Building a scalable authorization system: a step-by-step blueprint

Written by
Emre Baran, Cerbos CEO

cerbos

Table of contents

Chapter 1

Why does any application need an authorization layer?

Authorization is the backbone of any secure application. It determines what actions a user can perform within an application. Authorization ensures users can only access what they are allowed to. As software applications scale, authorization often becomes more complex, especially when dealing with microservices or distributed systems.

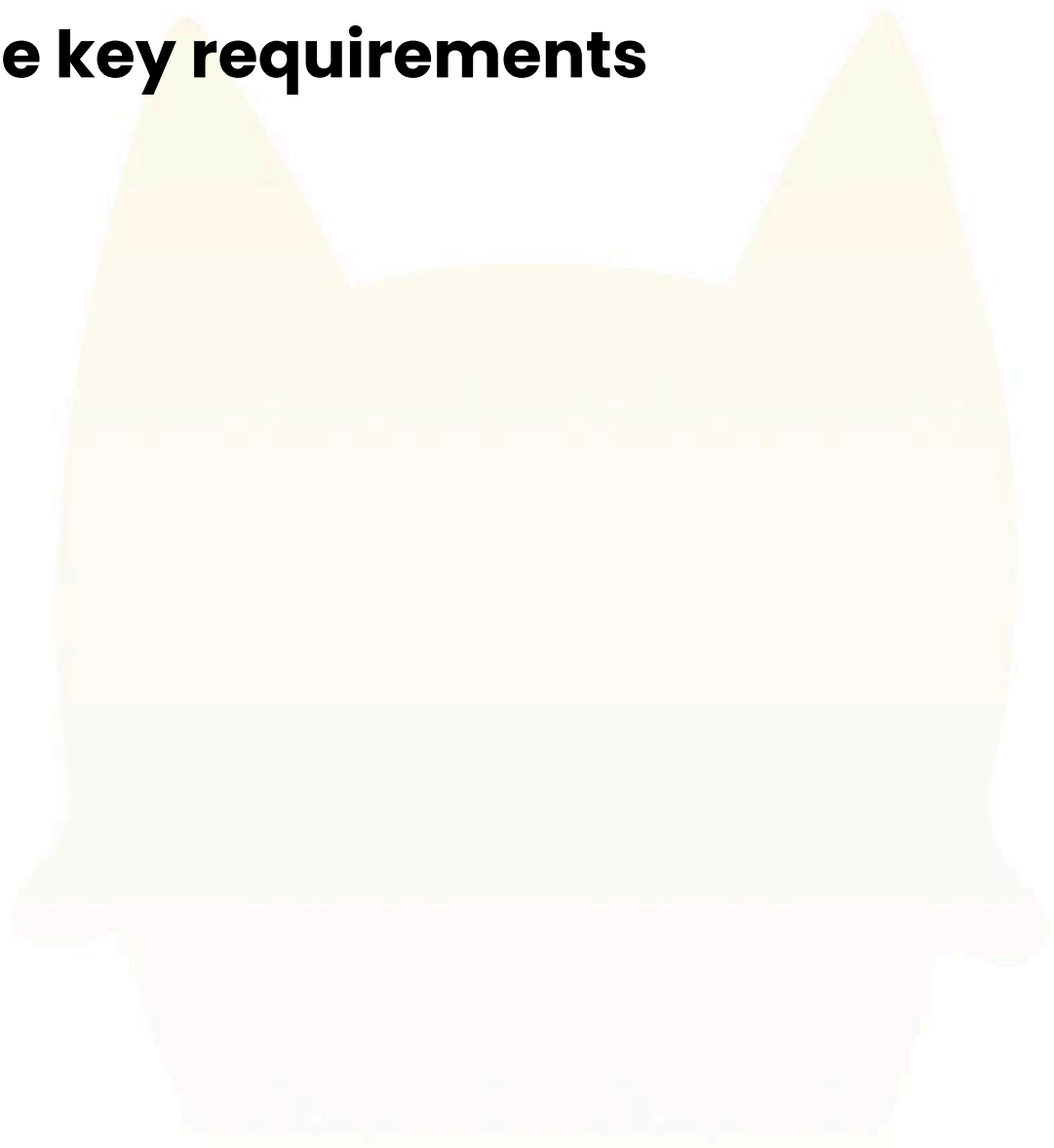
Alex Olivier (ex-Microsoft), Charith Ellawala (ex-Elastic), and I (ex-Google) founded Cerbos in 2021. Among us, we have years of experience building software applications in telecom, financial services, logistics, social networks, and in martech. At every company, we found ourselves building, and then rebuilding the authorization layer. No matter how hard we looked, we couldn't find a suitable off-the-box authorization solution we were happy with.

Therefore, we decided to build an authorization solution, Cerbos, that anyone can use. While building Cerbos, we gathered many requirements for building a versatile and scalable authorization layer. This blueprint is about those key requirements we've observed and how we approached them while building Cerbos.

If you are building your own authorization layer, we hope the requirements we observed, decisions we've made and the technologies we chose can serve you as important and useful points to consider.

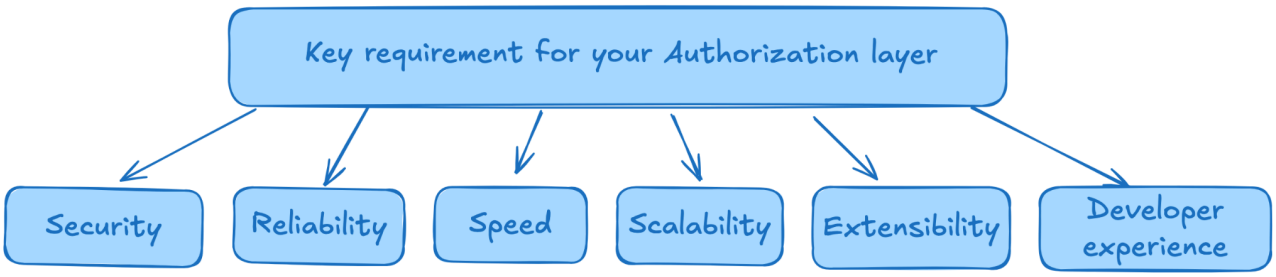
Chapter 2

The key requirements



Before building Cerbos Policy Decision Point, PDP, and Cerbos Hub, we spoke with over 500 developers, architects, IAM leads, EMs, and engineering leaders to understand their requirements for a secure, flexible authorization layer. From those interviews, we created what we call "**The Blueprint**"—a list of the most vital requirements.

