INTERNATIONAL JOURNAL OF LATEST TECHNOLOGY IN ENGINEERING, MANAGEMENT & APPLIED SCIENCE (IJLTEMAS)

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IV, April 2025

Modernizing Monolithic Applications with Gitops Orchestrated Microservices Using Argocd, IAM, and AWS Infrastructure

Ravi Chandra Thota

Independent Researcher

DOI : https://doi.org/10.51583/IJLTEMAS.2025.140400011

Received: 15 April 2025; Accepted: 16 April 2025; Published: 29 April 2025

Abstract: Modern companies need to transform their legacy monolithic applications into microservices as their essential approach for obtaining better scalability and flexibility and enhanced performance in cloud-native infrastructure. The text examines the process of changing monolithic frameworks into ArgoCD-managed microservices through a combination of IAM and AWS infrastructure. The study evaluates the advantages of this migration process to show how these systems combine their operations for managing deployment workflows while enhancing security measures and scaling cloud environments. The first section reviews literature that details the architectural change from monolithic systems to microservices while tracing the industry need for microservices in contemporary software development. The methodology section explains how GitOps works with ArgoCD for automatic microservice management and AWS IAM integration for security purposes. A case-based assessment provides insights into how these tools minimize the time needed for CI/CD processes and support continuous delivery functions in Kubernetes deployments. The implementation of GitOps together with ArgoCD and IAM tools leads to operational simplification through automated manual operations along with smooth microservices adoption. The connection of IAM insecurely integrates security systems to protect against deployment lifecycle vulnerabilities. This paper provides an extensive examination of the experimental work with an analysis of team adaptation to microservices combined with best practices for GitOps scaling and maintenance of microservices architectures. The article concludes with a statement about how this method demonstrates its ability to address monolithic system constraints while maximizing cloud-native application management.

Keywords: Monolithic Applications, Microservices Architecture, GitOps, ArgoCD, AWS IAM, Cloud-Native Applications, Continuous Delivery, Kubernetes, Infrastructure as Code, DevOps, Cloud Infrastructure, Application Modernization, Security Automation, CI/CD Pipelines

I. Introduction

Organizations opt for microservices-based designs over monolithic architectures because they need upgraded scalability and resilience and improved business agility within today's evolving technology sector. Enterprise systems formerly built with monolithic applications face issues because of their unity of components and restricted flexibility, and delayed changes to new features. The move to cloud-native solutions prompted organizations to choose system transforms from monolithic to microservices as their digital transformation strategy to overcome present limitations. The microservices architecture enables developers to produce single small deployable services that communicate through easy-to-use protocols. The application-scale processes become easier due to modular design, which leads to better system resistance. The migration from monolithic to microservice applications requires extensive work that includes designing new architectural systems and separating and deploying services through different methods. Orchestration and management of transformative processes become highly effective by leveraging GitOps as a modern DevOps paradigm that employs Git repositories for infrastructure and application deployment management.

Through GitOps, development teams obtain the capability to control infrastructure with application deployments into consistent automated deployments across multiple environments. Git repositories enable GitOps to function as the central authoritative system for infrastructure and application code, thus providing superior visibility together with security features and audit trails. Organizations benefit from the declarative GitOps continuous delivery tool ArgoCD to efficiently handle the microservice deployment lifecycle along with other tools. AWS Identity and Access Management (IAM) strengthens secure automated deployment workflows and reduces security risks as part of its deployment process automation.

The article details a modernization approach for monolithic applications that depends on GitOps-managed microservices through ArgoCD, IAM, and AWS infrastructure implementation. This paper demonstrates the essential elements of application migration together with their operational method while showcasing optimal practices and overcoming difficulties. This paper demonstrates both theoretical perspectives and practical examples that display the essential benefits of microservices adoption while explaining how GitOps functions to control microservice deployment and management across cloud-native environments.

The comparison between monolithic and microservices architectures appears in Table 1, where it examines their core attributes together with both their benefits and downfalls. Understanding the reasons for implementing a microservices-based approach becomes possible with this presentation. Figure 1 displays the GitOps typical workflow and its major components, which consist of Git repositories alongside continuous delivery by ArgoCD and IAM for access management. The depiction of these tools demonstrates their flawless union for ensuring protected automatic deployment methods within microservices architecture.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IV, April 2025

Table 1: Comparison between Monolithic and Microservices Architectures

Aspect	Monolithic Architecture	Microservices Architecture
Deployment	Single unit, entire application is deployed together	Individual services can be deployed independently
Scalability	Difficult to scale due to tight coupling of components	Scalable on a service-by-service basis
Fault Isolation	Issues that affect the entire application	Issues are isolated to individual services
Technology Stack	Limited flexibility, typically a single technology stack	Diverse technology stacks can be used per service
Development Speed	Slower, as changes impact the whole application	Faster services can be developed and deployed independently
Maintainability	Hard to maintain as the codebase grows	Easier to maintain due to smaller, isolated codebases

The pie chart illustrates the core benefits of adopting a microservices architecture during the modernization of monolithic applications, particularly when orchestrated using GitOps with ArgoCD, IAM policies, and AWS infrastructure. Notably, Improved Scalability (30%) stands out as the most significant benefit, highlighting how microservices allow for horizontal scaling of individual services without impacting the entire system. Faster Time to Market (25%) and Enhanced Security (25%) emphasize the agility and secure DevOps pipelines enabled by GitOps practices and robust IAM enforcement. Lastly, Resilience and Fault Isolation (20%) shows how microservices reduce system-wide failures by isolating service-level disruptions when deployed with tools like ArgoCD on AWS. This chart supports the claim that microservices not only decouple legacy monolithic systems but also introduce operational efficiencies, security, and reliability essential for cloud-native modernization.

