HITACHI
Inspire the Next

KUBECON + CLOUDNATIVECON NA 2023

Challenge to Implementing “Scalable” Authorization with Keycloak

11/07/2023

Yoshiyuki Tabata

OSS Solution Center
Hitachi, Ltd.

About the speaker

HITACHI
Inspire the Next

Yoshiyuki Tabata

- Senior OSS Consultant
- Hitachi, Ltd.
- GitHub: @y-tabata
- **CNCF Ambassador**

- Specialist in API authorization
  - Consulting for API management infrastructure and authentication/authorization systems in the financial, public, social, and industrial fields
- Contributor to OSS related to authentication, authorization, and API management
  - Keycloak (IAM OSS)
  - 3scale (API management OSS)
- Other activities
  - Speaker at events such as Apidays, API Specifications Conference, OAuth Security Workshop, etc.
  - Author of Keycloak books (Japanese) and writer of web articles about IAM (Japanese)

HITACHI
Inspire the Next

Contents

HITACHI
Inspire the Next

Contents

What is Authorization

Authorization is the process of verifying that a requested action or service is approved for a specific entity. (according to NIST Glossary)

- **Authorization** is different from **Authentication**.
  - **Authentication** is the process of verifying an entity's identity.
  - **Authenticated** does not mean **Authorized** to access all resources.
  - **Authentication** is not always required for accessing resources, there are public resources that can be accessed without **Authentication**.

Authorization in OWASP Top 10 API Security Risks

HITACHI
Inspire the Next

Three of the top 5 security risks include the word "**Authorization**".

* OWASP Top 10 API Security Risks - 2023 https://owasp.org/API-Security/editions/2023/en/0x11-t10/

#1 Broken Object Level Authorization

#3 Broken Object Property Level Authorization

#5 Broken Function Level Authorization

#2 Broken Authentication

#4 Unrestricted Resource Consumption

#6 Unrestricted Access to Sensitive Business Flows

#7 Server Side Request Forgery

#8 Security Misconfiguration

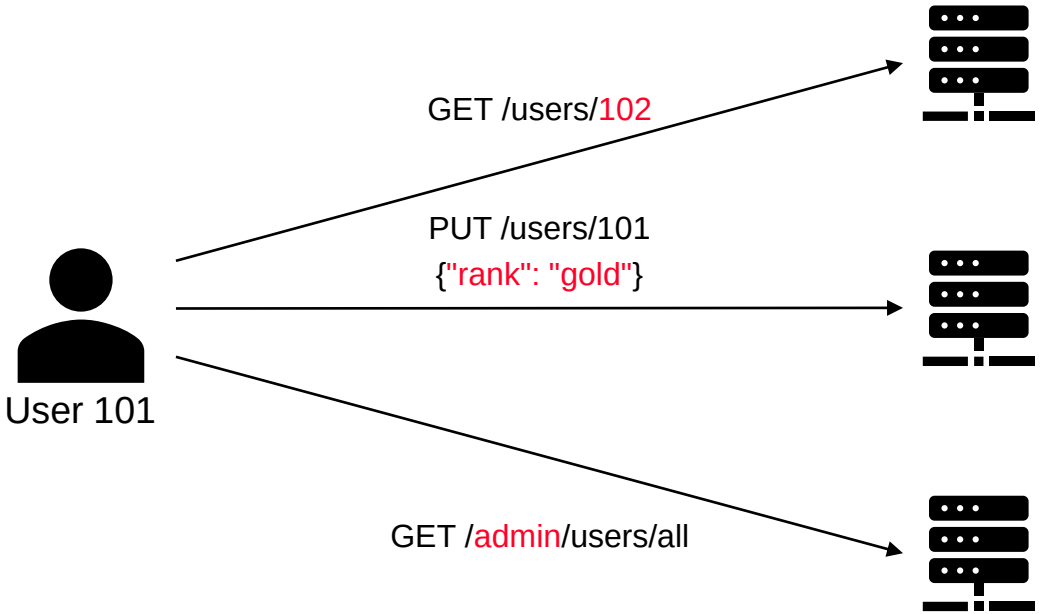
#9 Improper Inventory Management

#10 Unsafe Consumption of APIs

Authorization in OWASP Top 10 API Security Risks

HITACHI
Inspire the Next

There are various levels of **Authorization**.



#1 Broken Object Level Authorization

Must not allow user 101 to obtain user 102's resources.

#3 Broken Object Property Level Authorization

Must not allow a general user to change sensitive object properties like "rank".

#5 Broken Function Level Authorization

Must not allow a general user to call administrator function.

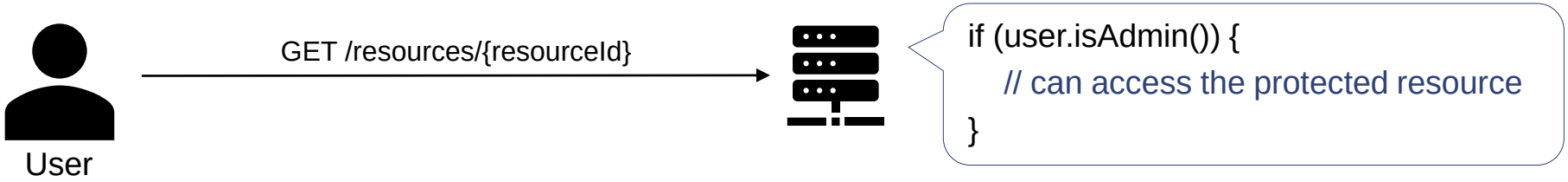
HITACHI
Inspire the Next

Contents

The Simplest Authorization Implementation

HITACHI
Inspire the Next

The simplest authorization implementation is implementing it in application logic.



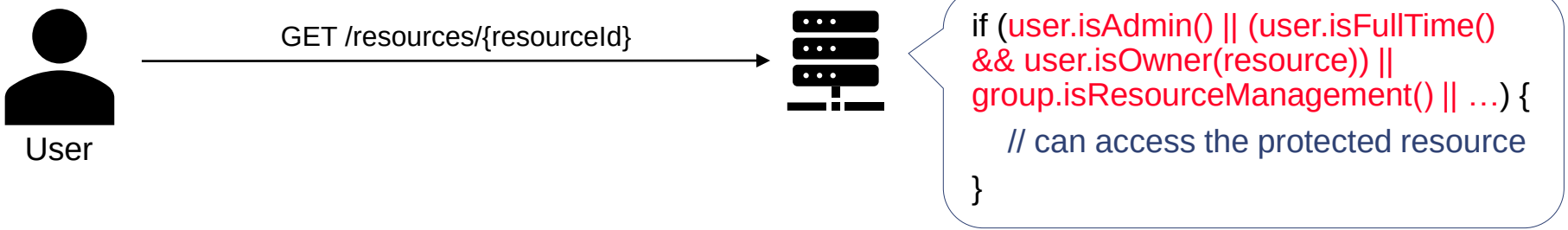
The Simplest Authorization Implementation

HITACHI
Inspire the Next

The simplest authorization implementation is implementing it in application logic.