Here's a high-level look at what an ideal authorization layer needs:



Let's dive deeper into each of those requirements.

Key requirement 1 – Security

The security of your authorization layer is of utmost importance. It should be your frontline of defence against incidents. Any vulnerability in AuthZ can compromise the entire software and lead to serious security breaches.

While security is vital for protecting your clients, employees, and company, it's also important for regulatory compliance. This is especially true if your company operates in the fintech, banking, insurance, medtech, or government industries. Security vulnerabilities, or even uncertainties, can make compliance a nightmare:

> *"Some of the compliance checks require us to have security policies up to date. Being able to demonstrate that an authorization provider, like Cerbos, is handling authorization for us is a great convenience,"* said Mohsin Kalam, CTO & Co-Founder, Loop

Given the complexities of global markets and regulatory rules like SOC2, ISO27001 and CCPA, a secure authorization serves both as a defence mechanism and a strategic tool.

Key Requirement 2 – Reliability

Unlike authentication, which is only used once in a session, your authorization layer is constantly making decisions as a user interacts with your software. It decides which user can access what data and sometimes even when they can access it. If it is unreliable, either users will not be able to access the data they need or users will be able to access data they shouldn't. That creates a lot of risk in your software which will affect your ability to serve demanding enterprise clients.

Mature, enterprise clients are risk-averse. They need a reliable experience from a solution so they can offer the same to their clients. If your security and compliance (including authorization) becomes a problem, it doesn't matter how good your product is, companies will not risk using it in production:

> *"If you want to sell to big companies, you can't have tedious or fragile disaster recovery programs. You have to be able to prove, not only where your data is flowing, but also that you control the foundations of the house,"* said Chuck Hardy, Head of Engineering, Salesroom

A reliable authorization layer is available and functional when it is needed. It should not be prone to crashes or errors, and should always be able to grant or deny access to resources as required.

Key Requirement 3 – Speed

The authorization layer can be a bottleneck in software performance. Every request to access a resource or to process an action must pass through it.

Unlike authentication — performed once per session to verify identity — authorization must evaluate each API request dynamically. Authorization requests cannot be cached in a token because every request can be different based on the actor, resource and the context.

To address this challenge, a fast authorization layer is crucial in ensuring that the software is responsive and performs well for all users. Careful attention needs to be paid to prevent slowdowns and to provide a good user experience.

> *"With the latency tolerances that we have, anything over a few milliseconds can be slow for us, depending on the context. If you imagine 10,000 people trying to buy the exact same instance of a physical product on our platform at the same time, that's what we have to deal with,"* said Steve High, Staff Engineer formerly at NTWRK

In fintech and e-commerce, latency can cost millions of dollars. It can also cause havoc with large auction sites where users may be struggling to get a bid in. So please do not ignore it.

Key Requirement 4 – Scalability

As your software grows and attracts more users, the authorization layer must be able to handle the increased complexity and maintain its performance. As you add new roles and permission rules, it often increases the complexity of both your software and the authorization layer.

> *"It was important for us to be able to add new user profiles one after another when we are building new products. It enabled us to ship our products faster," said* Engin Attar, Head of Product and Growth and co-founder, Debite

Therefore, hard-coded authorization makes your software error-prone, and increasingly difficult to scale, maintain, and test.

A scalable authorization layer is essential in ensuring that the software can continue to function! It also goes hand-in-hand with extensibility, as scalability ensures your system can grow, while extensibility enables it to adapt to changing needs:

> *"Products are live things. You gotta have that flexibility, otherwise, it's very hard to deliver quickly and to innovate,"* Edgar Rivera, CTO, 4G Capital

Key Requirement 5 – Extensibility

Software requirements change over time, along with your user base and their demands. Your authorization layer must be able to adapt to these changes.

> *"When you're building a company, you pivot, you introduce new products or new features. And it's very hard to keep updated policies for access controls. So we wanted an easy solution that we don't have to build again and again,"* said Engin Attar, Head of Product and Growth, and co-founder at Debite

An extensible authorization layer can accommodate changes in user roles and permissions, as well as in the structure of the resources that are being accessed. Extensibility and flexibility should be considered to ensure that the access management system can adapt and evolve with the software without too much hassle.

Key Requirement 6 – Great developer experience

> *"One of the things that developers don't like to do is to code access controls. They are very hard to update and maintain. Additionally, testing access control rules is a nightmare,"* Engin Attar, the Head of Product and Growth, and Co-Founder of Debite

Developers are the ones who have to deal with building, integrating, and maintaining authorization systems in their software. As a result, the experience they go through must be considered as a key requirement for any solution to be adopted and used efficiently.

