Quality Assurance and Traceability in Containerized Continuous Delivery Process

Oyewale Adedayo Oyelami, Alexander Poth, Johannes Hintsch, Andreas Riel

Quality Assurance and Traceability in Containerized Continuous Delivery Process

Oyewale Adedayo Oyelami[1], Alexander Poth[2], Johannes Hintsch[3], Andreas Riel[4]

[1/2] Volkswagen AG, Berliner Ring 2, D-38436 Wolfsburg, Germany
{oyewale.adedayo.oyelami|alexander.poth}@volkswagen.de
[3] Magdeburg Research and Competence Cluster, Universitätsplatz 2, D-39104 Magdeburg, Germany
johannes.hintsch@ovgu.de
[4] Grenoble Alps University, F-38031 Grenoble, France
andreas.riel@grenoble-inp.fr

Abstract. Companies embrace digitalization to have a shorter time to market. This drive for digitalization also demands that the quality of products and services is high. Containerization is one of the concepts that are core to achieving these objectives in information technology today as it is central to the development and delivery of software products. Therefore, to meet the expectations, the containerization process needs state-of-the-art quality assurance. To achieve this, a quality assurance methodical procedure is being developed with bi-directional traceability. Its usefulness is demonstrated by integrating and successfully applying it into the continuous integration and delivery (CI/CD) pipeline of the Volkswagen Group IT.

Keywords: Containers, Containerization Process, Agile Software Development, Quality Assurance (QA), Bi-Directional Traceability, CI/CD

1 Motivation

Fast and reliable software product delivery is crucial in today's software projects. IT companies as well as companies in different sectors, such as the automobile industry, are performing software product delivery in increasingly complex scenarios. To many companies, also and particularly automobile, quality assurance and traceability are important requirements for software product delivery. A key enabler of fast and reliable software product delivery has been containerization, augmenting existing processes and methodologies such as Agile and DevOps [1].

Containerization itself can be described as the process of performing operating-system-level virtualization by a software thereby allowing the existence of multiple isolated user-space instances called ***containers***. Therefore, software applications are "containerized" to harness the benefits and possibilities of a level of abstraction away from the host system, easier scalability, simpler dependency management, shared layering, composability, predictability and application versioning [2].

The need for containerization has therefore been growing due to its relevance in solving the challenges attributed to the delivery of software products. Challenges such

as maintenance, reliability, migration after development are part of the issues containerization is addressing which have led to its continual relevance growth [2].

The collective chaining of all the points where containers are being or can be used, how they are being used and what they are being used for in the software product journey from development to final delivery is what is contextually described as ”Containerization Process” here. It can, therefore, be said to be an individual or customizable process as its structure and components will vary from one organization to another depending on adoption level, obtainable level of expertise, capital and time, preferences, which all can contribute to the adoption depth.

However, the role and importance of traceability in this process can never be overstated as we need to be able to map and track the changes, stages and effects for a software product from the initiation (requirement) phase to the final phase, reflectively in a continuous delivery setup. A practical benefit is the root cause of failure, which can easily be traced from appearance location back to its origin. This is important for enterprises, for them to be responsive in the most obtainable shortest duration as they seek to stay ahead of quality assurance issues which can range from legal, meeting user requirements, cost-effectiveness to security [3].

Consequently, this paper proposes, based on detailed requirements and tool analysis, a methodology to achieve quality assurance and traceability in the delivery of software products that are containerized. This methodology was evaluated in an internal process improvement project of the Volkswagen Group IT.

The rest of the paper is organized as follows: after introducing the topic as well as the motivation for carrying out this work in the first section, we described the contextual description of a containerization process as well as quality assurance in the second section. In the third section, we described our methodical procedure for containerization quality assurance, the role of bi-directional traceability and a generic implementation approach. We concluded in the fourth section by itemizing the vital points necessary for consideration in implementing the methodical procedure and in the last section, we describehow this work fits into the SPI manifesto.

2 Containerization Process and Software Quality Assurance

Sequel to its earlier succinct description as a type of virtualization, containerization uses the same kernel as that of the host to run multiple virtual environments tagged “Containers”. These containers provide the isolated environments with required resources; memory, network, storage etc., which can be shared with the host or reside completely in the container as exemplified in Figure 1 and this makes them capable of executing application [22].

