NEANIAS AAI Documentation

NEANIAS AAI Team

Feb 16, 2022

Contents:

CHAPTER 1

Fit in NEANIAS Ecosystem

The NEANIAS AAI service offers a horizontal solution for all NEANIAS services to cover the common requirements of authenticated access, whether these come in the form of direct user requests or as cross service invocations. Furthermore, provisions to support some degree of the authorization of these requests is offered. The solution is based on widely accepted protocols and standards to ensure wide applicability of the approach.

The NEANIAS AAI Service is backed by the Open Source Keycloak (keycloak.org) application. The application consist of a set of web API accessible endpoints to complete the needed authentication protocol interactions, as well as a set of web graphical user interfaces to support login, profile managements and administrative operations.

CHAPTER 2

Protocol

2.1 OIDC

We currently support Open ID Connect as the main protocol for granting access to protected resources.

2.2 SAML

SAML is not currently activated but can be added as an option if needed.

CHAPTER 3

Realms

NEANIAS AAI currently support three realms, one for each deployment environment in a service's life-cycle.

3.1 Development

Applications that are in the development life-cycle should request access to the Dev Realm of NEANIAS AAI. You will need to contact the NEANIAS AAI administration team through the NEANIAS Service Management System (https://sms.neanias.eu) in order to register your application's redirect urls and you will be provided with a new client id and client secret.

3.2 Staging

Applications in pre-production should request access to the Staging Realm of NEANIAS AAI. You will need to contact the NEANIAS AAI administration team through the NEANIAS Service Management System (https://sms.neanias.eu) in order to register your application's redirect urls and you will be provided with a new client id and client secret.

3.3 Production

Production ready applications should request access to the Production Realm of NEANIAS AAI. You will need to contact the NEANIAS AAI administration team through the NEANIAS Service Management System (https://sms.neanias.eu) in order to register your application's redirect url and you will be provided with a new client id and client secret.

CHAPTER 4

Identity Providers

NEANIAS AAI supports the following Identity Providers:

4.1 Local

Support for NEANIAS AAI local users with username password credentials, restricted for internal usage.

4.2 NEANIAS SSO

Integration with NEANIAS SSO allows authentication to all NEANIAS partner members.

4.3 EOSC

NEANIAS AAI is integrating EOSC AAI to the list of supported federated Identity Providers, extending the authentication process to the full range of the european research community.

4.4 Role Granting

Authorization management takes place through the NEANIAS Service Management System (https://sms.neanias.eu). Interested parties will need to submit there resource access request through a respective ticket.

CHAPTER 5

Client Application

5.1 Process to create a client application

The process of creating new browser facing client applications in NEANIAS AAI is as follows

- You will need to specify to the AAI team in which realm your application will operate
- You will need to inform the AAI team the NEANIAS services that your app will need access to
- You will need to provide to the AAI team a valid redirect url in your app in order to acquire an application code
- You will receive through a secure channel the client id and client secret for your application

5.2 IntraService invocations

The process of creating client applications for intra service communication in NEANIAS AAI is as follows

- You will need to specify to the AAI team in which realm your application will operate
- You will need to inform the AAI team the NEANIAS services that your app will need access to
- You will receive through a secure channel the client id and client secret for your application

5.3 Roles

To enable role based autorization, two types of roles scopes and respective assignments are available:

- **Realm / Global scope:** A global set of roles is defined to facilitate cross client / service user role authorities. The usage and definition of these roles is restricted based on scope and cross service applicability
- **Client / Service Scope:** Each client defines the roles they requires users that utilize their services to be annotated with. Service Providers need to communicate this infroamtion to the AAI adminsitration team through the NEANIAS SMS for these roles to be set up

5.4 Groups

- **Realm / Global scope:** A global set of groups is defined to facilitate cross client / service user groups. The usage and definition of these groups is managed by the NEANIAS AAI administration team

CHAPTER 6

User Management

Through registration users created will have no authorization grants asociated. Service providers, based on their service needs restrict access to their services through the available user claims / roles / groups. Role and group assignment is managed from NEANIAS AAI though the NEANIAS SMS Access Request (https://sms.neanias.eu).

