CERN-Solid code investigation

Jan Schill

IT University of Copenhagen, Copenhagen, Denmark
schi@itu.dk

Abstract.

1 Introduction

2 Introduction Solid

The Web was created in 1989 by Tim Berners-Lee while working at CERN "[...] to allow people to work together by combining their knowledge in a web of hypertext documents" [1].

This brilliant idea has ever since grown as an essential part of our all lives. While it has given a new platform for all types of innovation, it has also evolved away from the initial idea of sharing knowledge freely. A new term has been coined describing the phenomena of isolating data from the public by creating the so-called *data silos*. The data in these silos is then only available to the organization controlling the application.

A multitude of problems reside with this, like the actual content creator not owning their own data, nor having full access to it. Another drawback is that the application owners decide what interfaces are publicly accessible, therefore, not allowing users easy migrations of their data.

This results in one user having to provide the same information to different applications: username, name, age and others depending on the domain. The same problem applies to traditional web applications when authenticating their users. Usually, applications will do the authentication themselves, but there are initiatives that decentralize this authentication, which is called single sign-on (SSO). Solid is aiming at solving these problems by standardizing an ecosystem where data is stored on data pods chosen and fully controlled by the users/agents, where they can decide who has access to what data; Linked Data is utilized to create interoperable data, for seamless migration between applications and pods; the user authenticates with one identity provider (IDP) to use multiple Solid applications with one username and password combination.

3 Overview of Solid

In Solid data is stored on personal and through the Web-accessible storages, these are called *data pods*. Data pods are personal in the sense of users configuring the access control to the data on their pods themselves. Web-accessible because the pods can be connected to as long as a connection to the Web exists and the correct access controls are given. Users or agents can freely choose from their favorite pod provider where they would like to store their data. As of writing this there are two major providers online, inrupt.net[1] and solidcommunity.net[2]. These providers are also used as IDPs to enable decentralized authentication, which shall be looked at more closely in a moment. The data pod storage architecture follows the Linked Data server specifications. It enables hierarchical resource discovery in a RESTful manner, where the path of the Uniform Resource Identifier (URI) gives information about the relation of its underlying data. Up until a URI does not end with a / character, every path segment resembles a container. A container is the collection of multiple resources. Resources are the data items stored on a pod. Every container holds information of the access control in form of an access control list (ACL) and information of what resources it contains. Both of these resources are returned in an Resource Description Framework (RDF) compliant format, mostly Turtle. The data pod differentiates between two resource types: RDF and binary/text. RDF is a framework to represent data on the Web. The basic structure of it follows a graph representation, where two nodes, the subject and object, are connected by an edge, the predicate. This structure is called a *triple*. In RDF, nodes and edges elevate the benefits of URIs, more specifically Internationalized Resource Identifier (IRI) – which are a generalization of URIs, offering more Unicode characters – by either using them as globally unique identifiers or globally unique and reusable property names. This allows interoperable data by reusing schemas with agreed-upon vocabulary to describe data and can be used to obtain more information by dereferencing the IRI.

```
@prefix jan: <https://janschill.solidcommunity.net/profile/card> .
@prefix schema: <http://schema.org/> .

jan:me schema:familyName "Schill" .
```

The other crucial part of Solid is decentralized authentication. This is realized with WebIDs. It works in a way that users need a globally unique identifier—a WebID URI—which can be used with every Solid application to identify an agent. These WebIDs are handed out by IDPs, which in most cases are also data pod providers. A WebID encompasses a profile document, describing the person in more detail who is the referent of the WebID URI.

[1] https://inrupt.net
[2] https://solidcommunity.net

WebID URI
https://janschill.solidcommunity.net/profile/card#me

Profile Document
https://janschill.solidcommunity.net/profile/card

Solid OpenID Connect (Solid OIDC) is the standard that is being used to authenticate within the Solid ecosystem.

TODO: go deeper into Solid OIDC

Several implementations exist to this day, that all do parts of the described solutions, but there no existing complete solution yet. The Node Solid Server (NSS) is the original implementation and is to this day the most complete open-source solution measured by passing Solid Test Suite cases. It is being deployed onto both providers 'inrupt.net' and 'solidcommunity.net', acting as a data pod and IDP. The Community Solid Server (CSS) is a new project aiming at replacing the NSS to become the new official open-source implementation of the Solid specifications. It is currently in beta version and actively developed. The Enterprise Solid Server (ESS) is Inrupt's commercial closed-source solution launched late this year 2020.

