “*” asterisk representing wildcard character. The order here is important as the first mapping will catch “wordpress:php8.1” image and other images will be checked with the latter mapping rule. Both “mappings” have an asterisk on the registry URL and repository name as only image tag is important in this case and Docker Hub registry was used in both cases. For “php8.1” image “policy_id” and “whitelist_ids” are kept as default, but the content is modified. For other images a new policy and whitelist was created. To distinguish which mapping is for which image, “php.8.1” image mapping name was left as “default” and other images mapping was named “testmapping” see Appendix 2.

Whitelists are created to offer a way to overrule policy-rule matches. It is a named group of exclusion criteria that match trigger outputs. Whitelist has the following values defined: “id”, “name”, “comment”, “version”, “items”. Items also include the following values: “gate”, “trigger_id”, “id”. “Gate” is the triggering factor vulnerabilities, packages etc. “Trigger_id” is specified trigger result match, triggers may have different “trigger_id” format. “id” is a unique identifier for the whitelist object rule. “Items” array can also be left as empty, this means there are no whitelist rules. In this case for “php8.1” image whitelisting rule for “CVE-2021-30474+*” was created as was decided in the beginning and whitelists id was left as default. For other images a new whitelist was created, and the “items” list is left as empty, thus nothing is whitelisted. “Php8.1” image whitelist id is “37fd763e-1765-11e8-add4-3b16c029ac5c” and other images “whitelist6.0”. These same id’s can be seen applied in the “mappings” arrays (Appendix 2).

Policies are JSON objects within the policy bundle. Policy consists of rules and each rule determines a specific check against the image resulting in an action if matched. Policy must have “id”, “name”, and “rules” defined. As “rules” holds one or more rule each one defines “action”, “gate”, “id”, “parameters”, and “trigger”. Each rule can have multiple parameters and they are passed as name, value pairs. Rule determines if matched result causes image evaluation stop, go, or warn action. Parameters are specific to selected gate and trigger, for example from gate “vulnerabilities” trigger “package” can be selected and then parameter “severity” can be selected and given

value “high”. This rule would mean only package vulnerabilities with severity high result in defined action. The default policy was untouched and used for “php8.1” image. New policy was created for other images and there was only one rule defined to cause “STOP” action if there are any packages with vulnerabilities higher than severity “medium”. Both policy id’s can be seen defined in the “mappings” arrays this way they are applied on the chosen images. For “php8.1” image, used policy was with id “48e6f7d6-1765-11e8-b5f9-8b6f228548b6” and for other images “policytest”. All modifications can be seen from Appendix 2.

If comparing old summary (Figure 41) and new resulting summary (Figure 47), instead of 18 stop actions, there are only 17 stop actions for “wordpress:php8.1” image resulting from the whitelisting. It can also be seen that because none “WARN” action rules were set in the “policytest” policy which was for all other images than “php8.1” the result for those is “0” for “wordpress:6.0” image.

Anchore Policy Evaluation Summary

Show entries Search:

Repo Tag	Stop Actions	Warn Actions	Go Actions	Final Action
docker.io/library/wordpress:6.0	18	0	0	STOP
docker.io/library/wordpress:php8.1	17	18	0	STOP

Showing 1 to 2 of 2 entries Previous **1** Next

Figure 47. New “Anchore Report” summary after modifications.

Checking the new report results (Figure 48), for “wordpress:php8.1” the gate action for vulnerability “CVE-2021-30474+libaom0” is “go” and for “wordpress:6.0” image “stop”, this is caused by the whitelisting. From the last column can be seen that the images use different policies that were applied before in the policy bundle “mappings”. “wordpress:6.0” is using the created “policytest” policy and “wordpress:php8.1” is using the default policy “48e6f7d6-1765-11e8-b5f9-8b6f228548b6”.

Image Id	Repo Tag	Trigger Id	Gate	Trigger	Check Output	Gate Action	Whitelisted	Policy Id
6bf5dbd4 2db9c69b dd042d54 1ef376a6 ddc748f4 d6425c94 40dee9f6 c23a8ebb	docker.io /library/w ordpres s:php8.1	CVE-2021- 30474+liba om0	vulnerabilities	package	CRITICAL Vulnerability found in os package type (dpkg) - libaom0 (CVE-2021-30474 - https://security- tracker.debian.org /tracker /CVE-2021-30474)	go	[object Object]	48e6f7d6-1765-11e8- b5f9-8b6f228548b6
66b89e8b 083b68f0 bc8c80a4 190bc16a 72368a1b 44e04b1e d625d954 a854c9ea	docker.io /library/w ordpres s:6.0	CVE-2021- 30474+liba om0	vulnerabilities	package	CRITICAL Vulnerability found in os package type (dpkg) - libaom0 (CVE-2021-30474 - https://security- tracker.debian.org /tracker /CVE-2021-30474)	STOP	false	policytest

Figure 48. New “Anchore Report” results after modifications.

5.2 Production implementation

Instance for Anchore was created into AWS cloud and instance type is Amazon EC2 T2 instance “t2.medium”. It has 2 vCPU's and 4GB RAM, which is sufficient for Anchore. (Amazon EC2 T2 Instances n.d.) System OS used in this project is “Rocky Linux 8.6.” Anchore requires PostgreSQL version “9.6.” or newer to provide storage for images, policies and data created during analysis. Around 2GB RAM should be enough for operating in a steady state. Access to registry is needed in this case Artifactory is used. Anchore needs to be able to synchronize feed data from Anchore Cloud Service that is collecting latest data on vulnerabilities, one end point is needed for this, “host: ancho.re TCP port: 443”. (Requirements 2020.) The stand-alone version of Anchore Engine should have at least 4GB of RAM and between 5GB to 10GB of disk space to be able to support larger docker images that are analyzed (Quickstart 2021). After testing setup mentioned above, instance type “t2.medium” was not sufficient for this environment and it was changed to type “t2.large” that has 8GB RAM (Amazon EC2 T2 Instances n.d). Anchore Engine documentation mentions in their page that they recommend a minimum of 8GB for each service for production environments (Requirements 2020). AWS instance has the following default ports open: 22, 80, 443, and additionally port 8228, this was not necessary for the Jenkins server after adding Jenkins agent

on top of Docker. This would enable instructing Anchore Engine to add images from Workstations though.

