Strategies for Promoting Releases in GitOps Environments

Thomas Stadler, BSc
FH-Burgenland, Eisenstadt, Österreich

ABSTRACT:
From the principles of DevOps came GitOps, as a collection of principles and best practices for the operation of software systems. At the center, or to be precise, the source - as the single source of truth - stands thereby the Git Repository as the revision control system. The state of the managed system is completely defined declaratively as code. A GitOps controller takes care of the continuous reconciliation between the declared state and the current state. The promotion of new software releases between several environments shows itself to be an open problem at present. Uniform standard practices are missing, as well as necessary tools in the open source community. This thesis has the goal, to address the promotion of releases in GitOps environments. Abstract models of deployment environments as well as promotion workflows will be defined. Based on the previously defined models, a standardized solution for the promotion of releases will be implemented according to the GitOps principles, and compared to the existing solutions.

1 INTRODUCTION

Increasingly more organizations are adopting a DevOps culture to develop new applications and services at high velocity. After all, a culture that encourages shared responsibility, transparency and rapid feedback, helps narrow the gaps between teams and thus accelerate processes. GitOps was born out of the need for rapid innovation.

GitOps is a set of principles for operating and managing software systems. These principles are derived from modern software operations, but also have their roots in existing and widely adopted best practices. The primary four principles, which serve as principles for the desired state of a system managed by GitOps are the following:

- **Declarative**
  A system managed by GitOps must have its desired state expressed declaratively.
- **Versioned and Immutable**
  Desired state is stored in a way that enforces immutability, versioning and retains a complete version history.
- **Pulled Automatically**
  Software agents automatically pull the desired state declarations from the source.

- **Continuously Reconciled**
  Software agents continuously observe actual system state and attempt to apply the desired state.

(opengitops.dev, 2023b)

These principles are defined in OpenGitOps version 1.0.0. OpenGitOps is a Cloud Native Computing Foundation (CNCF) sandbox project under the GitOps Working Group. The GitOps Working Group is a working group under the the CNCF App Delivery TAG, with the goal of establishing a clear vendor-neutral, principle-driven meaning of GitOps. This will provide a foundation for interoperability between tools, conformance and certification through enduring programs, documents and code (opengitops.dev, 2023a).

It should be noted that the term GitOps is not exclusively limited to the principles defined in OpenGitOps, but goes beyond that.

GitOps can be viewed as an evolution of Infrastructure as Code (IaC) that uses Git as a version control system for infrastructure configurations. GitOps lowers the cost of building self-service IT systems and enables self-service operations that were previously unjustifiable (Limoncelli, 2018). It improves the ability to operate systems securely by allowing unprivileged users to make major changes, without granting these users direct access to the target system. Through so-called pull requests, ordinary users can submit a request of their proposed changes. These pull requests can then be reviewed and accepted by more privileged users. If necessary, adjustments can be suggested or incorporated by the reviewers themselves.

Security can be improved by adding automated tests. These tests can include Linting (validation of syntax), enforcing style guides, running end-to-end and unit tests, or enforcing security policies. Security reviews and audits become easier because every change is tracked. Every change or commit is permanently recorded in the Git history (Limoncelli, 2018).

GitOps as a practice for releasing software has several advantages, but like other solutions, GitOps also has several shortcomings. Currently, there are several unresolved problems with GitOps. On the one hand, these are problems that often were a problem in an organization already before GitOps was implemented, on the other hand, a GitOps strategy brings with it some limitations, which lead to new problems. The organization Codefresh as a driving force for GitOps, - due to its developments of various GitOps tools - speaks of the following problems with the currently practiced GitOps:

- GitOps covers only a subset of the software lifecycle
- Splitting CI and CD with GitOps is not straightforward
- GitOps doesn't address promotion of releases between environments
- There is no standard practice for modeling multi-environment configurations
- GitOps breaks down with auto-scaling and dynamic resources
- There is no standard practice for GitOps rollbacks
- Observability for GitOps (and Git) is immature

- Auditing is problematic despite having all information in Git
- Running GitOps at scale is difficult
- GitOps and Helm do not always work well together
- Continuous Deployment and GitOps do not mix together
- There is no standard practice for managing secrets

(codefresh.io, 2023)

2 RELATED WORK

The following are current practices shown on the topic, how best to model different deployment environments with GitOps.

