IJRAR.ORG E-ISSN: 2348-1269, P-ISSN: 2349-5138

INTERNATIONAL JOURNAL OF RESEARCH AND ANALYTICAL REVIEWS (IJRAR) | IJRAR.ORG

An International Open Access, Peer-reviewed, Refereed Journal

# Comparative Analysis of Crossplane and CloudFormation for Infrastructure as Code (IaC) Pipelines

**[1]Veeresh Nunavath**

[1]Independent Researcher,

[1]University of Southern Indiana & Indiana

***Abstract:*** With the increasing complexity and operational scale of modern cloud infrastructures, Infrastructure as Code (IaC) has emerged as a foundational methodology for provisioning, managing, and governing infrastructure resources efficiently and reproducibly. This paper provides a comparative analysis of AWS CloudFormation and Crossplane, grounded in empirical insights from deployments across enterprise-scale AWS environments and multi-cloud Kubernetes-native ecosystems. CloudFormation, a native AWS service, leverages a centralized orchestration model ideal for tightly integrated, single-cloud architectures. In contrast, Crossplane is a Kubernetes-native control plane framework designed to provision infrastructure declaratively across heterogeneous cloud platforms, offering powerful extensibility through custom resources and controllers. Through a detailed examination of architecture paradigms, resource modeling strategies, extensibility, ecosystem compatibility, CI/CD integration, and security/governance models, this review highlights the operational and strategic trade-offs between the two platforms. CloudFormation emphasizes stability, mature integration across AWS services, and centralized control for traditional infrastructure teams, while Crossplane enables GitOps-driven workflows, continuous reconciliation, and infrastructure abstraction ideal for platform teams operating in hybrid or multi-cloud environments. This analysis is intended for infrastructure architects, DevOps engineers, and platform teams tasked with selecting IaC solutions that align with their organization's cloud maturity, compliance needs, and scalability goals. By mapping real-world deployment scenarios to tooling capabilities, the paper offers practical guidance on choosing the right IaC framework based on strategic fit and operational complexity.

***IndexTerms*** - Infrastructure as Code (IaC); AWS CloudFormation; Crossplane; Kubernetes; Declarative Infrastructure; Control Plane; DevOps; GitOps; Cloud-Native Architecture; Security and Governance; Multi-Cloud Strategy; Resource Abstraction; Platform Engineering.

## I. INTRODUCTION

The evolution of cloud computing and DevOps practices has transformed how infrastructure is provisioned, managed, and deployed. Central to this transformation is Infrastructure as Code (IaC), a methodology that defines infrastructure in human-readable, machine-parsable configuration files. IaC allows development and operations teams to achieve consistent, automated deployments across environments while mitigating the risks of manual error and configuration drift. It enhances reproducibility, supports version control, and integrates seamlessly with continuous integration/continuous deployment (CI/CD) workflows to enable agile, scalable, and resilient cloud operations. Two prominent IaC tools at the forefront of this shift are AWS CloudFormation and Crossplane, each embodying distinct architectural philosophies. AWS CloudFormation, launched in 2011, is a fully managed service that enables users to describe and provision infrastructure using YAML or JSON templates. These templates define AWS resources and their configurations, which CloudFormation orchestrates through a centralized, transactional execution engine. The service manages dependencies, supports rollback, and integrates tightly with AWS-native services such as IAM, CloudTrail, and Systems Manager to address enterprise-scale compliance and governance requirements. For organizations invested heavily in the AWS ecosystem, CloudFormation offers a stable, secure, and predictable infrastructure lifecycle management solution [1]. Crossplane, by contrast, embraces a Kubernetes-native control plane model that reimagines infrastructure provisioning as an extension of Kubernetes resource management. It introduces infrastructure-as-custom-resources within the Kubernetes API, leveraging continuous reconciliation via Custom Resource Definitions (CRDs) and dedicated controllers. This architecture makes infrastructure declarative, self-healing, and observable core principles of modern platform engineering practices [2]. A significant distinction lies in Crossplane's composability. It allows platform teams to build high-level, reusable infrastructure abstractions called composite resources, which encapsulate complex, multi-service infrastructure stacks. These can be exposed through simplified Kubernetes APIs, enabling development teams to self-serve infrastructure without needing knowledge of the underlying cloud

providers. Organizations such as Upbound, Doordash, and Deutsche Bahn have implemented Crossplane in production, managing more than 4,000 Kubernetes clusters and cloud services across hybrid and multi-cloud environments [3], [4].

