mondoo

From Bottleneck to Enabler

Policy as Code

for the Modern Enterprise

Executive Summary

Enterprises today face unprecedented security, compliance, and cost management challenges in an era of cloud-native infrastructure and rapid software delivery. Traditional policy enforcement struggles to keep pace, relying on manual processes and fragmented tools. This leads to security vulnerabilities, compliance drift, and unnecessary cloud spend.

Policy as Code (PaC) transforms this landscape by embedding security, compliance, and cost controls as code, enabling organizations to **automate policy enforcement, scale governance, and reduce operational overhead** - without slowing down innovation. Policies are automated throughout the entire software delivery lifecycle and fully integrated with platform and software engineering. They are not restricted to individual technologies or use-cases.

Our experience in the past decade and recent innovations in AI and large language models (LLMs) have changed the way PaC is approached. Security teams often lack software development expertise across the organization, which has hampered the adoption of PaC. The value of PaC lies not only in the use of existing policies, but also in shaping these policies to fit the unique situation and requirements of the company.

As LLMs became more reliable, we have successfully paired them with our pre-compiled, automatically verified policy framework. This has allowed teams to rapidly translate their requirements into code and adjust policies to their individual needs. Thanks to PaC, the output of these AI models is verified and refined with a feedback loop that prevents blunders and increasingly raises the quality of the resulting policies.

This whitepaper delves into the implementation of modern Policy as Code, its successful application, and the outcomes it produces.

Table of Contents

Introduction to Policy as Code

What is Policy as Code?

Policies are all the rules and guidelines an organization has for using IT resources. This includes security (don't get hacked), cost (don't waste money) and operational best practices (don't make a mess).

When we talk about policies **as Code**, we talk about turning these rules and guidelines into instructions that computers can execute. This allows us to easily check if our IT environments follow these requirements with the press of a button.

Codifying requirements

The idea of codifying requirements is entirely new. In fact, the first successful application of "as code" principles started nearly 20 years ago, with the rise of DevOps and tools like Ansible, Chef, and Puppet.

The goals of **Infrastructure as Code** are simple: Automate and standardize the provisioning of infrastructure. Allow us to scale and make it predictable. Help us to work together. And most of all: Stop us from having to manually change configurations in production on a Friday night because someone didn't read the manual.

Infrastructure as code (IaC) was successful.

Today, IaC is widespread everywhere. It helps platform teams to make changes faster, with fewer errors, at a much larger scale. Codifying these requirements also means that more people are able to work together and make changes that are fully tested and reviewed in CI/CD pipelines. IaC surpassed manual processes and tools and allowed organizations to reach unprecedented levels of scale.

These ideas were so successful that they quickly spread to other areas. Before long, security caught on.

As IaC was rising, InSpec became the first solution to market to show how Policy as Code (PaC) could be used to automate security requirements in the same way IaC had helped operations. It was built on the same ideas that had previously inspired platform teams around the world.

PaC is a standalone component. It works incredibly well with IaC solutions like Ansible and Terraform, but it doesn't require them. It adds guardrails to the automation and brings security and platform teams closer together, just like IaC had done for engineering and operations teams.

Over the last decade we have seen the success of the "as Code" approach as new solutions started to grow. In recent years, PaC underwent an evolution of its own, as more teams gathered experience and new technologies became available.

Benefits and outcomes

To understand what it means to be a modern Policy as Code solution, we first have to look at the expected outcomes. What are teams trying to achieve?

The primary goal, as you'd expect, is to **proactively reduce risk by improving the security posture of IT environments**. It sounds simple.

In reality, however, IT has become increasingly complex and the rise of security breaches in recent years has shown us that there are a lot of opportunities to improve the security posture of most companies against internal and external threats alike.

IaC demonstrated how to handle the increasing scale and complexity of infrastructure. It succeeded where countless standalone tools failed because it integrated engineering and operations into a shared software delivery lifecycle. The same approach is required to be effective at security, by making it a part of all aspects of software delivery.

