International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V73I5P119

Volume 73 Issue 5, 155-161, May 2025

Original Article

Evolution of CI/CD in Cloud-Native Development

Aditya Bhatia

New York University, New York, USA.

Corresponding Author : aditya.bhatia@nyu.edu

Received: 01 April 2025 Revised: 05 May 2025 Accepted: 17 May 2025 Published: 31 May 2025

Abstract *- As the conventional setup for Continuous Integration and Delivery (CI/CD) makes way for an architecture based on cloud-native principles, the criticality of having scalable, static workflows that are also automated and secure becomes ever more apparent. With the increasing integration of AI models in the software development lifecycle, GitOps principles provide critical traceability and reversibility for future CI/CD pipelines. My work in this area has led me to focus on GitOps and Kubernetes-native CI/CD for modern enterprises. These setups are not yet the de facto standard, but they represent an evolutionary step forward that increasingly looks to the principles of GitOps for implementing CI/CD in a Kubernetes landscape. This paper will not only interrogate these principles but also assess key technologies working behind these principles: Argo CD, Flux, Tekton, and Argo Workflows. It also presents some of the advancements and challenges in the field and what the future holds.*

Keywords *- CI/CD, Kubenetes, GitOps, Argo CD/Workflow, Flux, Tekton.*

1. Introduction

Continuous Integration (CI) and Continuous Delivery/Deployment (CD) are practices that automate building, testing, and releasing software. In the last decade, a lot has changed in the field. Traditional CI/CD tools (e.g. Jenkins) originated before containerization and were often used to run as monolithic services. With the increase in cloud-native development, which leverages containers, microservices, and dynamic infrastructure, CI/CD pipelines must handle container image builds, orchestration, and rapid deployment to platforms like Kubernetes. Cloud-native systems demand CI/CD systems that are distributed, scalable, and integrated with container orchestration. For example, previous monolithic Jenkins setups used to become bottlenecks or suffer from "Jenkins sprawl" as teams tried to scale the system.[2] New approaches are more scalable with microservices architecture and close integration with Kubernetes. Platforms like Tekton, which leverages Kubernetes custom resources to define pipelines, work well with the everything-as-code approach and GitOps. [1] Such CI/CD platforms align well with cloud-native principles for scalability and resilience.

In 2017, Weaveworks introduced GitOps, which extended the principles of DevOps in cloud-native systems. Devops focuses on collaboration between development and operations, with practices like Continuous Integration (CI), Infrastructure as Code (IaC), Continuous Delivery (CD), and monitoring. Gitops extended those practices by adding git-based version control to the above operations. The idea is to keep git repositories as a single source of truth, which means changes to the repository made with pull-request or merge-request should automatically apply to the underlying infrastructure. To do this, four core principles were defined by Alexis Richardson (Weaveworks CEO): (1) Declarative descriptions of the entire system, (2) Git as a Single Source of Truth Versioned and Immutable state stored in Git, (3) Kubernetes Operator Approved changes to the desired state are automatically applied to the system, and (4) Continuous Observability Software agents ensure correctness and alert on divergence. In short, Gitops extends the ideas of Iac and CD and accomplishes them by storing the state in git. Every change to the infrastructure and deployments is made to the repository in git, which allows all such changes to be reviewed as code changes. [2] [3] [4]

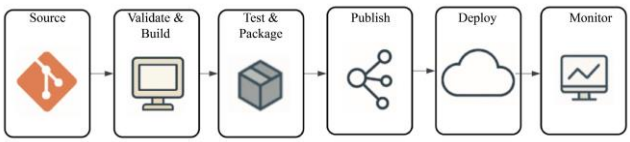

Fig. 1 Software Development Life Cycle

Kubernetes has become the standard orchestration platform for cloud-native applications in the industry. This has led to a tighter integration of the CI/CD systems with kubernetes. Traditional CI systems often run outside the cluster and push changes to the cluster. Although this approach works, it is less efficient and secure as the jobs need permission to modify the cluster. Some CI/CD solutions leverage the kubernetes cluster as an execution engine by running jobs in the cluster, such as screwdriver CI/CD. [5] The latest kubernetes native CI/CD systems are defined using CRD (custom resource definition) and run as

controllers inside the cluster. This design philosophy also aligns with GitOps's definition of everything as code. Using kubernetes for CI/CD provides scalability. CI/CD pipelines are defined as code in Git and can deploy swiftly and securely.

