JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

AUTOMATED KUBERNETES DEPLOYMENT WITH ARGOCD AND ARGO ROLLOUTS: A GITOPS-BASED APPROACH

[1]M.Pooja [2]S.Miruthula [3]M.Bhanupriya
[1, 2] Department of Computer and Science Engineering
[1, 2] Hindusthan Institute of Technology, Coimbatore, Tamil Nadu, India

ABSTRACT

The deployment methods of cloud-native applications need to be both reliable and efficient. Older methods of deploying applications such as rolling updates do not offer the specified control and rigidity newer applications need. By applying the GitOps model with tools like Argo CD and Argo Rollouts, continuous deployment is improved. Argo CD ensures that the application is in a desired state and managed through a Git-based declarative system, enabling deployment consistency across different environments [1]. Also, Argo Rollouts enables progressive delivery with active monitoring and automated rollback for advanced deployment strategies like blue-green and canary deployments [2]. This paper discusses the implementation of these tools and highlights their advantages over rudimentary methods, and potential in redefining Kubernetes deployments systems.

Keywords: Cloud-native applications, GitOps, Argo CD, Continuous deployment, Kubernetes deployment

INTRODUCTION

The integration of microservices and containerization has increased the demand for sophisticated deployment techniques for Kubernetes systems. Older methods marking an update as rolling where done incrementally lack the control that is required when dealing with systematic restrictions and minimal downtime [3]. Problems associated with complex systems have been solved with the introduction of GitOps. An example is a branch of GitOps that deals with continuous delivery of Kubernetes applications known as Argo CD. It implements declarative automation to application deployment and is able to maintain the application state in sync with the canonical Git repository. This gives the user the live and Git states under control and unifies their definitions each in one location [1]. In addition, Argo Rollouts extends the Argo framework by enabling blue-green and canary deployment strategies for more controlled and gradual deployment [2]. Enabling these observable automated deployment techniques shift the focus of developers and DevOps teams to maintaining availability of the applications and safely rolling out changes and new features [5][6].

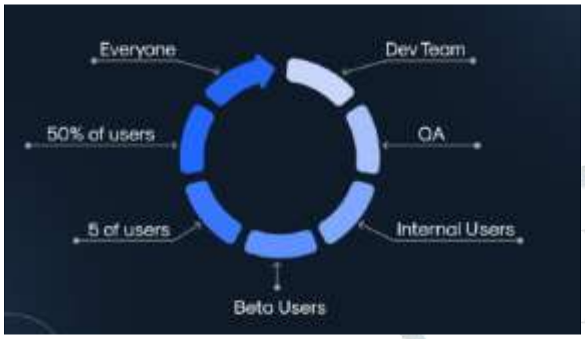


Figure 1: Improved Delivery

CURRENT SYSTEM AND ITS LIMITATIONS

Like most of the modern container orchestration systems, Kubernetes supports and provides well documented rolling updates and recreate strategies for deployment. While these techniques work, they do not provide any control over production rollouts. Updating all the pods one after the other (incrementally) or terminating all the old pods before creating new ones (recreate) are the only two options. Both of these options pose the risk of disruption on service if the new version fails [7]. In addition, reverting back to the previous version is known to take a lot of time and be prone to lethal mistakes [8]. Usually, there is not enough integration with user impact metrics which makes monitoring during the deployment really difficult [9]. This leads to the reduction of confidence to operational cycles while exposing potential operational hazards in intricate and critical availability environments [10].

OBJECTIVES:

- To set up a declarative application delivery through GitOps with Argo CD for rigid application delivery.
- To implement canary, blue-green, and A/B testing facilitation through integration of Argo Rollouts for sophisticated strategies.
- To decentralize the entire CI/CD workflow from committing code to deploying on Kubernetes, adopting contemporary DevOps technologies.
- To mitigate the risk of application downtimes and rollbacks by progressive and monitored deployments.
- To improve deployment observability through the addition of Prometheus and Grafana for enhanced monitoring.
- To demonstrate in contrast the proposed system against the traditional methods of deploying onto Kubernetes and showcase the advantages.
- To define reproducible and scalable workflows for production use around the setup sandboxed environments.
- To describe future work like anomaly detection using AI and microservice tracing on a per level basis.

PROPOSED SYSTEM AND ADVANTAGES:

The set of methods combines Argo CD and Argo Rollouts as a set into the existing deployment pipeline of Kubernetes under GitOps based on the flow described above. Assuming the existence of the Git repository as the single source of truth allows stating that all deployments are tracked, versioned, and recoverable at any point in time [1][4]. Argo CD executes a continuous comparison of the live Kubernetes environment as captured in the ecosystem and checks it against the mirror set up on Git, Agra Rollouts then implements additional deployment strategies of blue-green, canary, A/B testing, and shadow deployments [2][11]. These functionalities allow for controlled rollouts that can be reversed automatically upon failure. These functionalitites alongside rollback capabilities reduce rsik and operational downtime signficantly. Moreover, integration of observability tools like Prometheus or Grafana permit scaling, and rollback to be executed on metrics [12] allowing for a data driven and transparent deployment lifecycle.

METHODOLOGY

THE IMPLEMENTATION PROCESS CONSIST OF THE FOLLOWING:

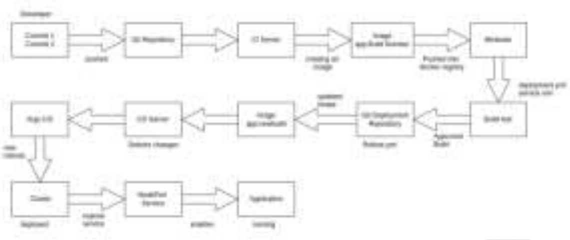


Figure 2: Implementation Steps

Developer Commits to Git Repository:

The developer pushes an updated code to a Git repository which gets picked by a CI server such as Jenkins or GitHub Actions [13].

