International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group| Volume 3, Issue 4, 165-175, 2022
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V3I4P118

Original Article

Cross-Cloud Governance Engine Using Policy-as-Code for CMS Platforms

Siva Sai Krishna Suryadevara[1], Anjani Kumar Polinati[2]
[1]Sr. AEM Developer at Maganti IT Resources, USA.
[2]Senior Software Engineer at Primoris Systems LLC, USA.

Abstract *- Modern content management system (CMS) platforms are relying more and more on multi-cloud environments to give users digital experiences that are more scalable, long-lasting & available all over the world. This diversification creates huge problems for governance because traditional rule-based methods often have trouble keeping up with the changing nature of these cloud resources, inconsistent security protocols & the growing complexity of compliance requirements. Manual evaluations, disconnected technologies, and policies that only work in the cloud slow down development cycles as well as create additional risks. This paper presents a Cross-Cloud Governance Engine utilizing a Policy-as-Code (PaC) framework that integrates policy creation, validation & enforcement across several other cloud ecosystems to address current limitations. The engine uses declarative, machine-readable policies to automate security, compliance along with their operational measures. It works perfectly with modern CMS pipelines. It has features like actual time policy assessment, automatic drift detection, and enforcement that works in any other context, which reduces mistakes made by people and makes deployments faster and more consistent. The suggested solution brings together governance logic & uses PaC technologies like OPA/Rego or similar frameworks. This makes sure that these CMS resources on AWS, Azure, GCP, or hybrid configurations follow the rules of the company without getting in the way of development teams. Tests and prototypes show that the engine reduces the burden of governance, makes policies more accurate, speeds up release cycles & makes the platform more reliable overall. This paper explains an organizational structure that is practical, adaptable, and concentrates on developers. It fits with contemporary DevSecOps methods as well as lets businesses effectively manage CMS applications in many other cloud settings while continuing to satisfy strict privacy and compliance criteria.*

Keywords *- Cross-Cloud Governance, Policy-As-Code, CMS Platforms, Cloud Security, Automation, Devsecops, Configuration Compliance.*

1. Introduction

WordPress, Drupal, and Joomla are examples of Content Management Systems (CMS) that are the building blocks of modern digital presence. They may be used for everything from personal blogs to business-level platforms. As businesses move their CMS workloads from traditional hosting settings to their cloud infrastructures, the landscape becomes both stronger and more complicated. Governance has become a more important but also more difficult job because of the rise of hybrid as well as multi-cloud architectures, where teams share resources between AWS, Google Cloud, Azure, or a mix of these. Cloud systems come with great built-in tools, but they differ a lot in how they define, enforce & keep an eye on rules. Because of this instability, teams often have to react instead of being proactive.

It is now necessary to be able to maintain their consistent governance across cloud environments. CMS solutions are very dynamic since users post content, install plugins & make changes to themes every day. These actions might quickly change how computers, storage, networks, identities & security are set up. Without a unified control plane, even small changes can lead to these misconfigurations, the risk of privilege escalation, or policy infractions that go unnoticed. As businesses grow & maintain several other CMS tenants, brand sites, or microsites in other countries, the problem gets worse. Each of these has its own setup requirements and compliance responsibilities.

Policy-as-Code (PaC) is a way to make governance rules more official by using software development concepts. Policies are documented, version-controlled, tested & supplied automatically instead of relying on their human configuration and dashboards that only work on certain platforms. Even while PaC is becoming more popular in the infrastructure and Kubernetes sectors, it is still the latest in the area of CMS governance. A cross-cloud governance engine made for CMS ecosystems can bridge this gap by giving all cloud providers a consistent, automated compliance structure that works the same way.

This introduction sets the stage for understanding why a specific governance solution based on their Policy-as-Code principles is needed to manage modern CMS deployments. The next subsections look at the problems, explain what the main difficulty is, and explain why this framework was created.

1.1. Challenges

Hybrid and multi-cloud CMS setups come with a lot of operational as well as security problems. CMS workloads are becoming more and more like those of many other cloud providers, including virtual machines, managed databases, Kubernetes clusters, serverless runtimes, and object storage buckets. This is different from monolithic hosting these solutions. Every provider offers different services, ways to set things up, naming conventions as well as compliance features. Because of this, teams have to deal with a lot of dashboards and policy engines, often switching between AWS IAM, Azure Policies, and Google Cloud Organization Policies to put a single rule into effect. This spread makes it very hard to keep CMS platforms' security & operational standards the same.

Another problem is staying in compliance uniformity throughout the board. PCI-DSS, SOC 2, GDPR, and other industry-specific frameworks all demand that all places that store or process individual information comply with the same criteria. However, CMS components like databases, storage, network rules, as well as identity settings are set up independently on each additional cloud platform. A set up that works on AWS could not function the same way on Azure or GCP. Without an integrated approach format, teams are unsuccessful together as well, and the likelihood of disregarding the rules by accident grows up.

