Quarkus - Using Security with an LDAP Realm

This guide demonstrates how your Quarkus application can use an LDAP server to authenticate and authorize your user identities.

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

Architecture

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

- `/api/public`
- `/api/users/me`
- `/api/admin`

The `/api/public` endpoint can be accessed anonymously. The `/api/admin` endpoint is protected with RBAC (Role-Based Access Control) where only users granted with the `adminRole` role can access. At this endpoint, we use the `@RolesAllowed` annotation to declaratively enforce the access constraint. The `/api/users/me` endpoint is also protected with RBAC (Role-Based Access Control) where only users granted with the `standardRole` role can access. As a response, it returns a JSON document with details about the user.

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-ldap-quickstart` directory.

Creating the Maven Project

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

```
mvn io.quarkus:quarkus-maven-plugin:1.7.2.Final:create \
    -DprojectGroupId=org.acme \
    -DprojectArtifactId=security-ldap-quickstart \
    -Dextensions="elytron-security-ldap, resteasy"
cd security-ldap-quickstart
```

This command generates a Maven project, importing the `elytron-security-ldap` extension which is a `wildfly-elytron-realm-ldap` adapter for Quarkus applications.

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

```
./mvnw quarkus:add-extension -Dextensions="elytron-security-ldap"
```

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

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-elytron-security-ldap</artifactId>
</dependency>
```

Writing the application

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

```
package org.acme.elytron.security.ldap;

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

@Path("/api/public")
public class PublicResource {

    @GET
    @PermitAll
    @Produces(MediaType.TEXT_PLAIN)
    public String publicResource() {
        return "public";
   }
}
```

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

```
package org.acme.elytron.security.ldap;

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

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

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

Finally, let's consider the `/api/users/me` endpoint. As you can see from the source code below, we are trusting only users with the `standardRole` role. We are using `SecurityContext` to get access to the current authenticated Principal and we return the user's name. This information is loaded from the LDAP server.

```
package org.acme.elytron.security.ldap;

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

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

    @GET
    @RolesAllowed("standardRole")
    @Path("/me")
    @Produces(MediaType.APPLICATION_JSON)
    public String me(@Context SecurityContext securityContext) {
        return securityContext.getUserPrincipal().getName();
    }
}
```

Configuring the Application

```
quarkus.security.ldap.enabled=true

quarkus.security.ldap.dir-
context.principal=uid=tool,ou=accounts,o=YourCompany,c=DE
quarkus.security.ldap.dir-context.url=ldaps://ldap.server.local
quarkus.security.ldap.dir-context.password=PASSWORD

quarkus.security.ldap.identity-mapping.rdn-identifier=uid
quarkus.security.ldap.identity-mapping.search-base-
dn=ou=users,ou=tool,o=YourCompany,c=DE

quarkus.security.ldap.identity-mapping.attribute-
mappings."0".from=cn
quarkus.security.ldap.identity-mapping.attribute-
mappings."0".to=groups
quarkus.security.ldap.identity-mapping.attribute-
mappings."0".filter=(member=uid={0})
quarkus.security.ldap.identity-mapping.attribute-
mappings."0".filter-base-dn=ou=roles,ou=tool,o=YourCompany,c=DE
```

The `elytron-security-ldap` extension requires a dir-context and an identity-mapping with at least one attribute-mapping to authenticate the user and its identity.

Testing the Application

The application is now protected and the identities are provided by our LDAP server. The very first thing to check is to ensure the anonymous access works.

```
$ curl -i -X GET http://localhost:8080/api/public
HTTP/1.1 200 OK
Content-Length: 6
Content-Type: text/plain;charset=UTF-8

public%
```

Now, let’s try a to hit a protected resource anonymously.

```
$ curl -i -X GET http://localhost:8080/api/admin
HTTP/1.1 401 Unauthorized
Content-Length: 14
Content-Type: text/html;charset=UTF-8

Not authorized%
```

So far so good, now let’s try with an allowed user.

```
$ curl -i -X GET -u adminUser:adminUserPassword
http://localhost:8080/api/admin
HTTP/1.1 200 OK
Content-Length: 5
Content-Type: text/plain;charset=UTF-8

admin%
```

By providing the `adminUser:adminUserPassword` credentials, the extension authenticated the user and loaded their roles. The `adminUser` user is authorized to access to the protected resources.

The user `adminUser` should be forbidden to access a resource protected with `@RolesAllowed("standardRole")` because it doesn’t have this role.

```
$ curl -i -X GET -u adminUser:adminUserPassword
http://localhost:8080/api/users/me
HTTP/1.1 403 Forbidden
Content-Length: 34
Content-Type: text/html;charset=UTF-8

Forbidden%
```

Finally, using the user `standardUser` works and the security context contains the principal details (username for instance).

```
curl -i -X GET -u standardUser:standardUserPassword
http://localhost:8080/api/users/me
HTTP/1.1 200 OK
Content-Length: 4
Content-Type: text/plain;charset=UTF-8

user%
```

Configuration Reference

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

Configuration property	Type	Default
🔒 `quarkus.security.ldap.enabled` The option to enable the ldap elytron module	boolean	`false`
🔒 `quarkus.security.ldap.realm-name` The elytron realm name	string	`Quarkus`
`quarkus.security.ldap.direct-verification` Provided credentials are verified against ldap?	boolean	`true`
`quarkus.security.ldap.dir-context.url` The url of the ldap server	string	required ❗
`quarkus.security.ldap.dir-context.principal` The principal: user which is used to connect to ldap server (also named "bindDn")	string	

`quarkus.security.ldap.dir-context.password` The password which belongs to the principal (also named "bindCredential")	string	
`quarkus.security.ldap.identity-mapping.rdn-identifier` The identifier which correlates to the provided user (also named "baseFilter")	string	`uid`
`quarkus.security.ldap.identity-mapping.search-base-dn` The dn where we look for users	string	required ❗
`quarkus.security.ldap.identity-mapping.attribute-mappings."attribute-mappings".from` The roleAttributeId from which is mapped (e.g. "cn")	string	required ❗
`quarkus.security.ldap.identity-mapping.attribute-mappings."attribute-mappings".to` The identifier whom the attribute is mapped to (in Quarkus: "groups", in WildFly this is "Roles")	string	`groups`
`quarkus.security.ldap.identity-mapping.attribute-mappings."attribute-mappings".filter` The filter (also named "roleFilter")	string	required ❗
`quarkus.security.ldap.identity-mapping.attribute-mappings."attribute-mappings".filter-base-dn` The filter base dn (also named "rolesContextDn")	string	required ❗

References

- LDAP
- Quarkus Security