

| SOLUTION **ARCHITECTURE** APPROACH DOCUMENT

ONEAUTH
FRAME-
WORK



Section -1

KeyCloak -
Comprehensive Outlook

Section -2

How To Authenticate
Your Spring Boot
Application with
Keycloak

Section-3

Secure Your Spring Boot
App with Spring Security
and GitHub Using
Keycloak

Section - 4

Comparison with other
identity Providers

ONE AUTH FRAMEWORK

OEPN SOURCE IDENTITY PROVIDER APPROACH - KEYCLOAK

This document illustrates the approach to build One AUTH (Authentication & Authorization) framework using Open-source identity provider KeyCloak.

The framework is build using Identity aggregator approach and helps to build common solution for security enablement in enterprise apps.

The approach document consists of KeyCloak features, its usage , distribution approach , integration, and common use cases.

The detailed steps for securing Spring Boot application using KeyCloak.

Additionally, KeyCloak unique feature of supporting Social identity provider for authentication/Authorization has been covered.

In the last section, we have outlined few points about features of different identity providers in the marketplace and its relevance.

As a next step, we will add additional information related to OAUTH2.0 and other concepts such as OPENID Connect, JWT, and other detailed authentication/authorization techniques relevant to application security

1.0 KeyCloak – Perspective and its relevance

What Is Keycloak?

It is a tool for “Identity and Access Management”, as written on their project page on GitHub. Additionally, Keycloak is an open-source tool currently licensed with Apache License 2.0. It is also an upstream project for Red Hat SSO, so if you are looking for something more enterprise-centered, you can check it.

The full list of supported platforms depends on which protocol you decide to use, currently Keycloak supports three different protocols, and can be viewed in documentation. Keycloak’s initial release took place in September 2014; the current version is **15.0.2 (2021-10-09)**. It is developed and maintained by people from Red Hat. They are open to new contributors if anyone is interested.

Keycloak Features

- Multiple Protocols Support

As for now Keycloak supports three different protocols, namely - OpenID Connect, OAuth 2.0 and SAML 2.0.

- SSO

Keycloak has full support for Single Sign-On and Single Sign-Out.

- Admin Console

Keycloak offers web-based GUI where you can “click out” all configurations required by your instance to work as you desire.

• User Identity and Accesses

Keycloak can be used as a standalone user identity and access manager by allowing us to create users database with custom roles and groups. This information can be further used to authenticate users within our application and secure parts of it based on pre-defined roles.

• External Identity Source Sync

In case when your client currently has some type of user database, Keycloak allows us to synchronize with such database. By default, it supports **LDAP** and **Active Directory** but you can create custom extensions for any user database using Keycloak User storage API. Keep in mind that such a solution may not have all data necessary for Keycloak to be fully functional, so remember to check if your desired functionality works.

• Identity Brokering

Keycloak can also work as a proxy between your users and some external identity provider or providers. Their list can be edited from Keycloak Admin Panel.

• Social Identity Providers

Additionally, Keycloak allows us to use Social Identity Providers. It has built-in support Google, Twitter, Facebook, Stack Overflow but, in the end, you have to configure all of them manually from admin panel. The full list of supported social identity providers and their configuration manual can be found in Keycloak documentation.

• Pages Customization

Keycloak lets you customize all pages displayed by it to your users. Those pages are in **.ftl** format so you can use classic **HTML** markups and **CSS** styles to make the page fit your application style and your company brand. You can even put custom **JS** scripts as part of pages customization so possibilities are limitless.

These are all of Keycloak features which I wanted to describe today. Of course, this tool offers even more possibilities, which are described in a much more detailed way in the documentation.

Distributions of Keycloak

Currently, Keycloak has three major distributions.

• Server

Standalone application is downloadable from Keycloak page in form of a tar or zip archive with all scripts, docs, and assets needed to work normally. As for now, there are two main versions of this distribution: one is powered by WildFly server while the other is powered by Quarkus. It is now in preview stage so some unexpected error may occur.

• Docker Image

Distribution appropriate for Docker, Podman, Kubernetes, and OpenShift. There are two official docker images for Keycloak: one is held in Quay Container Registry

- quay.io/keycloak/keycloak, the second one is held in Docker Hub

- jboss/keycloak. You can download both of them with a simple `docker pull` command.

• Operator

Distribution for Kubernetes and OpenShift based on Operator SDK.

As you can see, everybody can find an appropriate distribution. If you use Docker or Kubernetes you have Keycloak image and operator. On the other hand, if you prefer a more conventional deployment type you will also find a distribution for you. Even then Keycloak Docker image can be extremely useful for development and testing.

You can set up your test Keycloak server then do changes and test them. After tests you can restart your Docker image and all changes made to your Keycloak will be reverted and you will get a clear environment for further tests. All three distributions can be downloaded from here.

Keycloak Integrations

So now you know basics of Keycloak and its features. The last remaining question is - how to integrate it into your app?

Here I will speak mostly from the perspective of Java `eco-system` but I will mention also some other languages and frameworks. In Java currently the most popular frameworks like Spring Boot, Quarkus and Micronaut have some sort of adapters that make integrating with Keycloak really easy.

In case of Spring Boot, it is `spring-boot-keycloak-starter` while in case of Quarkus it is `quarkus-keycloak-authorization`.

On the other hand, in Python package `python-keycloak` seems pretty useful.

For Scala-based application library, `keycloak4s` also sounds good.

For C# based application Keycloak.Net looks like a handy lib.

All libraries are open source, developed and maintained by the community built around Keycloak. I will put respective links in the end of the article.

In the case of Spring Boot and Quarkus, thanks to the framework provided abstractions, the whole integration requires just a few lines of code and filling some configuration properties. In other cases, libraries only provide clients for Keycloak API so integration could be more complex.

Pointers - Keycloak

First of all, it is free. You may think that it is funny but in fact, most tools with such features like AuthO or Okta are paid.

Secondly, it supports three different authentication protocols which give you the possibility to cover many applications with different security demands with a single tool.

Additionally, you can choose an authentication protocol basing on what you need or what you think will be better for your application and you are not limited by the tool you are using. Keycloak is also an upstream project for Red Hat SSO product so you can be sure that it is a well written and well designed system.

Moreover, it has big community support which guarantees that there are a lot of examples of how to do something and that you can count on others to help you with your problems. Keycloak can be very useful when your client has some existing user database like LDAP or Active Directory because it has a built-in mechanism for synchronization with such identity providers.

Additionally, Keycloak supports social identity providers like Google or Facebook straight out of the box so if you want to use Social Login, Keycloak may be very useful for you and your team.

Furthermore, it provides web-based GUI which makes any configurations changes easier. In the end, thanks to Keycloak SSO support you can facilitate your users' access to multiple services run by your company.

When It May Not Be the Best Choice

I said why you should use Keycloak so, to stay objective, I will say something about when it may not be the best choice for you.

- Single application with just one client in Keycloak realm – lose all benefits of SSO
- No integration with AD or any other user data provider
- No Social Login

- Pure user database — Keycloak can be used this way but so can a database with specific tables, and it can be much easier to configure if you already have one
- Some kind of enterprise-level guarantees — Keycloak is still an open-source project so you do not have any guarantee provided by its producer about its working or road map and things like customer support are taken care of by Stack Overflow and surely with no hard deadlines for response time

Keep in mind that meeting any or even all of these conditions by your application does not straight away indicate that you should not be using Keycloak. It only means that it may be profitable for you to reconsider pros and cons of securing your application with Keycloak. Perhaps there is a less complex tool or solution that can be used.

2.0 How To Authenticate Your Spring Boot Application with Keycloak

A detailed walkthrough on setting up Keycloak to secure a Spring Boot app

[Keycloak](#) is an open-source product developed by the RedHat Community which provides identity and access management solutions to modern applications. It requires little to no code to secure your applications and services.

Sections Covered

- Key concepts and benefits of Keycloak
- How to secure a Spring Boot application with Keycloak

This Document serve as base document for following concepts.

Spring Security and third-party Social Media Identity Providers such as Google, GitHub, etc.

Overview of Keycloak

The key benefit of Keycloak is that it provides a **Single-Sign-On (SSO)** which makes it easier for users to access different applications. It's a friendly approach for both users and developers.

As a developer, you don't have to worry about implementing login forms or storing credentials in a database. As a user, you don't have to remember various passwords to login to each application.

The same approach is valid for a **Single-Sign-Out**. If you log out of Keycloak, you log out of all applications that are using it.

Another nice feature is that you can add an Identity Provider such as Facebook, Google, GitHub, etc. to your application without any code changes.

You can also use existing OpenID Connect or SAML 2.0 Identity Providers for authentication.

The user management is handled through an Admin Console. You can manage various things like roles, sessions, cookies, etc.

Users can manage their own account's settings through an Account Management Console.

Set Up Keycloak

Set the server up.

There are different ways to get started. The quickest approach is to pull a Docker image.

We need to start the server by providing a port number. We also need a username and a password to be able to log in to the Admin Console.

I prefer creating a `docker-compose.yml` file since it'll be easier to start the server in the future. The process is as follows:

```
version: "3.2"
services:
  keycloak:
    image: jboss/keycloak:12.0.2
    ports:
      - "9900:8080"
    environment:
      - KEYCLOAK_USER=admin
      - KEYCLOAK_PASSWORD=admin123
```

Start the server by executing this docker command in your Terminal:

```
docker-compose up
```

Once the container has started, open your browser on <http://localhost:9900/auth/> and choose Administration Console. Enter the credentials you've just provided in the `environment` section.

