

Trilha - Software Security

Elvis Rocha

Solution Architect @ Red Hat



Como o Keycloak pode resolver meus problemas com segurança em aplicações

Elvis Rocha elvis@redhat.com @elvisnaomorreu

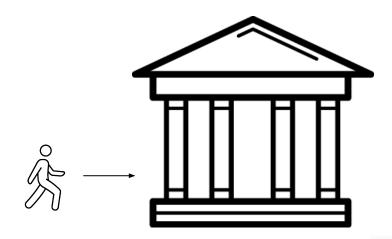


Agenda



- Segurança em aplicações
- Keycloak
- Authorization services

Como funciona a segurança para cidadãos hoje?



Quando alguém precisa entrar em algum lugar



Como funciona a segurança para cidadãos hoje?



Quem é a pessoa? Onde ela pode ir?



User	
Password	
	LOGIN



User	
Password	
	LOGIN

Basic Auth - O avô da autenticação



User	
Password	
	LOGIN

Basic Auth - O avô da autenticação

Funciona muito bem. Mas...



User	
Password	
1 43377314	
	LOGIN

E se a aplicação precisa utilizar Identity Providers externos?

E se a empresa faz uso de um serviço de diretório?

E se eu tiver aplicações em diferentes linguagens?

E se eu precisar também autenticar aplicações mobile?

E se eu quiser single sign-on entre as aplicações?

E se eu quiser uma autenticação customizada?

E o meu legado?



Qual a solução de autenticação para uma arquitetura de aplicações híbrida e distribuída?



Qual a solução de autenticação para uma arquitetura de aplicações híbrida e distribuída?





Qual a solução de autenticação para uma arquitetura de aplicações híbrida e distribuída?



Tickets, tokens e provedores de identidade



Porque utilizar tokens?

- Autenticação/autorização desacopladas da aplicação
- Não depende de um método específico
- Aplicações desktop, mobile e APIs podem se autenticar de forma diferente
- Assinado digitalmente
- Flexível
- Podem ter tempos de vida diferentes
- Podem ser invalidados





OAuth 2.0

 Federated identity: Você pode logar e acessar uma 'conta' (security context) usando outra.

• **Delegated authority**: Um serviço pode solicitar acesso a recursos em outro serviço em nome do usuário.





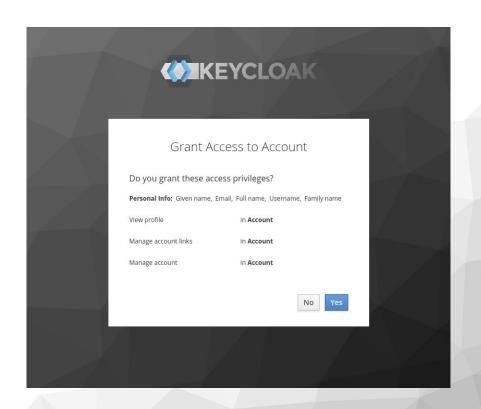
OAuth 2.0

- Resource Owner: Você mesmo (o usuário dono do recurso).
- **Resource Server**: Servidor que hospeda o dado a ser protegido (Ex.: Google que hospeda seu perfil).
- Client: Aplicação solicitando acesso a um recurso no Resource Server (Ex: uma applicação mobile).
- Authorization Server: Servidor que emite o token para o cliente. Esse token será usado por ele para acesso ao Resource Server.



OAuth 2.0

OAuth permite que as pessoas deleguem acesso a aplicativos para agir em nosso nome





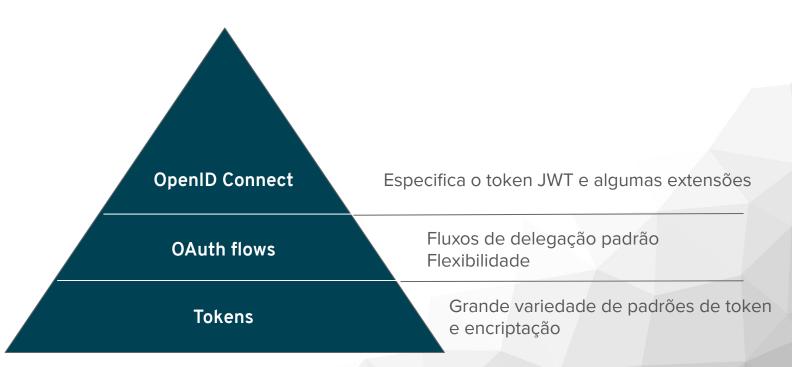
OpenID Connect

- Construído baseado no protocolo OAuth 2.0
- Permite aos clientes verificar a identidade de um usuário final
- Obtém informações básicas de perfil do usuário final
- RESTful HTTP API, usando JSON como formato padrão