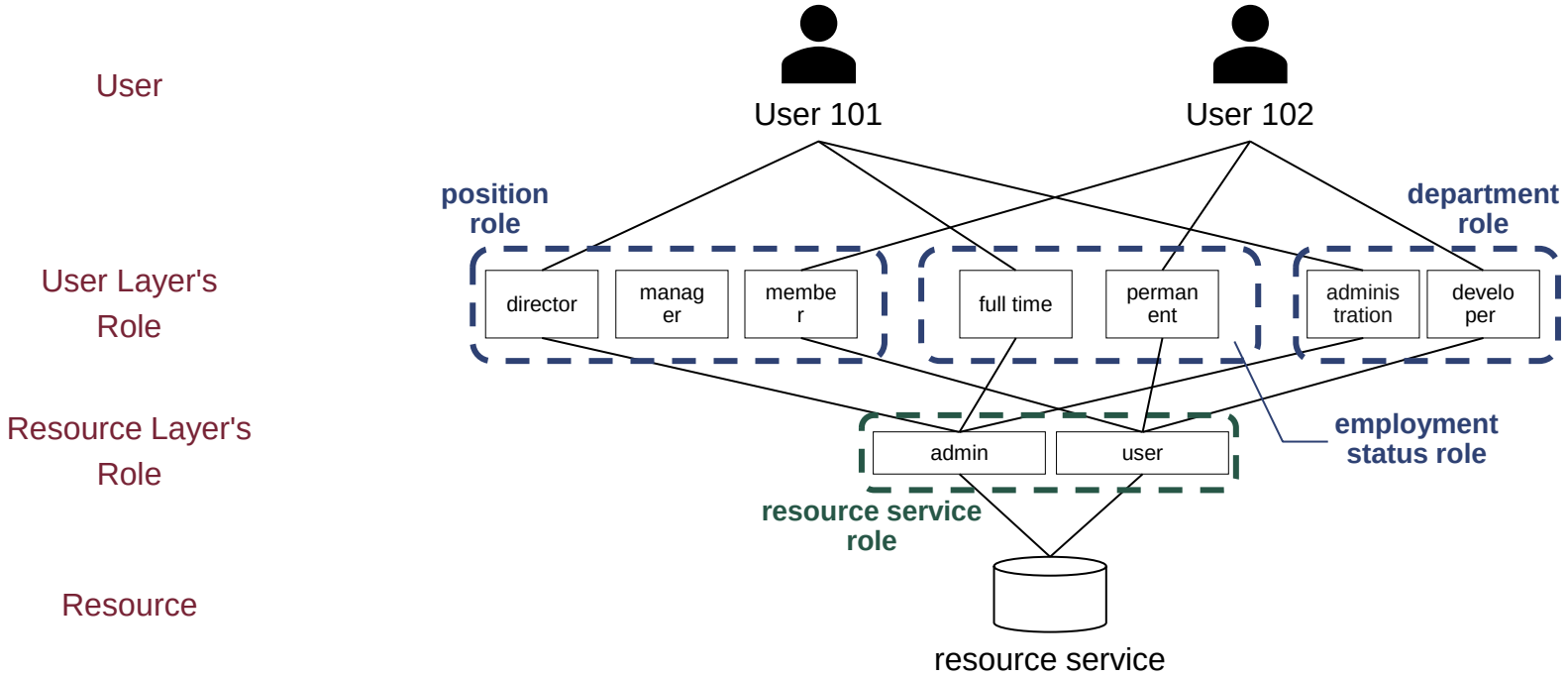
-> But, as the service grows, the authorization logic quickly gets difficult. Duplicate implementations are required in multiple places in the application logic for multiple services.

-> Low scalability...



Role Hierarchy Management

There is an approach to ensure scalability by managing roles in a hierarchical structure.



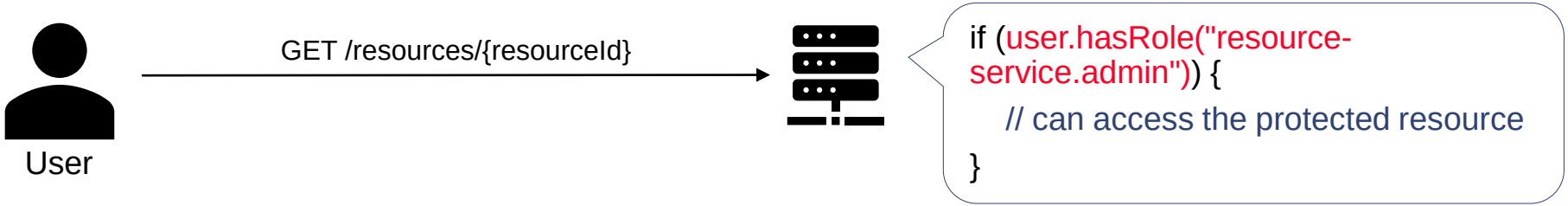
Role Hierarchy Management

HITACHI
Inspire the Next

There is an approach to ensure scalability by managing roles in a hierarchical structure.