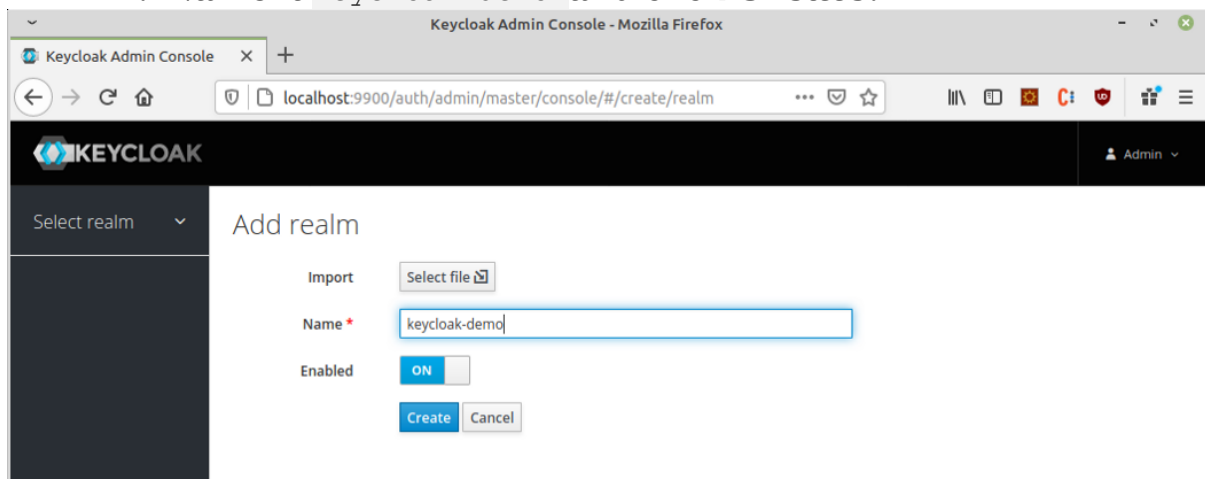
Create a realm

A realm manages a set of users, credentials, roles, and groups. The users you create in a realm belong to that realm. So, when they log in to Keycloak, they log in to the specified realm.

By default, you should be seeing the default **Master** realm right now.

1. Click on the upper left corner and choose **Add realm**.

2. Name it `keycloak-demo` and click **Create**.



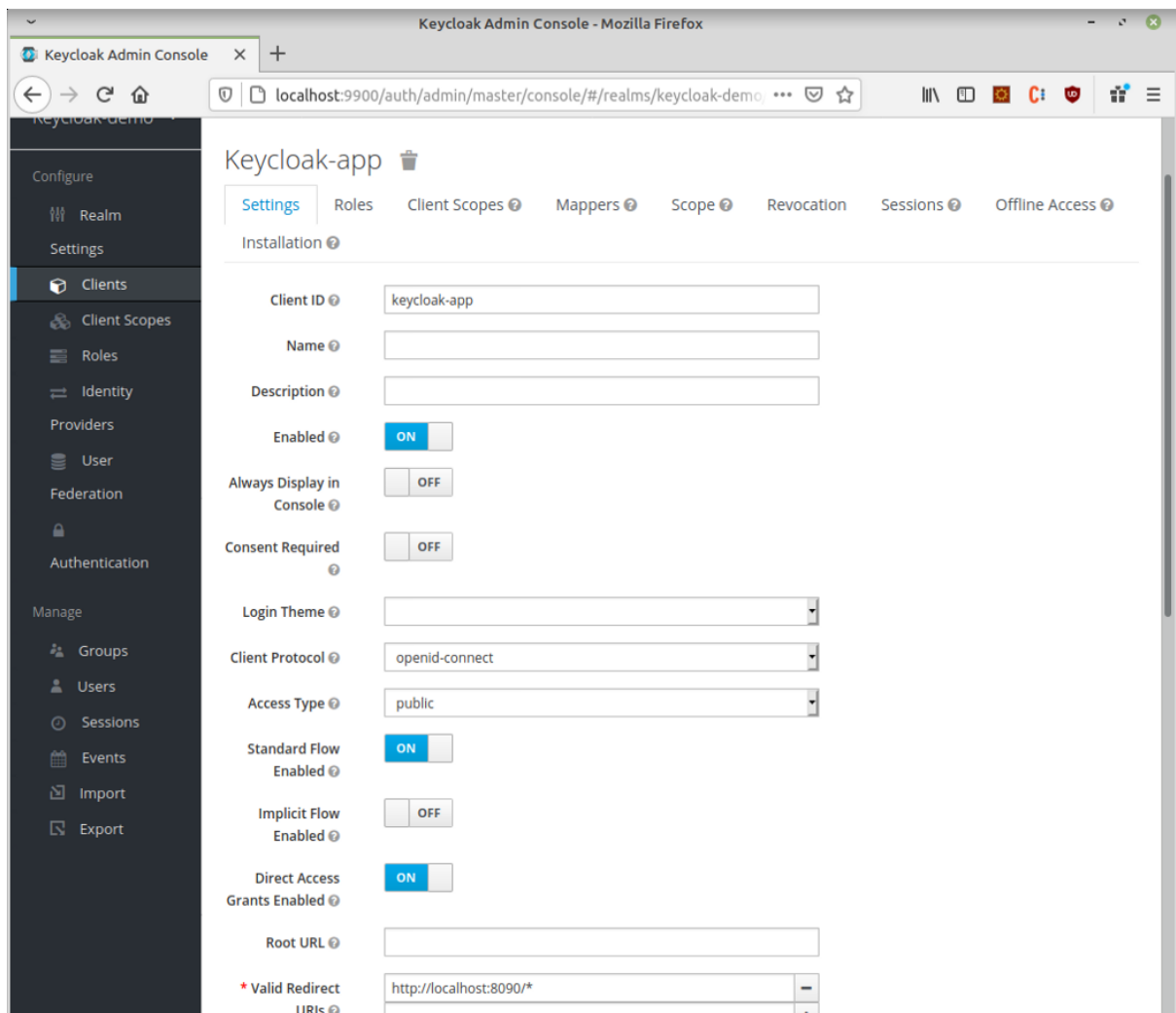
Creating a new realm

Create a client

Usually, clients are applications and services that want to use Keycloak for authentication. In our case, it'll be the Spring Boot app we're going to create shortly.

Let's add a client to our new realm.

1. Click the **Clients** menu item.
2. Select **Create** from the upper right corner.
3. Call it `keycloak-app`.
4. Save the client.
5. Click the **Settings** tab. Fill in the **Valid Redirect URL's** field. Our Spring Boot app will be running on <http://localhost:8090>. So, let's enter http://localhost:8090/*.



Creating a new client settings

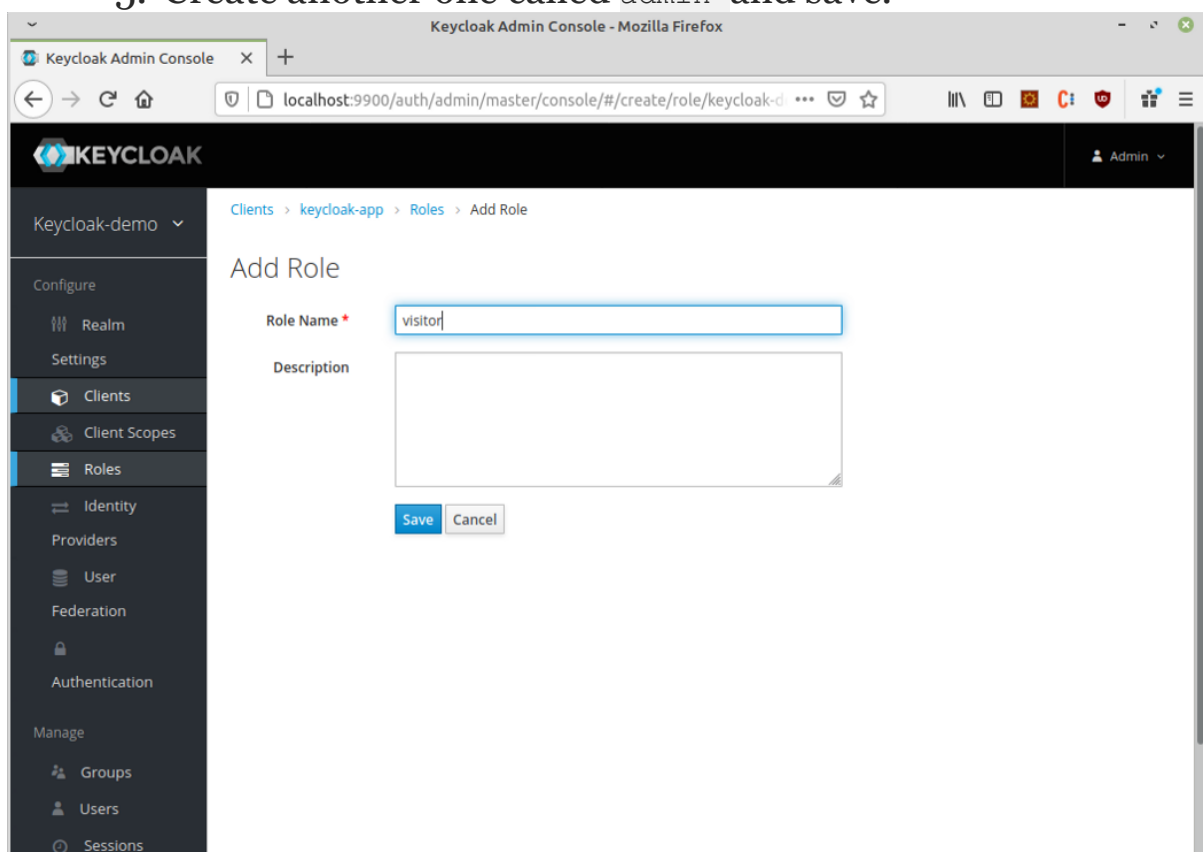
Create a role

Currently, there are three types of roles in Keycloak:

- realm roles — they're more like global roles that are specific for the realm.
- client roles — they're specific for each individual client.
- composite roles — as the name implies, they're roles that have multiple roles associated with them.

