

Quarkus - Using OpenID Connect Multi-Tenancy

This guide demonstrates how your OpenID Connect application can support multi-tenancy so that you can serve multiple tenants from a single application. Tenants can be distinct realms or security domains within the same OpenID Provider or even distinct OpenID Providers.

When serving multiple customers from the same application (e.g.: SaaS), each customer is a tenant. By enabling multi-tenancy support to your applications you are allowed to also support distinct authentication policies for each tenant even though if that means authenticating against different OpenID Providers, such as Keycloak and Google.

Please read the [Using OpenID Connect to Protect Service Applications](#) guide if you need to authorize a tenant using Bearer Token Authorization.

Please read the [Using OpenID Connect to Protect Web Applications](#) guide if you need to authenticate and authorize a tenant using OpenID Connect Authorization Code Flow.

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 application which offers a single land page:

- `/{tenant}`

The land page is served by a JAX-RS Resource and shows information obtained from the OpenID Provider about the authenticated user and the current tenant.

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-multi-tenancy` 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 \
  -DprojectId=org.acme \
  -DprojectId=security-openid-connect-multi-tenancy \
  -Dextensions="oidc, resteasy-jsonb"
cd security-openid-connect-multi-tenancy
```

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 `/{tenant}` endpoint. As you can see from the source code below it is just a regular JAX-RS resource:

```

package org.acme.quickstart.oidc;

import javax.inject.Inject;
import javax.ws.rs.GET;
import javax.ws.rs.Path;

import org.eclipse.microprofile.jwt.JsonWebToken;

import io.quarkus.oidc.IdToken;

@Path("/{tenant}")
public class HomeResource {

    /**
     * Injection point for the ID Token issued by the OpenID
    Connect Provider
     */
    @Inject
    @IdToken
    JsonWebToken idToken;

    /**
     * Returns the tokens available to the application. This
    endpoint exists only for demonstration purposes, you should not
     * expose these tokens in a real application.
     *
     * @return the landing page HTML
     */
    @GET
    public String getHome() {
        StringBuilder response = new
    StringBuilder().append("<html>").append("<body>");

        response.append("<h2>Welcome,
    ").append(this.idToken.getClaim("email").toString()).append("</h2>\n");

        response.append("<h3>You are accessing the application
    within tenant <b>").append(idToken.getIssuer()).append("
    boundaries</b></h3>");

        return
    response.append("</body>").append("</html>").toString();
    }
}

```

In order to resolve the tenant from incoming requests and map it to a specific `quarkus-oidc` tenant configuration in `application.properties`, you need to create an implementation for the `io.quarkus.oidc.TenantResolver` interface.

```

package org.acme.quickstart.oidc;

import javax.enterprise.context.ApplicationScoped;

import io.quarkus.oidc.TenantResolver;
import io.vertx.ext.web.RoutingContext;

@ApplicationScoped
public class CustomTenantResolver implements TenantResolver {

    @Override
    public String resolve(RoutingContext context) {
        String path = context.request().path();
        String[] parts = path.split("/");

        if (parts.length == 0) {
            // resolve to default tenant configuration
            return null;
        }

        return parts[1];
    }
}

```

From the implementation above, tenants are resolved from the request path so that in case no tenant could be inferred, `null` is returned to indicate that the default tenant configuration should be used.

Configuring the application

```

# Default Tenant Configuration
quarkus.oidc.auth-server-
url=http://localhost:8180/auth/realms/quarkus
quarkus.oidc.client-id=multi-tenant-client
quarkus.oidc.application-type=web-app

# Tenant A Configuration
quarkus.oidc.tenant-a.auth-server-
url=http://localhost:8180/auth/realms/tenant-a
quarkus.oidc.tenant-a.client-id=multi-tenant-client
quarkus.oidc.tenant-a.application-type=web-app

