CS411 Project 1 Presentation - Keycloak

Abdul Razak Daher Khatib
Efe Şaman
Çağla Ataoğlu
Alperen Can

Why we chose Keycloak

- Authentication is interesting
- Sensitive and critical processes
- Large variety of quality requirements
- Written in Java
- Well documented, plenty of resources

About Keycloak

- Authentication, authorization and user storage
- SSO (single sign on)
- Promises high security and reliable authentication
- Keycloak account or social identity providers
- OTP (one time password)
- Most accounts managed through LDAP (Lightweight Directory Access Protocol)
- Alternatives to LDAP also used

Technical Problems and Solutions

Groups’ Memberships Mismatch with LDAP

Problem

- LDAP can be used for book-keeping.
- Groups’ not automatically updated.
- Need to be manually updated or until user logs in.

Solution

- Easy sol.: Update immediately.
- Efficient sol.: Update periodically, or (for some sensitive groups) automatically.

Sessions Mis-management

Problem

- Multiple tabs = Multiple sessions.
- When one tabs is terminated all sessions are.
- Logical but confusing to some users.

Solution

- User can choose to terminate all.
- Or each tab is *actually* independent.
- So don’t remove access/refresh token pair for all tabs.

Sessions Mis-management

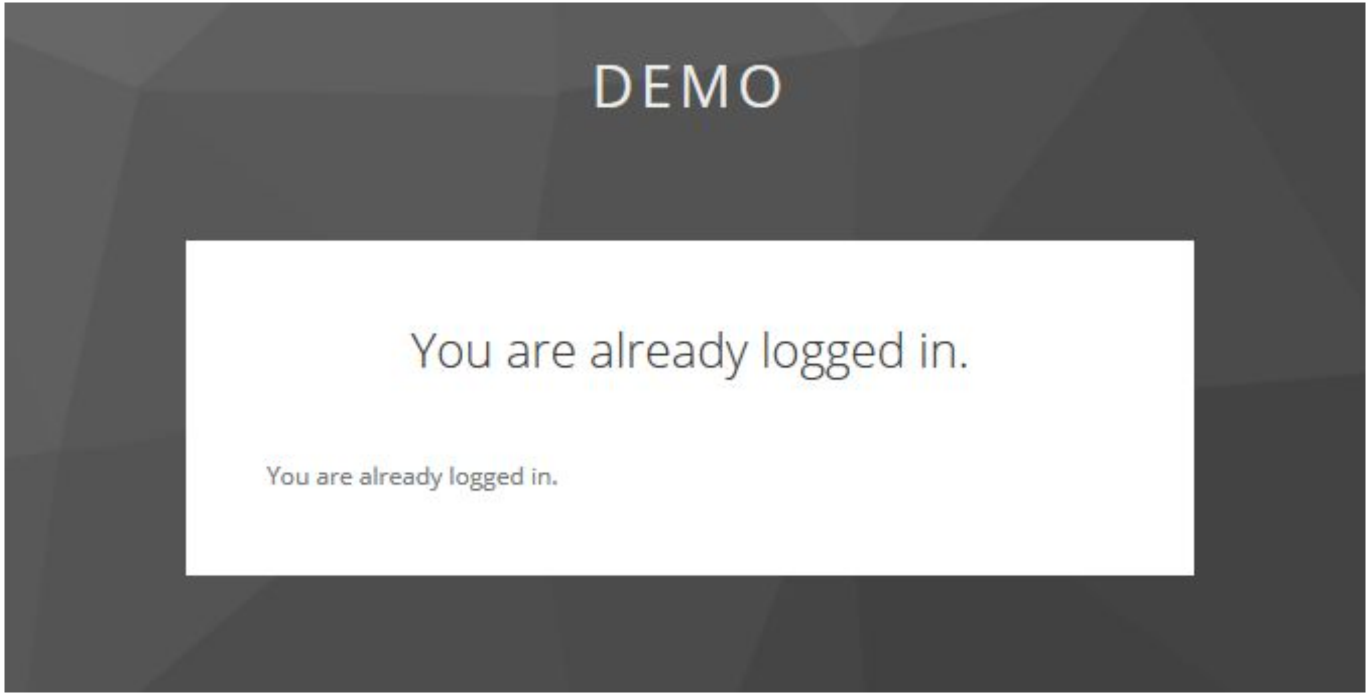
Problem

- Go throughout the login flow from another tab.
- Get “You already logged in”.
- Token is deleted so browser forgets till a request is sent.

Solution

- Don’t delete token.
- Or simply check if browser is already logged in another session when login tab is opened.

Sessions Management



You are already logged in.

You are already logged in.

Single-tenancy Performance Issues with Large Numbers of Realms

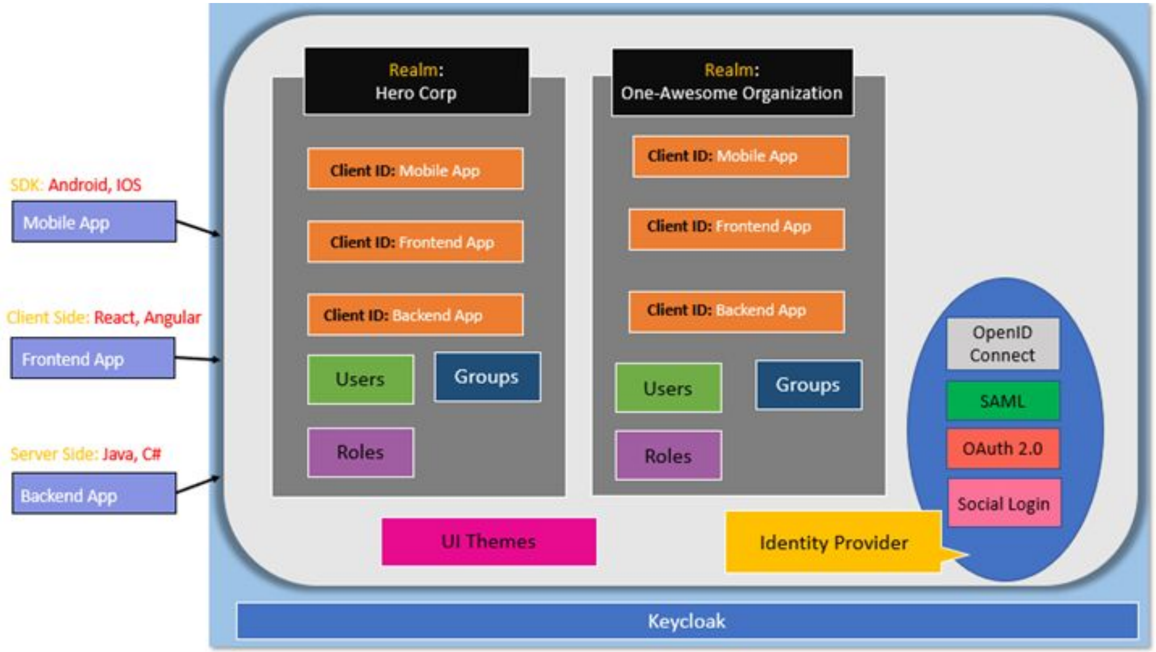
Problem

- No multi-tenancy.
- Realms are used to replicate the behaviour.
- Causes problems for big numbers of realms.

Solution

- Allow multi-tenancy by enabling application sharing between realms.

Single-tenancy Performance Issues with Large Numbers of Realms



Bulk Updates to the REST APIs in Keycloak

Problem

- Updates cannot be done in bulk.
- Single updates are very slow.

Solution

- Allow patching multiple users/realms at the same time.
- Do batch operations.

Readability Complications of Deployment and Property Configurations

Problem

- Property configuration is not separate from deployment.
- Reusability and maintainability.

