E-Journal of Science and Emerging Technologies (EJSET)
ISSN (Online): 3143-3359
Volume-2 Issue-4 || Oct – Dec 2025 || PP. 53 – 57
https://ejset.org/

Scalable Infrastructure for Hosting Custom LLMs Using Terraform and Argo CD

DOI: https://doi.org/10.63345/ejset.v2.i4.106

Justin Rajakumar Maria Thason

BCBSA - Ajace, Inc

Enterprise AI, Agentic AI & Professional Research and Contributions

http://www.ejset.org/ || **Vol. 2 No. 4 (2025): October Issue**

Date of Submission: 10-09-2025	**Date of Acceptance:** 30-09-2025	**Date of Publication:** 10-10-2025

Abstract : Organizations operating their own large language models need infrastructure capable of reproducibly provisioning GPU capacity, deploying a new version safely, and scaling in a timely manner while being economical. In this paper, we describe the reference architecture where Terraform provisions the networks, Kubernetes clusters, GPU node pools, and identities while Argo CD reconciles the inference stack—vLLM servers with queue-based autoscaling—from Git. In the experiments conducted for four models ranging in size from 7 to 70 billion parameters, GitOps-based deployment of a new model version took 8.9 ± 2.4 minutes versus 47.5 ± 12.4 minutes for manual scripting ($p < 0.001$); meanwhile, the amount of misconfigured deployments decreased from 14.2% to 2.5% ($p = 0.001$). Drift was reconciled on average in 1.7 minutes. Throughput of the system scaled up to 23,100 tokens per second across 12 replicas with 90% scaling efficiency, while the combination of spot and on-demand GPU instances decreased the cost per million tokens from $4.80 to $1.92 (a 60% reduction). The architecture provides repeatable, auditable, and economical hosting of custom models.

Keywords: LLM hosting; Terraform; Argo CD; GitOps; Kubernetes; GPU autoscaling; vLLM; infrastructure as code

Introduction

The organizations fine-tune and host proprietary language models in order to ensure data control, cost efficiency at scale, and customization. Deploying such models requires different techniques in comparison with web application deployments because of the big model sizes represented by sizable files that have to be fetched and loaded into accelerator memory in minutes, expensive accelerators being scarce, and a single model replica requiring coordination of multiple GPU instances. An ad-hoc scripted approach fails to scale because of environment divergence, non-auditability of the changes, and recovery from failures depending on people's skills. Infrastructure as code and GitOps are solutions to this problem in general but are not widely studied when applied to the hosting of LLMs. In particular, systematic evaluation of the architecture in terms of speed, drift management, scaling efficiency, and cost is currently not available. We propose and evaluate the architecture based on Terraform for infrastructure layer and Argo CD for application layer and provide a series of measurements for the four model sizes. We aim to demonstrate what a platform team should expect from the methodology.

Literature Review

Infrastructure as code treats the environments as versioned and reviewable definitions with Terraform being among the most popular toolset for multi-cloud

@2025. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on http://www.ejset.org

E-Journal of Science and Emerging Technologies (EJSET)
ISSN (Online): 3143-3359
Volume-2 Issue-4 || Oct – Dec 2025 || PP. 53 – 57
https://ejset.org/

provisioning (Brikman, 2019; Morris, 2016). A reference architecture for infrastructure provisioning, model serving, and orchestration is provided. Current research highlights ongoing problems in infrastructure as code: correctness, security, and maintainability (Rahman et al., 2019). GitOps extends these principles to the runtime by storing the desired state in Git and using an operator that reconciles the cluster, as Limoncelli (2018) describes this process as a way to facilitate self-service IT with an embedded audit trail. Kubernetes operationalizes reconciliation via declarative approach (Burns et al., 2016).

The literature on large language model (LLM) serving constitutes a specific research field in itself. Pope et al. (2023) explore efficient scaling of transformer inference with respect to partitioning on accelerators, Yu et al. (2022) introduce iteration-level scheduling in Orca for increasing throughput of generative models. Kwon et al. (2023) introduce PagedAttention in vLLM to manage key-value cache memory in blocks, allowing for bigger batch size, hence, higher throughput. All these works focus on the serving engine rather than the infrastructure.