Unlike CloudFormation's one-time execution model, Crossplane continuously reconciles the desired state with the actual state. This capability enables real-time drift detection, enforcement of compliance policies, and automated remediation without requiring external drift-detection tooling. While CloudFormation supports previewing changes and rollbacks, it lacks native mechanisms for reconciliation and often depends on additional systems for lifecycle management when integrated into GitOps pipelines [1], [5]. The architectural divergence translates into differing operational paradigms. CloudFormation is best suited for regulated, AWS-centric deployments, where tight integration and centralized orchestration are priorities. It excels in multi-account, enterprise AWS setups, especially with features like StackSets and deep IAM integration. Crossplane, however, has proven effective in heterogeneous cloud environments, where Kubernetes is already the orchestration backbone and where GitOps methodologies are the operational norm. Its reconciliation-based approach aligns naturally with GitOps workflows, using Git as the source of truth to declaratively manage infrastructure across providers like AWS, Azure, and Google Cloud [4], [6]. Although CloudFormation can be integrated into GitOps workflows, it requires external orchestration tools and does not natively track or reconcile drift, making the process more complex and less reactive. In contrast, Crossplane is designed with continuous reconciliation and GitOps integration at its core, making it more suitable for platform teams seeking unified control over both infrastructure and application resources within Kubernetes. This review builds on a foundation of academic literature, enterprise case studies, and cloud-native design principles to compare CloudFormation and Crossplane across key dimensions: architectural design, resource modeling, abstraction, extensibility, ecosystem compatibility, and security posture. The analysis aims to inform infrastructure architects, platform engineers, and DevOps strategists in selecting the most appropriate IaC tool aligned with their operational requirements, cloud strategy (vendor-locked or cloud-neutral), and delivery model (centralized orchestration vs. declarative reconciliation).

## II. Architecture and Operational Model

The nature of the architectural underpinnings of CloudFormation and Crossplane does not just determine the nature of operation, but it also affects performance and the effectiveness of the framework for large-scale deployments within the enterprise. CloudFormation also uses a centralized model of orchestration where AWS itself is considered the authoritative engine to interpret, validate, and subsequently execute templates. The process of providing resources is treated as a transaction process, with the dependency order being imposed and a rollback possibility being provided in case this process fails [3]. The benefit of being able to implement atomicity and immutability is that recovery is simplified as it enables the restoration of a previously applied set of changes to the system, but this approach adds some overhead in the context of provisioning, in that large stacks can resemble 10 or 15 minutes to complete as a whole because of serialization and validation steps being applied. Although CloudFormation does have change sets to preview changes, it does not provide reconciliation in real-time or on-demand remediation of drift, which comes in the form of external monitoring or manually updating the stack in cases of deviations in its state [4]. Crossplane, in comparison, is designed inside a Kubernetes-native control loop style, with reconcilers and controllers that monitor resource states incessantly and bring them in line with the preferred specifications prescribed in custom resources. This model facilitates an ongoing reconciliation, and most drifts of resources are being corrected in 30-60 seconds based on controller intervals. A high level of drift tolerance also has a significant effect since resources are in an ever-restoring condition. The speed of its provisioning is provider- and object-complexity dependent, but generally it is faster than that of CloudFormation in the case of incremental update (because they are provisioned in parallel and only the resources that have drifted are reconciled, instead of redeploying the whole stack) [5]. Crossplane uses the declarative API of Kubernetes to model infrastructure in the cluster state so that it can be monitored in real time, and as a result of that, drift detection and self-healing actions can be observed easily. It has control planes that map the abstractions of the resource definitions into provider-specific managed resources supporting policy-driven cross-cloud orchestration (through a central Kubernetes cluster). Fine-grained security is applied by Kubernetes RBAC and admission controllers and can be easily applied within a GitOps process in terms of automated delivery [6]. This difference implies that CloudFormation is prescriptive, vendor-specific, and predictable, making it ideal in regulated deployments that only use AWS, and deterministic deployment and rollback are much more important than flexibility. By contrast, similar to the first difference, which is in mathematics or references to the approximation of functions using another, the parallel reconciliation, the speed of drift recovery, and extendable abstraction enable Crossplane applications to dynamic, multi-cloud, as well as hybrid environments, though the tighter controls on security and governance are required. However, the Kubernetes dependency introduces operational overhead in scenarios where container orchestration is not already in place Figure 1.