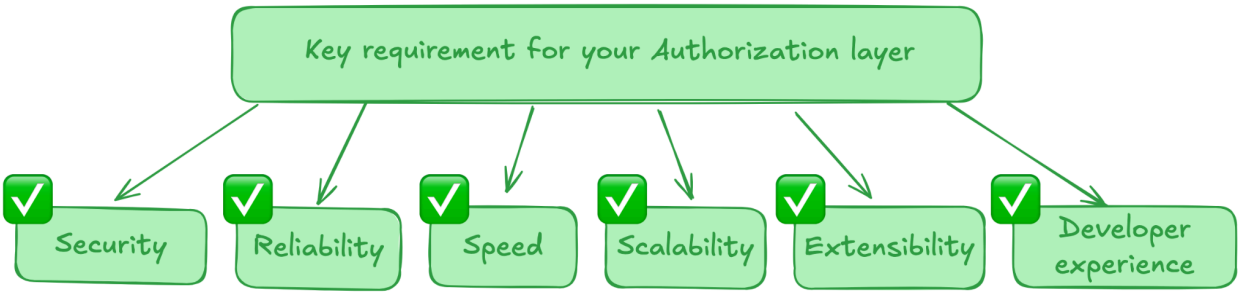
If the authorization layer adds friction to the development process, it increases the chance of developers making mistakes. A great, streamlined development process translates into less error-prone development.

The authorization layer should be easy to understand, integrate, and to use. It should provide simple APIs, clear documentation, and a supportive community to help developers overcome any challenges they may face. Developers want to focus on what they do best: solving the business problem at hand.

Chapter 3

Design considerations to address the requirements

The requirements we just discussed formed the foundation of Cerbos' architectural design. However, meeting each one of the key requirements was not a simple task. It took a lot of planning, problem-solving, and time.



In this chapter, we'll cover each of the key requirements one by one and let you know about the design considerations we made and the technologies we used.

Security

Since security was so essential, we built our authorization layer on a foundation of zero trust architecture. That created a few challenges that we had to address.

Network connection

The first major hurdle in delivering the promise of zero trust architecture was that Cerbos runs on the same virtual private cloud as the application it serves. So, we had to make sure unauthorized traffic couldn't access the authorization layer.

- To achieve this, **Cerbos employs a mutual TLS (Transport Layer Security) approach**. It communicates with the application via a secure channel. This mutual TLS ensures that the authorization layer is only accessible by the application and that any malicious actors or unauthorized requests are blocked.

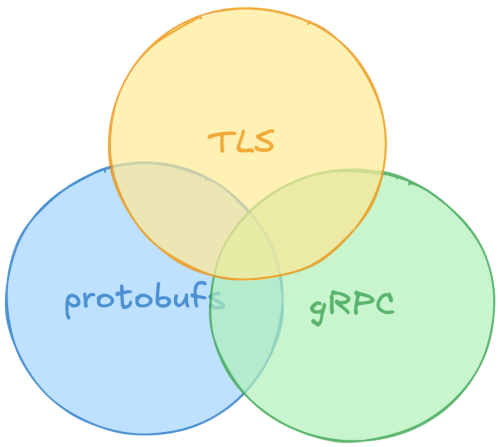
- This approach significantly increases the security of the authorization layer and consequently the entire software.

The second major hurdle was every request had to be evaluated on its own merits due to the contextual nature of every authorization request. As a result, speed became a major factor in maintaining usability with the constant verification and heightened security Cerbos was running.

- So, **we prioritized the speed of the connection between the application and our authorization layer**. We'll go over this in more detail below, under the speed section, but let's have a quick rundown here.
- Cerbos can be deployed as a sidecar alongside the application in the same container, and communication between the two can happen via loopback connections using sockets. This lessens the attack surface during the communication between the application and the Cerbos service.

The use of gRPC (Remote Procedure Calls) provides high-performance communication between the application and Cerbos. Internally, Cerbos uses protobufs to make data transfer very fast and efficient. The strong typing of them additionally gave us backward and forward compatibility in our messaging format, reducing the room for errors as the software evolves.

Thanks to protobufs, gRPC, and TLS connections, Cerbos provides a fast and secure communication layer that can easily integrate with any software.

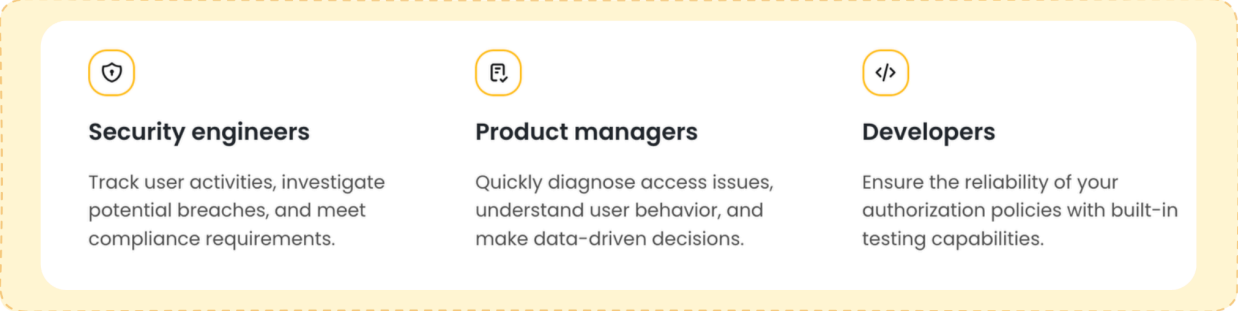


Auditability

Zero Trust requires verifying everything and trusting nothing by default. That's why we made auditability a major priority for Cerbos. To give the users in-depth visibility, **our authorization layer provides a complete audit trail** of all access requests and decisions made by it. The audit trail captures details such as:

- Time of the request
- User making the request
- Resource that was being accessed
- Decision made by the authorization layer and why, i.e. which policy caused the decision

These audit logs are made available in a structured format, making it easy for security engineers, product managers, and developers to understand and analyze the data:

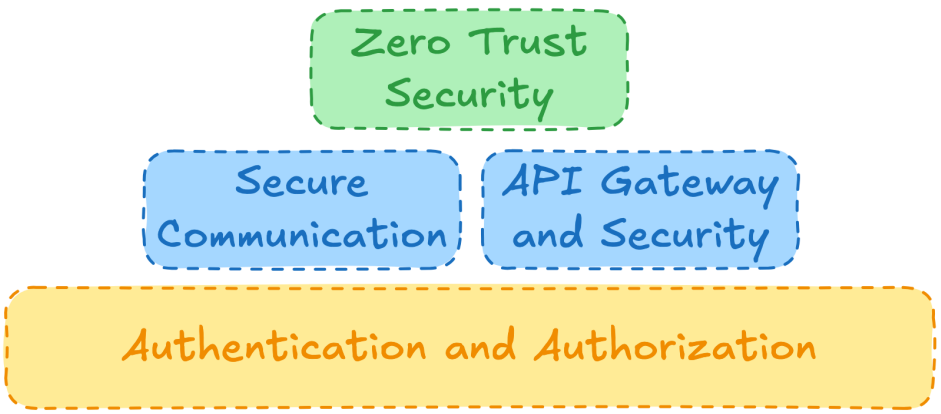


Cerbos audit logs can be viewed in Cerbos Hub and can easily be exported to various SIEM (Security Information and Event Management) tools and logging providers such as Datadog and Elastic. This allows organizations to leverage their existing logging infrastructure to capture Cerbos's audit logs, making it easy to monitor and analyze the authorization layer's activity.

Conforming to a GitOps model provides an additional level of auditability. With GitOps, all policy changes are made through pull requests, which are reviewed and approved before being merged into the main branch. This process provides a complete audit trail of all policy changes, including who made the changes, when they were made, and why they were made.

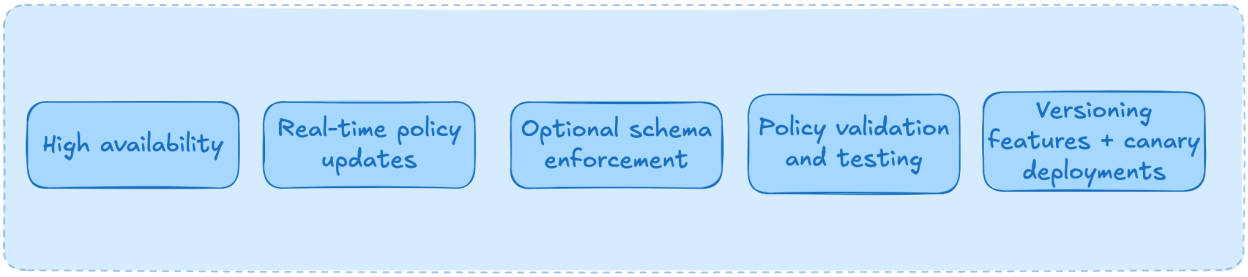
Moreover, GitOps provides **version control,** allowing organizations to maintain a history of all policy changes made to the authorization layer. This history can be used for debugging and to understand why certain decisions were made at a particular point in time.

Such a level of auditability allows companies to comply with regulatory requirements and provides transparency into the authorization layer's decision-making process, fully following zero trust security.



Reliability

To ensure our authorization layer is always functional when needed, we focused on building a highly reliable product. Our approach is supported by five main pillars of reliability, which you should consider if you're building your own authorization layer:



High availability

Since the entire application depends on the authorization decisions, any downtime makes the app unusable. That is why we focused on **maximizing the availability of the authorization layer as much as possible**. To do that, we concentrated on one of the aspects we had full control over, which is policy updates.

Policy updates in an authorization layer could lead to unwanted downtime, which wasn't acceptable. We had to come up with a way that our authorization layer could update policies without requiring downtime. So, we designed Cerbos to update the authorization logic across the entire stack in real-time once a policy change is made, without requiring a hard reboot or deployment of the binary. This ensures that the system remains up and running.

