Authz as a dev workflow: Architecting better cloud-native apps

- Dan at Cerbos
- KubeCon London 2025

About Me

- Open source nerd
- Oldschool sysadmin (devops, sre, etc)
- Really into opsec

The Authorization Paradox

- Authorization is critical for every application
- Often treated as an afterthought
- Disconnect between importance and implementation

The Evolution of Developer Primitives

- Consider how cloud-native development has evolved
- Networking: From manual configuration to service mesh and CNI
- Storage: From volume management to persistent volume claims and CSI
- Deployment: From manual scripts to GitOps and CI/CD pipelines
- Each evolved from infrastructure concerns to developer primitives
- Each transformation made developers more productive while improving quality

Why Authorization Has Lagged Behind

- Deep coupling with application logic: Authorization often lives inside business logic
- Domain-specific requirements: Financial services vs. SaaS vs. healthcare
- Complex relationships: Modern applications have intricate permission models
- No one-size-fits-all solution: Each framework reinvents authorization
- Result: Authorization remains stuck as an infrastructure concern rather than a developer primitive

The High Cost of Complexity

- Cognitive overhead: Developers constantly context-switch between business logic and auth
- Security gaps: Inconsistent implementation leads to vulnerabilities
- Velocity impact: Authorization changes require extensive regression testing
- Onboarding friction: New team members struggle to understand permission models
- This is both a security problem and a developer experience problem

The Workflow Opportunity

- What if authorization was a creative tool rather than just a constraint?
- Low friction is essential for good developer experience
- Authorization can be declarative rather than imperative
- The right abstractions can make authorization intuitive and maintainable
- Transition to workflow-first thinking opens new possibilities

Authorization as a Foundational Architectural Concern

- Traditionally, authentication gets architectural attention, authorization becomes scattered if/else statements
- The workflow-first approach makes authorization decisions first-class architectural elements
- This creates clear boundaries between business logic and permission logic
- Authorization becomes a service rather than embedded code
- Example: Netflix's approach to centralizing authorization for their microservices ecosystem

From Security Constraint to Product Feature

- Traditional question: 'What can't users do?' (negative framing)
- Better question: 'How do we enable the right access?' (positive framing)
- Permissions become part of product discussions - not just security reviews
- Authorization drives user experience through progressive disclosure
- Example: How Slack's workspace permissions enhance collaboration rather than just restrict it

The Cost of Afterthought Authorization

- Context switching kills developer flow
- Tightly coupled authorization is resistant to change
- Hard-to-test permission scenarios lead to bugs
- Architecture that doesn't scale with your organization

Core Design Principle: Domain-Driven Authorization

- Model permissions around business domains not technical constructs
- Permissions should speak the language of the product, not the codebase
- Ubiquitous language between product, development, and security teams
- Example: Document permissions that model 'Editor' and 'Reviewer' roles instead of CRUD operations
- This aligns security with how users think about your application

Core Design Principle: Declarative Policies

- Move from imperative checks scattered throughout code
- To declarative policies that describe intent
- Policies become human-readable artifacts
- Enables version control and review of authorization logic

Core Design Principle: External Decision Points

- Decouple authorization from application logic
- Application asks 'Can this user do this?' to a dedicated service
- Enables consistent enforcement across services
- Allows independent scaling of authorization logic
- Makes authorization logic testable in isolation

Core Design Principle: Context-Awareness

- Move beyond simple role-based access control
- Consider attributes like time, location, resource properties
- Authorization that adapts to context is more flexible
- Example: Different permission rules during business hours vs. after hours
- Enables fine-grained control without explosion of roles

Integration Points in the Development Lifecycle

- Requirements gathering: Define permission models alongside features
- API design: Document authorization requirements in API specs
- Implementation: Clear interfaces for authorization checks
- Testing: Dedicated test suites for permission logic
- Operations: Monitoring and observability for authorization decisions
- This creates a continuous thread of authorization awareness

Developer Experience Benefits

- Reduced context switching between application and security code
- Clear contracts between components
- Self-documenting permission models
- Faster onboarding for new team members
- Confidence that permission changes won't break functionality

The Authorization Ecosystem

- A rich ecosystem of open source tools has emerged
- CNCF projects leading standardization efforts
- Different tools for different architectural approaches
- All share the goal of making authorization more manageable
- Each addresses different parts of the authorization challenge

Open Policy Agent (OPA)

- CNCF graduated project for policy enforcement
- General-purpose policy engine beyond just authorization
- Rego policy language for expressing rules

