Quarkus - Using OpenID Connect to Protect Web Applications using Authorization Code Flow.

This guide demonstrates how to use Quarkus OpenID Connect Extension to protect your Quarkus HTTP endpoints using OpenId Connect Authorization Code Flow supported by OpenId Connect compliant Authorization Servers such as Keycloak.

The extension allows to easily authenticate the users of your web application by redirecting them to the OpenID Connect Provider (e.g.: Keycloak) to login and, once the authentication is complete, return them back with the code confirming the successful authentication. The extension will request ID and access tokens from the OpenID Connect Provider using an authorization code grant and verify these tokens in order to authorize an access to the application.

Please read the Using OpenID Connect to Protect Service Applications guide if you need to protect your applications using Bearer Token Authorization.

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 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:1.8.0.CR1:create \
    -DprojectGroupId=org.acme \
    -DprojectArtifactId=security-openid-connect-web-authentication
-quickstart \
    -Dextensions="oidc"
cd security-openid-connect-web-authentication-quickstart
```

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>
```

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:11.0.1
```

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.

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.

Redirection

When the user is redirected to the OpenID Connect Provider to authenticate, the redirect URL includes a `redirect_uri` query parameter which indicates to the Provider where the user has to be redirected to once the authentication has been completed.

Quarkus will set this parameter to the current request URL by default. For example, if the user is trying to access a Quarkus service endpoint at `http://localhost:8080/service/1` then the `redirect_uri` parameter will be set to `http://localhost:8080/service/1`. Similarly, if the request URL is `http://localhost:8080/service/2` then the `redirect_uri` parameter will be set to `http://localhost:8080/service/2`, etc.

OpenID Connect Providers may be configured to require the `redirect_uri` parameter to have the same value (eg. `http://localhost:8080/service/callback`) for all the redirect URLs. In such cases a `quarkus.oidc.authentication.redirect-path` property has to be set, for example, `quarkus.oidc.authentication.redirect-path=/service/callback`, and Quarkus will set the `redirect_uri` parameter to an absolute URL such as `http://localhost:8080/service/callback` which will be the same regardless of the current request URL.

If `quarkus.oidc.authentication.redirect-path` is set but the original request URL has to be restored after the user has been redirected back to a callback URL such as `http://localhost:8080/service/callback` then a `quarkus.oidc.authentication.restore-path-after-redirect` property has to be set to `true` which will restore the request URL such as `http://localhost:8080/service/1`, etc.

Logout

By default the logout is based on the expiration time of the ID Token issued by the OpenID Connect Provider. When the ID Token expires, the current user session at the Quarkus endpoint is invalidated and the user is 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.

The current user session may be automatically extended by enabling a `quarkus.oidc.token.refresh-expired` property. If it is set to `true` then when the current ID Token expires a Refresh Token Grant will be used to refresh ID Token as well as Access and Refresh Tokens.

User-Initiated Logout

Users can request a logout by sending a request to the Quarkus endpoint logout path set with a `quarkus.oidc.logout.path` property. For example, if the endpoint address is `https://application.com/webapp` and the `quarkus.oidc.logout.path` is set to "/logout" then the logout request has to be sent to `https://application.com/webapp/logout`.

This logout request will start an RP-Initiated Logout and the user will be redirected to the OpenID

Connect Provider to logout where a user may be asked to confirm the logout is indeed intended.

The user will be returned to the endpoint post logout page once the logout has been completed if the `quarkus.oidc.logout.post-logout-path` property is set. For example, if the endpoint address is https://application.com/webapp and the `quarkus.oidc.logout.post-logout-path` is set to "/signin" then the user will be returned to https://application.com/webapp/signin (note this URI must be registered as a valid `post_logout_redirect_uri` in the OpenID Connect Provider).

If the `quarkus.oidc.logout.post-logout-path` is set then a `q_post_logout` cookie will be created and a matching `state` query parameter will be added to the logout redirect URI and the OpenID Connect Provider will return this `state` once the logout has been completed. It is recommended for the Quarkus `web-app` applications to check that a `state` query parameter matches the value of the `q_post_logout` cookie which can be done for example in a JAX-RS filter.

Note that a cookie name will vary when using OpenID Connect Multi-Tenancy. For example, it will be named `q_post_logout_tenant_1` for a tenant with a `tenant_1` id, etc.

Accessing ID and Access Tokens

ID Token is always a JWT token. One can access ID Token claims by injecting `JsonWebToken` with an `IdToken` qualifier:

```
import javax.inject.Inject;
import org.eclipse.microprofile.jwt.JsonWebToken;
import io.quarkus.oidc.IdToken;
import io.quarkus.security.Authenticated;

@Path("/web-app")
@Authenticated
public class ProtectedResource {

    @Inject
    @IdToken
    JsonWebToken idToken;

    @GET
    public String getUserName() {
        return idToken.getName();
    }
}
```

Access Token is usually used by the OIDC `web-app` application to access other endpoints on behalf of the currently logged in user. The raw access token can be accessed as follows:

```
import javax.inject.Inject;
import org.eclipse.microprofile.jwt.JsonWebToken;
import io.quarkus.oidc.AccessTokenCredential;
import io.quarkus.security.Authenticated;

@Path("/web-app")
@Authenticated
public class ProtectedResource {

    @Inject
    JsonWebToken accessToken;

    // or
    // @Inject
    // AccessTokenCredential accessTokenCredential;

    @GET
    public String getReservationOnBehalfOfUser() {
        String rawAccessToken = accessToken.getRawToken();
        //or
        //String rawAccessToken = accessTokenCredential.getToken();

        // Use the raw access token to access a remote endpoint
        return getReservationfromRemoteEndpoint(rawAccesstoken);
    }
}
```

Note that `AccessTokenCredential` will have to be used if the Access Token issued to the Quarkus `web-app` application is opaque (binary) and can not be parsed to `JsonWebToken`.

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

User Info

Set `quarkus.oidc.user-info-required=true` if a UserInfo JSON object from the OIDC userinfo endpoint has to be requested. This will make an `io.quarkus.oidc.UserInfo` (which is a simple `javax.json.JsonObject` wrapper) object accessible as a SecurityIdentity `userinfo` attribute.

Token Claims And SecurityIdentity Roles

The way the roles are mapped to the SecurityIdentity roles from the verified tokens is identical to how it is done for the bearer tokens with the only difference being is that ID Token is used as a source of the roles by default.

Note if you use Keycloak then you should set a Microprofile JWT client scope for ID token to contain a

`groups` claim, please see the Keycloak Server Administration Guide for more information.

If only the access token contains the roles and this access token is not meant to be propagated to the downstream endpoints then set `quarkus.oidc.roles.source=accesstoken`.

If UserInfo is the source of the roles then set `quarkus.oidc.user-info-required=true` and `quarkus.oidc.roles.source=userinfo`, and if needed, `quarkus.oidc.roles.role-claim-path`.

Single Page Applications

Please check if implementing SPAs the way it is suggested in the Single Page Applications for Service Applications section can meet your requirements.

If you prefer to use SPA and `XMLHttpRequest`(XHR) with Quarkus applications, please be aware that OpenID Connect Providers may not support CORS for Authorization endpoints where the users are authenticated after a redirect from Quarkus. This will lead to authentication failures if the Quarkus application and the OpenID Connect Provider are hosted on the different HTTP domains/ports.

In such cases, set the `quarkus.oidc.authentication.xhr-auto-redirect` property to `false` which will instruct Quarkus to return a `499` status code and `WWW-Authenticate` header with the `OIDC` value. The browser script also needs to be updated to set `X-Requested-With` header with the `XMLHttpRequest` value and reload the last requested page in case of `499`, for example:

```
Future<void> callQuarkusService() async {
    Map<String, String> headers = Map.fromEntries([MapEntry("X-
Requested-With", "XMLHttpRequest")]);

    await http
        .get("https://localhost:443/serviceCall")
        .then((response) {
            if (response.statusCode == 499) {

window.location.assign("https://localhost.com:443/serviceCall");
            }
        });
    }
```

Configuration Reference

🔒 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 will be performed if the ID token has expired and, if successful, the local session will be updated with the new set of tokens. Otherwise, the local session will be invalidated and the user redirected to the OpenID Provider to re-authenticate. In this case the user may not be challenged again if the OIDC provider session is still active. For this option be effective the `authentication.session-age-extension` property should also be set to a non-zero value since the refresh token is currently kept in the user session. This option is valid only when the application is of type `ApplicationType#WEB_APP`}.	boolean	`false`
`quarkus.oidc.token.auto-refresh-interval` Token auto-refresh interval in seconds during the user re-authentication. If this option is set then the valid ID token will be refreshed if it will expire in less than a number of minutes set by this option. The user will still be authenticated if the ID token can no longer be refreshed but is still valid. This option will be ignored if the 'refresh-expired' property is not enabled.	Duration ❓	
`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 fetched from the OIDC provider as part of the authorization code flow. ID token is always verified on every user request as the primary token which is used to represent the principal and extract the roles. Access token is not verified by default since it is meant to be propagated to the downstream services. The verification of the access token should be enabled if it is injected as a JWT token. Access tokens obtained as part of the code flow will always be verified if `quarkus.oidc.roles.source` property is set to `accesstoken` which means the authorization decision will be based on the roles extracted from the access token. Bearer access tokens are always verified.	boolean	`false`
`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	
`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.session-age-extension` Session age extension in minutes. The user session age property is set to the value of the ID token life-span by default and the user will be redirected to the OIDC provider to re-authenticate once the session has expired. If this property is set to a non-zero value then the expired ID token can be refreshed before the session has expired. This property will be ignored if the `token.refresh-expired` property has not been enabled.	Duration ❓	`5M`

`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, access token, 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 will be performed if the ID token has expired and, if successful, the local session will be updated with the new set of tokens. Otherwise, the local session will be invalidated and the user redirected to the OpenID Provider to re-authenticate. In this case the user may not be challenged again if the OIDC provider session is still active. For this option be effective the `authentication.session-age-extension` property should also be set to a non-zero value since the refresh token is currently kept in the user session. This option is valid only when the application is of type `ApplicationType#WEB_APP}`.	boolean	`false`

`quarkus.oidc."tenant".token.auto-refresh-interval` Token auto-refresh interval in seconds during the user re-authentication. If this option is set then the valid ID token will be refreshed if it will expire in less than a number of minutes set by this option. The user will still be authenticated if the ID token can no longer be refreshed but is still valid. This option will be ignored if the 'refresh-expired' property is not enabled.	Duration ❓	
`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 fetched from the OIDC provider as part of the authorization code flow. ID token is always verified on every user request as the primary token which is used to represent the principal and extract the roles. Access token is not verified by default since it is meant to be propagated to the downstream services. The verification of the access token should be enabled if it is injected as a JWT token. Access tokens obtained as part of the code flow will always be verified if `quarkus.oidc.roles.source` property is set to `accesstoken` which means the authorization decision will be based on the roles extracted from the access token. Bearer access tokens are always verified.	boolean	`false`

`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.session-age-extension` Session age extension in minutes. The user session age property is set to the value of the ID token life-span by default and the user will be redirected to the OIDC provider to re-authenticate once the session has expired. If this property is set to a non-zero value then the expired ID token can be refreshed before the session has expired. This property will be ignored if the `token.refresh-expired` property has not been enabled.	Duration ❓	`5M`
`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`	

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
- Quarkus Security