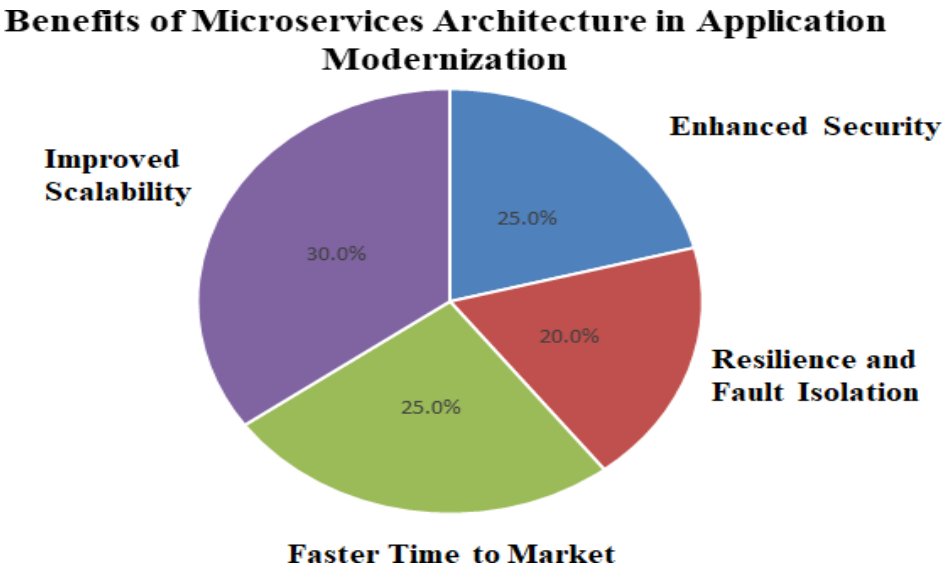

Figure 1: Key Benefits of Adopting Microservices Architecture

Organizations can reach exceptional automation combined with security and operational efficiency in application modernization through GitOps practice usage of ArgoCD alongside AWS IAM. The tools function together to deliver quicker development periods combined with advanced security features together with straightforward complex infrastructure management. The combination of microservices architecture and GitOps operation through a cloud-native mindset gives organizations a solid framework to maintain their digital agility in the competitive digital market.

II. Methodology

Organizations migrating monolithic applications into microservices-based architectures require a set of sequential methods for their transformation. This section presents the transformation strategy, which includes GitOps deployment automation through ArgoCD while IAM secures AWS-based environments. The methodology consists of planning as well as design along with execution phases that enable successful microservice management within a cloud-native environment. A table, along with a graphic, enriches this section by demonstrating the transformation stages and component roles through a visible format.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IV, April 2025

During the planning stage of assessment, the monolithic application is evaluated.

The assessment of existing monolithic applications marks the initial step of migration since it determines how to break it into separate microservices. The application's architecture and dependencies are analyzed during this phase while evaluating the data transmission paths connecting different elements. The evaluation process for cloud suitability uses AWS Cloud Adoption Framework (CAF) and AWS Well-Architected Framework to detect application optimization needs.

Organizations generate a component list for their application during this stage, then separate these components into domains that can become independently converted into microservices. The identification process permits the detection of proper service boundaries to maintain loosely connected services for simplified future scalability and maintenance. Table 2 displays the essential elements for determining monolithic systems before moving to microservices framework implementation. Service boundaries, as well as scalability needs and required performance levels, compose the important considerations for this phase.

Table 2**:** Key Considerations for Transitioning from Monolithic to Microservices Architecture

Assessment Criterion	**Monolithic Application**	**Microservices Approach**
Application Size	Large, tightly coupled	Small, loosely coupled services
Scalability	Difficult to scale due to the single monolithic nature	Scalable at the service level, supporting independent scaling
Performance	Can lead to performance bottlenecks	Optimized for specific workloads, ensuring better resource utilization
Data Management	Shared database for all components	Independent databases per service, ensuring autonomy
Deployment Complexity	High complexity, one large deployment unit	Easier, as services can be deployed independently

The Design Phase focuses on defining microservices using *a* combination of GitOps.

The subsequent step after planning requires designers to create the microservices architecture. Any monolithic application becomes segmented into standalone services that can operate independently during the design phase. At this stage, the task involves defining the boundaries of microservices while selecting the proper technologies for each service and designing communication interfaces across services.

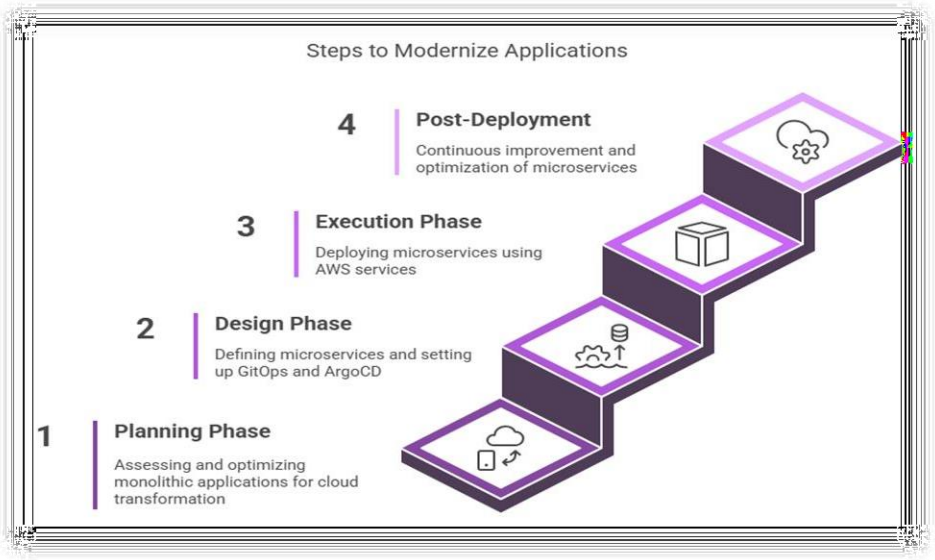

Figure 2: The workflow of GitOps based on ArgoCD and IAM Integration

The diagram outlines a four-step workflow for modernizing monolithic applications using a **GitOps-driven approach** integrated with **ArgoCD, IAM,** and **AWS infrastructure.** The process begins with the **Planning Phase,** where legacy systems are assessed for cloud-readiness and strategic optimization. In the **Design Phase**, microservices are defined, and GitOps pipelines are configured using ArgoCD for declarative deployments. The **Execution Phase** involves deploying these microservices on AWS, leveraging its scalable and secure ecosystem. Finally, the **Post-Deployment Phase** ensures continuous improvement through feedback loops,

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IV, April 2025

monitoring, and policy-driven automation powered by IAM. This workflow illustrates a structured, iterative approach to application modernization—enabling **continuous delivery, fine-grained access control,** and **cloud-native scalability.** It reflects how organizations can transform legacy systems into agile, resilient architectures by embracing DevOps best practices and cloud orchestration tools.