5.2.1 Installing required software

In this implementation there was no need to install Jenkins server as it already existed. In the case of Docker and Anchore Engine they were installed the same way as in the local implementation chapter 5.1.1. Anchore compose YAML resides in root home folder, and everything was done as root user. Anchore Container Image Scanner plugin was installed to Jenkins server and used in a Jenkins job. The version of the plugin was "1.0.23" as the older version of Jenkins server did not work with the newest version "1.0.24". Similar global configuration as in chapter 5.1.2 was set for the plugin with AWS instance IP address, port 8228 pointing to Anchore Engine's service API. Jenkins server could not access Anchore Engine and Jenkins agent needed to be installed on the AWS Instance on top of docker as a container to access the same "root_default" network as the Anchore Engine compose set is in. Jenkins master is connecting to Jenkins agent via SSH connection. Jenkins user was created to AWS instance for this purpose. Jenkins master executes shell script and connects to Anchore AWS Instance via SSH starting the Jenkins agent inside the container. See Figure 49 for the deployment diagram of this implementation.

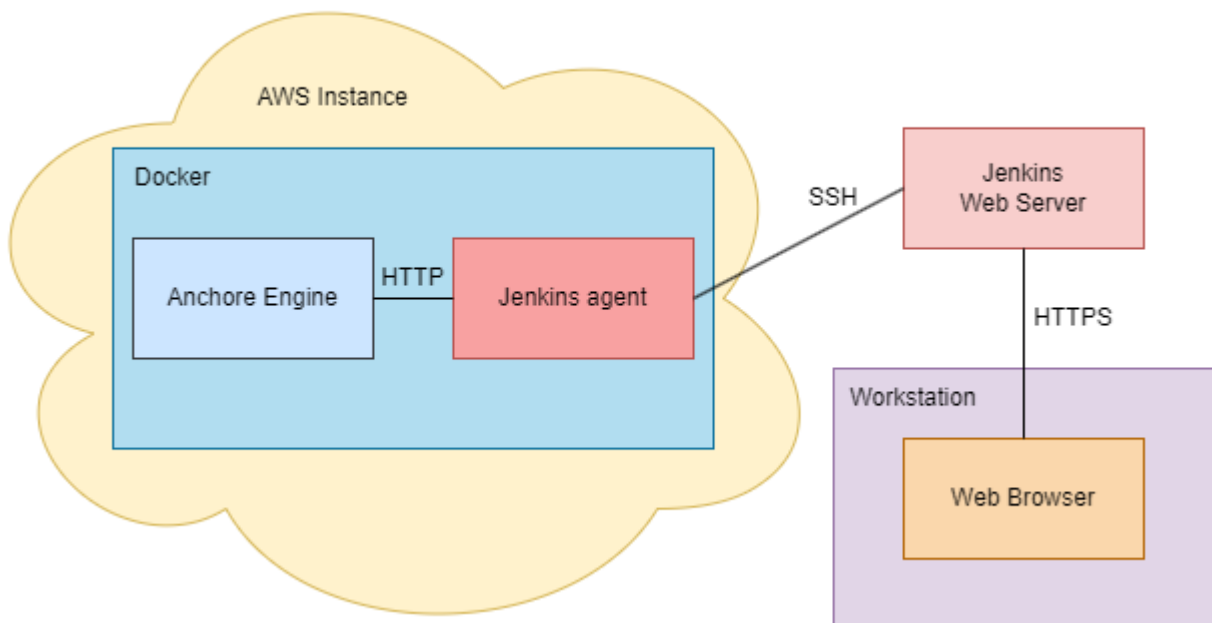


Figure 49. Production implementation deployment diagram.

5.2.2 Workflow of implementation

New Jenkins job was created to Jenkins server for Anchore analyzing. This was connected to the QRP pipeline in a way that when new QRP system-spec image is pushed to Artifactory, another Jenkins job gives the new image path to Anchore job as parameters and image is then pulled, analyzed, and evaluated by Anchore Engine. This way Anchore jobs build result "PASS" or "FAIL" is not currently intercepting the CI/CD workflow but gives analyze results on the side automatically, that the security team can go and check in the form of "Anchore Report". This implementation can be also triggered manually by giving image repository and tag as parameters "Build with Parameters" (Figure 50). Parameterization of Jenkins job was done in the job configurations "General" section by checking "This project is parameterized" and adding two string parameters. Parameters are used in the echo command that was first introduced in the local implementation chapter 5.1.2 see Figure 51. These parameters are used in both manual and automatically triggered build. Otherwise, this "Freestyle Project" based Jenkins job has similar settings to the local implementation and "Anchore Container Image Scanner" build step configurations are identical.

Project anchore-test

This build requires parameters:

DOCKER_REPO	<input type="text"/>
	Insert Docker Repo part of image, for example hello-world
DOCKER_TAG	<input type="text"/>
	Insert Docker tag part of image, for example latest

Build

Figure 50. Building Anchore job manually with parameters.

```

Execute shell
Command echo "[REDACTED]/$DOCKER_REPO:$DOCKER_TAG" > anchore_images

See the list of available environment variables

```

Figure 51. Parameterized Anchore echo command.

5.2.3 QRP image policy bundle and results

QRP policy bundle (Appendix 3) was created based on the requirements of representative of the security team and following requirements were discussed on: Policy id name shown in “Anchore Report” needs to be obvious, stop action for critical and high package vulnerabilities, warn actions for exposed port 22, stale feed data, and if vulnerability data is unavailable for some found issue. QRP policy bundle was compared with the default policy bundle (Appendix 1) and same image was evaluated with both. Empty “policybundle-qrp.json” file was created and modified for QRP policy bundle. After that it was added and activated the same way to the Anchore Engine as in the local implementation chapter 5.1.3 connecting to container as root user. “Anchore Report” results can be compared by seeing Figure 52 and Figure 53. QRP policy bundle report has a lot less “warn” actions and “Policy Id” is clear and indicative.

Anchore Policy Evaluation Summary

Show entries

Repo Tag	Stop Actions	Warn Actions
[REDACTED]	15	174

Anchore Policy Evaluation Report

Show entries

Image Id	Repo Tag	Trigger Id	Gate	Trigger	Check Output	Gate Action	Whitelisted	Policy Id
[REDACTED]	[REDACTED]	[REDACTED]	vulnerabilities	package	HIGH Vulnerability found in [REDACTED]	STOP	false	48e6f7d6-1765-11e8-b5f9-8b6f228548b6

Figure 52. Default policy bundle evaluation against QRP system-spec image.

