International Scientific Journal of Engineering and Management
Volume: 04 Issue: 10 | Oct– 2025
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata
ISSN: 2583-6129
DOI: 10.55041/ISJEM05063

Template-Driven AI Platforms: Using Infrastructure as Code and GitOps to Standardize and Scale AI/ML Environments

Santosh Pashikanti

Abstract

Enterprise AI and ML initiatives are often slowed down not by algorithms, but by fragmented, snowflake environments that are hard to reproduce, scale, and govern. Each new project tends to reinvent its own cloud setup, Kubernetes cluster, data access pattern, and ML toolchain, creating a long tail of operational toil and "hidden technical debt" in production systems [3]. In this paper, I present a template-driven approach for building AI platforms using Infrastructure as Code (IaC) and GitOps, with Terraform, Helm, and Kubernetes as the core building blocks. Drawing on cloud-native principles and lessons from real-world transformations, I describe how standardized templates, modular Terraform and Helm packages, and Git-centric workflows can deliver repeatable, compliant, and scalable AI/ML environments across teams, environments, and clouds. I outline the architecture of a template-driven AI platform, walk through a concrete multi-cloud implementation, and evaluate the approach along dimensions such as time-to-environment, deployment frequency, drift reduction, and compliance. I close with a discussion of trade-offs, organizational challenges, and practical recommendations for platform teams that want to industrialize AI without sacrificing flexibility for data scientists and ML engineers.

Index Terms

Infrastructure as Code, GitOps, Terraform, Helm, Kubernetes, MLOps, AI Platforms, Multi-Cloud, DevOps, Continuous Delivery.

I. Introduction

Over the last decade, AI/ML has moved from isolated innovation projects to mainstream production workloads in most large enterprises. Yet, the supporting platforms are still catching up. Many organizations are running dozens of AI use cases, each deployed on slightly different infrastructure: different VPC layouts, different Kubernetes cluster settings, inconsistent GPU configurations, ad-hoc secrets management, and one-off CI/CD pipelines. The result is a zoo of environments that are difficult to understand, nearly impossible to reproduce, and fragile to operate at scale.

In parallel, the DevOps and cloud-native communities have converged on Infrastructure as Code (IaC) as the backbone for managing modern infrastructure, and GitOps as a model for continuously reconciling runtime state from a version-controlled source of truth [1], [2]. Terraform has emerged as a dominant IaC tool for provisioning cloud resources, while Helm has become the de-facto package manager for Kubernetes applications. Together with GitOps controllers such as Argo CD and Flux, these tools offer a powerful foundation for building repeatable, self-service platforms that can span multiple clouds and regions.ACM Digital Library+1

In my experience designing cloud-native AI/ML platforms, the real unlock happens when we stop treating AI environments as bespoke projects and start treating them as products: opinionated, template-driven platforms that teams can consume in a self-service way. Instead of asking "How do I stand up a new ML environment for this project?" the question becomes "Which template do I use, and what parameters do I set?" This shift reduces cognitive load for teams, shortens time-to-first-model, and dramatically improves security, compliance, and cost visibility.

ISSN: 2583-6129
DOI: 10.55041/ISJEM05063

This paper is my attempt to codify that approach. I focus on the following contributions:

- A set of **system requirements and design principles** for template-driven AI/ML platforms built on Kubernetes and the public cloud.
- A **reference architecture** that combines Terraform modules, Helm charts, and GitOps controllers to standardize AI/ML environments.
- A **concrete implementation and case study** spanning AWS and GCP, with environment promotion and policy enforcement baked into the workflow.
- A **qualitative and quantitative evaluation** of the benefits and trade-offs of this pattern, especially for enterprises operating at scale.

The rest of this paper is organized as follows. Section II reviews related work and the current industry landscape. Section III outlines system requirements and cloud-native design principles. Section IV presents the architecture. Section V describes the implementation and case study. Section VI discusses evaluation and results. Section VII provides a broader discussion, and Section VIII concludes. Acknowledgments and references follow.

