ZTAuth*: A Paradigm Shift in AuthN, AuthZ, and Trusted Delegations

Introduction

nicola-gallo
Published in ztauth
10 min read · Nov 24, 2024

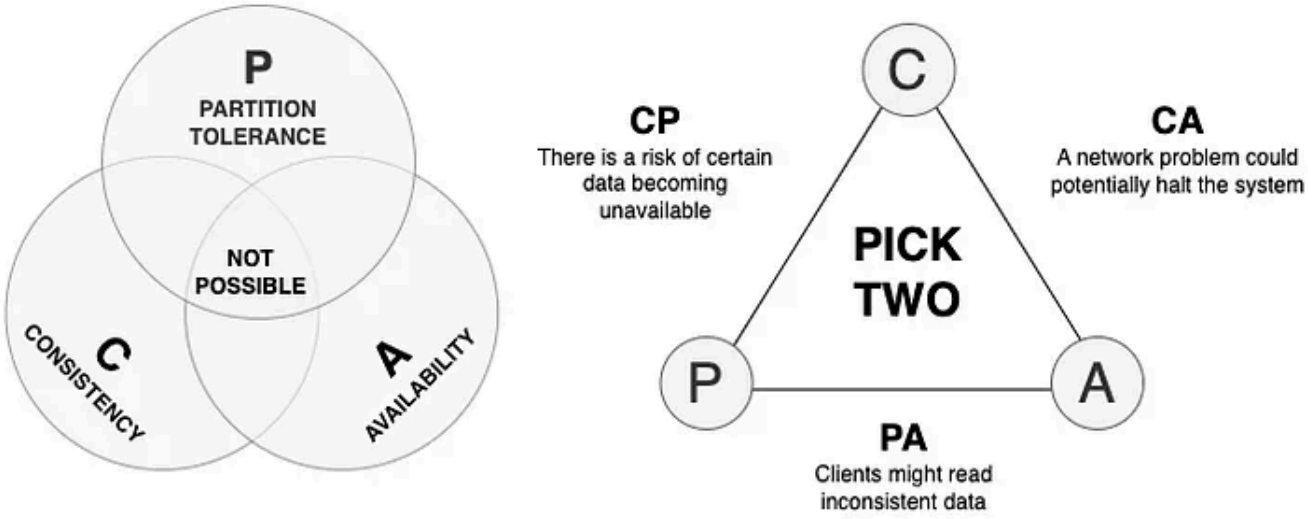
The purpose of this publication is to explore how **IAM** can be reimagined through the lens of **Zero Trust** principles. In a Zero Trust (**ZT**) approach, rather than assuming everything inside the corporate firewall is secure, the model operates on the premise that a breach is always possible. As a result, every request is treated as if it originates from an untrusted network, undergoing strict authentication, authorization, and encryption before access is granted.
Here, I aim to outline the principles necessary for the application layer to adapt to this evolving model. While Zero Trust principles can be applied across devices, networks, workloads, and people, my focus will primarily be on the application layer — a critical area that, in my view, often receives less attention than it deserves.

The purpose of this story is to provide a high-level introduction to the concept. I aim to discuss it through a series of chapters, each exploring different aspects of what I have chosen to name **ZTAuth***. I believe this term effectively captures the essence of

the ideas and principles that I will describe and discuss throughout the subsequent chapters.

> *You might be wondering what **ZTAuth*** stands for. Here's the answer: **Zero Trust Auth***. "Zero Trust" highlights the adherence to Zero Trust principles, while "Auth*" represents a comprehensive approach to authentication (AuthN) and authorization (AuthZ). Beyond this, it also includes related concepts like trusted delegation.*
>
> *The correct pronunciation is **Zee-Tee-Auth-Star**, with the "star" acting as a wildcard to signify not only AuthN and AuthZ but also other interconnected topics, such as trusted delegation.*
>
> *For simplicity, you can also use **ZTAStar** (**Zee-Tee-Ei-Star**) as a shortened version, maintaining the same principles and flexibility. This shorter form could eventually become the official name to make it even more accessible.*

Let's move on from the introduction and dive deeper into the subject. This idea arises from observing companies, especially enterprises, working to implement solutions around APIs in an increasingly connected world. However, "connected" often means "over the wire," and "over the wire" introduces the potential for failures. This is where the **CAP Theorem** comes into play.



