Google

Shift-Left Compliance

Bharath Modhipalli is a Staff Site Reliability Engineer (SRE) and Tech Lead at Google Workspace in Sunnyvale, California. He is lead for the Apps Core Infrastructure SRE team, where he focuses on the reliability and scalability of foundational systems for Google Workspace.

Site Reliability Engineering

Karanveer Anand is a Technical Program Manager for Google SRE, based in Sunnyvale, California. He oversees Workspace reliability, managing a portfolio of projects aimed at enhancing the reliability of infrastructure and ensuring customer compliance.

Disclaimer

This presentation does not endorse any specific product or service.

All examples used are hypothetical & simplistic to help make this talk more relatable

Agenda

- Problem Statement
- Fitting Compliance
- Shift Left Compliance
- Framework
- Outcomes & Challenges
- Conclusion

Problem Statement

Compliance: A Strategic Trust Imperative

Modernized frameworks are the cornerstone of customer and partner trust.

EU AI Act

Enforces risk-based management and explainability to ensure AI safety and accountability.

DORA

Mandates comprehensive Information and Communication Technology risk management and resilience testing for financial sectors.

CCPA / CPRA

Enforces strict privacy rights through granular data mapping and the "right to delete."

Compliance: A Strategic Trust Imperative

Deconstructing compliance into core technical infrastructure properties.

Automated Data Residency

Enforcing programmatic storage boundaries to restrict data to specific geographic jurisdictions.

Zero Trust Access Control

Identity-based granular security ensuring least privilege across all microservices.

Data Lifecycle & Lineage

Programmatic mapping to track and delete user data to satisfy CPRA/GDPR requests.

AI Governance & Explainability

Embedding traceable lineage and testing directly into ML pipelines for validation.

Fitting Compliance

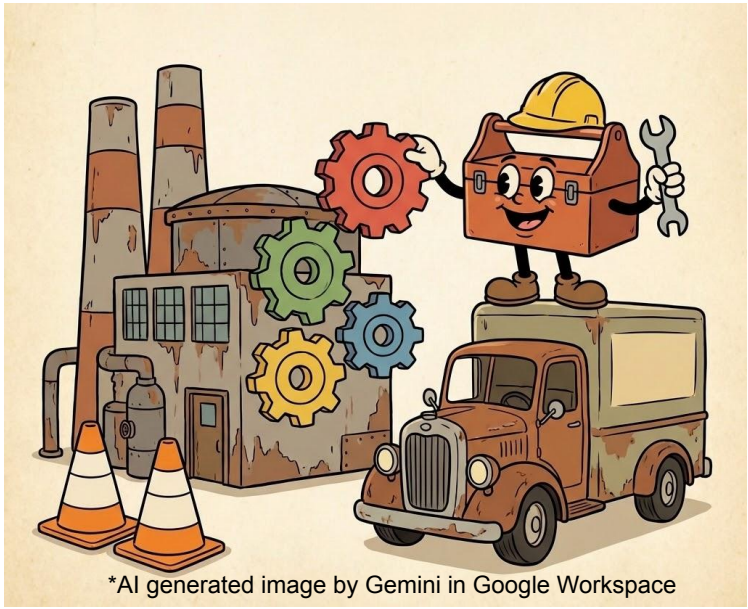
Retro-fitting Compliance is a High-Cost Technical Debt

Infrastructure Overhaul
Inherently slow and disruptive. Requires complex data migrations.

Core Redesign
Redesigning partitioning, security, and auditing pipelines retroactively.

Reactive Scramble
Unsustainable cycle of emergency fixes as systems evolve.

Innovation Drain
Compliance efforts divert focus from new feature growth.



