Quarkus - Using OpenID Connect to Protect Service Applications using Bearer Token Authorization

This guide demonstrates how to use Quarkus OpenID Connect Extension to protect your JAX-RS applications using Bearer Token Authorization where Bearer Tokens are issued by OpenId Connect and OAuth 2.0 compliant Authorization Servers such as Keycloak.

Bearer Token Authorization is the process of authorizing HTTP requests based on the existence and validity of a Bearer Token which provides valuable information to determine the subject of the call as well as whether or not an HTTP resource can be accessed.

Please read the Using OpenID Connect to Protect Web Applications guide if you need to authenticate and authorize the users using OpenId Connect Authorization Code Flow.

Please read the Using OpenID Connect Multi-Tenancy guide how to support multiple tenants.

Prerequisites

To complete this guide, you need:

- less than 15 minutes

- an IDE

- JDK 1.8+ installed with `JAVA_HOME` configured appropriately

- Apache Maven 3.6.3

- jq tool

- Docker

Architecture

In this example, we build a very simple microservice which offers three endpoints:

- `/api/users/me`

- `/api/admin`

These endpoints are protected and can only be accessed if a client is sending a bearer token along with the request, which must be valid (e.g.: signature, expiration and audience) and trusted by the microservice.

The bearer token is issued by a Keycloak Server and represents the subject to which the token was

issued for. For being an OAuth 2.0 Authorization Server, the token also references the client acting on behalf of the user.

The `/api/users/me` endpoint can be accessed by any user with a valid token. As a response, it returns a JSON document with details about the user where these details are obtained from the information carried on the token.

The `/api/admin` endpoint is protected with RBAC (Role-Based Access Control) where only users granted with the `admin` role can access. At this endpoint, we use the `@RolesAllowed` annotation to declaratively enforce the access constraint.

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-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:1.7.0.CR1:create \
    -DprojectGroupId=org.acme \
    -DprojectArtifactId=security-openid-connect-quickstart \
    -Dextensions="oidc, resteasy-jsonb"
cd security-openid-connect-quickstart
```

This command generates a Maven project, importing the `keycloak` extension which is an implementation of a Keycloak Adapter for Quarkus applications and provides all the necessary capabilities to integrate with a Keycloak Server and perform bearer token authorization.

If you already have your Quarkus project configured, you can add the `oidc` extension to your project by running the following command in your project base directory:

```
./mvnw quarkus:add-extension -Dextensions="oidc"
```

This will add the following to your `pom.xml`:

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-oidc</artifactId>
</dependency>
```

Writing the application

Let's start by implementing the `/api/users/me` endpoint. As you can see from the source code below it is just a regular JAX-RS resource:

```java
package org.acme.security.openid.connect;

import javax.annotation.security.RolesAllowed;
import javax.inject.Inject;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

import org.jboss.resteasy.annotations.cache.NoCache;
import io.quarkus.security.identity.SecurityIdentity;

@Path("/api/users")
public class UsersResource {

    @Inject
    SecurityIdentity securityIdentity;

    @GET
    @Path("/me")
    @RolesAllowed("user")
    @Produces(MediaType.APPLICATION_JSON)
    @NoCache
    public User me() {
        return new User(securityIdentity);
    }

    public static class User {

        private final String userName;

        User(SecurityIdentity securityIdentity) {
            this.userName =
securityIdentity.getPrincipal().getName();
        }

        public String getUserName() {
            return userName;
        }
    }
}
```

The source code for the `/api/admin` endpoint is also very simple. The main difference here is that we are using a `@RolesAllowed` annotation to make sure that only users granted with the `admin` role can access the endpoint:

```
package org.acme.security.openid.connect;

import javax.annotation.security.RolesAllowed;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/api/admin")
public class AdminResource {

    @GET
    @RolesAllowed("admin")
    @Produces(MediaType.TEXT_PLAIN)
    public String admin() {
        return "granted";
    }
}
```

Injection of the `SecurityIdentity` is supported in both `@RequestScoped` and `@ApplicationScoped` contexts.