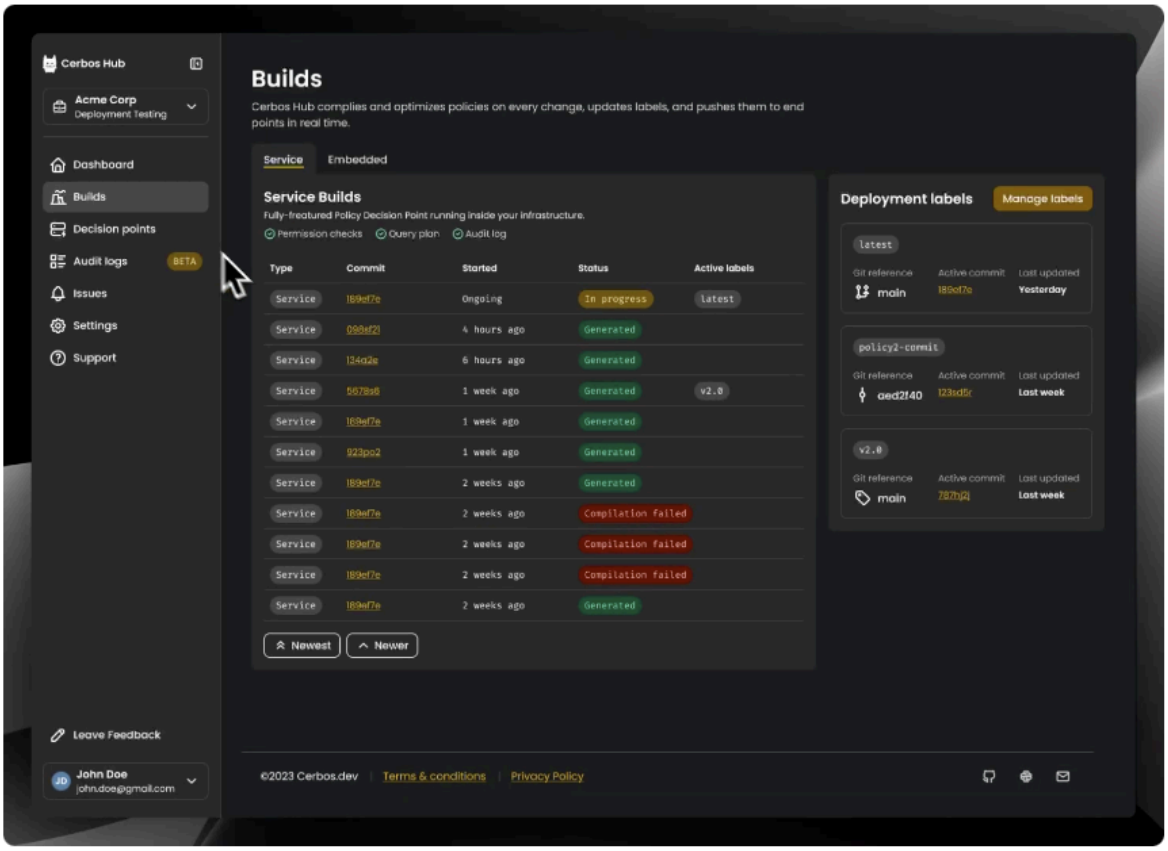
Optional schema enforcement

Authorization requirements can and do change over time. This constant evolution can introduce mistakes into the authorization layer, causing failures. So we made the decision early on to allow for optional schema enforcement. By rejecting data that doesn't fit into the expected schema, we made sure Cerbos does not silently fail just because a new parameter in the input is omitted.

Validation & testing

No matter how strong your authorization layer is, if your policy update is flawed your authorization layer is flawed. Cerbos' validation and testing framework is our answer to this vulnerability. It enables developers to test the policies before deploying them

to production, minimizing the risk of policy conflicts or errors that could cause downtime or other issues.



Canary deployments

Any time updates are rolled out, there is a potential for introducing new vulnerabilities in the system through human error. Rolling out updates to all users at the same time multiples those vulnerabilities across the system.

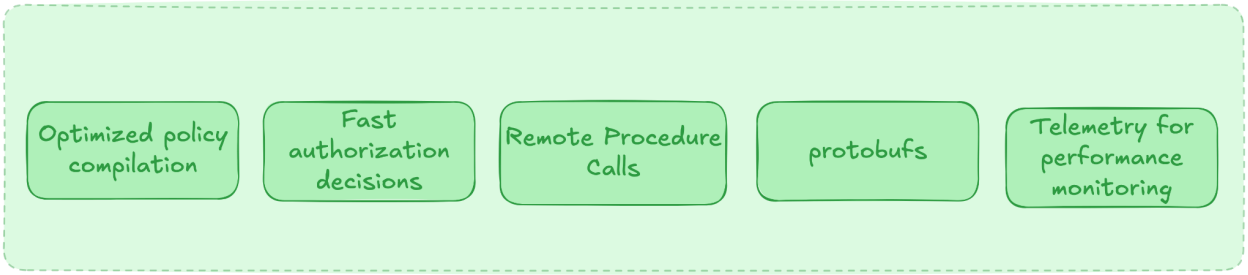
To combat this, we added versioning features that give developers the ability to test and validate policy changes in a small part of the software before rolling them out across the entire system. This minimizes the risk of unintended consequences or negative impacts on the system as a whole.

Speed

After talking to many developers for whom speed was just as critical as security, we've become laser-focused on this requirement for our authorization layer.

> *"If you imagine 10,000 people trying to buy the exact same instance of a physical product on our platform at the same time, that's what we have to deal with. Using Cerbos as a sidecar, we've been able to get permissions-checking latency down to microseconds, it's ridiculously fast,"* said Steve High, Staff Engineer, NTWRK

One of the key restrictions of building speed into our authorization layer was the fact that nothing can be cached, due to the contextual nature of authorization requests. Each authorization request must be evaluated on its own merits and cannot be pre-computed or cached. As a result, Cerbos is designed to make authorization decisions quickly and efficiently. Our authorization layer is **optimized for performance when evaluating authorization decisions,** which comes to such aspects:



Optimized policy compilation for faster decisions

A key consideration was how to compile policies in a smart way and efficiently serve the fastest response based on all the policies. To achieve this, Cerbos compiles policies to optimize its decision-making process. Our linear and predictable decision process coupled with not having a state enables Cerbos to short-circuit decisions if a

certain condition is met, allowing it to skip irrelevant rules and make authorization decisions even faster.

Minimizing impact on system response time

Fast decision-making is only half the battle. As we touched on in the security section above, the speed of network communication, between the software application and the authorization layer, is critical as well.

Every authorization decision is a blocking factor that directly impacts the response time of the system. Therefore, **every authorization decision must be made as quickly as possible** to avoid slow response times and high cloud bills on high-volume systems.

- To increase the speed of communication, we chose to use gRPC (as we mentioned before). This provides high-performance communication between the application and Cerbos.
- Internally, we decided to use protobufs to make data transfer very fast and efficient.
- The strong typing of protobufs additionally gave us backward and forward compatibility in our messaging format, reducing the room for errors as the software evolves.

The speed is crucial because the speed of the authorization API is directly tied to the perceived speed of your system. Slow response times can impact user experience and lead to lower user engagement.

Telemetry for performance monitoring

We knew it was critical to give developers and DevOps teams insight into how the authorization layer is performing and give them ways to diagnose issues and identify latency problems & bottlenecks. Which is why Cerbos fully supports OpenTelemetry for distributed tracing for end-to-end observability and exposes Prometheus metrics. Moreover, these integrations enable DevOps teams to monitor the performance and set up alerts if thresholds are exceeded.

Scalability

At every step during our development process, we worked to keep Cerbos as scalable as possible. A key design decision was building a stateless authorization system.