*AI generated image by Gemini in Google Workspace

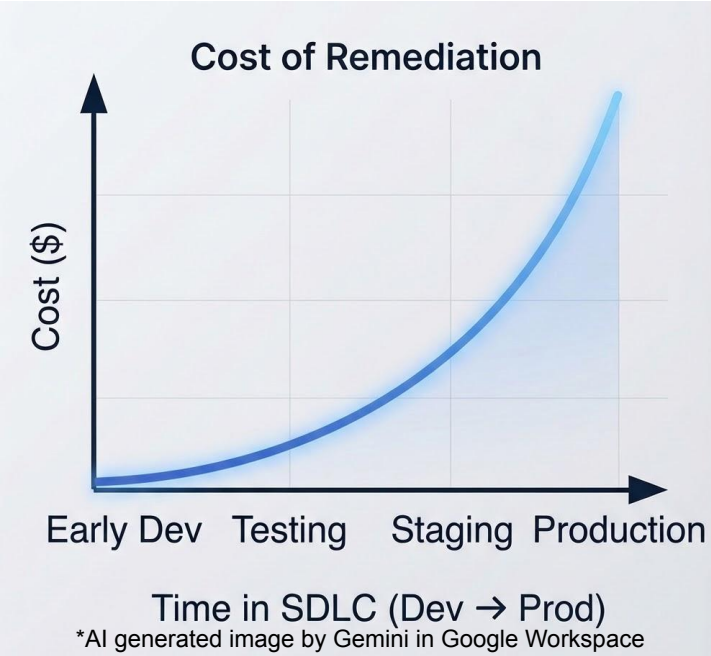
New Systems Inherit Compliance Debt by Default

Violations from Day One: Lack of upfront consideration leads to immediate mandate violations.

Immediate Backlogs: Reactive patching creates an instant queue of non-compliant systems.

Operational Complexity: Scaling makes foundational issues increasingly harder to address.

Compounding Debt: Engineering costs explode as violations are found later in SDLC.



*AI generated image by Gemini in Google Workspace

Shift Left Compliance

Google

Site Reliability Engineering

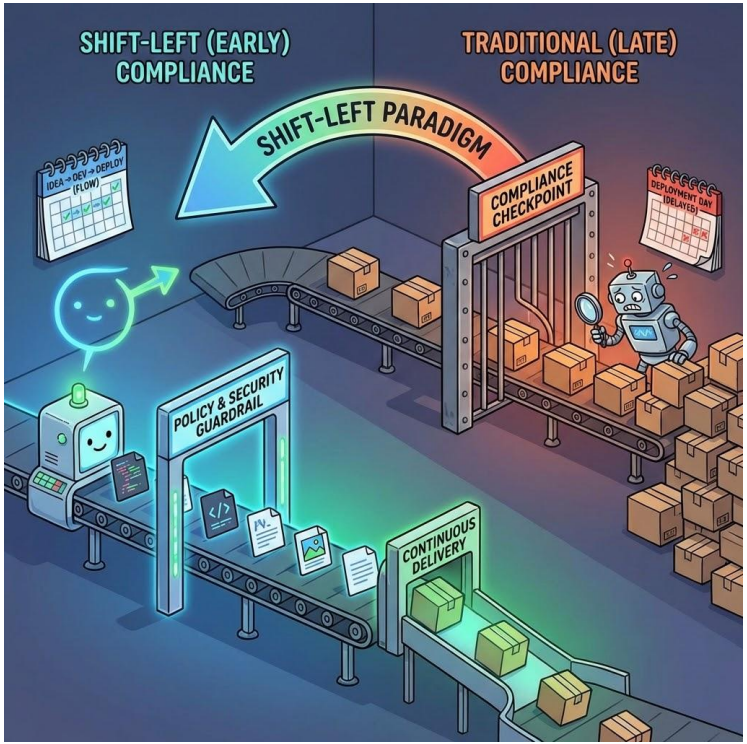
Shift-Left Compliance: A Paradigm Shift

Intrinsic Security
Integrates security requirements into design from day one as a core property.

Turn-Up Integration
Active enforcement ensures new infrastructure doesn't inherit compounding debt.

Workflow Absorption
Address mandates in small increments within standard development workflows.

Bending the Cost Curve
Reduces long-term costs by shifting from reactive to proactive processes.