CAP Theorem

The CAP Theorem states that in the event of a partition, a system must make a trade-off between **Consistency** and **Availability**. **Eventual Consistency** provides a practical solution by enabling a system to remain available during network partitions, delaying full consistency until the partition is resolved. This approach ensures the system upholds **Partition Tolerance** and **Availability**, even if achieving **Consistency** requires additional time.

> *Let's keep in mind the concept of Partitioning. In our specific context, the challenge lies in* ***Network Partitioning*** *— the same network we aim to make compliant with Zero Trust principles.*

To address **API-driven** challenges, the industry and community have been developing standards, frameworks and protocols such as **OAuth** and **OpenID Connect,** as well as introducing components like *Authorization Server*, Resource Server, *Identity Provider (IdP), User Management Service, Consent Management Service, Policy Decision Point (PDP), Policy Administration Point (PAP),* and others.

They represent a significant step forward and provide a solid foundation for both the industry and the community. However, when I began exploring this challenge, two questions came to mind:

- **Network Partitioning:** According to the CAP theorem, servers and devices can become disconnected. In an API-driven context, this is often treated as a failure, with efforts focused on managing the issue gracefully under the assumption that "the network is down." But what if, instead of viewing disconnection as a failure, we consider disconnected devices that occasionally regain connectivity as a normal state rather than a problem?
- **Zero Trust:** What is the best approach to introducing and implementing Zero Trust principles in this context?

Here, the journey toward addressing **Autonomous-Disconnected-Driven challenges** begins to take shape. If I have disconnected devices that occasionally regain connectivity and need to serve requests for locally connected users — perhaps physically connected to my server with their devices — how can this situation be managed?

For instance, if there is no access to critical components such as the **Identity Provider (IdP)**, the **Policy Decision Point (PDP)**, or the **Policy Administration Point (PAP)**, handling this scenario becomes crucial. Without any connection to these components, the challenge is how to ensure secure and policy-compliant operations in such an isolated environment.

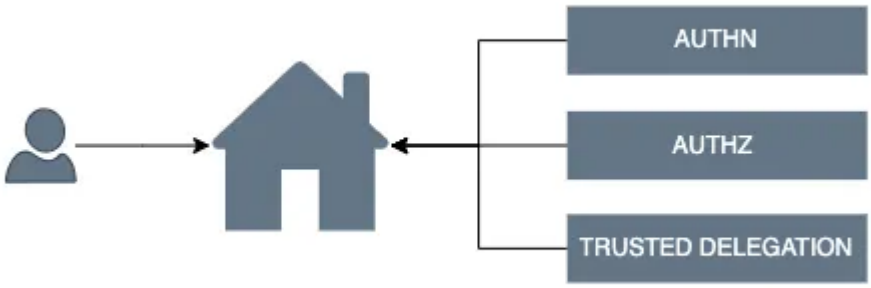
This brings up key questions:

- How do we verify identities and enforce policies locally when centralized services are unreachable?
- What mechanisms can we employ to synchronize policies, auditing logs, or identity information once connectivity is restored?
- How can a local fallback system be established to temporarily handle identity and authorization requests while still adhering to Zero Trust principles?

Let's consider a practical example to make this more tangible. Imagine I'm working in the context of a smart home. I need to unlock the front door, but it should only unlock if it can verify the identity of the person attempting access and confirm that this identity has the appropriate permissions to do so. This involves both recognizing the identity and ensuring the action is authorized.

Of course, the door is an autonomous component that must remain operational even in the absence of connectivity. I certainly wouldn't want to spend the night outside just because my internet provider had an outage! :)

In this case, having disconnected components that occasionally connect introduces intriguing possibilities. For example, it opens up new ways to manage the door, such as implementing policies that allow it to unlock only under specific conditions. Even more interestingly, what if I wanted to delegate access to someone I trust, enabling them to unlock the door on my behalf for a defined period of time — essentially, implementing **Trusted Delegation**?