# HTTP Security Configuration
quarkus.http.auth.permission.authenticated.paths=/*
quarkus.http.auth.permission.authenticated.policy=authenticated

```

The first configuration is the default tenant configuration that should be used when the tenant can not

be inferred from the request. This configuration is using a Keycloak instance to authenticate users.

The second configuration is the configuration that will be used when an incoming request is mapped to the tenant `tenant-a`.

Note that both configurations map to the same Keycloak server instance while using distinct `realms`.

You can define multiple tenants in your configuration file, just make sure they have a unique alias so that you can map them properly when resolving a tenant from your `TenantResolver` implementation.

Google OpenID Provider Configuration

In order to set-up the `tenant-a` configuration to use Google OpenID Provider, you need to create a project as described [here](#).

Once you create the project and have your project's `client_id` and `client_secret`, you can try to configure a tenant as follows:

```
# Tenant configuration using Google OpenID Provider
quarkus.oidc.tenant-b.auth-server-url=https://accounts.google.com
quarkus.oidc.tenant-b.application-type=web-app
quarkus.oidc.tenant-b.client-id={GOOGLE_CLIENT_ID}
quarkus.oidc.tenant-b.credentials.secret={GOOGLE_CLIENT_SECRET}
quarkus.oidc.tenant-b.token.issuer=https://accounts.google.com
quarkus.oidc.tenant-b.authentication.scopes=email,profile,openid
```

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`.

Now, follow the steps below to import the realms for the two tenants:

- Import the `default-tenant-realm.json` to create the default realm
- Import the `tenant-a-realm.json` to create the realm for the tenant `tenant-a`.

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-multi-tenancy-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-multi-tenancy-quickstart-runner
```

Testing the Application

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

- <http://localhost:8080/default>

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

Note that the requested path defines a `default` tenant which we don't have mapped in the configuration file. In this case, the default configuration will be used.

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.

If you try now to access the application at the following URL:

- <http://localhost:8080/tenant-a>

You should be redirected again to the login page at Keycloak. However, now you are going to authenticate using a different `realm`.

In both cases, if the user is successfully authenticated, the landing page will show the user's name and e-mail. Even though user `alice` exists in both tenants, for the application they are distinct users belonging to different realms/tenants.

Programmatically Resolving Tenants Configuration

If you need a more dynamic configuration for the different tenants you want to support and don't want to end up with multiple entries in your configuration file, you can use the `io.quarkus.oidc.TenantConfigResolver`.

This interface allows you to dynamically create tenant configurations at runtime:

```

package io.quarkus.it.keycloak;

import javax.enterprise.context.ApplicationScoped;

import io.quarkus.oidc.TenantConfigResolver;
import io.quarkus.oidc.runtime.OidcTenantConfig;
import io.vertx.ext.web.RoutingContext;

@ApplicationScoped
public class CustomTenantConfigResolver implements
TenantConfigResolver {

    @Override
    public OidcTenantConfig resolve(RoutingContext context) {
        String path = context.request().path();
        String[] parts = path.split("/");

        if (parts.length == 0) {
            // resolve to default tenant configuration
            return null;
        }

        if ("tenant-c".equals(parts[1])) {
            OidcTenantConfig config = new OidcTenantConfig();

            config.setTenantId("tenant-c");

            config.setAuthServerUrl("http://localhost:8180/auth/realms/tenant-
c");

            config.setClientId("multi-tenant-client");
            OidcTenantConfig.Credentials credentials = new
OidcTenantConfig.Credentials();

            credentials.setSecret("my-secret");

            config.setCredentials(credentials);

            // any other setting support by the quarkus-oidc
extension

            return config;
        }

        // resolve to default tenant configuration
        return null;
    }
}

```

The `OidcTenantConfig` returned from this method is the same used to parse the `oidc` namespace

configuration from the `application.properties`. You can populate it using any of the settings supported by the `quarkus-oidc` extension.

Disabling Tenant Configurations

Custom `TenantResolver` and `TenantConfigResolver` implementations may return `null` if no tenant can be inferred from the current request and a fallback to the default tenant configuration is required.

If it is expected that the custom resolvers will always infer a tenant then the default tenant configuration is not needed. One can disable it with the `quarkus.oidc.tenant-enabled=false` setting.