A lot of different libraries are built to enable development in the ecosystem. A subset as an example are client-side libraries for authentication with data pods; reading and writing RDF based resources; an SDK for React development. Additionally, efforts are put into the development of a Solid operating system (SolidOS), which can be deployed onto the data pod and supports the browsing of one's pod, editing files, parsing, and showing the data in a meaningful manner and other useful additions, such as Solid Panes, which is a set of Solid-compatible apps. These are useful for the richness of SolidOS. The core idea is to enable the operating system of Solid to bring an interface to the diverse group of linked data on a data pod. One core example is, that it aims at allowing a sensible representation of objects, which can extend to an address book showing all contacts of a user.

TODO: Show case some existing Solid solutions and talk about the government efforts in Flanders, Belgium and UK

4 Introduction CERN

CERN or the European Organization for Nuclear Research is the largest particle physics laboratory in the world, with its main site in Switzerland. It maintains the world's highest energy accelerator, the Large Hadron Collider, and houses a broad scientific program. With staff members, users, collaborating scientist and students from all around the globe it accounts to a total of 12,500. Great challenges arise when managing this number of scientists and their experiments. CERN has closed this gap by creating and maintaining several reliable software projects and a robust infrastructure.

A number of relevant open-source applications shall be introduced in the next section.

5 Overview of CERN

The following open-source software systems have proven to be of excellent operational quality while serving tens of thousands of users with a wide array of functionality.

CERN has met in the past difficulties to fulfill some core workflows in the authentication and authorization in their infrastructure expected by users of their platform. The new **CERN Authorization Service** is a centralized authentication and authorization service with components such as single sign-on, group management API, account management and computing resource lifecycles [2].

Invenio consists of three different parts. The **Framework** is package for building large scale digital repositories. Its main goals are scalability, security and long-term preservation of data. The other two parts RDM and ILS are not yet released but actively developed with planned releases in the year of 2021.

ILS is an integrated library system allowing cataloging with structure bibliographic records, a circulation workflow and much more in a modern user interface.

RDM stands for research data management and aims at opening a platform for researchers to share and preserve their research results.

Zenodo is a small layer on top of the Invenio Framework. The goal with Invenio RDM is to build a common RDM-platform from everyone can profit. Invenio RDM will be based on Zenodo and once done, Zenodo will be migrated over [3].

Indico is one of CERN's most sophisticated software projects. It is an event management tool, giving users a tool to organize complex meetings or conferences with an easy-to-use interface. It was started in 2002 as a "European project" and has been in production at CERN ever since. It is used daily to facilitate more than 600,000 events at the organization itself and has helped others like the UN "to put in place an efficient registration and accreditation workflow that greatly reduced waiting times for everyone" at conferences with more than 180,000 participants in total [4].

Indico is actively worked on by a team of six developers from CERN. Its open-source approach allows external participation. It is written in Python and is currently on version 2.7 and plans to move onto Python 3.x soon coupled with the release of Indico 3.0 scheduled to be released at the end of this year. Indico version 2.3 was released earlier this year—during the summer of 2020—updating the software with multiple features

with the help of the community, the team managing Indico at the United Nations Office at Geneva and funding from IEEE.

Indico is the proof of concept (POC) candidate for the after this research project followed Master's thesis. It is a suitable contender for applying the Solid principles as it is one of CERN's most reliable applications with a long history of operation. It does carry any incentives in for example its conference registration module. This part is responsible for administering the storage (and other necessary parts) of the given data from the attendee of a conference. The host of a conference decides what information is necessary to register. The information can go as far as being digital copies of physical identifications. This would be an ideal use-case to apply the Solid principle of decentralized storage on a data pod owned by the attendee.

CERN, being the birthplace of the Web, remains a High Energy Physics laboratory, hence, its main mission is to run an accelerator, its detectors and the relevant experiments. Computing is of paramount importance for filtering, storing, distributing, accessing, analyzing the experimental data. Nevertheless, due to its large and distributed user base, CERN has to offer sophisticated solutions on all software application fronts. Proprietary packages having been disappointing, in terms of price and transparency, CERN, following the rising world-wide awareness of personal data ownership and sovereignty, is interested in being part of Solid.

