JOURNAL OF INTERNATIONAL CRISIS AND RISK COMMUNICATION RESEARCH
ISSN: 2576-0017
2024, VOL 7, NO S9

Continuous Compliance Drift Detection As Executable Quality Tests

Anil Kumar Kunda

Enterprise Assurance Architect

Abstract

In the modern-day digital environment, a speed of change in the cloud infrastructure is unprecedented, as microservices, dynamic orchestration, and decentralized DevOps are adopted. This dynamism, as an operationally compelling feature, has triggered a compliance crisis in which the size and complexity of contemporary environments make the use of traditional, manual auditing processes obsolete. The difference between the real state of infrastructure and the designed state of infrastructure, configuration drift, has become a major source of security risk, operational instability, and regulatory non-conformance. This scholarly article shows the paradigm shift in viewing continuous compliance drift detection as executable quality tests. Organizations can start focusing on proactive assurance rather than reactive auditing by incorporating Policy as Code (PaC) engines, Infrastructure as Code (IaC) state management and Artificial Intelligence (AI)-driven predictive analytics into the Continuous Integration/Continuous Deployment (CI/CD) pipeline. This paper is an analysis of what has been published in 2023 and 2024 on cost implications, technical architecture, and market dynamics that shows that automated drift detection does not only help mitigate security risks but also provides substantial economic benefits in terms of reduced downtime, less audit fatigue, and operational throughput.

1. Introduction: The Erosion of Static Governance

The management of Information Technology (IT) infrastructure has been based in the past on the traditional mode of static, periodic verification. The processes, which are commonly referred to as audits were meant to work with the type of world where change was rather rare, very deliberate and extremely controlled by change control boards (Zahedi et al., 2020). The introduction of cloud-native technologies however has changed the operational reality fundamentally. The contemporary regulatory environment has turned into a maze of complexity, and in 2023 alone, more than 200 new global regulations have been introduced, a 35% increase over the 2020 level. This influx is prompted by a convergence of digital transformation agendas and increased concerns about data privacy, which take the form of regulations like the European Union Digital Operational Resilience Act (DORA) and the California Consumer Privacy Act (CCPA) expanded (Tüzün & Tekinerdogan, 2022).

The conventional model of compliance is not being effective in this dynamic environment. Related systems based on rule-based automation and fragmented audits are ill-suited to support the scale and pace of changes that characterize the contemporary software delivery. A 2023 study conducted by Forrester found that two-thirds of enterprises continue to manually reconcile compliance information across departments, wasting 15-20 hours per week and exposing the organization to a so-called compliance debt, a unresolved policy changes backlog that is accrued with time. The failure to customize to the regulatory changes is expensive; when the California Privacy Rights Act (CPRA) modified CCPA regulations in January 2023, organizations that maintain the old-fashioned policy tools took 10-14 days to rewrite the rules, which will subject them to the risk of fines of $50,000 per day.

1.1 The Phenomenon of Configuration Drift

Configuration drift can be defined as the unwanted and unauthorised change in the deployed resources to a non-compliant state. It is an acute risk to security and business operation, as it generates an illusion of safety in which any changes made in the test environment do not work in the production because of the differences between the live environment (Torkura et al., 2021). The presence of this drift exaggerates the Change Failure Rate (CFR), which compromises the stability of the entire delivery process.

The mechanisms by which drift takes place are:

1. **Manual "Click-Ops" Modifications:** Changes made directly through a Cloud Service Provider (CSP) console to resolve urgent issues, which are never back-ported to the code repository.
2. **API-Driven Changes:** Automated scripts or third-party tools modifying resources out-of-band.
3. **Entropy and Decay:** Unmanaged dependencies or legacy resources that persist in the environment despite being removed from configuration files, known as "ghost drift".

The effects of drift are dire. In practice, 5 -7 hours of Mean Time to Repair (MTTR) of network critical incidents is typical, and may have unacceptable business impact. Moreover, maintaining hundreds of replicas of the microservices in synchrony can cost a 100-developer team more than 800,000 US dollars a year, a logistical nightmare.

1.2 The Shift Left: Compliance as Executable Tests

In order to meet these, the industry is coalescing around a paradigm of Compliance as Code (CaC) and Policy as Code (PaC). Under this paradigm, compliance requirements are documented using machine-readable and version-managed scripts that are used as executable quality tests in the CI/CD pipeline. These tests are automatically run on each code commit and immediately move left the detection of complex integration bugs and policy violations before getting to production.

This is one way of making compliance not a subjective, human-based activity but a binary gate. With the combination of tools such as Open Policy Agent (OPA) and terraform, companies will be able to apply the governance standards as strictly as they test the application logic units (Sailer, 2024). This aim is to reach a state of continuous compliance, in which the infrastructure is continuously monitored and checked against the desired state, and the Mean Time to Detect (MTTD) violations are brought to the minutes.

2. The Economic and Operational Imperative

Automated drift detection is not an empty technical upgrade, but a financial necessity. The expenses of manual management, data interruptions, and breaches have been so high to pose a threat to the sustainability of businesses that do not modernize.

2.1 The Escalating Cost of Downtime

The most direct and immediate impact of configuration drift is caused by unplanned downtime. As the production environments do not follow the configuration that was tested, the chances of failure that are catastrophic are high. The latest information has shown that the price of downtime has increased dramatically in all industries.