Container’s architectural design make it relevant for several use-cases depending on the depth of adoption by each organization. There is the recommended use-cases category, which is documented and supported officially by the provider of the container-type, such as configuration simplification, multi-tenancy and deployment efficiency. There also exists the common use-cases category which is prevalent among application

developers, using it for software packaging, delivery and dependency management. Security and integration within infrastructure are use-cases that have also been adopted among cloud providers [1].

Given that continuous integration and delivery generally cut-across the three categories described. Using containers to achieve any of the use-cases, chaining the point of usage together in a continuous delivery approach is what we tagged "Containerization Process" and our example of such is described later.

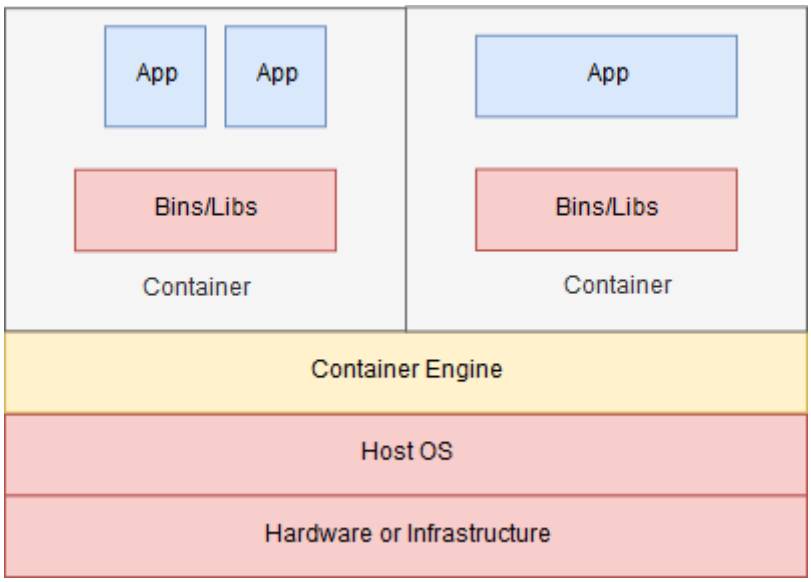

Figure 1: Architectural representation of a Container

Quality Assurance can be described as an orchestrated, systemic flow of actions, accessed necessary to ascertain the adequate level of confidence that an item or product must undergo to meet desired requirements, technically or otherwise.

- Quality: This is the summation of the characteristics of a product or service, which encapsulate its ability to meet the purpose of its creation.
- Quality Control: These are the actions in the assurance process where the characteristics of the product or service are measures against established or known metrics. Decisions, which are solely a function of the outcome of the evaluation are designed to be part of these actions. [3]

Software Quality Assurance (SQA) as defined in [4] is a process for providing adequate assurance that the specified requirements and established plans are adhered to through the product life cycle. These requirements and plans are the standards that define the quality characteristics of the product as well as the yardsticks with which the quality is evaluated [5]. The seeming complexity associated with an ecosystem of soft-

ware delivery from repositories, orchestrators to platforms with a high significant degree of automation makes the assurance of the quality thereof to be of utmost importance as the presence of containers in this generally enhance the ecosystem central part [1].

Quality assurance of (preferred and established stages) the containerization process as defined by each organization is important as it is a determinant of the quality of the product delivered thereof. This quality assurance can be categorized into compliance and monitoring, where the former contains the set of actions taken before the deployment and the latter contains the set taken to ensure that the deployed artifact stays immutable as designed and desired. Few of the possible and already established containerization issues are kernel exploitation, address resolution protocol spooling, compromised secrets, poisoned container image [6], denial of service [6, 8], container breakout [6, 7], as well as free or open-source license compliance.

3 Traceability in Containerized Process

Software Traceability which is also an important element of the qualities of a well-designed software system [9] can be defined as "the capability to monitor or study out in detail, or step by step, the history of a certain action of a procedure" [10], is now being extended and infused into the quality assurance process as the gap between both fields of software development and operations continues to fuse into a singular delivery process. Therefore, relating artifacts created during software development to describe the system from multiple perspectives is a precise description of the concept of traceability.

The major importance of traceability is first to: achieve process compliance and product improvement which is obtainable when it is performed as part of the product development life cycle and the outcome is directly applied on the product under consideration. Secondly, for software understanding and reuse which is when it is carried out on a completed project data for use in product and process analysis. [11]. Implementing traceability at every step and stages of a typical containerization process which is triggered by the respective project changes is the easiest and most efficient way to achieve this [12].