*AI generated image by Gemini in Google Workspace

Shift-left compliance integrates requirements from design

Addressing regulations early creates a proactive, manageable lifecycle.

Compliance-as-Code

Translating static regulations into machine-readable, version-controlled policies.

Compliance at Turn-Up

Ensuring new infrastructure is "born compliant" during initial system creation.

Continuous Enforcement

Blocking changes that violate policies to prevent drift.

Automated CI/CD Checks

Immediate feedback via automated validations at code commit and deployment.

Compliance-as-Code

Compliance-as-Code transforms static rules into dynamic software

Treating compliance as version-controlled, automated, and scalable logic.

Deterministic Translation

Moving from regulatory text to machine-readable rules. Logic dictates pass/fail, eliminating manual interpretation.

Software-Driven Governance

Policies are version-controlled and centrally managed, enabling automatic enforcement across the org.

Unified Enforcement

Unify policy via OPA and Rego across microservices, K8s, CI/CD pipelines, and API gateways.

Inherent Scalability

Support new regulations (HIPAA, GDPR) by appending policies rather than rebuilding audit infrastructure.

Compliance-as-Code: OPA Examples

The Scenario: A policy that fails any Terraform deployment if required tags (like owner, data_classification, or compliance_scope) are missing.

```
package terraform.governance.tags

# Define the mandatory tags for all resources
required_tags := {"owner", "compliance_scope", "data_classification"}

# Deny if any required tags are missing from a Google Compute Instance
deny[msg] {
    resource := input.resource_changes[_]
    resource.type == "google_compute_instance"

    provided_tags := object.keys(resource.change.after.labels)
    missing_tags := required_tags - set(provided_tags)

    count(missing_tags) > 0
    msg := sprintf("Metadata Violation: Instance '%s' is missing
mandatory compliance tags: %v", [resource.address, missing_tags])
}
```

→ List of mandatory tags

→ The check condition

→ Error message with details for the developer

Compliance-as-Code: OPA Examples

The Scenario: A policy that enforces Customer-Managed Encryption Keys (CMEK) on cloud storage rather than relying on default cloud encryption.

```
package terraform.gcp.encryption

# Deny rule if a Google Storage bucket lacks explicit CMEK encryption
configuration
deny[msg] {
   resource := input.resource_changes[_]
   resource.type == "google_storage_bucket"

   # Check if the encryption block is missing or empty
   not resource.change.after.encryption

   msg := sprintf("FedRAMP Violation: Storage Bucket '%s' does not have
explicit encryption configured.", [resource.address])
}
```

→ Checks for configured encryption settings

→ Error message with details for the developer

Compliance-as-Code: OPA Examples

The Scenario: A policy that checks the data_classification tag of a storage bucket and dynamically restricts the deployment region based on how sensitive that data is. For example, "PII" (Personally Identifiable Information) might be strictly confined to specific EU regions, while "Public" assets have more flexibility.

```
package terraform.gcp.residency

# Map data types to allowed regions
allowed_regions := {
    "pii": {"europe-west1", "europe-west3"},
    "public": {"us-central1", "europe-west1", "asia-northeast1"}
}
# Deny if bucket location isn't in the allowed list
deny[msg] {
    bucket := input.resource_changes[_]
    bucket.type == "google_storage_bucket"
    class := bucket.change.after.labels.data_classification
    loc := lower(bucket.change.after.location)
    not allowed_regions[class][loc]

    msg := sprintf("Residency Violation: '%s' data is prohibited in
'%s'", [class, loc])
}
```

→ Allowed regions based on data_classification

→ Checks configured regions against allowed regions

→ Error message with details for the developer

Compliance at Turn-Up

Born Compliant: Eliminating Debt at Creation

New systems are secured from inception through automated structural guardrails.

Golden Paths

Vetted Templates

Security teams create pre-approved infrastructure templates.

Hardened Defaults

Encryption and logging are natively preconfigured.

Guided Config

Restricted Choices

Developer inputs are limited to pre-approved, compliant options.

Geo-Fencing

Example: Enforcing approved regions for GDPR.