Accessing JWT claims

If you need to access `JsonWebToken` claims, you may simply inject the token itself:

```
package org.acme.security.openid.connect;

import org.eclipse.microprofile.jwt.JsonWebToken;

import javax.annotation.security.RolesAllowed;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/api/admin")
public class AdminResource {

    @Inject
    JsonWebToken jwt;

    @GET
    @RolesAllowed("admin")
    @Produces(MediaType.TEXT_PLAIN)
    public String admin() {
        return "Access for subject " + jwt.getSubject() + " is
granted";
    }
}
```

Injection of the `JsonWebToken` is supported in both `@RequestScoped` and `@ApplicationScoped` contexts.

Configuring the application

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

Configuring using the application.properties file

🔒 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default
🔒 `quarkus.oidc.enabled` If the OIDC extension is enabled.	boolean	`true`

`quarkus.oidc.tenant-id` A unique tenant identifier. It must be set by `TenantConfigResolver` providers which resolve the tenant configuration dynamically and is optional in all other cases.	string	
`quarkus.oidc.tenant-enabled` If this tenant configuration is enabled.	boolean	`true`
`quarkus.oidc.application-type` The application type, which can be one of the following values from enum `ApplicationType`.	`web-app`, `service`	`service`
`quarkus.oidc.auth-server-url` The base URL of the OpenID Connect (OIDC) server, for example, 'https://host:port/auth'. OIDC discovery endpoint will be called by default by appending a '.well-known/openid-configuration' path to 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	
`quarkus.oidc.discovery-enabled` Enables OIDC discovery. If the discovery is disabled then the following properties must be configured: - 'authorization-path' and 'token-path' for the 'web-app' applications - 'jwks-path' or 'introspection-path' for both the 'web-app' and 'service' applications 'web-app' applications may also have 'user-info-path' and 'end-session-path' properties configured.	boolean	`true`
`quarkus.oidc.authorization-path` Relative path of the OIDC authorization endpoint which authenticates the users. This property must be set for the 'web-app' applications if OIDC discovery is disabled. This property will be ignored if the discovery is enabled.	string	
`quarkus.oidc.token-path` Relative path of the OIDC token endpoint which issues ID, access and refresh tokens. This property must be set for the 'web-app' applications if OIDC discovery is disabled. This property will be ignored if the discovery is enabled.	string	

`quarkus.oidc.user-info-path` Relative path of the OIDC userinfo endpoint. This property must only be set for the 'web-app' applications if OIDC discovery is disabled and 'authentication.user-info-required' property is enabled. This property will be ignored if the discovery is enabled.	string	
`quarkus.oidc.introspection-path` Relative path of the OIDC RFC7662 introspection endpoint which can introspect both opaque and JWT tokens. This property must be set if OIDC discovery is disabled and 1) the opaque bearer access tokens have to be verified or 2) JWT tokens have to be verified while the cached JWK verification set with no matching JWK is being refreshed. This property will be ignored if the discovery is enabled.	string	
`quarkus.oidc.jwks-path` Relative path of the OIDC JWKS endpoint which returns a JSON Web Key Verification Set. This property should be set if OIDC discovery is disabled and the local JWT verification is required. This property will be ignored if the discovery is enabled.	string	
`quarkus.oidc.end-session-path` Relative path of the OIDC end_session_endpoint. This property must be set if OIDC discovery is disabled and RP Initiated Logout support for the 'web-app' applications is required. This property will be ignored if the discovery is enabled.	string	
`quarkus.oidc.connection-delay` 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 ❓	
`quarkus.oidc.public-key` Public key for the local JWT token verification. OIDC server connection will not be created when this property is set.	string	
`quarkus.oidc.client-id` The client-id of the application. Each application has a client-id that is used to identify the application	string	