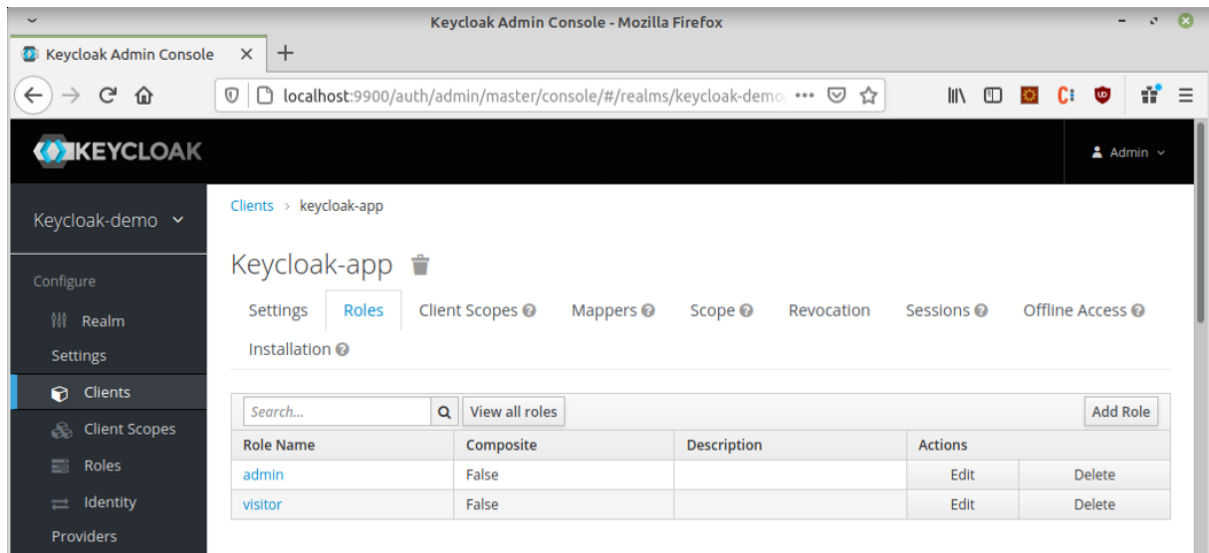
In our example, let's create a client role dedicated to our Spring Boot app:

1. Select the `keycloak-app` client from the **Clients** menu item.
2. Click the **Roles** tab.
3. Click **Add Role**.
4. Create a new role called `visitor` and save.
5. Create another one called `admin` and save.



Adding a new role for visitors

To verify that the roles have been successfully created, click the `keycloak-app` client, then the **Roles** tab and select **View all roles**:

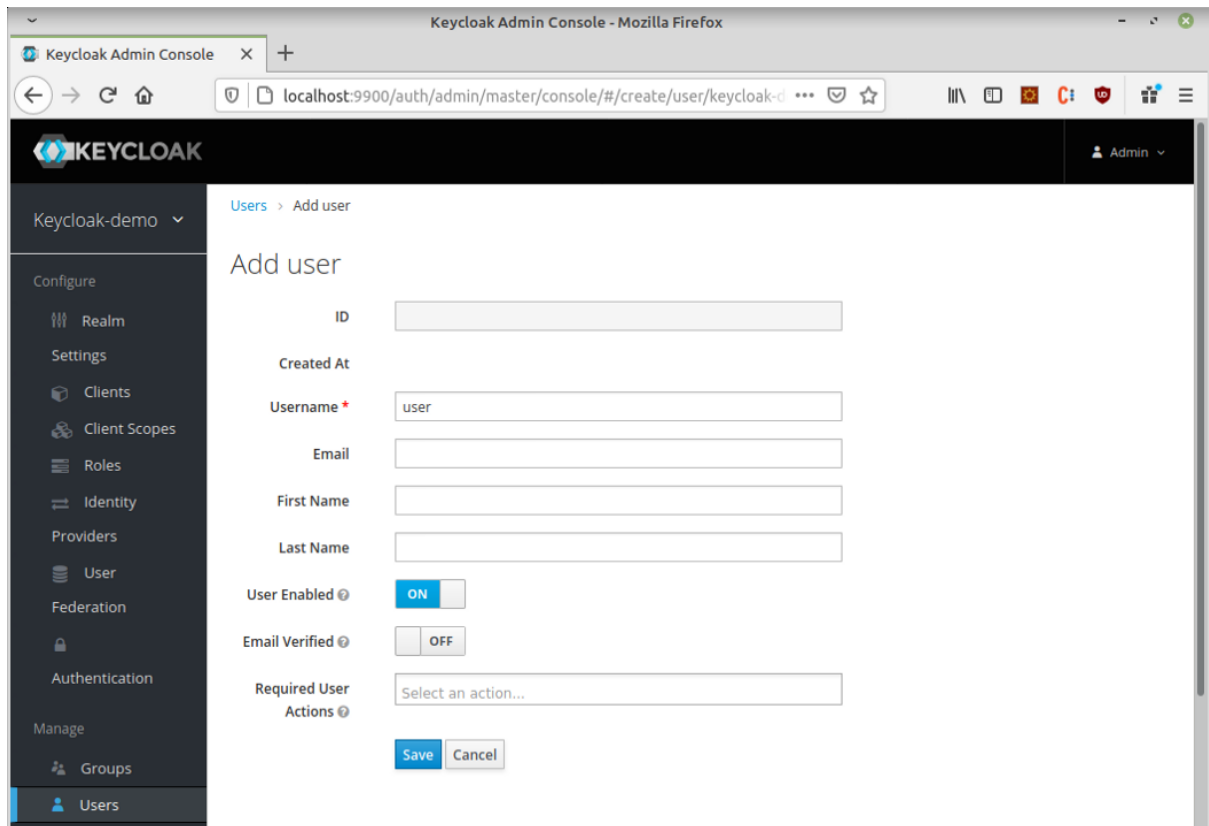


Client roles

Create users

Now we need to assign users to the roles we've just created.

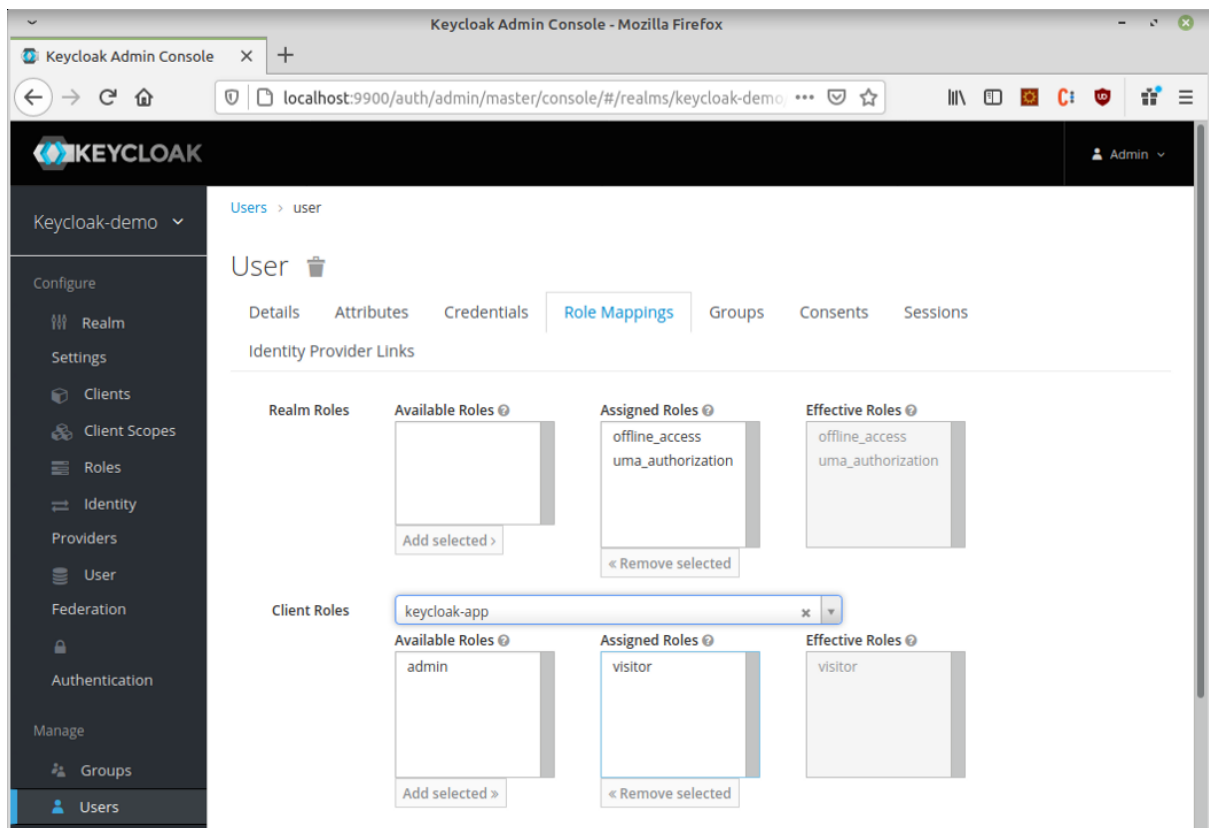
1. Click the **Users** menu item.
2. Select **Add User**.
3. Create a user called `user` and save it.