There exists also, attributed software delivery improvements due to the presence of traceability which is most effective if done in full automation [17] as it tracks, uniquely, software and their changes along the process from the initiation to the final stage.

Reliability, availability and fault tolerance are part of the identified concerns of high assurance systems and are factors that can influence them. The identified aspects of traceability that can be considered to mitigate these factors are [13]:

i. how a decision is made, hierarchically or not;
ii. visibility of the decisions made;
iii. the granularity of the architectural impact of the decision;
iv. architectural trade off if there is any;
v. the richness of the decision made as regard semantics.

3.1 Methodical Procedure for Quality Assurance of Containerization Process

The robustness and level of containerization process will vary from one organization to another; a reflection of the agreed objectives which will also determine how expansive or not the methodical quality assurance procedure will be. After an exhaustive examination of our system, we summarized our assurance objectives to be: security, performance, compliance and automation and therefore our methodical procedure contains six core steps described as follows:

i. Selection of Base Image

This is the first step of the process and it basically involves the selection of the starting image on which the eventual container to be built will be based. The selection at this stage will mostly be manual and likely to be made by the Engineer or the Person-in-charge of the product, who likely has an end-to-end overview of what the final product should look like. The technologies to be used in building the said application, the targeted platform of deployment, as well as the desired artifact size, are three notable factors that were identified with the possibility of influencing this decision [14].

There is also the possibility of using Distroless (language-focused container images without the operating system components) [21] images which do not generally negate the factors mentioned above but have the added advantage of reducing the possible surface area of attack, and disabling of secondary modifications.

ii. Code Security Checks

Considering the higher percentage of cyber-attacks recorded, with 84% to have taken place at the application layer of software systems [18] it is therefore very important to ensure that this is adequately mitigated against to further enhance the assurance of our quality. This is analyzing the codebase of the product being built as its being modified to identify the exploitable style, patterns and configurations.

This step ensures that the security level of the containerized application is therefore up to the acceptable standard at the organization as well as general standards and best practices. Security vulnerabilities such as SQL Injection, Cross-Site Request Forgery, Insecure server configuration and communication, cross-site scripting and command injection which are also part of the Open Web Application Security Project (OWASP) annual top ten lists need to be a check on any application.

iii. Dockerfile Validation

The vast majority of Docker images are built using the file called ***Dockerfile***; which determines what is or not possible in the image built thereof. Validation, as used here, is basically "linting" the Dockerfile. Linting is the process of passing source code or script through a tool generally called the "linter"; which analyzes and flags any form of errors or non-conformation to preferred best practices standard [10].

Therefore, validation of Dockerfile's compliance with best and agreed practices will further improve the quality of the image being produced especially as regards functionality and reliability.

iv. Free and Open Source Software (FOSS) Compliance Checks

There are multiple software existing licenses; Apache License 2.0, MIT license, GNU General Public License (GPL) just to mention but a few [16], from which software creators can choose from depending mainly on their underlying purpose of creating or distributing the software. Every user is meant to adhere to the terms and conditions of a software, either in the context of distribution, contribution or usage.

This step ensures that all the FOSS dependencies used and their context of usage in the containerized application is in compliance with the license. There are several compliance software tools which can be used to achieve this.

v. Dependencies Security Checks

Security checks on all the dependencies used in the application, especially for vulnerabilities is a good approach to further enhance and ascertain the quality of the resulting containers and the process. This checks go beyond FOSS checks but include all forms of dependencies, proprietary or not, application or container level dependencies to further enhance the overall security of the product produced.

vi. Container and Infrastructure Testing

There also exists the concept of testing containers base on what is expected of it especially in terms of performance and reliability and that of the infrastructure on which they will be deployed to ensure that negative quality deviations as a result of container's destination are prevented. Testing of the infrastructure is very important because it hosts the application and therefore must conform to the similar standard.