2. GitOps and Kubernetes CI/CD

2.1. GitOps Fundamentals

GitOps is a practice where Git repositories contain declarative descriptions of infrastructure and applications, and automated agents continuously ensure the deployed environment matches the source of truth in repositories. The following section will discuss some of the fundamental principles of GitOps.

2.1.1. Git as the Single Source of Truth

All environment definitions (Kubernetes manifests, Helm charts, etc.) live in Git. Changes are committed to Git, providing an immutable history of modifications. Every change can be traced to a commitment to improving transparency and auditability.

The Git history becomes the log of "who changed what, when," in the infrastructure, which makes it very easy to audit the trail of changes.

2.1.2. Declarative Desired State

A GitOps-managed system emphasizes defining environments declaratively (desired end state) rather than imperatively (step-by-step scripts). For Kubernetes, the desired cluster state (deployments, services, configs) is expressed in kubernetes YAML manifests.

2.1.3. Automated Reconciliation (Pull Model)

GitOps uses a pull-based deployment model; kubernetes controllers running in the target environment continuously pull the latest desired state from Git and reconcile the actual state to match the state defined in the git repository.

If the actual state drifts from what is in Git (due to manual changes or errors), the agent corrects it or alerts on the divergence to match back to Git changes.

This continuous controller reconciliation loop is key to drift detection – it ensures that what is running in the cluster is always what has been declared in Git, eliminating configuration drift.

2.1.4. Observability & Auditability & Security

Automated reconciliation allows GitOps to limit direct interaction with production systems, and all changes go through Git.

This has security benefits: access controls can be applied to the Git repos and CI process rather than granting many engineers direct cluster credentials. It also means security scans and policy checks can be done on the Git-stored configs before they reach production.

The GitOps agent usually has higher privileges in the cluster to make changes, so protecting that agent and the git-repo is critical for security considerations. Signed commits or trusted pipelines can decrease such concerns and ensure only authorized changes are applied.

2.1.5. Pull vs Push Deployments

GitOps favors pull-based deployments, where the cluster (through custom k8s controllers) pulls updates from Git instead of a CI server pushing changes into the cluster. Pull-based models are considered more secure and robust in GitOps since the cluster only needs read access to the repo, and external systems do not need direct write access to the cluster. Push-based CI/CD, for example, in systems like screwdriver CD or gitlab pipelines (where a pipeline script applies changes to the cluster using kubectl), can still be used. However, it breaks some GitOps principles by bypassing the automated state reconciliation in Git k8s, which can lead to configuration drift.

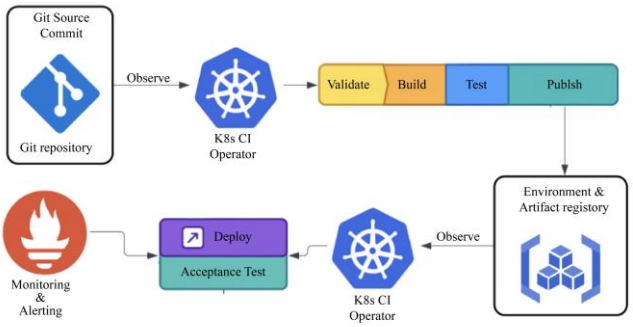

Fig. 2 CI/CD process evolution with GitOps

3. Tools using GitOps Principals

3.1. Kubernetes-Native Continuous Deployment

This section will explore open-source CD tools running on Kubernetes using GitOps principles.

3.1.1. Argo CD

A popular CNCF project that originated at Intuit. It is a declarative GitOps continuous delivery tool for Kubernetes. Argo CD watches one or more Git repositories and automatically syncs application definitions to Kubernetes clusters, which allows users to manage application deployments using Git repositories as the source of truth. It automates the deployment of the desired application states in the specified target environments. It supports deploying multiple clusters and has built-in role-based access control (RBAC) for multi-tenant use.