II. Related Work and Industry Landscape

The concepts behind template-driven AI platforms sit at the intersection of several mature streams of work: Infrastructure as Code, GitOps, and MLOps.

Infrastructure as Code (IaC): The IaC movement, popularized by Morris and others, advocates managing infrastructure through version-controlled code rather than manual configuration [1]. Terraform has emerged as a widely adopted, cloud-agnostic IaC tool for provisioning resources on AWS, GCP, Azure, and beyond. Terraform promotes declarative configuration, immutability, and modularization of infrastructure "stacks," which makes it particularly well-suited for multi-environment and multi-cloud patterns.O'Reilly Media+1

GitOps: Limoncelli's "GitOps: A Path to More Self-Service IT" articulated GitOps as the combination of IaC and pull-request workflows to drive operational changes [2]. In the Kubernetes ecosystem, GitOps is implemented by controllers like Argo CD and Flux that continually reconcile runtime state to the desired state in Git repositories. These tools provide drift detection, multi-cluster management, and auditable change histories, which are critical for regulated enterprises.Communications of the ACM+1

MLOps and AI Platforms: Sculley et al. highlighted the "hidden technical debt" in ML systems, including glue code, pipeline jungles, and configuration debt [3]. Breck et al. proposed the ML Test Score as a rubric for ML production readiness and technical debt reduction [4]. These works set the stage for MLOps practices focused on robust, testable ML pipelines and operational guardrails. In practice, this has led to Kubernetes-native platforms like Kubeflow, which provide an ecosystem of components for training, serving, and monitoring models on Kubernetes [5], [6].NeurIPS Proceedings+2Google Research+2

IaC + GitOps for MLOps: Recent industry blogs and open-source projects show how Terraform, Helm, and GitOps can be combined to build MLOps platforms. Cluster.dev demonstrates a GitOps flow where Terraform modules provision cloud infrastructure and Helm charts deploy platform services through GitOps controllers [5]. Spacelift and others describe step-by-step ways to integrate Terraform with GitOps workflows for consistent environment provisioning [6]. BigDataRepublic illustrates building an MLOps platform on GKE using GitOps and cloud-agnostic tooling [7].ClusterDev+2Spacelift+2

What is still missing in much of the literature is a **cohesive, template-driven architecture** specifically targeting AI/ML environments: one that aligns Terraform modules, Helm charts, GitOps controllers, and AI platform components into a repeatable pattern that scales across business units and clouds. This paper aims to fill that gap from a practitioner's perspective.

ISSN: 2583-6129
DOI: 10.55041/ISJEM05063

III. System Requirements and Cloud-Native Design Principles

When I design template-driven AI platforms, I start from the requirements and constraints that typically come from enterprise stakeholders. Across organizations, I see a recurring set of needs.

A. System Requirements

1. **Repeatability and Reproducibility:**
   - Ability to create identical AI/ML environments (network, Kubernetes, platform services, policies) from code for dev, test, staging, and production.
   - Deterministic provisioning such that the same Git commit yields the same environment, modulo cloud-provider idiosyncrasies.
2. **Multi-Environment and Multi-Cloud Support:**
   - Seamless promotion of workloads across non-prod and prod environments.
   - Option to deploy the same platform blueprint on AWS, GCP, and Azure, with cloud-specific modules hidden behind a common abstraction.
3. **Security and Compliance by Default:**
   - Enforced guardrails for network segmentation, encryption, IAM, and secrets management.
   - Policy-as-code (e.g., OPA/Gatekeeper, Kyverno) and guardrails integrated into both IaC and GitOps pipelines.
4. **Scalability and GPU-Aware Workloads:**
   - Horizontal scaling for stateless services and vertical/horizontal scaling for GPU-backed training and inference workloads.
   - Ability to attach different node pools (CPU, GPU, spot/preemptible) via templates, rather than bespoke cluster changes.
5. **Self-Service with Guardrails:**
   - Product teams can request new "AI workspaces" or "project namespaces" without filing tickets.
   - Platform team retains control through centrally managed templates and policy sets.
6. **Observability and Operational Readiness:**
   - Standardized metrics, logs, and traces for the platform and ML workloads.
   - Integrated checks against production readiness rubrics such as the ML Test Score [4].Google Research
7. **Cost Transparency:**
   - Tagging, labeling, and budget controls baked into Terraform modules and Helm values.
   - Ability to attribute costs to projects, departments, and environments.