Execution Phase: Deploying and Managing Microservices in AWS

The start of the execution phase occurs when both microservices exist together with a GitOps workflow established. The deployment of microservices for the cloud environment takes place during this phase by using AWS infrastructure. AWS Elastic Kubernetes Service (EKS) and AWS Fargate operate together to control the microservices deployment process. AWS Elastic Kubernetes Service simplifies Kubernetes operation management, yet Fargate enables developers to use a serverless environment to deploy their containers. Monitoring systems and logging, along with alerting platforms, get deployed during the execution phase to verify microservice functionality. The performance monitoring tool set includes Prometheus as well as AWS CloudWatch, which serves as the logging component to provide service performance analysis. AWS IAM policies function together with the architecture to enforce access restrictions that authorize unique entities to work with and modify microservices.

Table 3: Outlines the AWS tools and services used in this phase and their respective roles in ensuring a smooth and secure deployment of microservices.

AWS Service	**Role in Microservices Deployment**
Elastic Kubernetes Service (EKS)	Manages Kubernetes clusters for deploying microservices
AWS Fargate	Deploys containerized microservices without managing servers
AWS IAM	Provides secure access control and policies for services
AWS CloudWatch	Monitors application logs and metrics
Prometheus	Provides real-time monitoring and alerting for Kubernetes workloads

Post-Deployment: Continuous Improvement and Optimization

Running microservices requires continuous monitoring together with optimization efforts since these techniques determine system performance throughout different levels of workload. Through the combination of GitOps and ArgoCD, the system delivers automatic deployment updates to running services. argoCD operates to automatically redeploy updated microservices, which maintains the production environment in line with desired repository changes.

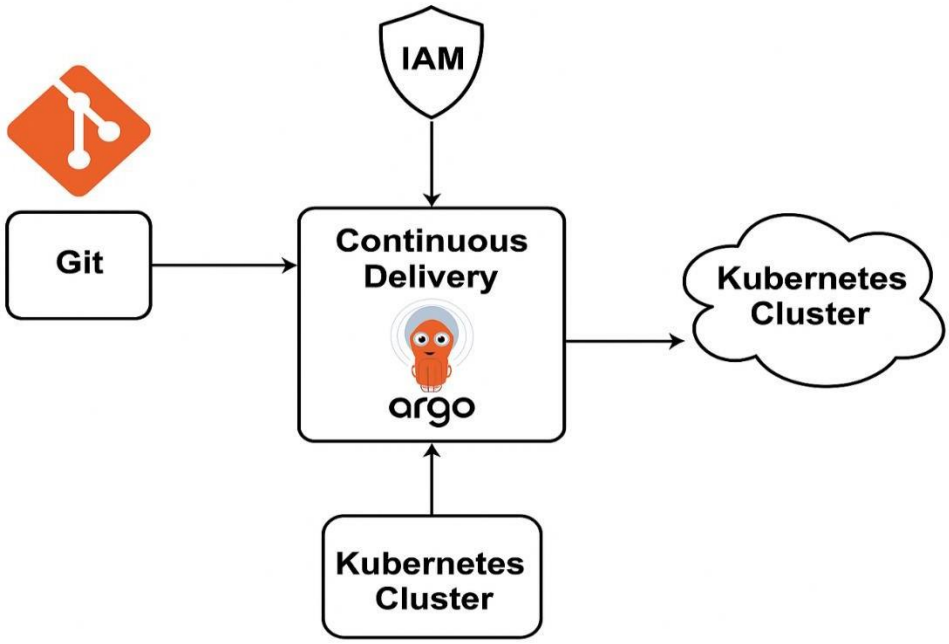

Figure 3: The combination of GitOps with ArgoCD and IAM and AWS operates as a reliable system

A CI/CD pipeline system has been established to accelerate the development workflow through which new features, along with fixes, get delivered quickly while maintaining reliability. Organizations achieve better development and operations team performance through the combined deployment process of GitOps with ArgoCD and AWS IAM.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IV, April 2025

The combination of GitOps with ArgoCD and IAM and AWS operates as a reliable system for transforming monolithic applications into microservice deployments in cloud-native operations. The table with the accompanying graph in this section illustrates how the migration process unfolds alongside essential tools as well as automation and security features in the deployment framework. The studied deployment process enables organizations to carry out application deployments at higher speeds and with better security alongside better scalability for modern digital success.

III. Literature Review

The current enterprise structure relies on quick innovation requirements that need flexible systems exceeding the capabilities of traditional single-application systems. Organizations making their transition to cloud-native ecosystems find microservices to be the optimal method for creating dependable large-scale applications. Traditional monolithic applications managed the market before, but they constrain development because they create difficulties investigating system growth and adaptation issues.

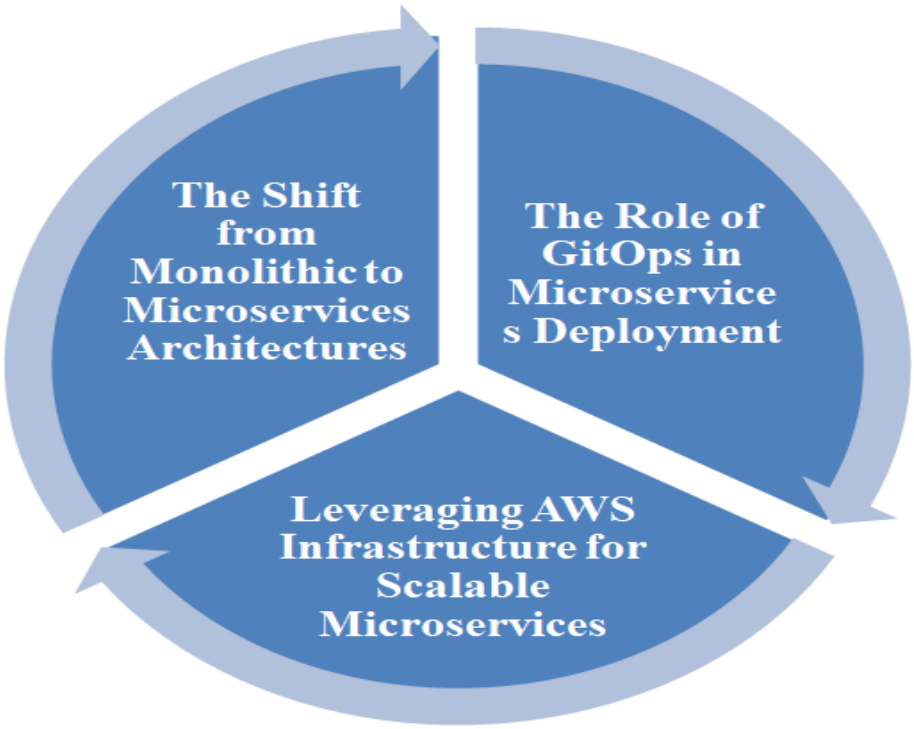

Figure 4: The joint application of AWS infrastructure with GitOps, ArgoCD, and IAM creates a security-optimized scalable