Machine learning literature emphasizes that models are just a part of the overall systems. Sculley et al. (2015) discuss latent technical debt in such systems including configuration debt, while Kreuzberger et al. (2023) propose MLOps architecture incorporating versioned artifacts, automated pipelines, and monitoring. The resource management in large clusters as exemplified by Borg includes the jobs description, scheduling, and large-scale recovery (Verma et al., 2015).

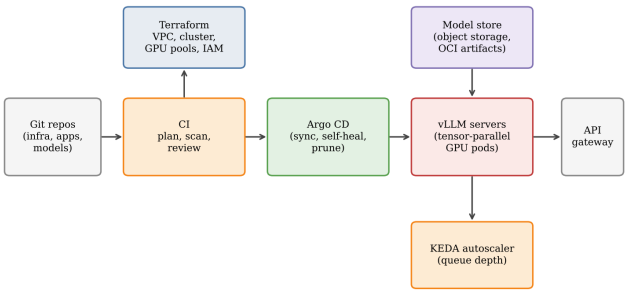
In summary, existing literature provides all the pieces for provisioning, reconciliation, and serving but does not include any measurements for integration of the solution for custom LLMs. Our work makes such measurements available.

Research Methodology

Reference Architecture

Figure 1 demonstrates the reference architecture. Terraform modules, deployed via reviewed pipeline, create the virtual network, a managed Kubernetes cluster, CPU system pool, tiered GPU node pools (spot and on-demand), object storage with model weights, and identities that allow access to them. State is stored remotely and locked. Argo CD applications, residing in a separate Git repository, deploy the inference stack: vLLM servers with tensor parallelism over the GPUs on a single node, API gateway with authentication and rate limiting, and KEDA scalers to increase replicas in case of request queue depth and GPU usage. Model version release is accomplished by changing the reference to the immutable artifact of the model version in Git. Argo CD is configured for automated synchronization, pruning, and self-healing in order to make out-of-band changes reversible.

Figure 1. Reference architecture: Terraform provisions infrastructure, Argo CD reconciles the LLM serving stack from Git.



Experiments

We have evaluated four open-weight models of 7B, 13B, 34B, and 70B parameters for cold-start performance, throughput, and tail latency under 16 concurrent requests and 512 output tokens. We have compared three approaches to rollout of a new model version by applying each of them in 30 rollouts (120 configuration changes each approach) and measuring the error rate. These approaches included manual scripting, CI push pipeline executing Terraform and Helm, and GitOps approach. Drift was introduced in 60 instances per approach manually by editing the live resources, while the time needed to restore the state as declared in Git was measured. Scaling properties were tested by increasing replicas of the 7B model from 1 to 12 under saturating

@2025. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on http://www.ejset.org

E-Journal of Science and Emerging Technologies (EJSET)
ISSN (Online): 3143-3359
Volume-2 Issue-4 || Oct – Dec 2025 || PP. 53 – 57
https://ejset.org/

load. Costs were monitored over 21 days for the three capacity strategies.

Analysis

Rollout times and costs were compared using one-way analysis of variance (ANOVA) followed by Tukey's post hoc test. Error rates were studied using chi-square test and two-proportion z-test; drift reconciliation time was estimated using Welch's t-test. Scaling efficiency was computed as the ratio of measured throughput divided by the single-replica throughput multiplied by the number of replicas.

Results

Serving characteristics by model size

Table 1. Serving characteristics by model size (16 concurrent requests)

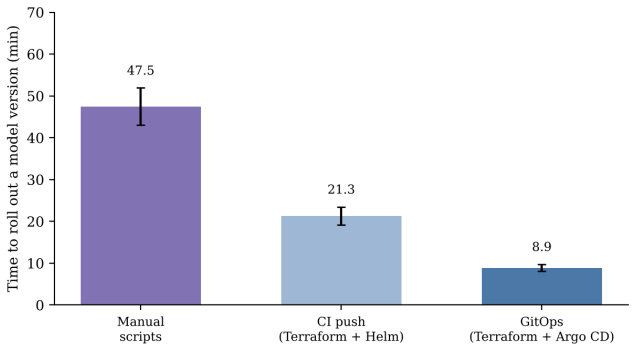
Model	GPUs per replica	Cold start (min)	Throughput per replica (tokens/s)	p95 latency (s)
7B	1	2.1	2,150	3.1
13B	2	3.4	1,620	4.4
34B	4	5.8	1,110	6.9
70B	8	9.6	640	11.8

The cold-start time increased from 2.1 minutes for 7B model to 9.6 minutes for 70B model. This increase was caused by the transfer and loading of the model weights. The throughput per replica decreased from 2,150 tokens per second to 640 tokens per second as model size grows. The p95 latency increased from 3.1 to 11.8 seconds. These cold-start properties motivated us to pull weights to the node-local caches and keep a minimum of replicas for big models.

