Implementing a Nuts Use Case

A Nuts use case enables organizations to that don't directly trust each other, but rely on trusted third parties, to find each other and securely exchange data. The use case specifies which data exchanged, on which authorization grounds and how API endpoints can be found.

This book is meant for those who want to understand how to implement a Nuts use case: how to configure and deploy a Nuts node, how to authorize data exchanges and, ... (TODO)

Deploying the Nuts-node in your environment

Deploy the Nuts Node and configure your network to make it work as it should

Deploy Nuts-node

Scenario if you want to use Nuts-node (default)

deploy Nuts-node using

Scenario if you want to use Knooppunt

You could use this deployment guide and/or this docker-compose file.

Deploy admin app

deploy Nuts-admin-app: https://github.com/nuts-foundation/nuts-admin

Configure url rewrite for OAuth

At Nuts uses the RFC8615 spec for .well-known URIs (https://www.rfc-editor.org/rfc/rfc8615.html). This means well-known resources, like the OAuth server, are resolved from the root of the (sub)domain. If you host the Nuts node on a subpath, you need to do some URL rewriting.

On Azure Application Gateway, it would look something like this (given your Nuts node runs on /nuts):

```
// Route well-known endpoints to Nuts node
    // The URL:
    //    https://<host>/.well-known/oauth-authorization-server/nuts/oauth2/<subject-id>
    // should map to:
    //    /.well-known/oauth-authorization-server/oauth2/<subject-id>
    path: '/.well-known/oauth-authorization-server/nuts'
    rewriteCondition: '/.well-known/oauth-authorization-server/nuts(.*)'
    rewriteRule: '/.well-known/oauth-authorization-server{var_uri_path_1}'
```

Configuration of Nuts identity

This chapter provides a manual for the initial configuration of the Nuts-node for an organization. It covers:

- Identity creation (once per care organization)
- Registration for a use case (once per care organization, per use case)

1. Create organization identity

First, a new identity, with associated DID(s), needs to be created for the tuple (organization, environment).

Using API

Perform the following REST API call. Note that `subject` is optional.

```
POST <internal Nuts interface>/internal/vdr/v2/subject
Content-Type: application/json


{"subject": "fooorg-acc"}
```

Using Nuts Admin

1. Log in to the **organization**'s Nuts Admin application. Click on "Identities".
2. Click on the plus sign on the right side.
3. Enter the subject: it consists of the tenant, dash, short form of the environment (e.g. `fooorg-acc`). This will act as key when interacting with the Nuts node's APIs for this instance. *Note: as deployments are isolated, theoretically, a static key could be used. But having the tenant and environment in the name reduces the possibility of leakage due to mixed-up environments.*
4. Click on "Create Identity".

2. Issue X509Credential

This document describes how to issue the X509Credential for a UZI server certificate, used to authenticate in several applications on Nuts/ use cases.

Introduction

Several applicatopns-on-nuts specify that care organizations are authenticated using a Verifiable Credential containing their URA number. As the CIBG doesn't issue these credentials yet, it's derived from the X.509 UZI Server Certificate in the form of an X509Credential.

The credential will then be used for 2 purposes:

- Registering on the Discovery Service, so other participants can resolve key material and API endpoints of the care organization(s).
- Authenticating to other participants when enrolling patients and exchanging data.

The X509Credential will be issued to a DID (Decentralized Identifier) that resides in the Nuts node into which the credential will be loaded. Afterward, the Nuts node can authenticate as the care organization by presenting the credential.

The X509Credential will contain the UZI certificate itself, and the extracted (from the certificate) URA, name, and locality of the care organization. The X509Credential is signed with the private key backing the certificate. Note that the X509Credential **will not contain the private key** itself.

For non-production environments (test, acceptance), see the last section of this document.