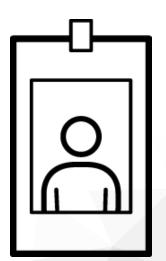
OpenID Connect





OpenID Connect

- Provê informação de identidade ao aplicativo do Authority Server
- Base64 encoded fácil de trabalhar



Nome: John Doe

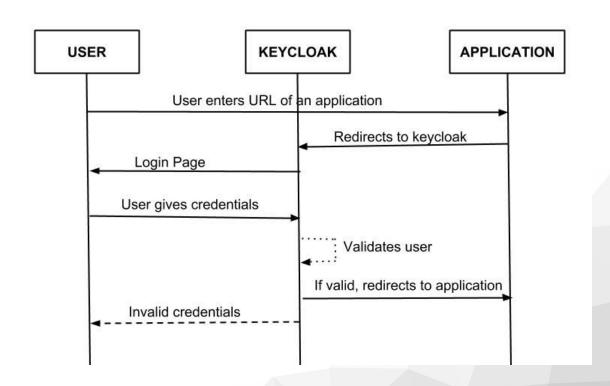
Tipo: Empregado

Empresa: ACME CO

Expiração: 02-06-2022

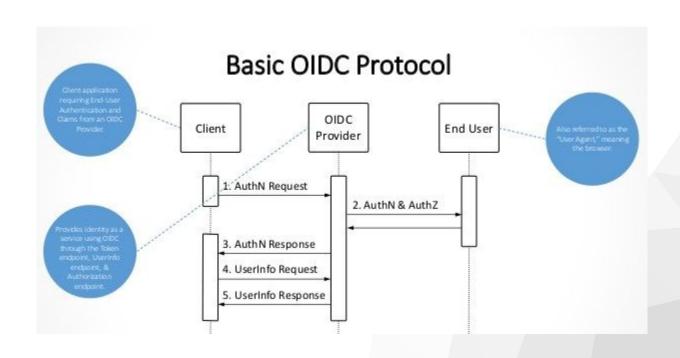


Fluxo Básico OAuth 2.0





Fluxo Básico OIDC

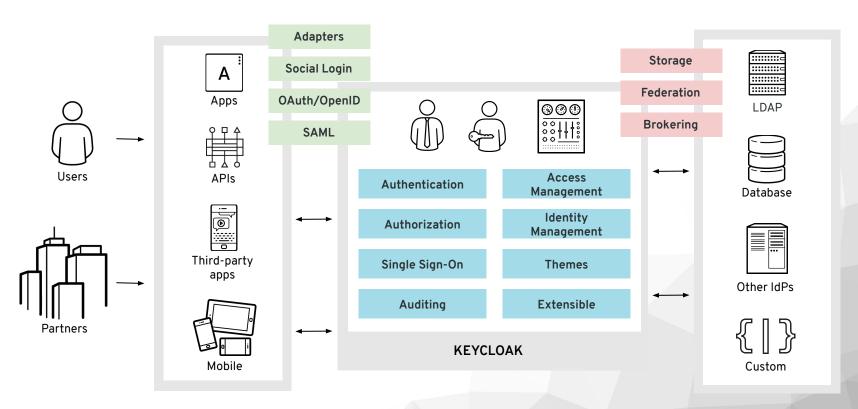






Um *Identity* e *Access Manager* self-service para aplicações modernas e APIs







Standards











Protocolos

OpenID Connect / OAuth 2.0

- JSON
- Simple
- Bearer token

Quando usar?

- App tradicional
- Single-page, mobile
- Serviços REST

SAML v2

- XML
- Mais maduro
- Mais complexo

Quando usar?