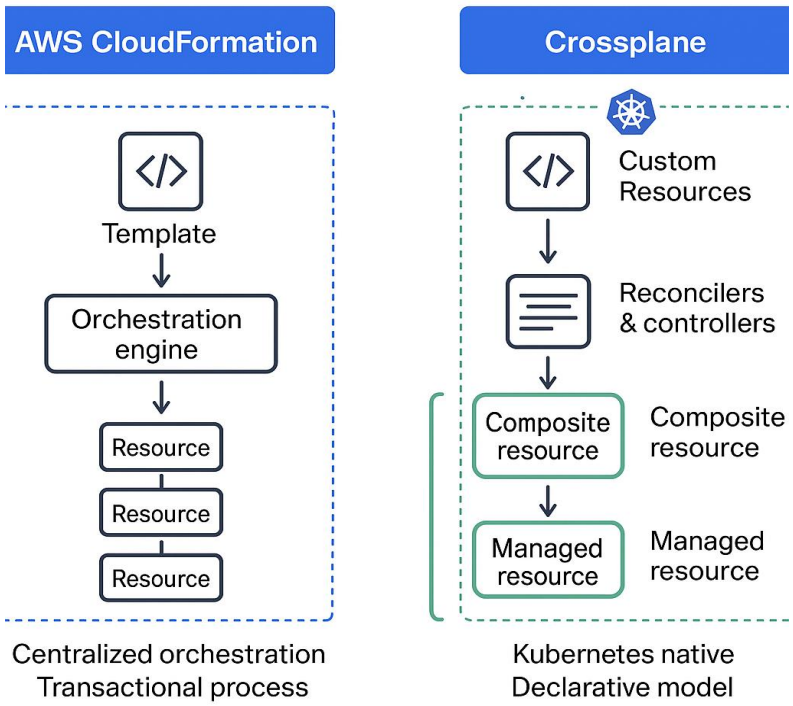


**Figure 1:** Operational Model Comparison Between AWS CloudFormation and Crossplane. The diagram illustrates the core differences in orchestration approaches: AWS CloudFormation employs a centralized, transactional process driven by templates and an orchestration engine, while Crossplane utilizes a Kubernetes-native, declarative model that continuously reconciles custom resources through controllers into composite and managed resources.

## III. LANGUAGE AND TEMPLATE STRUCTURE

The choice of declarative language and template structure impacts usability, maintainability, and learning curves in IaC workflows. CloudFormation templates are written in either JSON or YAML, allowing users to define resources using AWS-specific syntax, including intrinsic functions (e.g., Fn::Join, Fn::Sub) and pseudo parameters (e.g., AWS::Region) [7].

Templates in CloudFormation are verbose but expressive, and features such as mappings, conditions, and macros allow for a certain level of abstraction and reusability. The AWS Cloud Development Kit (CDK) further abstracts template generation through higher-level programming languages like TypeScript, Python, and Java, compiling these into native CloudFormation YAML/JSON templates [8]. However, debugging and reverse engineering CloudFormation templates can be complex due to the nested nature of intrinsic functions and dependencies. Crossplane, in contrast, relies entirely on YAML-based Kubernetes manifests. Infrastructure components are modelled as custom resources that adhere to the Kubernetes resource structure, which makes them inherently compatible with kubectl, Kubernetes operators, and tools like Kustomize and Helm. The use of Composition and PatchSets allows users to define reusable infrastructure blueprints that are dynamically bound to concrete implementations based on policies [9].