Argo CD focuses on continuous delivery (CD) and GitOps. It is not a CI engine but integrates with CI pipelines via webhooks or CLI. It is known for being relatively easy to use (a simple deployment and a polished UI) and is often the entry point for teams adopting GitOps. One trade-off is that Argo CD runs as a central service and is very resource-heavy; thus, scaling to very large numbers of applications or

clusters is challenging and may require sharding instances or using the Argo ApplicationSet controller to manage large numbers of deployments from a mono-repo and multiple clusters. [6][7]

3.1.2. Flux

Flux CD (now in its v2 incarnation) is another CNCF graduated project created initially by Weaveworks. Flux is a set of Kubernetes controllers (GitRepository, HelmRelease controllers, etc.) that continuously reconcile cluster state with what is in Git. Flux has no UI or server; it is a decentralized, modular architecture with a toolkit of controllers that can be assembled as needed.

Flux integrates with policy engines and other ecosystem tools (Helm, Kustomize, SOPS for secrets), making it more like a platform toolkit. Flux can be run on each cluster (agents per cluster), which naturally supports multi-cluster deployments. Flux is embedded in other platforms – GitLab's built-in GitOps feature uses Flux under the hood. [9]. Flux's design is very Kubernetes-native and flexible; it continuously auto-syncs cluster state to match Git and can even auto-promote images, making it powerful for specific use cases. However, compared to Argo CD, Flux lacks a native GUI (though third-party dashboards or plugins exist) and might have a steeper learning curve due to its modular and extensible nature. [8]

3.1.3. Fleet

Fleet is an open-source GitOps tool from Rancher (SUSE) aimed at GitOps at scale, specifically managing multiple clusters. [11] Fleet uses a centralized manager to deploy workloads to thousands of Kubernetes clusters based on Git repositories. It introduces the concept of bundling Kubernetes manifests and applying them to sets of clusters with placement rules. Fleet's design is lightweight enough for a single cluster, but it "really shines" when you have many clusters or teams to coordinate.

The fleet is fundamentally a set of Kubernetes custom resource definitions (CRDs) and controllers that manage GitOps for a single Kubernetes cluster or a large-scale deployment of Kubernetes clusters. For example, if a company has 100 Kubernetes clusters, Fleet can deploy a given application or config to all (or a subset) from one Git source and monitor the sync status across the fleet.

The trade-off is that Fleet is somewhat newer and more specialized; it may not have as feature-rich a GUI or community as Argo or Flux for single-cluster usage. Fleet fills a niche for large multi-cluster GitOps, emphasizing control and visibility across distributed clusters. [10]

3.2. Kubernetes-Native CI/CD Workflow Orchestration

Beyond deployment-focused GitOps tools, cloud-native development also requires CI/CD workflow engines to build, test, and release software in a Kubernetes-compatible way. Some key Kubernetes-native CI/CD orchestration products include Tekton Pipelines, Argo Workflows, Jenkins X and others. The following section will explore some of these products in detail.

3.2.1. Tekton Pipelines

Tekton is an open-source framework for creating CI/CD pipelines as Kubernetes resources. It defines a set of Kubernetes Custom Resources (CRs) such as Task, Pipeline, TaskRun, and PipelineRun that act as building blocks to assemble CI/CD pipelines. Each Tekton Task is a collection of steps (running in a container), and Pipelines stitch tasks together. Tekton controllers spin up pods to execute each step when a pipeline runs. Because Tekton defines everything in YAML and runs on Kubernetes, it fits the GitOps declarative and "pipeline as code" paradigms. Due to the high flexibility in defining pipelines using custom resources, it allows highly customizable pipelines for build, testing, and deployment.

The pipeline CRs are version-controlled in Git, along with application code and k8s deployment manifests. As Tekton pipeline configs are kubernetes objects, when accompanied by Gitops workflows like Argo and Flux, Tekton controllers in k8s also manage the pipelines. [12]

3.2.2. Argo Workflows

Argo Workflows is a container-native workflow engine for Kubernetes. With Argo, multi-step workflows can be defined (including DAGs – Directed Acyclic Graphs of tasks) as Kubernetes Custom Resource Definitions (CRDs). Each step in an Argo Workflow is a containerized operation, and Argo's controller schedules these steps on the cluster, handling dependencies and parallelism. Mainly, Argo Workflows are used to build pipelines for use cases like machine learning workflows, data processing, and batch jobs.

