Literature Review

Michael Lundquist

June 12, 2019

Effective collaboration and teamwork separates good software development teams from bad ones. In the past five years, new and exciting collaboration softwares have been released. This paper will document my experience as a junior developer introducing collaboration software to a team that was relatively unfamiliar with these new softwares. This paper could be an invaluable resource for other teams looking to modernize how they collaborate.

1 Current Situation

Relatively unfamiliar is a good term for the experience level of my team and many others like it. The new technologies enable collaboration in new ways. Before these new collaboration tools came out, teams would segregate operation and development of software. My team of developers was no exception we would write applications with little regard for how they would be deployed. I aim to fix this using this new class of software called DevOps software. The name DevOps comes from the softwares' ability to integrate development and operations. With the new DevOps mindset, our team would be simultaneously simpler, better and more marketable.

1.1 Development Only

Without DevOps software, developers concern themselves only with development. Issues faced by developers are fundamentally different than issues faced by operations staff. Developers care that their code works on their machine, not in the production environment. The phrase "it works on my machine" is said so often by developers that it has even become a meme used to mock careless developers. Our team was no different. Although operations staff see developers as careless, developers are very careful about the issues they face on a regular basis.

Although operations staff might not agree, getting software working on one machine is hard! In an amazing rant, Lion McLionhead explains the factors that induce stress, and inevitably insanity in programmers. Among the factors: politics, security, confusion, fear of things breaking (which they always do), overwhelming responsibility, deadlines, thinking in strict languages designed for computers not humans, and broken dreams (McLionhead, 2014). After hearing a list like that, "it works on my machine" incorrectly seems like a reasonable statement. Fortunately, developer's have tools to mitigate some of these problems, and DevOps introduces more!

Writing functional software is very hard when working with developers of diverse backgrounds who must be kept in sync while working on a project. For this reason, perhaps the most challenging

problem developers face is differing version of software. Developers use version control software to ensure there arent conflicting versions. For example, using a simple version control software like Google docs, a novelist could freely change their rough draft and know they could easily revert these changes without keeping an explicit file named "rough draft." My team, uses GitLab, for version control. GitLab's underlying system level technology, git, is very powerful because it allows multiple developers to keep software version controlled. GitLab provides a very useful interface to git, and recently GitLab has started integrating DevOps capabilities (more on this later).

To make matters worse, developers often neglect their code's documentation. Although documentation neglect incurs huge amounts of technical debt, it speeds up short term development on tight deadlines. Ideally, developers would write a full reference and a getting started tutorial for their code (Dagenais & Robillard, 2010). Without documentation, not only will operations staff struggle to implement the code in production, other developers will struggle to understand a project.

Along with the "it works on my machine" mind-set comes the monolith design pattern. Monolithic projects are designed to run entirely on one machine at a time. This design pattern is simpler for developers because components aren't networked. However, this simplicity incurs technical debt because monoliths have trouble scaling. For example, if an organization has a monolithic website and wants to build a mobile app, they would have to build an entirely new monolithic mobile app without reusing any components from their website.

1.2 Operations Only

Although operations staff are generally very responsive, their job is fundamentally not development. This segregation means operational staff may not know how changes to their servers they manage will affect developers.

Simply keeping servers running is a huge challenge. Without DevOps, operations staff generally create a virtual machine for each component that needs to run. Virtual machines have worked wonders for a long time, but they have some major disadvantages. For the following reasons, virtual machines are tied to the machine they're running on. First, virtual machines require memory and CPU sizes to be specified. Specifying these parameters in a virtual machine inherently makes it less portable. Second, moving a virtual machine requires taking a snapshot of the entire machine and loading it somewhere else. These snapshots are very large and working with them can be cumbersome. Finally, virtual machine managers are generally not open source or free, and will

try to prevent you from switching to a different manager. When operators use virtual machines then complain about developers' software not working in production, it's rather hypocritical. Fortunately, a DevOps concept called containerization enables operations staff to run multiple machines without tying the images to their environment.