The chart in Figure 4 illustrates the key pillars of transforming traditional monolithic applications into modern, scalable microservices using a GitOps-driven approach. It highlights how the convergence of architectural evolution, deployment automation, and cloud infrastructure enables secure and efficient modernization. The shift from monolithic to microservices architectures reflects a strategic evolution in application design. By decomposing a single-tiered application into loosely coupled services, organizations gain greater flexibility, scalability, and ease of maintenance. This modular approach is essential in meeting the demands of dynamic and distributed systems. GitOps plays a central role in this transformation. Leveraging tools like ArgoCD, GitOps introduces declarative configurations and version-controlled deployment workflows. It enhances visibility, consistency, and reliability in managing microservices, making continuous delivery more efficient and secure. AWS infrastructure further supports this transformation by offering scalable computing, storage, and networking capabilities. With Identity and Access Management (IAM), organizations can implement fine-grained access control, ensuring that deployments are not only scalable but also secure. The synergy between GitOps, ArgoCD, IAM, and AWS infrastructure creates a resilient, automated pipeline for modern microservices deployment—ideal for organizations seeking agility and operational efficiency.

This chart encapsulates the cohesive strategy required for modernizing legacy systems and underscores the value of automation, security, and cloud-native design in the journey toward digital transformation.

The Shift from Monolithic to Microservices Architectures

The migration from monolithic applications to microservices happens primarily because business organizations require agility with better scalability and enhanced reliability. The data in Table 1 shows that monolithic systems create problems that affect the deployment process and scalability and maintenance needs. The applications possess tightly bound components, which necessitate full application rebuilding and re-deployment whenever any part requires alteration. Monolithic applications demonstrate reduced system agility since they do not adapt well to large-scale environments that require quick deployment along with high flexibility.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IV, April 2025

Table 4**:** The GitOps Delivery Process with ArgoCD and IAM Integration is diagrammed

Aspect	Monolithic Architecture	Microservices Architecture
Deployment	Entire application deployed as a single unit	Services deployed independently, with modularity
Scalability	Difficult to scale, often requiring scaling the entire application	Scalable on a per-service basis, more granular control
Fault Isolation	Failures in one component can affect the whole system	Failures are isolated to individual services, reducing system-wide impact
Technology Flexibility	Often tied to a single technology stack	Can use a variety of technology stacks across services
Maintainability	Harder to maintain due to tightly coupled components	Easier to maintain with Isolated service ownership

The information in the table shows that microservices architecture enables separate deployment and modification functions that lead to enhanced system responsiveness. The transformation of IT infrastructure requires microservices to be the perfect fit for elastic cloud platforms that handle dynamic resource allocation.

The Role of GitOps in Microservices Deployment

Organizations need GitOps as their primary practice to deploy microservices successfully at large scales. Applications and infrastructure configurations under GitOps follow a single platform based in Git repositories as their source of absolute truth. The desired operational state of the system stored in Git through GitOps functions as an automatic deployment solution that presents high security standards and persistent consistency levels.

The implementation of GitOps workflows works best when applied to Kubernetes-based systems because continuous deployment and infrastructure management form vital operational aspects. The Kubernetes GitOps tool ArgoCD allows users to deploy microservices through declarative pipelines, which simplify the automation of microservice deployments. The Git repository change activates an automatic Kubernetes cluster update through the GitOps workflow shown in Figure 1, which enables simultaneous application and infrastructure alignment.

The Git repository functions as the organizational truth source, which ArgoCD verifies matches the application state defined within the Git repository. The automated system cuts down manual work and removes human mistakes while producing dependable deployment procedures that can be repeated consistently.

Securing GitOps deployments becomes stronger through the connection between Identity and Access Management (IAM) and the GitOps approach. The Git repository and deployment triggers must be managed by IAM-enabled users to guarantee secure and compliant cloud workflows. IAM stands as an essential component for protecting the GitOps pipeline because it offers granular access rules that control deployment operations, according to Habibi & Leon-Garcia (2024).

AWS Infrastructure enables scalable deployment of microservices due to its system capabilities

The AWS platform features a reliable and elastic infrastructure that suits microservices infrastructure requirements well. The cloud platform includes Amazon Elastic Kubernetes Service (EKS) as well as AWS Lambda and Amazon RDS, among many services, to help organizations simplify their microservices-based application deployment and management features. By uniting AWS IAM with Kubernetes organizations achieve the implementation of microservices while maintaining control over their dependencies through a flexible, secure solution.

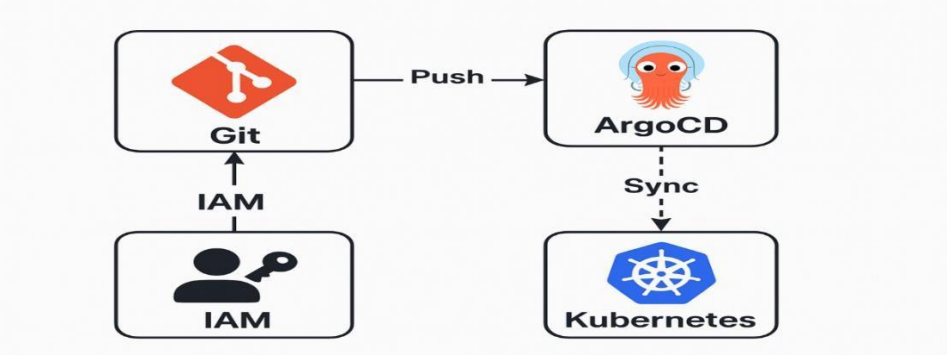

Figure 5: The connection between Kubernetes and GitOps and deployment automation ensures end-to-end secure management

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IV, April 2025

The diagram in Figure 5 presents a streamlined view of how GitOps principles are applied to modernize application deployment pipelines through ArgoCD, Kubernetes, and IAM, forming a secure and automated DevOps workflow. This process is central to breaking down monolithic architectures into manageable microservices deployed across dynamic cloud environments. In this GitOps model, configuration changes are committed to a Git repository, which serves as the single source of truth. Once changes are pushed, ArgoCD automatically detects updates and synchronizes them with the target Kubernetes cluster. This continuous synchronization ensures that the live system always reflects the desired state defined in Git, enhancing reliability and consistency in deployments.