Argo workflows can be used to build CI job pipelines to run tasks in kubernetes. Argo Workflows has features like a web UI to visualize workflow DAGs, artefact passing between steps, retries, and an extensive library of examples. Argo Workflows tends to be more user-friendly for complex workflows, especially with its visualization and easier DAG definition. For CI/CD specifically, Argo Workflows can integrate with Argo Events (to trigger workflows on Git or registry events) and Argo CD (for deployment steps), creating a full GitOps pipeline with a UI. [13]

3.2.3. Jenkins X

Jenkins X is an open-source CI/CD platform that began as an extension of Jenkins for cloud-native apps. However, it evolved into a distinct solution built around Kubernetes. Jenkins X v3+ leverages Tekton for running pipelines and focuses on GitOps for managing environments. Jenkins X automates the creation of CI/CD pipelines, manages the promotion of applications between environments via GitOps, and provides developer-friendly tooling. Although Jenkins X has opinions around the best practices around

CI/CD, for example, defining code structure in a specific way in the git repository can automatically set up build and deployment pipelines and handle deployments through GitOps without manually defining the pipelines. Jenkins X leverages Tekton for pipeline execution on kubernetes and Flux for manifest deployment. With its moderate complexity of usage, setting up Jenkins X can be a complex task because of multiple components that act as dependencies. [14]

3.2.4. Others

Other Workflow tools like Spinnaker are not Kubernetes' native tools. These tools do not really use Git as the source of truth regarding reconciling with the cluster. Tools like Screwdriver CD have patterns like gitlab CI and Github Actions. Such tools are based on the pipelines and git webhooks defined either in UI or a manifest yaml file in the repository. There are also tools like Concourse CI that use containers for tasks, but not specifically Kubernetes CRDs. Cloud-specific solutions like GitLab CI or GitHub Actions integrate with K8s via runners/operators. All these tools are outside strict "Kubernetes-native" classification and use a push model rather than a pull model seen in other tools like Argo and Tekton Pipelines.

4. Advancements in CI/CD Architectures

Many advancements in CI/CD architectures and innovations have improved automation and scalability in recent years. In the next section, I will discuss some of the areas of these advancements in more detail.

4.1. Event-Driven and Declarative Pipelines

Event-driven CI/CD pipelines are the new normal, which means the pipelines get automatically triggered for events such as code pushes, new container images, or other system events without manual intervention. Kubernetes' event-driven approach allows for the declarative definition of such pipelines using Kubernetes Custom Resources (CRs). For instance, in Tekton pipelines, a Tekton Triggers component allows pipelines to start based on external events (e.g., a GitHub webhook or a message on a broker). For example, a push to a git-repo can send a webhook that Tekton Triggers catches, creating a custom resource PipelineRun to build and deploy the new commit. Similarly, Argo Events is a dedicated event-driven workflow automation framework in the Argo ecosystem. It can listen for triggers such as webhooks, cron schedules, message queues, etc., and then kick off Argo Workflows (or other actions) in response. Like Tekton Triggers, Argo Events runs in the cluster and is configured declaratively (with sensor and trigger CRDs). This allows the building of sophisticated declarative event-driven CI/CD systems. For instance, an Argo Events can detect when a new Docker image is published to a registry and then launch a test workflow, or when a pull request is merged, it can start a deployment workflow. With this approach, pipeline trigger logic moves into the k8s cluster defined by Custom Resources (CRs), version controlled, rather than being defined in a CI service's proprietary config.

With this approach, pipeline trigger logic moves into the k8s cluster defined by Custom Resources (CRs), version controlled, rather than being defined in a CI service's proprietary config. Along with that, an event-driven approach leads to decoupled and responsive pipelines. Instead of a periodic poll or manual trigger, pipelines run exactly when needed, making the whole system much more scalable. Declaratively defining these triggers means they are versioned and reproducible like the rest of the system config.

The goal is to treat CI/CD not as a static sequence but as a set of reactive workflows that continuously respond to code, configuration, or environment changes.