- Apps monolíticas
- Apps com suporte a SAML
- Se há requisitos extravagantes



Adapters

OPENID

- Java (JBoss EAP, Wildfly, JBoss Fuse, Tomcat, Jetty, Spring Security, Spring Boot, Wildfly Swarm, Servlet Filter)
- Javascript (client side)
- Node.js (server side)
- C# (OWIN project)
- Python (OIDC project)
- Android (AppAuth, AeroGear)
- iOS (AppAuth, AeroGear)
- Apache (php) (mod_auth_openidc)
- Keycloak GateKeeper

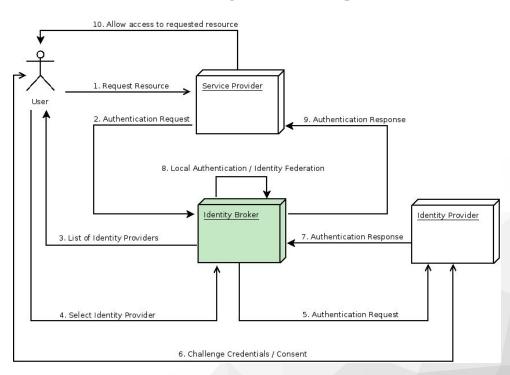
SAML

- Java (JBoss EAP, Wildfly, Tomcat, Jetty)
- Apache (php) (mod_auth_mellon)



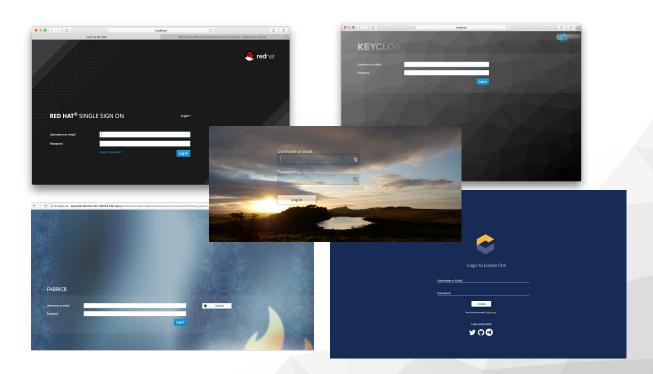


Identity Brokering



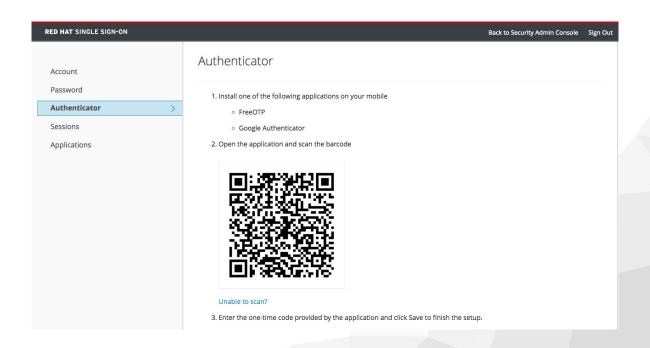


Temas





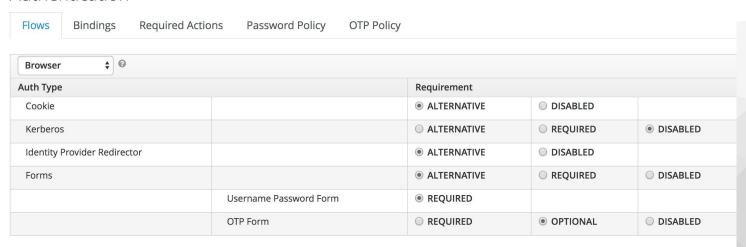
Two-factor authentication com OTP integrado





Fluxos de autenticação customizáveis

Authentication





Service Provider Interfaces

- Authentication
- Action Token
- Event Listener
- User Storage
- Realm
- User
- Role
- Group
- Client

```
Person.java >
           CorporativoUserStorageProvider.java
57
59
       @Local(CorporativoUserStorageProvider.class)
       public class CorporativoUserStorageProvider
                implements UserStorageProvider,
62
                UserLookupProvider,
63
                CredentialInputValidator,
                CredentialInputUpdater,
64
                UserQueryProvider,
65
                UserRegistrationProvider,
                OnUserCache {
67
```

