Towards DevOps in Multi-provider Projects

Masud Fazal-Baqaie[1], Baris Güldali[2], Simon Oberthür[3]

Abstract: DevOps addresses the continuity of development and operations activities in the software development lifecycle in order to achieve a better software experience via shorter development and release cycles with improved quality. A challenge in enterprise context is to achieve DevOps in multi-provider projects by synchronizing and coordinating various teams. We report on our experience in implementing DevOps principles in such a multi-provider environment and present good practices as well as open challenges.

Keywords: Multi-provider projects, DevOps, Continuous Software Engineering, Continuous Quality Management

1 Motivation

Big, enterprise-scale software systems are nowadays typically developed in a multi-provider environment with several onshore and offshore vendors. Adopting agile practices allows for frequent deliveries by the vendors, thus enabling frequent integration and early feedback by customers and reducing the project risks. In order to handle the customer requirements and the production incidents as soon as possible, companies are thriving to implement DevOps principles [Da16].

DevOps is based on and extends agile principles by fostering communication and collaboration. It advocates to overcome an "us-and-them" mentality, especially between people involved in software development (Dev) and people involved in software operations (Ops). Ideally, software teams have end-to-end responsibility for a software artefact throughout its whole lifecycle: from the product planning and the implementation, over its delivery and rollout to its operation. This is especially challenging in multi-provider environments that induce an additional layer of complexity for delivery management and quality management.

In this paper, we report on our experience based on a project from the financial domain. The client company is introducing a new client-centered, digital sales channel. Technically, this requires to open and extend existing backend systems for access via web-based interfaces for business partners and customers. Business processes need to be redeveloped and aligned, involving multiple business departments at once. The project consists of multiple sub-projects, where teams from the client company and teams from

[1] S&N CQM GmbH, Klingenderstraße 5, 33100 Paderborn, masud.fazal-baqaie@sn-cqm.de
[2] S&N CQM GmbH, Klingenderstraße 5, 33100 Paderborn, baris.gueldali@sn-cqm.de
[3] Mobile & Cloud Systems, Software Innovation Campus Paderborn, Zukunftsmeile 1, 33102 Paderborn, oberthuer@sicp.de

several providers work together, involving also an offshore provider. The project was set up to follow an agile methodology. After realizing several improvements by better aligning with agile practices [FR15], it now adopts more and more DevOps principles for further improvements.

2 Experiences in Dev and Ops in Multi-provider Projects

Applying DevOps in a multi-provider project poses additional challenges. The first factor is heterogeneity. On the one hand, multi-provider projects can have cultural heterogeneity on different levels: Based on responsibilities, e.g., development vs. operation, based on the region people are working in, e.g., when offshoring, or based on differing company philosophies, e.g., regarding formality and hierarchies. On the other hand, multi-provider projects can have legal differences based on regulations and contracts, e.g., who is allowed to work on what and how. The second factors is scale: multi-provider projects are facing additional complexity with respect to the coordination of all the activities due to the size of the project that is typically big.

In the following, we want to describe challenges and good practices based on our experience. We do this based on the software lifecycle from planning to operations. Adopting a DevOps software lifecycle requires continuous quality management. We describe also the integration of quality assurance activities throughout the lifecycle.

In **Product Planning,** the project is facing mainly two challenges. First, overall business processes need to be realized by the interplay of various IT systems, thus they need to be cut down to system-specific requirements and distributed among the teams. Here, the heterogeneity described in the introduction needs to be accounted for, e.g., differing delivery dates. Second, new requirements have to be prioritized together with defects and improvement changes reported from production. In our project, we have created a macro plan (Project backlog) specifying the features of components (Product backlogs) and delivery schedules. Visual workflow modelling using BPMN helps for better understanding of the workflows, the component interfaces and the SLAs. Dev teams of providers derive their requirements (Sprint Backlogs) based on the Product backlog and the SLAs. All backlogs and production incidents are transparent to the project management all the time. Continuous reporting enables monitoring of risks and synchronizing the sprint planning of provider teams (see also using the agile release train pattern [Le11]). As quality guards, we use agile metrics indicating the execution status of test cases and criticalities of defects. The defects are classified and prioritized in direct communication between business departments, project managements and vendor teams [FGS15].

For the **Implementation**, beside architectural topics, a main concern is to achieve transparency about the quality delivered by various teams. This varied heavily among the different teams and was an obstacle for judging about the overall quality of the product. We have developed a maturity model to get transparency about the quality levels of subprojects and to motivate them to reach the next maturity level. The maturity

levels define minimal requirements on quality aspects (e.g. unit testing, UI testing) and standards (e.g. minimum test coverage, code quality measures) [FGG16]. Defining standardized Git (source code management) workflows and continuous integration pipelines helps to onboard new teams quickly. Teams use common artifact repositories and common Docker base images, which enable quick reaction to security vulnerabilities and to performance issues. We defined a microservice-based architecture [Ne15], which improves the flexibility of applications and reduces the dependencies between functional components.