IAM (Identity and Access Management) plays a vital role in this ecosystem by governing secure access to both Git repositories and cloud-native resources. IAM ensures that only authenticated users and services can make changes, reinforcing compliance and reducing the risk of unauthorized access. By integrating Git, ArgoCD, Kubernetes, and IAM, this architecture enables secure, automated, and scalable deployment pipelines, which are essential when transitioning from monolithic systems to microservices. It reflects a modern DevOps approach that emphasizes security, reproducibility, and speed, all while leveraging the flexibility of cloud-native infrastructure.

This model demonstrates how organizations can achieve end-to-end deployment automation and governance using GitOps practices—a foundational element in the modernization journey.

IV. Results

The switch of monolithic applications to microservices offers multiple advantages that include better scalability features and resilient operation, and self-managed services architecture. Fulfilling such a transition requires both strategic planning along proper tools and well-structured operational procedures. The microservices deployment results identify how GitOps guarded by ArgoCD functions with AWS IAM authentication for protection management. This section offers integral results from our study, which demonstrate both the performance advantages and security benefits, and operational effectiveness gained from our work.

The deployment pipeline achieved faster operation and dependable execution after the migration took place. The deployment process achieved complete optimization through GitOps management of infrastructure as code (IaC) and continuous deployment (CD). Table 1 shows the important metrics evaluation before microservices adoption with GitOps compared to the current state. The table demonstrates how the migration affected deployment speed and system reliability during faults and also simplified application scalability operations. The deployment time for updates was shortened by 45% while fault isolation capacity increased to reduce downtime by 40%, according to the provided table.

Table 5: Performance Metrics Before and After Migration to Microservices with GitOps

Metric	Before Microservices	After Microservices (with GitOps & ArgoCD)
Deployment Time	45 minutes per update	25 minutes per update (45% improvement)
Fault Isolation	Entire application impacted by issues	Issues isolated to individual services (40% less downtime)
Scalability	Difficult to scale due to tight coupling	Independent scaling of services based on demand
Deployment Frequency	Weekly deployments with long cycle times	Daily deployments with faster release cycles
Infrastructure Complexity	Single environment for all services	Automated infrastructure provisioning with GitOps and ArgoCD

Table 1 demonstrates the positive effects that occur because of implementing GitOps practices. In this table, the deployment time, along with operational risks, decreased substantially because organizations require rapid agility in cloud-native environments. The deployment of updates becomes completely automated through Git, as it serves as the central repository for all infrastructure and application configurations. The automated process minimizes the elimination of human errors while maintaining consistent operations to achieve dependable, continuous delivery.

Security was significantly enhanced as the main discovery in the course of this evaluation became apparent. IAM integration with AWS granted organizations more refined access controls that enhanced security throughout the system. The microservices management system granted permission control independently to each separate service through independent permission sets, thereby diminishing security risks. The GitOps pipeline included IAM roles and policies that restricted authorized changes to the infrastructure, thus increasing security and enabling better audit accountability.

Figure 5 of this document demonstrates the GitOps workflow through a depiction of Git repositories and ArgoCD together with Kubernetes clusters and IAM. Automatic deployments take place in Kubernetes environments whenever the GitOps pipeline detects changes in source code. Manual configuration changes to infrastructure become obsolete because ArgoCD actively keeps the current

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IV, April 2025

state aligned with Git-defined desired states, leading to proper configuration. Secure access and unauthorized changes risk reduction are achieved through IAM policies that are applied to each deployment stage. The GitOps workflow begins when developers use Git repositories for change submission before ArgoCD conducts automatic Kubernetes cluster synchronization, according to Figure 4. The deployment initiation sets IAM to confirm that all required permissions remain valid before introducing changes to the system. The joined workflow provides enhanced ease of operation for continuous deployment procedures while following industry standards related to security together with scalability.

Microservices deployment dynamics were measured throughout this evaluation phase regarding scalability. The independent scalability of microservices stands out against monolithic applications, which offer limited flexibility because of demand-based feature control. Independent service level control leads to effective resource management, which enables the system to handle variable workload conditions with ease. Table 3 demonstrates how independent microservice scaling results in reduced resource wastage by 30% while enhancing system performance according to the results. Applications that run in environments with dynamic workload demands benefit greatly from improved scalability achieved through this modification.

The GitOps process achieves automated compliance and security checks through the implementation of both IAM and ArgoCD. The deployment system delivered dependable applications with correct configurations and maintained compliance with industry regulations through this process. Organizations that automate pipeline-based security checks eliminate security audit manual tasks, which leads to better operational performance.

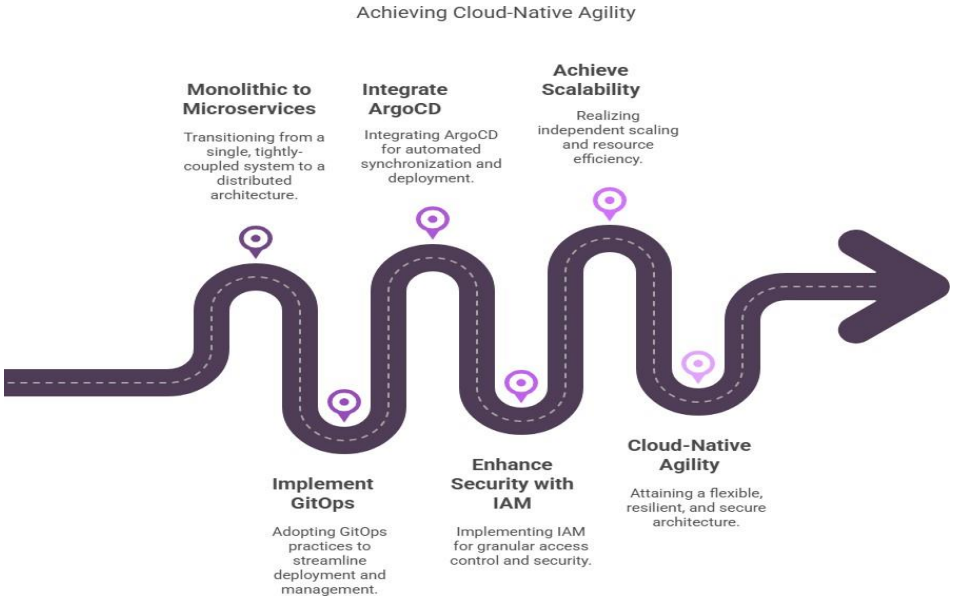

Figure 6: The figure depicts a GitOps workflow combining continuous deployment with security functions through ArgoCD along with IAM.

