GitOps: The Evolution of DevOps?

Florian Beetz
Distributed Systems Group,
University of Bamberg
Bamberg, Germany
florian1.beetz@stud.uni-bamberg.de

Simon Harrer
innoQ Deutschland GmbH
Nürnberg, Germany
simon.harrer@innoq.com

***Abstract*—With the rise of cloud computing platforms, the speed of developing software has drastically increased. This increased speed requires new models of operating hardware, as the separation of development and operations departments often leads to bottlenecks in the deployment process. DevOps techniques aim at improving the collaboration between development and operation teams, promising to reduce the time until new products reach the customer and improving software quality. GitOps is a new concept claiming to be the evolution of DevOps. This paper analyzes if GitOps is really an evolution of DevOps, as in making DevOps obsolete. As neither DevOps nor GitOps have a standard definition, this paper synthesizes a definition of both concepts and compares them by their key elements. We find that GitOps is not the evolution of DevOps although they share common concepts. However, both concepts are defined by their use in industry, so additional research is required.**

***Index Terms*—GitOps, DevOps, Infrastructure as Code**

I. Introduction

In recent years the speed of developing software has rapidly increased. While in the 1970s and 1980s developing new features and products took several years, it is not uncommon today to deploy hundreds of times per day [1]. However, the commoditization of hardware and the rise of cloud computing products enabling this speed-up also require different operation techniques. Such a new technique promising more efficient use of cloud resources is called GitOps.

The goal of this paper is to answer the question: *Is GitOps the natural evolution of DevOps as it is claimed by its inventors [2]?* Determining the effort required to transition from DevOps to GitOps is essential for an organization's decisions to switch to new operation models.

However, DevOps, as well as GitOps, are techniques that are mainly practiced in the industry but are not well researched. In the case of DevOps, this can be explained by the fact that it has no clear definition. While there are some common themes in its implementations, it is still interpreted very differently and different organizations and researchers focus on different parts of the technique in their definitions [3]. This is very different from GitOps, which is attributed to the company Weaveworks and has been described mostly by them [4]–[6]. But although GitOps has a clearer definition, the research on this technique is only now beginning, which can be attributed to its recent emergence.

The contribution of this paper is to first synthesize definitions for DevOps and GitOps and then analyze how these definitions overlap in the described principles and practices of both techniques. Throughout the literature on DevOps, there is a common notion of principles and practices of DevOps, which were used to derive the basis for the definition of DevOps that is used in this paper [1], [3], [7]. Due to the sparse peer-reviewed literature on GitOps, its definition is mainly based on descriptions of Weaveworks [4]–[6], instead of performing a systematic literature review. For the comparison of DevOps and GitOps, again the principles and practices of DevOps form the basis and are identified in GitOps.

II. Related Work

This section gives an overview of the literature that was used to synthesize the definitions of DevOps and GitOps used in this paper.

Kim et al. [1] provide a practical guide on how problems between Development and Operations teams can be addressed using DevOps. They first discuss the history of DevOps and also the influences of other approaches on DevOps. Based on several case studies, they first show up problems that arise in traditional development and operations approaches and then introduce concrete steps of action that help to mitigate these problems. In particular, they introduce the notion of DevOps principles and practices that will be used throughout this paper.

Jabbari et al. [3], [7] performed systematic literature reviews to identify common parts of DevOps definitions, its practices, benefits, and its similarities to other techniques. They found that DevOps has not yet been extensively defined, although there are parts that are shared in most definitions. Also, the DevOps practices show common patterns in various definitions, however, they found that many do not include any practices as a part of their definition, which they explain by the novelty of the concept. From the common practices, they related the claimed benefits of the reviewed studies and constructed a dependency network of benefits and practices. They found, that most reviewed studies only make a claim about the benefits, but very few studies actually demonstrate them through empirical research. According to the authors, DevOps is immature in the peer-reviewed literature and more empirical validation of the claims is needed.

