

WHITE PAPER

# Key Considerations: Authorization-as-code Accelerates Policy-as-code

AXIOMATICS
a LEONARDO company

## The ABCs of ABAC

Before delving into why the policy-as-code approach is becoming so popular, it's important to understand what organizations are ultimately trying to achieve. At the end of the day, it comes down to ensuring appropriate access controls are in place so an organization's sensitive applications and data are not exposed or compromised. To do this effectively, leveraging attribute-based access control (ABAC) ensures dynamic authorization is done at runtime.

ABAC is important because it ensures when a request comes in, the decision to approve is not simply based on a user's role. Instead, the organization's policies are the logic and attributes are the inputs used to determine an appropriate decision. The list of attributes can be quite extensive but the point is it's not a simple "yes" or "no" based on the user's role. ABAC considers what the user can do with that access, what they can see (i.e. maybe data is masked when the user is not on VPN), the relationship logic of the requestor to the subject of the data they are requesting, and so forth. If an organization's end goal is to ensure appropriate access controls, leveraging ABAC is a significant step to achieving this goal, while also enabling end-users to continue to do their jobs. Without ABAC, a singular, focused policy might not allow an employee who is not on VPN to access the application they need to do their job. Instead, a policy can be written with ABAC that still provides end-user access to the application with caveats that could include masking sensitive data. This ensures protections are in place, without disrupting business continuity.

## Why not code access policies into the application itself?

For many organizations, the current default method to achieve access control is to embed some security logic (likely based only on the user's role) into the application code itself. Since most organizations now use or create upwards of hundreds of applications, developers focus on the most sensitive of them and before an application is rolled into production, manually code some form of access control. Not surprisingly, this is problematic for several reasons.



### Developers are in high demand

Developers are highly specialized, valuable resources, which makes them in high demand. Having them spend time coding access controls into each application means they focus less on other projects that may directly generate revenue and/or increase organizational efficiency.



### The need for multiple attributes

More often than not, this one-off custom access control that a developer is coding into the application is not based on context derived from multiple attributes. Remember, this is tedious work and it's taking them away from more critical projects so in most cases, the user's role is the only attribute that will be taken into account. This doesn't consider any logic or "gray" areas which means it usually only results in a "yes'' or "no" decision. That is, "yes", the user can be let in, or "no," access is denied.

### Reacting to change

While efficiency and the ability to make changes later on. Imagine if an organization had the staffing resources to actually code in access policies to all their hundreds of applications. Now imagine a major government in one of the countries they operate in passes a new compliance bill and they now need to go back and make a change to the access policy in each one of the hundreds of applications. That is going to require a great deal of time and resources. It just doesn't make sense to tackle things in this manner.

## Why not move the enforcement point out of the application?

If an organization is going to shift to the ABAC approach, they are going to have to leverage policy enforcement points (PEP). PEPs are made reference to in security guidelines such as NIST (SP 800-2007) that delve into the concept of Zero Trust. The PEP is the point at which the policy logic is applied and it communicates with a policy decision point (PDP).

For every different application, the common approach is that an API will need to be created and leveraged to act as a PEP. Like the previous approach of coding directly into the applications themselves, this is a lot of work as there will be hundreds of APIs needed. Additionally, any time a change is required, the development process must be restarted for each API for every application. This approach increases control and security as it's easier to account for more attributes with APIs, however, the effort has just shifted the burden of hard coding RBAC into applications and replaced it with coding ABAC into APIs. This approach is not sustainable for many of the same reasons discussed earlier, and keeping track of a variety of API implementations and changes is difficult at scale.