`quarkus.oidc.roles.role-claim-path` 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". Use double quotes with the namespace qualified claim names. This property can be used if a token has no 'groups' claim but has the groups set in a different claim.	string	
`quarkus.oidc.roles.role-claim-separator` 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	
`quarkus.oidc.roles.source` Source of the principal roles.	`idtoken`, `access token`, `userinfo`	
`quarkus.oidc.token.issuer` Expected issuer 'iss' claim value.	string	
`quarkus.oidc.token.audience` Expected audience 'aud' claim value which may be a string or an array of strings.	list of string	
`quarkus.oidc.token.token-type` Expected token type	string	
`quarkus.oidc.token.lifespan-grace` Life span grace period in seconds. When checking token expiry, current time is allowed to be later than token expiration time by at most the configured number of seconds. When checking token issuance, current time is allowed to be sooner than token issue time by at most the configured number of seconds.	int	
`quarkus.oidc.token.principal-claim` Name of the claim which contains a principal name. By default, the 'upn', 'preferred_username' and `sub` claims are checked.	string	

`quarkus.oidc.token.refresh-expired` Refresh expired ID tokens. If this property is enabled then a refresh token request is performed and, if successful, the local session is updated with the new set of tokens. Otherwise, the local session is invalidated as an indication that the session at the OpenID Provider no longer exists. This option is only valid when the application is of type `ApplicationType#WEB_APP}`.	boolean	`false`
`quarkus.oidc.token.forced-jwk-refresh-interval` Forced JWK set refresh interval in minutes.	Duration ❓	`10M`
`quarkus.oidc.credentials.secret` Client secret which is used for a 'client_secret_basic' authentication method. Note that a 'client-secret.value' can be used instead but both properties are mutually exclusive.	string	
`quarkus.oidc.credentials.client-secret.value` The client secret	string	
`quarkus.oidc.credentials.client-secret.method` Authentication method.	`basic`, `post`	
`quarkus.oidc.credentials.jwt.secret` client_secret_jwt: JWT which includes client id as one of its claims is signed by the client secret and is submitted as a 'client_assertion' form parameter, while 'client_assertion_type' parameter is set to "urn:ietf:params:oauth:client-assertion-type:jwt-bearer".	string	
`quarkus.oidc.credentials.jwt.lifespan` JWT life-span in seconds. It will be added to the time it was issued at to calculate the expiration time.	int	`10`
`quarkus.oidc.proxy.host` The host (name or IP address) of the Proxy. Note: If OIDC adapter needs to use a Proxy to talk with OIDC server (Provider), then at least the "host" config item must be configured to enable the usage of a Proxy.	string	
`quarkus.oidc.proxy.port` The port number of the Proxy. Default value is 80.	int	`80`

`quarkus.oidc.proxy.username` The username, if Proxy needs authentication.	string	
`quarkus.oidc.proxy.password` The password, if Proxy needs authentication.	string	
`quarkus.oidc.authentication.redirect-path` Relative path for calculating a "redirect_uri" query parameter. It has to start from a forward slash and will be appended to the request URI's host and port. For example, if the current request URI is 'https://localhost:8080/service' then a 'redirect_uri' parameter will be set to 'https://localhost:8080/' if this property is set to '/' and be the same as the request URI if this property has not been configured. Note the original request URI will be restored after the user has authenticated.	string	
`quarkus.oidc.authentication.restore-path-after-redirect` If this property is set to 'true' then the original request URI which was used before the authentication will be restored after the user has been redirected back to the application.	boolean	`true`
`quarkus.oidc.authentication.remove-redirect-parameters` Remove the query parameters such as 'code' and 'state' set by the OIDC server on the redirect URI after the user has authenticated by redirecting a user to the same URI but without the query parameters.	boolean	`true`
`quarkus.oidc.authentication.verify-access-token` Both ID and access tokens are verified as part of the authorization code flow and every time these tokens are retrieved from the user session. One should disable the access token verification if it is only meant to be propagated to the downstream services. Note the ID token will always be verified.	boolean	`true`
`quarkus.oidc.authentication.force-redirect-https-scheme` Force 'https' as the 'redirect_uri' parameter scheme when running behind an SSL terminating reverse proxy. This property, if enabled, will also affect the logout `post_logout_redirect_uri` and the local redirect requests.	boolean	`false`
`quarkus.oidc.authentication.scopes` List of scopes	list of string	