Solution

- Config methods should be in a separate file from deployment methods.

Stakeholders

Developers

Fixing bugs, adding features, re-structuring the project, important especially in open source projects

Contributors

Readability is important, they want well organized, maintainable architecture.

Users

Admins of apps using Keycloak. They want to easy integration into their apps and to satisfy their end users

End Users

They want safety and security of their data. Speed, privacy, user-friendliness are important

Quality requirements

Performance

End users expect quick sign on. Requests should be handled quickly.

Security

Sensitive data such as passwords are handled. They should be transmitted with caution.

Reliability

Authentication is a sensitive process, needs to be fail-proof

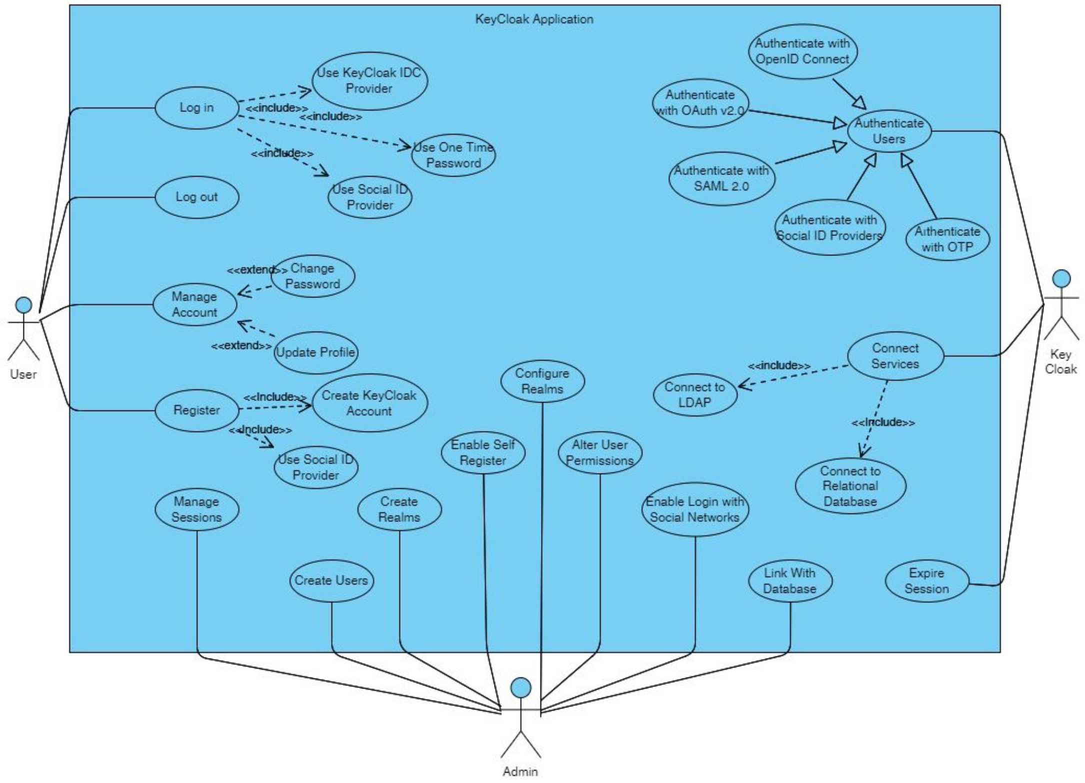
Usability

Application should be simple and smooth for the end user

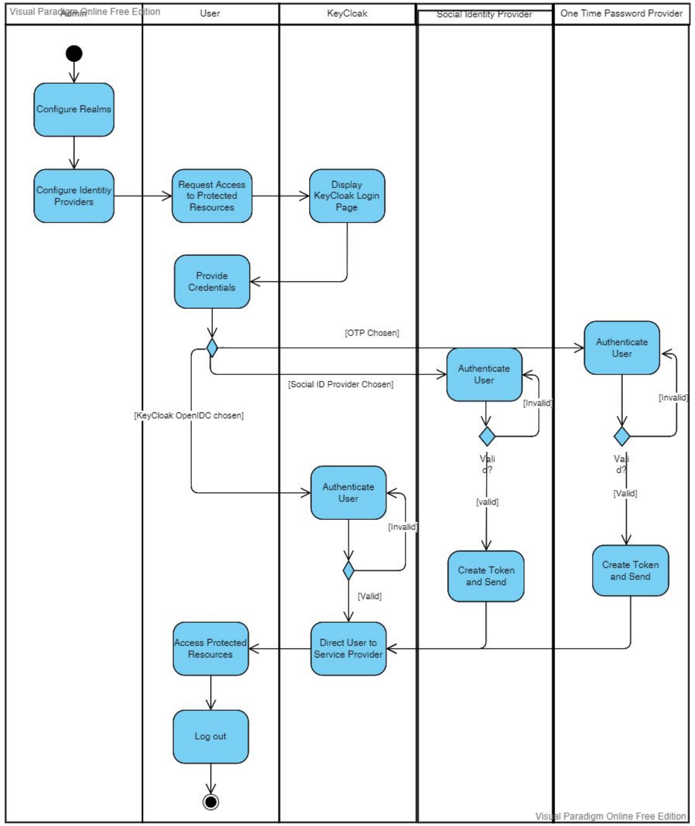
Customization

Users use Keycloak to fit different apps' needs, so it should be customizable.