B. Cloud-Native Design Principles

To meet these requirements, the platform architecture is guided by the following principles:

1. **Everything as Code:**
   - Infrastructure, Kubernetes objects, platform services, and even policies are expressed as YAML/HCL and stored in Git. This aligns with IaC best practices from Morris and others [1].Thoughtworks
2. **Declarative, Not Imperative:**
   - Rather than running imperative scripts, the platform describes a desired state that Terraform and GitOps controllers reconcile continuously.
3. **Modularity and Composability:**
   - Terraform modules encapsulate VPCs, Kubernetes clusters, and shared services. Helm charts encapsulate AI platform components (e.g., Kubeflow, MLflow, feature stores). Teams assemble environments by composing modules and charts.

ISSN: 2583-6129
DOI: 10.55041/ISJEM05063

4. **Separation of Concerns:**
   - Clear separation between:
     - **Cloud foundation** (accounts, networking, IAM),
     - **Platform** (Kubernetes, AI services, observability),
     - **Workloads** (models, pipelines, APIs).
5. **Immutable and Idempotent Changes:**
   - Changes are made via pull requests; environments are recreated or updated consistently from templates rather than patched manually.
6. **Git as the Single Source of Truth:**
   - All changes to infrastructure and applications flow through Git. Git history becomes the audit trail.
7. **Environment Parity with Parameterization:**
   - Templates are shared, but environment-specific differences (regions, instance types, quotas) are expressed via variables and overlays rather than forks.
8. **Progressive Delivery and Safe Rollbacks:**
   - GitOps controllers roll out changes gradually, with automated rollback on failure. This is essential when modifying critical AI platforms that multiple teams depend on.

IV. Architecture

The template-driven AI platform architecture can be thought of as layered and Git-centric. At a high level, I structure it around **four layers** and **three classes of Git repositories**.

A. Architectural Layers

1. **Cloud Foundation Layer (Terraform):**
   - Cloud accounts/projects and organizational units.
   - VPCs/VNets, subnets, routing, NAT, and private connectivity.
   - Shared identity primitives (IAM roles, service accounts, managed identities).
   - Centralized security tooling (CloudTrail/Cloud Audit Logs, GuardDuty/SCC, key management).
2. **Kubernetes Platform Layer (Terraform + Helm):**
   - Managed Kubernetes services (EKS, GKE, AKS) provisioned by Terraform modules.
   - Node pools for general compute, GPU workloads, and spot/preemptible capacity.
   - Cluster add-ons deployed via Helm and GitOps:
     - Ingress controllers (NGINX, Envoy-based gateways),
     - Service mesh (optional),
     - Observability stack (Prometheus, Grafana, Loki),
     - Policy controllers (Gatekeeper, Kyverno),
     - GitOps controller (Argo CD or Flux).
3. **AI/ML Platform Layer (Helm + GitOps):**
   - AI platform components such as Kubeflow, MLflow, feature stores, model registry, workflow engines, and API gateways. Kubeflow's modular architecture is particularly well-aligned with this layer [5], [8].Kubeflow+1
   - Deployed as Helm charts, with values files parameterized per environment and per project.
4. **Workload Layer (Helm, Kustomize, or plain YAML + GitOps):**
   - Project-specific namespaces, ML pipelines, training jobs, inference services, and batch/streaming jobs.
   - Application manifests stored in per-project Git repos; GitOps controller continuously syncs them to the target clusters.