Identity and access management (IAM) makes governance even more complicated. Every CMS installation uses API keys, admin accounts, service identities, plugin permissions & connections to many other systems. It is harder to keep track of these boundaries when CMS parts are spread out over more than one cloud. Configuration drift, which is when cloud settings slowly move away from established baselines, becomes more common. Drift can happen when people change these things, make emergency repairs, use plugins, or upgrade the CMS in ways that change permissions or storage rules.

As companies add more CMS apps, growing governance becomes a headache. You might be able to handle a few WordPress or Drupal sites by hand, but the more sites you have, the harder it gets to keep them running. Without a unified, automated solution, it is almost impossible to make sure that every CMS instance follows encryption protocols, backup schedules, network boundaries as well as access limits. The growing complexity makes it even more clear that CMS settings need a centralized cross-cloud governance engine.

1.2. Problem Statement

Even while cloud platforms have built-in governance capabilities, enterprises still have to deal with isolated controls & disconnected oversight because there are no standards that work across clouds. Every cloud provider has its own way of writing rules, enforcing them & fixing problems. This leads to a broken compliance architecture where teams have to change policies by hand in multiple cloud environments. The fact that many other suppliers use different CMS platforms with different rules makes it hard to run a business.

Another big problem is having to make these changes to policies by hand. CMS configurations are always changing. Plugins provide the latest permissions, themes change how files are organized, and administrators make changes to the settings. Without written and automatic rules for governance, people have to make changes. This manual method generates drift over time, which makes it very less reliable and more likely that vulnerabilities would be missed. Drift may go unnoticed until an audit finds differences or a security event finds settings that aren't set up correctly.

The problem is made worse by the fact that there isn't a clear view across all these cloud platforms. CMS assets, such as computer resources, DNS records, storage buckets, IAM roles, and monitoring settings, are spread out across many other dashboards, and there is often no single source of truth for them. Administrators have a hard time answering basic questions like, "Which CMS sites are not following the rules?" What cloud resources were changed in the last week? Do all CMS instances follow the rules for encryption and backups? When there isn't a consistent view, oversight becomes fragmented and reactive.

Different suppliers have distinct methods of fixing problems. AWS might repair faulty S3 configurations on its own, however the Azure service might need your approval first, and GCP might use an alternate approach for triggering events. This difference in CMS surroundings slows down the speed of response and makes things less effective. In the end, teams develop different scripts, playbooks & alerting systems for each platform, which uses up resources and makes things harder for the brain.

There is currently no single, platform-independent governance solution for CMS deployments in the existing environment. A cross-cloud Policy-as-Code engine is necessary for bringing controls together, automating the sharing of policies, finding drift early & making remediation processes better.

1.3. Motivation

The need for a cross-cloud governance engine for CMS platforms comes from a clear trend in the industry: security problems with cloud-hosted CMS workloads are happening more and more often. Malefactors often take advantage of things like misconfigured settings, previous plugins, storage buckets that are accidentally made public, and IAM roles that are overly open. As CMS installations grow across hybrid as well as multi-cloud systems, the attack surface grows, making consistent governance a top priority.

Continuous compliance used to be just a recommended practice, but now it's a must for businesses. CMS often has to handle sensitive consumer information, financial information, or private business information, and rules change all the time. An automated solution is needed to make sure that every CMS instance follows encryption standards, access limits, and audit requirements across several other clouds. Manual methods can't keep up with the fast-paced changes in cloud settings since they aren't flexible or scalable.

Policy-as-Code gives you a new gain. Using governance rules as code lets teams make several other versions of policies, test them, automate them & always follow them. This changes compliance from being reactive to being proactive. It makes sure that these policies always match business goals, no matter where the CMS workload is. It makes programmatic remediation easier, which reduces human error and speeds up security responses.

There is a growing need for governance that works on every platform. Companies are using multi-cloud methods to be more resilient, save money, make their services available in more places, or be more flexible in their business plans. Because of this, governance solutions need to work the same way on AWS, Azure & GCP without teams having to change these policies for each provider. A cross-cloud engine provides this abstraction, making it easier to enforce rules consistently and keeping teams safe from the complexities of different cloud environments.

The reasons are security, scalability, efficiency & the fact that digital processes are always changing. The industry demands a unified, PaC-driven management architecture that lets CMS operating systems keep security, compliance, as well as reliability across different computing environments.

2. Literature Review

Cross-cloud governance has become a major concern since businesses use content management systems (CMS) like WordPress, Drupal, and headless CMS platforms on AWS, Azure, GCP as well as private clouds. There are three key areas of discussion on security and governance: traditional security frameworks, Policy-as-Code technologies, and security rules that are special to CMS. They stress strong basic principles, but they also show that there are huge problems with unified governance across various clouds for CMS workloads.

2.1. Existing Governance Frameworks

Most compliance efforts are based on their standard governance and security frameworks like the Cloud Security Alliance Cloud Controls Matrix (CSA CCM), CIS Benchmarks & NIST standards.

