

Deploy

Deploy the Nuts node and, optionally, Nuts Admin. Put the node behind a reverse proxy with the right TLS certificates, and check that it's reachable.

- [Deploy the Nuts node](#)
- [Endpoints and TLS certificates](#)
- [Deploy admin app](#)
- [Configure url rewrite for OAuth](#)

Deploy the Nuts node

There are two ways to run the Nuts part of your system. Choose one:

Option	What it is	When to choose it
Nuts node (default)	The Nuts node on its own. You build the integration with your EHR yourself, following this book.	You want full control, or you already integrate with the Nuts node
Nuts Knooppunt	Has an embedded Nuts node, plus components for integrating with the MinVWS Generic Functions (Generieke Functies), such as a policy decision point (PDP) for access decisions and Mitz integration	You want more ready-made parts, or you integrate with the Generic Functions. See the Knooppunt section below

One Nuts node per vendor (and per environment) hosts all of the vendor's care organizations.

Nuts node

The [Nuts node documentation](#) describes deployment in detail. Start with the [recommended deployment](#) and [running on Docker](#).

Install

The Nuts node is published as:

Form	Where
Docker image	<code>nutsfoundation/nuts-node</code> on Docker Hub
Helm chart, for Kubernetes	Helm repository <code>https://nuts-foundation.github.io/nuts-node</code> , chart <code>nuts-node-chart</code> (source)
Binaries (Linux, amd64 and arm64)	GitHub releases

Use a fixed version (e.g. `nutsfoundation/nuts-node:<version>`), not `latest`. For the latest version, see the [GitHub releases](#). Docker images are signed; see [verifying image signatures](#).

Configuration

The settings you need for a spec v2 use case. Each can be set in `nuts.yaml` or as an environment variable (`NUTS_` + the name in capitals, with `_` instead of `.`, e.g. `NUTS_HTTP_INTERNAL_ADDRESS`).

Setting	Value	Why
---------	-------	-----

<code>url</code>	Your node's public URL, e.g. <code>https://nuts.example.com</code>	Used in your did:web DIDs and OAuth2 URLs. Must be HTTPS.
<code>strictmode</code>	<code>true</code> (default)	Forbids insecure settings. Only turn it off on development or test systems without production data.
<code>didmethods</code>	<code>web</code>	Spec v2 only uses did:web (see "Create a subject").
<code>http.internal.address</code>	e.g. <code>:8081</code>	The internal API. Default <code>127.0.0.1:8081</code> , which can't be reached from outside the container.
<code>storage.sql.connection</code>	Connection string to PostgreSQL, MySQL or SQL Server (including Azure SQL)	The default, SQLite, isn't meant for production.
<code>crypto.storage</code>	<code>vaultkv</code> (HashiCorp Vault) or <code>azure-keyvault</code>	Where private keys are stored. <code>fs</code> (on disk) is only for development.
<code>discovery.definitions.directory</code>	Directory with the Discovery Service definitions	See chapter "Configure the node for a use case".
<code>policy.directory</code>	Directory with the policy definitions	See chapter "Configure the node for a use case".

Network

- Put a reverse proxy in front of the node that terminates TLS. See "Endpoints and TLS certificates" for which endpoints to expose, and with which certificate.
- Never expose the internal API (`:8081`) to the internet.
- If the node runs on a subpath (e.g. `https://example.com/nuts`), rewrite the `.well-known` URLs. See "Configure url rewrite for OAuth".

Check that it works

- `GET <internal Nuts interface>/health` returns `200 OK` with status `UP`.
- After creating your first subject, check that its DID resolves from the internet (see "Create a subject").

Nuts Knooppunt

Knooppunt has an embedded Nuts node, so the steps in this book still apply. It also has a policy decision point (PDP) that helps you decide whether to allow an inbound query (see "Validate the access token").

See the [Knooppunt documentation](#), in particular:

- [Deployment](#)
- [Configuration](#)

- [Integration](#), including the PDP
- An example [docker-compose.yml](#)

Endpoints and TLS certificates

Which endpoints you expose, to whom, and with which TLS certificate. This repeats the [recommended deployment](#) for spec v2 use cases; see there for details.

Nuts node endpoints

Your reverse proxy terminates TLS for the public endpoints.