Adding a new user

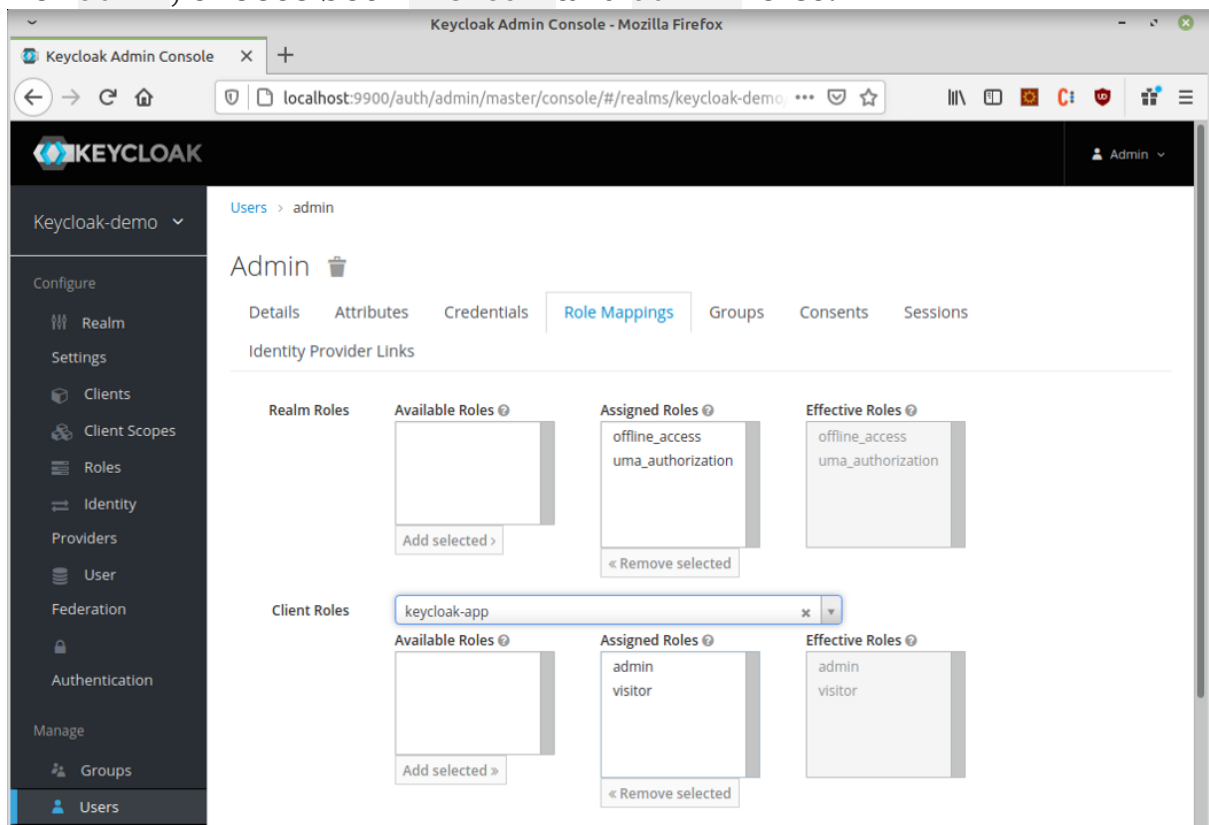
4. Add another user called `admin`.

5. To demonstrate how permissions work, let's assign different roles to the users. For `user`, go to the `Role Mappings` tab. From the **Client Roles** drop-down menu, select `keycloak-app` and assign `visitor`.



User client roles

For `admin`, choose both `visitor` and `admin` roles.



Admin client roles

6. To log in to the app, the users will need passwords. Go to the **Credentials** tab and set up a password. To avoid asking the user to change their password after the first login, switch the **Temporary** option off.

Create the Spring Boot App

We'll create a very simple app that will display custom text to the user based on their role. In our case, we'll distinguish `visitors` from `admins`.

I'm using Gradle to build this project. Create a New Gradle Project in your favorite IDE.

Get the dependencies

Open the `build.gradle` file and add the following dependencies:

Note that Keycloak's dependency includes the necessary Keycloak client adapters.

Create a controller

1. Create a new package, for example, `boot`.
2. Create a new Java class called `Controller` as follows:

The Spring Boot dependencies

This is a very simple controller that defines what message to display when you call the chosen URL path.

Create the main application

Create a new class called `App.java` as follows:
The main app class

This class will start our application.

Configure Keycloak

To integrate Keycloak authentication, we need to define a few settings.

1. Create an `application.properties` file under the `resources` folder.
2. Remember that we've defined **8090** as the valid redirection URL's port in Keycloak's Admin Console. Spring Boot runs at **8080** by default. To overwrite this behavior, define the `server.port` property like this:

```
server.port=8090
```

3. Add the Keycloak's settings by defining these properties:

You see that the realm property matches our realm's name and the resource equals our client's name. By setting `keycloak.use-resource-role-mapping` to true, we tell our app to operate with client roles.

4. Add the security constraints as follows:
Authentication config

What this means is that only users with an `admin` role will be able to access `visitor` and `admin` paths. Users with a `visitor` role will be allowed to view the `visitor` path only.

Test the Application

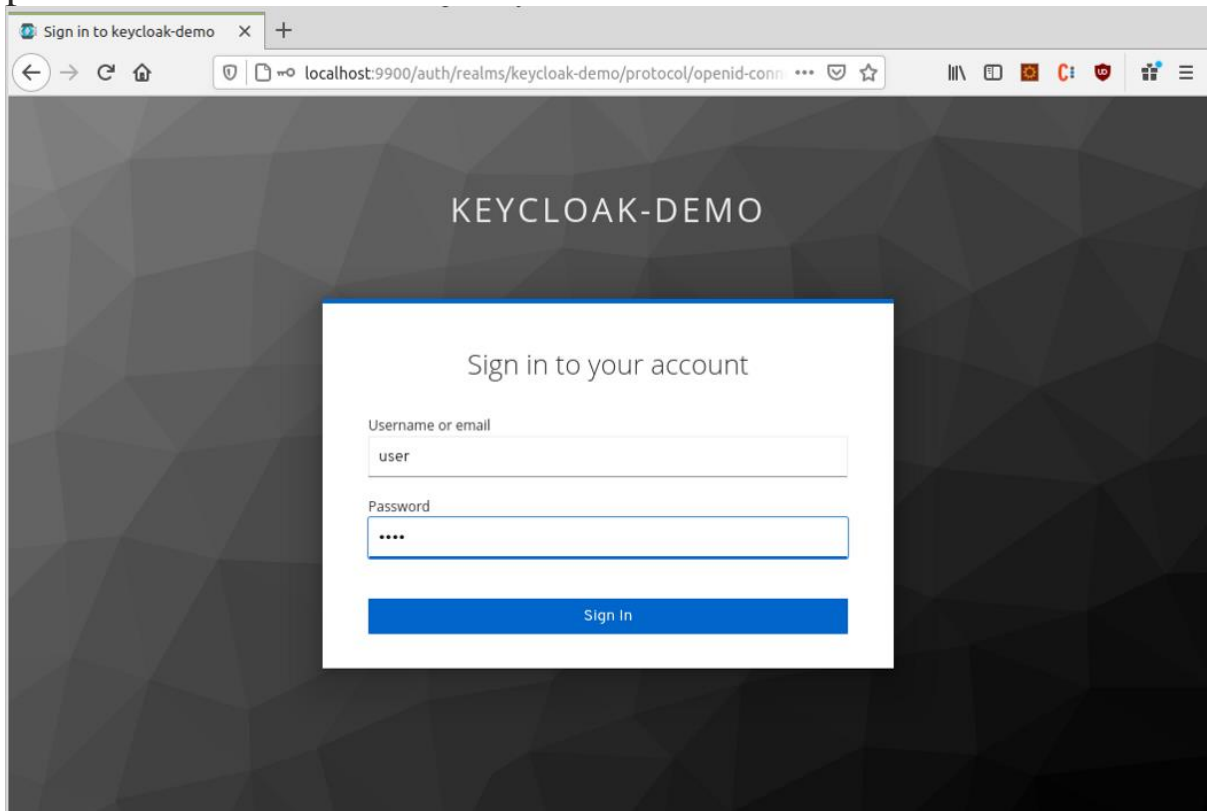
Start the app by executing this command in your Terminal:

```
gradle bootRun
```

Alternatively, right-click on `App.java` in your IDE and choose **Run App.main()**.

Type `localhost:8090/visitor` in your browser.

You'll be prompted to enter credentials. Enter `user`'s username and password.



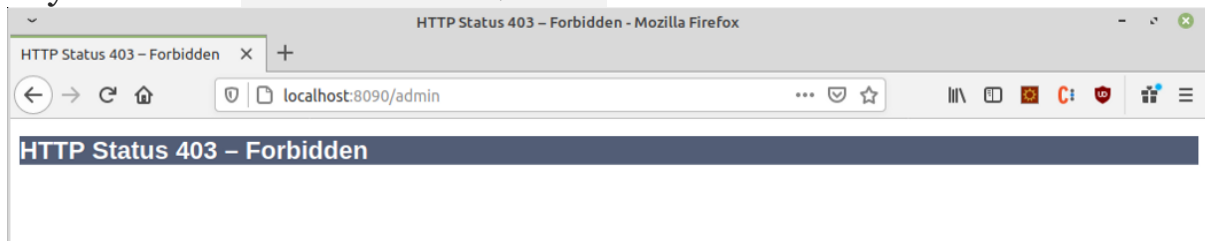
Login as user

You should see this message:



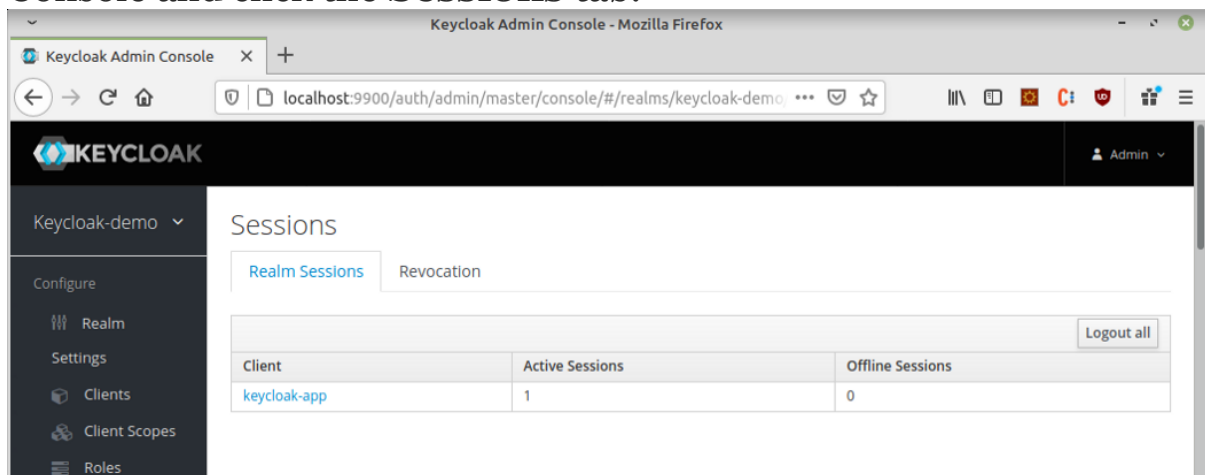
Successful login as a visitor

Try to access `localhost:8090/admin` :



Forbidden admin path as visitor

To manage the current user's session, go back to Keycloak's Admin Console and click the **Sessions** tab.