Image Build and Push:

A Docker image is created by the CI server and tagged with a build number that is pushed into a Docker Registry [14].

Deployment to Minikube Using K8s Manifests: The image is deployed into a Minikube instance using Kubernetes manifest files (deployment.yaml, service.yaml) [15].

Testing and Approval Cycle:

A tester will check a build for functionality, If it is found to be not functional the developer will need to recommit the repo – if it's approved the deployment configuration will be updated [16].

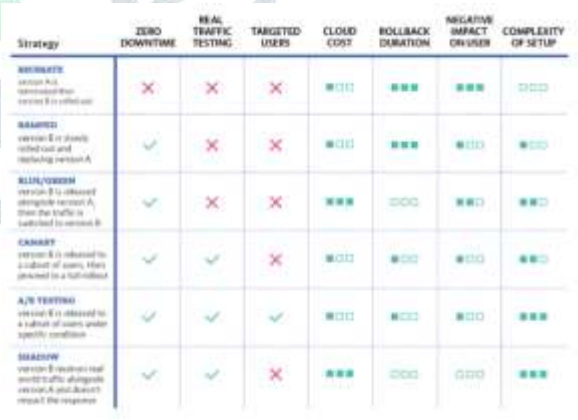
GitOps Repository Management:

The deployment configuration which also includes rollouts.yaml will be added into a distinct GitOps repository [1].

CD Server and ArgoCD Integration:

The Continuous Delivery server (ArgoCD) listens for changes in rollouts.yaml files to validate strategies along with image versions [2].

Rollout Strategy Execution:



Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE version A is terminated then version B is rolled out	✗	✗	✗	■□□	■■■	■■■	□□□
RAMPED version B is slowly rolled out and replacing version A	✓	✗	✗	■□□	■■■	■□□	■□□
BLUE/GREEN version B is released alongside version A, then the traffic is switched to version B	✓	✗	✗	■■■	□□□	■■□	■■□
CANARY version B is released to a subset of users, then proceed to a full rollout	✓	✓	✗	■□□	■□□	■□□	■■□
A/B TESTING version B is released to a subset of users under specific condition	✓	✓	✓	■□□	■□□	■□□	■■■
SHADOW version B receives real world traffic alongside version A and doesn't impact the response	✓	✓	✗	■■■	□□□	□□□	■■■

Figure 3: Kubernetes Strategies

Based on the set configuration, Argo Rollouts implements the corresponding rollout strategy—caniary or blue green—with full traffic shifting and monitoring [2][17].

Monitoring and Rollback:

Integration with observability platforms enables the capturing of key metrics during the rollout. Once certain thresholds are crossed, rollbacks are automatically enabled by Argo Rollouts

[18].

This approach guarantees full cycle operational transparency, automation, as well as deployment lifecycle resiliency throughout the entire system.

CONCLUSION

In summary, automated deployment strategies to Kubernetes offer sophisticated methods for deploying applications with minimal service interruption, reduced risk, and flexible rollback options [7]. Strategies such as Recreate, Rolling Updates, Blue-Green, Canary, A/B Testing, and Shadow Deployment methodologies add strategic value based on their specific use cases and business requirements [11][17]. The development of Argo CD and Argo Rollouts has facilitated the use of continuous delivery based on GitOps principles [1][2]. Future work may consider the incorporation of AI for anomaly detection during rollouts that could automatically suspend alters, or change the parameters of the deployment strategies based on behavioral metrics. [19].

REFERENCES

1. Codefresh. (n.d.). Argo CD: Kubernetes GitOps Continuous Delivery. Retrieved from https://codefresh.io
2. Argo Rollouts. (n.d.). Advanced Deployment Strategies for Kubernetes. Retrieved from https://argoproj.github.io/argo-rollouts
3. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. Communications of the ACM, 59(5), 50-57.
4. Weaveworks. (2021). GitOps - The Path to Modern DevOps. Retrieved from https://www.weave.works
5. Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps Handbook. IT Revolution.
6. Fowler, M. (2013). Blue-Green Deployment. MartinFowler.com.
7. Hightower, K., Burns, B., & Beda, J. (2017). Kubernetes: Up and Running. O'Reilly Media.
8. Red Hat. (2020). Kubernetes Deployment Strategies. Retrieved from https://redhat.com
9. HashiCorp. (2021). Deploying Microservices with Confidence. Retrieved from https://hashicorp.com
10. CNCF. (2022). State of Kubernetes 2022. Cloud Native Computing Foundation.
11. Google Cloud. (2020). Progressive Delivery with Spinnaker and Argo. Retrieved from https://cloud.google.com
12. Grafana Labs. (2022). Observability for Kubernetes Rollouts. Retrieved from https://grafana.com
13. Jenkins. (2021). CI/CD for Kubernetes with Jenkins. Retrieved from https://www.jenkins.io
14. Docker. (2022). Docker CI/CD Best Practices. Retrieved from https://docs.docker.com
15. Kubernetes.io. (2023). Deploy Applications with YAML. Retrieved from https://kubernetes.io
16. GitHub Docs. (2021). Automating Workflows with GitHub Actions. Retrieved from https://docs.github.com
17. Amazon Web Services. (2022). Safe Deployments with Canary Releases on EKS. Retrieved from https://aws.amazon.com

18. Prometheus. (2023). Monitoring Kubernetes Deployments. Retrieved from https://prometheus.io

19. Datadog. (2021). AI-Powered Deployment Metrics. Retrieved from https://www.datadoghq.com

20. Jaeger Tracing. (2022). Distributed Tracing for Microservices. Retrieved from https://www.jaegertracing.io