Endpoint	Interface	Who uses it	Protection
<code>/iam</code>	Public (<code>:8080</code>)	Other parties resolving your did:web DID documents	HTTPS with a publicly trusted server certificate
<code>/.well-known</code>	Public (<code>:8080</code>)	Other parties: DID documents, OAuth2 metadata	HTTPS with a publicly trusted server certificate
<code>/oauth2</code>	Public (<code>:8080</code>)	Other Nuts nodes requesting access tokens from you	HTTPS with a publicly trusted server certificate
<code>/statuslist</code>	Public (<code>:8080</code>)	Parties checking whether credentials you issued are revoked	HTTPS with a publicly trusted server certificate
<code>/discovery</code>	Public (<code>:8080</code>)	Only if your node runs a Discovery Service server	HTTPS with a publicly trusted server certificate
<code>/internal</code>	Internal (<code>:8081</code>)	Your own applications, Nuts Admin	Never public: restrict access through network separation
<code>/status</code> , <code>/status/diagnostics</code> , <code>/health</code> , <code>/metrics</code>	Internal (<code>:8081</code>)	Monitoring, health checks	Restrict access through network separation
<code>/n2n</code> , <code>/public</code> , gRPC (<code>:5555</code>)	—	Spec v1 (did:nuts) only	Don't expose them

Your data endpoints

Protect your FHIR API (or other data API) with mutual TLS (mTLS) based on PKIoverheid certificates: both your server and the calling vendor's client authenticate with a PKIoverheid certificate. Check the use case's IG for what's allowed in test environments.

Don't mix up the certificates

Certificate	Used for
UZI server certificate	Creating the care organization's X509Credential (see "Get an X509Credential from a UZI certificate")
Publicly trusted TLS certificate	The Nuts node's public endpoints
PKIoverheid certificate	mTLS on your data endpoints

Deploy admin app

Nuts Admin is an optional web application for managing your Nuts node by hand: create subjects, load and issue credentials, and register on Discovery Services. It's useful for testing and for small numbers of care organizations. For many care organizations, integrate with the node's internal API instead, as described in the rest of this book.

Run it

```
docker run -p 1305:1305 -e NUTS_NODE_ADDRESS=http://nutsnode:8081 nutsfoundation/nuts-admin:<version>
```

- `NUTS_NODE_ADDRESS` points to the node's **internal** API.
- Nuts Admin listens on port `1305` (change it with `PORT`).
- For the latest version, see the [tags on Docker Hub](#).

See the [Nuts Admin README](#) for all settings.

Secure it

Nuts Admin passes requests on to the node's internal API. Anyone who can reach Nuts Admin can manage your node.

- Turn on user login with OpenID Connect (`NUTS_OIDC_ENABLED=true`), plus the settings in the README), or
- restrict access to Nuts Admin in another way, e.g. network separation.

Never expose Nuts Admin to the internet without login.

Configure url rewrite for OAuth

Other parties look up your node's OAuth2 metadata at a `.well-known` URL, which always starts at the root of the domain ([RFC 8615](#)). If your node runs at the root of its own (sub)domain, e.g. `https://nuts.example.com`, you don't need to do anything.

If the node runs on a subpath, e.g. `https://example.com/nuts`, your reverse proxy must forward the OAuth2 metadata URL to the node. Example for a node on `/nuts`:

Other parties request	Forward to the node as
<code>https://example.com/.well-known/oauth-authorization-server/nuts/oauth2/<subject ID></code>	<code>/.well-known/oauth-authorization-server/oauth2/<subject ID></code>

In short: remove the subpath (`/nuts`) after the `.well-known/...` part.

The node serves all its other endpoints at the root too (`/iam`, `/oauth2`, `/statuslist`, ...). So your proxy must also remove `/nuts` from those, e.g. forward `https://example.com/nuts/oauth2/...` to `/oauth2/...`.

NGINX

```
location /.well-known/oauth-authorization-server/nuts/ {  
    proxy_pass http://nuts-node:8080/.well-known/oauth-authorization-server/;  
}
```

Azure Application Gateway

```
path: '/.well-known/oauth-authorization-server/nuts'  
rewriteCondition: '/.well-known/oauth-authorization-server/nuts(.*)'  
rewriteRule: '/.well-known/oauth-authorization-server{var_uri_path_1}'
```

Check that it works

From outside your network, open:

```
https://example.com/.well-known/oauth-authorization-server/nuts/oauth2/<subject ID>
```

It should return JSON with your node's OAuth2 metadata, e.g. an `issuer` of `https://example.com/nuts/oauth2/<subject ID>`.