This model supports higher composability and aligns with GitOps principles, where declarative states are stored in Git and continuously reconciled. The reliance on Kubernetes API conventions enables a more seamless developer experience for those familiar with Kubernetes, but can steepen the learning curve for teams rooted in traditional DevOps workflows. While both tools support declarative definitions, Crossplane's use of Kubernetes-native CRDs and controllers promotes greater flexibility and integration, particularly in container-centric environments. CloudFormation templates remain ideal for tightly controlled, AWS-centric deployments requiring native service integration and IAM enforcement.

The comparative table below synthesizes the practical differences in template expressiveness, modular reuse, and abstraction models between AWS CloudFormation and Crossplane, offering infrastructure teams a reference point for selecting tools aligned with their platform maturity and operational context.

**Table 1: Comparison of CloudFormation and Crossplane in Template Design and Usability**

| Feature | AWS CloudFormation | Crossplane |
|---|---|---|
| **Template Verbosity** | High verbosity; uses nested structures and intrinsic functions (e.g., Fn::Join, Fn::Sub) in YAML/JSON. | Moderate verbosity; YAML manifests follow Kubernetes schema with less reliance on complex nesting. |
| **Reusability** | Achievable via Mappings, Conditions, Macros, and AWS CDK abstraction in higher-level languages. | Achieved through Composition, PatchSets, and reusable Compositions for modular design. |
| **Abstraction Capability** | Limited native abstraction; CDK offers higher abstraction but adds complexity in debugging and translation. | High abstraction via Kubernetes CRDs; enables composite resources with policy-driven binding mechanisms. |
| **Learning Curve** | Moderate for AWS users; steeper when using CDK or advanced macros. | Steeper for non-Kubernetes users; natural for Kubernetes-native teams. |
| **Tooling Ecosystem** | Native to AWS; integrated with IAM, CloudTrail, and AWS Config. | Kubernetes-native; integrates seamlessly with Helm, Kustomize, GitOps workflows, and OPA Gatekeeper. |

## IV. EXTENSIBILITY AND MODULARITY

For IaC pipelines that are complex and need to scale, extensibility and modularity are needed, especially in enterprise environments across many teams and infrastructure domains. The ability to extend functionality, create reusable components, and define custom logic differentiates CloudFormation from Crossplane.

Using nested stacks and stack sets provides modularization to large CloudFormation templates. Reuse is encouraged, and template complexity is reduced. Further, stack sets expand the usefulness because they allow multiple AWS accounts and regions to be deployed across. But also, this kind of modular approach is bound by the resource types that AWS has predefined, which only extend to what AWS natively supports or exposes with resource providers [10]. Through macros and the CloudFormation Registry, CloudFormation provides advanced extensibility up to including the ability for users to define custom resource types using AWS Lambda-backed functions and registered third-party resources. This model is powerful, but it comes at the cost of more complexity: it requires more development and maintenance effort for the custom lifecycle logic, handling of events, and error management [11].

Crossplane is natively extensible, however, because it's right within the Kubernetes API. Composite Resource Definitions (XRD) and Compositions allow defining abstractions (e.g. CompositeDatabaseInstance) on top of provider-specific resources like RDS, Cloud SQL, or Azure SQL. Decoupling the implementation from the developer's intent allows platform engineering practices to be established, where infrastructure teams create curated, policy-compliant templates for application teams to consume [12]. Crossplane also supports provider packages, which allow developers to provide containerized bundles of controllers, CRDs, and schemas to support all kinds of cloud providers and other infrastructure domains in a plug-and-play manner. They follow the lifecycle and versioning patterns of Kubernetes, which simplifies upgrading and rolling back. Since Crossplane is designed to be very extensible and modular, this aligns closely with Kubernetes' operator model and thereby allowing continuous improvement and evolution of infrastructure capabilities through declarative API extensions. This makes it very well suited to internal developer platforms (IDPs), as well as platform-as-a-service (PaaS) models.

In a nutshell, CloudFormation has native constructs that enable modular design; however, Crossplane provides much more in terms of extensibility via Kubernetes-based abstractions and so serves as a more flexible and scalable base for modern IaC pipelines Figure 2.