The aspects that make Policy as Code different from other, often standalone solutions, are reflected in its unique approach:

1. Alignment of Engineering and Security teams

If you ever worked in either of these teams you know the story: One team is stuck, because "Security is blocking our release, we can't ship it!", while the security team defends itself: "Can't the platform team just secure these systems, it's not that hard!".

One of the key benefits of Policy as Code is that it brings security teams closer together with platform engineering teams. This isn't done by tools alone, but tools have considerably contributed to the divide we are seeing today. No engineer likes getting overwhelmed with critical tickets that lack proper information. Access to the security context is also often limited and many tools behave like black boxes. Sprinkle a complicated exception process into the mix and you'll have lunch breaks where no-one talks to each other.

Policy as code breaks down this barrier by unifying the stack that all of these teams use. This allows everyone to understand what is expected, contribute and extend it, and ultimately work together. We'll explore this more in the next section.

2. Fast delivery and scalability

Security is not known to be an accelerator of the software development process. However, with Policy as Code, fast delivery of software is one of the expected outcomes.

The benefits of codifying policies is that they can be used in every step of development. Engineers can use them to test components locally, they are deployed in CI/CD pipelines to find issues before they reach production, and they continuously assess all producing and testing environments. These policies meet developers where they are: In their editors, code repos, and their deployment automation.

Policies can express requirements for all stages of this development lifecycle. You can see the overprovisioned user before they are deployed into production. This speeds up testing for developers. By the time they spin up their test gauntlet they will have already found and fixed misconfigurations in their automation.

3. Automate all requirements

Policy as code may have started with security, but it's not limited to it. Because of its incredible flexibility, we have seen teams naturally express other requirements for cost-control and operational best-practices.

Once codified, these best-practices become a reality, instead of fading away in your team's wiki page. Through Policy as Code they are continuously and automatically tested and verified and your team is notified if there is are any issue.

This has helped organizations to catch fatal networking misconfigurations and prevented unused resources from further eating into your monthly cloud spend.

Policy as Code is not limited to cloud environments. It helps with all aspects of IT infrastructure, from company laptops, CI/CD, on-premises IT, hybrid cloud, SaaS, APIs, and even network devices.

Mondoo's Policy as Code

Mondoo's Policy as Code consists of YAML-based policies that contain a set of assertions called "checks" written in MQL, a lightweight test engine for platform and security teams. A check is used to verify a specific requirement automatically. Queries are added to collect contextual information in a graph-based asset inventory.

Essential Evolution

On paper, Policy as Code sounds like a no-brainer. In practice, most solutions fall short. They're often too complex, too rigid, or too disconnected from how teams actually work, making widespread adoption surprisingly difficult.

Complexity

Most Policy as Code solutions were originally created for developers, requiring users to learn not just the framework itself, but also general-purpose programming languages like Ruby and Python. In reality, the audience for these tools is much broader, platform engineers, operations teams, security analysts, and auditors, many of whom don’t have a software engineering background.

Organizations often lack dedicated expertise that would allow them to scale these solutions. Making these policies accessible to all team members, no matter their programming skills, is vital to make it work.

Mondoo’s MQL was designed from the ground up to be accessible to security teams. Inspired by the simplicity and accessibility of tools like Terraform, we removed unnecessary complexity to ensure that both programmers and non-programmers can use it effectively. When in doubt, ask the security team.

This has led to a structure that makes it very easy to read and use:

```
# Ensure AWS EC2 instances do not have a public IP address
aws.ec2.instances.all( publicIp == empty )

# Only allow approved ciphers in the SSH server configuration
sshd.config.ciphers.containsOnly( props.ApprovedCiphers )
```

Effort

The second major hurdle is creating a set of policies that reflect the organization's needs and requirements.

Mondoo comes out of the box with a rich library of common policies, industry best-practices and compliance frameworks. However, we also understand that every organization is unique and teams want to take advantage of the flexibility that PaC provides and express their own requirements.