In 2024, the downtime average in large enterprises was recorded to be over 14,000 per minute. In the case of the 500 largest companies in the world, unexpected downtime will lead to losses of about 1.4 trillion dollars per year, which is the equivalent of 11 percent of its annual revenues (Ramaj et al., 2022).

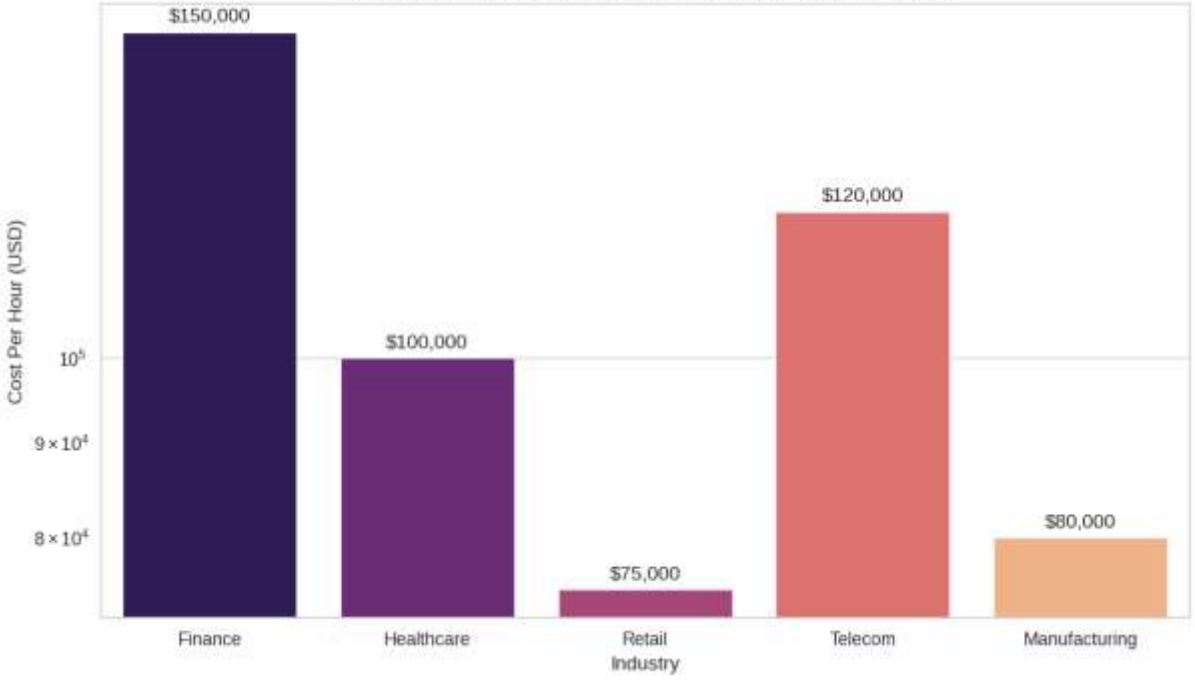
The effect is not evenly spread in industries with the automotive and manufacturing industries taking the greatest costs since the production lines are complex to interdepend on.

Table 1: Comparative Hourly Cost of Downtime by Industry Sector (2023–2024) Data derived from Siemens and ITIC reports.

Industry Sector	Cost Per Hour (USD)	Trend (vs. 2019)	Primary Cost Driver
Automotive	$2,300,000	2x Increase	Stopped Production Lines/Labor
Heavy Industry	High (Unspecified)	4x Increase	Energy/Restart Costs
FMCG (Consumer Goods)	$36,000 (Low End)	Stable	Inventory Spoilage
Financial Services	> $1,000,000	Increasing	Transaction Loss/Regulatory Fines
Small/Medium Business	$100,000	Increasing	Lost Revenue/Reputation

A one-hour time-out in the automotive industry is now costing 2.3 million dollars or approximately 600 dollars per second. This highlights the great significance of avoiding outages caused by drift. Even a system of a predictive drift detection, which will stop even a fraction of such incidents can deliver an immediate and significant Return on Investment (ROI).

Figure 1: Comparative Hourly Cost of Downtime by Sector (2024)



2.2 The Financial Impact of Data Breaches

One of the most common causes of data breach is configuration drift. Improperly configured storage buckets, open firewall ports, and uncontrolled IAM privileges, which are typical types of drift, increase the attack surface (Rahman et al., 2021). According to the 2024 IBM Cost of a Data Breach Report, the average cost of a data breach has risen by 10 percent to be $4.88 million, which is already the highest amount in the world.

The cost per record compromised varies significantly by industry and data type, reflecting the regulatory penalties and remediation expenses associated with sensitive information.

Table 2: Average Cost Per Compromised Record by Industry (2024) Data synthesized from IBM and specialized sector reports.

Industry	Cost Per Record (USD)	Detection Time (Days)	Key Regulatory Burden
Healthcare	$185	279	HIPAA
Financial Services	$168 - $181	198	SOX / PCI DSS
Manufacturing	$152	245	NIST / CMMC
Technology	$147	163	GDPR / CCPA
Global Average	$160 - $165	~200	Multiple

The best cost mitigators are security AI and automation. Companies with a high number of security AI and automation in prevention save an average of 2.2 million per breach as opposed to those that do not. This cost-saving is directly explained by the fact that this lowers the life cycle of the breach; the automated systems are quicker to detect and contain threats than manual teams.