Note that tenant specific configurations can also be disabled, for example: `quarkus.oidc.tenant-a.tenant-enabled=false`.

Configuration Reference

 Configuration property fixed at build time - All other configuration properties are 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.tenant-id</code> A unique tenant identifier. It must be set by <code>TenantConfigResolver</code> providers which resolve the tenant configuration dynamically and is optional in all other cases.	string	
<code>quarkus.oidc.tenant-enabled</code> If this tenant configuration 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, service</code>	<code>service</code>

<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'. 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.</p>	string	
<p><code>quarkus.oidc.discovery-enabled</code></p> <p>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.</p>	boolean	true
<p><code>quarkus.oidc.authorization-path</code></p> <p>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.</p>	string	
<p><code>quarkus.oidc.token-path</code></p> <p>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.</p>	string	
<p><code>quarkus.oidc.user-info-path</code></p> <p>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.</p>	string	
<p><code>quarkus.oidc.introspection-path</code></p> <p>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.</p>	string	

<p><code>quarkus.oidc.jwks-path</code></p> <p>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.</p>	string	
<p><code>quarkus.oidc.end-session-path</code></p> <p>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.</p>	string	
<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>	Duration 	
<p><code>quarkus.oidc.public-key</code></p> <p>Public key for the local JWT token verification. OIDC server connection will not be created when this property is set.</p>	string	
<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>	string	
<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". 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.</p>	string	
<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>	string	

<code>quarkus.oidc.roles.source</code>	Source of the principal roles.	<code>idtoken, access token, userinfo</code>	
<code>quarkus.oidc.token.issuer</code>	Expected issuer 'iss' claim value.	string	
<code>quarkus.oidc.token.audience</code>	Expected audience 'aud' claim value which may be a string or an array of strings.	list of string	
<code>quarkus.oidc.token.token-type</code>	Expected token type	string	
<code>quarkus.oidc.token.lifespan-grace</code>	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	
<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.token.refresh-expired</code>	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 to be effective the <code>authentication.session-age-extension</code> 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 <code>ApplicationType#WEB_APP</code> .	boolean	<code>false</code>

<p><code>quarkus.oidc.token.auto-refresh-interval</code></p> <p>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.</p>	<p>Duration ?</p>	
<p><code>quarkus.oidc.token.forced-jwk-refresh-interval</code></p> <p>Forced JWK set refresh interval in minutes.</p>	<p>Duration ?</p>	<p>10M</p>
<p><code>quarkus.oidc.credentials.secret</code></p> <p>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.</p>	<p>string</p>	
<p><code>quarkus.oidc.credentials.client-secret.value</code></p> <p>The client secret</p>	<p>string</p>	
<p><code>quarkus.oidc.credentials.client-secret.method</code></p> <p>Authentication method.</p>	<p>basic, post</p>	
<p><code>quarkus.oidc.credentials.jwt.secret</code></p> <p>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".</p>	<p>string</p>	
<p><code>quarkus.oidc.credentials.jwt.lifespan</code></p> <p>JWT life-span in seconds. It will be added to the time it was issued at to calculate the expiration time.</p>	<p>int</p>	<p>10</p>
<p><code>quarkus.oidc.proxy.host</code></p> <p>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.</p>	<p>string</p>	
<p><code>quarkus.oidc.proxy.port</code></p> <p>The port number of the Proxy. Default value is 80.</p>	<p>int</p>	<p>80</p>

<code>quarkus.oidc.proxy.username</code>		
The username, if Proxy needs authentication.	string	
<code>quarkus.oidc.proxy.password</code>		
The password, if Proxy needs authentication.	string	
<code>quarkus.oidc.authentication.redirect-path</code>		
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	
<code>quarkus.oidc.authentication.restore-path-after-redirect</code>		
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	<code>true</code>
<code>quarkus.oidc.authentication.remove-redirect-parameters</code>		
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	<code>true</code>
<code>quarkus.oidc.authentication.verify-access-token</code>		
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 <code>quarkus.oidc.roles.source</code> property is set to <code>accesstoken</code> which means the authorization decision will be based on the roles extracted from the access token. Bearer access tokens are always verified.	boolean	<code>false</code>
<code>quarkus.oidc.authentication.force-redirect-https-scheme</code>		
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 <code>post_logout_redirect_uri</code> and the local redirect requests.	boolean	<code>false</code>