2.1 branch-per-environment approach

Kapelonis (2021) published Codefresh's best practices on modeling environments in GitOps in the blog post "Stop Using Branches for Deploying to Different GitOps Environments". The problems with the branch-per-environment approach are shown. This approach is an anti-pattern, and to be avoided at all costs. A number of points are made and justified in detail.

- Using different Git branches for deployment environments is a relic of the past.
- Pull requests and merges between different branches is problematic.
- People are tempted to include environment specific code and create configuration drift.
- As soon as you have a large number of environments, maintenance of all environments gets quickly out of hand.
- The branch-per-environment model goes against the existing Kubernetes ecosystem.

(Kapelonis, 2021)

2.2 folder-per-environment approach

Kapelonis (2022) provides some best practices from Codefresh on the problem in the blog post: "How to Model Your Gitops Environments and Promote Releases Between Them." Es werden folgende Kategorien der "Umgebungskonfiguration" vorgeschlagen:

- **Application version** in the form of the container tag used. This is the setting that increases with each release and decreases with each rollback - and is always promoted between environments.
- **Kubernetes-specific settings** for the application (manifests).

- **Mainly static settings** for the business logic. These include external URLs, internal queue sizes, UI default values, authentication profiles, etc. By "mostly static" they mean settings that are defined once for each environment and then generally do not change. These settings should never be promoted between environments.
- **Non-Static Business Settings**. This is the same as the previous item, but it includes settings that are indeed promoted between environments. These can be a global VAT setting, parameters for the recommendation engine, the available bitrate encodings, and any other setting that is specific to the business or enterprise.

(Kapelonis, 2022)

The application version and non-static business settings are promoted between environments. Kapelonis (2022) suggests defining the configuration parameters per category in a separate file. As a result, the four categories can be promoted independently. The folder-per-environment approach is proposed. In general, all promotions are just copy operations. Unlike the branch-per-environment approach, with the folder-per-environment approach everything can be moved from any environment to any other environment without fear that the wrong changes will be applied. Especially when it comes to backporting configurations, the folder-per-environment approach shines, because configurations can be easily moved both "forward" and "backward", even between unrelated environments.

Scenario: Promotion of the application version from qa to staging-us.

```
cp envs/qa/version.yml envs/staging-us/version.yml
```

Scenario: Promotion of the application from prod-eu to prod-us together with the additional configuration. The settings file(s) are also copied.

```
cp envs/prod-eu/version.yml envs/prod-us/version.yml
cp envs/prod-eu/settings.yml envs/prod-us/settings.yml
```

Scenario: Backport all settings from qa to integration-non-gpu.

```
cp envs/qa/settings.yml envs/integration-non-gpu/settings.yml
```

Kapelonis (2022) notes, that the copy operations are for illustrative purposes only. In a real-world application, this operation would be performed automatically by a CI system or other orchestration tool. And depending on the environment perhaps a pull request should be created first instead of processing the folder in the main branch directly (Kapelonis, 2022).

Kapelonis (2022) suggests the following advantages of the folder-per-environment approach:

- The order of commits in the Git repository is irrelevant. When a file is copied from one folder to the next, the commit history doesn't matter.
- If only files are copied, only exactly what is needed is promoted, and nothing else.
- No need to use git cherry picks or other advanced git methods to promote releases. Only files need to be copied.

- The user is free to commit any change from any environment to any environment, without any restrictions on the proper "order" of environments.
- Using diff operations on files, it is easy to track what is different between environments in all directions (both in the source and target environments and vice versa).

(Kapelonis, 2022)

3 RESEARCH QUESTION

The goal of this thesis is to address the problem of promoting releases between different environments in GitOps.

Large organizations in particular usually have many non-production and production environments such as: dev, qa, staging-us, staging-eu, production-us, production-eu. Usually new releases are automatically rolled out in an environment such as qa - by means of a CI pipeline. Now the task is to promote new changes, which are brought about by a new release, into subsequent or other environments. Current GitOps tools do not have a simple answer to the question about what is the right approach for the process of promotion.

In particular, this thesis aims to explore existing strategies for the solution of the problem with existing tools. Furthermore, a prototype of a newly proposed strategy will be developed. Subsequently, the prototype will be tested in a laboratory experiment and compared to currently existing strategies.

To achieve the goal of the thesis, the following research questions were identified:

- How can the promotion of releases in GitOps environments be designed?
  - What possibilities do existing tools offer for the promotion of releases with multiple deployment environments?
  - How can deployment environments, as well as promotion workflows be modeled abstractly?
  - How can abstract modeling be used to implement a standardized solution for promoting releases according to GitOps principles?