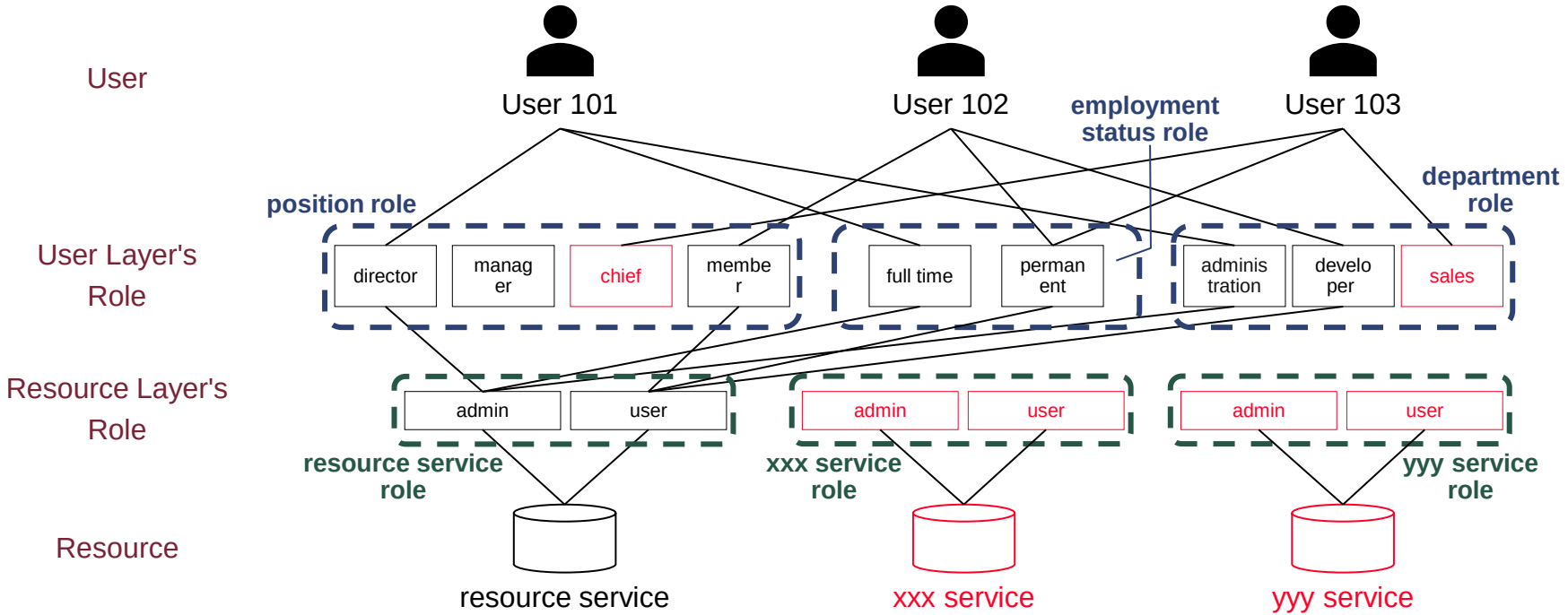
-> Even if the authorization conditions change later, the change will be absorbed by the role hierarchy, reducing the impact on application logic.

-> High scalability?



Role Hierarchy Management

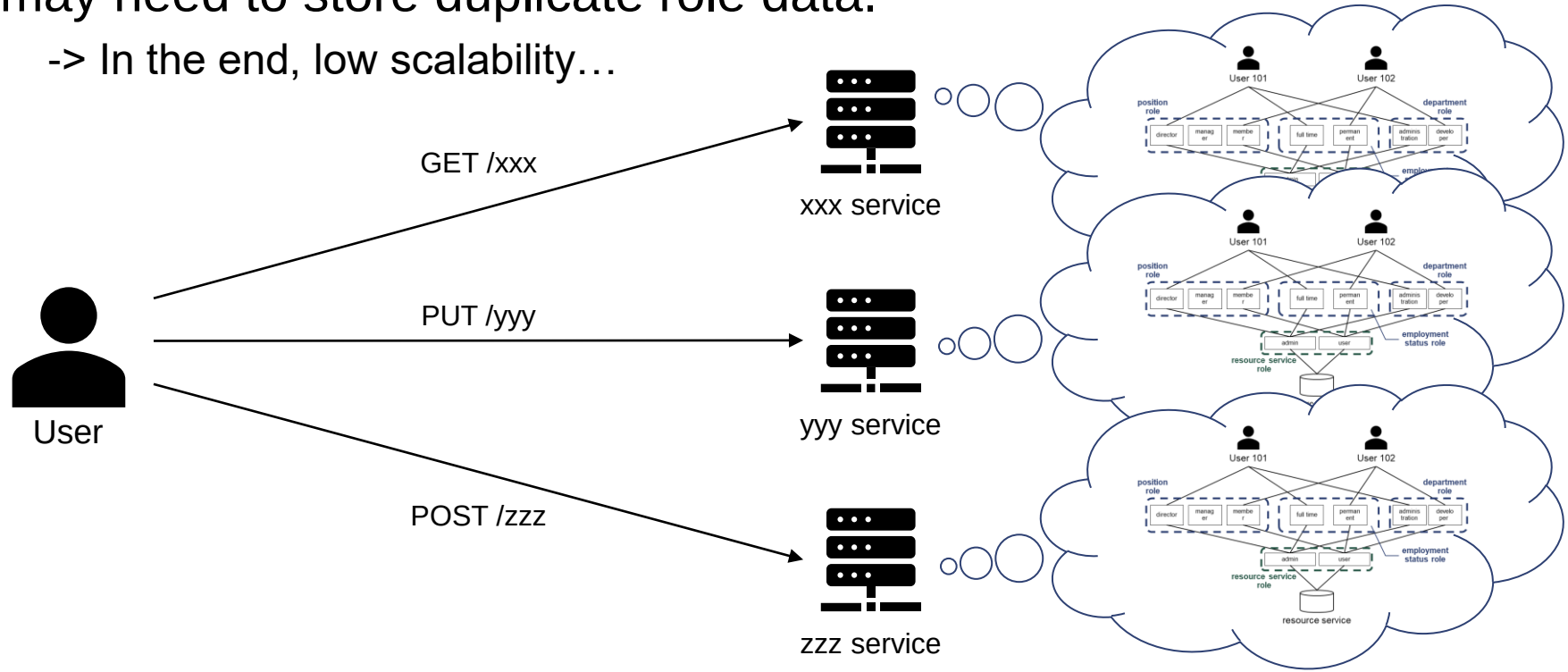
As the service grows, the number of roles increases and there is a risk of "**Role Explosion**".



Role Hierarchy Management

Furthermore, to authorize users by these roles, multiple services may need to store duplicate role data.