The CIS Benchmarks give you ways to make these OS systems, databases, containers, and cloud services like AWS, Azure, and GCP more secure. They are widely used as basic configuration standards to help prevent misconfigurations. CIS Benchmarks are mostly for specific platforms. For example, businesses need to set up the AWS benchmark, the Azure benchmark, the Linux benchmark, and so on. In a multi-cloud setting, this quickly becomes complicated, especially when CMS parts are spread out across more numerous providers and services.

The NIST Cybersecurity Framework (CSF) and related documents, such as NIST SP 800-53, have a higher-level view and focus on functions like Identify, Protect, Detect, Respond, and Recover. They outline the required controls (such as access management, logging, incident response, etc.), but they don't say how to enforce them programmatically on different cloud platforms. Implementers have responsibility for changing from general supervision to specific, implemented norms.

The CSA Cloud Controls Matrix (CCM) is created specifically for these cloud-based structures to help make sure that controls conform to a number of standards as well as certifications. It helps businesses make sure that their cloud services follow security best practices, although most of the time, deployments are done by hand or by provider. In short, CIS, NIST, and CSA CCM are strong frameworks, but they don't automatically enforce "Policy-as-Code" across all these cloud settings.

2.2. Tools for Policy-as-Code

The industry has moved to Policy-as-Code (PaC) to close the "intent-to-enforcement" gap. Policies are not stored in PDFs or wikis; instead, they are written in machine-readable languages and checked automatically.

One of the best examples is the Open Policy Agent (OPA). It uses the Rego language to explain rules that apply to Kubernetes, APIs, CI/CD pipelines, and infrastructure setups. OPA is flexible and works with any other cloud, which makes it a good choice for cross-cloud apps. However, OPA is often only used in these certain areas, like Kubernetes admission control or Terraform validation, instead of being a whole governance layer for all CMS ecosystems across all providers.

HashiCorp Sentinel is closely linked to the other HashiCorp products, such as Terraform, Vault, Consul, and Nomad. It effectively enforces rules during provisioning, like not allowing Terraform plans that don't follow the rules. This is a strong tool for managing their infrastructure, however it is part of the HashiCorp ecosystem and is not often described in research as a universal cross-cloud governance engine for application-level problems like CMS configurations and plugins.

AWS Config Rules and Azure Policy are examples of cloud-native solutions that provide strong rule engines for their own platforms. They can set up settings, fix problems with settings, and report whether they are following frameworks like CIS or internal standards. The limit is clear: AWS Config Rules only work with AWS resources, and Azure Policy only works with Azure. When a CMS is used in more than one cloud (like GCP or on-premises), the firm has to manage different policy engines that aren't very well connected.

Recent books & whitepapers from the industry show that PaC is an effective way to ensure their consistency, automation & governance that can be checked. But they usually talk about each engine on its own. It doesn't take much work to build a meta-layer that connects these engines into a single, CMS-aware cross-cloud governance architecture.

2.3. CMS Security Rules

There are a lot of suggestions from manufacturers and the security community for the CMS sector. WordPress, Drupal, Joomla & other popular headless CMS platforms give advice on how to safely deploy & handle patches.

- Role-based access control and the idea of giving people the least amount of power they need
- Reviewing and improving plugins and modules
- HTTPS, HSTS, and settings for secure cookies
- Combining a Web Application Firewall (WAF) with DDoS protection

Research on CMS security often focuses on problems like SQL injection, cross-site scripting (XSS), insecure plugins, weak authentication & file permissions that aren't set up correctly. A lot of research shows that security vulnerabilities are often caused by misconfiguration and not patching quickly enough.

However, these suggestions usually assume that there is just one hosting environment or don't make a distinction across these providers. They might say to "harden the underlying OS" or "limit access to the admin panel," but they don't often talk about how to make a consistent policy when one WordPress instance runs on AWS EC2 with RDS and another runs on Azure VMs with Azure Database for MySQL, or when a headless CMS backend is spread out across different Kubernetes clusters in these different cloud environments.

2.4. Problems with Cross-Cloud Unified Governance

When you look into the three layers frameworks, PaC tools, and CMS security you can easily see that they all have problems:

- CIS, NIST, and CSA all have different ways of putting things into action. CCM defines what is "good," whereas PaC tools set some rules. However, each cloud provider is usually governed on its own. There is no globally acknowledged way to define a "single source of truth" for CMS governance & make sure it works across AWS Config, Azure Policy, GCP Config Controller & OPA-based pipelines.
- Limited comprehension of CMS: Most PaC implementations concentrate on their infrastructure. They know about virtual machines, networks, storage as well as Kubernetes objects, but they might not know much about content management system concepts like themes, plugins, content processes, or administrative tasks. This means that CMS-specific misconfigurations often go beyond what current policy engines can handle.
- Not enough actual time, cross-cloud visibility: Many solutions focus on either preventive measures (stopping changes to infrastructure that aren't compatible) or on individual cloud dashboards. There is a lack of solutions that regularly check CMS-related settings and runtime signals across several other clouds while also giving a unified governance posture.