<code>quarkus.oidc.authentication.scopes</code>	list of string	
List of scopes		
<code>quarkus.oidc.authentication.cookie-path</code>	string	
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.		
<code>quarkus.oidc.authentication.user-info-required</code>	boolean	<code>false</code>
If this property is set to 'true' then an OIDC UserInfo endpoint will be called		
<code>quarkus.oidc.authentication.session-age-extension</code>	Duration 	<code>5M</code>
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 <code>token.refresh-expired</code> property has not been enabled.		
<code>quarkus.oidc.authentication.xhr-auto-redirect</code>	boolean	<code>true</code>
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 <code>false</code> then a status code of '499' will be returned to allow the client to handle the redirect manually		
<code>quarkus.oidc.tls.verification</code>	required, none	required
Certificate validation and hostname verification, which can be one of the following values from enum <code>Verification</code> . Default is required.		
<code>quarkus.oidc.logout.path</code>	string	
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.		

<code>quarkus.oidc.logout.post-logout-path</code>	string	
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.		
<code>quarkus.oidc.authentication.extra-params</code>	Map<String, String>	required 
Additional properties which will be added as the query parameters to the authentication redirect URI.		
Additional named tenants	Type	Default
<code>quarkus.oidc."tenant".tenant-id</code>	string	
A unique tenant identifier. It must be set by <code>TenantConfigResolver</code> providers which resolve the tenant configuration dynamically and is optional in all other cases.		
<code>quarkus.oidc."tenant".tenant-enabled</code>	boolean	<code>true</code>
If this tenant configuration is enabled.		
<code>quarkus.oidc."tenant".application-type</code>	web-app, service	service
The application type, which can be one of the following values from enum <code>ApplicationType</code> .		
<code>quarkus.oidc."tenant".auth-server-url</code>	string	
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.		
<code>quarkus.oidc."tenant".discovery-enabled</code>	boolean	<code>true</code>
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.		

<p><code>quarkus.oidc."tenant".authorization-path</code></p> <p>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.</p>	string	
<p><code>quarkus.oidc."tenant".token-path</code></p> <p>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.</p>	string	
<p><code>quarkus.oidc."tenant".user-info-path</code></p> <p>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.</p>	string	
<p><code>quarkus.oidc."tenant".introspection-path</code></p> <p>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.</p>	string	
<p><code>quarkus.oidc."tenant".jwks-path</code></p> <p>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.</p>	string	
<p><code>quarkus.oidc."tenant".end-session-path</code></p> <p>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.</p>	string	
<p><code>quarkus.oidc."tenant".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>	Duration 	

<code>quarkus.oidc."tenant".public-key</code>		
Public key for the local JWT token verification. OIDC server connection will not be created when this property is set.	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". 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	
<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".roles.source</code>		
Source of the principal roles.	idtoken, access token, userinfo	
<code>quarkus.oidc."tenant".token.issuer</code>		
Expected issuer 'iss' claim value.	string	
<code>quarkus.oidc."tenant".token.audience</code>		
Expected audience 'aud' claim value which may be a string or an array of strings.	list of string	
<code>quarkus.oidc."tenant".token.token-type</code>		
Expected token type	string	

