

# Quarkus - Protecting Web Applications Using OpenID Connect

This guide demonstrates how to use the OpenID Connect Extension to protect your application using Quarkus, where authentication and authorization are based on tokens issued by OpenId Connect and OAuth 2.0 compliant Authorization Servers such as [Keycloak](#).

The extension allows you to easily enable authentication to your web application based on the Authorization Code Flow so that your users are redirected to a OpenID Connect Provider (e.g.: Keycloak) to authenticate and, once the authentication is complete, return back to your application.

We are going to give you a guideline on how to use OpenId Connect to authenticate users using the Quarkus OpenID Connect Extension.

## Prerequisites

To complete this guide, you need:

- less than 15 minutes
- an IDE
- JDK 1.8+ installed with `JAVA_HOME` configured appropriately
- Apache Maven 3.5.3+
- [jq tool](#)
- Docker

## Architecture

In this example, we build a very simple web application with a single page:

- `/index.html`

This page is protected and can only be accessed by authenticated users.

## Solution

We recommend that you follow the instructions in the next sections and create the application step by step. However, you can go right to the completed example.

Clone the Git repository: `git clone https://github.com/quarkusio/quarkus-quickstarts.git`, or download an [archive](#).

The solution is located in the [security-openid-connect-web-authentication-quickstart](#) directory.

## Creating the Maven Project

First, we need a new project. Create a new project with the following command:

```
mvn io.quarkus:quarkus-maven-plugin:1.3.0.Alpha1:create \
  -DprojectId=org.acme \
  -DprojectArtifactId=security-openid-connect-web-authentication-quickstart \
  -Dextensions="oidc"
cd security-openid-connect-web-authentication-quickstart
```

## Configuring the application

The OpenID Connect extension allows you to define the configuration using the `application.properties` file which should be located at the `src/main/resources` directory.

### Configuring using the `application.properties` file

```
quarkus.oidc.auth-server-
url=http://localhost:8180/auth/realms/quarkus
quarkus.oidc.client-id=frontend
quarkus.oidc.application-type=web-app
quarkus.http.auth.permission.authenticated.paths=/*
quarkus.http.auth.permission.authenticated.policy=authenticated
```

This is the simplest configuration you can have when enabling authentication to your application.

The `quarkus.oidc.client-id` property references the `client_id` issued by the OpenID Connect Provider and, in this case, the application is a public client (no client secret is defined).

The `quarkus.oidc.application-type` property is set to `web-app` in order to tell Quarkus that you want to enable the OpenID Connect Authorization Code Flow, so that your users are redirected to the OpenID Connect Provider to authenticate.

For last, the `quarkus.http.auth.permission.authenticated` permission is set to tell Quarkus about the paths you want to protect. In this case, all paths are being protected by a policy that ensures that only `authenticated` users are allowed to access. For more details check [Security Guide](#).

# Starting and Configuring the Keycloak Server

To start a Keycloak Server you can use Docker and just run the following command:

```
docker run --name keycloak -e KEYCLOAK_USER=admin -e
KEYCLOAK_PASSWORD=admin -p 8180:8080
quay.io/keycloak/keycloak:8.0.1
```

You should be able to access your Keycloak Server at [localhost:8180/auth](http://localhost:8180/auth).

Log in as the **admin** user to access the Keycloak Administration Console. Username should be **admin** and password **admin**.

Import the [realm configuration file](#) to create a new realm. For more details, see the Keycloak documentation about how to [create a new realm](#).

## Running and Using the Application

### Running in Developer Mode

To run the microservice in dev mode, use `./mvnw clean compile quarkus:dev`.

### Running in JVM Mode

When you're done playing with "dev-mode" you can run it as a standard Java application.

First compile it:

```
./mvnw package
```

Then run it:

```
java -jar ./target/security-openid-connect-web-authentication-
quickstart-runner.jar
```

### Running in Native Mode

This same demo can be compiled into native code: no modifications required.

This implies that you no longer need to install a JVM on your production environment, as the runtime technology is included in the produced binary, and optimized to run with minimal resource overhead.

Compilation will take a bit longer, so this step is disabled by default; let's build again by enabling the **native** profile:

```
./mvnw package -Pnative
```

After getting a cup of coffee, you'll be able to run this binary directly:

```
./target/security-openid-connect-web-authentication-quickstart-runner
```

## Testing the Application

To test the application, you should open your browser and access the following URL:

- <http://localhost:8080>

If everything is working as expected, you should be redirected to the Keycloak server to authenticate.

In order to authenticate to the application you should type the following credentials when at the Keycloak login page:

- Username: **alice**
- Password: **alice**

After clicking the **Login** button you should be redirected back to the application.

## Logout

The extension only supports logout based on the expiration time of the ID Token issued by the OpenID Connect Provider. When the token expires, users are redirected to the OpenID Connect Provider again to authenticate. If the session at the OpenID Connect Provider is still active, users are automatically re-authenticated without having to provide their credentials again.