Anchore Policy Evaluation Summary

Show entries

Repo Tag	Stop Actions	Warn Actions
[REDACTED]	15	0

Anchore Policy Evaluation Report

Show entries

Image Id	Repo Tag	Trigger Id	Gate	Trigger	Check Output	Gate Action	Whitelisted	Policy Id
[REDACTED]	[REDACTED]	[REDACTED]	vulnerabilities	package	HIGH Vulnerability found in [REDACTED]	STOP	false	QRP-system-spec-image-policy

Figure 53. QRP policy bundle evaluation against QRP system-spec image.

6 Outcome

There were three research questions defined in the research layout. This thesis was able answer to all these questions:

- 1) How to deploy artifact scanning as a part of a CI/CD pipeline?
- 2) Is the artifact scanning coverage enough or is manual scanning needed?
- 3) What is the most beneficial spot for Anchore in a CI/CD pipeline?

6.1 Research question one - How to deploy artifact scanning?

The first research question had functional requirements defined and all of those were fulfilled. Requirements can be seen from chapter two referred to as "FR1-FR8". A few requirements for Anchore Engine QRP policy bundle were set and accomplished accordingly.

Two implementations were created in this project, intention was that the local implementation works as a basis for the production implementation and gives insight on how to install applications

from scratch due to stricter permission restrictions in the production environment. Another purpose for the local implementation was to check that all the software and versions work together without issues. Two ways of deploying artifact scanning as part of CI/CD pipeline were presented, also proving that this way of implementing is versatile and works in different type of environments. Implementation was to be deployed in a way that it does not hinder the current software development lifecycle, and this was achieved by adding the Anchore analyzing as a side job that gets input from the current QRP CI/CD pipeline. Results can be read from “Anchore Report” in Jenkins and final analyzing of the results is done by the security team. If there is something to be fixed this can be initiated by the security team, thus it will not affect the workflow.

The final version of this implementation operates through Jenkins server meaning Jenkins has access to Anchore Engine and there is no need to manually do things via command line referring to first functional requirement (FR1). The second requirement (FR2) was achieved by using the Jenkins Anchore plugin and that was selected as the best option. Third requirement (FR3) succeeded by adding Anchore Engine to the same environment as the Artifactory is residing in. FR4 was carried out in two ways, local implementation included modifying of the default policy bundle and in production environment new policy bundle was created. FR5 was tested in local implementation and the same vulnerability was whitelisted from one image and left to the other. This same feature could have been applied in the production environment if required by the representative of the security team. FR6 was the result of using Jenkins Anchore plugin, as the plugin creates “Anchore Report” from the results Anchore Engine returns to Jenkins. FR7, which required the ability to manually start Anchore analyzing on chosen image(s) from Jenkins succeeded in both implementations. Last requirement (FR8) demanded automatic analyzing of new QRP images when pushed to Artifactory, this was achieved in the production environment and updated QRP system-spec image information is given through another Jenkins job to Anchore analyze Jenkins job which then proceeds with analyzing those images.

QRP policy bundle used in the production environment had requirements that were decided with the representative of the security team. There were total of 5 requirements, one was obvious policy id name to help with readability and rest of the requirements were related to “stop” and “warn” actions in the policy adjusting the results of the “Anchore Report”. This policy bundle was

created to exclude unnecessary results and leave only the important ones. All the applied requirements can be seen from QRP policy bundle JSON file (Appendix 3). The original default policy bundle gave 15 “stop” actions and 174 “warn” actions and with the QRP policy bundle there were only “15” stop actions leaving out 174 unwanted warnings. This makes analyzing the results more effective.

6.2 Research question two - Artifact scanning coverage?

The second question was related to coverage of the implemented artifact scanning. Regarding image artifact scanning Anchore Engine is very versatile and configurations can be done in various ways making it suitable for different environments. Nevertheless, there was only one scanner implemented and Anchore is scanning only docker images, which means it is not as broad and concentrates on images. If there are some other artifacts in the Artifactory not included in the image this means Anchore Engine will not scan those and that needs to be done with another means or manually. There is also a higher possibility to get false-positives and not necessarily all vulnerabilities are found if there is only one scanner. There are different artifact scanners available and many of them have their own vulnerability databases, meaning other scanners might reveal vulnerabilities that Anchore does not include. The conclusion is that Anchore does not cover everything, results are directional and there is always room for improvement. Almost every day new vulnerabilities are found, which means scanners need to be maintained to keep data up to date, as well as adjustment of the configurations is recommended from time to time.

6.3 Research question three - Most beneficial spot for Anchore?

For the third question, the most beneficial spot for Anchore was investigated. To make the process cheaper and get faster feedback Anchore should be applied in the earliest stage of the CI/CD pipeline as possible, as it is a SCA scanner. In the case of the implemented Anchore Engine the image analysis happens as soon as updated QRP system-spec image is pushed to Artifactory, giving possibility to catch vulnerabilities early on and initiate mitigation operations. This method of development enables cheaper and faster repairs.

7 Conclusion

Constructive research method was chosen for this thesis and was based on the client's need for a new security scanner. It was carried out by solving a real-life problem and creating something that can be utilized by the client. There was also a lot of collaboration between researcher and client which produced suitable solution and configurations.

The aim of this bachelor's thesis was to deploy artifact security scanner to client's environment. Resulting implementation is a manually or automatically started Anchore scanning job that works via Jenkins that is part of QRP development lifecycle. Results can be read and analyzed by the security team. This implementation gives direct input of the frequently updated QRP system-spec images enhancing the current security testing. Results are given in the form of "Anchore Report" that can be checked from Jenkins.

7.1 Advantages, deficiencies, and challenges of the implementation

After looking at different implementation possibilities for Anchore it was decided to go with the official Anchore Jenkins plugin. This way Anchore analyze could be added as part of the current QRP development lifecycle via Jenkins. Good side of this plugin implementation is that it is fast to set up even in an environment that does not have much done beforehand. It can be integrated to various environments that have Jenkins as part of CI/CD pipeline and makes it versatile. As the plugin is set as a build step in a Jenkins job it is possible to include Anchore analyzing as an individual separate job or include it as part of already existing job and workflow. If Anchore analyzing is integrated to already existing workflow that is done via Jenkins it works automatically as is, but if it is created as a separate job, it can be manually started or configured to be automatically triggered by some wanted action.