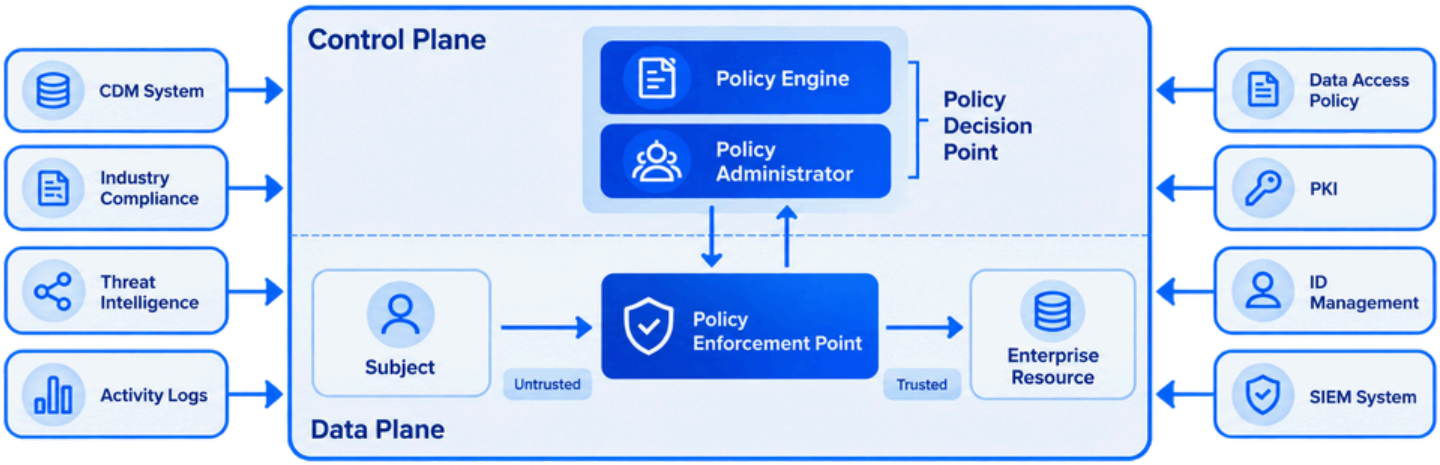



Fig. 1: The above diagram is based on a section of NIST SP 800-2007

## Enter policy-as-code

Until this point, both approaches have been tedious and involved many developers doing repetitive tasks. The policy-as-code approach changed this. While development teams still wrote policies, they uploaded and shared their code with teams across the organization and leveraged some version control. This increased efficiency because it meant teams could now make copies and modify existing code. In addition, by treating it as part of the DevOps process, changes could be tested before being rolled out to production. It sounds like common sense, but this moved what could be perceived as a very siloed, ad-hoc course to a new approach that brought visibility, shared resources and put proper controls in place.

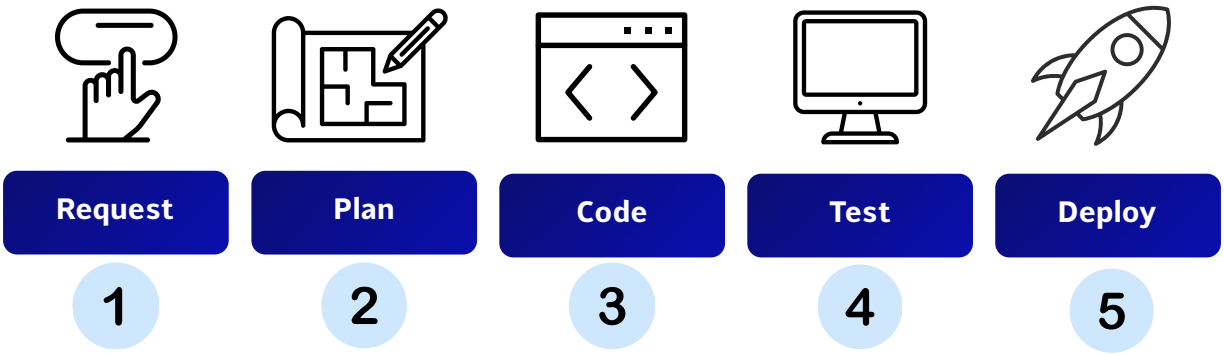



Fig. 2: Policy-as-code process takes the same approach as other code updates. The request comes in from the business, it is reviewed and a plan decided, followed by coding, testing and finally deployment.