SAMPLE 1 — Smart Home

This example led me to reflect on the **Autonomous-Disconnected-Driven challenge**. Since it must operate in a disconnected scenario, it became clear that I need local information for the most critical aspects: **AuthN**, **AuthZ**, and **Trusted Delegation**. This information must be managed and represented using a model that is fully compliant with Zero Trust principles.

This introduces the concept of ***Auth Models****, where "**Auth***" encompasses Authentication and Authorization. In a Zero Trust context, these models should possess the following properties:

- **Transferable**: Capable of seamlessly transitioning between systems and environments.
- **Versionable and Immutable**: Ensuring data integrity, auditability, and backward compatibility.
- **Resilient to Disconnection**: Designed to support eventual consistency in disconnected environments.

You might still be thinking, "Fine, cool, but enterprises working in industries like banking, insurance, healthcare, etc., face different challenges, so this is a different story." At first glance, it might seem that way, but I'd argue otherwise. Enterprises are increasingly adopting cloud-native architectures, which are essentially distributed systems connected over the wire. And guess what? Even though developers might not always be aware of the **CAP theorem**, it remains the foundational concept that governs these systems.

It's no coincidence that **NoSQL** has been widely discussed in this context. NoSQL databases leverage **eventual consistency** at the data layer, and the same principle

should apply to the application layer as well.

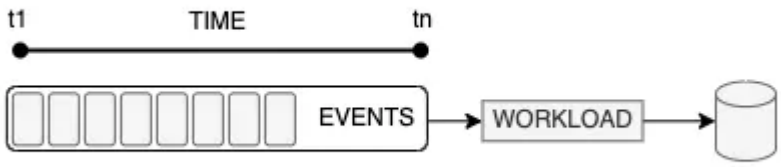
Let's consider an example: a **microservice architecture** implemented with **Kubernetes**. This is a cloud-native environment and, in other words, a distributed system. In such a setup, requests enter the system, often as **HTTP** or **gRPC** requests. These requests may be processed instantly or initiate asynchronous workflows. A common approach is to use tools like **Apache Kafka** to implement stream processing.

This allows for the creation of systems designed to handle **eventual consistency**, enabling resilience and scalability in distributed environments.

Let's focus on this last example. We need to build an **eventual consistency platform** that relies on stream processing with **Apache Kafka**.

Here's how it works:

1. **Request Handling:** A request is received by the system, often via HTTP or gRPC.
2. **Message Transformation:** The request is transformed into a message that encapsulates the necessary data and metadata.
3. **Message Publishing:** This message is published to an **Apache Kafka topic**, ensuring it is queued and ready for processing.
4. **Workload Processing:** At some point, the message will be consumed by a workload responsible for processing it and executing the required business logic, such as saving the data to a database.



Apache Kafka, Workload and Database

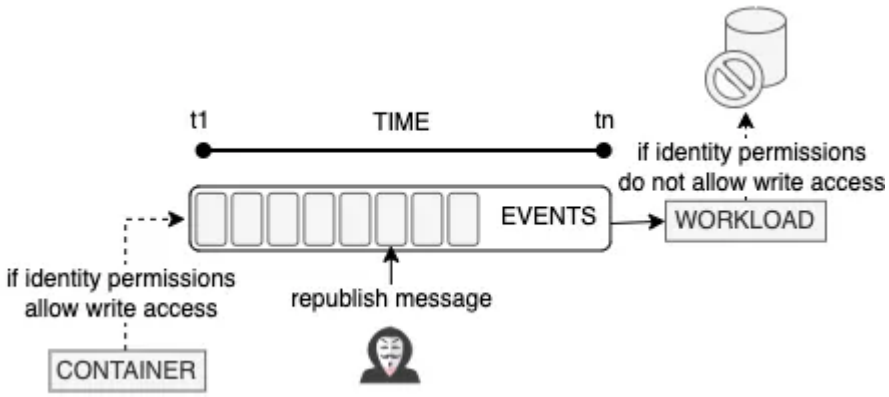
Let's suppose these three components—**Apache Kafka**, **Workload**, and **Database**—are connected to each other, but the rest of the network is disconnected due to an outage. While we have computation resources available and can process the request

and save it to the database, achieving eventual consistency with external systems, we are unable to connect to resources such as the **Identity Provider (IdP)**, the **Policy Decision Point (PDP)**, or the **Policy Administration Point (PAP)**.