-> In the end, low scalability…

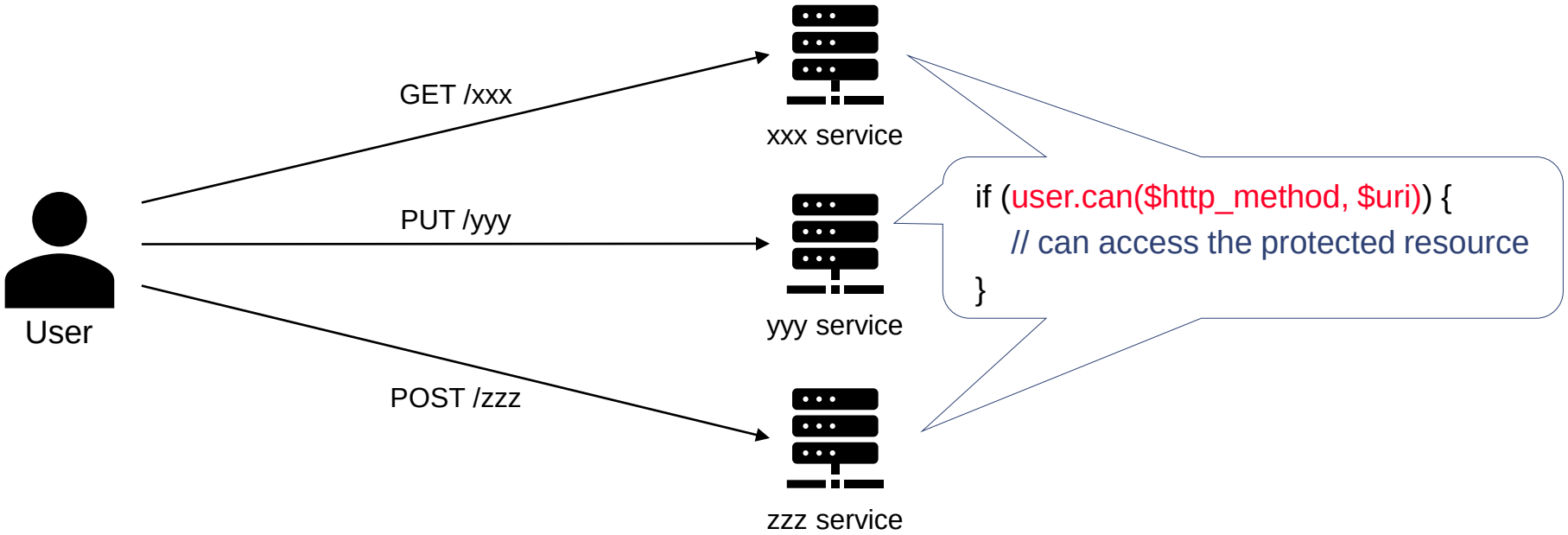


Ideal Scalable Authorization

HITACHI
Inspire the Next

To use the same application logic for multiple services and to eliminate duplicate data as much as possible.

-> Separate authorization from app logic and Centralize authorization data.



HITACHI
Inspire the Next

Contents

How to Separate Authorization from Application Logic

HITACHI
Inspire the Next

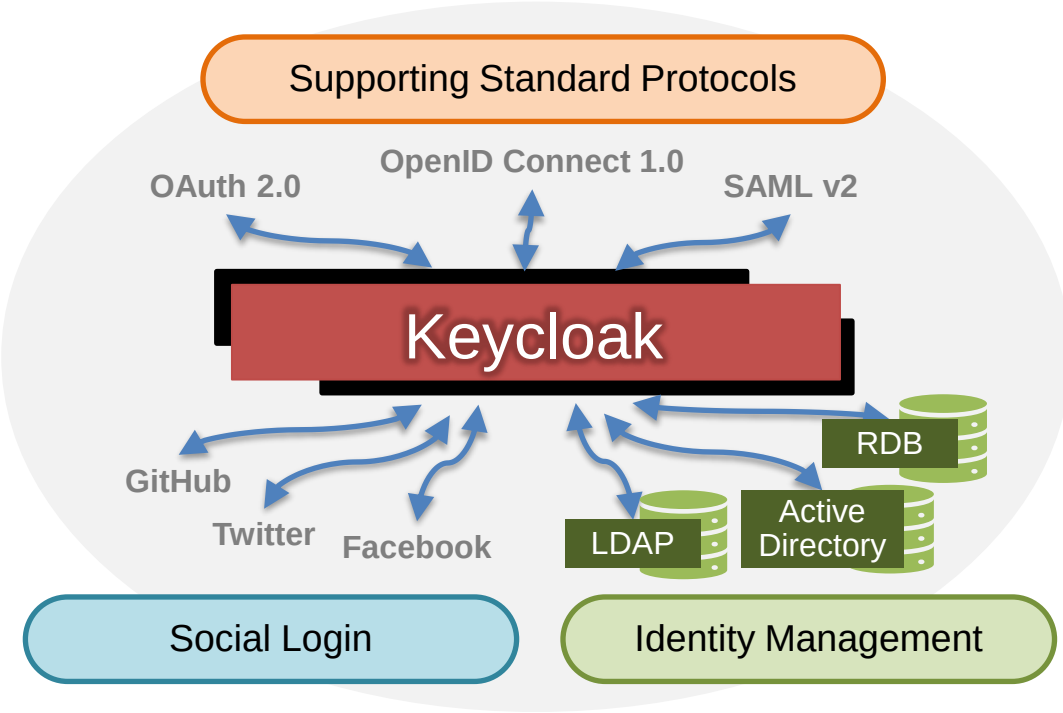
To separate authorization from application logic, implementing authorization logic from scratch or using an external authorization service is necessary.

- From scratch: If **simple-purpose**, from scratch, is better. However, it is difficult to achieve fine-grained authorization.
- External service: Many services allow to definition of **general-purpose** policies to achieve fine-grained authorization. However, the more general-purpose policy definitions are made possible, the higher the learning costs for defining policies tend to be.

- Here, I will introduce Keycloak's authorization service as a **just-right-purpose** authorization service.
  - Based on ABAC architecture, act as PDP.
  - Configure using GUI (i.e. Low-code/No-code).

What is Keycloak

- Keycloak is IAM (Identity and Access Management) OSS.
- Keycloak provides OAuth 2.0 authorization server feature and single sign-on.
- Keycloak became a CNCF incubating project this April.



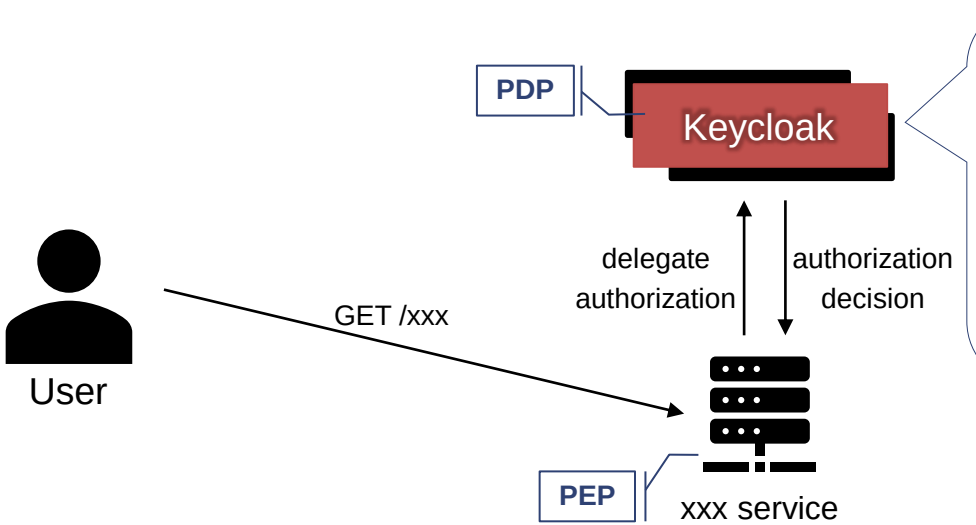
Major features

- Supporting standards. ex. OAuth 2.0, OpenID Connect 1.0, SAML v2, …
- Connect to existing user stores. ex. LDAP, Active Directory, …
- Login with social networks.
- **Policy-based authorization service.**

Keycloak's Authorization Service

HITACHI
Inspire the Next

Keycloak enables fine-grained authorization by **Resource, Scope, Policy, and Permission** management.