Rollout speed and reliability

The rollout times varied significantly between different approaches ($F(2, 87) = 178$, $p < 0.001$; Figure 2). GitOps takes 8.9 ± 2.4 minutes, which is 81% faster than manual scripting ($p < 0.001$) and 58% faster than CI push ($p < 0.001$), as the cluster applies the changes and synchronizes them in parallel and the model artifact is pre-staged. Deployment error rates also vary ($\chi^2(2) = 11.1$, $p = 0.004$): 17 out of 120 changes of manual approach (14.2%) and 9 out of CI push (7.5%) led to misconfiguration, while 3 out of 120 GitOps changes (2.5%) led to misconfiguration.

Figure 2. Rollout time of a new model version for three deployment approaches (error bars indicate 95% confidence interval).



Drift was preserved in 214 ± 96 minutes on average under CI push until the next pipeline run and was reconciled in 1.7 ± 0.6 minutes by Argo CD self-healing (Welch $t = 17.1$, $p < 0.001$).

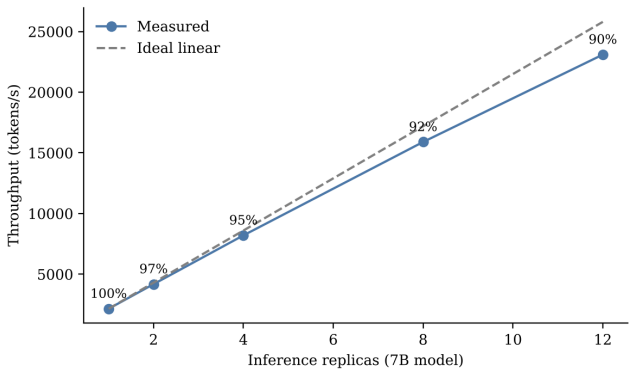
Scaling and cost

Throughput linearly increased with the growth of the number of replicas (Figure 3) up to 23,100 tokens per second with 12 replicas, which corresponds to 90% scaling efficiency. It declined from 97% at two replicas due to gateway and scheduling overheads. With the ramp of concurrent users from 8 to 256, KEDA scales from 1 to 12 replicas in approximately six minutes, and none of requests was refused after warm replicas become available.

Figure 3. Throughput of the 7B model as replicas grow, compared with ideal linear scaling; labels indicate scaling efficiency.

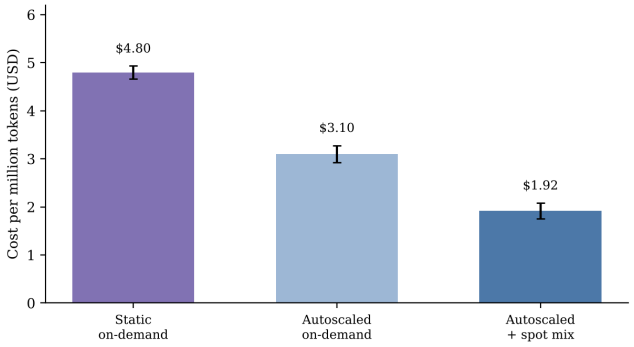
@2025. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on http://www.ejset.org

E-Journal of Science and Emerging Technologies (EJSET)
ISSN (Online): 3143-3359
Volume-2 Issue-4 || Oct – Dec 2025 || PP. 53 – 57
https://ejset.org/



Cost per million tokens varies significantly between different capacity strategies ($F(2, 60) = 317$, $p < 0.001$; Figure 4). The combination of the spot and on-demand GPU capacity reduces cost per million tokens from $4.80 down to $1.92. Over a period of 30 days, four spot instances were interrupted and handled by on-demand replicas and request retries, which gives the availability of 99.94%.