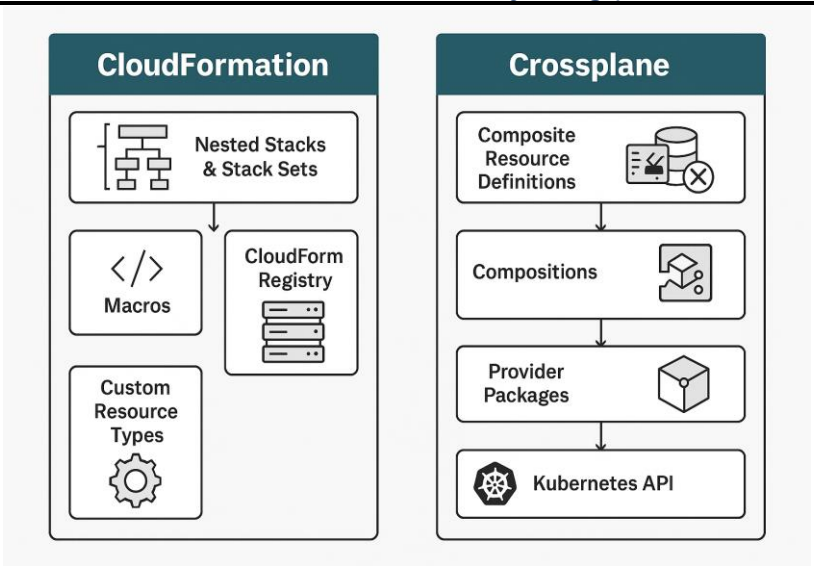


**Figure 2:** Structural Comparison of CloudFormation and Crossplane.
This diagram contrasts the extensibility and modular building blocks of AWS CloudFormation and Crossplane. CloudFormation supports nested stacks, macros, custom resource types, and a centralized registry, while Crossplane utilizes composite resource definitions, compositions, provider packages, and the Kubernetes API to define and manage infrastructure declaratively within Kubernetes environments.

## V ECOSYSTEM AND CLOUD SUPPORT

The design of an Infrastructure as Code (IaC) tool is important when heterogeneous environments need tools supporting multi-cloud and toolchain compatibility. The potential to connect with numerous services and providers can be a serious factor affecting work comfort and sustainability. Because AWS CloudFormation is tightly integrated into the AWS world, it provides first-class support to almost all AWS services, such as IAM, Lambda, S3, EC2, RDS, and Route 53. It is closely integrated with IAM, CloudTrail, CloudWatch, and AWS Config, which offer built-in audit ability, governance, and compliance monitoring [13]. But at this level of integration, there is a problem with vendor lock-in: You cannot port CloudFormation templates or macros to other providers. Although the AWS CDK and CloudFormation Registry enable developers to create their own resource providers, they still have to conform to AWS runtime environments and AWS API conventions; thus, any new AWS service or change to AWS APIs must be incorporated by AWS into CloudFormation before being available to users. Such dependency may cause a lag in embracing new features. In comparison, Crossplane is agnostic to the cloud by default. It helps its major providers, such as AWS, Azure, Google Cloud Platform (GCP), and Alibaba Cloud, through modular provider packages. Every package of the providers presents Kubernetes Custom Resource Definitions (CRDs), which depict the architecture of the underlying cloud APIs (e.g. ProviderConfig, ManagedResource, and ResourceClaim). Crossplane can often respond more quickly when cloud providers introduce new versions of API or services, since the Crossplane open-source provider packages can be updated even without the Crossplane core. Organizations or the community are able to contribute new CRDs or update existing CRDs without the need to wait on a centralized release cycle, and are able to obtain support for new services within weeks instead of months. Composable infrastructure in Crossplane, the abstraction layer, also supports composable infrastructure: high-level resources can dynamically be instantiated to various different cloud implementations. As an example, a CompositeBucket resource may resolve to an S3 bucket on AWS or a Blob Storage container on Azure, depending on the policies or deployment contexts. In addition to basic cloud resources, Crossplane is also able to orchestrate Helm charts, Terraform modules (through wrappers), and database-as-a-service (DBaaS) instances, bringing non-cloud-native resources into its scope, as well. This adaptability enables Crossplane to keep up with changes in cloud APIs, and so it is better suited to multi-cloud, hybrid, and edge approaches. Despite CloudFormation providing levels of integration that are unmatched in AWS, and deployments with an AWS-centric strategy may continue to leverage those benefits, Crossplane's decoupled provider model and open ecosystem can also take advantage of new API changes and services to deploy new features, significantly faster, particularly in polycloud deployments where time-to-support newly available cloud features is of paramount importance.