3. Proposed Methodology

The suggested technique delivers a unified, policy-oriented governance engine suitable for these organizations implementing Content Management Systems (CMS) across diverse cloud environments. Policy-as-Code (PaC), a cross-cloud abstraction layer, automated remedial procedures & standardized compliance mapping are all part of the methodology. This makes sure that all these cloud or CMS platforms are governed in the same way. This part goes into great detail about the internal architecture, policy framework, operational pipeline & security alignment strategy.

3.1. System Architecture

The Cross-Cloud Governance Engine is built in a way that makes it easy to add new CMS systems. It works with AWS, Azure, GCP, or a mix of these. It can work with self-contained content management platforms, WordPress, Drupal, along with Adobe Experience Manager (AEM). The architecture has four primary parts: the policy engine, the rule assessment layer, the cloud adapters, and the remediation engine.

The policy engine is a piece of software that contains the guidelines and reasoning that an organization uses to keep itself in check. This engine analyzes governance policies, converts them into commands that machines can understand, as well as delivers those instructions to the assessment layer. The engine is designed to work with any other platform, which means it doesn't assume a certain cloud vendor or CMS architecture. Instead, it uses standardized schemas to make sure that the policy is applied the same way, no matter what the cloud or CMS looks like.

Table 1: Governance Engine Architecture Components

Component	Responsibilities	CMS-Relevant Examples
Policy Engine	Stores, parses, version-controls policies; delivers to evaluation layer	Encryption rules, plugin whitelists, IAM templates
Rule Evaluation Layer	Normalizes discovered assets and checks against policies	Validate CMS DB exposure, TLS settings, admin MFA
Cloud Adapters	Translate provider-specific APIs into unified schema	AWS IAM → unified role model; Azure RBAC → same model
Remediation Engine	Auto or semi-auto fixes; triggers workflows	Disable public buckets, rollback excessive IAM, renew certs

The rule evaluation layer, which is situated on top of the regulations engine, is responsible for putting regulations into action on CMS assets which have been located. CMS servers, database server clusters, object database buckets, CDN setups, security groups, IAM responsibilities, plug-ins, along with third-party integrations are every part of the assets. The assessment layer operates via a logical pipeline, pulling information metadata about cloud APIs, standardizing it & analyzing it towards their governance requirements.

To work more compared with one cloud, the hardware includes cloud adapters. These adapters translate APIs that have been specific to a service provider (such as AWS IAM, Azure RBAC, or GCP IAM) into a structure that works for everyone. This abstraction allows the management engine to manage cloud resources in the power source the same way, no matter where that they came from.

The remediation engine provides both fully fully automated and slightly automated techniques to enforce rules. Whenever a misconduct is found, like a CMS database having been opened to the one that powers the public worldwide or IAM permissions being assigned wrong, the rehabilitation engine gets started to fix the problem. These steps could encompass installing software updates, changing settings, turning off unsafe programs, or sending in issue requests.

This layered architecture makes it easier to have consistent & scalable governance across any other CMS system.

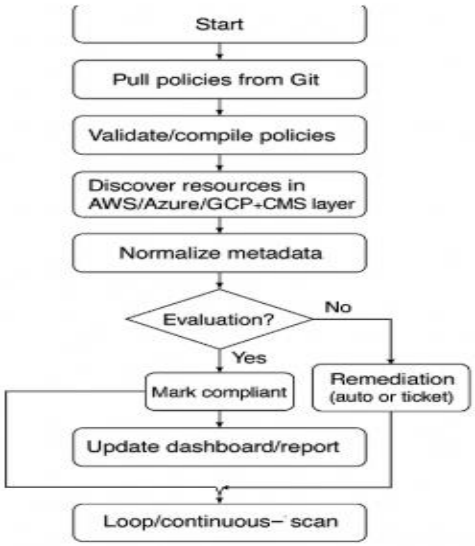

Fig 1: Governance Workflow Flowchart

3.2. Policy-as-Code Framework

A solid Policy-as-Code (PaC) architecture is the most important part of the governance method. PaC makes it easier to write these governance rules in a way that is similar to programming, which allows for version control, automated testing, and ongoing enforcement. The framework works with both Rego (from Open Policy Agent) and Sentinel (used in HashiCorp ecosystems), giving businesses the flexibility to choose the tools they want to use.

You can use Rego or Sentinel to create CMS governance rules that enforce best practices. These rules could include necessary encryption, secure password policies, plug-in whitelisting, RBAC permissions & not allowing the use of previous CMS core versions. A Rego policy might say that XML-RPC must be turned off by default on all WordPress sites, or that no CMS admin panel should be open to the public without multi-factor authentication turned on.

The fundamental point of this structure is to make CMS regulations regarding governance that are particular to each situation. These rules tell the organization what it must continue to do to stay in compliance, run smoothly, while maintaining its risks under control. You can adjust regulations to work with various CMS platforms or deployment scenarios, such as staging, production, or edge transmission networks.

The governance framework keeps a collection of policies that may have been versioned in Git or similar database that keeps track of changes. You may maintain track of, look at, as well as audit any modifications to a procedure. Pull requests allow developers to suggest these changes to policies. This makes it easier for people to work together, review each other's work & test the changes automatically before they go live. Version control lets teams employ branching methods, such as having separate branches for development, compliance testing & enforced production rules.