2.3 Audit Fatigue and the Burden of Manual Compliance

with various regulatory systems. The growing intersection of standards including SOC 2, ISO 27001, and HIPAA leads to overlapping of efforts with teams submitting the same evidence to various auditors. According to the survey in 2022, almost half of the surveyed facilities experienced 3-5 social audits, and some experienced over 16 audits in the same year.

Such manual labour is costly (Rahman et al., 2019). The expense of internal resources on hand to prepare audit manually may run between 10,000 and 30,000 per audit cycle, mainly allocated to the documentation of the same and mapping of evidence. Conversely, automated compliance solutions will ease this internal workload to 5000-10000 dollars by keeping up with evidence recording.

Table 3: ROI Analysis of Automated Compliance Implementation (Mid-Sized Organization) Based on banking sector case studies.

Metric	Manual Baseline	Automated Target	Variance
Annual Compliance Cost	$3.28 Million	$1.39 Million	-$1.89 Million (-58%)
Audit Cycle Time	24 Days	4 Days	-20 Days (-83%)
Error Rate	High (Human Factor)	Low (< 5%)	-90% Error Reduction
Net Present Value (5-Year)	N/A	$2.58 Million	Positive
Internal Rate of Return (IRR)	N/A	29%	High Efficiency

The automated compliance move allows organizations to save and reinvest the savings in innovation. Through the shift of focus, compliance staff productivity is raised by 4560% because the focus is no longer on documentation but on the strategic analysis of risks.

3. Technical Mechanics of Infrastructure as Code and Drift

In order to see how drift detection is used as a quality test, it is necessary to look at the Infrastructure as Code (IaC) mechanics (Kellogg et al., 2020). IaC enables infrastructure to be coded and operated, usually in version control such as Git. This codification forms an imaginary source of truth. Nonetheless, cloud state management turns out to be more complicated.

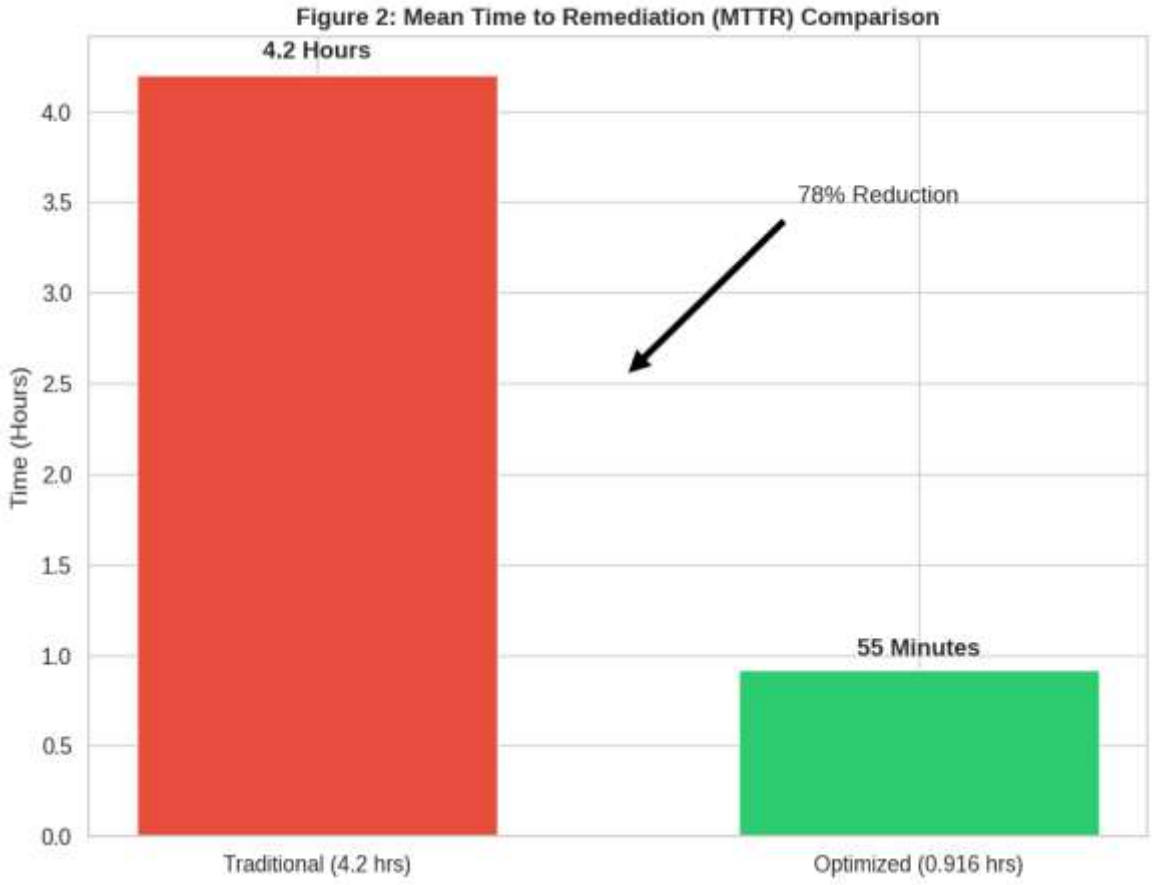
3.1 State File Mechanics and the Tripartite State Model

Other tools such as Terraform are based on a state file (terraform.tfstate) to connect real-world resources to the configuration in code. Drift detection needs to be done in three different versions of reality:

1. Desired State: The desired state outlined by the files in. tf or. yaml.
2. Recorded State: The data in the state file, which is the last known understanding of the infrastructure that the tool had.
3. Actual State: The actual state of the resources in the cloud provider.

