How MongoDB Atlas Scaled Authorization with Cedar-Based Resource Policies

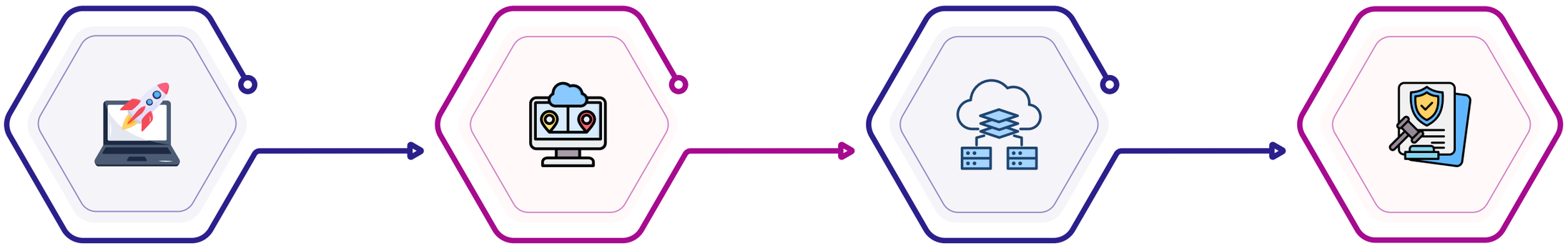
Authorization has a funny way of starting simple and getting complicated fast. A handful of user roles turns into a sprawling rulebook that cuts across services, regions, and compliance regimes. Stuffing that logic inside application code only makes things worse — policies end up scattered across repositories, nearly impossible to audit, and painful to update without a full release cycle.

The problem has grown sharper in the AI era. Modern applications pair transactional databases with vector stores, retrieval pipelines, streaming ingestion, and model inference services, each carrying its own governance boundary. Infrastructure changes faster than hand-coded permission checks can keep up with. To serve enterprise customers — including roughly three quarters of the Fortune 100 — MongoDB needed an authorization layer that could move at the pace of modern software. Its answer: Cedar.

From RBAC to Resource Policies

MongoDB Atlas has long relied on role-based access control. Permissions are granted across three tiers: organization, project, and cluster, with predefined roles such as Organization Owner and Project Cluster Manager. That model served customers well for years because it mapped neatly onto how enterprises structure their teams.

But as Atlas became the system of record for mission-critical workloads at companies like Coinbase, Deutsche Telekom, and Novo Nordisk, a limitation surfaced. RBAC answers “who can do what.” It cannot answer “under what conditions.” Platform teams running hundreds of projects and thousands of clusters needed to prevent problems that had nothing to do with identity — for instance:



Clusters deployed in regions that violated data residency laws.

IP access lists opened to 0.0.0.0/0, putting databases on the public internet.

Undersized instances that could not sustain production traffic.

TLS settings below the minimum versions demanded by compliance auditors.

RBAC could not close this gap without turning into an unmanageable maze of hyper-specific roles.

Weighing the Alternatives

MongoDB considered three options before settling on Cedar. Expanding RBAC would have meant creating a role for every possible combination of cloud, region, tier, and compliance rule — dozens quickly become thousands, and policy management collapses under its own weight. Embedding validation checks directly in Atlas provisioning code would have spread authorization logic across the codebase, turning each new constraint into a release-cycle event. Building a custom policy language from scratch would have pulled engineering attention away from MongoDB's core work in databases and required years of investment in parsers, validators, and security hardening.

Why Cedar

Cedar, an open-source policy language developed by AWS and now a Cloud Native Computing Foundation sandbox project, offered a cleaner path. It separates policy decisions from application logic, expresses policies as declarative, human-readable code, and supports both role-based and attribute-based rules. AWS also runs Cedar as a managed service through Amazon Verified Permissions, so the engine has been hardened against real multi-tenant workloads.

Three things made Cedar the right fit for MongoDB. First, separation of concerns: engineers maintaining cluster lifecycle APIs no longer need to understand every customer's compliance rules, and security teams can author policies that evolve independently of Atlas release cycles. Second, auditability: because Cedar policies are text, customers can version them in Git, review them alongside Terraform, and see exactly which guardrails apply to which resources. Third, open-source maturity: Cedar ships with formal verification, IDE integrations, a validation API, and a growing community — none of which MongoDB would have had time to build on its own.

Atlas Resource Policies in Practice

Atlas Resource Policies, introduced in 2025, let administrators define organization-wide guardrails. A single policy can enforce geographic restrictions, block wildcard IP ranges, require a minimum TLS version, cap cluster tiers or disk sizes, or restrict which cloud providers may be used.

01

The distinguishing feature is scope. Policies are authored at the organization level but evaluated against project- and cluster-level attributes. One policy can assert that no cluster anywhere in the organization may run TLS below 1.2. Governance is centralized, but enforcement is distributed across every provisioning path.

02

The developer experience also improves. RBAC denials typically say only “you do not have permission.” A Cedar-based rejection can say “your configuration violates policy XYZ.” Developers get immediate, actionable feedback rather than a dead end.

How the Evaluation Works

When a user creates or modifies a cluster, Atlas first runs its RBAC check. If the caller has the right role, Atlas gathers contextual attributes for the request — things like the cloud provider, target region, number of clusters already in the project, and the configured TLS version. Those attributes are handed to an embedded Cedar evaluator, together with the organization’s resource policies. The evaluator returns a binary decision, and Atlas either proceeds or blocks the request with a clear error message that names the violated policy.

A Concrete Example

Suppose a platform team wants clusters deployed only in us-east-1 and us-west-2. In the Atlas console, they open the organization’s Resource Policies page, click Create Policy, name it “Limit Access to certain AWS regions,” and paste in a short Cedar snippet describing the geographic restriction. Once saved, the policy is live. Atlas assigns it a unique ID that can be referenced by the Atlas Administration API or managed through Terraform. Anyone trying to spin up a cluster outside those two regions receives an Unsuccessful Deployment error that identifies the violating policy and the allowed regions, so the developer can fix the configuration on the spot instead of escalating a ticket.

Why It Matters

More than 1,500 organizations have adopted Atlas Resource Policies. For customers, the benefit is simple: platform teams get audit-ready guardrails while developers keep self-service speed. For MongoDB, policy updates no longer require code changes, testing, or deployment — they are declarative artifacts, iterated at the pace of compliance rather than software releases.

MongoDB plans to extend Cedar coverage to authentication methods, customer-managed encryption, and Atlas Search configurations. For any SaaS company building a multi-tenant platform, the pattern is worth studying: adopt a declarative policy engine, treat authorization as code, and enforce decisions at runtime, cleanly separated from business logic.