6 Evaluation of the Specifications

Initially reviewed the document: Editor's Draft, 13 November 2020
Revisited and partially updated: Editor's Draft, 4 December 2020

The Solid Ecosystem[3] is a by the Solid editorial team[4] published technical report. It is the official rewrite of the informal Solid specification[5], which was initially used to define the architecture of Solid servers and clients. This rewrite is still incomplete and being worked on continuously.

6.1 Summary

The Solid Ecosystem combines a set of carefully selected specifications that were adopted or newly defined, to bring together an architecture that aligns the principles and values of Solid. These components are loosely coupled, can therefore evolve as independently as possible, to ensure flexibility and robustness [5].

The main specification starts off by describing how a data pod and a Solid app should be implemented using the Hypertext Transfer Protocol (HTTP) protocol.

A data pod is a web server that responds to HTTP requests and returns HTTP responses. Its purpose is the storage of data and the management of who has access to this data.
A Solid app is a client that is sending requests to a data pod. It should be able to read and write depending on the access control to a data pod.

The URI plays an essential role in the Solid Ecosystem, for it is being used to identify users with WebID, with resources in the Linked Data Platform (LDP) and more generally give information about the hierarchy of stored information on the data pod.

A container resource is an organizing concept in the LDP [6]. It stores linked documents or information resources, which handle requests from clients for their creation, modification, traversal of the linked documents [6].

An auxiliary resource exists to give additional information, like configuration, processing, or interpretation about a Solid resource, for example: "A container linked to an auxiliary resource that includes access control statements for that container and the resources that belong to it" An example "A binary JPEG image linked to an auxiliary resource that includes information describing that binary JPEG." clarifies the necessity clearer, as a binary JPEG image does not carry any machine-readable information.
The ACL in Solid is realized with Web Access Control (WAC). The section for WAC is not yet written in the Solid specification but shall be given a short introduction.

WAC is similar to access control schemes used in file systems. Files, users, and groups are referenced by URLs. Users in particular are identified by WebIDs. Its functionality is cross-domain and can therefore have an ACL resource – holding the permissions for an agent – on domain A, while setting the permissions for a file on domain B. The supported modes of operation are read, write, append and control. Read and write are self-explanatory, whereas append and control introduce two interesting modes. Append allows the agent to add files to a container, without being able to read or write any of the container's files. The idea of an email inbox can be compared to this functionality. Control means that the agent with this permission has access to the ACL resource and can modify it.

As mentioned, Solid follows the specifications of the LDP to define its storage mechanism. In LDP resource representation is realized with RDF. Therefore, all resources that are created are Linked Data Platform

[3] `https://solid.github.io/specification/`
[4] `https://github.com/solid/process/blob/master/panels.md`
[5] `https://github.com/solid/solid-spec/`

Resource (LDPR) and in the Turtle format.

A WebID is an HTTP URI that denotes an agent on the Web. It is used as the primary agent identification in the Solid Ecosystem.

When making requests to a Solid server to create a resource on the server, HTTP POST, `PUT` or `PATCH` can be used. If the client wants to associate a specific URI with a resource, `PUT` or `PATCH` needs to be used. With the HTTP method `GET`, `HEAD OPTIONS` information about a resource can be requested. To remove a resource from the server the `DELETE` method can be used. A server must create all intermediate containers and containment triples according to `PUT` and `PATCH` requests.
On a `POST` request to `/`, the server needs to create a resource under `/slug`.
On a `POST` request to `/slug/`, the server needs to create a container for `/`.

Authentication in the Solid Ecosystem is supported through two ways. Solid OIDC is the Solid specific implementation of the widely used OpenID Connect (OIDC). The alternative, but not from the specification preferred method is WebID-TLS. OIDC enables the decentralized authentication and SSO mechanism needed for Solid. In Solid OIDC one key aspect is that the `Client ID` should be a WebID. The `Client ID` is needed in OAuth(/OIDC) for a `Client application` to identify itself with the IDP and resource server. Once authenticated with a username and password combination by an IDP, all Solid applications that need authentication will redirect to the chosen IDP. The browser uses the stored token from a set cookie to identify and is then able to use the application without login.