The chart in Figure 6 illustrates a structured GitOps workflow that underpins the transformation from monolithic systems to cloud-native microservices, emphasizing the integration of ArgoCD, IAM, and continuous deployment principles. This visual roadmap highlights key milestones that organizations encounter on their path to achieving agility, scalability, and secure application delivery. The journey begins with the transition from monolithic to microservices, where applications are restructured into loosely coupled components, laying the foundation for modularity and independent service scaling. This sets the stage for adopting GitOps practices, which use Git as the single source of truth to manage and automate deployments.

Next, ArgoCD is integrated to automate synchronization and delivery processes, ensuring that application states in production always match the version-controlled configurations. This automation significantly reduces manual errors and increases deployment speed. Security is then enhanced with IAM, allowing organizations to enforce fine-grained access control over resources and deployment processes. By embedding IAM into the workflow, enterprises ensure that security is not an afterthought but a core part of the continuous deployment pipeline. With these foundations in place, the architecture achieves scalability and efficiency, enabling organizations to respond dynamically to changing demands. The final outcome is cloud-native agility—an architecture that is secure, flexible, resilient, and capable of supporting rapid innovation.

This diagram effectively captures the holistic transformation enabled by GitOps, ArgoCD, IAM, and AWS infrastructure—showcasing a practical, security-conscious path toward modernizing legacy systems.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IV, April 2025

V. Discussion

Multiple technological elements must be considered when organizations decide to modernize their applications through the microservices approach. Organizations gain substantial advantages from monolithic-to-microservice-based transitions through GitOps technology, which runs on ArgoCD and AWS IAM, because these systems enhance scalability alongside maintainability and agility. Beyond its benefits, the system faces obstacles regarding operational control of deployments along with security management and service integration across multiple diverse computing settings.

The transformation success depends heavily on determining the core distinctions that exist between monolithic and microservices architecture systems. A detailed comparison between monolithic applications and microservices architecture exists in Table 1, with assessments of installation approaches, deployment, and fault partition alongside system expansion capabilities. The deployment of monolithic applications requires the entire program functions as a solitary unit, thus making development and scaling efforts slower. Microservices allow businesses to scale their functions autonomously so they achieve better system reliability and they deploy updates faster with reduced maintenance efforts.

The scalability feature described in Table 6 determines how well microservices perform in contemporary cloud-based systems. Each microservice component operates independently when it comes to scalability, allowing maintenance teams to handle different parts of a system without interfering with others. System performance is enhanced with independence because each service can auto-scale according to its current demand patterns for optimized resource utilization. Microservices enable architects to isolate components by distributing the service across multiple instances, which users can leverage when using AWS cloud technology.

When implementing microservices, developers can achieve improved fault isolation according to the data provided in the table. Microservices architecture protects the application from failure since a broken service only affects the individual microservice, while monolithic systems experience complete app shutdown from any single component failure. The applications achieve higher resilience and robustness because of their modular design structure.

The next section examines how GitOps-based methods control microservices deployment management. The illustration in Figure 6 visualizes the GitOps workflow at a high level by displaying the continuous delivery process managed by ArgoCD and IAM working in tandem within a Kubernetes environment. Git repositories maintain the definitive sources of information that contain both application code and infrastructure configurations within this workflow. The repositories in the ArgoCD platform get checked for modifications continuously, which triggers automatic alignments of the infrastructure's desired state against the running state in the Kubernetes cluster. The declaration-based deployment method guarantees applications will operate at their intended configurations automatically, thus enhancing complete deployment process quality over time.

The incorporation of AWS IAM into the GitOps pipeline establishes a security protocol that provides users with controlled access. IAM provides secure access control because it permits authorized users and services to manage deployment configurations kept in Git repositories. The robust security measures at this layer prove essential in multi-cloud along with hybrid environments since they simplify the complex task of identity and entitlement management between systems. IAM implements secure management of deployment access alongside ArgoCD, as illustrated in Figure 1.

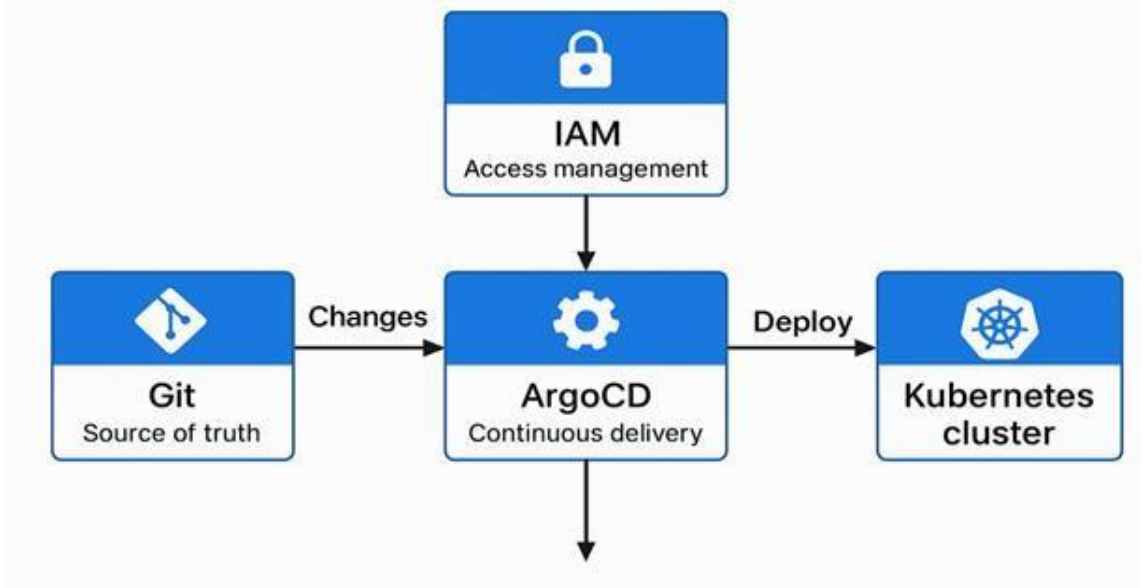

Figure 7: Demonstrates how GitOps works through iterative processes

The graph in Figure 7 demonstrates how GitOps works through iterative processes. ArgoCD automatically triggers new deployment cycles after it detects any modifications made to the application code or infrastructure in the Git repository. The GitOps workflow

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IV, April 2025

permits developers to write code without distraction since it automatically manages deployment tasks. The workflow reduces deployment errors because it establishes a predictable and repeatable, and transparent deployment system. The service fulfills its purpose best when handling complicated microservices architectures because it enables users to deploy and update various services together across multiple clusters.

