# GITOPS EXPLAINED: CONCEPTS, BENEFITS & GETTING STARTED



DevOps has changed the way we develop and manage applications, resulting in faster, more consistent, more collaborative development cycles. It has evolved further by incorporating microservices-based architectures and even databases in the form of database DevOps.

Now there's a new kind of Ops.

GitOps is rapidly gaining popularity to extend the scope of DevOps further to include application infrastructure. In this article, we will have a look at GitOps and how to leverage it for delivering cloud-native applications.

## What is GitOps?

GitOps is a set of practices that are aimed at managing the underlying infrastructure of an application. It utilizes Git as the source code management tool for managing the infrastructure code. In other words, GitOps is an evaluation of infrastructure as code and DevOps practices which uses Git as the single source of truth for provisioning infrastructure declaratively.

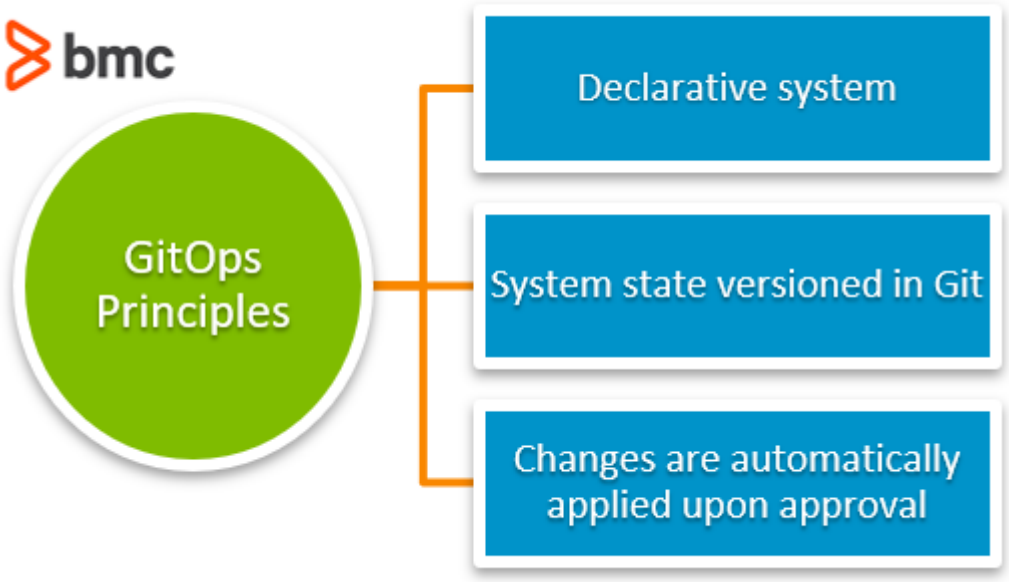
The term GitOps was confined by Weaveworks in 2017 and was primarily aimed at managing Kubernetes deployments. However, now it has evolved into supporting other infrastructure management solutions such as Terraform.

The goal of GitOps is to simplify and streamline the development process. This leads to building reproducible infrastructure with proper state management, which both:

- Increases the overall visibility
- Reduces the management overhead of the application infrastructure

GitOps allows developers or the Ops team to declare their infrastructure as code and version control them via Git. Whenever a new change is required, a pull request with the new change is created, executing the CI/CD pipeline to provision or modify the infrastructure.

Additionally, GitOps offers users the flexibility to select any tool, technology, or platform and use the same DevOps practices when creating infrastructure.




# Principles of GitOps

There are a few core principles that apply when implementing and dealing with GitOps. Let's take a look.

## Declarative system

In the GitOps model, the complete system is configured declaratively. This declarative approach is focused on the result (desired state) rather than the steps needed to achieve the required result.

This declarative approach allows users to specify the end goal without worrying about each explicit step needed as in an imperative approach. As a state-aware declarative approach, users can easily store these states in Git, facilitating convenient deployments and rollbacks.

## The system state is versioned in Git

All the declarative states are stored in the version-controlled system, which acts as the single source of truth. With this version-controlled approach, all the system infrastructure changes are available chronologically, enabling users to identify infrastructure changes over time easily. It is also helpful in:

- Troubleshooting
- Auditing

- Rollbacks

## Changes are automatically applied when approved

When a pull or merge request is made, it will be verified and then approved since all the changes are stored in Git.

Furthermore, it should be automated to apply changes to the system automatically when they are approved. GitOps prefers immediate automated deployments to achieve the desired state quickly.

# Benefits of GitOps approach

GitOps enables organizations to streamline their infrastructure provisioning and application deployment strategies. This leads to an extensive range of benefits/

## Ease of infrastructure management

GitOps allows users to manage infrastructure easily as a part of the overall DevOps process by testing and deploying changes quickly with the help of:

- CI/CD tools
- Automated deployments
- Shorter feedback loops

By having infrastructure as a part of the CI/CD pipeline, any application modification that requires an infrastructure change can be bundled and managed together. It also makes it easier to troubleshoot production bugs, such as network connectivity, as users have better visibility of the changes with the deployments.