Requirements

- DID (Decentralized Identifier) to which the X509Credential will be issued. Supplied by the Nuts node hoster (usually the vendor).
- CIBG UZI Server Certificate:
  - Full certificate chain (root CA certificate up until the leaf certificate) as PEM file
  - Decrypted private key of the leaf certificate as PEM file
- A system with Docker installed to run `nutsfoundation/go-didx509-toolkit` (https://hub.docker.com/r/nutsfoundation/go-didx509-toolkit)
  - The source code of the Docker image can be found here: https://github.com/nuts-foundation/go-didx509-toolkit
- For **non-production environments**: a checkout of https://github.com/nuts-foundation/go-didx509-toolkit that can run a Bash script, with `openssl` installed.

Procedure

1. Run the following command from a directory that contains a directory called `certs` with the certificate (`certificate.pem`) and private key (`privatekey.pem`) PEM files. Replace `<did>` with the proper DID.

```
docker run --rm -v "$(pwd)/certs:/certs" \
  nutsfoundation/go-didx509-toolkit:1.1.0 \
  vc /certs/certificate.pem /certs/privatekey.pem \
  "CN=UZI-register Private Server CA G1,O=CIBG,C=NL,2.5.4.97=#130e4e54524e4c2d3530303030353335" \
  <did>
```

Note: check https://hub.docker.com/r/nutsfoundation/go-didx509-toolkit/tags for the latest version.

This yields an X509Credential in JWT format which can then be loaded into the care organization’s credential wallet in its Nuts Node. E.g.:


```
eyJhbGciOiJQUzI1NiIsImtpZCI6ImRpZDp4NTA5OjA6c2hhMjU2OnN6cU1hVHBuRDZHTjBhUnJUOThlVjRiaEFvT2d5SX
RFWlZ5c2tZeUxfUWM6OnNhbjpvdGhlck5hbWU6Mi4xNi41MjguMS4xMDA3Ljk5LjIxMTAtMS0wLVMtMjE1MTQtMDAu
MDAwLTA6OnN1YmplY3Q6TDpOaWV1d2VnZWluOk86QW50b25pdXMlMjBaaWVrZW5odWlzIzAiLCJ0eXAiOiJKV1QiL
CJ4NWMiOlsiTUlJRDBqQ0NBcnFnQXdJQkFnSVVGVFBPK3BVazMyUVdzWXlMWWRsTFRtbFJXVmt3RFFZSktvWklodm
NOQVFFTEJRQXdHekVaTUJjR0ExVUVBd3dRUm1GclpTQlZXa2tnVW05dmRDQkRRVEFlRncweU5UQXhNamd4TmpF
eU1EUmFGdzB5TmpBeE1qZ3hOakV5TURSYU1JR0RNVUl3UUFZRFZRUURERGxoYm5SdmJtbDFjM3BwWld0bGJtaDF
hWE11WkdWMlpXeHZjRzFsYm5RdWFXNTBaV2R5WVhScGIyNHVlbTl5WjJKcGFtcHZkUzVqYjIweEhEQWFCZ05WQkF
vTUUwRnVkRzl1YVhWeklGcHBaV3RsYm1oMWFYTXhFekFSQmdOVkJBY01DazVwWlhWM1pXZGxhVzR4Q2pBSUJnTl
ZCQVVUQVRBd2dnRWlNQTBHQ1NxR1NJYjNEUUVCQVFVQUE0SUJEd0F3Z2dFS0FvSUJBUUR0RlRMbERPay9lU3NqN
WgrdE9Cb2lvUURrSTloTEFTRUlOTk9rOEpldnhmaW5aMkNySUJMMm9QVUxOY1pyZHBGUXE4VXVGdnl0bVI0SVZM
OE1HZVpDRFNiOVMvQnhpZ1RJM3RUenhyRVdCUTliR3paZVlKd29BbHRaejNUV01nVUcwaTVoek1vUFRtMnRxQW1P
VXk3ZmUyZldMNFUyQ0lGWXhHbzRMVHZOcUphdHROZWRYN1lxTE5qVnNUSGE5eDJjM3NlWDFyUTV4bll1Z1V4Ulh
4blFUVExQVUV6MmVyeFUxeTdDRk5LZDA1SGhoVVBqYUZSOUM5MXdCNDdRL2ZFekhuaEllUTBXT2dEUTJmMHNlW
VdiS010Q05aZ3VCWlRJRG9Qc0RvSkZrczdpT3ozM0c0NW1vei9QN2NsWHV2YTlJMDZQcTdlRXV6K1dud2kxVjRLREF
nTUJBQUdqZ2FRd2dhRXdlUVlEVllwbEJCWXdGQVlJS3dZQkJRVUhBd0VHQ0NzR0FRVUZCd01DTUVBR0ExVWRFUVE
1TURlZ05RWURWUVVGb0M0TUxESXVNVFl1TlRJNExqRXVNVEF3Tnk0NU9TNHlNVEV3TFRFdE1DMVRMVEl4TlRFME
xUQXdMakF3TUMwd01CMEdBMVVkRGdRV0JCUnQxSU5xc3NldW1aTkROZmg5UDF6a3FteTBPakFmQmdOVkhTTU
VHREFXZ0JTYnNjOUY4RGV6M1hqSWNpWDNINXU2Y3RRU3ZUQU5CZ2txaGtpRzl3MEJBUXNGQUFPQ0FRRUFoc3RY
elpBdm9WUUpsNGdMMEtxcUlCQ1RMbURsYktGdmVhemd3SFVsOGkvVStLTnNUQ2tscDJVM05FMXNuQmVXNVlVR3
VXMzhVS2QzL0p4Zm5pdk9XZXhyeFlwY0pCY2lJYVhYN0ZMSDYyNmlUbXhtZnpEblZFUUhWUVF4b1djSHhVbmlMWV
pxTDQzK2hYekFiMWRxKzlmaHN4djViSURDemw0SHFCRy9oMlA3N0tsUVZOMzJML0ZTZ1c2TUlNTFJDK0R6RXR5T25
kd0pYN2pSTmxpNGVPOWpXOHh2b25NelZwZmdHaGdjdVArLzlaY0NJb2c1QW95dEJDTE54a2tibmR0ZnRZRktXQjZ
3S2xzeG5JRWNJTDFET2FPalArQnUwZStFRzZuSVdYK1MxakxVKzZjYjk0VXNrRHJhS0VmUk9GQzlOWVlLQkF4cDdwT
```