Weaveworks [4]–[6] have coined the term GitOps as their way of running operations in the company. They provide extensive descriptions of how they manage their Kubernetes clusters with the process. Since their first release about GitOps in August 2017, they continuously refine its definition and address users questions about it.

III. DevOps

The term DevOps is a portmanteau of the words Development and Operations [8]. This combination of two words is quite fitting, as DevOps expresses the idea of improving the collaboration between Development and Operations teams in corporations or even joining these two teams together. The traditional separation of Development and Operation teams results in conflicts of interest, as the Development team is encouraged to introduce changes, while the Operation team tries to avoid changes to maintain the stability of the system [9]. Such conflicts often result in increased time to market, lower quality of software and more outages, as the Operations teams is incentivized to enforce a high-ceremony deployment process, leading to fewer deployments with more changes in one deployment [1]. Fewer, but larger deployments have a higher potential for errors and leave no opportunity for early customer feedback.

DevOps tries to give a solution to this downward spiral of ever-increasing ceremony, size of deployed changes and risk by adjusting Development and Operations team to the same shared organizational goal [1], [8], [10], [11]. This mainly requires a cultural shift in organizations, focusing on transparency and communication, blamelessly acknowledging human error and collaboratively solving problems.

While DevOps is not conclusively defined, several principles and practices related to the Agile movement are commonly identified in DevOps [3], [8], [12], [13].

A. DevOps Principles

The principles of DevOps are the core guidelines to achieve faster feedback loops and reduced risk in software development. These principles are partly loaned from the Agile movement and the principles of Lean Production [8]. In some cases, these principles also overlap or are the result of another principle.

a) Iterations: Iterative software development is one of the core principles of the Agile movement and also DevOps [1], [8]. The core idea is developing software in short cycles.

b) Increments: Increments are a logical consequence of using iterations [1]. After each iteration, an improvement over the last version should be achieved.

c) Continuity: Continuity describes the idea, that every step in development is performed continuously [8] – be it testing, deploying or planning.

d) Automation: Automating repeated tasks, such as building or deployments, reduces the risk of human error [1], [10], [11].

e) Self-Service: Self-service in DevOps is the idea of reducing the time it takes to process requests to other departments – for example setting up environments – by allowing teams to execute their request themselves [1].

f) Collaboration: Collaboration is arguably one of the most general principles, but can be simply understood as improving communication in teams by colocation, removing hand-offs or focusing on personal contact [1], [8], [11].

g) Holism: Holism states that teams should be organized as such, that they can independently develop, test and deploy value to customers, instead of organizing them after their function or the architectural layer they are working on [1].

B. Practices of DevOps

While the principles of DevOps are general guidelines, the practices of DevOps are concrete methods that can be employed to adopt DevOps principles. However, as DevOps is not conclusively defined, these practices are merely methods that are employed commonly in DevOps settings, but not strictly prerequisites to do DevOps.

a) Continuous Integration: Continuous Integration (CI) is a software development practice where developers integrate their work often – commonly at least daily [14]. CI defines a clear, automated process of integrating many small pieces of work frequently.

b) Continuous Delivery: Continuous Delivery (CD) takes Continuous Integration a step further, by ensuring that the result of the pipeline is a potentially deployable artifact and the infrastructure is in a state, where the artifacts can be deployed [15].

c) Communication: Arguably the most important part of DevOps as a change of culture is to improve the communication within an organization [1], [8]. Communication as a practice of DevOps means making communicating within a team more efficient.

d) Monitoring and Logging: Another practice of DevOps is the use of monitoring and logging technologies [1], [7], [8], [16]. This practice includes collecting metrics about the runtime behavior of software, such as current load, response time or performance metrics in general.

e) Infrastructure as Code: Using Infrastructure as Code (IaC) configuration management tools such as Puppet, Ansible, Chef or Terraform is the logical consequence of the self-service and automation principles of DevOps [1], [10], [17].

f) Microservices: Another consequence of the new team structures proposed with the holism principle of DevOps is that the scope of work of one team must be smaller and more focused than before. In practice this is often achieved by using microservice architecture design approaches [1], [10], [16].