If keeping servers up to date and working weren't enough, operations staff have to deal with developers. Although the classic call in for support model generally works, it's very frustrating. It requires developers design their programs to work on a server they don't manage, so naturally they don't know all the ins and outs of working with the server. This customer support model is often known as fire fighting because the problems operations staff face are endless and horrible.

2 DevOps in an Ideal World

In an ideal world, DevOps can solve many of the problems discussed above. Simply put DevOps is a collection of tools and protocols that remove the strict segregation between development and operations.

2.1 What is DevOps? UDOM

In "Toward Unified DevOps Model" by Wahabala et. al., the fundamental components of DevOps are discussed.

(Wahaballa, Wahballa, Abdellatief, Xiong, & Qin, 2015)

2.2 Docker

("Docker Documentation", 2019)

2.3 GitLab

(*Introduction to CI/CD with GitLab*, n.d.)

integration Testing (CI/CD) kubernetes obviously git

2.4 Documentation

(Dagenais & Robillard, 2010)

2.5 Microservices

(Waseem & Liang, 2017)

3 Reality

my personal experience

portainer

installing GitLab

politics

References

Dagenais, B., & Robillard, M. P. (2010). Creating and evolving developer documentation: Understanding the decisions of open source contributors. In *Proceedings of the eighteenth acm sigsoft international symposium on foundations of software engineering* (pp. 127–136). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1882291.1882312` doi: 10.1145/1882291.1882312

The authors of this paper are extremely prolific (Dagenais with 16 other papers on ACM and Robillard with 78). This resource informs the discussion on the importance of documentation and how to write good documentation.

Docker documentation [Computer program manual]. (2019, 5). Retrieved from `https://docs.docker.com/`

Docker is the world leader in containerization software. Docker is the building block of DevOps. This resource is the authoritative reference on Docker, where as Docker In Action is written to be user-friendly.

Introduction to ci/cd with gitlab. (n.d.). GitLab. Retrieved from `https://docs.gitlab.com/ee/ci/introduction/`

GitLab is one of many DevOps-enabled software development products, but it is the best one, so I'll discuss it in the paper. Like the docker resource above, this website is mostly technical documentation so pulling from it may be hard.

Kersten, M. (2018, 3). A cambrian explosion of devops tools. *IEEE Software*, *35*(2), 14-17. doi: 10.1109/MS.2018.1661330

Dr. Kersten is the Founder and CEO of Tasktop, he's been a PhD in Computer Science nearly 20 years ago now. This resource shows how new DevOps is, which shows why documentation is used.

McLionhead, L. (2014). *Coding sucks: Why a job in programming is absolute hell.* Retrieved 2019-06-11, from `https://www.youtube.com/watch?v=MticYPfFRp8`

Nickoloff, J. (2016). *Docker in action.* Shelter Island, NY: Manning Publications.

Mr. Nickoloff has been a Software Engineer working in web-development for 20 years. This book covers Docker, a containerization tool that lies at the root of the DevOps revolution. This book is a tutorial on Docker.

Wahaballa, A., Wahballa, O., Abdellatief, M., Xiong, H., & Qin, Z. (2015, 9). Toward unified devops model. In *2015 6th ieee international conference on software engineering and service science (icsess)* (p. 211-214). doi: 10.1109/ICSESS.2015.7339039

A. Wahballa is a post doctorate fellow at the University of Electronic Science and Technology of China;s School of Information and Software Engineering. He, like O. Wahballa has a handful of publications on IEEE. This paper can help Clarify confusions companies have when working with a specific vendor's DevOps product as they'll already have a framework for understand DevOps. This paper will help build models.

Waseem, M., & Liang, P. (2017, 12). Microservices architecture in devops. In *2017 24th asia-pacific software engineering conference workshops (apsecw)* (p. 13-14). doi: 10.1109/APSECW.2017.18

P. Liang has 25 papers many related to DevOps and programmer collaboration. M.Waseem is relatively unpublished. Microservices is used in DevOps to limit the scope of a component of an IT archetecture. This enables collaboration and DevOps.