ISSN: 2583-6129
DOI: 10.55041/ISJEM05063

B. Repository Topology

To keep things manageable at scale, I generally recommend three main classes of repositories:

1. **Infra-Foundation Repos:**
   - iac-org-foundation – Organization-level Terraform code for accounts/projects, shared networking, security, and baseline policies.
   - iac-landing-zones – Environment-scoped Terraform modules (e.g., aws-landing-zone, gcp-landing-zone) with opinionated defaults.
2. **Platform Repos:**
   - platform-k8s-clusters – Terraform modules to create and customize Kubernetes clusters (EKS/GKE/AKS), including node pools and basic add-ons.
   - platform-ai-services – Helm charts and values files for AI platform components (Kubeflow, MLflow, feature store, model registry).
   - platform-gitops-config – GitOps manifests defining which clusters track which Git paths (e.g., Argo CD Applications or Flux Kustomizations).
3. **Workload Repos:**
   - Per-team or per-product repos, such as fraud-ml-service, personalization-ml-pipelines.
   - These repos contain Helm charts, Kustomize overlays, and pipeline definitions, but **do not** carry their own infrastructure code.

Platform teams own the infra and platform repos; product teams own the workload repos. GitOps controllers run in "management clusters" and watch a combination of platform and workload repos to continuously reconcile the runtime state.ClusterDev+1

C. Template Design

The heart of the approach is in how templates are designed:

- **Terraform Modules as Environment Templates:**
  - eks_cluster_template, gke_cluster_template, aks_cluster_template modules encapsulate tagging, security groups, logging, and baseline configuration.
  - Variables control project name, environment, region, GPU capacity, and network CIDRs.
  - Outputs expose kubeconfig, ARNs, and other integration points for GitOps controllers.
- **Helm Charts and Values as Platform Templates:**
  - A base Kubeflow chart with overlays for dev, test, and prod.
  - Standardized values for resource requests/limits, node selectors for GPU pools, and integration with shared observability stacks.
- **GitOps Manifests as Synchronization Templates:**
  - For each environment, a GitOps Application (Argo CD) or Kustomization (Flux) defines which Git paths to sync to which clusters, with health checks and sync options.

Using these templates, creating a new AI environment becomes a matter of:

1. Creating a short YAML/HCL file that instantiates the relevant Terraform modules with environment-specific values.
2. Pointing GitOps controllers to the appropriate platform and workload repo paths.
3. Merging a pull request.

ISSN: 2583-6129
DOI: 10.55041/ISJEM05063

V. Implementation and Case Study

To make this concrete, I will outline a representative implementation pattern for a large enterprise deploying a template-driven AI platform on AWS and GCP.

A. Scenario

A global financial services organization is rolling out multiple AI use cases: fraud detection, credit risk scoring, personalization, and operational forecasting. Historically, each use case had its own environment, leading to duplicated infrastructure, inconsistent controls, and slow onboarding for new teams. The organization wants:

- A **standard AI platform** on EKS and GKE;
- **Self-service project environments** that comply with corporate security policies;
- **Multi-cloud** capability for portability and resilience.

B. Tooling Stack

The chosen stack is:

- **Terraform** for all cloud infrastructure.
- **Helm** for Kubernetes and AI platform components.
- **Argo CD** as the GitOps controller (Flux would be equally valid; AWS prescriptive guidance documents the trade-offs between the two [9]).AWS Documentation
- **Kubeflow** for ML pipelines, training, and serving on Kubernetes [5], [8].
- **Prometheus, Grafana, Loki** for observability.
- **OPA/Gatekeeper** for policy-as-code on Kubernetes.

All code is stored in GitHub, and CI pipelines (e.g., GitHub Actions) are used to run validation and apply steps.

C. Step-By-Step Implementation

1. **Cloud Foundation with Terraform:**

The iac-org-foundation repo defines:

- AWS Organizations and GCP Folders/Projects dedicated to AI workloads.
- Network blueprints (hub-and-spoke VPC/VNet patterns, shared services networks).
- Shared security services (logging buckets, KMS keys, security analytics).