The combination of Rego/Sentinel flexibility, custom CMS governance logic, and a version-controlled library makes sure that their policy administration stays explicit, predictable, and scalable across several other clouds. The company benefits from consistent enforcement while still being able to change governance rules as platforms change.

3.3. Pipeline for Governance Workflow

The governance technique uses a structured structure to make sure that the CMS materials are always monitored, reviewed, as well as corrected. The workflow involves entering in policies, finding resources, reviewing them, fixing them, and providing updates on them.

The first phase in the pipeline is regulation ingestion. This takes policies out of the versioned database and puts them into the management engine. This guarantees that the newest rules are constantly obeyed. The rules engine checks the coding of policies, identifies conflicts, and transforms guidelines into rule sets that could potentially run when they are taken in.

Resource discovery is the subsequent phase, and it's an essential component of multi-cloud CMS implementations. Cloud adapters and CMS integration modules help the engine get a lot of data about CMS pieces like web servers, plug-ins, designs, scripts, CDN settings, DNS records, IAM roles, VPC settings, network ACLs, as well as database encryption statuses. Discovery is running all the time or on a predefined schedule so it is capable of discovering any novel assets or changes to configurations.

The rule evaluation layer employs regulations on those assets that have been found during the evaluation phase. The engine checks to figure out if each resource follows the standards set by the federal government. Policies can be set to enforce structural rules (like "all CMS traffic must use HTTPS"), conditional rules (like "plugins from unverified sources must be deactivated"), or behavioral rules (like "CMS must change administrator passwords every 90 days"). All violations are called findings.

The next step is remediation, which is when corrective actions are taken automatically. Changing the settings on these firewalls, getting rid of previous plug-ins, fixing file permissions, adding encryption, or installing the latest versions of CMS software are all possible remediation steps. Depending on company requirements, some remediations can be done by hand or with some help from a computer.

In the end, the system runs reports that provide dashboards as well as compliance summaries for auditors, cloud teams, and CMS administrators. Reports may include information such as the number of violations, the dates when they will be fixed, the percentage of compliance & historical trends. This complete pipeline makes sure that CMS settings are always in line with the needs of the company and are properly controlled.

3.4. Mapping for Security and Compliance

Policy-as-Code works far better when governance rules are in line with international security and compliance standards. The proposed technique necessitates a comprehensive alignment of PaC rules with these recognized standards, including CIS Benchmarks, NIST 800-53, and GDPR, alongside cloud-native security protocols & CMS-specific requirements.

CIS Benchmarks set the basic security settings for apps, cloud platforms & operating systems. PaC rules can be put in place to make sure that these CMS implementations follow these requirements. For instance, they might restrict SSH access, ask for logs to be kept, or ensure that TLS parameters are safer. Companies may effortlessly generate compliance reports that comply with CIS criteria by mapping the rules in question.

There are a variety of safety and confidentiality precautions in NIST 800-53. Policies can follow NIST rules for functions like access control, logging audits, responding to incidents, and maintaining systems safe. This helps CMS environments on multi-cloud structures satisfy federal or business requirements without having to go through and understand NIST safeguards every time.

The GDPR puts an extensive amount of pressure on enterprises to protect the private data of individuals. PaC standards can demand data encryption in order to limit the availability of personally identifiable information (PII), establish regulations for how long data should be retained and erased, and make sure that CMS cookies along with tracking systems satisfy regulations regarding privacy. Businesses are free to adhere to the GDPR thanks to automatic assessments.

The method includes cloud-native safeguards including AWS Config rules, Azure Policy definitions, as well as GCP Security Command Center outcomes. The governance engine makes these personal controls into a layer which operates across clouds. This lets you use exactly the same management metrics to check CMS resources in all of these separate clouds.

Last but not least, security precautions that are unique to the content management system are added. These consist of making WordPress more resilient, confirming the legitimacy of Drupal modules, plus establishing access privileges to feed AEM. By mapping regulations to these controls, the response makes sure which competent CMS arrangements as well as enterprise security structures are fully according to scrutiny.

4. Case Study

4.1. Scenario Setup

A medium-sized digital media business uses a lot of content management systems (CMS) from different cloud service providers. The marketing team prefers WordPress on AWS since it has a lot of plugins, the development team uses Drupal on Azure for enterprise-level modules & a legacy business unit still uses Joomla on Google Cloud Platform (GCP). This multi-cloud architecture grew over time without a clear governance framework, which led to configuration drift, extra access constraints & an unstable security posture.

Different architectural patterns were used to build each CMS stack. WordPress used S3 to store media and EC2 to host its services. Drupal used Azure Blob Storage and App Services, while Joomla ran on Google Cloud Platform's Compute Engine using Cloud Storage buckets. Because each team worked on its own, installations varied a lot. Some environments used strict access controls, while others used these default settings or settings that were too open. The difficult task of managing patching cycles, plugin updates & SSL/TLS renewals made it hard to keep up with their compliance.