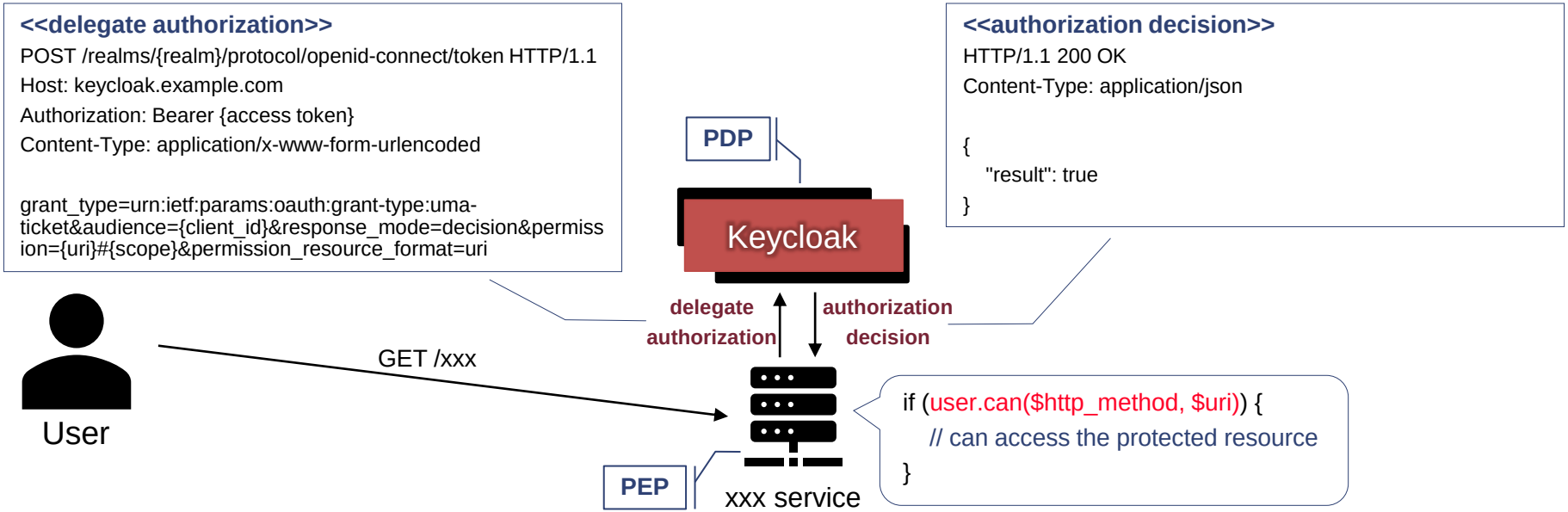
- **Resource**: the protected resource
- **Scope**: action that can be performed on resources
- **Policy**: condition that must be satisfied to access or perform operations
- **Permission**: coupling the policy with the protected resources and scopes

Scalable Authorization w/ Keycloak's Authorization Service

HITACHI
Inspire the Next

Delegating authorization to Keycloak and getting an authorization decision previously required multiple API requests, but now you can get an authorization decision with just one API request.

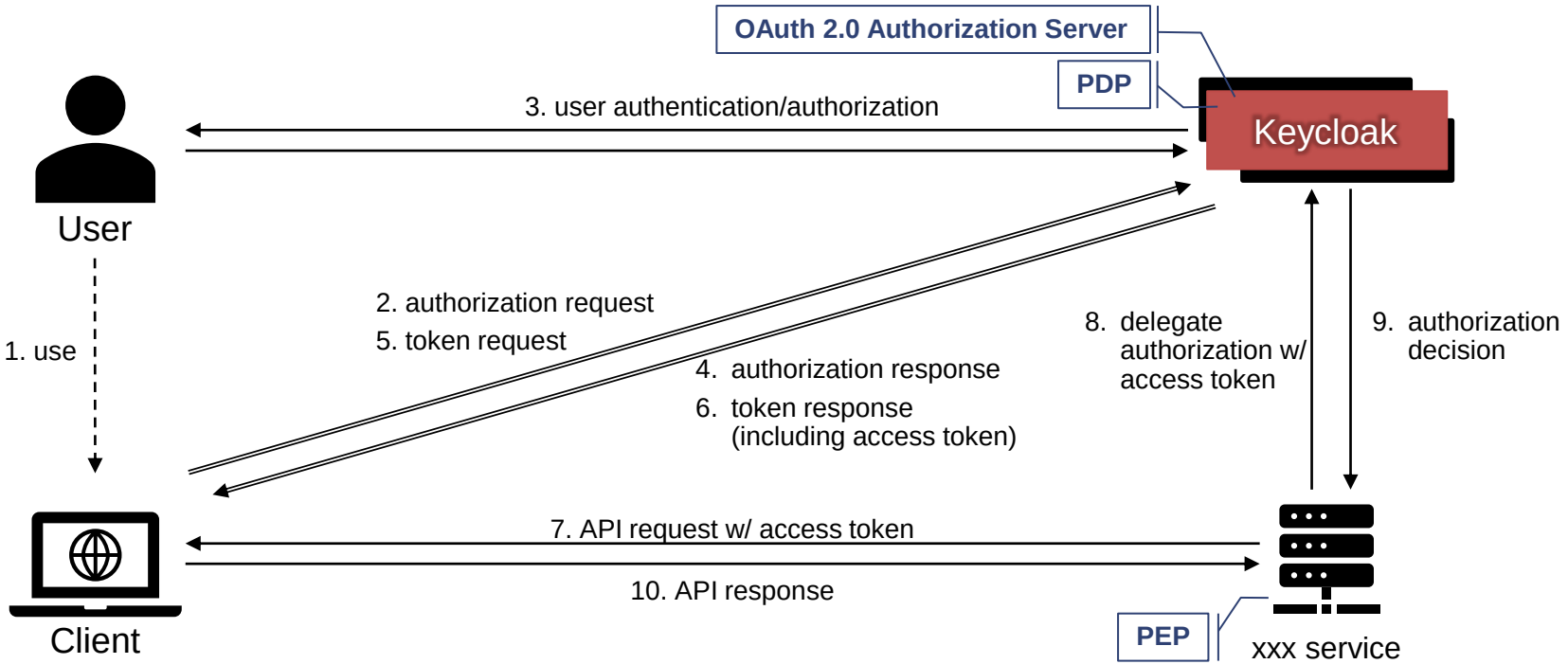
*enhanced in Keycloak 22 (PR#20237), committed by the speaker.