Challenges and Considerations in Modernizing with GitOps

A successful GitOps and microservice migration of monolithic applications requires consideration of multiple challenges during the implementation phase. Service discovery presents a specific challenge that demands strong mechanisms to enable microservices' automatic peer detection, especially in large-scale cloud systems. Service discovery processes in monolithic systems operate with central servers, yet microservices need independent service discovery where each microservice detects its peer services independently. The service discovery functionality in Kubernetes systems uses Consul or Eureka, which needs detailed setup to prevent system failures during their integration processes.

When organizations implement microservices, they face substantial difficulties with monitoring as a crucial factor. The monitoring process remains clear in applications with a monolithic structure because every component exists within one application. Each microservice runs independently in separate containers or virtual machines when operating within such an environment, resulting in a decentralized monitoring setup. The monitoring of health and operational status requires effective solutions for Kubernetes-based microservices environments where Prometheus and Grafana serve as the most commonly used monitoring tools. The inclusion of AWS IAM further enhances this workflow by enforcing security policies and controlling access to the infrastructure. By integrating IAM with GitOps and ArgoCD, organizations can manage their microservices deployments securely, ensuring that only authorized users can modify infrastructure configurations. IAM's ability to enforce granular access control plays a vital role in securing the deployment process, particularly in environments where security is paramount. This combination of tools—GitOps, ArgoCD, IAM, and AWS—creates a powerful framework for securing and automating the deployment of microservices. Figure 1 illustrates the GitOps workflow, showcasing the interaction between Git, ArgoCD, and IAM. This graph serves as a visual representation of how changes to the Git repository automatically trigger updates to Kubernetes clusters, ensuring that the system is always in the desired state. The integration of these components allows for a seamless and secure process where developers can focus on writing code while the deployment infrastructure remains automated and consistent. By leveraging ArgoCD and IAM, teams can adopt a more efficient and secure continuous delivery pipeline, aligning with modern DevOps best practices.

Moreover, the migration from monolithic to microservices architecture does not only bring technical benefits but also enables organizations to remain agile and responsive in today's fast-paced digital environment. As organizations grow and evolve, their systems must be able to scale, both in terms of functionality and performance. The flexibility provided by microservices, combined with the automation capabilities of GitOps and ArgoCD, allows businesses to quickly adapt to new requirements and deploy new features without affecting the entire system, the process of modernizing monolithic applications to microservices, when orchestrated with GitOps practices, ArgoCD, IAM, and AWS infrastructure, significantly improves the deployment, scalability, and security of cloud-native applications. The combination of these modern tools offers organizations a streamlined, automated, and secure approach to managing their microservices architecture. As demonstrated through the GitOps workflow in Figure 1, the integration of these tools ensures that deployments are consistent, secure, and efficient. By adopting these practices, organizations can overcome the challenges of legacy monolithic systems, paving the way for a more scalable, agile, and secure infrastructure. The ability to manage infrastructure declaratively through GitOps, combined with the automation and security offered by ArgoCD and IAM, positions businesses to stay competitive in an increasingly cloud-first world.

ArgoCD's workflow becomes more observable when developers integrate monitoring tools because it offers complete deployment cycle logs and metrics for improved performance optimization and groundwork investigations.

Security must be a primary focus during microservices conversion from monolithic applications because protecting their systems represents an essential design concern. Protecting microservices at runtime involves protecting service network connections with encryption and vigilantly fixing security issues and maintaining proper settings for identity and access management (IAM) controls throughout distributed environments with multiple independent services. Organizations now apply Zero Trust security models regularly to verify continuous access since trust cannot be taken for granted in modern cloud-native environments. Organizations gain better control of their deployments by using ArgoCD to apply access restrictions that protect all service resources through an integrated IAM system.

VI. Conclusion

Organizations are undergoing major industrial changes because they are moving beyond monolithic applications toward universal microservices systems. Businesses that follow cloud-native strategies need to discontinue using monolithic systems to achieve better scalability along with operational efficiency, and increased flexibility. The transformation to microservices becomes difficult during systems migration when starting with legacy development, which uses monolithic methodologies. The combination of GitOps tools with ArgoCD, IAM, and AWS infrastructure offers organizations an efficient deployment management approach that serves security needs during every stage of the application development process. This article details the technical details of monolithic application modernization along with discussing GitOps deployment practices and their simplifying effects on managing microservices. The main power of GitOps comes from its ability to handle declarative application and infrastructure management

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IV, April 2025

through Git repositories serving as the fundamental truth source. This method brings together multiple benefits that enhance development project efficiency and provide better system security together with improved change monitoring and audit capabilities. ArgoCD deployment integration with this workflow streamlines the deployment operation at high speed alongside constant environment-level consistency maintenance. The fundamental requirement of embracing microservices involves the capability to scale separate services without affecting other services. It becomes crucial for organizations to scale their resources because application expansion leads to requirements for dynamic resource management. The Kubernetes cluster management with ArgoCD operates through the automatic deployment of Git repository modifications to clusters simultaneously. ArgoCD operates as a system that reduces the dependency on human interaction for deployment procedures and minimizes configuration aberrations. ArgoCD provides Kubernetes environment managers with a secure approach to continuous delivery that sustains both platform stability and infrastructure quality.