CHAPTER 7

Auditing

All authorizing requests are audited and accounted according to the service privacy statement.

CHAPTER 8

Authorization

8.1 Static Roles / Groups

Authorization can be achieved using the static roles / groups defined globally or in the scope of a client. User roles and groups are made available to each application through the access token. Each application can use this information to authorize the user to the protected resources.

CHAPTER 9

Development

For development reasons we provide a docker image that you can use in you local instalation to integrate with keycloak.

The image extends the original keycloak image and allows you to specify realms, clients and users for development reasons.

In order to import realms, users and clients you will need to mount a volume with the appropriate *-realm.json files to /opt/jboss/realms

9.1 Realm configuration

In order to import realms, users and clients to keycloak you will need to provide json files with name {realmName}-realm.json and the following format:

```
{
    //Realm Name
    "realm": "example",
    //Whether the realm is enabled or disabled by default
    "enabled": true,
    "sslRequired": "external",
    //Whether the realm allows registration
    "registrationAllowed": true,
    "privateKey": "",
    "publicKey": "",
    //What kind of credentials the realm supports
    "requiredCredentials": [ "password" ],
    //Bootstrap Users
    "users" : [
        {
            "username" : "test-user",
            //Whether the user is enabled or disabled by default
            "enabled": true,
            "email" : "test-user@example",
            "firstName": "Test",
```

(continues on next page)

(continued from previous page)

```
            "lastName": "User",
            //User credentials. Type of credential and value
            "credentials" : [
                {
                    "type" : "password",
                    "value" : "password",
                    //The credential is temporary. User will be prompted to set a␣
↪pass when he tries to sign in
                    "temporary": true
                }
            ],
            //Realm scoped roles
            "realmRoles": [ "user" ],
            //Client scoped roles
            "clientRoles": {
                "account": ["view-profile", "manage-account"]
            }
        }
    ],
    //Role configuration
    "roles" : {
        "realm" : [
            {
                "name": "user",
                "description": "User privileges"
            },
            {
                "name": "admin",
                "description": "Administrator privileges"
            }
        ]
    },
    "scopeMappings": [
        {
            "client": "test-client",
            "roles": ["user"]
        }
    ],
    //Client configuration
    "clients": [
        {
            "clientId": "test-client",
            "enabled": true,
            "publicClient": true,
            "baseUrl": "http://localhost:4200/",
            "redirectUris": [
                "http://localhost:4200/*"
            ],
            "webOrigins": []
        }
    ],
    "clientScopeMappings": {
        "account": [
            {
                "client": "test-client",
                "roles": ["view-profile"]
            }
```

(continues on next page)

(continued from previous page)

```
            ]
        }
}
```

9.2 Execution

In order to start a local instance of AAI Service image open a terminal and execute

```
docker run -e USER=<USERNAME> -e PASSWORD=<PASSWORD> -e IMPORT=true -p 8080:8080 -v /
↪path/to/*-realm.json/:/opt/jboss/realms gitlab.neanias.eu:5050/aai-service/server
```

This will start an instance of AAI Service at http://localhost:8080

The **USERNAME** and **PASSWORD** parameters will be used to create the initial credentials for the administrator of the AAI Service Installation

9.3 Misc

By default, restarting the AAI Service container will delete data and configuration from previous runs.

In order to make your changes persistent you can add a mounted volume from your filesystem to /opt/jboss/keycloak/standalone/data inside the container

For example run:

```
docker run -e USER=<USERNAME> -e PASSWORD=<PASSWORD> -e IMPORT=true -p 8080:8080 -v /
↪path/to/*-realm.json/:/opt/jboss/realms -v ./aai-service-db:/opt/jboss/keycloak/
↪standalone/data gitlab.neanias.eu:5050/aai-service/server
```