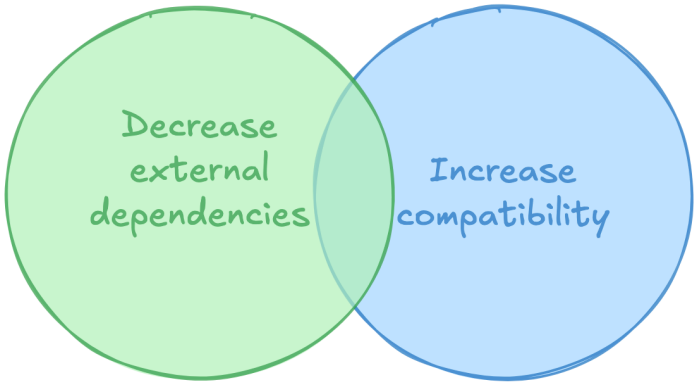
Stateless design

The stateless design was chosen to ensure that Cerbos can infinitely scale with any application. So, as long as the system scales, Cerbos can run alongside it.

In making it stateless, it was crucial for Cerbos not to store any data between requests, making it easy to deploy multiple instances of the authorization layer to handle high volumes of requests. This scalability is crucial when deployed as a sidecar, as it can handle the request load of the main application while ensuring that authorization is performed consistently and efficiently.

This stateless design allowed us to:

- Decrease external dependencies
- Increase compatibility



Decreased external dependencies

Because of our stateless architecture, Cerbos is not dependent on anything external other than consuming the policies at boot time and only updating them at run time upon a change. This simplicity ensures that Cerbos can run with minimal overhead, resulting in fast and efficient authorization decisions without causing any overhead load on the rest of the system.

Increased compatibility

Cerbos is designed to run as a microservice to ensure that it is compatible with different tech stacks and programming languages. Deployed as a microservice, it can be integrated into different software architectures, including microservices, monoliths, and serverless architectures. This compatibility enables teams to adopt it into their existing infrastructure, without having to overhaul their tech stack or programming languages.

Furthermore, the microservice architecture of Cerbos ensures that it can communicate with different components of the software stack. The use of an API with SDKs in all major languages provides an easy-to-use interface for other services to interact with the authorization layer.

Cerbos is also designed to be agnostic to the deployment method used. It can be deployed on bare metal, via Kubernetes services, or even as a cloud function. This flexibility allows users to choose the deployment method that suits their specific use case, and let the deployment service decide when to scale it up and down.

Extensibility

From day one, we wanted to build a product that was as future-proof as possible. To go beyond traditional Role-Based Access Control (RBAC), we chose to support Attribute-Based Access Control (ABAC), enabling contextual decisions based on context and the state of both the principal and the resource being acted upon.

> *"We had a look at Cerbos and we said, "Yeah, this is future-proofing our ABAC engine," if I can put it that way,"* said Je Sum Yip, Chief Engineer, Human Managed.

Contextual roles

We designed Derived Roles to extend the existing traditional roles that user directories assign to users. A "Derived Role" is something a principal can assume based on the context of the authorization action.

Let's take, for example, a role called "OWNER". In a user directory, you'll almost never find a user with the role "OWNER" of a resource. A user is an owner of a resource if they are the creators of it or if they are specifically assigned to be so. When trying to codify this role, you may make assumptions like the ID of a principal matching the owner ID of a resource. When designing the policy structure of Cerbos, we made codifying these situations very simple using any attribute of a principle and resource. Furthermore, these derived roles can then be used as building blocks of more complex policies.

Flexible policy conditions

Authorization decisions need contextual calculations and processing of logic. In order to be able to perform complex calculations on principal and resource attributes, we decided to include Google's Common Expression Language, CEL, in our policy language, rather than building our own logic engine.

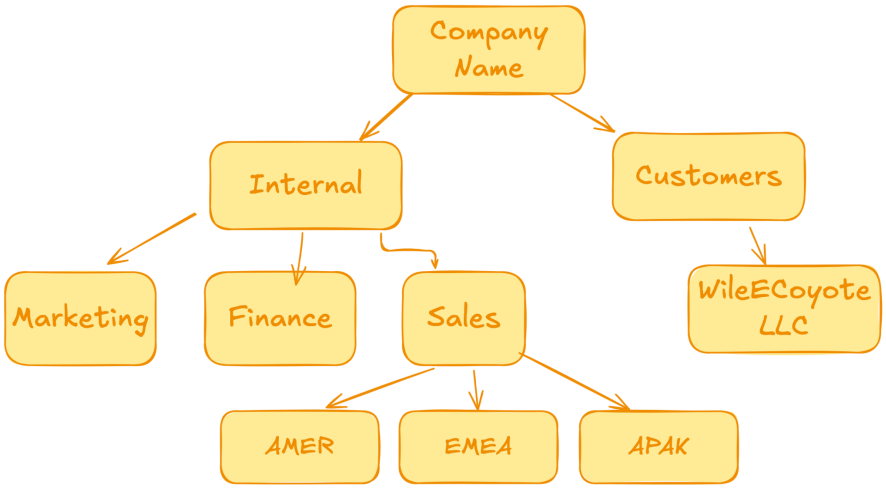
CEL provides robust logic operators to make calculations in order to evaluate conditions within each incoming request. This allows us to have a very comprehensive list of logic operators without having to invent a whole new language of our own.

Policy hierarchies and scopes

Many organizations have hierarchical departments, roles, or geographies. The authorization requirements of these hierarchies tend to follow the internal company

structure. So, it was important for us to give companies the ability to make hierarchical rules along with the ability to enhance and override those rules based on the structure.

In order to address such complexity we built Scoped Policies in Cerbos. This allows overwriting policies for a given group that is beyond the default set.