In terms of the actual programming language used for writing policies, it usually depends on the tools the organization leverages and could include Rego, Python, YAML etc. However, if coding policies that are specifically about authorization, there is a domain-specific language specifically created for writing access-control policies. ALFA, (available via the Organization for the Advancement of Structured Information Standards) is a language organizations may adopt to develop enterprise authorization policies that include which attributes need to be considered, under what conditions and even in relation to each other. Since ALFA was purpose-built for writing fine-grained authorization policies, it takes the policy-as-code approach a level deeper and the term authorization-as-code makes more sense here.

The benefits of a domain specific language for authorization is that such a language has the terms and features to express authorization policy in a manner which closely fits to the way humans and organizations think of authorization. For instance, ALFA has a special data type for a decision, which can be Permit, Deny, NotApplicable or Indeterminate. This is directly how people think of authorization. A policy can Permit or Deny a special action, or the policy might not matter for this situation. "Indeterminate" is used when there is an error in the system. ALFA also allows grouping of policies based on topics in a hierarchy and priority is expressed based on natural concepts like "deny has priority over permit", etc. Overall this means that policies in ALFA have a declarative form which is close to the organizational requirements, which makes it easy to understand and organize the policies. In contrast with a general purpose language, programmers have to implement these concepts themselves. Also, the restricted, functional nature of ALFA policies makes reverse query possible, which helps with review and analysis policies.

## Authorization-as-code drastically increases speed policy

While the policy-as-code approach provides new efficiencies, authorization-as-code kicks things into warp speed. The authorization-as-code approach takes the ABAC process out of APIs and applications, and externalizes it into one central access decisions service (ADS) for all applications within an organization. This means instead of every development team mastering the ABAC concept, they simply write a standard API that connects to the ABAC REST API. That way, whenever a new application comes on board or a change needs to be made, one development team can easily do that coding in one central location. This also means that if a change is required due to a new compliance law, it can be made in one location for all applications that have policies affected by the change.