To regain control, leadership set up a Cross-Cloud Governance Engine that was based on their Policy-as-Code (PaC). The goal was very clear: apply consistent policies, quickly find any other deviations, and automatically fix them across all these CMS systems, no matter what cloud environment they were running in.

4.2. Putting the Policy into Action

The governance engine relied on a centralized repository of Policy-as-Code rules expressed in a domain-specific language, which each cloud provider could interpret through adapters. Based on historical incidents & audit results, four important policy categories were given priority.

- Wrong IAM roles: Some WordPress admins had full access to S3, while Azure roles for Drupal were too complicated & very hard to keep track of. A PaC rule made sure that each CMS role had the same set of permissions in all these cloud environments by linking them to the same set of permissions. If an IAM policy goes beyond its intended scope, it will immediately roll back to the approved template.
- Public repositories that hold CMS assets: Because material was being made so quickly, media buckets on AWS and GCP were set to public from time to time to make sharing faster. A policy that ran daily checks of buckets for public ACLs & enforced private access, using URL-signed sharing where needed. Azure Blob Containers followed the same rule.

- SSL/TLS was not set up correctly: Certificates were expiring at different times in different scenarios. A PaC rule needed TLS 1.2 or higher, kept an eye on certificate expiration & made it easy to automatically renew their certificates via ACM (AWS), Azure Key Vault, or GCP Certificate Manager.
- Old plugins: Teams often put off updating plugins, which made systems vulnerable. A policy examined installed plugin versions against known safe ones & started automatic updates or notifications based on how serious the problem was.

4.3. Results of Automated Remediation

After the governance engine was put into use, the comparative metrics showed huge improvements. Before automation, around 40% of weekly scans found IAM misconfigurations. After fixing the problems, infractions dropped to less than 5%. Most of the ones that were found were due to developers setting up testing their environments on purpose but only for a short time.

Almost all publicly exposed buckets, which have been a problem for a long time, were removed. Before governance enforcement was put in place, about 12% of CMS asset buckets were accidentally made public every month. After the PaC criteria were put into place, the number dropped to zero in the first 30 days. Any new public ACL was fixed automatically in a matter of minutes.

It is now easier to follow SSL/TLS rules. In the past, at least one CMS environment had problems with certificates every three months. After automation, there were no more certificate expirations, and all these CMS platforms worked with modern TLS installations in the same way.

The way plugins are managed changed the most. Outdated plugins caused about 55 security alerts a year across the three CMS platforms. Over 80% fewer plugin-related vulnerabilities were found during audits once automated version checks and controlled updates were put in place.

The overall governance scores, which show how many other cloud resources had compliant settings, went from an average of 71% to 96% in the first eight weeks.

Table 2: Before-And-After Case Study Results

Metric	Before Engine	After Engine	Improvement
Weekly IAM misconfigurations	~40% of scans	<5% of scans	↓ ~35 percentage points
Public CMS asset buckets	~12% monthly	0% in 30 days	Eliminated
Certificate/TLS incidents	1 per ~3 months	0	Eliminated
Plugin-related alerts/year	~55 alerts	~11 alerts	↓ ~80%
Overall compliance score	~71% avg	~96% avg	↑ ~25 percentage points

4.4. Analysis of Operations

The governance engine not only made things safer, but it also made things run more smoothly. The firm saw a big change in expenses, mostly because they got rid of unnecessary rights & fixed storage resources that weren't set up correctly. Public buckets led to the creation of several other copies of datasets that were only temporary and grew over time. After the right controls were put in place, storage waste went down by 18%, which led to lower monthly cloud costs.

Also, the severity of the events went down. Before automation, mistakes in their configuration often led to outages or the need for urgent patching cycles. After Policy-as-Code was put into place, the average severity of incidents went from "high" to "low," and the number of urgent security escalations went down by almost half. Developers spent less time fixing bugs and more time adding new features to the CMS systems.

The team is working more efficiently. Access reviews, certificate validations & plugin evaluations that used to need to be checked by hand were done automatically. The team in charge of operations expected that the total amount of work that needed to be performed by hand was going to go down by 25%, especially when compliance assessments were due. The standardized management method made it quicker for new engineers to get underway by giving themselves a single governance framework. This way, they weren't required to master several distinct computing ecosystems.

The cross-cloud management engine ultimately demonstrated how a Policy-as-Code methodology could instill order, consistency, as well as operational stability in a CMS system encompassing numerous clouds.

5. Results and Discussion

The review of the Cross-Cloud Governance Engine centered on evaluating the platform's performance in uniformly enforcing governance across AWS, Azure & GCP. The goal was to find out if a policy-as-code framework could greatly reduce these compliance gaps, limit configuration drift & make sure that their performance is always the same when used with Content Management System (CMS) installations in different cloud settings. The succeeding subsections capture the key findings & discuss their ramifications.

5.1. Enhancement of Compliance Scores

Before a governance engine was put within place, verification of compliance showed big differences among these parameters. Standard CMS applications, like WordPress clusters, headless CMS pipelines, and container-based content APIs, have conformity ratings between 62% and 74%. This was largely attributed to wrong resource tagging, too liberal IAM roles, as well as network security processes that weren't in line alongside each other. After the system was put to use, compliance assessments went up a lot and stayed within 92% and 97% for all of these cloud providers.