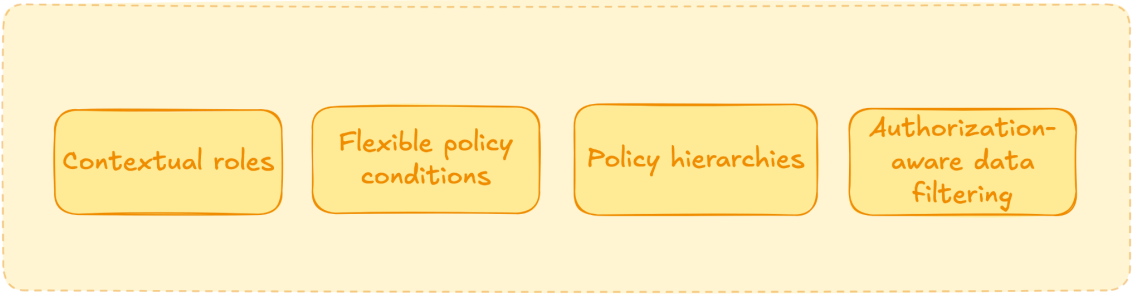


Authorization-aware data filtering

Not every authorization request in an application can be answered with a simple ALLOW or DENY as an answer. Sometimes an application needs to evaluate a huge set of records so it can choose only the ones the user has access to. The naive way of addressing this requirement would be to fetch every record from the database and evaluate whether the principal can perform each action one by one or in a batch operation. However, this is a costly operation, especially with large data sets.

At Cerbos, we built the Query Planner API, to allow the application developers to run authorization-aware queries in their data fetching operation. Query Planner returns an Abstract Syntax Tree of conditions based on the authorization policies one should apply to their data fetching query. For simplicity, we've built adapters for popular ORMs like Prisma, SQLAlchemy and Mongoose.

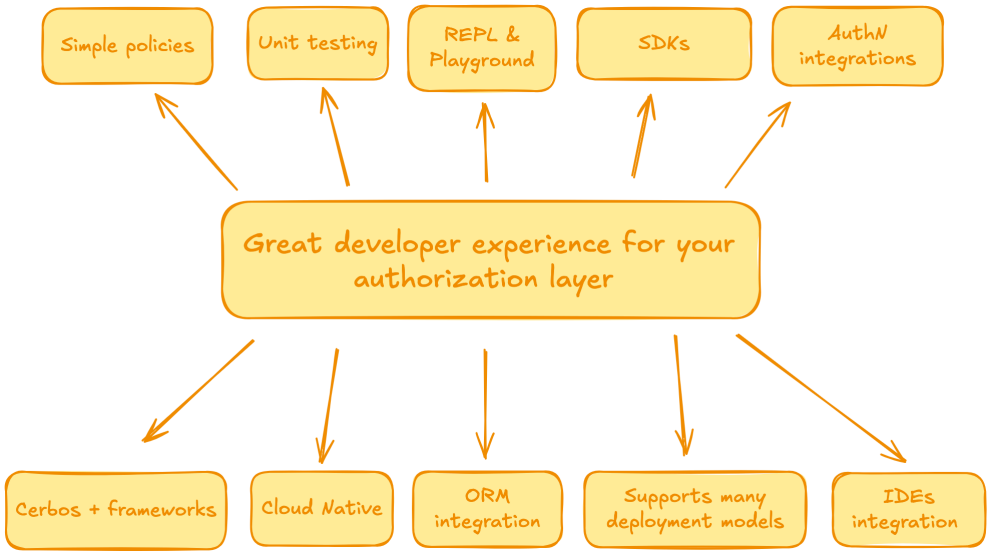
These four design decisions support the extensibility requirement in Cerbos' authorization layer.



Great developer experience

Although listed last here, most of our focus goes into making sure working with Cerbos is a great experience for developers.

We are pioneering the decoupling and externalization of authorization logic. As a result, we realize that we need to focus on making the onboarding and day-to-day usage of it as frictionless as possible to increase the chances of adoption. For our authorization layer, the developer experience consisted of many parts:



Simple policies

Most of the developers we interviewed before building our own authorization layer did not want to spend time learning a whole new programming language to deal with AuthZ. So, we chose policy as a configuration method and YAML as the policy definition standard.

We chose policy as a configuration to take advantage of three main benefits. It is

- deterministic
- stateless
- easily testable

For authorization logic, we chose YAML because we believe it should be easily accessible to people who are not developers. And YAML's simple syntax makes it human-readable. We've found that product managers and security team members appreciate that very much.

> *"We love the fact that policies and rules are all written in YAML because, from the early get-go, we decided that YAML is going to be like our bread and butter for a lot of things that we do. So we love that. And the fact that the YAMLs can be written with the Google CEL language makes it so, so powerful,"* said Je Sum Yip, Chief Engineer, Human Managed

Unit testing framework

Nobody wants to have unexpected corner-case scenarios threaten their product's security. So, we made sure users feel confident about every change they roll out. To give them that confidence, we built a robust testing framework for both predicted cases and full matrix testing.

REPL & Playground

From some perspectives, Cerbos decision point is a black box where policies go in with a request and a response comes out. Although a unit testing framework makes

programmatic detection of how it behaves easy to interact with, the users still need an interactive way to develop policies.

Cerbos REPL provides a simple interactive mode to evaluate how Cerbos would interpret policies and conditions. However, in most cases, REPL only answers basic, simple single calculations. So, we built Cerbos Playground, an in-browser IDE in which users can test and evaluate how policies and requests work in the presence of each other without installing or running anything.

IDEs integration

Having a simple DSL configuration is not enough according to our interviews. So for developers using IDEs, we went an extra step and published specs so their preferred IDEs can automatically highlight issues with policies and provide autocomplete capabilities.

SDKs for all the major languages

Developers use different programming languages to build their apps, so we decided to add SDKs for all popular languages and frameworks. SDKs make it easier for developers to build and integrate their applications with other software. They provide a set of tools and resources, such as APIs, code libraries, and documentation, that developers can use to access and interact with the features and functionality of the software. It can save developers a lot of time and effort, and it can also help to ensure that integrations are done correctly and efficiently.