These are applied once per region. Terraform remote state is stored in an encrypted backend (e.g., S3 + DynamoDB, GCS + Cloud KMS).

2. **Kubernetes Clusters as Templates:**

In the platform-k8s-clusters repo, the platform team defines:

- eks_cluster_template module:
  - Control plane settings (version, logging),
  - Node groups (CPU, GPU, spot),
  - Managed add-ons (CNI, CoreDNS, metrics).
- gke_cluster_template module with equivalent constructs.

ISSN: 2583-6129
DOI: 10.55041/ISJEM05063

Parameters control environment (dev, test, prod), region, node counts, and GPU sizes. Spacelift and similar platforms provide real-world examples of such Terraform patterns for Kubernetes and AI workloads [6], [10].Medium+1

For each environment, a small Terraform configuration file instantiates these modules. CI validates, plans, and then applies the changes after review.

3. **GitOps Management Cluster:**

The platform team provisions a small "management cluster" on each cloud, also via Terraform. Argo CD is deployed via a Helm chart. Argo CD is configured to:

- Watch specific Git paths in platform-ai-services and workload repos.
- Manage application manifests across EKS and GKE clusters (multi-cluster mode).Argo CD+1

4. **AI Platform Services via Helm + GitOps:**

In platform-ai-services, the team defines:

- A kubeflow Helm chart (or reference to upstream) with:
  - Profiles, Notebook Servers, Pipelines, KFServing (or KServe), and Katib.
  - Values files for dev/test/prod with different quotas and resource limits.
- Charts for:
  - MLflow server + backend store,
  - Feature store (e.g., Feast),
  - Model registry,
  - Shared libraries and sidecars.

Argo CD Applications map each environment's cluster to the appropriate Helm release and values files. The result: every EKS and GKE environment receives the same AI platform layout, tuned by environment.

5. **Workload Onboarding via Templates:**

When a new ML team wants to onboard, they:

- Request a **project namespace** (e.g., via a service catalog or simple pull request template).
- The platform team (or automated process) creates:
  - A new namespace,
  - Resource quotas and limit ranges,
  - RBAC bindings,
  - A dedicated Git repo with a starter template (Helm chart, pipeline examples).

In the GitOps config, a new Argo CD Application is added that maps this namespace to the team's repo path. The team starts by customizing values files to define their training jobs, batch pipelines, and online inference services.

6. **Policies and Guardrails:**

OPA/Gatekeeper or Kyverno policies enforce:

- No public LoadBalancers by default,
- Mandatory labels/tags for cost allocation,
- Required encryption for data volumes,

ISSN: 2583-6129
DOI: 10.55041/ISJEM05063

- Resource limits for pods.

These policies are treated as code and packaged as part of the platform Helm charts, versioned in Git, and applied via GitOps.bigdatarepublic.nl

7. **Observability and Production Readiness:**

Standard dashboards and alerts are deployed for:

- Cluster health and resource usage,
- Pipeline success/failure rates,
- Model serving latency and error rates.

In addition, the platform introduces a simple **"ML Readiness Checklist"** inspired by the ML Test Score [4], spanning data quality checks, monitoring, rollback strategies, and dependency management. Teams must pass a threshold score before their workloads are promoted to production.

VI. Evaluation and Results

To evaluate the template-driven AI platform, I focus on both **quantitative** and **qualitative** dimensions. The numbers below are representative but align with what I have seen in practice when organizations move from ad-hoc environments to IaC + GitOps-based platforms.

A. Time-to-Environment

Before adopting template-driven IaC, standing up a new AI environment often involved weeks of coordination between infrastructure, security, networking, and data teams. After standardizing on Terraform modules and GitOps templates:

- Provisioning a new EKS/GKE-based AI environment became a matter of:
  - Cloning an environment config file,
  - Adjusting parameters (region, capacity),
  - Running a Terraform pipeline,
  - Merging a GitOps config change.

Typical time-to-environment dropped from **4–6 weeks** to **1–3 days**, with most of the delay being change-management approvals rather than technical work.