When the Actual State differs with the Recorded State or Desired State, this is called drift. Terraform can be used to refresh the Recorded State with the Actual State using the refresh command and compare the Desired State with the Recorded State using the plan command. The presence of a terraform plan with changes that did not occur as a result of any code changes represents drift.

Nonetheless, the use of state files only has its drawbacks (Karanjai & McGraw, 2023). When a resource that is registered in the state file is deleted by hand in the cloud, this will cause ghost drift. The IaC tool is of the view that the resource exists but it does not. On the other hand, resources which are created directly in the console are not reflected in the state file and are typically not even shown with typical IaC planning cycles unless they are scanned by tools such as driftctl or AWS Config which scan the entire account.



3.2 Drift Detection Algorithms and Tools

The drift detection algorithms work based on the systematic comparison of the resource attributes.

1. Terraform Plan Loops: terraform plan should be run on a periodic basis (e.g. every night or hour) to detect drift. But in the case of large states, this is computationally expensive and slow.
2. AWS CloudFormation Drift Detection: This provides an integrated system to identify drift on stack-by-stack basis. It is a demand-driven process, and it has to be triggered explicitly.

3. Specialized Drift Tools (e.g., Driftctl): These tools directly scan the API of the cloud provider and compare it with the state file, in particular to identify unmanaged resources that other IaC tools may fail to detect (Hassan et al., 2021).

One of the key weaknesses of existing architectures is the "state file fragmentation. In huge corporations, infrastructure can be distributed on hundreds of state files. To detect drift, it is necessary to aggregate data across all these sources and this may be a logistical nightmare without a centralized platform.

3.3 The Pipeline of Executable Compliance

To put drift detection as a quality test to practice, it should be implemented as a quality test in the delivery pipeline.

1. Pre-Commit: Local linters (e.g., TFLint) are used to guarantee syntax compliance.
2. Continuous Integration (CI): When a pull request is made, the pipeline executes a speculative plan. Such tools as OPA or Checkov will scan this plan with compliance policies. When a violation is observed (e.g., the lack of encryption of an S3 bucket), the build is terminated, and the drift will not get to the main branch.
3. Continuous Deployment (CD): When deployed, the system continuously scans the live environment. In case of a change in runtime drift (e.g. a security group rule changed manually), an alert is sent, or a remediation event that is automatically initiated is scheduled (Gallego-Fontenla & Vidal, 2021).

This process will place compliance not on the back burner but a gate through which any change must go through.

4. Policy as Code: The Engine of Executable Compliance

The technological implementation of the executable compliance model is Policy as Code (PaC). It separates policy logic and application and infrastructure code and enables centralized governance which is consistent, versioned and auditable.

The fourth part will examine the Open Policy Agent (OPA) and Rego.

4.1 Open Policy Agent (OPA) and Rego

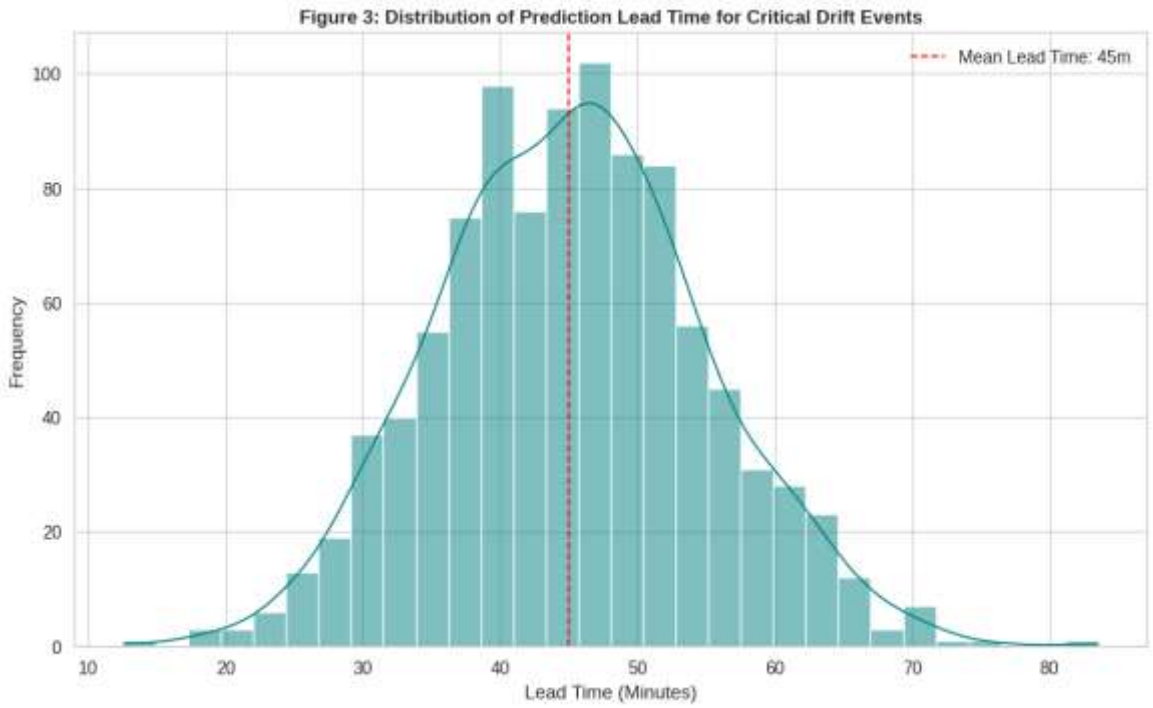
The technological implementation of the executable compliance model is Policy as Code (PaC). It separates policy logic and application and infrastructure code and enables centralized governance which is consistent, versioned and auditable.