The **Delivery Management** coordinates the deliveries of various teams on the basis of macro planning and product backlogs. After we have experienced some heavy delays of critical components, which resulted in delay of the whole project, we have worked on backward-compatibility of components. Teams have to implement database changes and interface changes in a backward compatible way. Thus, we can deploy components of various vendors as soon as they are delivered. Automated test scripts help to quickly validate that no regressions are injected between component interfaces due to unsynchronized deliverables.

Rollout Management: Deployments of many components by different vendors can be complex because of dependencies and thus need to be carefully planned. Typically such deployments contain applications, microsservices and databases. If offshore providers are involved, time zone differences and holidays may be a real problem for rollout management. Also the conditions of the hosting providers must be considered in rollout planning. The more components and teams a deployment involve, the longer is the installation time and the higher is the risks that something goes wrong. We made use of container technologies, e.g. Docker, for efficiently preparing installation packages and push them to various environments in very short times. Automated sanity tests validate, whether the productive system behaves as expected in production environment. Both the container technology and the backward compatible deliveries help us to reduce the risks of rollbacks in case of production problems.

Operations: Using an agile delivery model allows to continuously improve the product based on the feedback from the operations. Especially for enterprise-scale software in a multi-provider environment, it is important to maintain an overview and to associate incidents with responsible components/providers. In order to ensure the reliability and high availability of our systems, we have implemented clustering and failover mechanisms. Monitoring techniques detect performance issues and if components are not available they restart them. Meanwhile emergency teams are informed to resolve incidents, when automated start/stop scripts do not manage to resolve the production problems. We have used central logging for collecting runtime data from all components and created a log dashboard for various teams. We had to fulfill some special legal requirements for logging in order to supply offshore teams with logs for debugging.

3 Outlook

While we are making progress in our project with adopting DevOps principles, we are at no means at the end. One interesting aspect we are seeing is that in the past, effort was put in test mainly to increase the quality. Today with DevOps, automated software test is an enabler for shortening the release cycles while keeping or even increasing the quality. Thus, quality assurance can lead not only to increased quality but also to reduction of cost. Our vision is to push our concept of quality guards further, such that they become a self-contained part of the software lifecycle. Quality guards at different stages of the complete development and operation pipeline (from unit tests, over integration tests, to monitoring in production) assure that the requirements are fulfilled and, if they are not met, appropriate measures can be applied. The tool and framework landscape, which support such processes is today manifold, but needs proper selection and integration. A continuous quality management is therefore required. In this management, the quality guards have to be defined. Also important is to implement proper reactions for when a guard is not met. If, for example, an integration test fails, the team which added a new version of a component must be informed. If a guard fails in production, e.g. the monitoring detects the failure of components, unavailability of a service or unusual memory usage, resilient mechanisms must take place. Embracing the failure, e.g., like Netflix [Tse13] is doing by injecting failure to the productive system (Simian Army) is a good way to force every involved party to build systems to be able to heal itself. Independent delivery of chunks of functionality/parts of an application can help to keep speed for new/changing functionality. Building on an architecture like microservices [Ne15] and having end-to-end ownership (from dev over deploy to run) allows teams to deploy on their own speed independently.

Literaturverzeichnis

[Da16] Daly, D. et.al.: Enterprise DevOps – Building a Service Oriented Organization. Atos, 2016. http://ascent.atos.net/?wpdmdl=12439

[FGG16] Fazal-Baqaie, M.; Güldali, B.; Grieger, M.: Ganzheitliches Qualitätsmanagement in agilen Groß- Projekten. In (Engstler, M. et.al. Hrsg.): Proc of Projektmanagement und Vorgehensmodelle 2016. Köllen Druck+Verlag GmbH, Bonn, 2016, S. 109-120

[FGS15] Fazal-Baqaie, M.; Grieger, M.; Sauer, S.: Tickets without Fine - Artifact-based Synchronization of Globally Distributed Software Development in Practice. In (Abrahamsson, P.; Corral, L.; Oivo, M.; Russo, B. Hrsg.): 16th Int. Conf. of Product Focused Software Development and Process Improvement. Springer, LNCS, vol. 9459, 2015, S. 167-181

[FR15] Fazal-Baqaie, M.; Raninen, A.: Successfully Initiating a Global Software Project. In: Industrial Proc of the 22nd European Systems Software & Service Process Improvement & Innovation Conference (EuroSPI²2015). WHITEBOX, Denmark, 2015.

[Le11] Leffingwell, Dean: Agile Software Requirements. Addison-Wesley, 2011.

[Ne15] Newman, S.: Building Microservices: Designing Fine-Grained Systems. O'Reilly, 2015.

[Ts13] Tseitlin, A.: The antifragile organization. In: Commun. ACM 56, 8 (August 2013), S. 40-44. DOI=http://dx.doi.org/10.1145/2492007.2492022