> *It should be clear that this scenario mirrors the smart home example mentioned earlier in Sample 1. At the moment the workload picks up the message from Apache Kafka, with connectivity limited to the database, we are confronted with an* ***Autonomous-Disconnected-Driven challenge****.*

Of course, we want to ensure compliance with **Zero Trust principles**, which brings several challenges that need to be addressed:

- **Message Integrity and Authenticity**: How can we ensure that the message has not been tampered with or replayed by an attacker?
- **Authorization Validity**: How can we verify that the identity still has the rights to execute this operation? For instance, the user might have lost those rights in the meantime.



> *These challenges highlight the importance of secure, resilient mechanisms for identity verification and policy enforcement, even in disconnected or partially connected scenarios.*

The challenge can be extended further with new use cases:

- Does the workload have the appropriate rights to handle that message?
- How can the system detect if an unauthorized server or device has executed a certain action?
- How can the action be audited to ensure it was executed by the workload but on behalf of the original identity?

This leads to the concept of Trusted Delegation, which must also be auditable to maintain transparency and accountability.

We have just outlined a few use cases that highlight the importance of designing systems with the assumption that disconnected states are not a failure but a normal condition. Embracing this mindset allows us to address security challenges more effectively and build solutions that align closely with Zero Trust principles, ultimately resulting in more robust and secure systems.

Here's one last example: What if the workload reading the message from Apache Kafka needs to call an external API? Given that there is no JWT token available (as there's no guarantee of when the message will be processed), and passing the token through Apache Kafka would be a very bad practice, which identity should the workload use to invoke the external API on behalf of the original identity?

> *Of course, I am not suggesting that current solutions based on frameworks and protocols such as OAuth and OpenID Connect are inadequate. Instead, I am highlighting the need to approach the problem from a new perspective and extend these frameworks and protocols to tackle the emerging challenges posed by disconnected and autonomous systems.*

That is why I am introducing the **ZTAuth* models**. Addressing the **Autonomous-Disconnected-Driven challenge** requires models that are **eventually consistent** with the security framework, synchronized locally, and compliant with both **Zero Trust principles** and **Corporate Requirements**.

These **Auth* models** introduce new practices based on shared and interoperable schemas. These schemas enable the precise description of resources and actions within a **domain-driven design framework**, allowing for the abstraction of entities and their seamless integration with **ontologies**, resulting in a cohesive and extensible structure.

In the upcoming chapters, we will explore several critical aspects, including:

- How to build all of this while adhering to **Zero Trust principles**.
- How to partition systems and organizations into **accounts** that can be federated.
- How to manage and secure multiple, distinct **environments**.

- How to design and handle **multi-tenant systems** and **environments**.
- How an **Auth* model** should be represented, synchronized, and maintained as **eventually consistent**.
- How to implement **Trusted Delegation** effectively.
- How to adopt and utilize **Policy as Code**, including the best approaches and languages.
- How to evolve organizations by classifying **documents** and **processes** and ensuring they are accessible only to the appropriate identities, whether human or system.
- How to integrate these frameworks with **distributed systems**, including **blockchain** technologies.

I believe these use cases would be highly beneficial across various industries, including banking, healthcare, government, and more.

Here begins the journey.

Next Chapter: Resources, Actions andAccounts in the context of Autonomous and Disconnected Challenges

Zero Trust | Identity | Authentication | Authorization | Policy As Code

ZTAuth*

Following

Published in ztauth

4 Followers · Last published 6 days ago

ZTAuth*: Redefining AuthN, AuthZ, and Trusted Delegation with Transferable, Immutable, and Resilient Models for a Zero Trust World

Edit profile

Written by nicola-gallo

9 Followers · 4 Following

A tech entrepreneur specializing in cloud and enterprise strategic technology, with a focus on distributed systems.

Responses (1)

What are your thoughts?

Respond

pancrazio auteri
12 days ago

This article really connects with the direction things are heading with **AI agents** taking autonomous actions for users or organizations. A couple of thoughts.

The dynamic trust model you describe is spot on for AI agents. These systems are constantly.....

Read More

6 1 reply Reply