How to Deliver Faster with CI/CD Integrated Testing Services?

Alexander Poth(✉), Mark Werner(✉), and Xinyan Lei(✉)

Volkswagen AG, Berliner Ring 2, 38436 Wolfsburg, Germany
{alexander.poth, mark.werner, xinyan.lei}@volkswagen.de

Abstract. After Volkswagen AG has setup its hybrid cloud products and services are expected to have a significant shorter time-to-market and a high quality level. To satisfy these expectations, the existing testing procedures should have to speed up, so that it won'tbe the bottle neck for new features that have to be shipped fast. In order to realize a faster delivery of products and services Testing as a Service is developed with lean/agile methods [1, 2] and integrated into the CI/CD pipeline of the Volkwagen Group IT Cloud.

Keywords: Agile software development · Quality assurance (QA) Cloud · Testing · CI/CD

1 Motivation and Context for the Demand of Testing as a Service

To meet the needs for testing in a cloud native fashion, Testing as a Service (TaaS) is setup as one of the Volkswagen Group IT cloud (GITC) services. The focus is to offer a testing service that is able to scale natively based on cloud technologies. The service focuses on offering state of the art testing capability for both functional and non-functional testing in the cloud environment. The motivation for developing TaaS was a missing link in existing cloud native tools, which could also be run on premise in a private or hybrid cloud. For a public cloud approach there exists different tools like BrowserStack, BlazeMeter, etc. But these tools did not offer any possibility to run on premise in a private cloud and therefore could not be used for testing sensitive data/applications. As a matter of fact filling a gap like this leads mostly to create a tool or to change workflows in IT context. As a result of that TaaS has to make a choice of creating tools or workflows to achieve the goal. Furthermore nowadays companies have a cloud-first/cloud-only strategy for development of applications. The new approach leads to a change in thinking and designing new IT applications. The existing well-proofed development concepts must be reconsidered if we want to make the best out of the new found possibilities of developing cloud native applications. Based on the experiences that were gathered during the development of TaaS, it was recognized that both an API for machine usage and a self-service portal (SSP) for human interaction are a must for TaaS to be implemented. If we want to make the most out of TaaS for our company, we have to take the integration concept of the new service into existing processes into consideration. Most of the benefits from TaaS could be achieved only if

X. Larrucea et al. (Eds.): EuroSPI 2018, CCIS 896, pp. 401–409, 2018.
https://doi.org/10.1007/978-3-319-97925-0_33

the TaaS can be integrated smoothly in the existing processes in the organization. In the following chapters it will be described, how the TaaS team improved itself during the development and achieved success step by step by motivating others and keeping the focus on the needs of customers.

2 Rapid Development of a Product or a Service

This section describes the implementation of TaaS. To implement TaaS quickly and to make it customer (user) centric, the GITC implements an intrapreneurship [3] friendly environment. This environment focuses on selling a "product vision" for the service and applying for an initial financial support. This basic financial setup is used to implement a proof of concept (PoC). The 1st venture capital was given in the fourth quarter of 2016 to develop a PoC for the GITC exhibition in December. After delivering a successful PoC, the GITC council offers further venture capital for product increments. The product owner has to apply for the next increment of the service. Each increment is called a minimal viable product (MVP). This approach is used to analyze the customer's feedback and acceptance of the growing service. Furthermore, the financial support can be stopped quickly in case of not fulfilling the expectations of performance or business perspectives. Depending on the speed of revenue generation by the service, the amount of financial support can be adjusted easily to service "incomes" and support other services with the money "saved".

This intrapreneurship-based environment encourages the TaaS team to develop more than MVPs. Because MVPs are not "attractive" enough for the customer and have sometimes many constraints. The TaaS team decides to develop SLCs [4]. SLC is the abbreviation for simple, lovable and complete. This gives the option to run TaaS every time as a stable product in case the venture capital flow stops and the income is not enough to cover finance complex stabilization tasks and enhancements to make the service "user-acceptable".

Based on the described environment, the TaaS DevOps team focuses on a strategic view to optimize TaaS for:

- Making testing faster
- Making it more comfortable and easier to use
- Reducing service resources and costs

This motivates the TaaS team to work more in a feedback driven fashion rather than specification driven. Also it forces the TaaS team to fix bugs as fast as possible and have a low technical debt in TaaS for fast and nimble reactions to feedbacks.