Scalable Authorization w/ Keycloak's Authorization Service

HITACHI
Inspire the Next

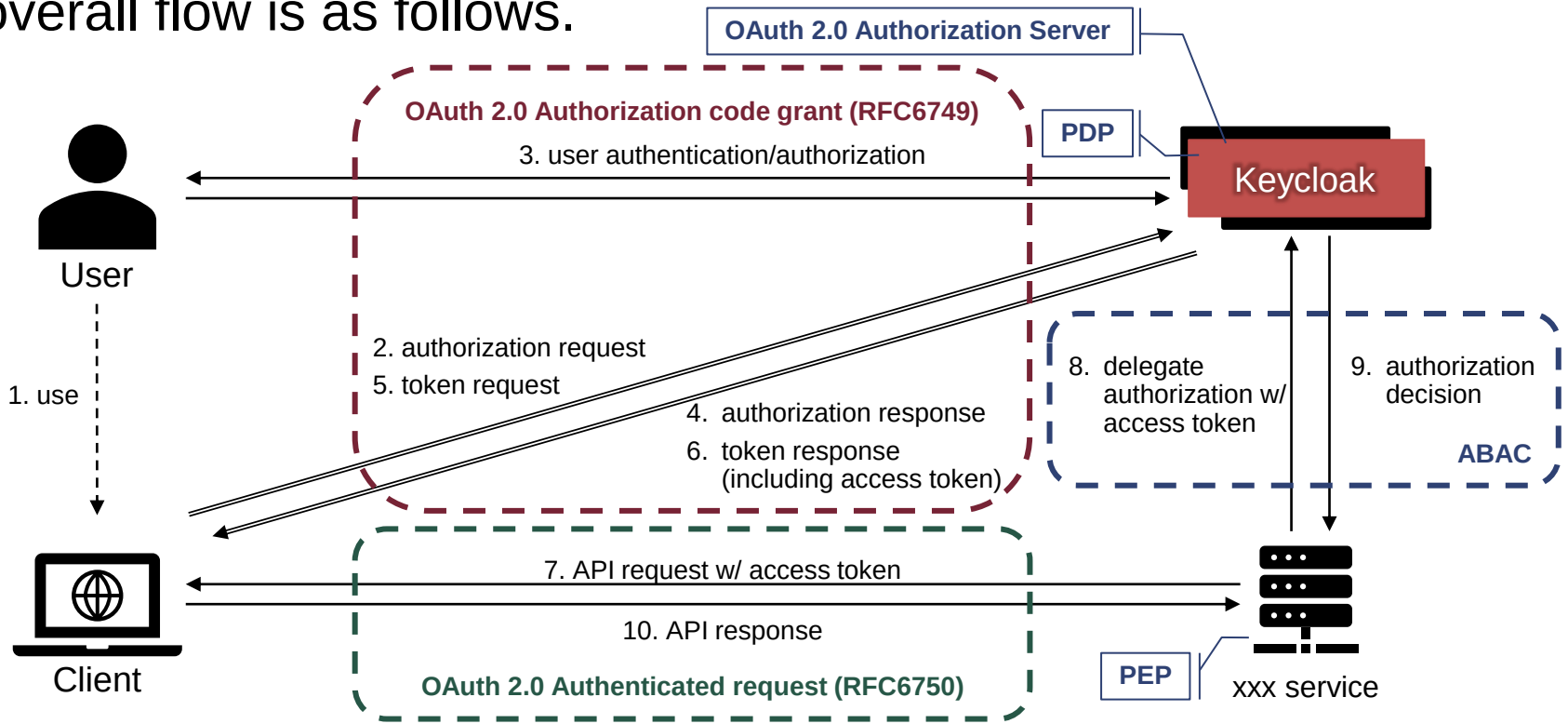
According to OAuth 2.0, the standard protocol for API authorization, the overall flow is as follows.



Scalable Authorization w/ Keycloak's Authorization Service

HITACHI
Inspire the Next

According to OAuth 2.0, the standard protocol for API authorization, the overall flow is as follows.



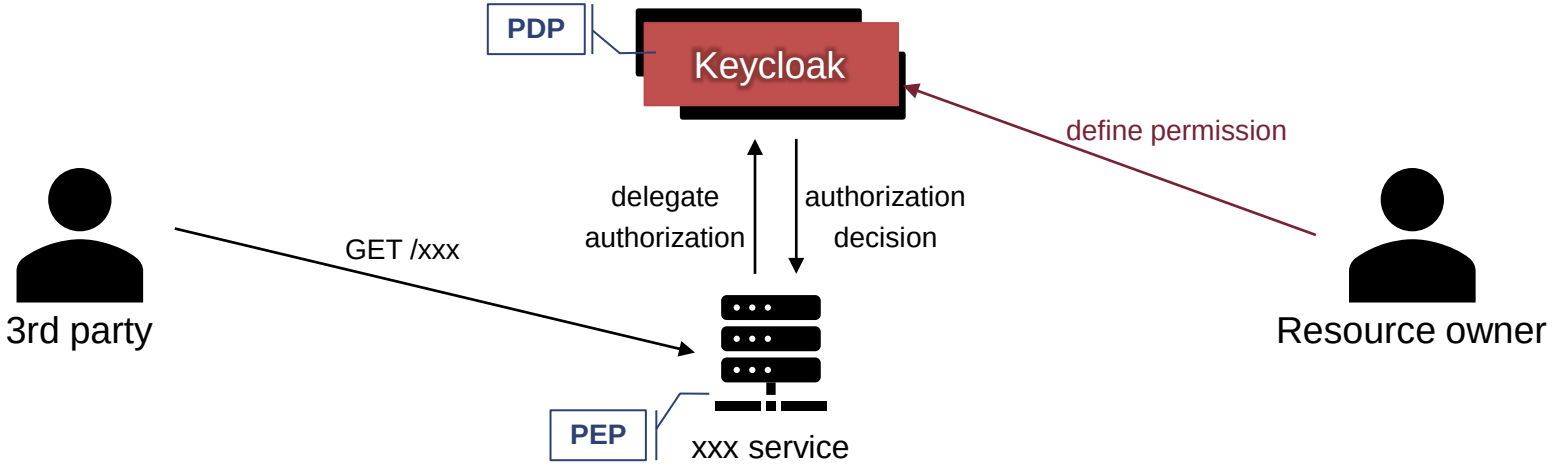
cf. User-Managed Access (UMA) 2.0 Grant

HITACHI
Inspire the Next

Keycloak is a UMA 2.0-compliant authorization server that provides most UMA capabilities.