IV. GitOps

GitOps is a term coined by Weaveworks in 2017 [4]–[6], [18], [19]. They define GitOps as a model for operating Kubernetes clusters or cloud-native applications in general using the Version Control System Git as single source of truth. This means, environments are only operated via the contents of a Git repository, which includes creating, changing, and destroying environments. According to Weaveworks, this creates a developer-centric experience when managing applications, as

Git and corresponding tools such as CI and CD pipelines are already an essential for developing applications.

GitOps envisions declarative descriptions of an environment to be stored as IaC in a Git repository [4]. CI/CD mechanisms then can automate the deployment of an environment.

A. *Push-based GitOps*

Combining IaC with CI/CD pipelines results in the basic GitOps pipeline as described by Weaveworks [4]. Fig. 1 shows the setup of such a pipeline. GitOps uses two different kinds of repositories. One kind of repository contains all configuration describing the current production environment, while the other contains all application source code and the IaC configurations required to run this specific application. However, while there is typically only one environment repository, there might be multiple application repositories, with each containing only an independently deployable component of the application. The CI pipeline of the application repositories updates the configuration of its environment in the environment repository.

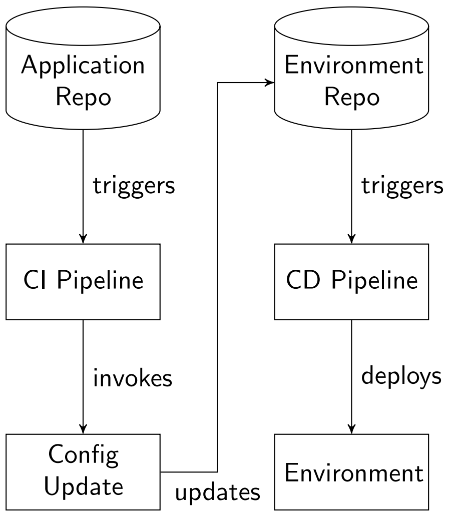

Fig. 1. Push-based GitOps Workflow [20]

Updates to the environment repository then trigger a CD pipeline which applies the IaC configurations and brings the environment to the new desired state [4], [19]. Different Git branches can be deployed to different environments, allowing for maintaining different stages.

Using this way of deploying applications and managing infrastructure has several benefits [4], [21]. First, this automated approach of deploying applications and creating new or changing existing infrastructure results in faster feedback loops, that developers can use to further improve the product. Due to the self-service nature of this approach, the infrastructure can be created quickly and without the organizational overhead. Second, due to using Git, there is complete transparency of when, why and by whom a change to the infrastructure was made, as Git stores the complete history of all changes ever made to a repository. This can be used as an audit log to meet compliance criteria for security, reliability and confidentiality concerns. Third, GitOps increases the reliability of the application itself, as Git allows to easily revert commits and thus rolling back the environment changes. Fourth, Git also allows for stronger security guarantees, as it allows to prove authorship and origin of changes by digitally signing commits. Lastly, this approach allows to quickly restore infrastructure if need be. Since the complete state of the infrastructure is stored at any point, it can be reproduced on different hardware or cloud providers.

B. *Pull-based GitOps*

One alteration described by Weaveworks is the use of a so-called Operator, which is mainly applicable to container orchestration platforms, such as Kubernetes [4]. The core difference between this and push-based GitOps is how the CD pipeline is implemented. As shown in Section IV-B, this role is now taken over by the operator.