nVnUWNCVUE9PSIsIk1JSUM5akNDQWQ2Z0F3SUJBZ0lVUkZDcVByTDNRUWRCTk9xa3dtWFdOZ3g5cGRRd0RRWUp
Lb1pJaHZjTkFRRUxCUUF3R3pFWk1CY0dBMVVFQXd3UVJtRnJaU0JWV2trZ1VtOXZkQ0JEUVRBZUZ3MHlOREV4TVRF
eE5ERTFNVGhhRncwek5ERXhNRGt4TkRFMU1UaGFNQnN4R1RBWEJnTlZCQU1NRUVaaGEyVWdWVnBKSUZKdmIzU
WdRMEV3Z2dFaU1BMEdDU3FHU0liM0RRRUJBUVVBQTRJQkR3QXdnZ0VLQW9JQkFRRFQ1SjhnS2R5TUpOaTNjdUFt
SitNSUxyTXV3ckt5VFJZaGpVVUZISG41cmNWYUhOMGh6QjZ2NXQ3NE50NDB4VVhSTmFvbURjY2xCSU9sd3Q4ZjYy
SkEycC9qODNFTmZkTHJYdlV1OU5NVGhrcVp3WjlkelJ3SzdsM1VaQnE4TlRRVU83NFc0TTJxeDhuclhxMzFlV29neFVV
SUZjMVhPUmg1ZWNlYmVMNW1VYjJFNlVsbURtTmdtMmZHZVNtbWlzOHppZUkrS0tZT2hpL2hZdHllaXhyZzdyeFA0d
jBWUnJFc3RjV0FldFJnWFdRWDBFbEF4czBWcnN5Ni92djNwRXRYaHg4d2Iyd2kyeFkxNGQ5SWg4SGRlTkkrKzN3SWJ
aejZXVk0zZkQ1UUZlVjJFWkJlK3NvbzBwZktqMnRIc2FEejNGUE11TXpJTHQ2VTZQVDRBTElkQWdNQkFBR2pNakF3TU
E4R0ExVWRFd1FJTUFZQkFmOENBUUF3SFFZRFZSME9CQllFRkp1eHowWHdON1BkZU1oeUpmY2ZtN3B5MUJLOU1B
MEdDU3FHU0liM0RRRUJDd1VBQTRJQkFRQWhscGt6Njh4MmRHcE9MWDNGekFiOEVlK1kyT1YrUldGcHNNRTlaVkRV
MDZKRVRQZlBDajAyUEg4MmxnVW5jNGplUjgxclBTc0l0MnNzcW0yUzR6YjAyTmlwNTk1Y0FxQ0t2bUJmRWM5aFBQ
VzJ1Z3BOeFQ4WlJVNExLcnFwVjRuSjZuQnZEcW1HdUg1dXE5Tmc5bDlTbk0zZUttZFp0SktjK1pOQVBLeFZBaXVlTFR
kcjZXMlVibUtvWkFSUVEwSkxrRm5aT3huVWtyOHBRZnhVekVJVWtlZzJkV2FhSS80d280UG5pN3hYZ2dGb1BEcFZ6d
HUvaVAzM1hCTHFYSnd4eEhYaHE5bmM5SlUva0VYRHQ3ajhFZ295Sm83SmpTS2NqcFJmcEdrRTVncXFCNFNhOHdB
c0FQVUszalJyZXV5dGxsQXRRVVpSYkN0SGJ4Y2xjOXlBll0sIng1dCI6Ikl5UjBhb1dHWDc2WEVTd3FCVlpfQkM4T2Exdy
J9.eyJleHAiOjE3Njk2MTY3MjQsImlzcyI6ImRpZDp4NTA5OjA6c2hhMjU2OnN6cU1hVHBuRDZHTjBhUnJUOThlVjRiaEFv