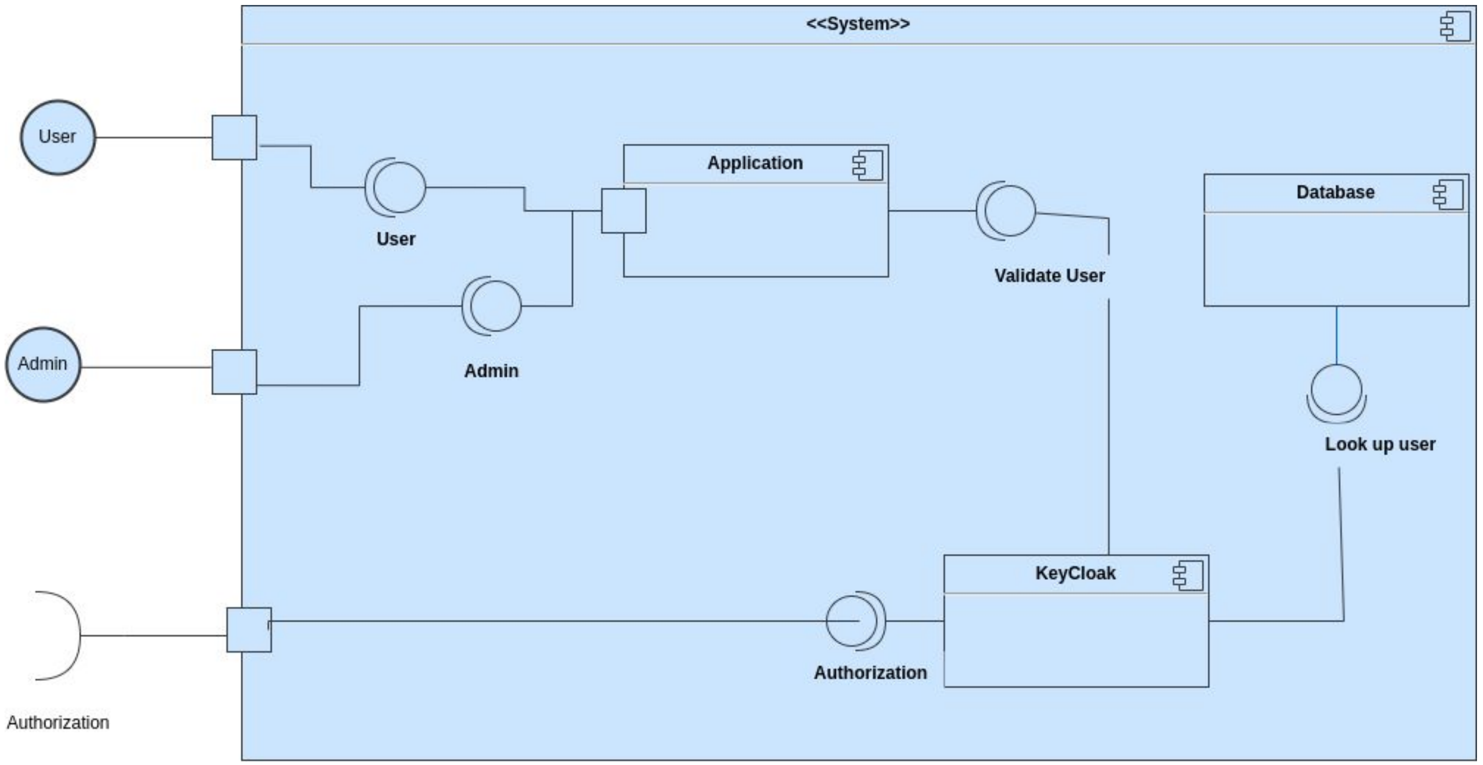
UML Diagrams



Use case diagram


Admin
User
KeyCloak
Social Identity Provider
One Time Password Provider
Configure Realms
Configure Identitiy Providers
Request Access to Protected Resources
Display KeyCloak Login Page
Provide Credentials
[OTP Chosen]
[Social ID Provider Chosen]
[KeyCloak OpenIDC chosen]
Authenticate User
[Invalid]
Valid?
[valid]
[Valid]
Create Token and Send
Direct User to Service Provider
Access Protected Resources
Log out