On the other hand, there are some restrictions that come with the plugin as it is a ready-made package. Every Jenkins build gives "Anchore Report" as result of the analysis and this report cannot be modified. This means there can be unnecessary information shown that can hinder the analyzing of the report, it would be good to have the possibility to modify the appearance of the report and information it includes. Another thing that can be time-consuming is when moving to more advanced configuring or if the policy bundle needs to be updated frequently. Policy bundles

need to be fetched from the Anchore Engine if not saved anywhere, and after modification the new policy bundle needs to be added and activated in Anchore Engine to take it into use. One solution would be creating a script that automatically adds and activates the modified policy bundle. Another more expensive solution would be upgrading to Anchore Enterprise, the commercial version of Anchore that offers GUI. In the GUI it is possible to upload and download the policies and store them as JSON files in the system, policies can be modified via the GUI which makes it more effortless and faster (Anchore Enterprise Architecture 2022).

As mentioned in chapter 6.2 only one scanner was deployed and Anchore only analyzes docker images, this could be counted as restriction, even though Anchore Engine is a very good tool for docker image artifact analyzing with its versatile configuration options.

There were also problems related to the production environment deployment, such as networking issues, problems with the AWS instance resources, and placement of the Jenkins agent. Jenkins agent was first installed into the Anchore AWS instance. "java-8-openjdk" package was installed into the instance for Jenkins agent to work properly. It was residing in the same machine but in a different network than the Anchore Engine, which is on top of docker. This did not work, and network errors appeared in Jenkins build logs about connection timeout towards the Anchore Engine API: "Failed to add image(s) to anchore-engine due to an unexpected error org.apache.http.conn.HttpHostConnectException: Connect to 172.18.0.6:8228 [/172.18.0.6] failed: Connection timed out (Connection timed out)". After this it was attempted to add Anchore containers on the docker host network which means sharing networking namespace with the host, in this case the AWS instance. Network mode "host" was added to "docker-compose.yml" file to each Anchore service. This option did not work as the containers did not get any IP addresses. Final solution was adding Jenkins agent as a container on top of docker and adding the container to the same "root_default" network as Anchore Engine is in. Installing java packages separately was not needed in this case. Even after this the connection timeout error did not disappear and it was caused by AWS instance resources. After the instance was changed from "t2.medium" to "t2.large" with more RAM from 4GB to 8GB, analyzing was possible and Jenkins builds were successful.

AWS instance had 500GB of disk space, but it was soon noticed that it was filled up. The reason for this was Jenkins agent container that was using up all the space by collecting logs. This was fixed by adding log rotation for the container logs. Every Anchore micro service container logging was also changed from “100m” to “10m” to save more space.

A few problems were encountered while creating policy bundles. Policy bundle needs to include mandatory JSON arrays or else it will not be accepted by the Anchore Engine and gives error. Same with the JSON objects inside the arrays if some mandatory key value pair is missing. For all the JSON arrays and objects Anchore documentation did not seem to mention which ones are mandatory and which ones are not, this required some testing. Some misconfigurations may go through unnoticed to Anchore Engine but will at the latest show in the “Anchore Report” as unwanted results.

One thing that was hindering problem solving was that Anchore does not have as much information and guides on the internet as it is a newer software. The best source of information is Anchore’s own documentation, which is different for the Anchore Engine and commercial product Anchore Enterprise. Many guides related to the Jenkins Anchore plugin were two to three years old and partly outdated. A lot of problems needed to be solved by trial and error. In that way this research could be used as an updated guide for creating a similar implementation. It is also good to notice that Anchore Engine is strict about HTTP protocol and HTTPS protocol did not work with the API calls.

7.2 Ethicalness

Ethically it is good to point out that because of the permission restrictions in the production implementation, I could not do everything on my own. Tickets were created but someone else carried out the changes needed. These were mostly related to AWS instance creation, networking, port openings, and Jenkins server configurations. The only thing that I could do in Jenkins was configure the Jenkins Anchore job itself. Global configurations and Jenkins agent setting were done by someone else. All implementation related functional requirements and requirements related to QRP policy bundle was agreed with the representative of the security team and not decided only by one person.

7.3 Utilization of outcome

Anchore scanner is currently used in the client's production environment and results can be checked by the security team. Feedback was given from the security team that the implementation is useful. The implementation will be used in the future and configured more.

7.4 Further Development

With the current implementation policy bundle will have to be manually edited, added, and activated for the Anchore Engine. As a further development this could be partly automated by adding a shell script that does the commands automatically. Or even more automated by adding policy JSON to version control and after change has been made a trigger would execute the shell script and updated policy bundle would be taken into use automatically. Most expensive option would be moving to commercial version of Anchore Engine, Anchore Enterprise that offers a GUI, which enables adding and editing policy bundles.

Quite minimal configurations were done for the QRP policy bundle as it was the first version. As a further development policy bundle could be modified to be even more precise and removing known false positives from the "Anchore Report" results. Expiration dates can be set for specific whitelist items, this could be utilized to reduce unnecessary results for a given period. "Anchore Report" results output itself cannot be modified but one improvement object could be moving "Anchore Report" results to different location or send results as an email so that the results do not have to be always checked from Jenkins server.

As Anchore is analyzing only docker images it could be a good idea to implement another scanner that scans other artifacts to get broader coverage or use existing Xray for this. As only one scanner was deployed it would be good to create some way of comparing Xray results to Anchore Engine results to get a better overview. Using multiple security scanners would enhance the rate of finding vulnerabilities as scanner vulnerability databases are not identical.

References

Accessing Registries. 2020. Anchore webpage. Accessed on 26 March 2022. Retrieved from <https://engine.anchore.io/docs/general/concepts/registries/>.

Accessing the Engine. 2020. Anchore webpage. Accessed on 26 March 2022. Retrieved from https://engine.anchore.io/docs/general/concepts/accessing_engine/.

Amazon EC2 T2 Instances. N.d. Amazon Web Services webpage. Accessed on 11 August 2022. Retrieved from <https://aws.amazon.com/ec2/instance-types/t2/>.

Analyzing Images. 2020. Anchore webpage. Accessed on 26 March 2022. Retrieved from <https://engine.anchore.io/docs/general/concepts/images/>.

Anchore Engine Installation. 2020. Anchore webpage. Accessed on 26 March 2022. Retrieved from <https://engine.anchore.io/docs/install/>.