Proactive Validation

Pre-Provisioning

Real-time validation against Compliance-as-Code policies.

Fail-Fast

Violations halt deployment before resources exist in cloud.

Born Compliant: Eliminating Debt at Creation

The Scenario: Data residency mandates require isolating data to specific geographic jurisdictions. Retrofitting an existing binary to meet this standard costs an estimated two weeks of engineering time.

Guided Templating

During service creation, a simple prompt ("Enable Regionalization?") dynamically generates isolated EU and US environments based on developer input.

Hardened Defaults

The platform automatically injects mandatory compliance-tracking libraries directly into the new service's manifest.

Pre-Provisioning Validation

Presubmit checks validate static routing rules, blocking deployment if EU shards attempt to route to non-EU backends.

The Impact: Guaranteed regionalization before production, eliminating the two-week retrofitting penalty per service.

Google

Continuous Enforcement

Site Reliability Engineering

Continuous Enforcement Prevents Compliance Drift

Automated CI/CD checks ensure that code changes never erode your compliance posture.

The Problem: Compliance Drift

The gradual erosion of compliance (or "backsliding") due to code changes, configuration updates, or new dependencies.

The Solution: CI Guardrails

Every pull request triggers the same suite of compliance checks used at turn-up, blocking non-compliant modifications instantly.

Developer Guidance

PR scans provide clear, immediate instructions on violations and how to fix them, reducing friction.

Iterative Correction

Instant re-verification of corrected code ensures compliance is achieved before any merge occurs.

Immutable Artifacts & Audit Integrity

Every check, pass, and fail is logged as persistent audit proof. This replaces manual evidence gathering with an automated, high-assurance trail for auditors.

Continuous Enforcement: GitHub Actions & OPA

Automated policy enforcement prevents compliance drift by validating every Pull Request against machine-readable rules.

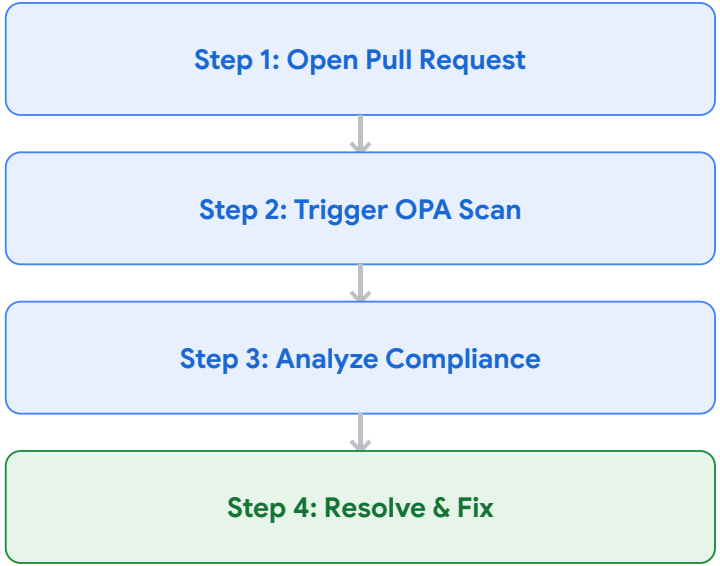
The Automation Workflow

1. Trigger: Engineer opens a Pull Request (PR) to merge a feature branch into the main branch.

2. Execution: GitHub Actions automatically triggers an Open Policy Agent (OPA) scan job.

3. Evaluation: Proposed code changes are analyzed against established compliance baselines.

4. Resolution: Developer receives immediate feedback on violations and must push a fix to proceed.



Automated CI/CD Checks

Automated CI/CD Checks

CI/CD tools execute automated continuous enforcement from code commit to deployment.

Static Analysis

Detects insecure data practices and hard-coded credentials in application code.

Infrastructure as Code Validation

Scans infrastructure code to enforce encryption or least-privilege network settings.

Dependency Scan

Identifies vulnerabilities or licensing issues in third-party libraries.

Data Governance

Verifies data storage and retention practices adhere to policies during build.