## Increased productivity

The source-controlled, validated infrastructure reduces configuration errors that can occur during deployments, saving time for Ops teams to diagnose and fix those errors. Besides, source control allows multiple teams to work on different parts of the infrastructure without interfering with each other's work.

This leads to increased productivity in both the Dev and Ops teams, which ultimately results in faster developments and deployments.

## Improved reliability & stability

In a version-controlled infrastructure approach, infrastructure changes are validated, and the state is always maintained. When coupled with the declarative approach to provisioning infrastructure, it can drastically reduce errors in the application infrastructure.

On top of that, version control allows users to audit changes and roll back to previous states with ease. This, in turn, improves the stability and reliability of the application.

Another factor is managing config drifts. As a modern application, there will be requirements to apply manual hotfixes or small changes that can lead to config drifts. With GitOps, users can:

- Easily identify such drifts between the declared infrastructure and the actual infrastructure
- Quickly mitigate them

## Standardization

GitOps helps to standardize infrastructure deployments. Infrastructure can undergo almost the same verification and validation process for application code with consistent:

- End-to-end workflows
- Standardized code structures
- Documentation
- Testing methods

This introduces standardized and fully reproducible infrastructure configurations.

## Enhanced security

GitOps approach helps organizations enforce security best practices and track all the infrastructure changes and corresponding states available via Git SCM. Moreover, this organized approach enables proper audit trails to identify details related to infrastructure changes such as

- Responsible users
- Deployment data time
- Affected resources
- Etc.

The GitOps approach also helps to streamline the management of authentication and authorization requirements for infrastructure modification. Since infrastructure is a part of the CI/CD pipeline, individual developers do not require direct access to resources, hence not needing credentials to access said resource.

This also makes it necessary for users to only provide credentials at the time of execution in the pipeline. This further enforces strict access controls to underlying resources reducing attack vectors to the infrastructure.

However, we have to properly implement GitOps as a part of the delivery pipeline to gain all the above benefits. In the next section, we'll at how to implement a GitOps workflow properly.

# How to implement GitOps

If your organization already has a properly implemented DevOps pipeline using Git as the SCM tool, implementing GitOps to cover the infrastructure is a pretty straightforward process. Simply:

1. Add the infrastructure code into the Git repository.
2. Configure the CI/CD pipeline to include the infrastructure repository as a part of the delivery pipeline.

On the other hand, if you start from scratch, the first thing to consider is the Git repository. As GitOps is platform-agnostic, users can utilize any local or cloud-based Git repository such as:

- GitHub

- BitBucket
- Azure Repos
- GitLab
- Etc.

Then comes the CI/CD pipeline platform, which boils down to your preferred and familiar platform tools. Tools like Jenkins and CircleCI can be used with any git repository. BitBucket Pipeline and GitLab Pipelines prefer their own code repositories. Whatever the selected pipeline platform, its primary goal will be to:

- Automate the delivery process.
- Facilitate a clear workflow between the Git repository and the infrastructure management platform.

*(Learn how to build a CI/CD pipeline.)*

The differentiating factor of a GitOps pipeline is the GitOps operator. This mechanism sits between the pipeline and the infrastructure platform, acting as a middleman for facilitating communications between them. There are multiple available operators such as:

- Kubernetes Operator
- Terraform Cloud Operator
- Azure Service Operator
- Etc.

## GitOps workflow

Now, we've got a Git repository and a properly configured CI/CD pipeline. An infrastructure engineer will:

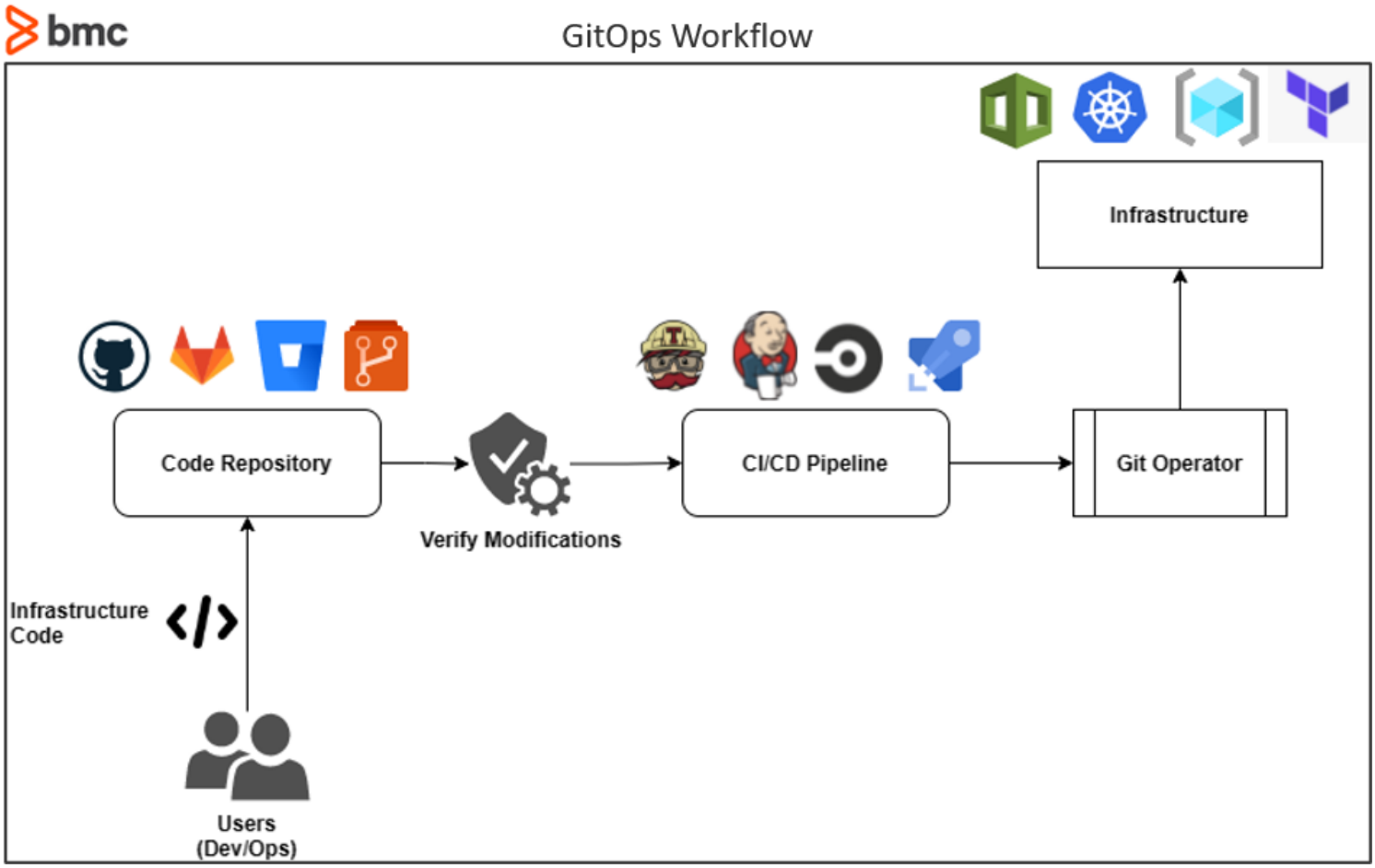
1. Declare the infrastructure as code.
2. Push the code to a Git repository.
3. Create a pull request.

After that, this code can be reviewed and scrutinized by another team member and finally get approved—which triggers the CI/CD pipeline. The pipeline will then inform the Git Operator, which will pick up all the modifications and the desired new state.

The Git operator will then compare the state of the existing infrastructure and desired state. This relates to the observability concept, which is the ability to measure the internal states of a system. This observability ensures that the desired state and the observed state of the infrastructure are the same.

If the states are different, it will seamlessly orchestrate and provision the underlying infrastructure to match the desired state.

This workflow can be further extended by incorporating multiple infrastructure deployments such as staging and production. This way, the initial deployment will be carried out in a staging environment, which acts as a further fail safe before the final deployment of the infrastructure modifications to the production environment. Likewise, a GitOps pipeline offers limitless possibilities to extend and integrate infrastructure to meet any needs.




# GitOps example

Let's take a real-world scenario of GitOps where a web application is deployed in a cloud environment.

Assume there is a sudden spike in traffic to the application, which creates performance issues, causing an unsatisfactory user experience when using the web application.

To address this issue, the delivery team wants to increase the resource allocation for the web application. With GitOps, users can define the resource increments and push the changes to the Git repository. Then these changes can be quickly reviewed and verified by other team members and approved for production deployment.

This triggers the GitOps CI/CD pipeline and initializes the git operator. Then the git operator will compare the states. This new configuration will identify this as a state change and automatically orchestrate the underlying infrastructure to match the desired state.

The delivery team only has to monitor any failures, which will also get automatically notified to the delivery team via the CI/CD pipeline. If there are no issues with the underlying infrastructure, it will be successfully modified with new resource allocations to meet the user demands.

But, what happens if the deployment fails, or a configuration error causes a networking issue?

GitOps approach allows users to quickly roll back to a previous state of the infrastructure seamlessly. Then they can again review and create infrastructure changes that address all these issues and trigger the CI/CD pipeline to deploy the changes automatically.

Without GitOps, reverting to a previous configuration state is a complicated task that becomes

nearly impossible if the changes are not properly documented.

## Streamlined, automated pipelines

GitOps translates all the manual, complicated infrastructure management tasks to a streamlined, automated pipeline. This declarative version-controlled approach makes the time-consuming, daunting task of managing infrastructure painless while increasing the visibility, reliability, and stability of the system.

## Related reading

- BMC DevOps Blog
- GitHub vs GitLab vs Bitbucket: What's The Difference & How To Choose
- How To Use Elastic Enterprise Search with GitHub
- Kubernetes Guide, a tutorial series
- Automation In DevOps: Why & How To Automate DevOps Practices
- What Is Cloud Native DevOps?