Basics

The basics needed for every use case.

- Subject registration
- Obtaining credentials
- Service activation
- Service search
- Access tokens
- Authentication & authorization

Subject registration

The first thing you need to do is create some public/private key material for your subjects. We use the term subjects for everything related to vendors, tenants, organizations and clients. Behind every subject there can be multiple DIDs (decentralized identifiers). For most of the interactions with the Nuts node, you'll be using the subject identifier.

Create a subject

```
POST /internal/vdr/v2/subject


{
  "subject": "my_subject_identifier"
}
```

See API

The `subject` in the POST body is optional. If not given, the subject identifier will be generated for you and returned in the result:

```
{
  "subject": "my_subject_identifier",
  "documents": [
    {
      "id": "did:nuts:B8PUHs2AUHbFF1xLLK4eZjgErEcMXHxs68FteY7NDtCY",
      ...
    },
    {
      "id": "did:web:example.com:iam:657f064a-ebef-4f0f-aa87-88ed32db3142",
      ...
    }
  ]
}
```

You can now use these DIDs to issue and receive Verifiable Credentials.

List DIDs

For some operations you still need DIDs. Issuing credentials is an example. You can find your subject's DIDs by calling the following API:

```
GET /internal/vdr/v2/subject/my_subject_identifier
```

Result:

```
[
  "did:nuts:B8PUHs2AUHbFF1xLLK4eZjgErEcMXHxs68FteY7NDtCY",
  "did:web:example.com:iam:657f064a-ebef-4f0f-aa87-88ed32db3142"
]
```

Obtaining credentials

After you created a subject with some DID documents, it's time to give meaning to these identifiers. Without Verifiable Credentials, DIDs are just useless identifiers. They receive meaning when another party issues credentials to one of those DIDs. Attestations in the credentials will later be used for authentication & authorization.

Issue a Verifiable Credential

With the Nuts node, you can issue any kind of credential from one of your subjects to any resolvable DID (did:nuts, did:web, did:key and did:jwk). What the meaning of the credential is and how it can be used is determined by the use case.

```
POST /internal/vcr/v2/issuer/vc


{
  "@context": "https://nuts.nl/credentials/v1",
  "type": "NutsOrganizationCredential",
  "issuer": "did:web:example.com:iam:issuer",
  "credentialSubject": {
    "id": "did:web:example.com:iam:657f064a-ebef-4f0f-aa87-88ed32db3142",
    "organization": {
        "name": "Care bears",
        "city": "Care town"
    }
  },
  "withStatusList2021Revocation": true
}
```

See API

The `withStatusList2021Revocation` option allows for credential revocation.

Response:

```
{
  "@context":[
    "https://nuts.nl/credentials/v1",
    "https://w3id.org/vc/status-list/2021/v1",
```

```
    "https://w3c-ccg.github.io/lds-jws2020/contexts/lds-jws2020-v1.json",
    "https://www.w3.org/2018/credentials/v1"
  ],
  "type":["NutsOrganizationCredential","VerifiableCredential"],
  "id":"did:web:example.com:iam:issuer#f2fb02d3-6216-47b9-8e54-b30438a1090e",
  "issuanceDate":"2024-09-09T14:09:50.901123+02:00",
  "issuer":"did:web:example.com:iam"issuer",
  "credentialStatus":{
    "id":"https://example.com/statuslist/did:web:example.com:iam:issuer/1#5",
    "statusListCredential":"https://example.com/statuslist/did:web:example.com:iam:issuer/1",
    "statusListIndex":"5",
    "statusPurpose":"revocation",
    "type":"StatusList2021Entry"
  },
  "credentialSubject":{
    "id":"did:web:example.com:iam:657f064a-ebef-4f0f-aa87-88ed32db3142",
    "organization":{
      "name":"Care bears",
      "city":"Care town"
    }
  },
  "proof":{...}
}
```

Load a Verifiable Credential

The Nuts node currently doesn't support a new (Nuts network is deprecated) method for exchanging Verifiable Credentials from issuer to holder. The issuer will have to send the credential to the holder out-of-band. The holder can then upload the credential:

```
POST /internal/vcr/v2/holder/did:web:example.com:iam:657f064a-ebef-4f0f-aa87-88ed32db3142/vc


{
  "@context":[
    "https://nuts.nl/credentials/v1",
    "https://w3id.org/vc/status-list/2021/v1",
    "https://w3c-ccg.github.io/lds-jws2020/contexts/lds-jws2020-v1.json",
    "https://www.w3.org/2018/credentials/v1"
  ],
  "type":["NutsOrganizationCredential","VerifiableCredential"],
```

```
  "id":"did:web:example.com:iam:issuer#f2fb02d3-6216-47b9-8e54-b30438a1090e",
  "issuanceDate":"2024-09-09T14:09:50.901123+02:00",
  "issuer":"did:web:example.com:iam"issuer",
  "credentialStatus":{
    "id":"https://example.com/statuslist/did:web:example.com:iam:issuer/1#5",
    "statusListCredential":"https://example.com/statuslist/did:web:example.com:iam:issuer/1",
    "statusListIndex":"5",
    "statusPurpose":"revocation",
    "type":"StatusList2021Entry"
  },
  "credentialSubject":{
    "id":"did:web:example.com:iam:657f064a-ebef-4f0f-aa87-88ed32db3142",
    "organization":{
      "name":"Care bears",
      "city":"Care town"
    }
  },
  "proof":{...}
}
```

Service activation

Before another party can interact with you, they will first need to find your endpoints. The discovery service in the Nuts node can help you. The Nuts node can act as both server and client. From a use case definition you'll receive the following information/files:

- **Discovery Service Definition**. A file that contains the ID, server address, allowed DID methods and Presentation Definition for the discovery service. The Presentation Definition lists a number of constraints. The constraint identifiers can be used as additional search parameters.
- **Use case specific endpoints**. Endpoint types like: FHIR, notification, etc. These have to be added through the additional registration parameters.