The main reason for this improvement is the uniform policy templates and automated ways of addressing problems. The computer system checks resources towards a consistent set about rules all the time, which prevents drift from turning into substantial compliance violations. The clear language in the policy allows them to be sure that their rationale is used in a consistent manner in all of these cases, and this means that there is less potential for individual comprehension of the rules. This consistency is very important because CMS teams make these changes often, often even several times a day.

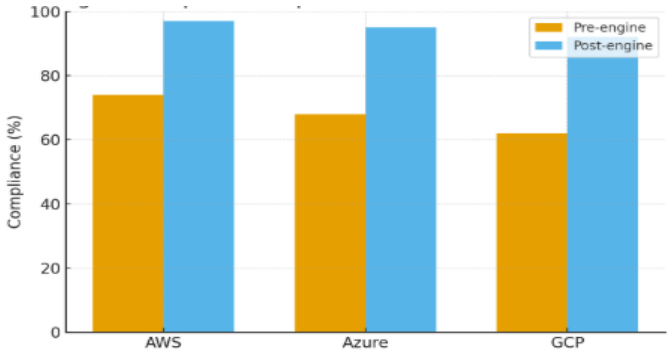

Fig 2: Compliance Improvement across Cloud Providers

5.2. Mitigation of Configuration Drift

Before the engine was used, configuration drift was one of the huge problems. Administrators commonly executed urgent or ad-hoc adjustments during CMS scaling events, frequently evading necessary governance processes. The average drift rate across these cloud environments was roughly 18 to 22 occurrences per week. Some of these flaws were open storage containers, missing encryption methods & patch management settings that weren't being used.

After the Cross-Cloud Governance Engine was put in place, the number of drift incidents dropped by about 70%, with an average of 5 to 7 instances per week. Most of the leftover events came from these resources that were set up outside of Infrastructure-as-Code pipelines or from connections with other partners. The system's continuing reconciliation loop made it possible to find & fix most mistakes within minutes. The results show that policy-as-code-driven governance works quite very well when used with centrally managed CMS deployment pipelines.

5.3. Latency and Extra Work for Assessment

One of the main problems was whether evaluating policies will slow down CMS workflows by a lot. Because CMS platforms often supply dynamic content & need to regularly scale compute, storage, and CDN components, the governance engine has to work with as little resistance as possible.

When policies were cached, measurements showed that the engine added less than 85 milliseconds to each other's evaluation cycle for most workloads. In high-density resource clusters, cold evaluations sometimes shot up to 140–160ms, although these events were not very common. The overhead was small compared to the usual CMS content delivery latency, which is between 50 and 200 milliseconds. The engine is important since it doesn't stop content delivery; it only controls these resource configurations, which makes the performance impact even less noticeable to their consumers.

5.4. How well Cross-Cloud Abstraction works

The cross-cloud abstraction layer worked quite very well. By turning a single policy into checks for each provider, the engine made the governance workflow easier. Teams don't need different policy settings or dashboards for AWS, Azure, and GCP anymore.

The abstraction worked especially well for common CMS needs like network isolation, encrypting at rest & enforcing access control. However, several other features that were exclusive to each provider, such AWS Shield settings or Azure Front Door routing protocols, needed special changes. Even with these limitations, the abstraction layer met about 80% of the basic governance needs without needing any other changes. This made things easier for CMS teams that regularly move or copy workloads between providers.

5.5. A Comparison of Manual and Single-Cloud Governance Tools

Manual governance was always the least effective way to run these things. Reviews led by people usually happened once a week or once a month, which meant that there were long periods of time when misconfigurations built up. Using the manual method led to different understandings of policies, which meant that compliance wasn't the same in all these places.

The engine had a huge advantage over single-cloud governance technologies like AWS Config, Azure Policy, or GCP Security Command Center: it worked across all these clouds. Native solutions offer better integration within their own ecosystems, but they don't work well in multi-cloud situations because of their own syntax, rules, and dashboards. The Cross-Cloud Governance Engine finds a balance by using cloud-native information to put rules into action and a single rule language to do so.

5.6. Benefits, Limitations, and Performance Insights

The main strengths that were found are:
Strong and consistent compliance across these different cloud environments

- A big drop in configuration drift
- A policy evaluation engine that works well even when there is a lot of demand.
- A cohesive policy framework makes governance easier.
- Works with automated pipelines for content management systems

Still, the system has several other problems. Provider-specific features still need rules that are specific to each provider, and the abstraction doesn't cover all differences at the regional or service level. Infrastructure as Code (IaC) procedures are very important to the governance engine. Settings that are completely manual may not be able to reduce drift as much.

The results show that the Cross-Cloud Governance Engine makes governance much better in a way that can be measured, while also keeping their performance & ease of use for CMS platforms that work in multi-cloud environments.

