

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.



This technology is considered preview.

In *preview*, backward compatibility and presence in the ecosystem is not guaranteed. Specific improvements might require to change configuration or APIs and plans to become *stable* are under way. Feedback is welcome on our [mailing list](#) or as issues in our [GitHub issue tracker](#).

For a full list of possible extension statuses, check our [FAQ entry](#).

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:1.5.2.Final:create \
  -DprojectGroupId=org.acme \
  -DprojectArtifactId=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:10.0.2
```

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> .	web-app, service	<code>service</code>
<code>quarkus.oidc.connection-delay</code> The maximum amount of time the adapter will try connecting to the currently unavailable OIDC server for. For example, setting it to '20S' will let the adapter keep requesting the connection for up to 20 seconds.	Duration 	

<code>quarkus.oidc.auth-server-url</code>		
The base URL of the OpenID Connect (OIDC) server, for example, 'https://host:port/auth'. OIDC discovery endpoint will be called by appending a '/.well-known/openid-configuration' path segment 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	
<code>quarkus.oidc.introspection-path</code>		
Relative path of the RFC7662 introspection service.	string	
<code>quarkus.oidc.jwks-path</code>		
Relative path of the OIDC service returning a JWK set.	string	
<code>quarkus.oidc.end-session-path</code>		
Relative path of the OIDC end_session_endpoint.	string	
<code>quarkus.oidc.public-key</code>		
Public key for the local JWT token verification.	string	
<code>quarkus.oidc.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.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.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.token.issuer</code>		
Expected issuer 'iss' claim value.	string	

<code>quarkus.oidc.token.audience</code>	list of string	
Expected audience 'aud' claim value which may be a string or an array of strings.		
<code>quarkus.oidc.token.lifespan-grace</code>	int	
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.		
<code>quarkus.oidc.token.principal-claim</code>	string	
Name of the claim which contains a principal name. By default, the 'upn', 'preferred_username' and sub claims are checked.		
<code>quarkus.oidc.token.refresh-expired</code>	boolean	false
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 .		
<code>quarkus.oidc.credentials.secret</code>	string	
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.		
<code>quarkus.oidc.credentials.client-secret.value</code>	string	
The client secret		
<code>quarkus.oidc.credentials.client-secret.method</code>	basic, post	
Authentication method.		
<code>quarkus.oidc.credentials.jwt.secret</code>	string	
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".		


<code>quarkus.oidc.credentials.jwt.lifespan</code>		
JWT life-span in seconds. It will be added to the time it was issued at to calculate the expiration time.	int	10
<code>quarkus.oidc.proxy.host</code>		
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	
<code>quarkus.oidc.proxy.port</code>		
The port number of the Proxy. Default value is 80.	int	80
<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	true
<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	true

<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	false
<code>quarkus.oidc.authentication.scopes</code> List of scopes	list of string	
<code>quarkus.oidc.authentication.cookie-path</code> 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	
<code>quarkus.oidc.tls.verification</code> Certificate validation and hostname verification, which can be one of the following values from enum <code>Verification</code> . Default is required.	required, none	required
<code>quarkus.oidc.logout.path</code> 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	
<code>quarkus.oidc.logout.post-logout-path</code> 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	
<code>quarkus.oidc.authentication.extra-params</code> Additional properties which will be added as the query parameters to the authentication redirect URI.	Map<String, String>	required 
Additional named tenants	Type	Default
<code>quarkus.oidc."tenant".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".tenant-enabled</code>		
If this tenant configuration is enabled.	boolean	true
<code>quarkus.oidc."tenant".application-type</code>		
The application type, which can be one of the following values from enum <code>ApplicationType</code> .	web-app, service	service
<code>quarkus.oidc."tenant".connection-delay</code>		
The maximum amount of time the adapter will try connecting to the currently unavailable OIDC server for. For example, setting it to '20S' will let the adapter keep requesting the connection for up to 20 seconds.	Duration 	
<code>quarkus.oidc."tenant".auth-server-url</code>		
The base URL of the OpenID Connect (OIDC) server, for example, 'https://host:port/auth'. OIDC discovery endpoint will be called by appending a '/.well-known/openid-configuration' path segment 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	
<code>quarkus.oidc."tenant".introspection-path</code>		
Relative path of the RFC7662 introspection service.	string	
<code>quarkus.oidc."tenant".jwks-path</code>		
Relative path of the OIDC service returning a JWK set.	string	
<code>quarkus.oidc."tenant".end-session-path</code>		
Relative path of the OIDC end_session_endpoint.	string	
<code>quarkus.oidc."tenant".public-key</code>		
Public key for the local JWT token verification.	string	
<code>quarkus.oidc."tenant".client-id</code>		
The client-id of the application. Each application has a client-id that is used to identify the application	string	

<code>quarkus.oidc."tenant".roles.role-claim-path</code>		
Path to the claim containing an array of groups. It starts from the top level JWT JSON object and can contain multiple segments where each segment represents a JSON object name only, example: "realm/groups". 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".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.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."tenant".token.principal-claim</code>		
Name of the claim which contains a principal name. By default, the 'upn', 'preferred_username' and sub claims are checked.	string	
<code>quarkus.oidc."tenant".token.refresh-expired</code>		
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
<code>quarkus.oidc."tenant".credentials.secret</code>		
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	

<code>quarkus.oidc."tenant".credentials.client-secret.value</code>		
The client secret	string	
<code>quarkus.oidc."tenant".credentials.client-secret.method</code>		
Authentication method.	basic, post	
<code>quarkus.oidc."tenant".credentials.jwt.secret</code>		
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	
<code>quarkus.oidc."tenant".credentials.jwt.lifespan</code>		
JWT life-span in seconds. It will be added to the time it was issued at to calculate the expiration time.	int	10
<code>quarkus.oidc."tenant".proxy.host</code>		
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	
<code>quarkus.oidc."tenant".proxy.port</code>		
The port number of the Proxy. Default value is 80.	int	80
<code>quarkus.oidc."tenant".proxy.username</code>		
The username, if Proxy needs authentication.	string	
<code>quarkus.oidc."tenant".proxy.password</code>		
The password, if Proxy needs authentication.	string	
<code>quarkus.oidc."tenant".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."tenant".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	true
<code>quarkus.oidc."tenant".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	true
<code>quarkus.oidc."tenant".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	false
<code>quarkus.oidc."tenant".authentication.scopes</code> List of scopes	list of string	
<code>quarkus.oidc."tenant".authentication.extra-params</code> Additional properties which will be added as the query parameters to the authentication redirect URI.	Map<String, String>	required 
<code>quarkus.oidc."tenant".authentication.cookie-path</code> 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	
<code>quarkus.oidc."tenant".tls.verification</code> Certificate validation and hostname verification, which can be one of the following values from enum <code>Verification</code> . Default is required.	required, none	required
<code>quarkus.oidc."tenant".logout.path</code> 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	

<code>quarkus.oidc."tenant".logout.post-logout-path</code>		
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](#)
- [Google OpenID Connect](#)