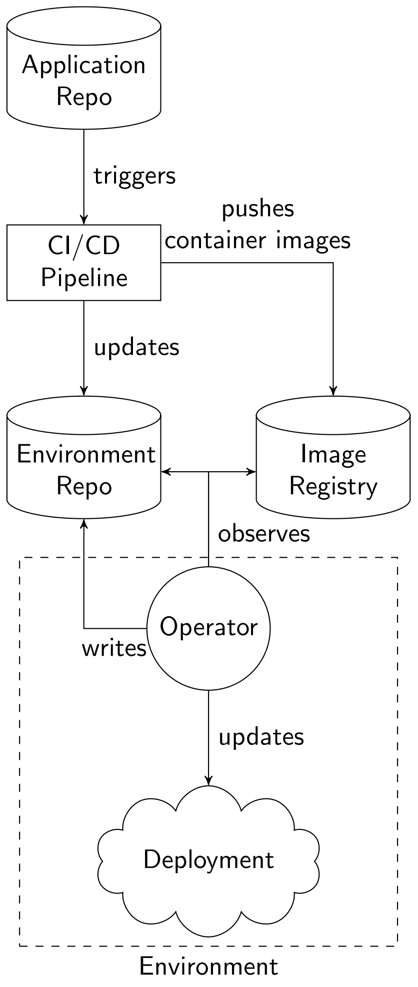

Fig. 2. Pull-based GitOps Workflow

The operator is an application, which is deployed in the environment itself [4]. It continuously checks the environment repository for changes and mirrors them to the environment. This essentially reverses the usual flow of information in Continuous Deployment pipelines. In contrast to the push-based approach before (see Fig. 1), where the previous step always triggers the next step, this model is pull-based, because the operator continuously queries for changes in the environment repository.

This pull-based approach has several additional benefits over the push-based approach. For one, the continuously pulling operator can observe the current state of the deployed infrastructure and then also react on deviations of the state described by the environment repository and deploy the infrastructure again [4]. This not only automatically tries to restore the desired state, but the operator can also automatically alert developers when the observed state does not match the desired state for some time. Another benefit of this approach is that the deployment happens completely from inside of the environment. Credentials can stay inside the environment, whereas the push-based approach requires them to be available in the CD pipeline.

V. Comparison of DevOps and GitOps

At the first glance, the most striking commonality of DevOps and GitOps is certainly the similarity of their names. Both incorporate the term operations abbreviated as Ops in their name. This makes sense, as they both share the overall goal of optimizing traditional IT operations in some way. However, the way both technologies try to achieve this is very different. DevOps focuses on changing company culture to deliver more value to customers quicker [1]. GitOps, on the other hand tries to make operating IT infrastructure a more developer-centric experience by employing a new CD technique.

This makes GitOps much more focused on concrete technology stacks, whereas DevOps allows organizations to pick tools that work for their use case without any restrictions. GitOps requires the use of a VCS and although any VCS can be used, it implies with its name that Git should be used. GitOps also strictly requires a CI/CD provider, whereas this is only a commonly used practice of DevOps. Lastly, according to Weaveworks [4], GitOps is tailored to Kubernetes or other container orchestration platforms.

A. Comparison by Principles

Although both techniques appear very different at the first glance, DevOps and GitOps are not mutually exclusive. Both concepts are only contrary in one regard, that is DevOps' Holism principle and the developer-centric experience of GitOps. All other principles of one technology do not contradict the other technology or are even employed in both technologies, as shown in Fig. 3.

The incremental approach of DevOps can also be found in GitOps. While DevOps' increments are more focused on advancing the product as a whole, GitOps produces increments of the used infrastructure. Also, the continuity principle can be applied to GitOps, as it essentially is an elaborate CD technique following a certain pattern. For this reason, also the automation principle is important for GitOps. CI/CD pipelines require building and testing to be executable automatically. Lastly, both techniques share the self-service principle since

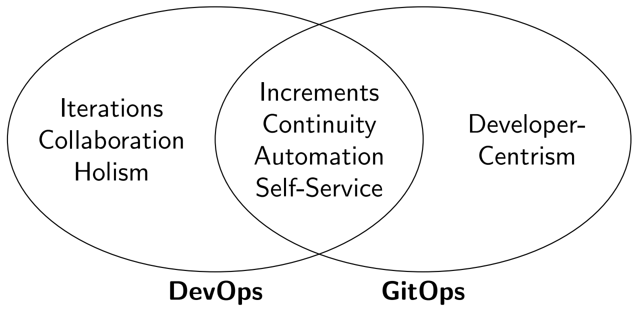

Fig. 3. Mapping of DevOps and GitOps Principles

development teams can make requests to create environments themselves through the environment repository.