Writing all of these policies wasn't easy - until now.

Generating and updating policies via LLMs

Thanks to recent developments, large language models (LLMs) have become more useful. Mondoo policies are written in YAML, which is easy to build and fill with content thanks to LLMs today.

More importantly, Mondoo's testing engine, MQL, is pre-compiled, statically typed and fully schema-based. This allows us to pair it incredibly well with AI, making sure it works as intended and understand the data that it will process.

Mondoo delivers a pre-trained policy generator that can be used to create new policies or update existing ones.

The combination of LLM and MQL combines the best of both worlds. Mondoo is able to ingest requirements and turn them into policies thanks to the generative capabilities of LLMs. MQL complements it perfectly, by validating the generated content, allowing users to collaborate on it and test it continuously to avoid unintended behavior.

Thanks to Mondoo's engine, users can create snapshots of their environments' setup and configuration to validate any policy changes they make in the future. This level of testing and validation removes a layer of uncertainty and weird behavior that we often see in AI.

Create great Policies

In this final section we will look at how Policy as Code has evolved and what great policies look like today.

Unified across pipelines

A software pipeline to deliver modern infrastructure often contains code for frameworks like Terraform and Ansible. Unified policies can check requirements at all stages of the development process, including their IaC configuration and provision systems.

For example: If you require all public buckets to follow a mandatory labeling schema, you can enforce it in both Terraform HCL as well as every cloud this is rolled out to. A single check can handle all variants of this requirement and test it locally, in CI/CD and in production.

Unified policies make changes to the check or exceptions a lot easier than distributed policies. Everything is one place, whenever parameters are changed.

Properties can be bundled with the policy, so that changes affect all variants of the query. For example: If you want to change a mandatory tag, the change to the property will affect all tests equally. This way things are consistent between your IaC code and the various cloud or on-prem deployments.

Exception Management

Exceptions are where most Policy as Code strategies fall apart. In the real world, exceptions are inevitable—risk acceptances, mitigations, and false positives must be tracked, reviewed, and enforced. But in most organizations, these exceptions are handled manually across scattered tools, email threads, and siloed pipelines. The result? Delays in shipping, wasted engineering cycles, inconsistent enforcement, and mounting operational costs. Every out-of-band exception request represents lost time, increased risk, and real money left on the table.

Mondoo’s unified Policy as Code solution makes exception handling consistent, immediate, and universal, no matter where policies are evaluated. Whether running locally during development, in CI/CD pipelines, or in production runtime, approved exceptions take effect automatically. Developers see exceptions applied in local development in the same way they are handled in production.

There is no need to manage exceptions in multiple systems or reinvent approval processes for each environment. As a core component of the Mondoo platform, exception management allows teams to define overrides once, review and approve them securely, and enforce them everywhere - eliminating redundant effort, reducing risk, and saving valuable time and money.

Exceptions can also be preloaded into the system. Whenever we define an exception that applies to e.g. a specific team, a set of services, or assets with certain tags, we add it as an extension to the policy. These exceptions are automatically applied to components whenever they match the requirement. Thus, these exceptions become a part of the policy itself.

Exceptions and policies are inevitably linked together. Ultimately, a tailored policy allows users to embed these rules and scale them consistently across their teams and infrastructure.

Extensibility

Policy as Code isn't limited to individual systems or technologies. It is a way to express requirements for all kinds of assets.

To provide this level of flexibility, Mondoo's PaC engine is fully extensible. Any technology you can connect to can be added as a provided. All resources and fields are defined in a schema, which make it easier for others to use it, avoid errors, and produce high-quality prompt (LLM) results.

Context and relationships

Most policy frameworks only deal with one system and one set of data at a time. However, we work with increasingly interconnected objects in modern IT environments. These need to be expressed in policies.

Mondoo's MQL provides full access to the underlying resources graph of your infrastructure. This allows complex relationships to be expressed and queries. For example: Create a check that allows only certain users to open ports on internet-facing systems.