The formal 4 MVPs in 2017 focused on shipping non-functional load testing, performance testing, functional testing for responsive web design (RWD) and multi-browser testing on a user perspective. On a cloud perspective the rollout to GITC was focused. The MVP roadmap in 2018 focuses on more rollout and enhancement of existing functions based on user feedback. To get user feedback for every new key feature, at least one pilot customer has to act as a development partner of TaaS. Furthermore, after a new feature was released into production, the customers would be

asked for feedback about these particular features..These feedback sessions were based on the starfish retro approach [5].

The TaaS team started with Scrum, but realized later on, that there were dependencies (for example database), that often blocked the implementation of TaaS. This observation led to a change in Kanban. In 2018 the TaaS team is practicing a Scrumban approach. The Kanban component is used for operational issues of the DevOps teams.

This agile and lean development method can also have negative impacts to the DevOps TaaS team. TaaS started in 2018 with no budget. These days were difficult for the team because it was unclear how long it needs to convince the GITC council for the next MVP/SLC and how much venture capital will be offered for the rollout to AWS and feature development for the upcoming MVP/SLC.

A strategic decision was to integrate TaaS into the Volkswagen Group IT devstack of the Volkswagen AG. Devstack is a project-customizable CI/CD (continuous integration/continuous deployment) toolchain for Volkswagen AG. Devstack is not only available for GITC users but offers cross-selling options to a much bigger market for TaaS. This integration idea TaaS@devstack was born in 2017 but will need some time for management commitment. Furthermore, the TaaS DevOps team can reduce ops activities by the delegation of support tasks to the 24 * 7 devstack operation team in the third quarter of 2018. This will give the TaaS team more flexibility to focus on user demands and a more stable "financial basis".

3 Deep Dive into TaaS

TaaS is designed to support all kinds of tests from the commonly known test automation pyramid [6]. In a classic test pyramid approach there are the following kinds of tests: unit tests; integration tests; and acceptance tests. The focus is a little bit different for TaaS because TaaS uses a cloud native approach and therefore from TaaS point of view the pyramid would be like this:

- unit tests for micro-services on the base layer
- integration tests in the mid-section
- system or business related tests on top

So at the end, several abstraction levels are covered. TaaS is designed to write tests once and run often – in best case integrated into a CI/CD toolchain. This enables the user to build a broad base of automated tests, which are executed on each triggered build. This can ensure a high level of quality of the product built, because the execution of tests are automated and will become native to the development team. Providing automated tested applications will also reduce the time-to-market because it ensures that the developed features of the application will be tested on each build. Using iterative sprints and a set of regressions tests to build up an application ensures correct functionalities of the developed application.

Developers focus on developing new features and using an agile approach but the tests have to be automated and integrated into the build/deployment process. This approach ensures that the tests are still in the focus of developers and that they are responsible to deliver new features. A benefit of this setup is that all shipped/delivered

applications are also tested automatically using regression tests and therefore ensure an objective quality of the delivered application.

By using a Definition of Done (DoD) concept, the development team ensures that every micro-service is tested automatically. This is a strong basis to build more complex integration tests to cover the interaction between micro-services. It would be possible to verify a whole system of business workflows through these kind of tests.

Because TaaS supports different testing tools, a wide range of testing is possible, from API (application programming interface) tests for example REST-based to GUI (graphical user interface) tests using different browsers. Depending on the testing tools, TaaS supports not only functional testing but also non-functional testing.

Depending on the processed data (sensitive or private) it is not an option for testing these applications in a public cloud. Using standard already existing testing tools will miss the capabilities/benefits, that cloud native approach offers. TaaS is designed to be cloud native and therefore benefit from the different advantages that a cloud native application will offer. TaaS makes use of well-known open source software such as Selenium [7], JMeter [8], Docker [9], Kubernetes [10] etc.

TaaS does not use any proprietary tools and interfaces and can be easily integrated into different CI/CD-toolchains without a complex setup. This open design prevents also any tool constraints, which could lead to a tool lock-in later on.

TaaS does not limit the development team to follow either a top-down approach or a bottom-up approach because it supports both ways of testing an application. Following the top-down approach, the tests have to follow defined business processes and cover the acceptance criteria of customer journeys. Using this approach, we are starting from the top of the test pyramid to cover relevant customer journey and features.