4.2. Multi-Cluster and Hybrid Cloud Delivery

With the increase in building scalable and failure-resistant software, a common practice is to deliver software to multiple kubernetes clusters, sometimes across different cloud providers or on hybrid (cloud/on-prem). Also, achieving cloud provider agnostic CI/CD means that the pipeline and GitOps process should not be tightly coupled to any cloud provider's services. To achieve this, a common strategy is to use universal tools that can run CI/CD over Kubernetes, such as Tekton pipelines for CI and Argo CD for CD. Such a strategy provides GitOps at scale without being dependent on any cloud provider. Another strategy is to build infrastructure as code (IaC) by defining cloud resources (networks, clusters, etc.) with tools like Terraform or Crossplane. Crossplane extends multi-cloud GitOps declarative pattern by defining the cloud resources using kubernetes CRDs stored in the Git repository. IaC ensures that the provisioning and configuration of cloud services are automated and versioned, supporting GitOps for application infrastructure.

One of the approaches to handle multi-cluster configuration with gitOps is to use multiple repository patterns. A single bootstrap repository is created, holding the configuration of different clusters. Such a repository is reconciled on a manager/bootstrap k8s cluster using tools like Flux and Crossplane to spawn new workload clusters or update existing clusters. Once a workload cluster is created, multiple application repositories are on-boarded by storing the application configuration in a separate operational repository, mapping the application repositories to multiple clusters. In such a way, changes in the application repository trigger reconciles in multiple clusters to deploy the latest changes to the cluster.

In summary, Kubernetes-native CI/CD tools and GitOps workflows provide the functionality for continuous deployments in any environment. By strategically laying out the git repositories, leveraging infrastructure-as-code, and using the gitOps tools described above, multi-cluster and

hybrid cloud delivery pipelines can be operationalized and maintained.

4.3. Securing CI/CD

With enhanced automation achieved by using GitOps, advancements in security policy and secret management of the cluster are necessary. Tools like (Open Policy Agent) OPA (oh-pa), and Kyverno are cloud-native policy engines that allow different policies needed in the cluster to be defined declaratively as Code. The policy controllers run continuously, intercepting changes in the cluster. These policy definitions can implement GitOps by adding another "control loop" layer over the GitOps reconciliation loop in which the GitOps operator reconciles the Git state to the cluster state. Once the Policy stored in the git repository is reconciled in the cluster, the policy controller watches the policy states, dynamically enforces the required policies, and manages configurations.

The automation provided by gitpops based on the code in the repository brings security to the git repository access policy. The automation minimizes manual access to production clusters, but a compromised repository with a weak access policy can impact multiple clusters. Some of the practices which can increase the security with GitOps principles are:

4.4. Securing Git Repository

As Git becomes the source of truth, it is very critical to restrict changes to the GitOps repository without proper authorization and validation to ensure consistency, security, and compliance across the deployment pipeline. Some of the common patterns are managing deployment branches separately, disabling changes to protected branches, using mandatory pull requests to enforce code review processes, and maintaining a clear separation of concerns in the deployment workflow.

4.5. Secret Management

Managing secrets outside of the git repository via encrypted stores or Kubernetes secret operators (like Mozilla SOPS integration with key management systems like AWS KMS, GCP KMS, Azure Key Vault or Hashicorp Vault) can improve security, maintainability, and compliance by ensuring that sensitive information is not exposed.

4.6. Least Privilege Access

The GitOps controllers (Argo CD, Flux) should be run with the minimum necessary permissions. Due to heightened control by GitOps agents running in the cluster, a compromised GitOps control plane could affect the entire cluster or even multiple clusters simultaneously. So, isolating the controller access, for example, limiting the access for a project to a specific namespace or API groups and using tools like network policies and strong authentication for web UIs/CLIs, is critical.

4.7. Audit Logs

Everything in Git gives a nice audit trail by default. Teams should leverage this by linking commits to change management or ticketing systems and regularly reviewing the Git history for unexpected changes. Leveraging signed commits and automatic pull-request labelling when changing access control policies can help audit security gaps. Some tools, like OpsMx ISD as an add-on for Argo CD, provide an audit log on what was deployed, when, and by whom.

5. Challenges and Future Direction

As CI/CD and GitOps practices become a critical components in cloud-native development, several challenges are still present and future directions that the industry is taking.