The choice of protobufs made the production of SDKs in most popular languages very easy for us.

AuthN ecosystem integrations

Authorization is usually the next step after authenticating a user. There are many authentication methods out there, username/password tables to OAuth2 access tokens. To make our authorization layer useful to more customers, we knew we had to make Cerbos able to consume authenticated user information no matter what format it came in.

By making it very open and building lots of examples with a variety of authentication providers, we aimed to show developers how they can integrate Cerbos with their preferred authentication system.

Examples of using Cerbos in major frameworks

Developers often use frameworks to improve their productivity, rather than starting from scratch. This makes it simpler to build complex applications. But, no one can agree on the best framework. Because of the variation among frameworks, how Cerbos will be implemented in an application will always be different among different code bases.

So by showing how Cerbos can integrate with your favourite framework, we want to demonstrate where in your application it should be used and how we suggest it be used.

Cloud Native tooling and integrations

Cerbos was born in the Cloud Native era. Making the software building in cloud-native environments a pleasant developer experience was also a key requirement. Therefore, we wanted to make sure that all the cloud native requirements were first-class citizens from the first line of code. So, from the get-go, Cerbos supports OpenTelemetry, and Jaeger tracing. Cerbos is a system that many applications will rely on during every API call. Therefore, making each request metric measurable and traceable was a key experience to deliver.

ORM integration for data-aware authorization

Some applications use object-relational mapping (ORM) frameworks to interact with their data storage layers. To ensure that data fetching can be authorization aware and only fetch relevant data, integration with ORM frameworks such as Prisma and SQLAlchemy were considered as great simplifications that make a lot of the authorization translation to data querying restrictions transparent to developers.

Supports many deployment models

Today's application deployment landscape has a variety of options. Based on their requirements, the developers can choose from a variety of infrastructures and methods to run their applications. These can vary from cloud functions to running applications on bare metal hardware.

Making our application indifferent to how it is deployed and run was a key requirement for us. Cerbos is a binary that can support pretty much all the current architectures. Because of its stateless nature and the fact that it does not depend on anything in the environment, it can be wrapped and run anywhere.

Should you consider all these architectural and product decisions to make your authorization layer developer-friendly? Our experience has shown that every detail matters—it's taken us three years of building and continuous improvement to get our solution where it is today.

Parting thoughts

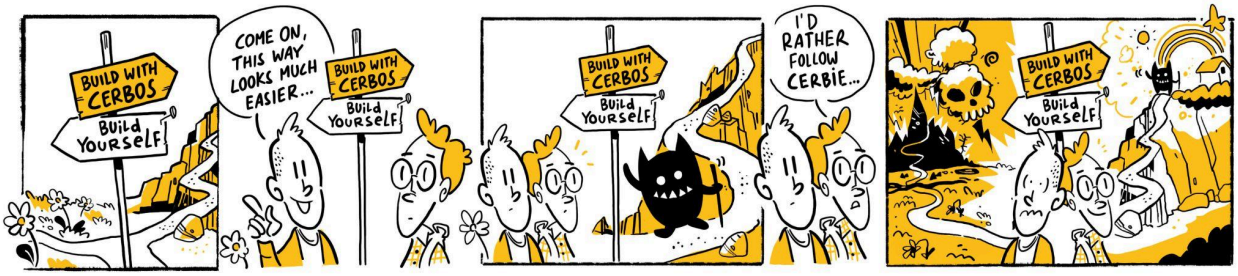
> *"We implemented an in-house authorization system at my previous company. It kind of did the job but was difficult to update. It couldn't support all the use cases, and we never had time to develop it after release. This caused a lot of confusion and we often worried it wasn't working correctly,"* said David Workman, Senior Engineer at Salesroom

Building a future-proof, scalable, extensible, and performant authorization layer is complex and time-intensive. So when the authorization layer is added into development as an afterthought, software engineers triage their hours, and adopt a "minimum viable solution" approach. This gets your software up and running, but creates technical debt that's usually ignored until it becomes critical. By that time, resolving the issue is a monumental task that diverts a significant amount of time and attention from the engineering team.

To avoid technical debt, you need to have a team that can not only build your software but also take the time and effort necessary to build a flexible authorization layer. That's a lot of developer hours, and it adds up quickly.

> *"When I did an audit, I found that the cost of managing authorization and authentication in-house over the entire lifespan of the company was in the seven figures,"* said Chuck Hardy, Head of Engineering, Salesroom

At Cerbos, we want to make sure developers spend as little time as possible building software infrastructure so they can focus on their domain problems. So, we built an open source authorization layer that addresses the requirements of modern software applications right out of the box. Now, developers can stop spending hours building their own authorization layer, or managing the technical debt of a baked-in solution.



Our externalized authorization solution allows developers to avoid the technical debt of authorization MVPs, time spent on building and maintaining the authorization layer, and all the developer costs associated with it. We built our product for developers, adding everything to make the experience of working with Cerbos great.

Cerbos' ecosystem includes SDKs in every major programming language and integration with popular frameworks and authentication providers. It is secure, reliable, and scalable.

> *"Deploying Cerbos has allowed our engineers to spend their time on what really matters for Salesroom's success. And I sleep better at night,"* said Chuck Hardy, Head of Engineering, Salesroom

Cerbos PDP is available under an Apache2 license, ensuring that developers can use and modify it without any restrictions. Cerbos PDP can be found in Github and its documentation can be found at docs.cerbos.dev.

Whichever approach you choose - building your own authorization layer, or using an off-the-shelf solution, we wish you the best of luck and hope you found this ebook helpful. If you have any questions, our engineers will be happy to give you guidance and chat with you.

cerbos

Authorization management solution for authoring, testing, and deploying access control policies. Implement scalable and secure fine-grained authorization.

cerbos.dev