Anchore Engine Overview. 2020. Anchore webpage. Accessed on 26 March 2022. Retrieved from <https://engine.anchore.io/docs/general/>.

Anchore Enterprise Architecture. 2022. Anchore webpage. Accessed on 17 September 2022. Retrieved from <https://docs.anchore.com/current/docs/overview/architecture/>.

Atzmony, A. 2021a. JFrog Artifactory. JFrog webpage. Accessed on 4 March 2022. Retrieved from <https://www.jfrog.com/confluence/display/JFROG/JFrog+Artifactory>.

Atzmony, A. 2021b. JFrog Xray. JFrog webpage. Accessed on 2 April 2022. Retrieved from <https://www.jfrog.com/confluence/display/JFROG/JFrog+Xray>.

AWS Well-Architected. N.d. Amazon Web Services webpage. Accessed on 6 May 2022. Retrieved from <https://aws.amazon.com/architecture/well-architected/>.

Ben-Zvi, S. 2021. What is a software artifact?. JFrog webpage. Accessed on 19 March 2022. Retrieved from <https://jfrog.com/knowledge-base/what-is-a-software-artifact/>.

Berman, D. 2021. Guide to Software Composition Analysis (SCA). Snyk webpage. Accessed on 29 April 2022. Retrieved from <https://snyk.io/blog/what-is-software-composition-analysis-sca-and-does-my-company-need-it/>.

Chacon, S. & Straub, B. 2014. Pro git: Everything you need to know about Git. Second edition. Apress. Accessed on 18 February 2022. Retrieved from <https://git-scm.com/book/en/v2>.

Chen, S. -J., Pan, Y. -C., Ma Y. -W. & Chiang C. -M. 2022. The Impact of the Practical Security Test during the Software Development Lifecycle. 2022 24th International Conference on Advanced Communication Technology (ICACT), pp. 313-316, doi: 10.23919/ICACT53585.2022.9728868. Accessed on 2 April 2022. Retrieved from <https://janet.finna.fi>, IEEE Xplore Digital Library.

CI / CD Integration. 2020. Anchore webpage. Accessed on 26 March 2022. Retrieved from https://engine.anchore.io/docs/general/concepts/integrations/ci_cd/.

CNCF Cloud Native Interactive Landscape. N.d. Cloud Native Computing Foundation webpage. Accessed on 8 April 2022. Retrieved from <https://landscape.cncf.io/>.

Containers vs. Virtual Machines (VMs): What's the Difference?. 2021. IBM webpage. Accessed on 8 December 2021. Retrieved from <https://www.ibm.com/cloud/blog/containers-vs-vms>.

Continuous Integration vs. Delivery vs. Deployment. N.d. JetBrains webpage. Accessed on 11 March 2022. Retrieved from <https://www.jetbrains.com/teamcity/ci-cd-guide/continuous-integration-vs-delivery-vs-deployment/>.

DevOps: 8 Reasons for DevOps to use a Binary Repository Manager. 2021. JFrog webpage. Accessed on 18 March 2022. Retrieved from <https://jfrog.com/whitepaper/devops-8-reasons-for-devops-to-use-a-binary-repository-manager/>.

DevSecOps. 2020. IBM webpage. Accessed on 11 March 2022. Retrieved from <https://www.ibm.com/cloud/learn/devsecops>.

Docker overview. N.d. Docker documentation webpage. Accessed on 11 November 2021. Retrieved from <https://docs.docker.com/get-started/overview/>.

Docker. 2021. IBM webpage. Accessed on 3 November 2021. Retrieved from <https://www.ibm.com/cloud/learn/docker>.

Dörnenburg, E. 2018. The Path to DevOps. IEEE Software, vol. 35, no. 5, pp. 71-75, doi: 10.1109/MS.2018.290110337. Accessed on 11 March 2022. Retrieved from <https://janet.finna.fi>, IEEE Xplore Digital Library.

Dynamic Application Security Testing (DAST). 2014. Techopedia webpage. Accessed on 2 April 2022. Retrieved from <https://www.techopedia.com/definition/30958/dynamic-application-security-testing-dast>.

El Rhaffari, I. & Roudies, O. 2013. Reducing the gap between security audit and software engineering methods. 2013 Science and Information Conference, pp. 255-262. Accessed on 15 February 2022. Retrieved from <https://janet.finna.fi>, IEEE Xplore Digital Library.

Frequently Asked Questions. 2020. Anchore webpage. Accessed on 26 March 2022. Retrieved from <https://docs.anchore.com/3.0/docs/faq/>.

Glossary. 2021. Kubernetes webpage. Accessed on 5 March 2022. Retrieved from <https://kubernetes.io/docs/reference/glossary/>.

Grype Integration. 2021. Anchore webpage. Accessed on 26 March 2022. Retrieved from <https://engine.anchore.io/docs/grype/>.

Gunja, S. 2021. What is DevOps? Unpacking the rise of an IT cultural revolution. Dynatrace webpage. Accessed on 11 March 2022. Retrieved from <https://www.dynatrace.com/news/blog/what-is-devops/>.

Image Analysis Process. 2020. Anchore webpage. Accessed on 26 March 2022. Retrieved from <https://engine.anchore.io/docs/general/concepts/images/analysis/>.

Image and Tag Watchers. 2020. Anchore webpage. Accessed on 26 March 2022. Retrieved from <https://engine.anchore.io/docs/general/concepts/images/watchers/>.

InsightVM Features. N.d. Rapid7 webpage. Accessed on 2 April 2022. Retrieved from <https://www.rapid7.com/products/insightvm/features/>.

Jenkins User Documentation. N.d. Jenkins webpage. Accessed on 4 March 2022. Retrieved from <https://www.jenkins.io/doc/>.

Jenkins. N.d. Jenkins webpage. Accessed on 4 March 2022. Retrieved from <https://www.jenkins.io/>.

Jfrog artifactory. N.d. JFrog webpage. Accessed on 19 March 2022. Retrieved from <https://jfrog.com/artifactory/>.

Kananen, J. 2015. Opinnäytetyön kirjoittajan opas: näin kirjoitan opinnäytetyön tai pro gradun alusta loppuun. Jyväskylä: Jyväskylän ammattikorkeakoulu, Liiketoimintayksikkö.