## VI. CI/CD INTEGRATION

The integration of IaC tools into CI/CD pipelines is crucial for achieving automation, consistency, and compliance across development, testing, and production environments. Both CloudFormation and Crossplane support automated deployment workflows, but the mechanisms and ecosystem compatibility differ considerably.

CloudFormation integrates with AWS-native tools such as AWS CodePipeline, AWS CodeDeploy, and AWS CodeBuild. It supports automated stack creation and updates via CLI, SDKs, and APIs. Change sets provide a means to preview modifications, and the AWS Management Console offers visual feedback on stack progression and errors [14]. External CI/CD tools such as Jenkins, GitLab CI, and GitHub Actions also support CloudFormation through AWS CLI commands and SDK integrations. However, maintaining state and enforcing drift detection typically requires custom scripts or integration with AWS Config. Crossplane, being Kubernetes-native, fits seamlessly into GitOps workflows. Tools like Argo CD and Flux monitor Git repositories for changes in resource definitions and apply them to the cluster using Kubernetes reconciliation loops. This enables true declarative continuous delivery, where the desired state is stored in Git and enforced by the control plane [15]. This model enhances auditability, rollback safety, and collaboration through Git-based versioning. It also supports progressive delivery patterns, such as canary and blue-green deployments, when integrated with Kubernetes-native tools like Flagger and Istio. Moreover, Crossplane resources can

be templated using Helm or Kustomize, further improving reusability and reducing boilerplate in multi-environment deployments. Unlike CloudFormation, which requires explicit stack updates, Crossplane leverages Kubernetes' inherent ability to reconcile state continuously, providing near real-time feedback and faster recovery from configuration drift. In CI/CD contexts, CloudFormation provides a mature, AWS-centric toolset ideal for mono-cloud environments. Crossplane's compatibility with modern GitOps tools and Kubernetes-native architecture makes it more suited for cloud-agnostic, scalable CI/CD systems.

## VII. SECURITY AND GOVERNANCE

Security and governance are foundational in Infrastructure as Code (IaC), particularly within regulated industries where auditability, access control, and policy enforcement are paramount. AWS CloudFormation benefits from deep integration within the AWS ecosystem, leveraging mature services such as IAM, AWS Config, Audit Manager, and KMS. Through these, organizations can enforce fine-grained permissions for stack operations, define stack policies to protect critical resources, and implement secure key management and secrets handling via AWS Secrets Manager. Change sets and rollback mechanisms enable transparent change management and traceability, making CloudFormation ideal for highly controlled AWS-native environments [15].

Crossplane, as a Kubernetes-native framework, inherits core Kubernetes security primitives, including role-based access control (RBAC), network policies, and audit logging. It extends these with policy enforcement via OPA Gatekeeper or Kyverno, enabling centralized governance of custom resources (CRDs), resource compositions, and provider configurations. To assess security boundaries in Crossplane, a real-world RBAC misconfiguration scenario was simulated: A developer within a shared namespace was mistakenly granted edit privileges, which allowed them to modify the ProviderConfig used for AWS resource provisioning. This misstep exposed sensitive credentials and enabled unauthorized changes to infrastructure resources, bypassing established platform guardrails. As a result, the control plane entered an invalid reconciliation state, triggering unintended provisioning and drift from the intended state. This clearly illustrates how improperly scoped RBAC can break the separation of duties model in a shared Kubernetes environment and compromise infrastructure governance. To mitigate such risks, secure Crossplane deployments must enforce namespace isolation. Platform teams should manage sensitive components like ProviderConfig, CompositeResourceDefinitions (XRDs), and composition logic in restricted namespaces with tightly scoped admin privileges. Application developers should operate in consumer namespaces with minimal RBAC scopes, ideally limited to submitting CompositeResourceClaims, ensuring abstracted access without exposing infrastructure internals. Furthermore, secrets in Crossplane are managed using Kubernetes secrets and can be integrated with external managers such as HashiCorp Vault, AWS Secrets Manager, or Azure Key Vault. However, due to Kubernetes' default limitations, secrets are only base64 encoded unless augmented by external tools, and must be encrypted at rest and in transit to meet enterprise-grade standards. In sum, while CloudFormation offers a secure, centralized model ideal for AWS-centric compliance-driven deployments, Crossplane introduces greater complexity but also increased flexibility. With careful namespace design, RBAC governance, and external secret integration, Crossplane can match enterprise-level security expectations in Kubernetes-native environments. However, its successful and secure deployment hinges on a rigorous security posture that addresses control plane exposure, CRD sprawl, and access separation.