T2d5SXRFWlZ5c2tZeUxfUWM6OnNhbjpvdGhlck5hbWU6Mi4xNi41MjguMS4xMDA3Ljk5LjIxMTAtMS0wLVMtMjE1MT
QtMDAuMDAwLTA6OnN1YmplY3Q6TDpOaWV1d2VnZWluOk86QW50b25pdXMlMjBaaWVrZW5odWlzIiwianRpIjoiZ
GlkOng1MDk6MDpzaGEyNTY6c3pxTWFUcG5ENkdOMGFSclQ5OGVWNGJoQW9PZ3lJdEVaVnlza1l5TF9RYzo6c2FuO
m90aGVyTmFtZToyLjE2LjUyOC4xLjEwMDcuOTkuMjExMC0xLTAtUy0yMTUxNC0wMC4wMDAtMDo6c3ViamVjdDpM
Ok5pZXV3ZWdlaW46TzpBbnRvbml1cyUyMFppZWtlbmh1aXMjZjcwODVhZDEtYTQ4Mi00MjU4LWE5NDgtYWQ5YTd
iMjRmZDZkIiwibmJmIjoxNzM5NDQyODQ1LCJzdWIiOiJkaWQ6d2ViOmFudG9uaXVzemlla2VuaHVpcy5kZXZlbG9wb
WVudC5pbnRlZ3JhdGlvbi56b3JnYmlqam91LmNvbTpudXRzOmlhbTozYmYxY2NmMS0wMWI5LTRmNDUtYTA2Yy05
NTdkMGIwZWJjOTkiLCJ2YyI6eyJAY29udGV4dCI6WyJodHRwczovL3d3dy53My5vcmcvMjAxOC9jcmVkZW50aWFscy9
2MSJdLCJjcmVkZW50aWFsU3ViamVjdCI6W3siaWQiOiJkaWQ6d2ViOmFudG9uaXVzemlla2VuaHVpcy5kZXZlbG9wb
WVudC5pbnRlZ3JhdGlvbi56b3JnYmlqam91LmNvbTpudXRzOmlhbTozYmYxY2NmMS0wMWI5LTRmNDUtYTA2Yy05
NTdkMGIwZWJjOTkiLCJzYW4iOnsib3RoZXJOYW1lIjoiMi4xNi41MjguMS4xMDA3Ljk5LjIxMTAtMS0wLVMtMjE1MTQtMD
AuMDAwLTAifSwic3ViamVjdCI6eyJMIjoiTmlldXdlZ2VpbiIsIk8iOiJBbnRvbml1cyBaaWVrZW5odWlzIn19XSwidHlwZSI
6WyJWZXJpZmlhYmxlQ3JlZGVudGlhbCIsIlg1MDlDcmVkZW50aWFsIl19fQ.IjO1udVtlnFfEN6qs3ZjTmCyUyCEPSyJHa
BApZeMdhPJ8mtoqSQk6xsr8c4vmQhYfHCJLkibR0y2DvuzX13kstsn3vDeidosIRf_yTxChUUa1b2FQMLBc-
t2i01PGIj2B7WMhbbRE1PmDrVAUn2fwv9n0V5XMVJstzzNKR9DdFoZ3dr4f0oG-
aaNe2xzS4IM_NpchKGpprXqzROh1iavZ6zjnd1GhDH3tq1RMXGBmNQBwlon2Bk2bfB-3oeGkEzhDvt5Mo8sBlDI-E-
4htqZx17UR6uC5J_bVf4fklULxZzeUQ7-y7qeEz5bcyYOin3hKlecWDjRa4yQvCxAjz2ICQ