Kshitiz, S. 2021. Installing and configuring Jenkins in Linux. Red Hat webpage. Accessed on 4 March 2022. Retrieved from <https://www.redhat.com/sysadmin/install-jenkins-rhel8>.

Kubernetes. N.d. Kubernetes webpage. Accessed on 5 March 2022. Retrieved from <https://kubernetes.io/>.

Labouardy, M. 2021. Pipeline as Code. Manning Publications. Accessed on 11 March 2022. Retrieved from <https://livebook.manning.com/book/pipeline-as-code/>.

Lago, P. 2019. Architecture Design Decision Maps for Software Sustainability. 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS), pp. 61-64, doi: 10.1109/ICSE-SEIS.2019.00015. Accessed on 14 February 2022. Retrieved from <https://janet.finna.fi>, IEEE Xplore Digital Library.

Lukka, K. 2001. Konstruktiivinen tutkimusote. Metodix webpage. Accessed on 20 May 2022. Retrieved from <https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/>.

Masarwa, M. 2021. What is an artifact repository?. JFrog webpage. Accessed on 4 March 2022. Retrieved from <https://jfrog.com/knowledge-base/what-is-an-artifact-repository/>.

Networking in Compose. N.d. Docker documentation webpage. Accessed on 19 August 2022. Retrieved from <https://docs.docker.com/compose/networking/>.

Nodes. 2022. Kubernetes webpage. Accessed on 5 March 2022. Retrieved from <https://kubernetes.io/docs/concepts/architecture/nodes/>.

O'Brien, T. 2010. "Why Nexus?" for the Non-Programmer. Sonatype webpage. Accessed on 18 March 2022. Retrieved from <https://blog.sonatype.com/2010/04/why-nexus-for-the-non-programmer/>.

Open Source Vulnerability Scanning: Methods and Top 5 Tools. N.d. Aqua Security webpage. Accessed on 2 April 2022. Retrieved from <https://www.aquasec.com/cloud-native-academy/vulnerability-management/open-source-vulnerability-scanning/>.

Overview of SCA Tools: Core Features and Benefits of Deployment. 2021. Debricked webpage. Accessed on 29 April 2022. Retrieved from <https://debricked.com/blog/sca-tools-overview/>.

Paine, L. 2022. DAST, IAST, SCA: Deeper coverage in a single scan. Invicti webpage. Accessed on 29 April 2022. Retrieved from <https://www.invicti.com/blog/web-security/dast-iaast-sca-deeper-coverage-single-scan/>.

Policy Bundles. 2020. Anchore webpage. Accessed on 26 March 2022. Retrieved from <https://engine.anchore.io/docs/general/concepts/policy/bundles/>.

Policy. 2020. Anchore webpage. Accessed on 26 March 2022. Retrieved from <https://engine.anchore.io/docs/general/concepts/policy/>.

Quickstart. 2021. Anchore webpage. Accessed on 11 August 2022. Retrieved from <https://engine.anchore.io/docs/quickstart/>.

Qvantel Flex BSS. N.d. Qvantel webpage. Accessed on 19 May 2022. Retrieved from <https://www.qvantel.com/flex-bss>.

Qvantel LinkedIn. N.d. Qvantels LinkedIn page. Accessed on 19 May 2022. Retrieved from <https://www.linkedin.com/company/qvantel/about/>.

Qvantel Oy. N.d. Kauppalehti webpage. Accessed on 19 May 2022. Retrieved from <https://www.kauppalehti.fi/yrietykset/yrietykset/qvantel+oy/20580807>.

Requirements. 2020. Anchore webpage. Accessed on 11 August 2022. Retrieved from <https://engine.anchore.io/docs/install/requirements/>.

Sengupta, S. 2021. SAST, DAST, IAST, RASP: alphabet soup explained. Crashtest Security webpage. Accessed on 2 April 2022. Retrieved from <https://crashtest-security.com/sast-dast-iaast-rasp/>.

Sharma, A. 2018a. A Brief History of DevOps, Part I: Waterfall. CircleCI webpage. Accessed on 27 April 2022. Retrieved from <https://circleci.com/blog/a-brief-history-of-devops-part-i-waterfall/>.

Sharma, A. 2018b. A Brief History of DevOps, Part II: Agile Development. CircleCI webpage. Accessed on 27 April 2022. Retrieved from <https://circleci.com/blog/a-brief-history-of-devops-part-ii-agile-development/>.

Software Composition Analysis (SCA): What You Should Know. N.d. Aqua Security webpage. Accessed on 29 April 2022. Retrieved from <https://www.aquasec.com/cloud-native-academy/devsecops/software-composition-analysis-sca/>.

Springett, S. N.d. Component Analysis. OWASP Foundation webpage. Accessed on 2 April 2022. Retrieved from https://owasp.org/www-community/Component_Analysis.

Subscriptions. 2020. Anchore webpage. Accessed on 26 March 2022. Retrieved from <https://engine.anchore.io/docs/general/concepts/subscriptions/>.

Toikko, T. & Rantanen, T. 2009. Tutkimuksellinen kehittämistoiminta. Näkökulmia kehittämissessiin, osallistamiseen ja tiedontuotantoon. Tampere: Tampere University Press. https://trepo.tuni.fi/bitstream/handle/10024/100802/Toikko_Rantanen_Tutkimuksellinen_kehittamistoiminta.pdf.

Using the Anchore CLI. 2020. Anchore webpage. Accessed on 26 March 2022. Retrieved from https://engine.anchore.io/docs/usage/cli_usage/.

Version Control Software: An Overview. N.d. Bitbucket webpage. Accessed on 18 February 2022. Retrieved from <https://bitbucket.org/product/version-control-software>.

Vulnerability Scanning Tools. N.d. OWASP Foundation webpage. Accessed on 2 April 2022. Retrieved from https://owasp.org/www-community/Vulnerability_Scanning_Tools.

What is a Container?. N.d. Docker webpage. Accessed on 11 November 2021. Retrieved from <https://www.docker.com/resources/what-container>.

What is a Hypervisor?. N.d. VMware webpage. Accessed on 8 December 2021. Retrieved from <https://www.vmware.com/topics/glossary/content/hypervisor>.

What is a Kubernetes cluster?. N.d. VMware webpage. Accessed on 5 March 2022. Retrieved from <https://www.vmware.com/topics/glossary/content/kubernetes-cluster.html>.