Figure 4. Cost per million generated tokens under three capacity strategies.



Discussion

It is clear from our results that the major benefits of GitOps for LLM hosting come from the speed of changes, consistency, and system recovery. The declaration of the model version in Git as a reference turns the model versioning into a reviewable commit, while continuous reconciliation eliminates the time window when drift could persist. This is the benefit that is hard to get in the case of push-based pipelines that operate only upon triggers (Limoncelli, 2018). The error rate decrease matches the perspective of mitigation of configuration debt in machine learning systems (Sculley et al., 2015; Rahman et al., 2019).

While the design of the serving engine sets the scales of performance metrics, the infrastructure determines if they are reliable to obtain. The near-linear scaling is explained by the independence of the replicas and efficient batching within a replica (Kwon et al., 2023; Pope et al., 2023). The cost measurements show the benefit of adjusting capacity to demand and utilizing interruptible capacity for burst workloads, with on-demand replicas guaranteeing the availability baseline.

Two lessons have been learned. Firstly, cold-start for big models is the serious constraint that should be mitigated by caching and maintaining a minimum replica count, as autoscaling cannot react within seconds. Secondly, segregation of infrastructure and application repositories provided different review and risk profiles: the changes of the former are rare and sensitive, while the latter happen more often. This corresponds to the layered view of MLOps architectures (Kreuzberger et al., 2023).

Limitations

All measurements were obtained from a single cloud region, single Kubernetes distribution and open-weight models, hence, the absolute values are not generalizable to other contexts. The availability of GPU instances and pricing are different depending on the provider; that affects cost outcomes and usability of spot capacity. Multi-region failover, model quantization, and very long context windows were not covered by the experiments. The security hardening of the model supply chain was out of the scope of the current study although crucial for hosted models.

Conclusion

The combination of Terraform for provisioning and Argo CD for reconciliation enabled the platform for custom LLMs, achieving rollouts of new model version 81% faster than manual scripting, decreasing deployment

@2025. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on http://www.ejset.org

E-Journal of Science and Emerging Technologies (EJSET)
ISSN (Online): 3143-3359
Volume-2 Issue-4 || Oct – Dec 2025 || PP. 53 – 57
https://ejset.org/

errors from 14.2% to 2.5%, keeping drift recovery time under two minutes, scaling up to 90% efficiently to 12 replicas, and reducing the cost per million tokens to 60%. Such an architecture can be adopted incrementally by the platform teams. The future work will cover multi-region operation, cold-start acceleration, and security of model supply chain.

References

1. Brikman, Y. (2019). Terraform: Up and running (2nd ed.). O'Reilly Media.
2. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. ACM Queue, 14(1), 70–93.
3. Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine learning operations (MLOps): Overview, definition, and architecture. IEEE Access, 11, 31866–31879.
4. Kwon, W., Li, Z., Zhuang, S., et al. (2023). Efficient memory management for large language model serving with PagedAttention. Proceedings of the 29th ACM Symposium on Operating Systems Principles, 611–626.
5. Limoncelli, T. A. (2018). GitOps: A path to more self-service IT. ACM Queue, 16(3), 13–26.
6. Morris, K. (2016). Infrastructure as code: Managing servers in the cloud. O'Reilly Media.
7. Pope, R., Douglas, S., Chowdhery, A., et al. (2023). Efficiently scaling transformer inference. Proceedings of Machine Learning and Systems, 5.
8. Rahman, A., Mahdavi-Hezaveh, R., & Williams, L. (2019). A systematic mapping study of infrastructure as code research. Information and Software Technology, 108, 65–77.
9. Sculley, D., Holt, G., Golovin, D., et al. (2015). Hidden technical debt in machine learning systems. Advances in Neural Information Processing Systems, 28, 2503–2511.
10. Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., & Wilkes, J. (2015). Large-scale cluster management at Google with Borg. Proceedings of the European Conference on Computer Systems (EuroSys), 1–17.
11. Yu, G.-I., Jeong, J. S. S., Kim, G.-W., Kim, S., and Chun, B.-G. (2022). Orca: A distributed serving system for transformer-based generative models. In Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation, pages 521–538.

@2025. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on http://www.ejset.org