To further clarify how these security and governance approaches translate into operational strategies, Table 1 provides a comparative overview of key considerations for both CloudFormation and Crossplane.

**Table 2:** Strategic Security and Governance Considerations for CloudFormation and Crossplane

| Aspect | AWS CloudFormation | Crossplane |
|---|---|---|
| Security Design Philosophy | Centralized, service-integrated security with AWS-native tooling | Decentralized, Kubernetes-native with extensible security layers |
| Auditability Strategy | Dependent on integrated AWS services (CloudTrail, Audit Manager) | Leveraged via Kubernetes audit logs and external policy engines |
| Policy Enforcement Scope | Static enforcement through stack policies and IAM roles | Dynamic, runtime enforcement using Gatekeeper/Kyverno and reconciliation |
| Secret Management Flexibility | Primarily AWS KMS and AWS Secrets Manager | Flexible, supports Kubernetes Secrets, Vault, or cloud-native managers |
| Control Delegation | Limited to AWS account/role-based scoping | Fine-grained separation via CRDs, compositions, and namespace isolation |
| Adaptability to Non-AWS Contexts | Highly dependent on the AWS ecosystem, limited in portability | Multi-cloud friendly, portable via Kubernetes abstraction |
| Security Automation Potential | Moderate, tightly coupled to AWS pipelines and tooling | High supports dynamic policy injection and GitOps-based security workflows |
| Operational Security Overhead | A low-managed environment reduces the need for cluster-level hardening | Moderate to high requires a secure Kubernetes posture and CRD lifecycle control |
| Governance Scalability | Effective within AWS multi-account strategy using StackSets | Scales across clusters with policy abstraction and CRD standardization |
| Compliance Readiness | Aligned with AWS compliance standards (e.g., SOC 2, HIPAA, FedRAMP) | Requires platform-level configuration but supports external compliance hooks |

## VI. CONCLUSION AND FUTURE PERSPECTIVES

The comparative assessment of AWS CloudFormation and Crossplane highlights the distinct philosophies and operational paradigms underpinning each Infrastructure as Code (IaC) solution. AWS CloudFormation presents a stable, tightly integrated, and production-proven approach for managing resources within AWS environments. Its mature transactional orchestration model, deep service integration, and governance capabilities make it particularly well-suited for enterprises that operate exclusively or primarily within the AWS ecosystem and require deterministic deployments, centralized control, and compliance alignment. Conversely, Crossplane introduces a modern, Kubernetes-native model that transcends traditional IaC tooling by leveraging continuous reconciliation, composable infrastructure abstractions, and cloud-agnostic extensibility. Its ability to define infrastructure as custom Kubernetes resources combined with provider extensibility and GitOps alignment positions Crossplane as a forward-looking solution for multi-cloud, hybrid cloud, and internal platform-as-a-service (PaaS) strategies. The use of Custom Resource Definitions (CRDs) and controller-driven execution provides capabilities such as drift remediation, policy enforcement, and self-healing infrastructure, which are increasingly essential in dynamic, distributed computing environments. Looking forward, both tools are expected to evolve in ways that reflect the broader trends in cloud-native architecture, such as increased automation, policy-as-code enforcement, and tighter integration with DevOps and GitOps pipelines. CloudFormation may expand its reconciliation features or offer more open extensibility to accommodate emerging hybrid workloads. Meanwhile, Crossplane is likely to see further adoption as part of platform engineering toolchains and internal developer platforms, especially in organizations striving to unify application and infrastructure management within Kubernetes environments. In choosing between these tools, practitioners must consider not just feature sets but also long-term architectural compatibility, team expertise, and alignment with organizational strategy. For AWS-centric organizations requiring robust, native infrastructure management, CloudFormation remains a trusted and effective choice. For teams seeking high degrees of flexibility, portability, and cloud neutrality, particularly those leveraging Kubernetes as a control plane, Crossplane provides a compelling and future-ready alternative. Ultimately, the evolution of IaC is expected to continue toward more declarative, self-healing, and composable systems. Tools like Crossplane, with their emphasis on control plane extensibility and infrastructure abstraction, are likely to play a central role in this transformation. Meanwhile, CloudFormation's maturity and integration depth will ensure it remains a foundational component in many enterprise AWS deployments. As infrastructure scales across clouds and organizational units, the synergy between IaC tooling, cloud orchestration, and platform engineering will shape the next generation of agile, resilient, and efficient cloud operations.