6.2 Comments

The Solid Ecosystem does a good job in the claims from the beginning. It does not go into best practices on how to build a Solid server or client, but solely focusses on the clear definition on what Solid is when looked at technically. Other documents like Linked Data Primer and Best Practices are written to describe common patterns in the development with Linked Data. This would also be also of value for the Solid Ecosystem. Further, the review process seems sophisticated and lively in its discussion. Contributions to the specifications are heavily discussed using the GitHub issue and pull request features, but also chat platforms like Gitter[6]. A review of such a contribution follows strict regulations. A contribution is encouraged to come with a sophisticated explanation on why this change is appropriate. Each topic within the specifications have editors to them assigned who are responsible. Because Solid is open-source and therefore benefits from an active contribution from all parties, it is highly recommended to participate in its development.

Clearly stating that the Solid Ecosystem document has its purpose in defining the implementation requirements for a data pod and makes suggestions to other documents that do a thorough job on speaking out use-cases and best-practices is a good structural decision.

No Justification For the Usage Of Linked Data Even though it might not be the proper place to explain the reasons for choosing specific technologies like Linked Data—as those discussions happen prior to defining the technologies in the documentation—but it seems some clarifications why Linked Data as a technology is being used for data representation might be valuable beyond just stating that is used because of "resource discovery and lifecycle management" [5].

Limited Information on Solid Client Section 2.1.2[7] in the specifications goes into the requirements for a Solid client implementation and is limited in its details. It only states it needs to be an `HTTP/1.1` client, must implement the HTTP Authentication framework[8] and the `Content-Type` HTTP header for `PUT`, `PATCH`, and `POST` requests. From a commit[9] to the Solid specification repository it can be assumed that a section for client implementation was planned, but reprioritized and delinked from the main document. A lot of Solid clients exist and of course the Solid ecosystem – as stated in the beginning – is not a document for best-practices, it would be highly beneficial to have such documents explicitly giving good implementation details for developers. This is a remark to the missing supporting documents for the Solid Ecosystem, such as the existing ones for the LDP. Section 7 "Evaluation of Solid Implementations" will look more closely at existing solutions on the server and client-side.

Incomplete Supporting Documents The Solid Ecosystem uses not only its own specifications, but also external supporting specifications and capitalizes on sophisticated technologies like the HTTP. But it also references some technologies that have not been around for as long as HTTP, like WebID. WebID in itself is also defined in an incomplete technical report. It being incomplete as well, creates a chain of uncertainty towards their definitions. In the case of WebID it might not be crucial, as it is a straight-forward specification, but the supporting document of Solid OIDC is also fairly new specification, where only time and implementations tell its readiness. If a missing section in the Solid Ecosystem links to an external specification, one could use that document as a source of truth, but if it is also incomplete, the risk of building something that becomes inaccurate increases.

[6] `https://gitter.im/`
[7] `https://solid.github.io/specification/#http-client`
[8] `https://httpwg.org/specs/rfc7235.html`
[9] `https://github.com/solid/specification/commit/d387e332f3bbc9af8e7ad596fa742530262a76a9/`

Users Have Too Much Control WAC allows the owner of a pod to configure his access controls. With Solid gaining more popularity the user base grows with it and also the diversity in technical proficiency. Having full control over the ACLs a minor mistake in giving a malicious person root access could yield catastrophic results. Therefore, to make a data pod more user friendly this should be addressed. Proposals such as acccess control policiess (ACPs) are being discussed and wanted in the specifications, but are not written and merged in yet[10].

Complexity Even though the document does a great job in going into detail on specific areas, it is still demanding to follow with only limited knowledge in web technologies. This can be justified by the incomplete status of the document, but also by its complex nature with many areas that need to be studied. One example of this is the concept of Linked Data and all its components. It cannot be assumed of the Solid Ecosystem to explain all of its linked concepts – as it would render the document redundantly convoluted – but the fact remains that it is challenging to follow.

Incomplete Draft Due to the fact that the specifications are work in progress and even some crucial *sub-specifications*, like WAC existing draft[11], are not even started, makes a review challenging as the documents are subject to additions, removals, or changes. Even though it can be assumed the general direction of its underlying principles does not change. An application developed to the rules of today's Solid rules could result in the same application not conforming to tomorrow's set of rules.