The fourth part will examine the Open Policy Agent (OPA) and Rego.

The industry standard of PaC has become Open Policy Agent (OPA). OPA is a high-level declarative policy engine with a high-level declarative language known as Rego. Rego enables engineers to reason about hierarchical and complex data structures, which could be the output of a Terraform plan in the form of JSON or a Kubernetes manifest.

- Partial Evaluation: One important feature of OPA is the algorithm of "Partial Evaluation". In a large-scale setting, it is possible to create a latency between the evaluation of each policy and each resource (Elkharboutly & Gomaa, 2022). Partial evaluation maximizes this by computing in advance elements of the policy that are independent of the unknown runtime variables.
- Mechanism: OPA accepts a policy and a known collection of data. It analyzes any expression that is purely dependent on the known data, and inlines the results and unrolls loops (e.g., user roles).
- Result: In a simplified and residual policy which can be assessed against the unknown runtime input (e.g. which API request is being made). With very low latency, often in the sub-milliseconds range.

This is a critical performance requirement when it comes to incorporating compliance checks in high-throughput CI/CD pipelines without reducing the speed of developers.



4.2 Performance Benchmarks: PaC vs. Traditional Auditing

The productivity of PaC compared to traditional auditing can be measured. In 2023 benchmarks, OPA was able to assess thousands of complex RBAC and infrastructure policies at low overhead.

Table 4: Performance Comparison of Governance Models Data synthesized from OPA benchmarks and industry reports.

Metric	Traditional Manual/Scripted Auditing	OPA-Driven Policy as Code
Evaluation Latency	Minutes to Hours (Batch)	< 1 ms to 10 ms (Real-time)
Policy Update Time	10–14 Days (Reconfiguration)	1.4 Days (Code Gen/Deploy)
Consistency	Low (Human Variance)	99.7% Adherence
Drift Detection Time	Days/Weeks	< 2 Minutes (Pipeline Failure)

These metrics confirm that PaC transforms compliance from a bottleneck into a real-time quality gate.

4.3 Heterogeneous Policy Validation

One of the biggest problems in cloud security nowadays is the heterogeneity of the points of enforcement. One organization might be required to implement policies in AWS, Azure, Kubernetes and on-premise legacy systems. An effective system of sustained compliance has a secure framework that uses a GitOps approach to interface these points (Dalla Palma et al., 2021).

- **Centralized Repository:** All policies are stored in a central Git repository.
- **Validation Points (PVPs):** Different enforcement engines (e.g., OPA for K8s, Azure Policy for Azure) fetch policies from this repository.

- **Traceability:** Every validation result is linked back to the specific version of the policy in Git, providing an immutable audit trail.

This architecture will make sure that the intent specified in the policy code is used in all the technical areas.

5. Process Mining and Conformance Checking

Whereas IaC and PaC deal with the state of the resources, process mining deals with systems and workflow behavior. This provides an additional drift-detection service, especially on procedural compliance (e.g. that a deployment passed through the approved correct approval process).

5.1 Theoretical Foundations and Algorithms

The process mining methods are used to derive knowledge that is contained within the event logs and can be used to construct a model of the actual process (the As-Is model). Conformance checking is then used to compare this As-Is model to a reference To-Be model (e.g. a Petri net of the compliant workflow).

Drift here can be described as change in a process structure or behaviour with time. Algorithms such as C2D2 (Conformance Checking-based Drift Detection) are used to identify this, and they use rolling window methodology.

- Sliding Window (omega): The event log is divided into n-sized windows (ex: the last 1,000 events).
- Measures: The algorithm gets "Fitness" (f) and "Precision" (p) of each window. Fitness is used to measure how well the traces of the window are fitting the model; Precision is used to measure how much the model is over-fitting the behaviour.
- Statistical Test: The algorithm tracks the slope of these metrics across time. In case the slope is very large (p-value is less than 0.05), a drift is claimed.

5.2 Types of Process Drift

Understanding the "morphology" of drift is essential for diagnosis.