In a bottom-up approach, tests are designed and developed for micro-services, which will be deployed. This ensures that all the developed micro-services are covered with automated tests. These could be integrated with integration tests covering basic steps.

3.1 The API and the Self-service-Portal

As already noted before, TaaS is designed to write a test once and run it often (see Fig. 1). In the first step, the tests which should be executed later are designed and written in the IDE (integrated developer environment) of the tester. This also helps to reduce the time-to-market because the development team can use all the well-known tools to write necessary tests.

The registration in TaaS and the execution of tests via TaaS is implemented using a two-step process. Senior user/administrator double checks and therefore limits the valid URLs (unified resource locator) or IP (internet protocol)-addresses/endpoints which could be used for the tests to prevent DoS (denial of service) attacks using the provided cloud resources. The endpoints for the test object are fixed and cannot be changed by the provided test cases. After a valid configuration of the tests, that includes which test cases to run and how often, the configuration of the test is completed.

Depending on the project requirements the tests can be triggered manually using the self-service-portal or can be integrated into an already existing CI/CD-tool chain.

After executing the tests and finishing the test runs, the logs of the test execution are collected and stored for analyzation. Depending on the test results, the development team has to decide how to process it further:

- fixing the bug, that was found during the execution
- adjusting the tests according to the acceptance criteria of the feature

In both cases, the quality of the developed application is raised and enables a faster time-to-market. In the first case, a fixed bug means one error less in the developed application. The second case raises the quality because a test, which tests the wrong use case/feature does not affect the overall quality of the application. Tests have to be handled like code, which means that test cases have to be refactored from time to time.

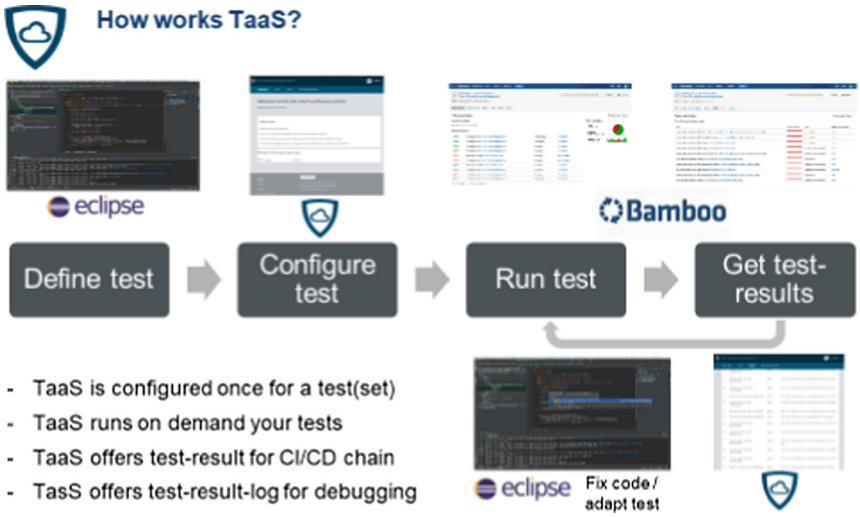

Fig. 1. An overview of the workflows with TaaS

3.2 Load and Performance Testing

The kind of tests which cover load and performance testing is also possible to be executed using TaaS. Performance testing is used to determine the performance of a test objective in terms of responsiveness under a defined work load. In general the minimum, maximum and average response times are measured. A load test is a specific form of performance testing because it tests the system behaviour under a defined work load. The work load could be a number of users/sessions performing specific number of actions within a defined duration. These tests are used to identify bottlenecks in the application. To support these kinds of testing, the open source tool JMeter is supported by TaaS and can be found in the self-service portal.

To cover several scenarios different versions of JMeter are offered from which the customer could use the desired version. JMeter was designed to simulate test behaviour and measure performance. By design, JMeter is capable to test the performance on

static and also dynamic web applications. It provides a simulator for heavy load to check the overall performance of the test objective under different load scenarios. The most commonly used protocols are web protocols (http, https) and REST based web services.

JMeter's built-in multi-threading framework offers the possibility to run concurrent tests by many threads and could scale natively on a cloud platform. The customer is able to select the desired amount of JMeter workers using the self-service portal. To offer the customer horizontal scaling options, a cluster is set up with one master and depending on the customer needs several slaves. The master controls the slaves and manages the distribution of tests to be executed. The slaves for the heavy work are testing the defined test objects with the desired load as defined in the test configuration.