Example:


```
{
  "id": "coffeecorner",
  "did_methods": ["web", "nuts"],
  "endpoint": "https://example.com/discovery/coffeecorner",
  "presentation_max_validity": 36000,
  "presentation_definition": {
    "id": "coffeecorner2024",
    "format": {
      "ldp_vc": {
        "proof_type": [
          "JsonWebSignature2020"
        ]
      },
      "jwt_vp": {
        "alg": ["ES256"]
      }
    },
    "input_descriptors": [
      {
        "id": "NutsOrganizationCredential",
        "constraints": {
          "fields": [
            {
              "path": [
```

```
            "$.type"
          ],
          "filter": {
            "type": "string",
            "const": "NutsOrganizationCredential"
          }
        },
        {
          "id": "organization_name",
          "path": [
            "$.credentialSubject.organization.name"
          ],
          "filter": {
            "type": "string"
          }
        },
        {
          "path": [
            "$.credentialSubject.organization.city"
          ],
          "filter": {
            "type": "string"
          }
        }
      ]
    }
  ]
 }
}
```

The constraint for organization name (`organization_name`) can be used as search parameter.

Activate service for subject

```
POST /internal/discovery/v1/coffeecorner/my_subject_identifier


{
  "registrationParameters": {
    "fhir": "https://api.example.com/fhir",
```

```
    "contact": "alice@example.com"
  }
}
```

As you can see, contact information can also be added. Follow the use case requirements.

The response can be a `200 OK` which means the service has been activated immediately. It can also be a `202 Accepted` which means that the Nuts node received your request but was unable to send the activation to the discovery server. The Nuts node will retry periodically. Others can't find you yet. The logs will keep you posted.

To know if you're findable for others, you can perform a search on the registered subject.

Service search

You can find participants of a use case via their registered services. A use case defines the service identifier and which credentials are required through the presentation definition. Any constraint in the presentation definition with an identifier can also be used in the search query.

```
GET /internal/discovery/v1/coffeecorner?organization_name=Care*
```

Matching is case-insensitive and an `*` can be used as wildcard. (Similar to the SQL `LIKE %`)

Some valid examples for query parameters:

- `credentialSubject.givenName=John`
- `credentialSubject.organization.city=Town`
- `credentialSubject.organization.name=Hospital*`
- `credentialSubject.organization.name=*clinic`
- `issuer=did:web:example.com`

See API

Result:


```
[
 {
  "id":"did:web:example.com:iam:657f064a-ebef-4f0f-aa87-88ed32db3142#049fb56e-1ba8-4e9e-a7af-0071342a1378",
  "credential_subject_id":"did:web:example.com:iam:657f064a-ebef-4f0f-aa87-88ed32db3142",
  "vp":...
  "fields": {
   "organization_name":"Care Bears"
  },
  "registrationParameters": {
   "authServerURL":"https://example.com/oauth2/other_subject_identifier",
   "fhir": "https://api.example.com/fhir",
   "contact": "alice@example.com"
  }
 }
]
```

As you can see in the search results:

- `organization_name` has been added because of the constraint mapping from the presentation definition.
- `registrationParameters` from the service activation are included.
- `authServerURL` has been added autmatically. You need this for the access token request.

Access tokens

After finding a service endpoint to interact with, it's time to request an access token. You can request one via your own Nuts node:

```
POST /internal/auth/v2/my_subject_identifier/request-service-access-token


{
  "authorization_server": "https://example.com/oauth2/other_subject_identifier",
  "scope": "coffee",
  "token_type": "Bearer",
  "credentials": [...]
}
```

- The `authorization_server` parameter is taken from the `authServerURL` registration parameter from the search result.
- The scope is determined by the use case.
- The `token_type` by default is `DPoP`, you can also choose for `Bearer`.
- You can pass additional holder credentials via `credentials`. This is a way to embed user identity tokens.

The `scope` is mapped by a policy file to a presentation definition. Policy files are provided by the use case. If your wallet contains the correct credentials according to the presentation definition, an access token will be given:

```
{
  "access_token": "ciOiJSUzI1NiIsInR5cCI6Ikp",
  "token_type": "Bearer",
  "expires_in": 3600,
}
```

The `access_token` can then be put in the HTTP Authorization header.

Authentication & authorization

When a request comes in at your resource endpoint, an access token should be available in the HTTP Authorization header. You can validate the token by calling:

```
POST /internal/auth/v2/accesstoken/introspect


token=ciOiJSUzI1NiIsInR5cCI6Ikp
```

Note: the content-type of this call is `application/x-www-form-urlencoded`.

And the result:

```
{
  "active":true,
  "iss": "https://example.com/oauth2/other_subject_identifier",
  "client_id": "https://example.com/oauth2/my_subject_identifier",
  "scope": "coffee",
  "organization_name":"Care Bears"
}
```

Like with the discovery service, any constraint in the presentation definition of the policy file is also added as key/value pair to the introspection result. This is the mechanism to use attestations from presented credentials for authorization purposes.

The token introspection result is the last thing the Nuts node can do for you. From this point you have to apply the authorization policy...