-> Resource owners can define permissions for their resources to **third parties**.

-> More flexible authorization support is possible with various use cases!



HITACHI
Inspire the Next

Contents

Centralized Authorization vs Distributed Authorization

HITACHI
Inspire the Next

Advanced challenge that reducing the disadvantages of **centralized authorization** (using Keycloak) compared to **distributed authorization** at the edge of each service.

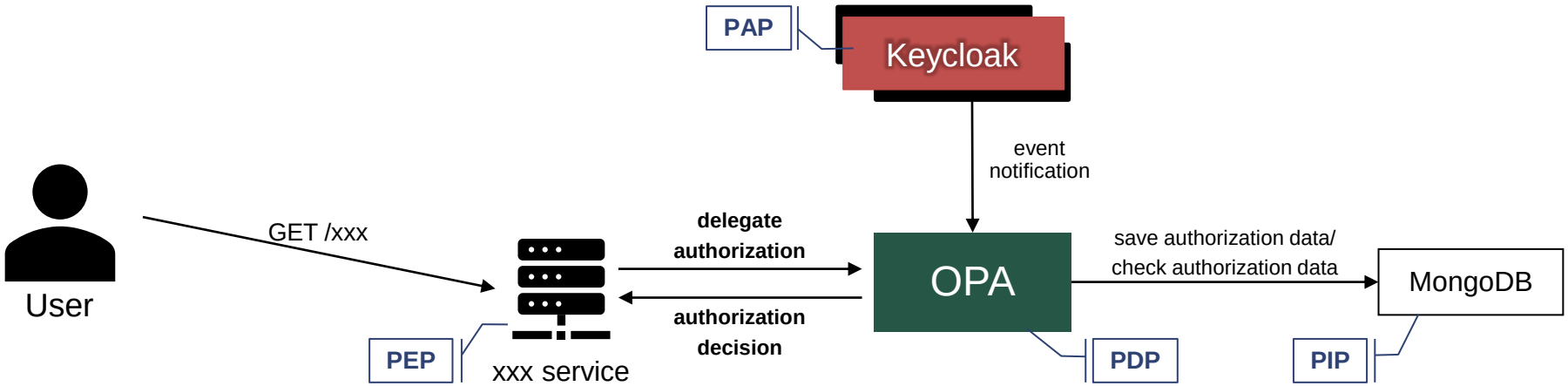
	Centralized Authorization	Distributed Authorization
Scalability/ Data Management	✓ • Multiple services do not need to store duplicate authorization data and authorization logic.	✗ • Multiple services need to store duplicate authorization data and authorization logic.
Performance	✗ • **Services need to access the centralized authorization service for every API request.**	✓ • Services only need to access local authorization logic.
Availability	✗ • **When the centralized authorization service goes down, all services deny every API request.**	✓ • Even if some services go down, the rest services go well using their local authorization logic.
Consistency	✓ • Only the centralized authorization service makes authorization decisions so consistency is guaranteed.	✗ • Local authorization logics make authorization decisions individually so consistency is not guaranteed.

Tackle Performance Challenge

HITACHI
Inspire the Next

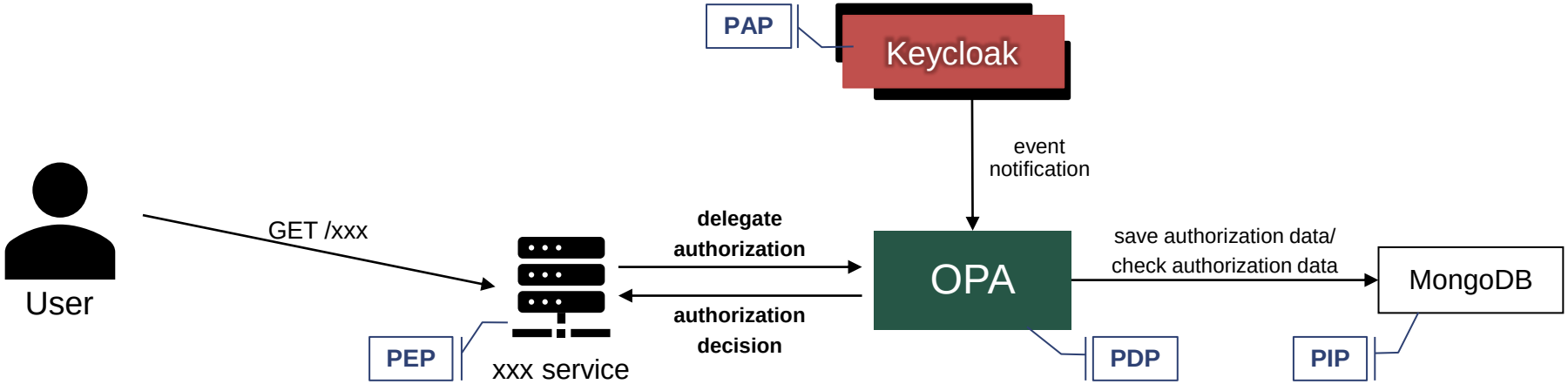
Fortunately, Keycloak is in CNCF family and can be built in local Kubernetes, communication to the centralized authorization service **less** has negative impact.

-> When performance requirements are severe, Keycloak can be combined with **OPA**.



Keycloak + OPA

- Option-1: OPA as just a cache. Performance improvement can be expected by simply caching authorization decisions from Keycloak in sidecar OPA. Of course, there is a trade-off with consistency in authorization decisions.
- Option-2: OPA as PDP. Moving the PDP functions to OPA and dedicating Keycloak to PAP (notifying policy update events, etc.) is possible, as shown below. Of course, OPA requires storing duplicate authorization information.