Additional procedure for non-production environments

For non-production (e.g., test or acceptance) environments, CIBG issues test-grade certificates that don’t match the actual care organization’s name, URA and locality. In those cases, a self-signed test CA is used to self-issue a “fake” UZI server certificate. The self-signed test CA can be found in the `test_ca` directory in https://github.com/nuts-foundation/go-didx509-toolkit. Use the following command (in the `test_ca` directory) to issue a certificate:

`./issue-cert.sh <domain (CN)> <name (O)> <locality (L)> <uzi> <ura> <agb>`

Note that the values for domain, UZI and AGB are arbitrary and not used in the Shared Care Planning ecosystem.

It outputs the generated certificate chain PEM file and private key PEM file in the `out` directory. Use these in the procedure documented above.

Also, make sure to replace the DN of the certificate CA in the issuance command (`CN=UZI-register Private Server CA G1...`) with `"CN=Fake UZI Root CA"` .

The command should then looks as follows:

```
docker run --rm -v "$(pwd)/certs:/certs" \
  nutsfoundation/go-didx509-toolkit:1.1.0 \
  vc /certs/somedomain-chain.pem /certs/somedomain.key \
  "CN=Fake UZI Root CA" \
  <did>
```

3. Load X509Credential into organization wallet

Using API

To load the credential into the Nuts node via its REST API, perform the following HTTP request:

```
POST <internal Nuts interface>/internal/vcr/v2/holder/<subjectID>/vc
Content-Type: application/json


"<JWT>"
```

Replace `<subjectID>` with the subject, and `<JWT>` with the credential in JWT format (a JSON string enclosed in double quotes)

Using Nuts Admin

1. Log in to the **care organization**'s Nuts Admin application.
2. Click on "Identities".
3. Click on the wallet to load the credential into (there should be only one).
4. Click on "Load Credential".
5. Enter/paste the credential in the input field.
6. Click on "Load credential".

4. Discovery Service registration

Before another party can interact with you, they will first need to find your endpoints. The discovery service in the Nuts node helps you. You do this once per organization, per use case.

Using API

```
POST <internal Nuts interface>/internal/discovery/v1/<servicedefinition-id>/<subject-id>
Content-Type: application/json


{
  "registrationParameters": {
    "abc": "xyz",
    "someparameter": "somevalue"
  }
}
```

Replace `<servicedefinition-id>` with the servicedefinition-id and replace `<subject-id>` with the subject-id of the organization.