What Is a Vulnerability Scan, and Why Is It Important?. N.d. NETdepot webpage Accessed on 19 May 2022. Retrieved from <https://www.netdepot.com/blog/what-is-a-vulnerability-scan-and-why-is-it-important>.

What Is an Artifact? Everything You Need to Know. 2020. Artifacts webpage. Accessed on 19 March 2022. Retrieved from <https://artifacts.ai/what-is-an-artifact/>.

What is cloud computing?. N.d.a. Amazon Web Services webpage. Accessed on 17 March 2022. Retrieved from <https://aws.amazon.com/what-is-cloud-computing/>.

What is cloud computing?. N.d.b. Microsoft Azure webpage. Accessed on 17 March 2022. Retrieved from <https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/>.

What is DevOps? Waterfall to DevOps 2.0. 2018. Astadia webpage. Accessed on 27 April 2022. Retrieved from <https://www.astadia.com/blog/what-is-devops-a-complete-history-waterfall-to-devops-2-0>.

What is DevOps?. N.d. Amazon Web Services webpage. Accessed on 11 March 2022. Retrieved from <https://aws.amazon.com/devops/what-is-devops/>.

What is DevSecOps?. 2018. Red Hat webpage. Accessed on 11 March 2022. Retrieved from <https://www.redhat.com/en/topics/devops/what-is-devsecops>.

What is Jenkins?. N.d. CloudBees webpage. Accessed on 4 March 2022. Retrieved from <https://www.cloudbees.com/jenkins/what-is-jenkins>.

What is Kubernetes?. 2020. Red Hat webpage. Accessed on 5 March 2022. Retrieved from <https://www.redhat.com/en/topics/containers/what-is-kubernetes>.

What is Kubernetes?. 2021. Kubernetes webpage. Accessed on 5 March 2022. Retrieved from <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>.

What is Software Engineering?. N.d. Cast software webpage. Accessed on 4 February 2022. Retrieved from <https://www.castsoftware.com/glossary/what-is-software-engineering-definition-types-of-basics-introduction>.

What is version control?. N.d. Atlassian webpage. Accessed on 18 February 2022. Retrieved from <https://www.atlassian.com/git/tutorials/what-is-version-control>.

Why Docker?. N.d. Docker webpage. Accessed on 3 November 2021. Retrieved from <https://www.docker.com/why-docker>.

Wichers, D., itamarlavender, will-obrien, Worcel, E., Subramanian, P., kingthorin, coadaflorin, hblankenship, GovorovViva64, pfhorman, GouveaHeitor, Gibler, C., DSotnikov, Abraham, A., Rathaus, N. & Jang, M. N.d. Source Code Analysis Tools. OWASP Foundation webpage. Accessed on 2 April 2022. Retrieved from https://owasp.org/www-community/Source_Code_Analysis_Tools.

Appendices

Appendix 1. Anchore Engine default policy bundle JSON file.

```

GNU nano 2.9.8                                policybundle.json
{
  "blacklisted_images": [],
  "comment": "Default bundle",
  "id": "2c53a13c-1765-11e8-82ef-23527761d060",
  "mappings": [
    {
      "id": "c4f9bf74-dc38-4ddf-b5cf-00e9c0074611",
      "image": {
        "type": "tag",
        "value": "*"
      },
      "name": "default",
      "policy_id": "48e6f7d6-1765-11e8-b5f9-8b6f228548b6",
      "registry": "*",
      "repository": "*",
      "whitelist_ids": [
        "37fd763e-1765-11e8-add4-3b16c029ac5c"
      ]
    }
  ],
  "name": "Default bundle",
  "policies": [
    {
      "comment": "System default policy",
      "id": "48e6f7d6-1765-11e8-b5f9-8b6f228548b6",
      "name": "DefaultPolicy",
      "rules": [
        {
          "action": "STOP",
          "gate": "dockerfile",
          "id": "ce7b8000-829b-4c27-8122-69cd59018400",
          "params": [
            {
              "name": "ports",
              "value": "22"
            },
            {
              "name": "type",
              "value": "blacklist"
            }
          ],
          "trigger": "exposed_ports"
        },
        {
          "action": "WARN",
          "gate": "dockerfile",
          "id": "312d9e41-1c05-4e2f-ad89-b7d34b0855bb",
          "params": [
            {
              "name": "instruction",
              "value": "HEALTHCHECK"
            },
            {
              "name": "check",
              "value": "not_exists"
            }
          ],
          "trigger": "instruction"
        },
        {
          "action": "WARN",
          "gate": "vulnerabilities",
          "id": "6b5c14e7-a6f7-48cc-99d2-959273a2c6fa",
          "params": [
            {
              "name": "max_days_since_sync",
              "value": "2"
            }
          ],
          "trigger": "stale_feed_data"
        },
        {
          "action": "WARN",
          "gate": "vulnerabilities",
          "id": "3e79ea94-18c4-4d26-9e29-3b9172a62c2e",
          "params": [],
          "trigger": "vulnerability_data_unavailable"
        }
      ]
    }
  ]
}

```

```

    {
      "action": "WARN",
      "gate": "vulnerabilities",
      "id": "6063fdde-b1c5-46af-973a-915739451ac4",
      "params": [
        {
          "name": "package_type",
          "value": "all"
        },
        {
          "name": "severity_comparison",
          "value": "="
        },
        {
          "name": "severity",
          "value": "medium"
        }
      ],
      "trigger": "package"
    },
    {
      "action": "STOP",
      "gate": "vulnerabilities",
      "id": "b30e8abc-444f-45b1-8a37-55be1b8c8bb5",
      "params": [
        {
          "name": "package_type",
          "value": "all"
        },
        {
          "name": "severity_comparison",
          "value": ">"
        },
        {
          "name": "severity",
          "value": "medium"
        }
      ],
      "trigger": "package"
    }
  ],
  "version": "1_0"
},
"version": "1_0",
"whitelisted_images": [],
"whitelists": [
  {
    "comment": "Default global whitelist",
    "id": "37fd763e-1765-11e8-add4-3b16c029ac5c",
    "items": [],
    "name": "Global Whitelist",
    "version": "1_0"
  }
]
}