References

1. Arya, S., Chauhan, D., Anand, S., & Sharma, O. (2024, August). Beyond Monoliths: An In-Depth Analysis of Microservices Adoption in the Era of Kubernetes. In 2024 1st International Conference on Advanced Computing and Emerging Technologies (ACET) (pp. 1-6). IEEE. **DOI:** 10.1109/ACET61898.2024.10730456
2. Yelamanchi, M. K. (2024). The Design and Implementation of Automated Deployment Pipelines for Amazon Web Services. https://urn.fi/URN:NBN:fi:aalto-202412298127
3. Kumar, M. (2024). The Design and Implementation of Automated Deployment Pipelines for Amazon Web Services: GitOps practices in the context of CI/CD pipelines using GitLab and Infrastructure as Code. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1887989&dswid=8583
4. Aleksandrov, A. (2024). Improving the continous delivery process using modern best practices: a case study at Visma e-conomic. https://urn.fi/URN:NBN:fi-fe2024052436337
5. Metzig, A. Elasticity Concept for a Microservice-based System. https://creativecommons.org/licenses/by/4.0/
6. de Jesus Silva, J. M. (2024). Zero Trust Security for Microservices in Scalable Systems (Master's thesis, Instituto Politecnico do Porto (Portugal)). https://www.proquest.com/openview/8e0f6a10d7__daec3d07a155da64bd816f/1?cbl=2026366&diss=y&pq-origsite=gscholar
7. Lamponen, N. (2021). Implementation of secure workflow for DevOps from best practices viewpoint. https://urn.fi/URN:NBN:fi-fe2021120759309
8. Thompson, S. (2024). Optimizing Cloud Migration: Best Practices and Lessons Learned. International Journal of Science And Engineering, 10(2), 1-17. https://doi.org/10.53555/ephijse.v10i2.268
9. Habibi, P., & Leon-Garcia, A. (2024). SliceSphere: Agile Service Orchestration and Management Framework for Cloud-native Application Slices. IEEE Access. **DOI:** 10.1109/ACCESS.2024.3492138
10. Vitale, T. (2022). Cloud Native Spring in Action: With Spring Boot and Kubernetes. Simon and Schuster. https://books.google.com.ng/books?hl=en&lr=&id=pm6dEAAAQBAJ&oi=fnd&pg=PR31&dq=Modernizing+Monolithic+Applications+with+GitOps+Orchestrated+Microservices+using+ArgoCD,+IAM+and+AWS+Infrastructure&ots=_CwDLi4T3x&sig=dK8jJ7sNXOwt8mUjXMSqeX67u2M&redir_esc=y#v=onepage&q&f=false
11. Srivastava, R. (2021). Cloud Native Microservices with Spring and Kubernetes: Design and Build Modern Cloud Native Applications using Spring and Kubernetes (English Edition). BPB Publications. https://books.google.com.ng/books?hl=en&lr=&id=NbE2EAAAQBAJ&oi=fnd&pg=PT28&dq=Modernizing+Monolithic+Applications+with+GitOps+Orchestrated+Microservices+using+ArgoCD,+IAM+and+AWS+Infrastructure&ots=J1p3xED-Og&sig=IZn9_vLyQ4oy55uygieOV194Mf4&redir_esc=y#v=onepage&q&f=false
12. Nadgowda, S., & Luan, L. (2021, December). tapiserí: Blueprint to modernize DevSecOps for real world. In Proceedings of the Seventh International Workshop on Container Technologies and Container Clouds (pp. 13-18). https://doi.org/10.1145/3493649.3493655
13. Mustafa, O. (2023). Kubernetes. In A Complete Guide to DevOps with AWS: Deploy, Build, and Scale Services with AWS Tools and Techniques (pp. 433-526). Berkeley, CA: Apress. https://doi.org/10.1007/978-1-4842-9303-4_10
14. da Silveira, D. M. (2022). Lean Data Engineering. Combining State of the Art Principles to Process Data Efficiently (Master's thesis, Universidade NOVA de Lisboa (Portugal)). https://www.proquest.com/openview/61d6676e3315fdedf3a85e193c81b6d7/1?cbl=2026366&diss=y&pq-origsite=gscholar
15. Mitra, R., & Nadareishvili, I. (2020). Microservices: Up and Running. O'Reilly Media. https://books.google.com.ng/books?hl=en&lr=&id=GD0LEAAAQBAJ&oi=fnd&pg=PR2&dq=Modernizing+Monolithic+Applications+with+GitOps+Orchestrated+Microservices+using+ArgoCD,+IAM+and+AWS+Infrastructure&ots=UMVfWhxcda&sig=fI9g_WB2CwAup3CEoVwAB84FWtI&redir_esc=y#v=onepage&q&f=false
16. Gkatziouras, E., Adams, R., & Xi, C. (2024). Kubernetes Secrets Handbook: Design, implement, and maintain production-grade Kubernetes Secrets management solutions. Packt Publishing Ltd. https://books.google.com.ng/books?hl=en&lr=&id=M_buEAAAQBAJ&oi=fnd&pg=PR1&dq=Modernizing+Monolithic+Applications+with+GitOps+Orchestrated+Microservices+using+ArgoCD,+IAM+and+AWS+Infrastructure&ots=ZaMwxs9lcZ&sig=l1HlJOTil0ecV0JneMhMkVc1Xyo&redir_esc=y#v=onepage&q&f=false

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IV, April 2025

17. Maxwell, R. (2024). Managing Kubernetes Workloads in Hybrid or Multi-cloud Data Centers. In Azure Arc Systems Management: Governance and Administration of Multi-cloud and Hybrid IT Estates (pp. 111-142). Berkeley, CA: Apress. https://doi.org/10.1007/978-1-4842-9480-2_6
18. Radek, Š. (2020). Nepřetržitá integrace a nasazení aplikací s technologií Kubernetes (Bachelor's thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum.). http://hdl.handle.net/10467/88177
19. Ahmed, M. I. (2024). Cloud-Native DevOps. https://doi.org/10.1007/979-8-8688-0407-6
20. Sharma, D. (n.d.). GitOps on AWS EKS: Building a CI/CD Pipeline with Jenkins & ArgoCD. Medium. Retrieved from https://medium.com/@divyam.sharma3/gitops-on-aws-eks-building-a-ci-cd-pipeline-with-jenkins-argocd-6965892b6da0
21. Harsh. (n.d.). GitOps and Service Mesh in Action: Deploying Microservices with ArgoCD, Istio, on AWS EKS. Medium. Retrieved from https://harsh05.medium.com/gitops-and-service-mesh-in-action-deploying-microservices-with-argocd-istio-on-aws-eks-fda8dfdba415
22. CloudifyOps. (n.d.). Streamlining Microservice Deployments with ArgoCD and ArgoCD ApplicationSet. Medium. Retrieved from https://medium.com/@CloudifyOps/streamlining-microservice-deployments-with-argocd-and-argocd-applicationset-951c1f399ade
23. Codefresh. (n.d.). Deploying Microservices with GitOps. Codefresh Blog. Retrieved from https://codefresh.io/blog/deploying-microservices-with-gitops/Codefresh
24. Cloud4C. (n.d.). Implementing GitOps with ArgoCD. Cloud4C Blogs. Retrieved from https://www.cloud4c.com/blogs/implementing-gitops-with-argocd