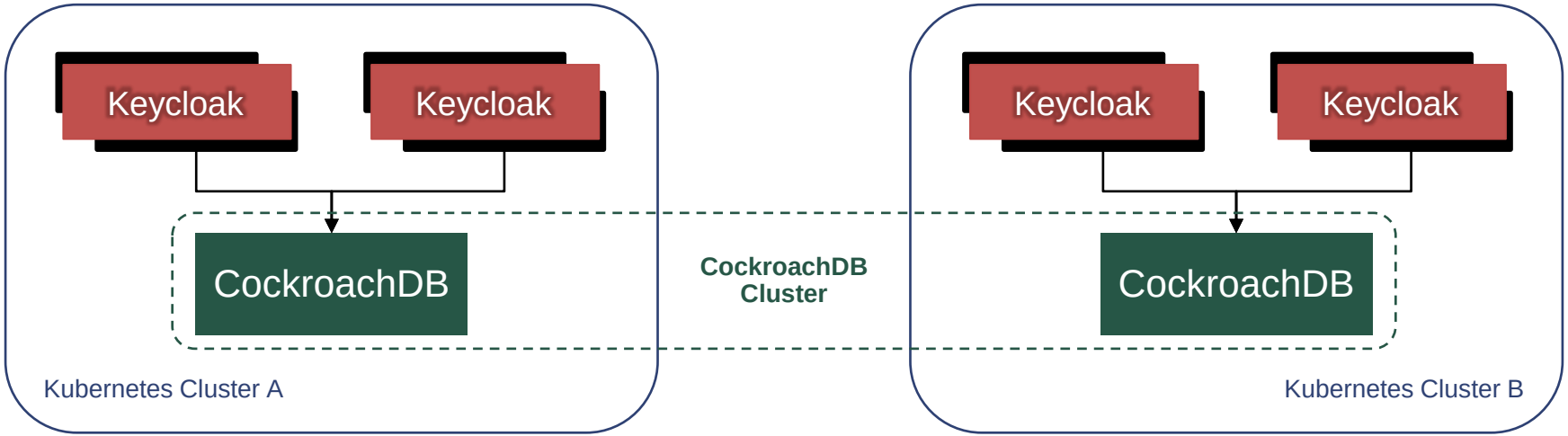
Tackle Availability Challenge

HITACHI
Inspire the Next

Combined with OPA, mentioned before, Keycloak resolves availability challenge to some extent.

-> For mission-critical services that cannot stop, combined with **CockroachDB**, Keycloak can withstand regional failures and operate in multi-cloud environments.

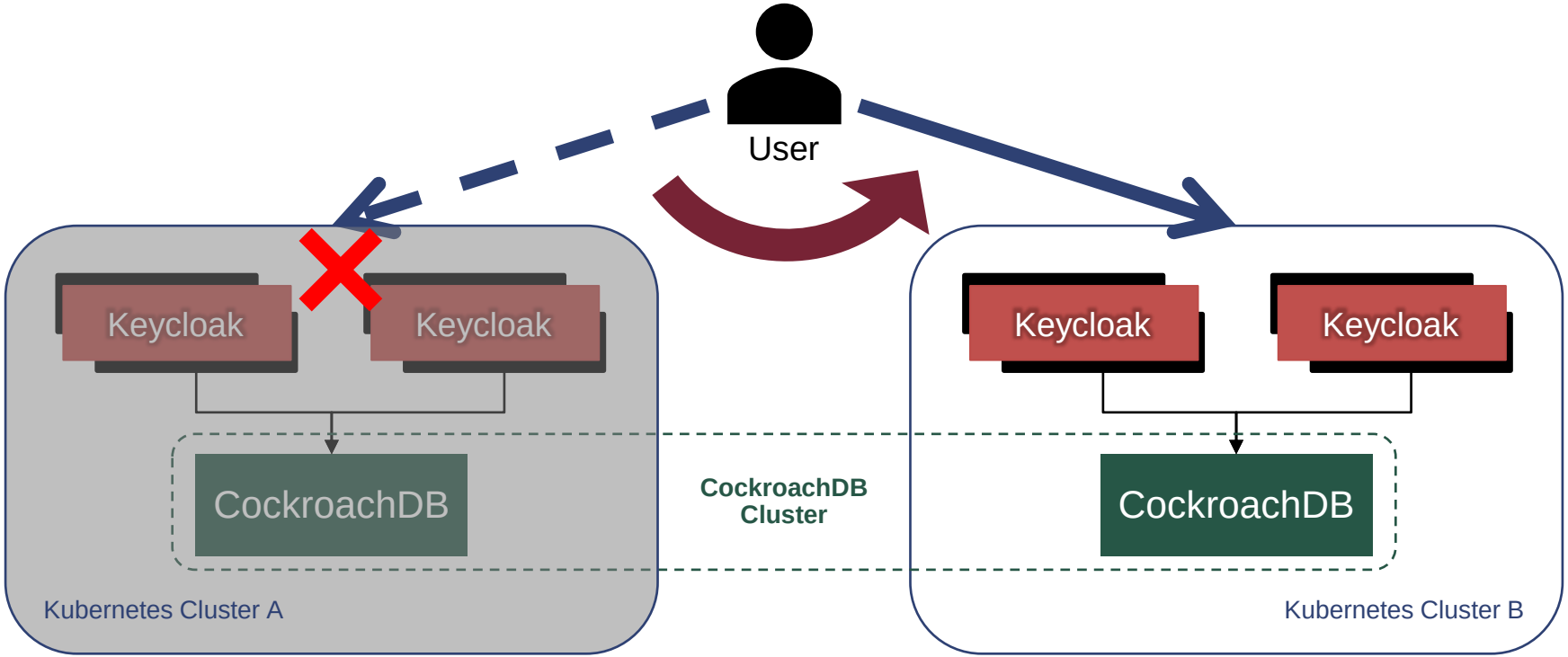
* Keycloak plans to support CockroachDB, but does not officially support it as of September 2023.



Keycloak + CockroachDB

HITACHI
Inspire the Next

- Even in the event of regional failures or large-scale cloud failures, services can go well by changing the connection destination.



Summary

HITACHI
Inspire the Next

- Authorization is becoming more and more important to security considerations.
  -> Three of the top 5 security risks include "Authorization."

- To achieve "Scalable" Authorization, separating authorization from application logic and centralizing authorization data is the keywords.

- Centralized Authorization vs Distributed Authorization.
  -> Combined with OPA / CockroachDB, centralized authorization using Keycloak can improve performance / availability.

Trademarks

- OpenID is a trademark or registered trademark of OpenID Foundation in the United States and other countries.
- GitHub is a trademark or registered trademark of GitHub, Inc. in the United States and other countries.
- Red Hat is a registered trademark of Red Hat, Inc. in the United States and other countries.
- NGINX and NGINX Plus are registered trademarks of F5, inc. in the United States and other countries.
- Twitter is a trademark or registered trademark of X Corp. in the United States and other countries.
- Facebook is a trademark or registered trademark of Meta Platforms, Inc. in the United States and other countries.
- Other brand names and product names used in this material are trademarks, registered trademarks, or trade names of their respective holders.

HITACHI
Inspire the Next

END

Challenge to Implementing “Scalable” Authorization with Keycloak

11/07/2023

Yoshiyuki Tabata

OSS Solution Center

Hitachi, Ltd.


Hitachi Social Innovation is
POWERING GOOD

QR CODE FOR FEEDBACK

HITACHI
Inspire the Next