> For each use case a `discovery service definition` is specified that specifies the servicedefinition-id and the registrationParameters.

Using Nuts Admin

to do

Data exchange using Nuts

This chapter describes how to perform inbound or outbound data exchanges using Nuts. It assumes you've deployed and configured your Nuts node in the preceding chapters.

Requesting access (outbound)

To access APIs secured through Nuts, callers need an access token issued by the OAuth2 Authorization Server of the API owner. This page describes how to acquire an access token.

Requesting Service Access Token

This section describes which value(s) need to be specified in the service access token request.

- In the request URL:
  - `subjectID`: the ID of the local requester, which was provided by the Nuts node when the subject and its DIDs was created.
- In the request body:
  - `authorization_server`: the OAuth2 issuer URL of the party that grants access, found in the service discovery search result.
  - `scope`: specifies what resources the access token will give access to. This is specified by the use case.
  - `credentials` (optional): one or more credentials to provide to the authorization server that are not in the requester's wallet. This is typically used to provide an `NutsEmployeeCredential` to the authorization server. See the section below for how to provide this.
  - `token_type` (optional): by default, tokens are of type DPoP that mitigate token theft. Alternatively, the `Bearer` token type can be specified, but you'll be more vulnerable to MITM attacks.

Example

```
POST <internal Nuts interface>/internal/auth/v2/<subjectID>/request-service-access-token
Content-Type: application/json

{
  "authorization_server": "https://example.com/oauth2/hospital_x",
  "scope": "eOverdracht-sender"
}
```

Providing additional credentials

The service access token request allows you to supply credentials to the request, that are not in the subject's wallet but required for authentication. For instance, an `NutsEmployeeCredential` that contains information about the current logged-in user for logging purposes. These credential don't need to be signed: in that case they will be "self-attested" (e.g., the `NutsEmployeeCredential`); the

Verifiable Presentation's signature will provide authenticity.

Example

The example below shows an example access token request with an `NutsEmployeeCredential`.

```
POST <internal Nuts interface>/internal/auth/v2/<subjectID>/request-service-access-token
Content-Type: application/json

{
  "authorization_server": "https://example.com/oauth2/hospital_x",
  "scope": "eOverdracht-sender",
  "credentials": [
     {
       "@context": [
         "https://www.w3.org/2018/credentials/v1",
         "https://nuts.nl/credentials/v1"
       ],
       "type": ["VerifiableCredential", "NutsEmployeeCredential"],
       "credentialSubject": {
         "name": "John Doe",
         "roleName": "Nurse",
         "identifier": "123456"
       }
     }
   ]
}
```

Authorizing access (inbound)

TODO

Discovery of organizations and endpoints

Discovery of Organizations and Endpoints

Once organizations have activated themselves for a use case (covered in previous chapters), they become discoverable. This chapter focuses exclusively on how clients find organizations and their service endpoints using the Service Discovery API:

```
GET <internal Nuts interface>/internal/discovery/v1/{serviceID}
```

This API returns all organizations registered for a given service, optionally filtered by search parameters. It is the primary mechanism for discovering:

- Which organizations participate in your use case
- What endpoints they expose (FHIR URL, OAuth server, etc.)

Example

For instance, to search the "eOverdracht" service for a care organization that contains "Thuiszorg" in its name, perform the following HTTP query:

```
GET <internal Nuts
interface>/internal/discovery/v1/eOverdracht?credentialSubject.organization.name=*Thuiszorg*
```

Could yield (some fields omitted for brevity):


```
[
 {
  "credential_subject_id": "did:web:example.com",
  "fields": {
    "organization_name": "Thuiszorg de Zonnebloem"
  },
  "id": "did:web:example.com#1",
  "registrationParameters": {
   "fhirBaseURL": "https://example.com/fhir"
  },
  "vp": {
```

```
      // etc
    }
  }
]
```

What `fields` and `registrationParameters` are returned depends on the use case. Review the use case's Discovery Service's Presentation Definition for more information.