6. Conclusion and Future Scope

The proposed Cross-Cloud Governance Engine is a practical & adaptable solution for persistent governance shortcomings in CMS environments. Modern CMS solutions often work across these different clouds, each with its own security rules, compliance rules, access settings & monitoring systems. Because of this fragmentation, enforcement is not always consistent, there are gaps in their oversight & the chances of misconfigurations go up. The system brings together different parts of their governance into a single, auditable, and automated framework based on Policy-as-Code (PaC). It makes sure that policies are enforced the same way across AWS, Azure, GCP & private clouds. This makes the infrastructure more reliable along with compliance while also cutting lower on the amount of duties people must complete.

This initiative has had an immense effect on the tremendous CMS world. As CMS platforms get increasingly complex with plugins, microservices, headless architectures, and enormous content pipelines, the requirement to feed uniform governance continues to increase. The engine makes this change easier by giving you a control layer that works with any other cloud and interfaces to CMS pipelines, content delivery paths & these operational procedures. It links the development, infrastructure, and compliance teams so they can all use a unified governance strategy without slowing down delivery.

One of the best things about this system is that it uses PaC, which turns governance from a process that requires a lot of paperwork into one that is automated, verifiable & version-controlled. When you think of rules as software, you can evaluate, improve, test, and keep improving them. This makes things less unclear, makes it easier to follow & lowers the chance of human error. In the end, it makes governance more flexible and open.

The future plan for this engine lays the groundwork for better intelligence & more freedom. AI-powered auto-policy creation will help teams turn compliance structures or internal requirements into code that can be run. Engineers can use LLM-based repair suggestions to fix these violations quickly and safely. Adding the engine to containerized CMS installations, such

as WordPress, Drupal & headless CMS running on Kubernetes, would improve runtime controls and workload separation. Dynamic risk-scoring frameworks can ultimately improve the consistency of their administration. This helps CMS administrators decide which to do based on the present scenario, the kind of threats that happen to be happening, and how company operations will be influenced. The emerging ideas suggest that CMS governance is going to become more automated, smarter, and able to handle different scenarios in the cloud in future generations.

References

[1] De Leusse, Pierre, and Krzysztof Zielinski. "Toward governance of cross-cloud application deployment." *arXiv preprint arXiv: 1203.0432* (2012).

[2] Ahmed, Usama, Imran Raza, and Syed Asad Hussain. "Trust evaluation in cross-cloud federation: Survey and requirement analysis." *ACM Computing Surveys (CSUR)* 52.1 (2019): 1-37.

[3] Wang, Huaimin, Peichang Shi, and Yiming Zhang. "Jointcloud: A cross-cloud cooperation architecture for integrated internet service customization." *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE, 2017.

[4] Raj, Pethuru, and Anupama Raman. "Multi-cloud management: Technologies, tools, and techniques." *Software-defined cloud centers: Operational and management technologies and tools*. Cham: Springer International Publishing, 2018. 219-240.

[5] Domaschka, Jörg, et al. "Axe: A novel approach for generic, flexible, and comprehensive monitoring and adaptation of cross-cloud applications." *European Conference on Service-Oriented and Cloud Computing*. Cham: Springer International Publishing, 2015.

[6] Fortis, Teodor-Florin, and Victor Ion Munteanu. "From cloud management to cloud governance." *Continued Rise of the Cloud: Advances and Trends in Cloud Computing*. London: Springer London, 2014. 265-287.

[7] Baur, Daniel, et al. "A model driven engineering approach for flexible and distributed monitoring of cross-cloud applications." *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2018.

[8] Zeginis, Chrysostomos, et al. "Towards cross-layer monitoring of multi-cloud service-based applications." *European Conference on Service-Oriented and Cloud Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[9] Carrasco, Jose, Francisco Durán, and Ernesto Pimentel. "Trans-cloud: CAMP/TOSCA-based bidimensional cross-cloud." *Computer Standards & Interfaces* 58 (2018): 167-179.

[10] Jinlong, E., et al. "CoCloud: Enabling efficient cross-cloud file collaboration based on inefficient web APIs." *IEEE Transactions on Parallel and Distributed Systems* 29.1 (2017): 56-69.

[11] RAZA, IMRAN, and SYED ASAD HUSSAIN. "Trust Evaluation in Cross-Cloud Federation: Survey and Requirement Analysis." (2018).

[12] Goonasekera, Nuwan, et al. "CloudBridge: A simple cross-cloud python library." *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*. 2016.

[13] Carrasco, Jose, et al. "Bidimensional cross-cloud management with TOSCA and Brooklyn." *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 2016.

[14] Hou, Fu, and Xinjun Mao. "Cross-clouds services autonomic management approach based on self-organizing multi-agent technology." *Concurrency and Computation: Practice and Experience* 28.11 (2016): 3213-3237.

[15] Vishnubhatla, Sudhir. "Migrating Legacy Information Management Systems to AWS and GCP: Challenges, Hybrid Strategies, and a Dual-Cloud Readiness Playbook." *Hybrid Strategies, and a Dual-Cloud Readiness Playbook (November 20, 2017)* (2017).