## Configuration Reference

 Configuration property fixed at build time -  Configuration property overridable at runtime

Configuration property	Type	Default
 <code>quarkus.oidc.enabled</code> If the OIDC extension is enabled.	boolean	<code>true</code>
 <code>quarkus.oidc.application-type</code> The application type, which can be one of the following values from enum <code>ApplicationType</code> .	<code>web-app</code> , <code>service</code>	<code>service</code>

<p> <code>quarkus.oidc.connection-delay</code></p> <p>The maximum amount of time the adapter will try connecting to the currently unavailable OIDC server for. For example, setting it to '20S' will let the adapter keep requesting the connection for up to 20 seconds.</p>	<p>Duration </p>	
<p> <code>quarkus.oidc.auth-server-url</code></p> <p>The base URL of the OpenID Connect (OIDC) server, for example, 'https://host:port/auth'. All the other OIDC server page and service URLs are derived from this URL. Note if you work with Keycloak OIDC server, make sure the base URL is in the following format: 'https://host:port/auth/realms/{realm}' where '{realm}' has to be replaced by the name of the Keycloak realm.</p>	<p>string</p>	
<p> <code>quarkus.oidc.introspection-path</code></p> <p>Relative path of the RFC7662 introspection service.</p>	<p>string</p>	
<p> <code>quarkus.oidc.jwks-path</code></p> <p>Relative path of the OIDC service returning a JWK set.</p>	<p>string</p>	
<p> <code>quarkus.oidc.public-key</code></p> <p>Public key for the local JWT token verification.</p>	<p>string</p>	
<p> <code>quarkus.oidc.client-id</code></p> <p>The client-id of the application. Each application has a client-id that is used to identify the application</p>	<p>string</p>	
<p> <code>quarkus.oidc.roles.role-claim-path</code></p> <p>Path to the claim containing an array of groups. It starts from the top level JWT JSON object and can contain multiple segments where each segment represents a JSON object name only, example: "realm/groups". This property can be used if a token has no 'groups' claim but has the groups set in a different claim.</p>	<p>string</p>	
<p> <code>quarkus.oidc.roles.role-claim-separator</code></p> <p>Separator for splitting a string which may contain multiple group values. It will only be used if the "role-claim-path" property points to a custom claim whose value is a string. A single space will be used by default because the standard 'scope' claim may contain a space separated sequence.</p>	<p>string</p>	

 <code>quarkus.oidc.token.issuer</code>		
Expected issuer 'iss' claim value	string	
 <code>quarkus.oidc.token.audience</code>		
Expected audience <code>aud</code> claim value which may be a string or an array of strings	list of string	
 <code>quarkus.oidc.token.principal-claim</code>		
Name of the claim which contains a principal name. By default, the 'upn', 'preferred_username' and <code>sub</code> claims are checked.	string	
 <code>quarkus.oidc.credentials.secret</code>		
The client secret	string	
 <code>quarkus.oidc.authentication.redirect-path</code>		
Relative path for calculating a "redirect_uri" parameter. If set it will be appended to the request URI's host and port, otherwise the complete request URI will be used. It has to start from the forward slash, for example: "/service"	string	
 <code>quarkus.oidc.authentication.scopes</code>		
List of scopes	list of string	
<b>Additional named tenants</b>	<b>Type</b>	<b>Default</b>
 <code>quarkus.oidc."tenant".connection-delay</code>		
The maximum amount of time the adapter will try connecting to the currently unavailable OIDC server for. For example, setting it to '20S' will let the adapter keep requesting the connection for up to 20 seconds.	Duration 	
 <code>quarkus.oidc."tenant".auth-server-url</code>		
The base URL of the OpenID Connect (OIDC) server, for example, 'https://host:port/auth'. All the other OIDC server page and service URLs are derived from this URL. Note if you work with Keycloak OIDC server, make sure the base URL is in the following format: 'https://host:port/auth/realms/{realm}' where '{realm}' has to be replaced by the name of the Keycloak realm.	string	
 <code>quarkus.oidc."tenant".introspection-path</code>		
Relative path of the RFC7662 introspection service.	string	

 <code>quarkus.oidc."tenant".jwks-path</code>		
Relative path of the OIDC service returning a JWK set.	string	
 <code>quarkus.oidc."tenant".public-key</code>		
Public key for the local JWT token verification.	string	
 <code>quarkus.oidc."tenant".client-id</code>		
The client-id of the application. Each application has a client-id that is used to identify the application	string	
 <code>quarkus.oidc."tenant".roles.role-claim-path</code>		
Path to the claim containing an array of groups. It starts from the top level JWT JSON object and can contain multiple segments where each segment represents a JSON object name only, example: "realm/groups". This property can be used if a token has no 'groups' claim but has the groups set in a different claim.	string	
 <code>quarkus.oidc."tenant".roles.role-claim-separator</code>		
Separator for splitting a string which may contain multiple group values. It will only be used if the "role-claim-path" property points to a custom claim whose value is a string. A single space will be used by default because the standard 'scope' claim may contain a space separated sequence.	string	
 <code>quarkus.oidc."tenant".token.issuer</code>		
Expected issuer 'iss' claim value	string	
 <code>quarkus.oidc."tenant".token.audience</code>		
Expected audience <code>aud</code> claim value which may be a string or an array of strings	list of string	
 <code>quarkus.oidc."tenant".token.principal-claim</code>		
Name of the claim which contains a principal name. By default, the 'upn', 'preferred_username' and <code>sub</code> claims are checked.	string	
 <code>quarkus.oidc."tenant".credentials.secret</code>		
The client secret	string	

 <code>quarkus.oidc."tenant".authentication.redirect-path</code> Relative path for calculating a "redirect_uri" parameter. If set it will be appended to the request URI's host and port, otherwise the complete request URI will be used. It has to start from the forward slash, for example: "/service"	string	
 <code>quarkus.oidc."tenant".authentication.scopes</code> List of scopes	list of string	

#### *About the Duration format*

The format for durations uses the standard `java.time.Duration` format. You can learn more about it in the [Duration#parse\(\) javadoc](#).



You can also provide duration values starting with a number. In this case, if the value consists only of a number, the converter treats the value as seconds. Otherwise, `PT` is implicitly prepended to the value to obtain a standard `java.time.Duration` format.

## References

- [Keycloak Documentation](#)
- [OpenID Connect](#)
- [JSON Web Token](#)