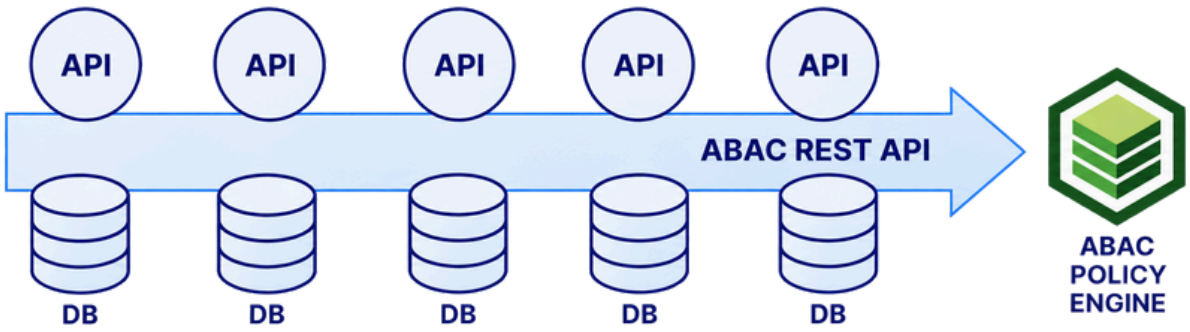
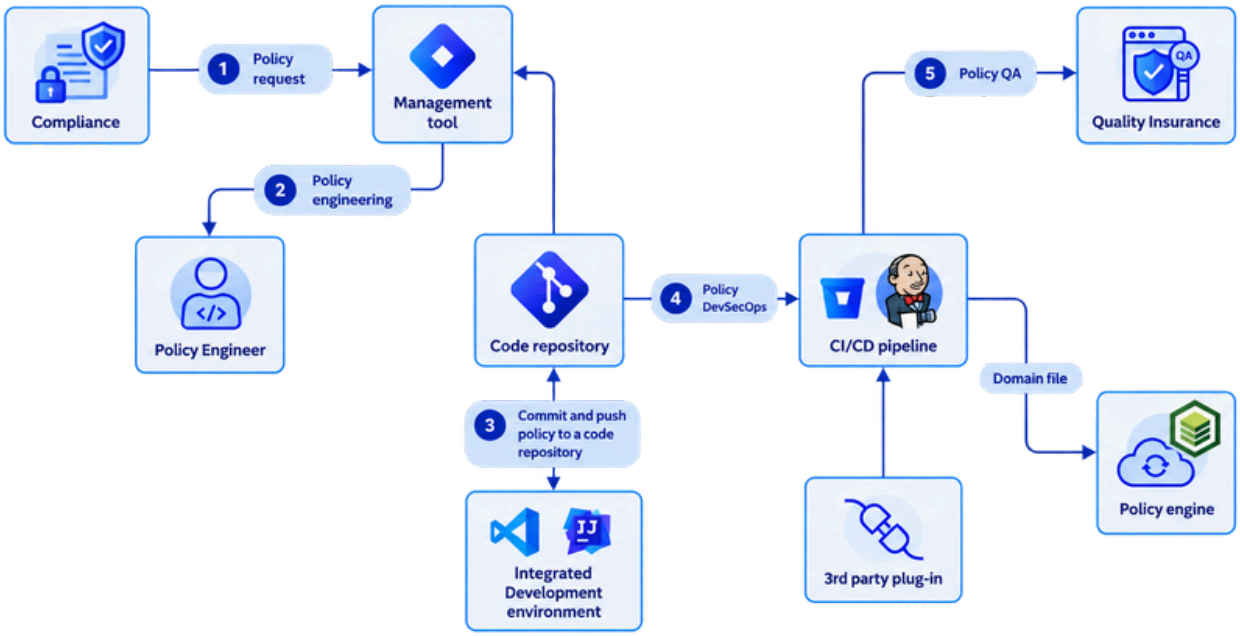



Fig. 3: Whenever a new application comes on board or a change needs to be made, one development team can easily do the ABAC coding in one central location.

This process is still engaged as a standard DevOps process and treated as code. A change can be requested in whatever system the organization uses (i.e. Jira), the developer can access, commit and push to the source repository and then it can be treated as regular code changes through the CI/CD process before being tested by the quality assurance team. The difference is that instead of every development team doing this for every application, requiring the application server to push the change out to each application separately, only one team works on this, and the result t is leveraged across the rest of the organization by all applications via the ADS server, without touching any application code. Consider the efficiency and agility such an approach provides, not to mention the security benefits of being able to deploy a new policy - fast.



# Take the next steps

Learn how how a security vendor can help secure your access control while allowing you to focus on your core business initiatives.



Meet with our solution experts for a demo of our Axiomatics Authorization Management Platform and discuss how our approach can save your organization time and money.

## Stay connected and informed

Get the latest insights, announcements, and assets from our thought leaders and customer success team:

- Blog articles
- Press releases and announcements
- Resources, solution briefs, and tools
- Join us on LinkedIn
- Subscribe to our Linkedin Newsletter
- Subscribe to our YouTube channel



## About Axiomatics

Axiomatics - a Leonardo Company is the originator and leading provider of runtime, fine-grained authorization delivered with attribute-based access control (ABAC) for applications, APIs, data, and AI systems. The world's largest enterprises and government agencies rely on the Axiomatics Authorization Management Platform to centrally define and enforce dynamic access control.

Learn more or request a demo of our solution:

axiomatics.com
info@axiomatics.com
US Office: +1 (312) 374-3443
Europe Office: +46 8 51 510 240