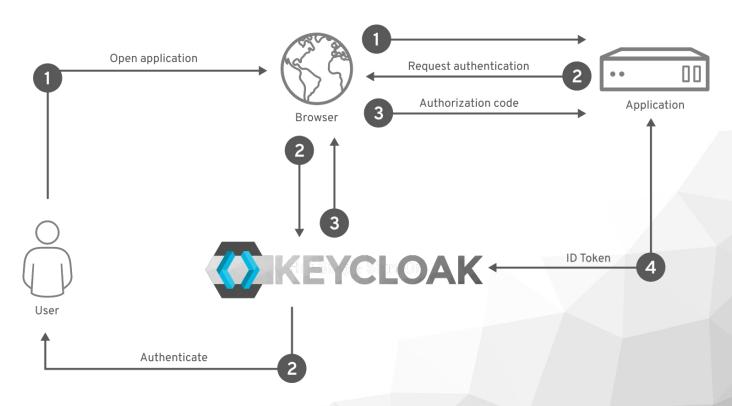


Porque o Keycloak?

- Delegue sua segurança!
- Não há necessidade de gravar usuários e senhas
- Crie temas, formulários e ações customizadas
- Integre as aplicações novas e o legado
- Não reimplemente o que já está implementado!
- Single Sign-On e uma penca de coisa legal!

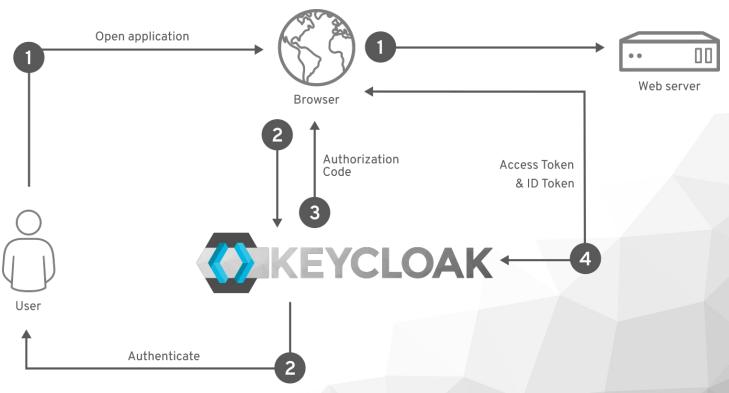


Fluxo de uma app monolítica





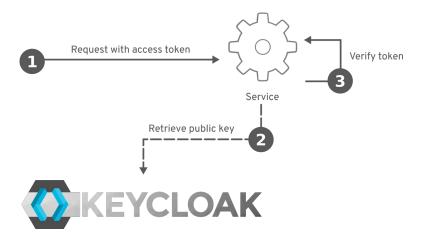
Fluxo de uma app Single-Page Application



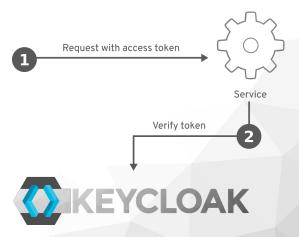


Fluxo de um Serviço

Verificação offline com assinatura digital

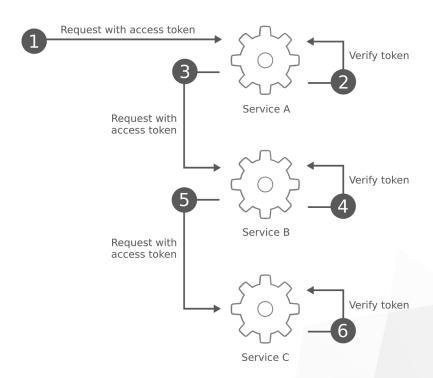


Verificação online





Fluxo de microsserviços





OAuth 2.0

• **Federated identity**: Você pode logar e acessar uma 'conta' (security context) usando outra.

 Delegated authority: Um serviço pode solicitar acesso a recursos em outro serviço em nome do usuário.





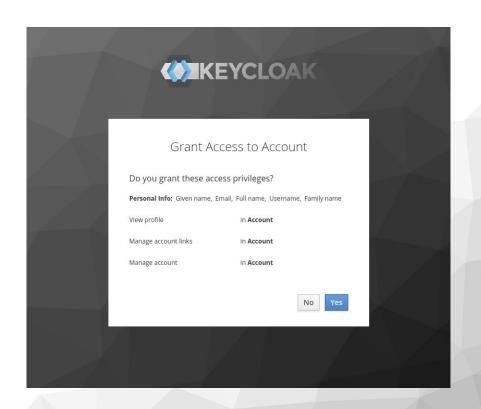
OAuth 2.0

- Resource Owner: Você mesmo (o usuário dono do recurso).
- **Resource Server**: Servidor que hospeda o dado a ser protegido (Ex.: Google que hospeda seu perfil).
- Client: Aplicação solicitando acesso a um recurso no Resource Server (Ex: uma applicação mobile).
- Authorization Server: Servidor que emite o token para o cliente. Esse token será usado por ele para acesso ao Resource Server.