OpenFGA: Fine-Grained Authorization

- CNCF Sandbox project focused on relationship-based authorization
- Based on Google's Zanzibar paper - same system that powers Google Drive permissions
- Excels at modeling arbitrary relationships between objects

AuthZen (OpenID Foundation)

- Working group standardizing authorization APIs and interface
- Building on success of OAuth 2.0 and OpenID Connect
- Focus on interoperability between authorization systems
- Important for long-term evolution of the authorization ecosystem

Cerbos: Developer-Centric Authorization

- Human-readable YAML policies that product teams can understand
- IDE integrations for real-time policy validation
- Designed for frictionless developer workflows
- Particularly strong for testing and debugging authorization logic

Choosing the Right Tool

- No one-size-fits-all solution
- Consider your architecture and team structure
- OPA: Broad policy enforcement beyond just authorization
- OpenFGA: Complex relationship-based permissions at scale
- Cerbos: Developer workflow integration and simplicity
- Many teams use multiple tools for different aspects

Pattern Categories Overview

- We'll explore three categories of practical implementation patterns
- Architectural patterns: How authorization fits into your system architecture
- Development patterns: How to integrate with developer workflows
- Testing patterns: How to verify authorization logic effectively
- Each pattern addresses a different aspect of the authorization challenge

Architectural Pattern: Policy as a Service

- Encapsulate authorization logic in a dedicated service
- Applications make explicit authorization requests
- Enables consistent enforcement across services
- Simplifies policy updates without changing application code
- Example: implementation with Cerbos REST API or gRPC service

Architectural Pattern: Sidecar Enforcement

- Deploy authorization engine alongside each service instance
- Low latency for authorization decisions
- No network hop for every decision
- Maintains consistent policies via centralized distribution
- Perfect for Kubernetes environments
- Example: Envoy + OPA service mesh integration

Architectural Pattern: Multi-Layer Authorization

- Authorization at multiple system layers
- Coarse-grained at API gateway or ingress
- Service-level for business logic authorization
- Data-level for fine-grained access control
- Each layer handles what it's best suited for
- Example: Combining API gateway rules, OPA service policies, and row-level security

Development Pattern: Policy-Driven Design

- Start with authorization requirements before implementation
- Define policies in domain-specific language
- Generate test cases from policy definitions
- Use policies to drive API design
- Enables clear separation of business logic and authorization
- Example: Cerbos YAML policy defined before service implementation

Development Pattern: Request Context Enrichment

- Augment requests with rich context information
- Move beyond simple user IDs to full context objects
- Context includes user attributes, resource metadata, environmental factors
- Enables sophisticated, context-aware decisions
- Implementation using request middleware or interceptors

Testing Pattern: Policy Unit Testing

- Test policies independent of application code
- Table-driven testing for multiple scenarios
- Verify both allowed and denied cases
- Test for edge cases and unusual conditions
- Example: OPA's testing framework or Cerbos test utilities
- This catches policy bugs before they reach production

Testing Pattern: Authorization Test Fixtures

- Create reusable test users with different permission profiles
- Standard test resources with various access patterns
- Authorization-aware test helpers for common operations
- Makes writing permission-aware tests much easier
- Example: Test fixture library that sets up complex permission scenarios

Practical Workflow Integration

- Local development with policy simulation
- IDE plugins for real-time policy validation
- PR checks for policy consistency
- Documentation generation from policies
- Debugging tools for authorization decisions
- Creates a seamless experience from dev to production

Key Transformations

- We've explored three fundamental transformations in how we approach authorization
- From friction to flow in the development process
- Security by design rather than security as an add-on
- Empowered developers making better security decisions

From Friction to Flow

- Traditional approach: Authorization interrupts development flow
- New approach: Authorization tooling enhances productivity
- The difference is measurable in developer satisfaction and output
- When authorization is seamless, creativity flourishes

Security by Design

- Security isn't something you add at the end
- Authorization primitives built into your architecture from day one
- This shift fundamentally changes the security posture of your applications
- The result: more secure systems that are easier to maintain

Empowered Developers

- Knowledge is power: developers who understand authorization make better decisions
- Clear patterns and tools democratize security expertise
- This creates a positive security culture across development teams
- When developers are empowered, the entire organization benefits

What Next?

- Start with one workflow enhancement
- Join the community building better authorization primitives

Thank You

- Authorization should enable great software, not hinder it
- The tools and patterns exist today to transform your approach
- Thank you for joining me on this journey
- Brief invitation for questions