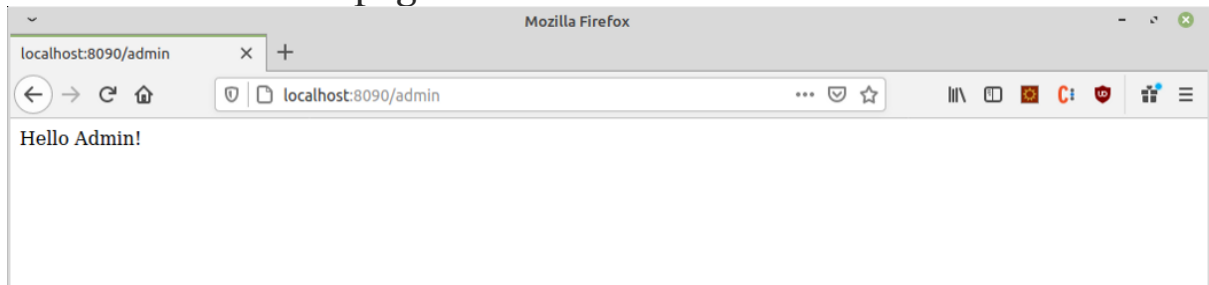
Logged on user sessions

You can force logout by clicking **Logout all**.

Note that you might need to restart the Spring Boot app since the browser stores the current session. Otherwise, you can use a new browser window.

Now, let's login as `admin`. Load the `localhost:8090/admin` URL and enter the `admin` user's credentials.

You should see this page:



Successful login as admin

Now let's try to access the `visitor` path:



Successful access to visitor path as admin

Perfect! The app is working as expected.

3.0 Secure Your Spring Boot App With Spring Security and GitHub Using Keycloak

Add a social media identity provider to securely authenticate your app

How to authenticate a Spring Boot app with [Keycloak](#).

If you're not familiar with Keycloak's basic concepts, recommended to refer KeyCloak Official guide.

Concepts to covered in the approach Document:

- How to integrate Spring Security with Keycloak
- Identity brokering concepts
- How to authenticate your app using a third-party social media identity provider (e.g. GitHub)

Add Spring Security to Your Spring Boot App

- We've developed a simple Spring Boot app running on `localhost:8090`.
- The Keycloak server is running on `localhost:9090` and authenticates users who want to log into the Spring Boot app.
- Users with a `visitor` role can access the `localhost:8090/visitor` page and users with an `admin` role can access both `localhost:8090/visitor` and `localhost:8090/admin`.

Refer simple Spring Boot app from the Setup done on Section - Spring Boot & Keycloak

Update the dependencies

Modify the `build.gradle` file and add dependencies for Spring Security. Your final file should look like this:

```
plugins
{
    id 'org.springframework.boot' version '2.3.3.RELEASE'
    id 'io.spring.dependency-management' version '1.0.8.RELEASE'
    id 'java'
}

group = 'com.keycloakdemo'
version = '0.0.1-SNAPSHOT'

repositories {
    mavenCentral()
}
```

```

dependencies {
    implementation 'org.keycloak:keycloak-spring-boot-starter'
    implementation 'org.springframework.boot:spring-boot-starter-security'
}
dependencyManagement {
    imports {
        mavenBom "org.keycloak.bom:keycloak-adapter-bom:11.0.2"
    }
}
test {
    useJUnitPlatform()
}

```

Configure Spring Security

Open the Spring Boot project in your IDE and add a new Java class called `KeycloakConfig.java`. Each method will be explained below:

```

// Defines
all
annotations
that are
needed to
integrate
Keycloak in
Spring
Security

@KeycloakConfiguration
class KeyCloakConfig extends KeycloakWebSecurityConfigurerAdapter {

    // Disable default role prefix ROLE_
    @Autowired
    public void configureGlobal(
        AuthenticationManagerBuilder auth) throws Exception {

        KeycloakAuthenticationProvider keycloakAuthenticationProvider
            = keycloakAuthenticationProvider();
    }
}

```



```

        keycloakAuthenticationProvider.setGrantedAuthoritiesMapper(
            new SimpleAuthorityMapper());
        auth.authenticationProvider(keycloakAuthenticationProvider);
    }

    // Use Spring Boot property files instead of default keycloak.json
    @Bean
    public KeycloakSpringBootConfigResolver keycloakConfigResolver() {
        return new KeycloakSpringBootConfigResolver();
    }

    // Register authentication strategy for public or confidential
    applications
    @Bean
    @Override
    protected SessionAuthenticationStrategy
    sessionAuthenticationStrategy() {
        return new RegisterSessionAuthenticationStrategy(
            new SessionRegistryImpl());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        super.configure(http);
        http.authorizeRequests()
            .antMatchers("/visitor").hasRole("visitor")
            .antMatchers("/admin").hasRole("admin")
            .anyRequest()
            .permitAll();
    }
}

```

Note that the `KeycloakConfiguration` annotation defines all annotations we need to integrate Keycloak in Spring Security.

The `KeycloakWebSecurityConfigurerAdapter` is a convenient base class for creating a [WebSecurityConfigurer](#) instance.

Roles are prefixed with `ROLE_` by default. Since we don't want this behavior, we've defined the `configureGlobal(AuthenticationManagerBuilder auth)` method to prevent default prefixing. This way, our roles' names will be the same as in the Keycloak admin console.

The `sessionAuthenticationStrategy()` method defines the authentication strategy.

How do you decide which strategy to use? There are two types of strategies:

- `RegisterSessionAuthenticationStrategy` is used for public or confidential applications.
- `NullAuthenticatedSessionStrategy` is used for bearer-only applications (login from the browser is not allowed).

Our app is public, so we've chosen `RegisterSessionAuthenticationStrategy`.

By default, the Spring Security adapter looks for a `keycloak.json` configuration file to read Keycloak's configuration. The `keycloakConfigResolver()` method tells the Spring Security adapter to look for a Spring Boot properties file instead. This way, we'll be able to use our own configuration.

Finally, we've defined the role permissions in our `configure(HttpSecurity http)` method. Now, we can delete or comment out the constraints from the `application.properties` file that we had configured in the previous tutorial.

The final `application.properties` should look like this:

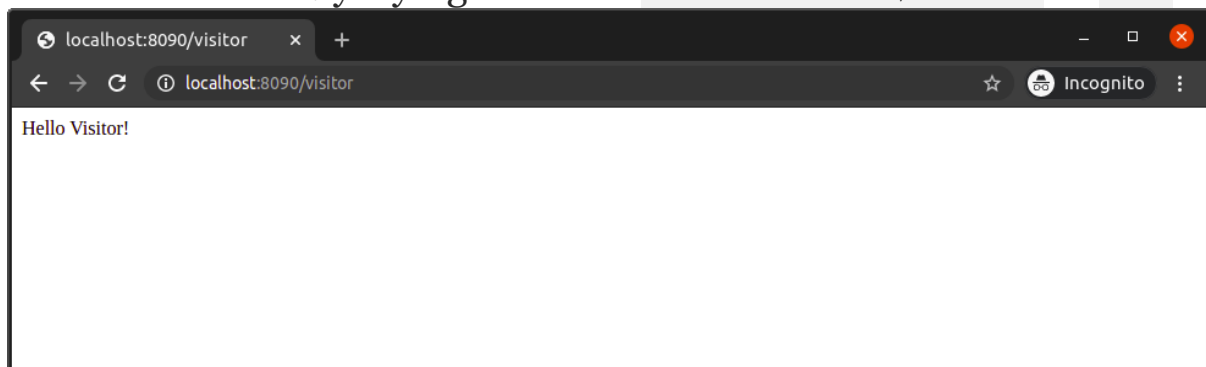
```
server.port=8090

keycloak.realm=keycloak-demo
keycloak.resource=keycloak-app
keycloak.auth-server-url=http://localhost:9900/auth
keycloak.ssl-required=external
keycloak.public-client=true
keycloak.use-resource-role-mappings=true
```

Testing the Application

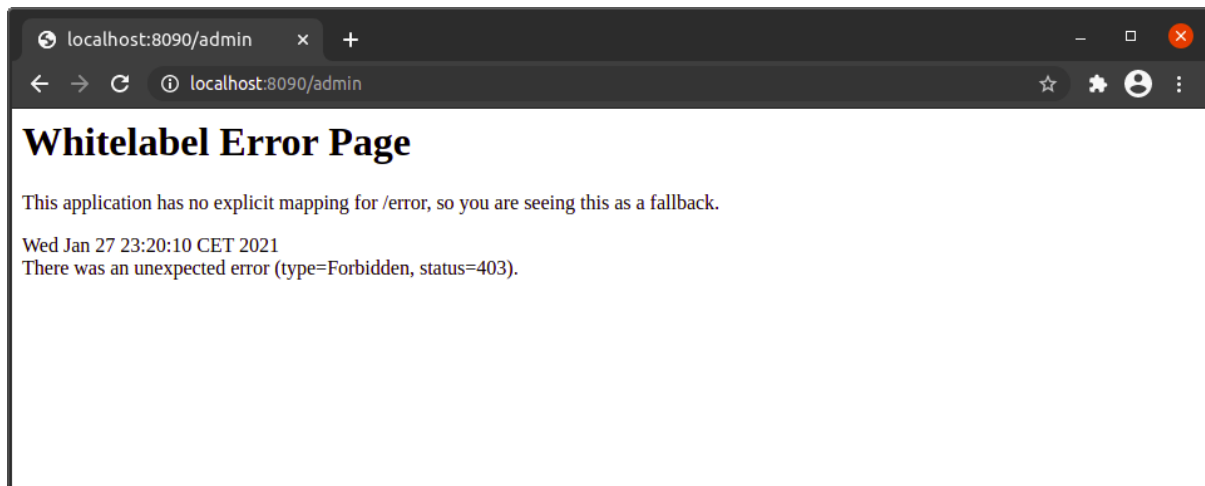
We expect the same behavior as in the previous article.