3.3 Functional Testing

To offer responsive web design and multi-browser testing, dynamic grids with required settings are dynamically allocated. To quickly offer the required and specific settings, different Docker containers with the demanded browsers, like Firefox [11] or Chrome [12] and desired versions are setup in the grid. Selenium is used as the test engine.

To support this kind of testing TaaS supports the usage of Selenium to enable the testing of different browsers and versions. Selenium is an automated testing framework for web applications and provides features like playback of formerly recorded business processes. Using this approach no specific knowledge is needed to be able to build test steps.

For each test run, a preconfigured Docker image is used in which the target browser, version and the tests to be executed are prepared. So the customer can decide which web browser and which version he needs to use for testing his business processes. In the self-service portal, the customer is able to configure relevant meta data for his test execution including the size. Using the provided WebDrivers from Selenium, most web browsers are supported and can be controlled by several programming languages (Java, C#, JS, …). This enables the development team to write tests in the preferred programming language, which reduces the time for writing tests and setting up testing tools etc. An agile development team could focus on delivering new features, which are covered by tests written in the preferred programming language.

For enabling the testing of web browsers running on remote machines Selenium offers the Selenium Grid. Since the Selenium Grid has a scalable design, it allows executing tests in parallel on different machines. This approach matches perfectly with the cloud-native approach of TaaS. In the end, a highly scalable and flexible infrastructure could be offered to the customer to support the testing of different web browsers. Without the need to setup an instance of Selenium or Selenium Grid, TaaS enables the customer to execute tests for different web browsers at a fingertip. After the test execution, TaaS enables downloading the test results for analyzation of the test runs.

TaaS supports the whole testing life cycle of software testing starting from providing the necessary testing infrastructure (JMeter, Selenium), offering a highly scalable and flexible architecture to scaling horizontally.

To offer maximum flexibility to the customers of TaaS the application specific setups and configurations are shipped with the customers Docker container.

To minimize the overhead of managing the different configurations of testing setup, the TaaS design uses Docker containers. These containers are prepared in advance with the different versions of the offered testing tools. The customized test configuration including the tests to execute are mounted onto the prepared Docker containers, which are started according to the customer setup.

Depending on the setup, several nodes are spun up in the existing Kubernetes cluster which starts the desired number of pods to support the customer specific test configuration. This lean architecture allows a fast and flexible way to manage several different customer setups.

Important to notice is that each customer uses his/her own network, which is generated on demand for each test run. This limits the potential (negative) side effects of running instances in parallel or shared resources etc.

From the customer point of view, the whole setup of testing tool is already done and he can focus on the design and preparation of his specific tests. He does not need to know how to setup and maintain an architecture on a cloud but can benefit from the offered TaaS solution.

He is an expert in his domain of the test objective and can now use the saved time to invest in more and better tests. Using the possibility to integrate TaaS in an already existing CI/CD toolchain could improve the overall quality of the developed application, because all defined tests run automatically after each application deployment. This helps the customer of TaaS to get a fast feedback of the quality of the build and the delivered application.

Using TaaS enables the customer to control the development process very tightly to ensure that the development team focuses on delivering new and tested business values. The development team is still aware of the new developed features and is therefore very quick in fixing defects that are found and also in adjusting new business features. In the past there was a large time gap between developing new business features and getting feedback of the testing on the productive environment. The development team may have been changed in the meantime or they are working on different projects. In either way, it costs time to recall the past activities and to focus again on the new task.

This container contains technology specific things for example for an Angular application the protractor [13] test-stack and the specific tests for execution. The figure below (Fig. 2) shows a setup with cucumber [14].

3.4 Outlook

The next functionality on the roadmap of TaaS is mobile testing integration. The SLC approach leads the TaaS team to focus on 80% of the solution with Appium [15].

The goal is to test an Android app to ensure a good scalable approach. This will enhance the feature set provided by TaaS to support the wide range of mobile development and therefore testing.