Edit: as of lately 05.12.2020 the WAC section has been added.

6.3 Conclusion

As of now the specifications are to be seen as *almost complete*, the incomplete sections were either finished or removed. There are still some areas of improvements, but so far nothing major. But what is completely left out so far is, that the specifications is not only the published document in itself, but also the work around it. Actual implementations of the specification that are tested against the specification, applications that consume Solid servers and practice the Solid principles with the help of Solid servers. To call the specification ready would therefore not help anyone. It is now up to the development to seek shortages in the definitions in the specs.

A rough estimate from a Solid developer and spec writer is the second quarter of 2021 to be the time when a *call for implementations* will be officially made. Enough confidence in the specs will be gathered up until then for it to have enough stability to not introduce breaking-changes and be technically and ethically mature.

7 Evaluation of the Solid Implementations

The ecosystem of Solid is already diverse in existing implementations. Attempts to transfer the Solid specifications into software have been carried out with different programming languages and to different completion levels. These servers or data pods have different goals in mind and even though a server needs to adhere to the specifications it does not make them the same. In the following, various existing Solid servers shall be looked at. In the second part of this section, libraries for development in the ecosystem and actual developed Solid applications will be evaluated.

7.1 Solid Servers

A Solid server is a web server enabling storage through data pods and may optionally offer IDP implementation as well [7]. In Solid a server only needs to enable the authentication through Solid OIDC, which requires an IDP, if this IDP is controlled by the user through the usage of an existing Solid server that is hosted on their own infrastructure or they are using an identity-as-a-service vendor is up to them [5].

Node Solid Server The original Solid server was developed at the Massachusetts Institute of Technology (MIT) by PhD students. This server is still to this day the only server that passes most test cases of the Solid Test Suite[12], which is a set of checks developed to test an implementation against the Solid specifications. The Test Suite for Solid is also still in development and constantly extended by more tests for the different categories of a Solid server. This server is completely open-source, written in JavaScript with the help of the web framework Node.js[13] and is commonly referred to as NSS. NSS implements a pod server and and IDP, meaning users can register a WebID, create a data pod and authenticate with it. `https://inrupt.net/` and `https://solidcommunity.net/` [14] are currently the two domains hosting the NSS and allowing users to register and use these services.

Because NSS was started as a research project, the code base was subject to a lot of experiments. These experiments were sometimes successful and improved the server experience by implementing useful

[10] `https://github.com/solid/authorization-panel/blob/2d80b870dd0f71ae1d89a2dda908554687cde553/proposals/acp/index.md`
[11] `https://www.w3.org/wiki/WebAccessControl`
[12] `https://github.com/solid/test-suite/`
[13] `http://nodejs.org/`
[14] `https://solidproject.org//users/get-a-pod`

functionality, but sometimes it would also introduce vulnerabilities or not yield the expected outcomes. Often these implementations were not completely well-designed or made self-contained resulting in code that was hard to remove and therefore just left in. This increased its complexity to a level, where it is difficult to find enthusiastic developer to maintain the implementation.

Community Solid Server The CSS is the from the Solid community-driven development of new open-source software to provide a way for everyone to host a data pod. It aims at giving developers the opportunity to create new Solid apps and also test them against a working implementation of the Solid specifications, while making sure no legacy code from older experiments influence the testing, such as in NSS.

Another key feature of CSS is its modular architecture. Because Solid is just in the beginning and there is still a lot of different ideas and a road map full of features for the future, CSS tries to enable by high cohesion and loose coupling in its modules a highly flexible platform for easy integration of experiments and new ideas. That can be implemented without altering existing code by plugging the experiment in as a self-contained module, which can just as easily removed again. This is to prevent the same mistakes NSS went through.

This is where the decision was made to rewrite an open-source Solid server. Inrupt is sponsoring this development with two imec researchers and one developer. On December 3rd, 2020 the first beta version of CSS was released. This marks a significant milestone in the journey for open-source Solid servers. Developers working in the Solid ecosystem are encouraged by the core developers of CSS to switch over to CSS when developing new applications. This prepares the new applications to work with in the future available open-source servers, but also gives the opportunity to spot bugs or features that have not been accounted and therefore help with the progress of CSS.