Let's check it out by trying to access `localhost:8090/visitor` as user:



Successful login as GitHub user with visitor role

Great! Now, let's try to access `localhost:8090/admin:`



Access denied to user trying to access the admin page

The application is working as expected. `user` is not allowed to view the `admin` page and received a 403 error.

Configure a Social Media Identity Provider

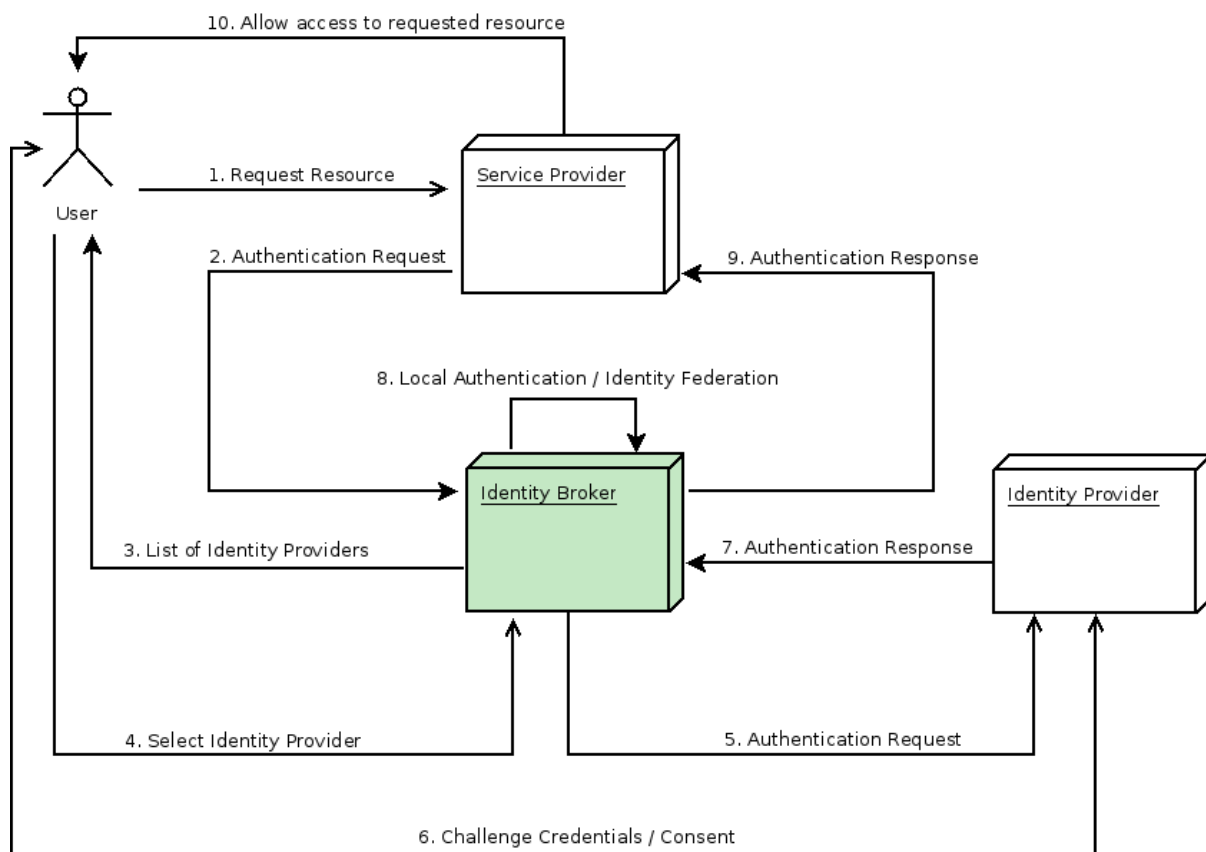
Basic concepts

First, let's understand the key concepts of identity brokering.

An identity provider is a service that is used to authenticate users. For example, GitHub, Google, Facebook. Keycloak itself is an identity provider too.

An identity broker is an intermediary service that is responsible for establishing a trusted connection between multiple service providers and different identity providers. For example, Keycloak can be used as an identity broker and authenticate an app through GitHub.

Take a look at the following diagram to follow the complete authentication process:



Source: [Keycloak docs](#)

Advantages of identity brokering

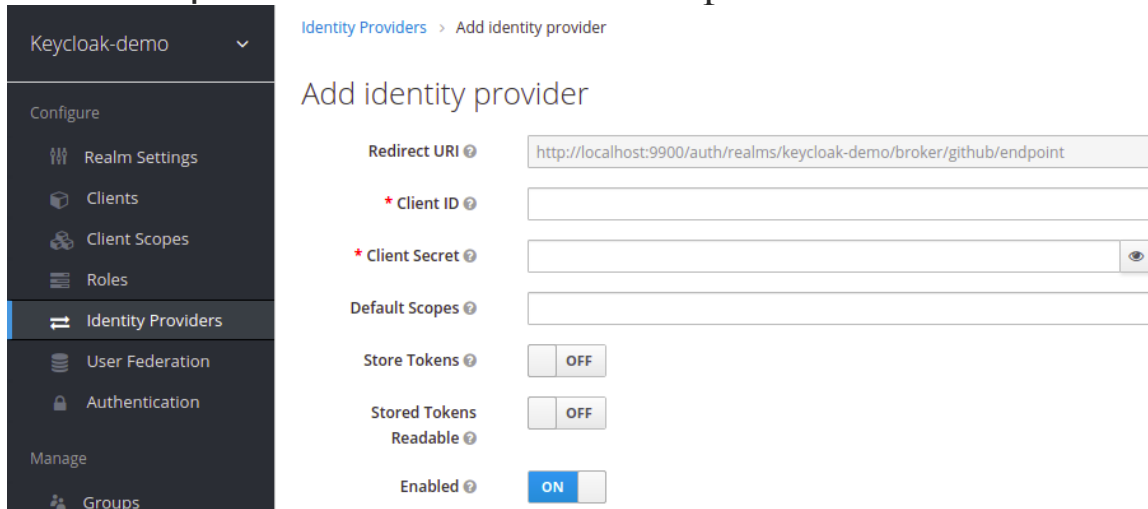
Using an identity broker simplifies the login process. The user is not required to register and remember any new credentials. Instead, they can simply authenticate through an existing account (e.g. Google).

Add an identity provider in Keycloak

In this demo, we're going to use GitHub as an identity provider.

1. Log into the Keycloak admin console with your admin credentials.

2. Click the “Identity Providers” menu item.
3. Click the “Add Identity Provider” button.
4. Select GitHub from the dropdown list.



Keycloak-demo

Configure


- Realm Settings
- Clients
- Client Scopes
- Roles
- Identity Providers**
- User Federation
- Authentication


Manage



- Groups


Identity Providers > Add identity provider


Add identity provider


Redirect URI 


* Client ID 

* Client Secret  

Default Scopes 

Store Tokens  ☐ OFF

Stored Tokens Readable  ☐ OFF

Enabled  ☒ ON

5. Take a note of the “Redirect URI” link.

We’ll fill out the rest of the required fields shortly.

Set up an OAuth app in GitHub

1. Assuming you have a GitHub account, navigate to Settings -> Developers menu or click the [following page](#).
2. Choose the “OAuth Apps” menu item.
3. Select “New OAuth App” from the upper right corner.
4. You can specify any name for your app.
5. Provide a URL that will lead to your Spring Boot app.

6. Paste the endpoint URL you copied from Keycloak to the “Authorization callback URL” field. Your settings should look like this:

Register a new OAuth application

Application name *

KeycloakDemo

Something users will recognize and trust.

Homepage URL *

http://localhost:8090/

The full URL to your application homepage.

Application description

Application description is optional

This is displayed to all users of your application.

Authorization callback URL *

http://localhost:9900/auth/realms/keycloak-demo/broker/github/end

Your application's callback URL. Read our [OAuth documentation](#) for more information.

Register application

Cancel

GitHub OAuth App registration

5. Click “Register application.”

6. Open the newly created app:

KeycloakDemo



kirshiyin89 owns this application.

Transfer ownership

You can list your application in the [GitHub Marketplace](#) so that other users can discover it.

List this application in the Marketplace

0 users

Revoke all user tokens

Client ID

c1c8c96fcfbf03c3ebf

Client secrets

Generate a new client secret

You need a client secret to authenticate as the application to the API.