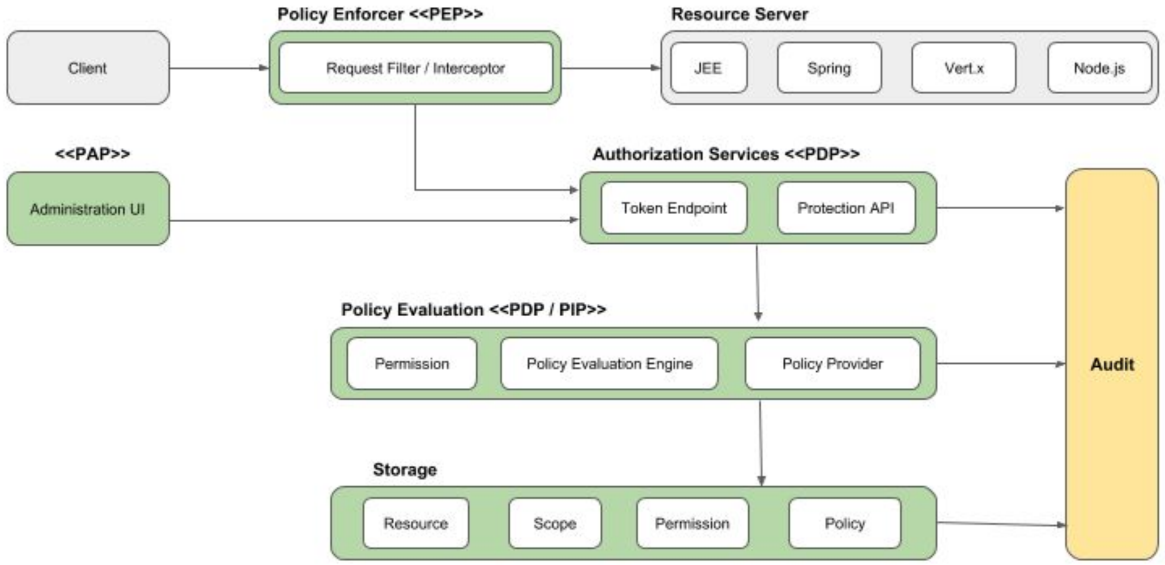
Activity diagram



Component diagram

Rationale Behind the Architecture

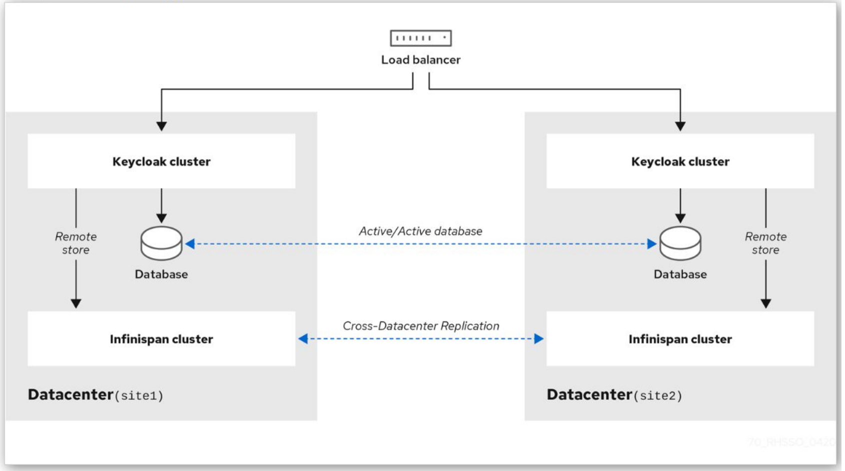
- Authorization Services Architecture:
  - Security



Rationale Behind the Architecture

- Cross-site Replication Architecture
  - Reliability & Performance

Example Architecture Diagram



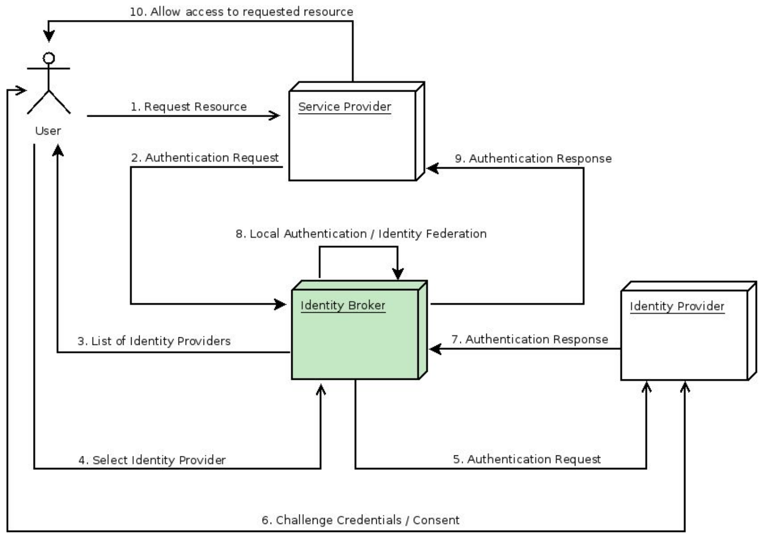
Based on the environment, you have the option to decide if you prefer:

- Reliability - which is typically used in Active/Active mode.
- Performance - which is typically used in Active/Passive mode.