	Type	Default
`quarkus.oidc.authentication.cookie-path` Cookie path parameter value which, if set, will be used for the session and state cookies. It may need to be set when the redirect path has a root different to that of the original request URL.	string	
`quarkus.oidc.authentication.user-info-required` If this property is set to 'true' then an OIDC UserInfo endpoint will be called	boolean	`false`
`quarkus.oidc.authentication.xhr-auto-redirect` If this property is set to 'true' then a normal 302 redirect response will be returned if the request was initiated via XMLHttpRequest and the current user needs to be (re)authenticated which may not be desirable for Single Page Applications since XMLHttpRequest automatically following the redirect may not work given that OIDC authorization endpoints typically do not support CORS. If this property is set to `false` then a status code of '499' will be returned to allow the client to handle the redirect manually	boolean	`true`
`quarkus.oidc.tls.verification` Certificate validation and hostname verification, which can be one of the following values from enum `Verification`. Default is required.	`required`, `none`	`required`
`quarkus.oidc.logout.path` The relative path of the logout endpoint at the application. If provided, the application is able to initiate the logout through this endpoint in conformance with the OpenID Connect RP-Initiated Logout specification.	string	
`quarkus.oidc.logout.post-logout-path` Relative path of the application endpoint where the user should be redirected to after logging out from the OpenID Connect Provider. This endpoint URI must be properly registered at the OpenID Connect Provider as a valid redirect URI.	string	
`quarkus.oidc.authentication.extra-params` Additional properties which will be added as the query parameters to the authentication redirect URI.	`Map<String,String>`	required ❗
Additional named tenants	**Type**	**Default**

`quarkus.oidc."tenant".tenant-id` A unique tenant identifier. It must be set by `TenantConfigResolver` providers which resolve the tenant configuration dynamically and is optional in all other cases.	string	
`quarkus.oidc."tenant".tenant-enabled` If this tenant configuration is enabled.	boolean	`true`
`quarkus.oidc."tenant".application-type` The application type, which can be one of the following values from enum `ApplicationType`.	`web-app, service`	`service`
`quarkus.oidc."tenant".auth-server-url` The base URL of the OpenID Connect (OIDC) server, for example, 'https://host:port/auth'. OIDC discovery endpoint will be called by default by appending a '.well-known/openid-configuration' path to 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	
`quarkus.oidc."tenant".discovery-enabled` Enables OIDC discovery. If the discovery is disabled then the following properties must be configured: - 'authorization-path' and 'token-path' for the 'web-app' applications - 'jwks-path' or 'introspection-path' for both the 'web-app' and 'service' applications 'web-app' applications may also have 'user-info-path' and 'end-session-path' properties configured.	boolean	`true`
`quarkus.oidc."tenant".authorization-path` Relative path of the OIDC authorization endpoint which authenticates the users. This property must be set for the 'web-app' applications if OIDC discovery is disabled. This property will be ignored if the discovery is enabled.	string	
`quarkus.oidc."tenant".token-path` Relative path of the OIDC token endpoint which issues ID, access and refresh tokens. This property must be set for the 'web-app' applications if OIDC discovery is disabled. This property will be ignored if the discovery is enabled.	string	