<p><code>quarkus.oidc."tenant".token.lifespan-grace</code></p> <p>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.</p>	int	
<p><code>quarkus.oidc."tenant".token.principal-claim</code></p> <p>Name of the claim which contains a principal name. By default, the 'upn', 'preferred_username' and <code>sub</code> claims are checked.</p>	string	
<p><code>quarkus.oidc."tenant".token.refresh-expired</code></p> <p>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 to be effective the <code>authentication.session-age-extension</code> 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 <code>ApplicationType#WEB_APP</code>.</p>	boolean	<code>false</code>
<p><code>quarkus.oidc."tenant".token.auto-refresh-interval</code></p> <p>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.</p>	Duration ?	
<p><code>quarkus.oidc."tenant".token.forced-jwk-refresh-interval</code></p> <p>Forced JWK set refresh interval in minutes.</p>	Duration ?	<code>10M</code>
<p><code>quarkus.oidc."tenant".credentials.secret</code></p> <p>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.</p>	string	
<p><code>quarkus.oidc."tenant".credentials.client-secret.value</code></p> <p>The client secret</p>	string	

<p><code>quarkus.oidc."tenant".credentials.client-secret.method</code></p> <p>Authentication method.</p>	<p>basic, post</p>	
<p><code>quarkus.oidc."tenant".credentials.jwt.secret</code></p> <p>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".</p>	<p>string</p>	
<p><code>quarkus.oidc."tenant".credentials.jwt.lifespan</code></p> <p>JWT life-span in seconds. It will be added to the time it was issued at to calculate the expiration time.</p>	<p>int</p>	<p>10</p>
<p><code>quarkus.oidc."tenant".proxy.host</code></p> <p>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.</p>	<p>string</p>	
<p><code>quarkus.oidc."tenant".proxy.port</code></p> <p>The port number of the Proxy. Default value is 80.</p>	<p>int</p>	<p>80</p>
<p><code>quarkus.oidc."tenant".proxy.username</code></p> <p>The username, if Proxy needs authentication.</p>	<p>string</p>	
<p><code>quarkus.oidc."tenant".proxy.password</code></p> <p>The password, if Proxy needs authentication.</p>	<p>string</p>	
<p><code>quarkus.oidc."tenant".authentication.redirect-path</code></p> <p>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.</p>	<p>string</p>	

<pre>quarkus.oidc."tenant".authentication.restore-path-after-redirect</pre> <p>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.</p>	boolean	true
<pre>quarkus.oidc."tenant".authentication.remove-redirect-parameters</pre> <p>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.</p>	boolean	true
<pre>quarkus.oidc."tenant".authentication.verify-access-token</pre> <p>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 <code>quarkus.oidc.roles.source</code> property is set to <code>accesstoken</code> which means the authorization decision will be based on the roles extracted from the access token. Bearer access tokens are always verified.</p>	boolean	false
<pre>quarkus.oidc."tenant".authentication.force-redirect-https-scheme</pre> <p>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 <code>post_logout_redirect_uri</code> and the local redirect requests.</p>	boolean	false
<pre>quarkus.oidc."tenant".authentication.scopes</pre> <p>List of scopes</p>	list of string	
<pre>quarkus.oidc."tenant".authentication.extra-params</pre> <p>Additional properties which will be added as the query parameters to the authentication redirect URI.</p>	Map<String, String>	required 
<pre>quarkus.oidc."tenant".authentication.cookie-path</pre> <p>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.</p>	string	

<p><code>quarkus.oidc."tenant".authentication.user-info-required</code></p> <p>If this property is set to 'true' then an OIDC UserInfo endpoint will be called</p>	boolean	false
<p><code>quarkus.oidc."tenant".authentication.session-age-extension</code></p> <p>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 <code>token.refresh-expired</code> property has not been enabled.</p>	Duration ?	5M
<p><code>quarkus.oidc."tenant".authentication.xhr-auto-redirect</code></p> <p>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 <code>false</code> then a status code of '499' will be returned to allow the client to handle the redirect manually</p>	boolean	true
<p><code>quarkus.oidc."tenant".tls.verification</code></p> <p>Certificate validation and hostname verification, which can be one of the following values from enum <code>Verification</code>. Default is required.</p>	required, none	required
<p><code>quarkus.oidc."tenant".logout.path</code></p> <p>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.</p>	string	
<p><code>quarkus.oidc."tenant".logout.post-logout-path</code></p> <p>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.</p>	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](#)
- [Google OpenID Connect](#)
- [Quarkus Security](#)