Architectural Patterns

Broker

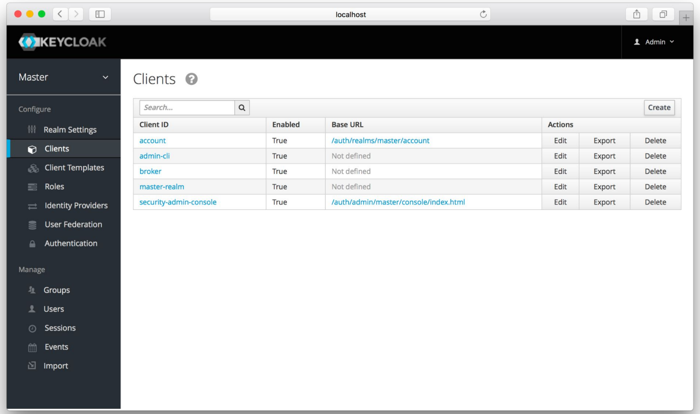
Broker component distributes requests across service providers and manages the responses. For example, identity brokers offer clients various authentication methods and distribute authentication requests among identity service providers. Keycloak can be used as an identity broker.


10. Allow access to requested resource
User
1. Request Resource
Service Provider
2. Authentication Request
9. Authentication Response
8. Local Authentication / Identity Federation
Identity Broker
Identity Provider
3. List of Identity Providers
7. Authentication Response
4. Select Identity Provider
5. Authentication Request
6. Challenge Credentials / Consent

Architectural Patterns

Client-server

Application is separated into client and server. Server contains application logic and clients request server to perform the logic (service). Keycloak users can initialize a server first and then configure clients. For example, a client application can request the server to handle SSO.



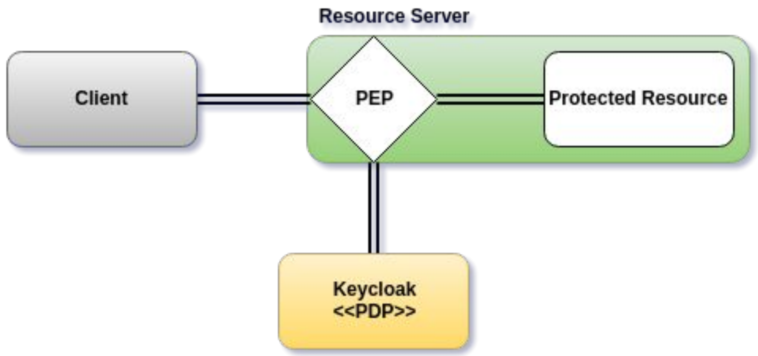
Clients

Client ID	Enabled	Base URL	Actions		
account	True	/auth/realms/master/account	Edit	Export	Delete
admin-cli	True	Not defined	Edit	Export	Delete
broker	True	Not defined	Edit	Export	Delete
master-realm	True	Not defined	Edit	Export	Delete
security-admin-console	True	/auth/admin/master/console/index.html	Edit	Export	Delete

Architectural Patterns

Policy Enforcement Point (PEP)

PEP is used for access control. PEPs are placed where authorization logic and security should be enforced. Requests are sent to the policy decision point (PDP). This way, the API is not exposed. Keycloak provides tooling for users to implement PEPs in their apps.



Proposed Extension

Evolution area

- Sessions managements (Usability).
- Groups' membership update (Usability).
- Multi-Tenancy and different databases providers based on info sensitivity (scalability/performance).
- Batch REST operation (performance).
- Separate configurations management class (reusability).

Proposed Extension

Architectural change

- Classify groups as “update frequently”, or “don’t update frequently”.
- Efficient periodic, and (in some cases) automatic updates to “update frequently” groups.
- RootAuthentication token not removed when a single Authentication token is.
- Keep a token just to signal already-logged-in, and fetch access/refresh pair for the tab.