OAuth 2.0

OAuth permite que as pessoas deleguem acesso a aplicativos para agir em nosso nome





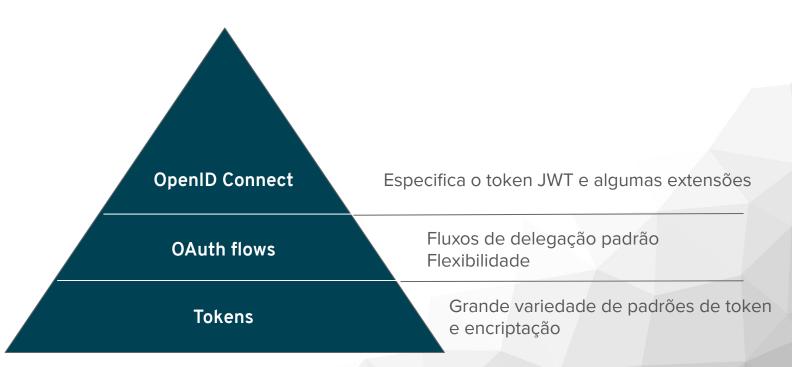
OpenID Connect

- Construído baseado no protocolo OAuth 2.0
- Permite aos clientes verificar a identidade de um usuário final
- Obtém informações básicas de perfil do usuário final
- RESTful HTTP API, usando JSON como formato padrão





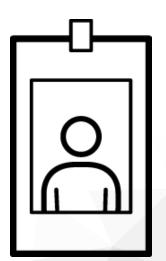
OpenID Connect





OpenID Connect

- Provê informação de identidade ao aplicativo do Authority Server
- Base64 encoded fácil de trabalhar



Nome: John Doe

Tipo: Empregado

Empresa: ACME CO

Expiração: 02-06-2022



AUTHORIZATION SERVICES

Como funciona a autorização hoje?

Role Based Access Control (RBAC)

- Hoje a grande maioria das aplicações tomam decisões baseado em Roles.
- As Roles s\u00e3o usadas para mapear funcionalidades e decidir se o usu\u00e1rio tem acesso a recursos protegidos.



Como funciona a autorização hoje?

Role Based Access Control (RBAC)

Limitações no uso de Roles:

- Recursos e Roles possuem alto acoplamento, uma mudança em um local impacta o outro e vice-versa.
- Mudança em requisitos pode implicar em mudança no código
- Gerenciar roles é difícil e propenso a erros
- Não é um mecanismo flexível. As vezes as roles não representam toda a informação e falta um contexto.



O que é

- Authorization Services é a parte de autorização oferecida pelo Keycloak que implementa a especificação User-Managed Access (UMA).
- O UMA é uma extensão do OAuth criada para fornecer aos Resource Owners um gerenciamento mais granular através de políticas de autorização em um servidor centralizado.
- Com base nessas políticas oferecem concessão de acesso em favor do usuário para dizer o quê, quem e por quanto tempo o dado poderá ser acessado.



Benefícios

- Ideal para os ambientes heterogêneos de hoje onde há uma demanda por compartilhamento da informação em diferentes regiões, locais e dispositivos com diferentes políticas
- Proteção ao recurso, políticas de autorização granulares e diferentes mecanismos de controle de acesso
- Centralização do gerenciamento de recurso, permissão e políticas
- Workflows pré-definidos de autorização e User-Managed Access (UMA)
- Evita código replicado entre projetos e se adapta a mudanças nos requisitos de segurança em tempo de execução