5.1. Standardization of Pipeline Definitions

A unified standard for defining pipelines is lacking with multiple CI/CD frameworks (Gitlab CI, Tekton, Argo Workflows, Jenkins pipelines, GitHub Actions). Interoperability becomes a huge challenge when changing the underlying CI/CD solution. For example, a pipeline defined for Gitlab CI must be rewritten to run on Argo and vice versa. Initiatives like the CD Foundation's CDEvents [15] are getting more traction, working on standardizing the events in CI/CD (so tools can interoperate on triggers and results). Tekton's CRDs could also serve as the lowest common denominator (since it was born with standardization in mind). It is not trivial to map all features between systems.

This is like how, in the past, Docker Compose, Kubernetes YAML, and CloudFormation all described deployments differently – eventually, Kubernetes YAML became a de facto standard for container orchestration. The current challenge is getting broad adoption without slowing down innovation. In the future, something like the Open Pipeline Format might become the standard for defining the workflow that any CI/CD engine can execute. This would significantly enhance tool interoperability and reduce switching costs.

5.2. Observability and Traceability of Pipelines

As pipelines become more complex and spread across multiple systems (CI engines, CD controllers, etc.), observing them end-to-end is quite challenging. Developers need to identify where a failure occurred quickly, and with the setup of multiple systems, developers might have to check logs in different systems. This leads to the effort of a unified pipeline observability, which includes some of the improvements like

5.3. User Experience Improvements

Giving a unified user experience is critical for developers to find which steps in the workflow failed and the ability to scan through the logs to find the reasons for the failure. Also, features like a "rerun" pipeline or "dry-run"

are helpful in case of failures. Mitigation or better diff and approval mechanisms can provide observability into what changes and when.

5.4. Pipeline Analytics

Gathering metrics on pipelines (success rates, durations, frequency) and analyzing trends is valuable for engineering productivity. Currently, the platforms do not provide analytics off the shelf. Future CI/CD platforms will likely have more built-in analytics dashboards, possibly automatically feeding into engineering DORA metrics tracking. For instance, tracking Deployment Frequency and Change Failure Rate could be automated by the GitOps CD tool and easily visualized by the Grafana dashboard. This will give engineering teams continuous feedback on their performance.

5.5. Pipeline Tracing

In the current CI/CD platforms, there is no easy way to see and analyze the flow of change from commit to production. Tracing CI/CD with OpenTelemetry (OTEL) adds traces to various steps in the pipeline, providing visibility across the entire pipeline execution by instrumenting and monitoring each pipeline step.

5.6. Integration of AI for automation

In recent years, with the advancement of various AI models, AI has been leveraged to automate parts of DevOps that require human intervention. Some of how AI is being used are discussed below.

5.6.1. Infrastructure Workload Forecasting

Some time-series forecasting models (such as ARIMA) are being used to forecast the workload in the infrastructure. [15] In such a case, the model prediction is integrated with CI pipelines and using GitOps, and the infrastructure can be updated without any manual intervention. Essentially, the system "learns" that traffic will spike and, ahead of time, commits a change to accommodate it – the GitOps pipeline then deploys this change automatically. This mix of AI (for prediction) and GitOps (for execution) can significantly reduce manual operational intervention and improve resilience. [16]

5.6.2. Anomaly Detection, Auto-Remediation and Pipeline Optimization

AI models could automatically detect anomalies in build/test logs or deployment metrics. For example, an ML model might learn the pattern of a typical deployment and flag if a deployment's log or performance profile looks unusual. The pipeline could then pause for human review. Based on anomaly detection, AI could trigger automatic remediation if a deployment or infrastructure change causes a problem. A simple form is already present (rollbacks on failed health checks). However, more advanced AI could try to pinpoint the component causing failure (maybe using knowledge of recent changes) and rectify it (for instance, by scaling up resources or clearing a stuck job). Machine learning can also be used to analyze historical data of multiple pipeline runs and code changes to suggest optimizations to pipeline runs (like test selection or caching strategies), which can improve the performance of the pipelines, for example, by only selecting tests based on the code changes.