7. Copy the client ID and paste it into the “Client ID” field in Keycloak’s admin console.

8. Generate a new client secret.

9. Paste it into the “Client Secret” field in Keycloak’s admin console.

Your identity provider settings should look like this:

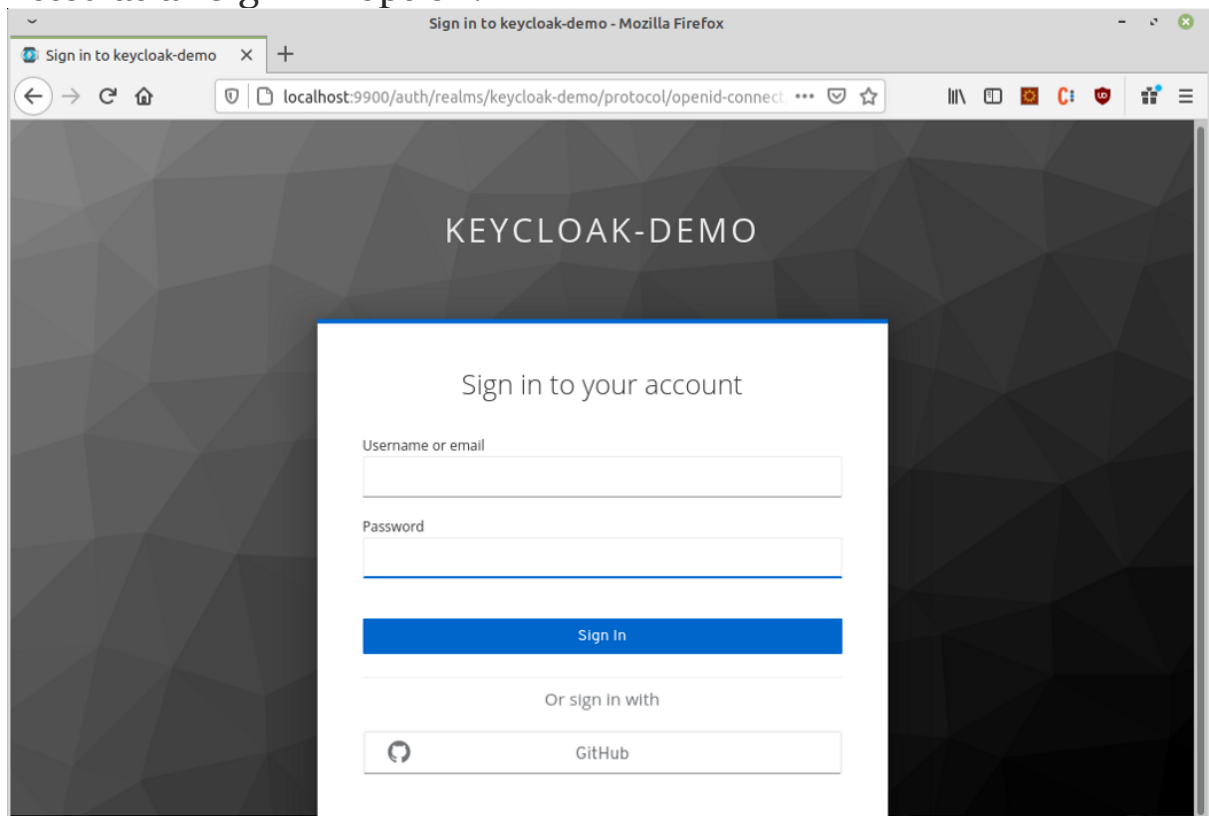
The screenshot shows the Keycloak Admin Console interface. On the left is a sidebar with a menu for 'Keycloak-demo' containing options like 'Configure', 'Realm Settings', 'Clients', 'Client Scopes', 'Roles', 'Identity Providers' (highlighted), 'User Federation', 'Authentication', 'Manage', and 'Groups'. The main content area is titled 'Add identity provider' and contains the following fields and controls:

- Redirect URI**:
- * Client ID**:
- * Client Secret**: (with an eye icon to toggle visibility)
- Default Scopes**:
- Store Tokens**: ☐ OFF
- Stored Tokens Readable**: ☐ OFF
- Enabled**: ☒ ON

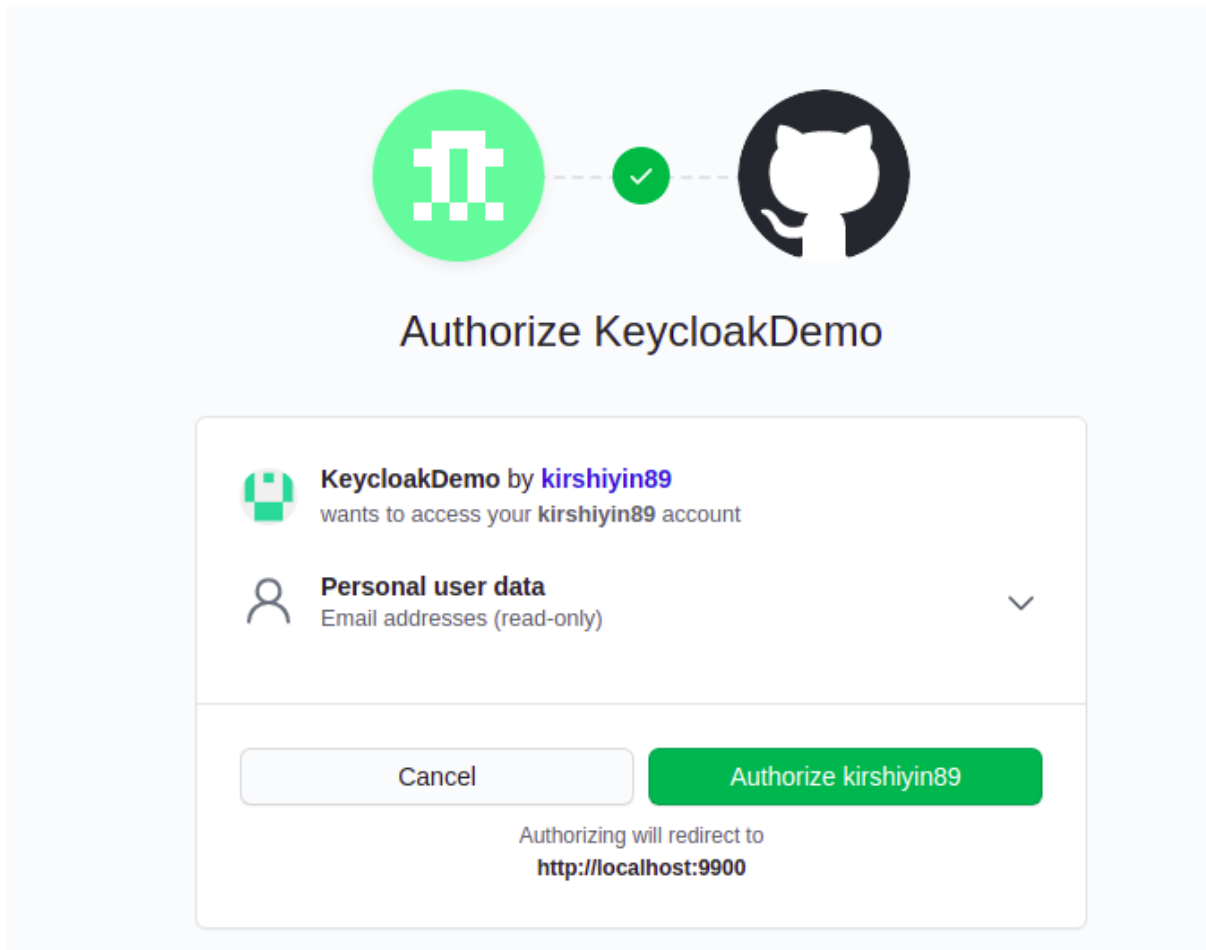
10. Save the settings.

Testing the GitHub Authorization

When you try to access your Spring Boot app, you should see GitHub listed as a “Sign In” option:



You will be prompted to authorize the app:



After successful authorization, you'll be redirected to Keycloak.

Note that your GitHub user hasn't been granted any roles for the Spring Boot app, so you won't be able to see any pages. To fix this, let's grant `visitor` permissions to the GitHub user.

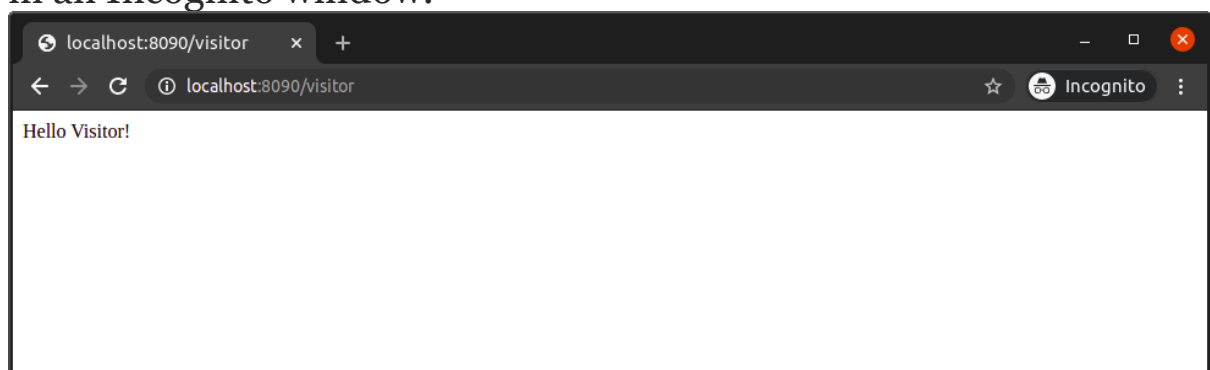
In the KeyCloak Demo Panel

Click the "Users" menu item. Find your user and click the "Role Mappings" tab. As we did in the previous tutorial, select the `keycloak-app` client from the client dropdown list. Assign the `visitor` role to the GitHub user:

The screenshot shows the Keycloak user management interface. On the left is a sidebar with a 'Keycloak-demo' header and a 'Configure' section containing 'Realm Settings', 'Clients', 'Client Scopes', 'Roles', 'Identity Providers', 'User Federation', and 'Authentication'. Below this is a 'Manage' section with 'Groups', 'Users' (highlighted), 'Sessions', 'Events', 'Import', and 'Export'. The main content area is titled 'Users > kirshiyin89' and shows the user 'Kirshiyin89' with a trash icon. There are tabs for 'Details', 'Attributes', 'Credentials', 'Role Mappings', 'Groups', 'Consents', 'Sessions', and 'Identity Provider Links'. The 'Details' tab is active, showing fields for ID, Created At, Username, Email, First Name, and Last Name. Below these are toggle switches for 'User Enabled' (ON) and 'Email Verified' (OFF), a 'Required User Actions' dropdown, and an 'Impersonate user' button. At the bottom are 'Save' and 'Cancel' buttons.

What if you wanted to automatically assign a role to the new users?
It's possible to achieve this by defining [default roles](#).

Let's try to access the `localhost:8090/visitor` path again. Force logout to end the current user's session. Alternatively, open the URL in an Incognito window:



Successful login as GitHub user

Awesome, it's working as expected!

Conclusion

The identity brokering concepts and seen how Spring Security works with Keycloak. We've also learned how to set up an identity provider, such as GitHub, in Keycloak.