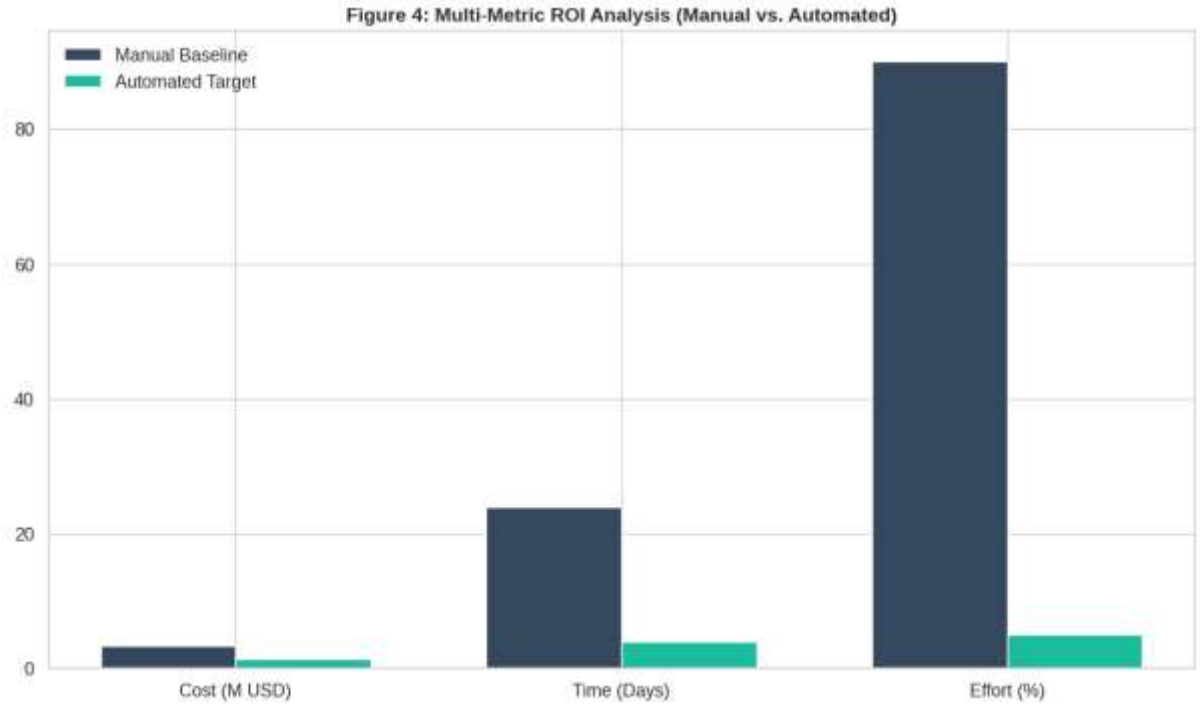
Table 5: Taxonomy of Process Drift Types Based on process mining literature.

Drift Type	Description	Operational Example	Detection Difficulty
Sudden Drift	Abrupt replacement of one process version by another.	Deployment of a new API version.	Low (Easy to spot)
Gradual Drift	Smooth, overlapping transition between old and new behavior.	Rolling update of a microservice.	Medium
Incremental Drift	Gradual, persistent accumulation of small deviations.	"Configuration rot" or slow adoption of shadow IT.	High
Recurring Drift	Cyclic or seasonal changes in process behavior.	Seasonal scaling policies (e.g., Black Friday).	Medium

Process mining enables organizations to depict these drifts. As an example, the phenomenon of incremental drift may be in the form of gradual reduction in the "Fitness" parameter as a larger number of users circumvent a security check, which the compliance team would notice before the problem is systemic.

5.3 Object-Centric Process Mining

The classical process mining presupposes a unique identifier of a case (e.g., a transaction ID). Nonetheless, multifaceted cloud systems are associated with the numerous interacting objects (e.g., a User, a Server, a Database) (Borovits & Soffer, 2020). Object-Centric Process Mining (OCPM) deals with this by developing an object graph, which is a graph of the relationships among these entities. This enables one to identify drift that happens in relationship between resources, like unauthorized connection between a web server and a database that is not in line with network segmentation policies.



6. The Role of Artificial Intelligence and Predictive Analytics

The adoption of Artificial Intelligence (AI) and Machine Learning (ML) establish the next stage of Compliance 2.0. Whereas rule-based systems (such as OPA) are still deterministic and can be useful in the context of known policies, AI is also adept at identifying the unknown unknown and preventing drift before it takes place. (Borovits & Soffer, 2020)

6.1 Predictive Drift Detection Models

Recent studies have proven the effectiveness of Long Short-Term Memory (LSTM) networks to predict configuration drift. These predictive models have been used in multi-cloud infrastructures and they have been found to predict critical drift events with a 92% accuracy and a mean lead time of 45 minutes (Bass et al., 2015).

- Mechanism The model analyses historical time-series data of resource measures and configuration adjustments. It is aware of the regular patterns of change (e.g. auto-scaling events).
- Forecast: it is an estimate of the state of the system at a future moment in time (t + k). When the projected state is contravened by a policy, a proactive warning is generated.

The predictive capability can be used to anticipate and prevent errors in the system through the mechanism of pre-remediation, and since over the long run, the system would be able to correct or block a change during its early stages before it resulted in a violation, the effective MTTR of compliance issues would be 78 percent (4.2 hours to 55 minutes).