`quarkus.oidc."tenant".user-info-path` Relative path of the OIDC userinfo endpoint. This property must only be set for the 'web-app' applications if OIDC discovery is disabled and 'authentication.user-info-required' property is enabled. This property will be ignored if the discovery is enabled.	string	
`quarkus.oidc."tenant".introspection-path` Relative path of the OIDC RFC7662 introspection endpoint which can introspect both opaque and JWT tokens. This property must be set if OIDC discovery is disabled and 1) the opaque bearer access tokens have to be verified or 2) JWT tokens have to be verified while the cached JWK verification set with no matching JWK is being refreshed. This property will be ignored if the discovery is enabled.	string	
`quarkus.oidc."tenant".jwks-path` Relative path of the OIDC JWKS endpoint which returns a JSON Web Key Verification Set. This property should be set if OIDC discovery is disabled and the local JWT verification is required. This property will be ignored if the discovery is enabled.	string	
`quarkus.oidc."tenant".end-session-path` Relative path of the OIDC end_session_endpoint. This property must be set if OIDC discovery is disabled and RP Initiated Logout support for the 'web-app' applications is required. This property will be ignored if the discovery is enabled.	string	
`quarkus.oidc."tenant".connection-delay` 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 ❓	
`quarkus.oidc."tenant".public-key` Public key for the local JWT token verification. OIDC server connection will not be created when this property is set.	string	
`quarkus.oidc."tenant".client-id` The client-id of the application. Each application has a client-id that is used to identify the application	string	

`quarkus.oidc."tenant".roles.role-claim-path` 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". Use double quotes with the namespace qualified claim names. This property can be used if a token has no 'groups' claim but has the groups set in a different claim.	string	
`quarkus.oidc."tenant".roles.role-claim-separator` 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	
`quarkus.oidc."tenant".roles.source` Source of the principal roles.	`idtoken`, `accesstoken`, `userinfo`	
`quarkus.oidc."tenant".token.issuer` Expected issuer 'iss' claim value.	string	
`quarkus.oidc."tenant".token.audience` Expected audience 'aud' claim value which may be a string or an array of strings.	list of string	
`quarkus.oidc."tenant".token.token-type` Expected token type	string	
`quarkus.oidc."tenant".token.lifespan-grace` Life span grace period in seconds. When checking token expiry, current time is allowed to be later than token expiration time by at most the configured number of seconds. When checking token issuance, current time is allowed to be sooner than token issue time by at most the configured number of seconds.	int	
`quarkus.oidc."tenant".token.principal-claim` Name of the claim which contains a principal name. By default, the 'upn', 'preferred_username' and `sub` claims are checked.	string	

`quarkus.oidc."tenant".token.refresh-expired` Refresh expired ID tokens. If this property is enabled then a refresh token request is performed and, if successful, the local session is updated with the new set of tokens. Otherwise, the local session is invalidated as an indication that the session at the OpenID Provider no longer exists. This option is only valid when the application is of type `ApplicationType#WEB_APP`}.	boolean	`false`
`quarkus.oidc."tenant".token.forced-jwk-refresh-interval` Forced JWK set refresh interval in minutes.	Duration ❓	`10M`
`quarkus.oidc."tenant".credentials.secret` Client secret which is used for a 'client_secret_basic' authentication method. Note that a 'client-secret.value' can be used instead but both properties are mutually exclusive.	string	
`quarkus.oidc."tenant".credentials.client-secret.value` The client secret	string	
`quarkus.oidc."tenant".credentials.client-secret.method` Authentication method.	`basic`, `post`	
`quarkus.oidc."tenant".credentials.jwt.secret` client_secret_jwt: JWT which includes client id as one of its claims is signed by the client secret and is submitted as a 'client_assertion' form parameter, while 'client_assertion_type' parameter is set to "urn:ietf:params:oauth:client-assertion-type:jwt-bearer".	string	
`quarkus.oidc."tenant".credentials.jwt.lifespan` JWT life-span in seconds. It will be added to the time it was issued at to calculate the expiration time.	int	`10`
`quarkus.oidc."tenant".proxy.host` The host (name or IP address) of the Proxy. Note: If OIDC adapter needs to use a Proxy to talk with OIDC server (Provider), then at least the "host" config item must be configured to enable the usage of a Proxy.	string	
`quarkus.oidc."tenant".proxy.port` The port number of the Proxy. Default value is 80.	int	`80`