B. Deployment Frequency and Lead Time

With Git and GitOps as the control plane:

- Platform teams began shipping platform updates (e.g., new Kubeflow version, policy tweaks) multiple times per week, instead of quarterly.
- Product teams adopted feature branch + PR workflows for model and pipeline changes, and could deploy new versions multiple times per day if needed.

This aligns with industry findings that GitOps and IaC significantly reduce lead time for changes by eliminating manual handoffs [2], [5].Communications of the ACM+1

ISSN: 2583-6129
DOI: 10.55041/ISJEM05063

C. Drift Reduction and Operational Stability

Pre-GitOps, clusters frequently drifted from their documented configuration due to hotfixes and manual kubectl changes. After adopting Argo CD:

- Drift was detected automatically; out-of-band changes were surfaced in the UI and reverted or codified in Git.
- Incident investigations became faster because the exact configuration at any point in time could be reconstructed from Git history.

This reduced the frequency of environment-related incidents (e.g., missing sidecars, misconfigured ingress, disabled metrics) and created more predictable rollout behavior.

D. Compliance and Auditability

Regulated enterprises care deeply about "Who changed what, when, and why?" Template-driven platforms improved this in several ways:

- All infrastructure and platform changes were captured as Git commits, with associated tickets and approvals.
- Policy-as-code ensured that non-compliant changes were rejected at review time or blocked at admission control.
- Audit teams could trace production changes back to specific PRs, authors, and reviews.

These practices echo the motivations behind GitOps and the ML Test Score, both of which emphasize testability, monitoring, and traceability as key to reducing ML technical debt [2], [4].Google Research+1

E. Developer and Data Scientist Experience

From a user perspective, the biggest impact was at the edges:

- Data scientists received **pre-configured workspaces** (Kubeflow notebooks, pipelines, MLflow tracking) without having to understand VPCs or IAM in depth.
- Platform teams could **say "yes" more often** because onboarding a new project was a parameter change, not a multi-week infrastructure effort.
- Teams had clear expectations around production readiness via the standardized checklist.

The main complaint was the "learning curve" of working with Terraform and GitOps workflows, especially for teams used to unmanaged notebooks or script-driven deployments. However, with good documentation and training, this settled into a productive equilibrium.

VII. Discussion

While the benefits are significant, template-driven AI platforms are not a silver bullet. There are clear trade-offs and failure modes.

A. Balancing Standardization and Flexibility

Too much standardization can turn the platform into a rigid monolith that cannot accommodate legitimate variations (e.g., specialized GPU types, regional compliance nuances, or alternative model serving stacks). The key is to design templates with **extension points**:

ISSN: 2583-6129
DOI: 10.55041/ISJEM05063

- Allow teams to plug in additional Helm charts into their namespaces.
- Offer "golden paths" but not a single path; for example, support both Kubeflow Pipelines and Argo Workflows where justified.
- Maintain a backlog for platform evolution driven by real product needs.

B. Managing Template Sprawl

As more templates are added (for different clouds, regions, or platform variants), there is a risk of combinatorial explosion. I have found the following patterns useful:

- Establish a **small set of canonical templates** (e.g., "GPU-heavy training environment", "balanced training + serving environment") and discourage forks.
- Use **parameterized overlays** rather than copying and modifying whole templates.
- Periodically **refactor and deprecate** older templates based on usage.

C. Organizational and Cultural Shifts

The hardest part is often not technical:

- Platform teams must embrace a **product mindset**, treating templates as products with versioning, SLAs, and roadmaps.
- Data science and ML engineering teams need to build muscle around Git workflows, code reviews, and CI/CD.
- Governance bodies (security, risk, compliance) must align policies with the platform capabilities, ideally codifying rules as policy-as-code.

Without these cultural shifts, even the best-designed templates will degrade into one-off overrides and manual edits over time.