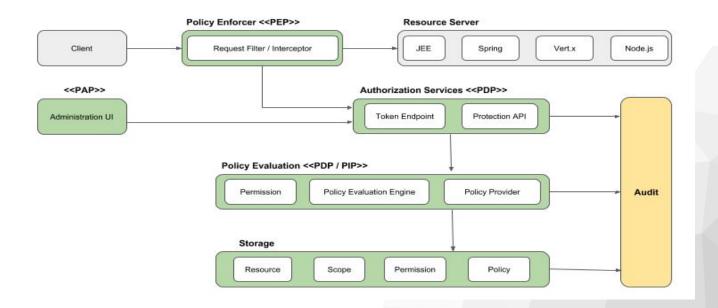


Padrões da Arquitetura UMA

- Policy Administration Point (PAP)
 - Uls para gerenciar Resource Servers, Resources, escopos, permissões e políticas.
- Policy Decision Point (PDP)
 - Ponto de decisão onde requisições são enviadas e avaliadas de acordo com a política atual e permissões definidas.
- Policy Enforcement Point (PEP)
 - Implementações para diferentes ambientes para impor as decisões ao Resource Server
- Policy Information Point (PIP)
 - Para obter informações e atributos durante a avaliação das políticas



Padrões da Arquitetura UMA





Processo de autorização

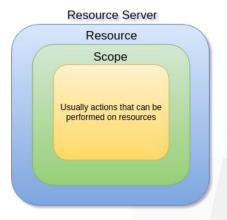
- O processo de autorização de um recurso envolve 3 passos:
 - Gerenciamento do Recurso
 - Gerenciamento das políticas e permissões
 - Imposição das políticas





Processo de autorização

Na interface de administração do Keycloak é possível definir as aplicações como
 Resource Servers e começar a gerenciar os recursos e escopos do projeto





Processo de autorização

Resource

 Um recurso pode ser qualquer coisa, uma classe, um EJB, uma API REST ou um grupo de objetos

Bank Account

Representa uma bank account genérica Políticas para esse resource são comuns a todas as contas

Alice Account

Representa a conta da Alice Somente ela (owner) pode ver e alterar os dados



Processo de autorização

Resource

- Um recurso pode ser qualquer coisa, uma classe, um EJB, uma API REST ou um grupo de objetos
 - 1. Define o Resource Server
 - 2. Define os Resources que serão protegidos
 - 3. Configura permissões e políticas





Processo de autorização

Scope

 Um escopo é uma extensão que define o que é possível fazer em um único ou um conjunto de resources.

Exemplo:

- 1. Resource:
 - Bank Account
- 2. Escopos:
 - Visualizar, Editar, Deletar



Processo de autorização

Permissão

- A permissão associa o objeto protegido a políticas que serão avaliadas para definir se o acesso pode ser concedido
- Ou seja, Permissão é o link da Policy com Resource/Scope

X pode fazer **Y** no recurso **Z**:

- X representa um ou mais usuários, roles ou grupos
- Y representa a ação (visualizar, editar)
- Z representa o recurso protegido (/accounts)



Processo de autorização

Policy

- A partir dos recursos/escopos definidos é possível criar políticas (condições que devem ser satisfeitas para acessar ou executar operações em algo)
- Uma policy está relacionada a diferentes mecanismos para proteger o recurso
 (ACMs). É possível ter policies agregadas.

Mecanismos de controle de acesso (ACMs):

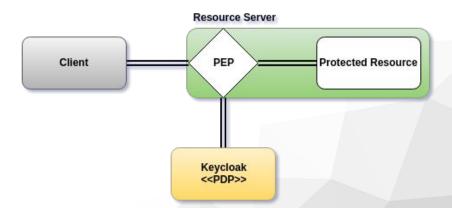
- Attribute-based (ABAC)
- Role-based (RBAC)
- User-based (UBAC)
- Context-based (CBAC)



Processo de autorização

Policy Enforcement

- Envolve os passos necessários para impor autorização ao Resource Server
- Após habilitar o PEP no Resource Server ele estará apto a comunicar com o
 Authorization Server para pedir dados de autorização e controle de acesso.





Baseado nas especificações OAuth2 e User-Managed Access 2 (UMA 2 Profile)

Clientes OAuth2 (ex.: app frontend) obtém Access Tokens através do Token
 Endpoint no RH-SSO e usam o mesmo token para acesso a recursos
 protegidos em um Resource Server (ex.: app backend).

Da mesma forma através do Authorization Services é possível prover extensões ao protocolo OAuth2 para permitir a emissão de tokens baseados nas políticas de acesso ao recurso protegido solicitado.