`quarkus.oidc."tenant".proxy.username` The username, if Proxy needs authentication.	string	
`quarkus.oidc."tenant".proxy.password` The password, if Proxy needs authentication.	string	
`quarkus.oidc."tenant".authentication.redirect-path` Relative path for calculating a "redirect_uri" query parameter. It has to start from a forward slash and will be appended to the request URI's host and port. For example, if the current request URI is 'https://localhost:8080/service' then a 'redirect_uri' parameter will be set to 'https://localhost:8080/' if this property is set to '/' and be the same as the request URI if this property has not been configured. Note the original request URI will be restored after the user has authenticated.	string	
`quarkus.oidc."tenant".authentication.restore-path-after-redirect` If this property is set to 'true' then the original request URI which was used before the authentication will be restored after the user has been redirected back to the application.	boolean	`true`
`quarkus.oidc."tenant".authentication.remove-redirect-parameters` Remove the query parameters such as 'code' and 'state' set by the OIDC server on the redirect URI after the user has authenticated by redirecting a user to the same URI but without the query parameters.	boolean	`true`
`quarkus.oidc."tenant".authentication.verify-access-token` Both ID and access tokens are verified as part of the authorization code flow and every time these tokens are retrieved from the user session. One should disable the access token verification if it is only meant to be propagated to the downstream services. Note the ID token will always be verified.	boolean	`true`
`quarkus.oidc."tenant".authentication.force-redirect-https-scheme` Force 'https' as the 'redirect_uri' parameter scheme when running behind an SSL terminating reverse proxy. This property, if enabled, will also affect the logout `post_logout_redirect_uri` and the local redirect requests.	boolean	`false`

`quarkus.oidc."tenant".authentication.scopes` List of scopes	list of string	
`quarkus.oidc."tenant".authentication.extra-params` Additional properties which will be added as the query parameters to the authentication redirect URI.	`Map<String,String>`	required ❗
`quarkus.oidc."tenant".authentication.cookie-path` Cookie path parameter value which, if set, will be used for the session and state cookies. It may need to be set when the redirect path has a root different to that of the original request URL.	string	
`quarkus.oidc."tenant".authentication.user-info-required` If this property is set to 'true' then an OIDC UserInfo endpoint will be called	boolean	`false`
`quarkus.oidc."tenant".authentication.xhr-auto-redirect` If this property is set to 'true' then a normal 302 redirect response will be returned if the request was initiated via XMLHttpRequest and the current user needs to be (re)authenticated which may not be desirable for Single Page Applications since XMLHttpRequest automatically following the redirect may not work given that OIDC authorization endpoints typically do not support CORS. If this property is set to `false` then a status code of '499' will be returned to allow the client to handle the redirect manually	boolean	`true`
`quarkus.oidc."tenant".tls.verification` Certificate validation and hostname verification, which can be one of the following values from enum `Verification`. Default is required.	`required`, `none`	`required`
`quarkus.oidc."tenant".logout.path` The relative path of the logout endpoint at the application. If provided, the application is able to initiate the logout through this endpoint in conformance with the OpenID Connect RP-Initiated Logout specification.	string	
`quarkus.oidc."tenant".logout.post-logout-path` Relative path of the application endpoint where the user should be redirected to after logging out from the OpenID Connect Provider. This endpoint URI must be properly registered at the OpenID Connect Provider as a valid redirect URI.	string	

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.

Example configuration:

```
quarkus.oidc.auth-server-
url=http://localhost:8180/auth/realms/quarkus
quarkus.oidc.client-id=backend-service
```

Configuring CORS

If you plan to consume this application from another application running on a different domain, you will need to configure CORS (Cross-Origin Resource Sharing). Please read the HTTP CORS documentation for more details.