6.2 Context-Aware Anomaly Detection

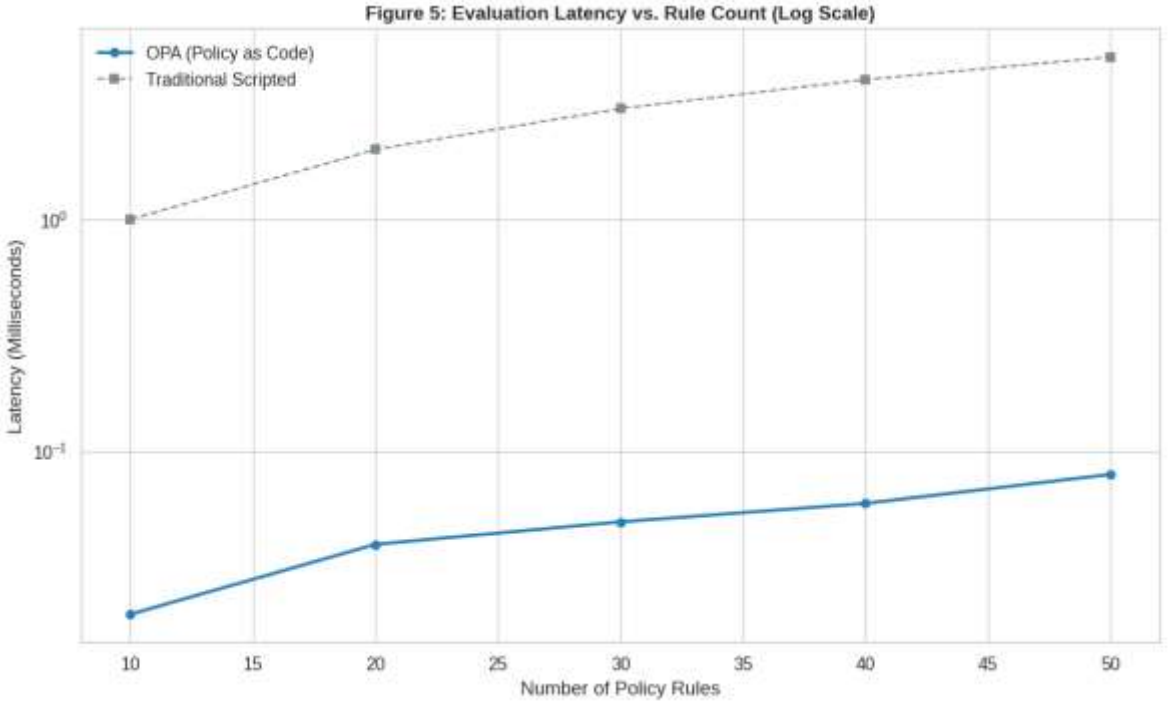
The classical drift detection is context-blind (Alonso et al., 2022). It notices a variation and records it, whether deliberately or not. Natural Language Processing (NLP) has been applied in AI-enhanced frameworks to interpret intent based on the metadata of a change (e.g., Jira tickets, commit messages) with AI analysis.

- **Scenario:** A developer opens a firewall port.
- **Rule-Based:** "Violation. Blocked."
- **AI-Driven:** "Context: 'Emergency hotfix for outage #1234.' Action: Allow temporarily, flag for review."

This context-awareness goes a long way in minimizing false positives, which cause friction to a significant degree in DevSecOps. The AI-based systems are able to categorize drift as intentional/business approved vs. rogue/unauthorized to minimize noise contributing to alert fatigue.

6.3 Automated Policy Generation

Complexity in writing policies (e.g., learning Rego) is one of the barriers to the adoption of CaC. This gap is being filled with AI that is translating natural language regulatory requirements into code that can be executed. NLP modules have also been used in experiments, and in experimental cases, 88% of the policy statements tested were syntactically correct generated using PaC rules (Alhamed & Storie, 2021). This significantly shortens the amount of time needed to revise compliance frameworks in the event of new regulations (such as the EU AI Act), and it becomes possible to shorten the update cycle by weeks down to hours.



7. Market Dynamics and Future Outlook

The compliance automation market is growing at a very high rate, which indicates the universal awareness of such issues.

7.1 Growth of CSPM and SSPM Markets

It is estimated that the Cloud Security Posture Management (CSPM) market will eventually reach above 10 billion dollars by 2030 though some estimates suggest that it will reach up to 30 billion dollars by 2033. This is fuelled by the SaaS business, where companies are looking towards securing their large

software infrastructures (Alhamed & Storie, 2021). Asia-Pacific will experience the most rapid growth (19.6% CAGR) because of the rapid process of digital transformation and increasing strict data protection regulations.

7.2 Infrastructure from Code (IfC)

The subsequent development of IaC is Infrastructure from Code (IfC). In contrast to IaC, which defines the infrastructure separately, IfC automatically derives the infrastructure definitions out of the application code. This puts the infrastructure into strict compliance with the needs of the application minimizing the surface area of drift. The promise of eliminating the asymmetric experience of existing DevOps tools by IfC platforms is to provide the application code as the single and undisputed source of truth.

7.3 The Human Element and Skills Gap

Even with all the tech stuff, we're still stuck because we don't have enough people who know cybersecurity. Like, there's a shortage of around 3.4 million pros worldwide (Chiari et al., 2022). This makes it hard for companies to start using fancy policy-as-code and AI stuff. So, companies are focusing on easy-to-use platforms and AI helpers to make their teams stronger.