D. Technology Choices and Lock-In

This paper describes a stack centered on Terraform, Helm, and Argo CD, but the underlying principles are tool-agnostic:

- Terraform could be swapped for OpenTofu or another IaC engine, provided it supports multi-cloud and modular stacks.
- Helm could be complemented with Kustomize or templating frameworks.
- Argo CD could be replaced with Flux or a managed GitOps service.

What matters is the **pattern**: desired state in Git, declarative reconciliation, and clear layering between foundation, platform, and workloads.

VIII. Conclusion

AI is no longer a side project; it is becoming a core part of enterprise digital transformation. As that happens, the way we build and operate AI platforms must mature. Manual, one-off environments are simply not sustainable when dozens or hundreds of teams depend on AI infrastructure every day.

In this paper, I have outlined how Terraform, Helm, and GitOps can be combined into a **template-driven AI platform** that delivers repeatable, compliant, and scalable AI/ML environments. By codifying cloud foundation, Kubernetes clusters, AI platform services, and workloads as modular templates, and by using Git as the control plane, organizations

ISSN: 2583-6129
DOI: 10.55041/ISJEM05063

can dramatically reduce time-to-environment, improve operational stability, and provide a better experience for both platform teams and data scientists.

Equally important, this approach shines a light on the hidden technical debt that traditionally accumulates in ML systems and creates a path to paying that debt down through automation, testing, and clear ownership. While each organization will tailor the details to its own constraints, I believe the core pattern **template-driven, IaC-first, GitOps-native AI platforms** will remain foundational for running AI at scale in the cloud.

References

[1] K. Morris, *Infrastructure as Code: Managing Servers in the Cloud*. Sebastopol, CA, USA: O'Reilly Media, 2016.ACM Digital Library+1

[2] T. A. Limoncelli, "GitOps: A path to more self-service IT," *ACM Queue*, vol. 16, no. 3, pp. 20–39, Sep. 2018.ACM Digital Library+1

[3] D. Sculley *et al.*, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, 2015, pp. 2503–2511.NeurIPS Proceedings+1

[4] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ML test score: A rubric for ML production readiness and technical debt reduction," in *Proc. IEEE Int. Conf. Big Data*, 2017, pp. 1123–1132.Google Research+1

[5] V. Tsap, "GitOps for Terraform and Helm with Cluster.dev," Cluster.dev Blog, Nov. 29, 2023. [Online]. Available: cluster.dev/blog/gitops-for-terraform-and-helm-with-cluster-dev/ClusterDev

[6] J. Grześkowiak, "How to use Terraform with GitOps [Step-by-step tutorial]," Spacelift Blog, Jul. 23, 2024. [Online]. Available: spacelift.io/blog/terraform-gitopsSpacelift

[7] J. van der Lugt, "Essentials of an MLOps platform – Part 2: Infrastructure," BigDataRepublic, Dec. 14, 2022. [Online]. Available: bigdatarepublic.nl/articles/essentials-of-an-mlops-platform-part-2-infrastructure/bigdatarepublic.nl

[8] S. Kaul, "Introduction to Kubeflow: MLOps," *Medium*, Jul. 24, 2024. [Online]. Available: medium.comMedium

[9] Amazon Web Services, "Argo CD and Flux use cases – AWS prescriptive guidance," AWS Docs. [Online]. Available: docs.aws.amazon.com/prescriptive-guidance/latest/eks-gitops-tools/use-cases.htmlAWS Documentation

[10] H. Hirsi, "MLOps platform – end-to-end DevOps setup, data engineering, and machine learning," GitHub repository, accessed Apr. 2024. [Online]. Available: github.com/hamidhirsi/mlops-platformGitHub

[11] "Argo CD – Declarative GitOps CD for Kubernetes," Argo CD Documentation, accessed Apr. 2024. [Online]. Available: argo-cd.readthedocs.ioArgo CD

[12] NetApp, "Advancing AI projects with open-source MLOps tools," *Tech ONTAP Blog*, Jun. 27, 2024. [Online]. Available: community.netapp.comcommunity.netapp.com