Baseado nas especificações OAuth2 e User-Managed Access 2 (UMA 2 Profile)

- Na prática os Resource Servers podem impor restrições de acesso ao recurso protegido baseado em políticas diversas definidas.
- O token emitido com essas políticas é chamado de RPT (Request Party Token)













UMA

Privacidade

Permissões concedidas com base nas políticas definidas pelo usuário

Party-to-Party Authorization

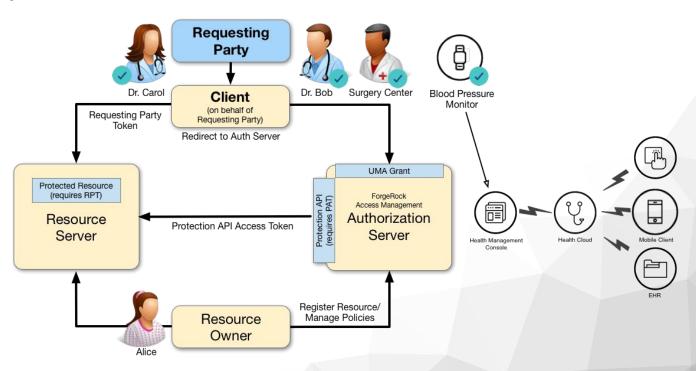
 Usuários (resource owners) podem gerenciar o acesso aos recursos e autorizar terceiros a acessar esses recursos de maneira totalmente assíncrona

Resource Sharing

 Usuários (resource owners) tem permissão para gerenciar permissões a seus recursos e decidir quem pode acessar um recurso em particular e como.



UMA





UMA

- PASSO 1
 - Cliente solicita acesso a um Resource sem mandar o RPT

curl -X GET \
http://\${host}:8080/my-resource-server/resource/1bfdfe78-a4e1-4c2d-b142-fc92b75b986f



UMA

PASSO 2

Resource Server responde com o permission ticket e o parâmetro as_uri

```
HTTP/1.1 401 Unauthorized WWW-Authenticate: UMA realm="${realm}", as_uri="https://${host}:${post}/auth/realms/${realm}", ticket="016f84e8-f9b9-11e0-bd6f-0021cc6004de"
```



UMA

PASSO 3

 Agora que o client tem o permission ticket e a localização do SSO Server, ele manda uma solicitação de autorização ao endpoint Token

```
curl -X POST \
http://${host}:${port}/auth/realms/${realm}/protocol/openid-connect/token \
-H "Authorization: Bearer ${access_token}" \
--data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
--data "ticket=${permission_ticket}
```



UMA

PASSO 4

 Se o SSO validar a solicitação de permissão ele emite o RPT com as permissões associadas

```
HTTP/1.1 200 OK
Content-Type: application/json
...
{
    "access_token": "${rpt}",
}
```



UMA

PASSO 4

Se o cliente n\u00e3o tiver permiss\u00e3o ele responde com um 403:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
...
{
    "error": "access_denied",
    "error_description": "request_denied"
}
```



UMA

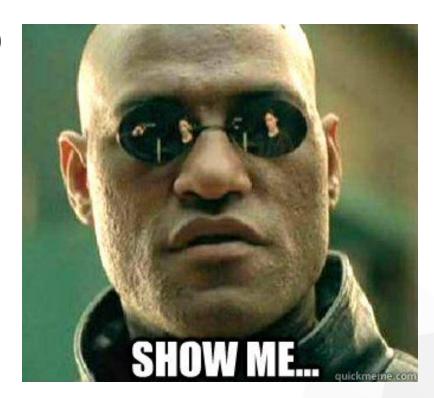
PASSO 5

 Em alguns casos, apps clientes precisam iniciar um "Asynchronous Authorization Flow" para solicitar ao owner do Resource se ele permite ou não o acesso ao recurso.

```
curl -X POST \
  http://${host}:${port}/auth/realms/${realm}/protocol/openid-connect/token \
  -H "Authorization: Bearer ${access_token}" \
  --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
  --data "ticket=${permission_ticket} \
  --data "submit_request=true"
```



demo



time!



Obrigado

Mais?

Keycloak website https://www.keycloak.org/

Keycloak github https://github.com/keycloak/keycloak

Demo https://github.com/redhat-sa-brazil/demo-authz

Community mailing list https://lists.jboss.org/mailman/listinfo/keycloak-user

Twitter @keycloak