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:11.0.0
```

You should be able to access your Keycloak Server at 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.

If you want to use the Keycloak Admin Client to configure your server from your application you need to include the `quarkus-keycloak-admin-client` extension.

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-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-quickstart-runner
```

Testing the Application

The application is using bearer token authorization and the first thing to do is obtain an access token from the Keycloak Server in order to access the application resources:

```
export access_token=$(\
    curl -X POST
http://localhost:8180/auth/realms/quarkus/protocol/openid-
connect/token \
    --user backend-service:secret \
    -H 'content-type: application/x-www-form-urlencoded' \
    -d 'username=alice&password=alice&grant_type=password' | jq
--raw-output '.access_token' \
 )
```

The example above obtains an access token for user `alice`.

Any user is allowed to access the http://localhost:8080/api/users/me endpoint which basically returns a JSON payload with details about the user.

```
curl -v -X GET \
   http://localhost:8080/api/users/me \
   -H "Authorization: Bearer "$access_token
```

The http://localhost:8080/api/admin endpoint can only be accessed by users with the `admin` role. If you try to access this endpoint with the previously issued access token, you should get a `403` response from the server.

```
curl -v -X GET \
    http://localhost:8080/api/admin \
    -H "Authorization: Bearer "$access_token
```

In order to access the admin endpoint you should obtain a token for the `admin` user:

```
export access_token=$(\
    curl -X POST
http://localhost:8180/auth/realms/quarkus/protocol/openid-
connect/token \
    --user backend-service:secret \
    -H 'content-type: application/x-www-form-urlencoded' \
    -d 'username=admin&password=admin&grant_type=password' | jq
--raw-output '.access_token' \
 )
```

Token Claims And SecurityIdentity Roles

SecurityIdentity roles can be mapped from the verified JWT access tokens as follows:

- If `quarkus.oidc.roles.role-claim-path` property is set and a matching array or string claim is found then the roles are extracted from this claim. For example, `customroles`, `customroles/array`, `scope`, `"http://namespace-qualified-custom-claim"/roles`, `"http://namespace-qualified-roles"`, etc.

- If `groups` claim is available then its value is used

- If `realm_access/roles` or `resource_access/client_id/roles` (where `client_id` is the value of the `quarkus.oidc.client-id` property) claim is available then its value is used. This check supports the tokens issued by Keycloak

If the token is opaque (binary) then a `scope` property from the remote token introspection response will be used.

Additionally a custom `SecurityIdentityAugmentor` can also be used to add the roles as documented here.

Single Page Applications

Single Page Application (SPA) typically uses `XMLHttpRequest`(XHR) and the Java Script utility code provided by the OpenId Connect provider to acquire a bearer token and use it to access Quarkus `service` applications.

For example, here is how you can use `keycloak.js` to authenticate the users and refresh the expired tokens from the SPA:

```html
<html>
<head>
    <title>keycloak-spa</title>
    <script
src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script
>
    <script
src="http://localhost:8180/auth/js/keycloak.js"></script>
    <script>
        var keycloak = new Keycloak();
        keycloak.init({onLoad: 'login-required'}).success(function
() {
            console.log('User is now authenticated.');
        }).error(function () {
            window.location.reload();
        });
        function makeAjaxRequest() {
            axios.get("/api/hello", {
                headers: {
                    'Authorization': 'Bearer ' + keycloak.token
                }
            })
            .then( function (response) {
                console.log("Response: ", response.status);
            }).catch(function (error) {
                console.log('refreshing');
                keycloak.updateToken(5).then(function () {
                    console.log('Token refreshed');
                }).catch(function () {
                    console.log('Failed to refresh token');
                    window.location.reload();
                });
            });
        }
    </script>
</head>
<body>
    <button onclick="makeAjaxRequest()">Request</button>
</body>
</html>
```

References

- Keycloak Documentation

- OpenID Connect

- JSON Web Token