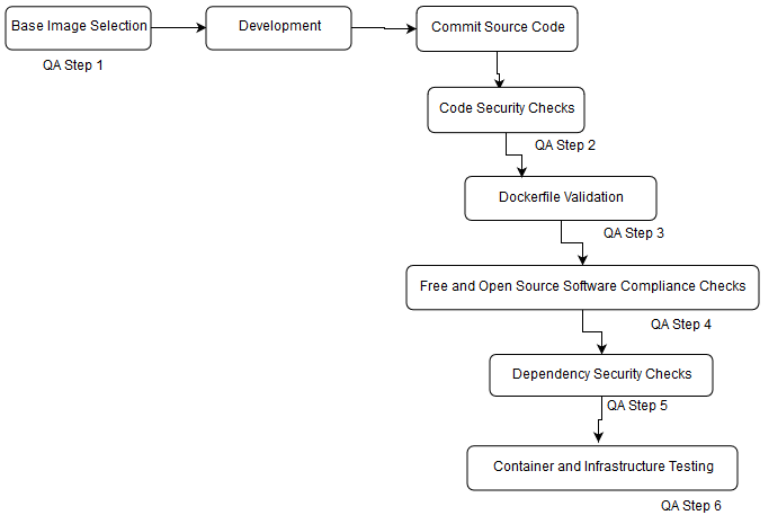

Fig. 2. A methodical procedure for Quality Assurance of Containerization Process

3.2 Generic Methodology to Bi-Directional Traceability Implementation

Infusing bi-directional traceability into a containerization quality assurance process is largely dependent on how comprehensive the process is as well as the versatility of the tools adopted for implementation by the respective organization. For our six-step methodical procedure, we described here, an abstracted flow of our implementation.

The containerization process, which got initiated, automatically by changes to the application source code is being monitored to effectively offer this bi-directional traceability feature. In the implementation, the output of every step is tagged aggregately based on the chosen tagging implementation for each step and the aggregated tag is what the final artifact carries. The final artifact produced by the process can, therefore, be validated for completeness before moving it to the next stage in the deployment pipeline.

The final artifact with the aggregated tag can be used as the release tag for the stage in the system. It thereby becomes feasible to trace, in both ways, the initiated code changes, which repository or change led to a final tag as well as the last step completed, successfully or not.

Table 1 shows a trial sample of three initiation of the containerized process by a Project A.

Attempts/Procedure Steps	1	2	3	4	5	6	Final Artifacts
I	X	Xa	Xab				Xab
II	Y	Ya	Yab	Yabc			Yabc
III	Z	Za	Zab	Zabc	Zabcd	Zabcde	Zabcde

The X, Y and Z as used can be the unique commit identification in case of a source-code versioned initiated process or any other preferred means of uniquely identifying changes to the project. The alphabets a, b, c, d and e, as used, also represents the associated tag of each step and should be carefully aggregated in such a way that they can easily be used and process in the context of the final artifact. Keeping in mind that only artifacts that passes the entire procedure successfully should be considered for use in production.

Notable points for the implementation are that:

i. The steps are procedural.
ii. All the sub-actions in each of the steps are grouped together as one and therefore seen as such at the traceability level.
iii. The tagging scheme or algorithm for each step should be unique and aggregated along the process.
iv. Our entire infrastructure complements this process, especially as regards toolchain and artifacts management.

Considering the context of containerization process of each organization, there is a need to carefully examine possible existing structure, evaluation of tools, quality assurance objectives as well as available resources; tools, infrastructure and expertise. This we have done, which reflect on the scope of our process and procedure, tools adopted for usage and our implementation.

4 Conclusion

In this paper, we have presented a contribution to making industrial IT quality assurance more agile thanks to a procedure for the containerization of the continuous delivery process. We have implemented this procedure at Volkswagen AG Group IT cloud service instantiations, whose challenges can be considered representative for process-driven industrial organizations of similar size. The procedure as implemented also brings other non-development but equally important units of the organization together, to be on the same level, style and speed of operation as the product development unit and we can thereby conclude that:

i. A sequence of actions is needed for the maturity of a container.
ii. The sequence should be clearly defined by propagation criteria.
iii. Propagation leads to handover in the process which demands for traceability
iv. Traceability need to be established by tooling for CI/CD
v. Traceability has to be part of the DevOps culture for effective product delivery.

This paper also shows our effort to ensure quality even with a high development velocity and our experiences about agile/lean product development in large and sometimes bureaucratic corporations. From a software process improvement view, the process and method authority is not sufficient to deliver added value. So, in the future, the involved technological principles, which are the drivers for operative excellence in a fast-changing and scaling environment, should be better understood.