Balanced Enforcement: Audit vs. Blocking Modes

Strategic deployment modes enable a smooth transition from visibility to prevention.

Audit-Only Mode (Observational)

Passive Logging: Records violations without halting deployment.

Gradual Adoption: Ideal for baseline assessments of legacy systems.

Visibility: Scopes affected systems before enforcing strict rules.

Blocking Mode (Preventative)

Immediate Interruption: Instantly halts pipelines upon policy violation.

Fast Feedback: Provides actionable failure details early in workflow.

Zero Drift: Prevents non-compliant code from reaching production.

Exemptions

Exemption balance strict enforcement with developer velocity and targeted rollouts.

Developer Self-Service

Automatic Validation: Compliance checks verify existing exemptions before surfacing violations to the user.

Unblocking Velocity: Allows developers to bypass checks in case of tool bugs or broken infrastructure without manual intervention.

Audit Trail: Every exemption is logged and scoped, ensuring visibility without halting critical pipeline progress.

Strategic Rollout Mode

Baseline Exemption: All existing violations were exempted at creation to focus strictly on new violations.

Blocking for New Systems: The check is enabled in blocking mode immediately for any new violations created post-rollout.

Debt Containment: Prevents further accumulation of compliance debt while allowing existing systems to migrate on their own timelines.

Outcomes & Challenges

Shift-Left Compliance Drives Long-Term Efficiency

Baking compliance into the start of the development cycle eliminates debt and automates protection.

Born Compliant

New systems avoid legacy compliance debt, reducing the risk of non-compliant feature launches and costly remediation.

Reduced Engineering Cost

Strategic upfront investment in compliance-as-code prevents massive resource drains caused by reactive, late-stage efforts.

Backsliding Prevention

Vigilant CI/CD guardians automatically flag and block non-compliant changes before they reach production environments.

Scalable Across Programs

Policy-as-code adapts to new regulations (e.g., GDPR, HIPAA) by adding rules rather than rebuilding management systems.

Shift-Left Enhances Speed and Culture

Integrating compliance into the workflow optimizes developer experience and automates audit readiness.

Improved DevEx

Automated feedback reduces cognitive load, allowing developers to address issues early. This results in faster, more confident development cycles.

Simplified Auditing

Every pipeline check is automatically logged, creating an immutable trail of evidence that slashes manual audit preparation time.

Culture of Compliance

Breaks down silos between engineering, security, and compliance, driving collective ownership of system state.

Shift-Left Compliance has Practical Limits

Success requires managing architectural changes, developer fatigue, and organizational culture.

Architectural Shifts

Cannot eliminate the need for fundamental re-architecture when regulations change significantly.

Developer Fatigue

Poorly tuned automated checks can block pull requests and create fatigue with non-actionable warnings.

Cultural Inertia

Frameworks don't solve silos. Success depends on a manual shift in mindset across engineering and security.

Initial Investment

Requires heavy upfront costs to build compliance-as-code and CI/CD integration.

AI Transforms Compliance from a Burden to a Developer Accelerator

Context-Aware IDE Plugins

AI plugins act as real-time compliance coaches within the developer workflow, ensuring adherence to standards before a commit is even made.

AI Generated Fixes

AI can help move beyond just flagging violations; they automatically generate exact fixes for non-compliant infrastructure to eliminate alert fatigue.

Democratized Policy-as-Code

Teams can describe compliance intent in plain English. AI-powered agents then translate this into deterministic, testable policy code.

Conclusion

Moving from reactive remediation to scalable, "born compliant" systems.

Proactive Paradigm

Weaving requirements into the fabric of development so systems are "born compliant."

Efficiency & Sustainability

Bends the cost curve by replacing disruptive retrofitting with incremental updates.

Continuous Enforcement

Automated CI/CD checks prevent drift and enable continuous enforcement.

Cultural Evolution

Breaks down silos between engineering, security, and compliance teams.

Strategic Investment

Compliance as code becomes an enabler of trust and long-term success.

Thank you

Questions?