6. Conclusion

Continuous Integration and Delivery in the cloud-native era has undergone a significant evolution. What began as a set of scripts and CI servers has transformed into declarative, Git-centered workflows operating in and alongside Kubernetes. GitOps has emerged as a key paradigm, enabling operations to be treated as code: deployments are reproducible, auditable, and self-correcting via automated reconciliation.

With the integration of AI in development tasks and the usage of Git as the source of truth for infrastructure and application state, teams achieve unprecedented transparency and control. Every change is traceable and reversible, which boosts confidence in making frequent changes. This directly contributes to improved developer velocity (deploy faster, with fewer fears) and reliability (less model drift, easier rollback).

CI/CD systems are now built as cloud-native applications themselves. This ensures they are portable, modular, and can scale horizontally. Kubernetes-native pipelines like Tekton and Argo Workflows have reimagined CI/CD, running natively on the same platform as applications and thus offering scalability and consistency that traditional external systems struggled to provide.

GitOps and Kubernetes-native CI/CD introduce their complexities. Teams need to learn Kubernetes and Git intricacies, manage new kinds of resources (CRDs for pipelines, etc.), and handle the cognitive load of distributed systems.

However, the trend is towards unification and simplification through internal developer platforms or tighter integrations between projects. The ecosystem converges on patterns that work at scale, as evidenced by the shared learnings in research and conferences. Such open unified formats, potentially becoming Open Pipeline format, would simplify pipeline interoperability across fragmented CI/CD solutions to a great extent. Leveraging GPT models to do such transformations is also a great start.

In conclusion, the evolution of CI/CD toward GitOps and Kubernetes-native workflows is a positive feedback loop: as more organizations adopt these practices, tools improve, making adoption easier.

Acknowledgements

I want to express gratitude to the team of Argo CD, Tekton, Flux CD, Jenkins, Screwdriver CD, and Rancher for providing detailed documentation, which was used extensively to write this paper.

References

[1] Florian Beetz, and Simon Harrer, "GitOps: The Evolution of DevOps?," *IEEE Software*, vol. 39, no. 4, pp. 70-75, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[2] Tekton vs. Jenkins: What's Better for CI/CD Pipelines on Red Hat OpenShift? Martin Sumner, 2023. [Online]. Available: https://www.redhat.com/en/blog/tekton-vs-jenkins-whats-better-cicd-pipelines-red-hat-openshift

[3] Yann Albou, and Sébastien Féré, GitOps and the Millefeuille Dilemma, 2020. [Online]. Available: https://www.sokube.io/en/blog/gitops-and-the-millefeuille-dilemma-en

[4] GitOps Principles. [Online]. Available: https://github.com/open-gitops/documents/blob/main/PRINCIPLES.md

[5] Screwdriver CI/CD. [Online]. Available: https://screwdriver.cd/

[6] Argo CD Docs. [Online]. Available: https://argo-cd.readthedocs.io/

[7] Argo CD vs Tekton vs Jenkins X: Finding the Right GitOps Tooling. [Online]. Available: https://platform9.com/blog/argo-cd-vs-tekton-vs-jenkins-x-finding-the-right-gitops-tooling/

[8] Flux CD Documentation. [Online]. Available: https://fluxcd.io/flux/concepts/

[9] GitOps with GitLab: What you need to know about the Flux CD Integration. [Online]. Available: https://about.gitlab.com/blog/2023/02/08/why-did-we-choose-to-integrate-fluxcd-with-gitlab/

[10] Fleet Github Repo. [Online]. Available: https://github.com/rancher/fleet

[11] Rancher Documentation. [Online]. Available: https://www.rancher.com/

[12] Tekton Documentation. [Online]. Available: https://tekton.dev/docs/

[13] Argo Workflows Documentation. [Online]. Available: https://argo-workflows.readthedocs.io/en/latest/

[14] Jenkins X. [Online]. Available: https://jenkins-x.io/

[15] CDEvents WhitePaper. [Online]. Available: https://cdevents.dev/docs/wpaper/

[16] Bohdan Fedoryshyn, and Olena Krasko, "Migration of Services in a Kubernetes Cluster Based on Workload Forecasting," *Information and Communication Technologies, Electronic Engineering*, vol. 4, no. 2, pp. 82-92, 2024. [CrossRef] [Publisher Link]