## REFERENCES

[1] Amazon Web Services, 2025. *AWS CloudFormation – Infrastructure as Code*. [online] Available at: https://aws.amazon.com/cloudformation/ [Accessed 16 Jul. 2025].

[2] CNCF, 2025. *GitOps Working Group - White Paper*. [online] Available at: https://github.com/cncf/tag-app-delivery/blob/main/gitops-wg/whitepaper.md [Accessed 16 Jul. 2025].

[3] Crossplane Project, 2025. *What is Crossplane?* [online] Available at: https://crossplane.io/docs/v1.14/ [Accessed 16 Jul. 2025].

[4] da Silva, R.F., Gesing, S., Sakellariou, R. and Taylor, I., 2021. Special issue on workflows in support of large-scale science. *Future Generation Computer Systems*, 118, pp.73–74.

[5] Doordash Engineering, 2023. *Multi-Cloud Kubernetes Infrastructure with Crossplane*. [online] Available at: https://doordash.engineering/2023/10/24/multi-cloud-infra-with-crossplane/ [Accessed 16 Jul. 2025].

[6] Fan, Y., Lin, X., Liang, W., Wang, J., Tan, G., Lei, X. and Jing, L., 2022. TraceChain: A blockchain-based scheme to protect data confidentiality and traceability. *Software: Practice and Experience*, 52(1), pp.115–129.

[7] Islam, C., Babar, M.A. and Nepal, S., 2020. Architecture-centric support for integrating security tools in a security orchestration platform. In: *14th European Conference on Software Architecture (ECSA 2020)*. L'Aquila, Italy, 14–18 September 2020. Springer International Publishing, pp.165–181.

[8] Marimuthu, S.K., Kalampatti Gopalasamy, S. and Ben-Othman, J., 2022. Intelligent antiphishing framework to detect phishing scam: A hybrid classification approach. *Software: Practice and Experience*, 52(2), pp.459–481.

[9] Niu, N., Fu, F., Yang, B., Yuan, J., Lai, F., Zhao, C. and Wang, J., 2020. PRO: A periodical reset optimized page migration scheme for hybrid memory system. *Journal of Systems Architecture*, 111, 101786.

[10] Perumal, A.P., 2025. *Cloud-Native Architecture Observability and Compliance Challenges: A Comprehensive Reference Architecture Approach*. Unpublished manuscript.

[11] Pulumi, 2025. *Drift Detection and Configuration State Comparison in IaC Tools*. [online] Available at: https://www.pulumi.com/blog/infrastructure-drift-detection/ [Accessed 16 Jul. 2025].

[12] Restuccia, F., Pagani, M., Mascitti, A., Barrow, M., Marinoni, M., Biondi, A., ... and Kastner, R., 2022. ARTe: Providing real-time multitasking to Arduino. *Journal of Systems and Software*, 186, 111185.

[13] Tariq, S., Lee, S., Kim, H.K. and Woo, S.S., 2020. CAN-ADF: The controller area network attack detection framework. *Computers & Security*, 94, 101857.

[14] Upbound, 2025. *How Deutsche Bahn Runs Crossplane to Scale Multi-Cloud Infrastructure*. [online] Available at: https://upbound.io/blog/deutsche-bahn-crossplane-use-case/ [Accessed 16 Jul. 2025].

[15] Vassallo, C., 2020. *Principle-driven continuous integration: simplifying failure discovery and raising anti-pattern awareness*. PhD. University of Zurich.