References

https://www.keycloak.org/docs/latest/securing_apps/#_spring_boot_adapter

https://www.keycloak.org/docs/latest/server_admin/#social-identity-providers

4.0 Comparison of Identity Providers

We'll be comparing these products on the following functionalities

- Compliancy
- Extensibility
- Protocol support
- Pricing
- Federation support
- Integration support
- MFA support

Compliance

Depending on the field you're working in and the customer data that you will be storing in your identity and access management solution, you may have to be compliant with certain regulations. Often of the shelf SSO products are compliant with some of these regulations, lowering the burden on your development team since they'll no longer have to deal with it. This can be an important driver for deciding on a solution. We've created an overview for the subset of products we've selected.

Note that in this overview we're comparing compliance with regulations when the solution is hosted in a managed environment. Keycloak is exclusively a self-hosted solution so it cannot guarantee compliance. For that reason Keycloak has been excluded from the overview.

Ping Identity	Auth0	Amazon Cognito	Okta
ISO27001	ISO27001	ISO27001	ISO27001
SOC 2	ISO27018	ISO27017	ISO27017
CSA STAR	HIPAA BAA	ISO27018	ISO27018
	EU - US Privacy Shield Framework	ISO9001	SOC 2 Type II
	Gold CSA STAR	HIPAA	CSA STAR
	PCI-DSS	PCI-DSS	PCI-DSS
		SOC	SOX
			HIPAA

A few things immediately jump out here. Keycloak does not offer any support for compliance with regulations and instead leave it up the user of their platform. The next one that jumps out is that the cloud native platforms, Auth0, Amazon Cognito and Okta, offer the most with regards to compliance, including PCI-DSS which is a big one if you're in finances or handling card data and HIPAA which is important for the medical and insurance fields. The reason that the cloud platforms can do this is that the data is stored in environments that are controlled by the suppliers, so they can take all the required actions with regards to the data at rest and in transit through the system. The last thing that's important to note is that Ping Identity offers FAPI compliance, which is important when dealing with PSD2.

Extensibility

While of the shelf solutions try to offer a wide variety of functionality and connections to external systems for datasources, federation, etc. It's impossible for them to be able to catch all use cases. In order to resolve this, most platforms offer some way to create custom extensions. These custom extensions can hook into the solution to deal with use cases that cannot be dealt with the build in functionality. See the table below for an overview of the ways that the products we've selected can be extended.

	PingIdentity	Auth0	Keycloak	Amazon Cognito	Okta
Language	Java	JavaScript / NodeJS	Java	JavaScript / NodeJS	-
Hooks	Authentication Themes OTP Datasource connections & more	Pre and post authentication hooks in login flows	Authentication Themes OTP Role mappers Toke claim mappers & more	Pre and post authentication hooks in login flows	Pre and post authentication hooks in login flows

What stands out here is that the cloud native options have limited extensibility. They only offer some predefined hooks in flows where scripts can be executed or allow you to create limited custom integrations. The on premise options offer more extensibility, generally speaking in the form of SDKs that offer interfaces for certain functionality. Developers are then free to implement those and can load their code into the runtime of the server through dynamic class loading. The end result is that your own implementations show up as configurable options in administration screens.

Protocol Support

In the first post of the series we've already gone over the most prominent protocols that are used for SSO in today's landscape. Here we provide an overview of what protocols are supported by each product.

Protocol	Ping Identity	Auth0	Keycloak	Amazon Cognito	Okta
OAuth					

Protocol	Ping Identity	Auth0	Keycloak	Amazon Cognito	Okta
OAuth2.0					
OIDC					
SAML					

As expected, all platforms support all major protocols. One thing to note here is that only Auth0 still supports OAuth, however it's not really relevant because OAuth has been replaced by OAuth2.0 for a while now and shouldn't be used anymore.

Pricing

Ping Identity

Ping Identity requires you to buy licenses per product in their catalog. The prices per license are not publicly defined on their website. In order to know the costs you'll have to request a quote. Generally speaking the costs are based on the amount of expected users managed by the solution.

Auth0

Auth0 uses a freemium model. It's possible to use Auth0 for free if you have a relatively small user base (up to 7000 users) and don't need any of the advanced features like role management, customized emails and log retention. Next to the free plan there are two premium plans that do offer the advanced features. Here the pricing is defined based on the amount of active users. A user is considered active if they've logged in at least once for the month. The pricing for the pro plans is clearly defined on Auth0's website. Finally there's an enterprise variant. The enterprise variant offers a high SLA environment with enterprise grade support, compliancy with various standards and more extensive log retention. In order to know the pricing of the enterprise version you have to request a quote.

Keycloak

Keycloak is an open source solution and as such is free to use. The enterprise variant of Keycloak, Red Hat SSO, has its pricing based on the amount of CPU cores that are running the software. This is a common pricing model for Red Hat products.

Amazon Cognito

Amazon Cognito charges per monthly active user, similar to Auth0 a user is only considered active if they've logged in at least once that month. The first 50K monthly active users are free, and the first 50 monthly active users federated through federation are free. After these amounts there's a cost per monthly active user that scales down the more monthly active users you have. A similar scaling pricing model is used for the additional security features that Amazon Cognito offers. For the advanced security features there is no pricing difference between federated and regular users.

Okta

Okta has separate pricing for workforce identity and customer identity. In the case of workforce identity you pay a fixed price per user per month. Each feature is priced separately, and you can pick and choose the features that you need for your specific use case. Volume discounts become available at 5000+ users. Here the pricing is per user, regardless of if they're active or not. There is a trial option available for the workforce identity solution.

The Customer identity product has three editions: developer, one app and enterprise. The developer edition is free up to 1000 monthly active users, with the price slowly going up as you need more users. The developer edition only offers basic SSO functionality and doesn't have the option of installing addons. The developer edition also has a cap of 50000 monthly active users and only offers support through email. The one app solution and the enterprise solution offer options that allow for unlimited monthly active users and have the option of installing addons. Next to that they both have more extensive support options like the ability to call someone. The only difference between one app

and enterprise is that the one app edition has a limit of 5 OIDC clients, and the enterprise edition has no cap on the allowed clients. Pricing of the one app and enterprise editions is based on a quote, but a ballpark figure is given on Okta's website.

Addons for the customer identity product are things like multi-factor authentication, API access management, B2B integration, SSO integrations and more.

Conclusion

As we can see each product has its own unique way of pricing, but generally based on the same principles. The cloud variants, Auth0, Okta and Amazon Cognito, allow you to start for free and start charging per user after a certain threshold has been reached, after that the pricing per additional user is well defined. Ping Identity and Auth0 offer enterprise variants that require you to request a quote. Generally speaking though the costs are then again based on the amount of users. Keycloak stands out here since it's an open source product and can be used for free. The enterprise version of Keycloak, Red Hat SSO, also stands out because it's the only enterprise option that doesn't charge per monthly active user, but by CPU core instead. Okta is unique because it allows you to pick and chose the components that you need ensuring that you only pay for what you need.

Federation Support

In today's landscape it's common to allow consumers to log in using one of their already existing accounts like their Google, Facebook or Active Directory account. This is called user federation. With federation support we're taking a look at to which extend the products support logging in with already existing accounts.

Feature	Ping Identity	Auth0	Keycloak	Amazon Cognito	Okta
Out of the box connections					
Custom connections via portals					
Custom connections via SDK implementations					

	Ping Identity	Auth0	Keycloak	Amazon Cognito	Okta
Vendor supported SDKs	Java .NET PHP	JavaScript iOS Android PHP .NET NodeJS Python	Java JavaScript	JavaScript React Native iOS	JavaScript Android iOS Python Java .NET NodeJS PHP GO React Native Vue.JS Angular

Feature	Ping Identity	Auth0	Keycloak	Amazon Cognito	Okta
Marketplace with additional out of the box connections					

All products offer support for federation through out of the box connections. These are available for all the major players in the industry, like Google, Facebook, Twitter and LDAP. They also all offer support for configuring custom connections to identity providers that implement the OAuth2.0, OIDC or SAML protocols. The on-premise solutions, Ping Identity and Keycloak, also offer the option to build custom connections to other systems you want to federate with even if these systems don't implement one of the SSO protocols through SDKs. Finally, Ping Identity and Okta also offer a marketplace with connections to various identity providers allowing you to offer the option to log in with for example AWS, Sharepoint, Slack and more.

Integration Support

Setting up the SSO platform is only half of the journey. The next step is integrating your own applications with the SSO platform. We see integration support as "how easy is it to integrate the platform into my own application". We think this is an important indicator when selecting a SSO product since good integration support can save you loads of time and will keep your developers happy. We've taken a look at SDKs provided by each product.

While the language support of SDKs may seem limited, it's important to

note that all products implement the SSO protocols mentioned and there are loads of generic libraries for OAuth2.0/OIDC and SAML out there for each language. All of these should also work with these products. All products are also documented reasonably well and provide enough support to integrate easily.

MFA Support

The final indicator we'll be looking at is multi-factor authentication support offered by the products. People expect to have the option to use multi-factor authentication nowadays and regulations require it to be in place. Because of that it's important to know what options are available when selecting a product.

Ping Identity	Auth0	Keycloak	Amazon Cognito	Okta
OTP Marketplace Extendable with custom implementations	OTP via Google Authenticator or similar SMS verification Email verification Push notifications DUO Security	OTP via Google Authenticator or similar Extendable with custom implementations	OTP via Google Authenticator or similar SMS verification	OTP via Google Authenticator or similar SMS verification Email verification Push notifications FIDO2 (WebAuthn)

All products support MFA through one time passwords via google authenticator or similar apps. Things to note here are that Keycloak and Ping Identity allow you to write custom MFA implementations. Ping Identity also offers a marketplace with many MFA implementations. This includes features like SMS, push notifications, passwordless logins and more. Okta offers all of these features out of the box which makes them more easily accessible.

In this blog we've looked at five SSO solutions and we've compared them on some key functionalities that are relevant when selecting a product to integrate with. With this we also hope that we've given you some criteria that you can use to determine if a product is a good fit for you even if you're not looking at SSO solutions in our list.

TBD :

In [products in action](#) as we make a deep dive into some live SSO use cases that will tackled using these products.