Consumer-driven contract testing is another candidate on the roadmap of TaaS. With this feature TaaS would cover all the levels of the testing pyramid that was

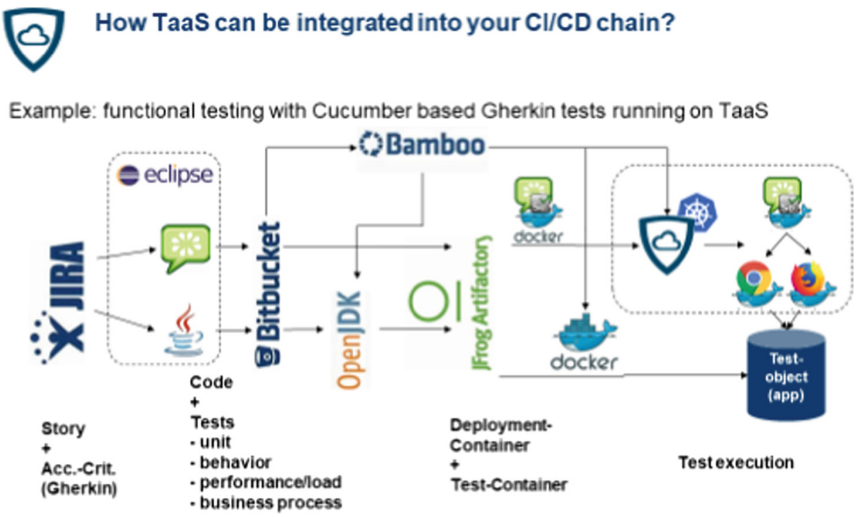

Fig. 2. An example setup of CI/CD integrated functional test

introduced in Sect. 3. The ultimate goal of TaaS is to be an open testing platform with extensible capabilities.

4 Conclusion

In this paper, we have presented the history and details about TaaS in Volkswagen AG. The product enables developers to enjoy the benefits of rapid development velocity in the cloud native age. TaaS supports different kinds of tests. With the help of TaaS, tests can be easily integrated into a CI/CD toolchain and can be implemented with different frameworks. TaaS hides the complexity of setting up and managing the test-environment away from software developers, so that software developers can focus more on developing business values. In order to ensure all the benefits and the ongoing agility, the TaaS team developed the service in a more feedback driven way rather than a specification driven way. The TaaS team has worked closely with pilot users in order to keep the feedback loop shorter for all the core functionalities. It also helps to focus on implementing the SLCs.

This paper 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. It also shows a possible way to improve the innovative nature in a large corporation with the help of an intrapreneurship friendly environment.

From a software process improvement view, the summary should be that in the future the process and method authority is not enough to deliver added value without understanding the involved technological principals, which are the drivers for an operative excellence in a fast changing and scaling environment.

5 SPI Manifesto Reflection

The presented TaaS approach also follows the values and principles that are described in the SPI Manifesto [16, 17, 18]. The TaaS team has tried to motivate all the people involved, for example the venture capital and the customers. As a part of the motivation, the TaaS team has improved the development process based on the experiences gained with customers from MVP to SLC. It is a dynamic and adaptable way to satisfy customer needs with an agile/lean mindset. The SPI Manifesto's values and principles are a guide for the TaaS team to make the best choice for the team and its customers.

References

1. Sutherland, et al.: http://www.agilemanifesto.org
2. Sutherland, et al.: http://agilemanifesto.org/principles.html
3. Pinchot, G.: Intrapreneuring: Why You Don't Have to Leave the Corporation to Become an Entrepreneur. Berrett-Koehler Publishers, 2. Auflage (1985). ISBN 1-57675-082-5
4. https://blog.asmartbear.com/slc.html
5. Schwaber, K.: Agile Project Management with Scrum. Microsoft Press, Redmond (2004)
6. Cohn, M.: Succeeding with Agile: Software Development Using Scrum. Pearson Education (2010)
7. https://www.seleniumhq.org/
8. https://jmeter.apache.org/
9. https://www.docker.com/
10. https://kubernetes.io/
11. https://www.mozilla.org/de/firefox/
12. https://www.google.com/chrome/
13. https://www.protractortest.org/
14. https://cucumber.io/
15. http://appium.io/
16. Korsaa, M., et al.: The SPI Manifesto and the ECQA SPI manager certification scheme. J. Softw.: Evol. Process **24**(5), 525–540 (2012)
17. Messnarz, R., et al.: Social responsibility aspects supporting the success of SPI. J. Softw.: Evol. Process **26**(3), 284–294 (2014)
18. Sanchez-Gordon, M.L., Colomo-Palacios, R., Amescua, A.: 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 (2013)