The aspects that are a part of DevOps, but are not necessary for GitOps are the iterative approach and the focus on collaboration. While GitOps does not strictly require working iteratively, automatic deployments of increments of environments certainly make it easy to do so.

Since GitOps is more an approach on how to concretely implement CD, GitOps also misses the focus on collaboration of DevOps. The complete self-service nature of GitOps already reduces collaboration with an operations team to a minimum. However, improving collaboration to achieve an overall better quality of a system is possible and not contradictory to GitOps.

B. Comparison by Practices

Also, almost all of the practices of DevOps can also be found in GitOps. Fig. 4 shows that only the Communication and Microservices practices of DevOps are not directly employed in GitOps. However, similarly to the DevOps principles mentioned before, GitOps does not restrict using any of these practices in any way, it just focuses more on the deployment and operation of applications, while these practices focus more on the development approach. With the focus on deployment and operation, the self-service nature of GitOps the declaratively describable environment makes communication less important, than for more diverse IT infrastructure.

The most central DevOps practices, that can also be found in GitOps is the heavy use of CI/CD pipelines. DevOps uses these mainly for continuously performing all necessary steps to keep applications deliverable at any point in time. GitOps extends this with continuous deployment of the infrastructure.

Similar to DevOps, this continuously repeated deployment of IT infrastructure also leads to the adoption of IaC in GitOps. Compared to DevOps, GitOps again specifies how this practice should be employed much more detailed. While DevOps uses IaC only as a way to formulate requests to self-service environments, GitOps specifies exactly when the IaC should be applied.

Monitoring is another practice that is a part of both DevOps and GitOps. Its role in DevOps is mainly used to continuously receiving feedback about the runtime behavior of software in production. GitOps extends this requirement of observability also to infrastructure. This allows the operator in the pull-based

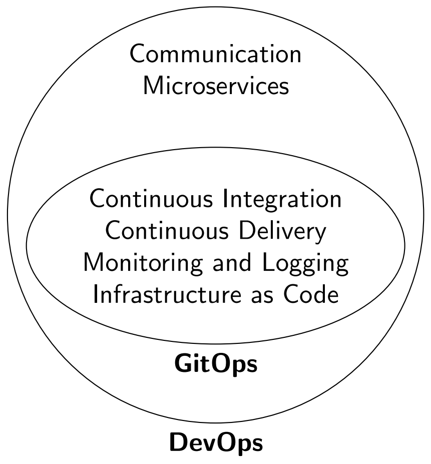

Fig. 4. Mapping of DevOps and GitOps Practices

deployment approach to recreate misbehaving infrastructure and applications automatically. However, also for this practice, GitOps uses it to base more automation on it and specifies how it should be employed in more detail than DevOps.

VI. Conclusion and Future Work

Today's market requires organizations to adapt to faster release cycles, which often includes eliminating the bottleneck of traditional IT operations. DevOps and GitOps claim to address these problems. GitOps even claims to be the evolution of DevOps. This paper presented the principles and practices of both approaches and compared them.

The comparison shows that the principles and practices of GitOps do not supersede those of DevOps, but most are shared in both approaches. It also showed that using one technique does not hinder the use of the other. So there is no evolution in the sense that GitOps replaces DevOps although they certainly share a common goal. The way how both approaches try to achieve this goal, however, is very different. DevOps tries to focus more on abstract concepts, such as company culture, whereas GitOps really gives organizations concrete tools to use to implement a Continuous Delivery workflow.

The result of this comparison is, by its nature, dependent on the definitions of DevOps and GitOps. Currently, no extensive definition of either concept exists. While GitOps is mainly the product of Weaveworks' work, they continue to refine their definition constantly. DevOps, on the other hand, is not attributed to only one organization, but mainly derives its definitions from how it is used in practice. Due to this fact, additional research on both topics is needed.