5 SPI Manifesto Reflection

The presented methodical procedure for quality assurance for containerization process also follows the values and principles that are described in the Software Process Improvement (SPI) Manifesto [18][19][20]. Especially the following aspects are strongly related to SPI:

i. The change comes from the technology push of automation options.
ii. Drives the business to pull to be faster on the market (Conway's law: organization follows the architecture of CI/CD technology).

iii. The people get benefits of this technology-driven SPI to deliver an approach to be more efficient with containers and deterministic procedures based on traceability in the new DevOps mindset, which is a new culture of product delivery in an agile continuous improvement version.

The team has worked with people and units involved, for example, the open source compliance, infrastructure and security teams. The outcome of this work is available like a template for projects which want to set up a container-based delivery chain. As good SPI practice, the suggestion should be customized to the needs of the projects and feedbacks will enhance the "template" in the future.

References

[1] Martin, A., Raponi, S., Combe, T. and Di Pietro, R., 2018. Docker ecosystem–vulnerability analysis. Computer Communications, 122, pp.30-43.[2] https://www.digitalocean.com/community/tutorials/the-docker-ecosystem-an-overview-of-containerization

[3] Buckley, F.J. and Poston, R., 1984. Software quality assurance. IEEE Transactions on Software Engineering, (1), pp.36-41.

[4] Docker Image Specification: https://github.com/moby/moby/blob/master/image/spec/v1.md

[5] ISO 12207 (ISO/IEC12207.0-96)

[6] Yasrab, R., 2018. Mitigating docker security issues. arXiv preprint arXiv:1804.05039.

[7] https://www.heise.de/security/meldung/Sicherheitsforscher-brechen-aus-Docker-Container-aus-4276108.html

[8] Pinkpreet, K., Anuoama, G., & Er. Harpreet, K. A novel system defined network solution for user legitimacy Assurance in Docker Containers. December 2018

[9] Cleland-Huang, J., Gotel, O.C., Huffman Hayes, J., Mäder, P. and Zisman, A., 2014, May. Software traceability: trends and future directions. In Proceedings of the on Future of Software Engineering (pp. 55-69). ACM.

[10] Park, Y. and Lee, C.P., 2008. The Impact of RFID-based Traceability System on Perceived Competitive Advantage in the Food Industry. Oklahoma State University,[White Paper], pp.4301-4306.

[11] Spanoudakis, G. and Zisman, A., 2005. Software traceability: a roadmap. In Handbook Of Software Engineering And Knowledge Engineering: Vol 3: Recent Advances (pp. 395-428).

[12] Sundaram, S.K., Hayes, J.H., Dekhtyar, A. and Holbrook, E.A., 2010. Assessing traceability of software engineering artifacts. Requirements engineering, 15(3), pp.313-335.

[13] Mirakhorli, M. and Cleland-Huang, J., 2011, May. Tracing architectural concerns in high assurance systems (NIER track). In Proceedings of the 33rd International Conference on Software Engineering (pp. 908-911). ACM.

[14] https://www.ca.com/en/blog-developers/docker-containers-os-base-image.html

[15] https://en.wikipedia.org/wiki/Lint_(software)

[16] https://opensource.org/licenses

[17] Gotel, O., Cleland-Huang, J., Hayes, J.H., Zisman, A., Egyed, A., Grünbacher, P. and Antoniol, G., 2012, September. The quest for ubiquity: A roadmap for software and systems traceability research. In 2012 20th IEEE international requirements engineering conference (RE) (pp. 71-80). IEEE..

[18] Korsaa, M., Biro, M., Messnarz, R., Johansen, J., Vohwinkel, D., Nevalainen, R., & Schweigert, T. (2012). The SPI Manifesto and the ECQA SPI manager certification scheme. Journal of Software: Evolution and Process, 24(5), 525-540.

[19] Messnarz, R., Sicilia, M. A., Biro, M., Garcia Barriocanal, E., G. Rubio, M., Siakas, K., & Clarke, A. (2014). Social responsibility aspects supporting the success of SPI. Journal of Software: Evolution and Process, 26(3), 284-294.
[20] Sanchez-Gordon, M. L., Colomo-Palacios, R., & Amescua, A. (2013). Towards measuring the impact of the spi manifesto: a systematic review. In Proceedings of European System and Software Process Improvement and Innovation Conference (pp. 100-110).
[21]. https://github.com/GoogleContainerTools/distroless/tree/master/base
[22 Kozhirbayev, Z. and Sinnott, R.O., 2017. A performance comparison of container-based technologies for the cloud. Future Generation Computer Systems, 68, pp.175-182..