4 METHODOLOGY

In the beginning, existing literature on the topic, as well as the concrete research questions are discussed. Furthermore, various opinions and approaches of pioneering organizations in the GitOps community are analyzed.

Next, existing software solutions, which are available under open source licenses, will be evaluated for their appropriate use for the present research. The aim is to find out the extent to which the defined

research questions can be addressed with existing solutions. Finally, a baseline is defined as state-of-the-art.

Furthermore, abstract models for deployment environments and promotion workflows will be defined. These serve as a basis for the design of the prototype in the next step.

As far as possible and reasonable, existing toolkits and best practices will be used for the prototype. It is desirable to solely extend the Kubernetes-API and the GitOps-Toolkit to integrate the prototype natively into the existing ecosystem around GitOps within the CNCF.

According to Houde und Hill (1997) a prototype is any representation of a design idea, regardless of the medium. A prototype is something that serves as a model or inspiration for later developments (Houde & Hill, 1997).

The developed prototype will be compared to existing approaches to solving the problem in a laboratory experiment.

Montgomery (2017) defines an experiment as a test or series of runs in which intentional changes are made to the input variables, in order to then observe and identify reasons for changes in the output result (Montgomery, 2017).

5 RESULTS

The results of the literature review so far show, that there is no standard practice for the promotion of releases between multiple environments in GitOps. The existing GitOps tools do not provide a native solutions for this. As a best practice, it is suggested that the promotion operations should be outsourced to the CI system - in the form of simple file copy operations. According to the literature as well as current observations, any kind of standardization is missing, when it comes to defining multiple environments with GitOps and promoting releases between them.

As the primary outcome of this thesis, an answer to the research questions is expected, by presenting a software solution as a prototype. The developed prototype should ideally serve as a modular extension to Kubernetes and existing GitOps tooling.

The researched results, the developed solution approach, as well as the proposed prototype, can finally be be donated to the CNCF.

6 CONCLUSION AND FUTURE WORK

Although the adoption of GitOps brings some advantages on the one hand, on the other hand, it creates new problems that did not exist before, or were better solved with previously followed strategies. In addition, with GitOps the demands on good software have increased - as so often - and new problems have been identified as a result. These newly identified problems may currently turn out to be merely luxury problems for many organizations. However, as an organization grows, these current luxury problems with the increasing growth of an organization and the increasing demands that come with it, may emerge as pressing problems.

The higher the complexity and the needed degree of automation of an organization, as well as the adoption of DevOps and the principles like Continuous Integration (CI), Continuous Delivery (CD), Continuous Deployment (CDP) - the more important it becomes to fully automate the promotion of releases in GitOps environments.

References

argoproj.io. (2023). Get More Done with Kubernetes [(Accessed on 01/01/2023)]. https://https://argoproj.io/

codefresh.io. (2023). The pains of GitOps 1.0 [(Accessed on 01/01/2023)]. https://codefresh.io/blog/pains-gitops-1-0/

Houde, S., & Hill, C. (1997). Chapter 16 - What do Prototypes Prototype? In M. G. Helander, T. K. Landauer & P. V. Prabhu (Hrsg.), *Handbook of Human-Computer Interaction (Second Edition)* (Second Edition, S. 367–381). North-Holland. https://doi.org/https://doi.org/10.1016/B978-044481862-1.50082-0

Kapelonis, K. (2022). How to Model Your Gitops Environments and Promote Releases between Them [(Accessed on 01/01/2023)]. https://codefresh.io/blog/how-to-model-your-gitops-environments-and-promote-releases-between-them/

Kapelonis, K. (2021). Stop Using Branches for Deploying to Different GitOps Environments [(Accessed on 01/01/2023)]. https://codefresh.io/blog/stop-using-branches-deploying-different-gitops-environments/

Limoncelli, T. A. (2018). GitOps: A Path to More Self-Service IT. *Commun. ACM*, *61*(9), 38–42. https://doi.org/10.1145/3233241

Montgomery, D. C. (2017). *Design and analysis of experiments*. John wiley & sons.

opengitops.dev. (2023a). GitOps Documents v1.0.0 [(Accessed on 01/01/2023)]. https://github.com/open-gitops/documents/blob/v1.0.0/README.md

opengitops.dev. (2023b). GitOps Principles v1.0.0 [(Accessed on 01/01/2023)]. https://github.com/open-gitops/documents/blob/v1.0.0/PRINCIPLES.md