References

[1] G. Kim, J. Humble, P. Debois, and J. Willis, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, 2016.

[2] D. Safar and WeaveWorks, "DevOps: The Next Evolution - GitOps," 2017, accessed 2019-06-27. [Online]. Available: https://www.weave.works/blog/devops-the-next-evolution-gitops

[3] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, "What is DevOps?: A Systematic Mapping Study on Definitions and Practices," in *Proceedings of the Scientific Workshop Proceedings of XP2016, Edinburgh, Scotland, UK, May 24, 2016*. ACM, 2016, p. 12.

[4] Weaveworks, "GitOps," accessed 2019-05-02. [Online]. Available: https://www.weave.works/technologies/gitops/

[5] ——, "GitOps – Operations by Pull Request," accessed 2019-05-02. [Online]. Available: https://www.weave.works/blog/gitops-operations-by-pull-request

[6] ——, "What Is GitOps Really?" accessed 2019-05-02. [Online]. Available: https://www.weave.works/blog/what-is-gitops-really

[7] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, "Towards a benefits dependency network for DevOps based on a systematic literature review," *Journal of Software: Evolution and Process*, vol. 30, no. 11, p. e1957, jul 2018.

[8] J. Davis and R. Daniels, *Effective DevOps: building a culture of collaboration, affinity, and tooling at scale*. "O'Reilly Media, Inc.", 2016.

[9] D. Sato, *Devops in Practice: Reliable and automated software delivery*. Casa do Código, 2014.

[10] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," *IEEE Software*, vol. 33, no. 3, pp. 94–100, 2016.

[11] L. E. Lwakatare, P. Kuvaja, and M. Oivo, "Dimensions of devops," in *Agile Processes, in Software Engineering, and Extreme Programming - 16th International Conference, XP 2015, Helsinki, Finland, May 25-29, 2015, Proceedings*, ser. Lecture Notes in Business Information Processing, C. Lassenius, T. Dingsøyr, and M. Paasivaara, Eds., vol. 212. Springer, 2015, pp. 212–217. [Online]. Available: https://doi.org/10.1007/978-3-319-18612-2_19

[12] C. Haddad, "DevOps = DevOps Principles + DevOps Practices," 2014, accessed 2019-06-08. [Online]. Available: https://dzone.com/articles/devops-devops-principles

[13] Amazon Web Services, Inc., "What is DevOps?" 2019, accessed 2019-06-08. [Online]. Available: https://aws.amazon.com/devops/what-is-devops

[14] M. Fowler, "Continuous Integration," 2006, accessed 2019-06-06. [Online]. Available: https://martinfowler.com/articles/continuousIntegration.html

[15] L. Chen, "Continuous Delivery: Huge Benefits, but Challenges Too," *IEEE Software*, vol. 32, no. 2, pp. 50–54, 2015.

[16] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," *IEEE Software*, vol. 33, no. 3, pp. 42–52, 2016.

[17] N. Paez, "Versioning Strategy for DevOps Implementations," *2018 Congreso Argentino de Ciencias de la Informática y Desarrollos de Investigación (CACIDI)*, 2018.

[18] V. Farcic, *The DevOps 2.4 Toolkit: Continuous Deployment to Kubernetes: Continuously deploying applications with Jenkins to a Kubernetes cluster*. Packt Publishing Ltd, 2019.

[19] B. Burns, E. Villalba, D. Strebel, and L. Evenson, *Kubernetes Best Practices: Blueprints for Building Successful Applications*. O'Reilly Media, Inc., 2020.

[20] Google, "GitOps-style continuous delivery with Cloud Build," 2019, accessed 2019-06-25. [Online]. Available: https://cloud.google.com/kubernetes-engine/docs/tutorials/gitops-cloud-build

[21] T. A. Limoncelli, "GitOps: A Path to More Self-service IT," *ACM Queue*, vol. 16, no. 3, p. 20, 2018.