```

Appendix 2. Modified Anchore Engine default policy bundle JSON file for local implementation.

```
{
  "blacklisted_images": [],
  "comment": "Default bundle",
  "id": "2c53a13c-1765-11e8-82ef-23527761d060",
  "mappings": [
    {
      "id": "c4f9bf74-dc38-4ddf-b5cf-00e9c0074611",
      "image": {
        "type": "tag",
        "value": "php8.1"
      },
      "name": "default",
      "policy_id": "48e6f7d6-1765-11e8-b5f9-8b6f228548b6",
      "registry": "*",
      "repository": "*",
      "whitelist_ids": [
        "37fd763e-1765-11e8-add4-3b16c029ac5c"
      ]
    },
    {
      "image": {
        "type": "tag",
        "value": "*"
      },
      "name": "testmapping",
      "policy_ids": [
        "policytest"
      ],
      "registry": "*",
      "repository": "*",
      "whitelist_ids": [
        "whitelist6.0"
      ]
    }
  ],
  "name": "Default bundle",
  "policies": [
    {
      "comment": "System default policy",
      "id": "48e6f7d6-1765-11e8-b5f9-8b6f228548b6",
      "name": "DefaultPolicy",
      "rules": [
        {
          "action": "STOP",
          "gate": "dockerfile",
          "id": "ce7b8000-829b-4c27-8122-69cd59018400",
          "params": [
            {
              "name": "ports",
              "value": "22"
            },
            {
              "name": "type",
              "value": "blacklist"
            }
          ],
          "trigger": "exposed_ports"
        },
        {
          "action": "WARN",
          "gate": "dockerfile",
          "id": "312d9e41-1c05-4e2f-ad89-b7d34b0855bb",
          "params": [
            {
              "name": "instruction",
              "value": "HEALTHCHECK"
            },
            {
              "name": "check",
              "value": "not_exists"
            }
          ],
          "trigger": "instruction"
        }
      ]
    }
  ]
}
```

```

    {
      "action": "WARN",
      "gate": "vulnerabilities",
      "id": "6b5c14e7-a6f7-48cc-99d2-959273a2c6fa",
      "params": [
        {
          "name": "max_days_since_sync",
          "value": "2"
        }
      ],
      "trigger": "stale_feed_data"
    },
    {
      "action": "WARN",
      "gate": "vulnerabilities",
      "id": "3e79ea94-18c4-4d26-9e29-3b9172a62c2e",
      "params": [],
      "trigger": "vulnerability_data_unavailable"
    },
    {
      "action": "WARN",
      "gate": "vulnerabilities",
      "id": "6063fdde-b1c5-46af-973a-915739451ac4",
      "params": [
        {
          "name": "package_type",
          "value": "all"
        },
        {
          "name": "severity_comparison",
          "value": "="
        },
        {
          "name": "severity",
          "value": "medium"
        }
      ],
      "trigger": "package"
    },
    {
      "action": "STOP",
      "gate": "vulnerabilities",
      "id": "b30e8abc-444f-45b1-8a37-55be1b8c8bb5",
      "params": [
        {
          "name": "package_type",
          "value": "all"
        },
        {
          "name": "severity_comparison",
          "value": ">"
        },
        {
          "name": "severity",
          "value": "medium"
        }
      ],
      "trigger": "package"
    }
  ],
  "version": "1_0"
},

```

```

{
  "comment": "Policy testing",
  "id": "policytest",
  "name": "Policy for wordpress6.0",
  "rules": [
    {
      "action": "STOP",
      "gate": "vulnerabilities",
      "id": "rule123",
      "params": [
        {
          "name": "package_type",
          "value": "all"
        },
        {
          "name": "severity_comparison",
          "value": ">"
        },
        {
          "name": "severity",
          "value": "medium"
        }
      ],
      "trigger": "package"
    }
  ],
  "version": "1_0"
},
{
  "comment": "Default global whitelist",
  "id": "37fd763e-1765-11e8-add4-3b16c029ac5c",
  "items": [
    {
      "gate": "vulnerabilities",
      "id": "testrule1",
      "trigger_id": "CVE-2021-30474+*"
    }
  ],
  "name": "Global Whitelist",
  "version": "1_0"
},
{
  "id": "whitelist6.0",
  "items": [],
  "name": "whitelist wordpress 6.0",
  "version": "1_0"
}
]
}

```

Appendix 3. QRP policy bundle JSON file.

```

GNU nano 2.9.8 policybundle-qrp.json
{
  "id": "policy-bundle-qrp",
  "version": "1_0",
  "name": "Policy bundle QRP system-spec images",
  "comment": "This bundle evaluates artifacts found from QRP system-spec docker images",
  "whitelisted_images": [],
  "blacklisted_images": [],
  "mappings": [
    {
      "id": "all-qrp-images",
      "image": {
        "type": "tag",
        "value": "*"
      },
      "name": "default",
      "policy_id": "QRP-system-spec-image-policy",
      "registry": "*",
      "repository": "*",
      "whitelist_ids": [
        "qrp-whitelist"
      ]
    }
  ],
  "whitelists": [
    {
      "comment": "QRP images whitelisting",
      "id": "qrp-whitelist",
      "items": [],
      "name": "QRP Whitelist",
      "version": "1_0"
    }
  ],
  "policies": [
    {
      "name": "Policy QRP system-spec images",
      "version": "1_0",
      "comment": "Includes policies set for QRP system-spec images",
      "id": "QRP-system-spec-image-policy",
      "rules": [
        {
          "action": "STOP",
          "gate": "vulnerabilities",
          "id": "high-critical-packages",
          "params": [
            {
              "name": "package_type",
              "value": "all"
            },
            {
              "name": "severity_comparison",
              "value": ">"
            },
            {
              "name": "severity",
              "value": "medium"
            }
          ],
          "trigger": "package"
        },
        {
          "action": "WARN",
          "gate": "dockerfile",
          "id": "docker-port22-exposed",
          "params": [
            {
              "name": "ports",
              "value": "22"
            },
            {
              "name": "type",
              "value": "blacklist"
            }
          ],
          "trigger": "exposed_ports"
        },
        {
          "action": "WARN",
          "gate": "vulnerabilities",
          "id": "warn-stale-data-2days",
          "params": [
            {
              "name": "max_days_since_sync",
              "value": "2"
            }
          ],
          "trigger": "stale_feed_data"
        },
        {
          "action": "WARN",
          "gate": "vulnerabilities",
          "id": "no-vuln-data",
          "params": [],
          "trigger": "vulnerability_data_unavailable"
        }
      ]
    }
  ]
}

```

```
trigger : vulnerability_data_unavailable  
  }  
  ]  
}
```