Proposed Extension

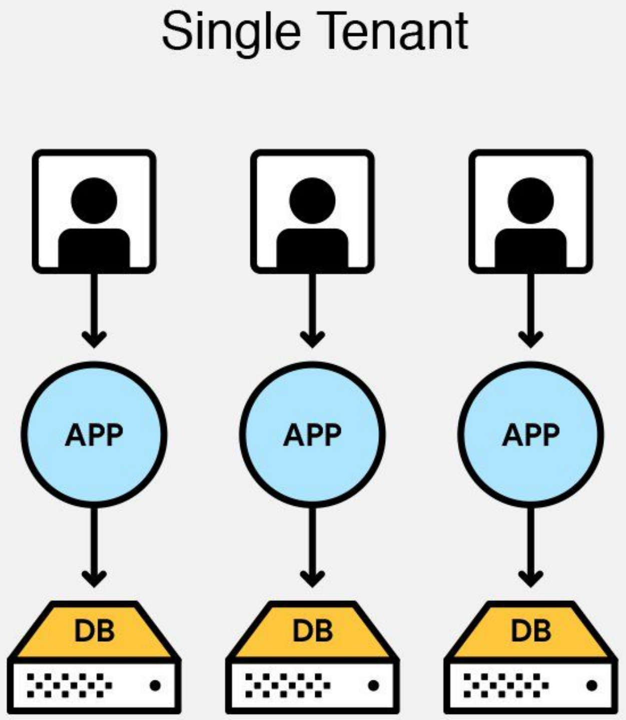
Architectural change

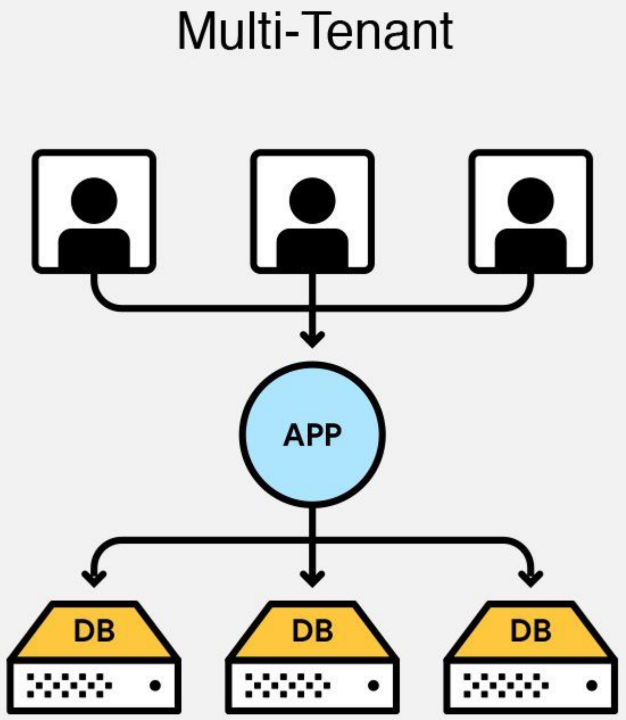
- Use multi-tenancy for better performance.
- Allowing batch operations for REST; avoiding overhead of repeating many times.
- Separate configurations manager class.

Proposed Extension

Rationale

- From single Tenancy to Multi-tenancy.
- Beneficial in terms of performance.
- Multiple applications are using multiple services, makes sense to group clients.
- Cost of starting and maintaining servers would be cut.


Single Tenant
APP
APP
APP
DB
DB
DB


Multi-Tenant
APP
DB
DB
DB

Architectural patterns

Architectural change

- Hybrid Cloud Model:
- Complimentary to Multi-Tenancy.
- Critical data on-premise.
- Non-confidential data on service providers, AWS, etc.
- Change to REST API used in KeyCloak to allow batch operations.
- Stronger compliance with Single-purpose architecture.

Hybrid Cloud – Leveraging Both Public & Private Cloud

PRIVATE CLOUD

Benefits:

- ✓ Fully Customizable
- ✓ Robust Management
- ✓ Secure by Design

PUBLIC CLOUD

Benefits:

- ✓ Low Entry Cost
- ✓ Pay As You Go
- ✓ Highly Elastic

HYBRID CLOUD = BETTER OUTCOMES

Benefits:

- ✓ Scalability
- ✓ Flexibility
- ✓ Security
- ✓ Cost efficiency

Thanks!

Your questions are welcome!