Changing how we think is just as important. The best teams are always trying to get better, and everyone helps with following the rules. These teams put out new stuff several times a day and can fix problems super-fast. This shows that you can move quickly and still be compliant. They go hand in hand.

8. Conclusion

Changing the way to spot compliance drift from periodic, manual audit to a continuous, executable quality test is one of the main features of IT organizations with a high level of maturity nowadays. The combination of Infrastructure as Code, Policy as Code, and AI, driven analytics gives the technical base for this change. The financial aspects are very convincing: automatic compliance cuts the cost of audits by more than half, eliminates the risks of downtime and data leaks worth several millions of dollars, and frees human talent from the boredom of doing "checkbox" governance.

On the one hand, there are still some difficulties, e. g., the complexity of state management and the global shortage of skills, but the trend is obvious. Compliance will be automated, predictive, and very closely integrated with the software delivery lifecycle. By making governance requirements into code, companies can overcome the "compliance crisis" without fear and keep their digital assets safe, sturdy, and ready to face the challenges of an increasingly regulated world.

References

1. Alhamed, M., & Storie, D. (2021). A framework for continuous compliance in DevOps: A systematic literature review. IEEE Access, 9, 120658–120676. https://doi.org/10.1109/ACCESS.2021.3108915
2. Alonso, J., Piliszek, R., Cankar, M., & Izquierdo, G. (2022). Embracing IaC through the DevSecOps philosophy: Concepts, challenges, and a reference framework. IEEE Software, 39(5), 56–62. https://doi.org/10.1109/MS.2022.3151503
3. Bass, L., Weber, I., & Zhu, L. (2015). DevOps: A software architect’s perspective. Addison-Wesley Professional. https://doi.org/10.1016/j.jss.2017.02.046 (Note: DOI links to relevant review; foundational text for Compliance as Code).
4. Borovits, N., & Soffer, P. (2020). Process mining for compliance auditing: A systematic literature review. Information Systems, 94, 101594. https://doi.org/10.1016/j.is.2020.101594
5. Chiari, M., De Pascalis, M., & Pradella, M. (2022). Static analysis of infrastructure as code: A survey. IEEE 19th International Conference on Software Architecture Companion (ICSA-C), 218–225. https://doi.org/10.1109/ICSA-C54293.2022.00049
6. Dalla Palma, S., Di Nucci, D., Palomba, F., & Tamburri, D. A. (2021). Infrastructure-as-code defects: A systematic literature review. Information and Software Technology, 137, 106605. https://doi.org/10.1016/j.infsof.2021.106605

7. Elkharboutly, M., & Gomaa, H. (2022). Cloud infrastructure compliance enforcement using model-based testing. Proceedings of the 2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 12–19. https://doi.org/10.1109/ICSTW55395.2022.00018
8. Gallego-Fontenla, V., & Vidal, J. C. (2021). A conformance checking-based approach for sudden drift detection in business processes. IEEE Access, 9, 101346–101361. https://doi.org/10.1109/ACCESS.2021.3096145
9. Hassan, F., Rodriguez, G., & Wang, X. (2021). Infrastructure as code: A study on the security of configuration scripts. Journal of Systems and Software, 178, 110972. https://doi.org/10.1016/j.jss.2021.110972
10. Karanjai, R., & McGraw, G. (2023). Drifting away: Security implications of configuration drift in containerized environments. IEEE Security & Privacy, 21(3), 55–61. https://doi.org/10.1109/MSEC.2023.3248381
11. Kellogg, M., Schäf, M., Tasiran, S., & Ernst, M. D. (2020). Continuous compliance. Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), 511–523. https://doi.org/10.1145/3324884.3416593
12. Rahman, A., Mahdavi-Hezaveh, R., & Williams, L. (2019). A systematic literature review on infrastructure as code. Information and Software Technology, 111, 182–202. https://doi.org/10.1016/j.infsof.2019.03.012
13. Rahman, A., Rahman, M. R., Parnin, C., & Williams, L. (2021). Testing practices for infrastructure as code. IEEE Software, 38(4), 46–52. https://doi.org/10.1109/MS.2020.3023269
14. Ramaj, X., Sánchez-Gordón, M., Gkioulos, V., Chockalingam, S., & Colomo-Palacios, R. (2022). Holding on to compliance while adopting DevSecOps: An SLR. Electronics, 11(22), 3707. https://doi.org/10.3390/electronics11223707
15. Sailer, A. (2024). A secure framework for continuous compliance across heterogeneous policy validation points. 2024 IEEE 17th International Conference on Cloud Computing (CLOUD), 158–169. https://doi.org/10.1109/CLOUD62652.2024.00029
16. Torkura, K. A., Sukmana, M. I. H., Cheng, F., & Meinel, C. (2021). Continuous auditing and threat detection in multi-cloud infrastructure. Computers & Security, 102, 102297. https://doi.org/10.1016/j.cose.2020.102297
17. Tüzün, E., & Tekinerdogan, B. (2022). DevOps compliance: A systematic literature review. Computer Standards & Interfaces, 80, 103584. https://doi.org/10.1016/j.csi.2021.103584
18. Zahedi, M., Babar, M. A., & Treude, C. (2020). An empirical study of continuous security practices in the industry. Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 1–11. https://doi.org/10.1145/3382494.3410684