The context of resources and assets often influences how we evaluate risks. Mondoo's security model builds on Policy as Code which makes it fully configurable. For example: Critical systems, identified via their tags, should not contain remote code execution vulnerabilities.

Simplicity

Finally, to be successful at Policy as Code, it needs to be simple and accessible to all involved users. This is done via comprehensive policies with remediations, audit steps, and readable MQL.

Thanks to Mondoo's pre-trained LLM, more users than ever before can access and contribute to policies. This makes it feasible for far more security organizations and allows them to work more effectively with their platform and engineering teams.

Mondoo Case Study

A leading Fortune 500 corporation operates a vast, highly distributed technology environment spanning on-premises data centers, multiple cloud providers, and thousands of connected devices supporting its supply chain, manufacturing, and software-defined technologies. The company's engineering teams manage thousands of infrastructure components across Kubernetes clusters, cloud services, and embedded systems, requiring strict governance over security, compliance, and cost controls.

Challenges: Fragmented security and compliance enforcement

As the company accelerated software-driven innovation, its security and compliance teams faced several roadblocks:

- Inconsistent policy enforcement across on-prem, multi-cloud, and hybrid environments.
- Manual security audits and compliance checks were delaying development cycles.
- Lack of visibility into infrastructure misconfigurations, leading to security drift.
- Cost overruns due to ungoverned cloud resource provisioning.
- Cross-team friction between security, DevOps, and application teams regarding policy implementation.

With infrastructure scaling rapidly, the company needed a centralized and automated solution to enforce security, compliance, and financial guardrails across its entire ecosystem without slowing down development.

Solution: Mondoo Policy as Code

The corporation implemented Mondoo's PaC framework to automate and standardize security, compliance, and cost controls across its global technology infrastructure. By implementing Mondoo, the company achieved the following:

- Unified policy enforcement across cloud, on-prem, and Kubernetes environments, ensuring a consistent security posture.
- Automated security guardrails embedded into CI/CD pipelines, eliminating misconfigurations before deployment - without slowing down development.
- Compliance automation for industry standards (ISO 27001, NIST, NIS2, DORA, and SOC 2), reducing manual audit workloads.
- Cloud cost control policies that enforced instance sizing, scaling limits, and mandatory resource tagging to prevent overspending.
- Cross-team alignment through a single policy engine, reducing friction between security, DevOps, and IT teams.
- Fully automated the onboarding process (with Mondoo Terraform Provider), enabling rapid adoption and enterprise-wide scalability of the Mondoo platform.

Business outcomes:
Improved security, lower costs, and higher efficiency

By adopting Mondoo's Policy as Code platform, the company realized significant improvements:

- **90% reduction in security misconfigurations** across cloud and Kubernetes environments.
- **40% improvement in compliance audit readiness**, cutting manual policy enforcement efforts.
- **20% reduction in cloud infrastructure costs** through automated cost control policies.
- **Thousands of hours saved** in manual work performing audits, onboarding systems, and configuring policies.
- **Faster software delivery** with security checks integrated into development workflows rather than being reactive.

By adopting Mondoo Policy as Code, this Fortune 500 corporation transformed its security, compliance, and cloud governance strategies—enabling faster innovation, reducing risks, and optimizing cloud spending at scale. Mondoo's ability to enforce policies consistently across hybrid infrastructure allowed the company to shift from reactive security to proactive, automated governance while maintaining development velocity.

Summary

Policies covering important requirements, best-practices and guidelines can be automated thanks to Policy as Code. It leverages two decades of proven ideas from infrastructure automation to increase velocity, scale, and reduce manual errors.

Mondoo's approach to Policy as Code evolves the practice and makes it accessible to more organizations. It is built around simplicity and accessibility, incorporates LLMs for content-generation, and validation for quality control.

As the variety of IT and its use-cases continue to increase, Policy as Code allows us to keep pace with innovation without losing control.

Schedule a demo and learn more about Mondoo:

https://mondoo.com/contact