One of the greatest benefits of the development of CSS now is the Solid specifications are in a much more mature state they were when the development of NSS started. In-fact no such specifications existed and the experimental nature of Solid back then harmed the quality of the software.

Not only can it be developed against a mature specification, but also has a test-suite constantly checking if the development of CSS adheres to the specifications. This increases the quality of the code substantially because it does not depend on manual checks if it is still on track with the specifications. The Solid Test-Suite is a sophisticated collection of test cases to make sure an implementation complies with the specifications. The test-suite is not yet complete and still lacks in the category of access control policies. It is not crucially consequential, as it is the alternative to the Solid preferred WAC, which is on the other hand well covered.

The CSS language of choice is TypeScript (TS)[15]. TS is a statically typed programming language bringing strict types to the dynamic language of JavaScript (JS). The TS compiler transpiles TS source code into JS source code.

An estimate by a developer was given, that by the second quarter in 2021 CSS could be production-ready.

Enterprise Solid Server Inrupt the American-base company Tim Berners-Lee cofounded develops the ESS. It is a commercial and closed-source alternative based on Trellis[16]. Trellis is a platform to build scalable Linked Data applications in Java. In November 2020 Inrupt released the first major version 1.0. Besides developing a Solid server behind closed doors, they are also active in the open-source community, having developed applications like a PodBrowser[17], allowing the browsing of one's data in a pod or a set of libraries helping developers get started with the development of Solid applications.

Not much more of the implementation of the ESS can be evaluated. Inrupt does offer a practical journey for new customers, where access is given to the server with introductions to the open-source developer tools or a well-defined and in great detail outlined roadmap containing the design of a proof of concept, proof of value, pilot stage and ready for production with a service level agreement (SLA). Considering the untrustworthiness of the NSS and that any open-source solution of a Solid server will not come with any guarantees of bug fixes within a timely manner.

PHP Solid Server The standalone PHP Solid Server is a project from PDS Interop[18] funded by the NLnet foundation[19]. It is actively maintained and passes a good amount of cases from the Solid Test Suite, even more than the NSS in the area of CRUD. Besides the standalone version PDF Interop is also developing a plugin[20] for Nextcloud[21]. Nextcloud is an open-source collection of client-server software enabling file hosting services. The plugin makes Nextcloud compatible with Solid

Hosting a Solid Server NSS is the most complete server in terms of passing the Solid Test Suite. It currently used by Inrupt and the Solid Community to offer free data pods for development, experiments and to get familiar with Solid. NSS was also used to set up an own instance on the `janschill.de` domain.

This write-down mostly follows this guide[22] from the official Solid website, the documentation in the repository[23] of the NSS.

[15] `https://www.typescriptlang.org`
[16] `https://www.trellisldp.org`
[17] `https://inrupt.com/products/podbrowser/`
[18] `https://pdsinterop.org/`
[19] `https://nlnet.nl`
[20] `https://github.com/pdsinterop/solid-nextcloud`
[21] `https://nextcloud.com/`
[22] `https://solidproject.org/for-developers/pod-server/`
[23] `https://github.com/solid/node-solid-server/`

Web Server Before installing the NSS, a physical web server, preferably running a Linux distribution is needed. A domain should be configured to point to this web server. This can be done at the DNS hosting and domain name registration service that holds the domain. The domain that will be used in this example is `janschill.de`.

Digital Wildcard Certificates NSS uses instead of a subdirectory approach a subdomain one to create the space for an isolated user pod. This means a new user registers and gets a pod location at the address https://username.janschill.de/ and not https://janschill.de/username/. This is a design decision and there has been some [discussion](https://github.com/solid/node-solid-server/issues/1349) about moving or allowing the setting of the latter. There are benefits and drawbacks to these approaches that shall not be discussed in this context. One drawback of this needs to be addressed – as it is essential for this setup. It is the need for wildcard certificates[24]). This is only a drawback, if a developer has not heard about this concept or has never set up digital certificates in general, as the process is quite straightforward. In short a wildcard certificate allows a certificate to be used with multiple subdomains and is created with `certbot`, a program offered by Let's Encrypt, as follows:

Listing 1.1. Certification issuing with certbot.

```
# Install certbot
apt install certbot
# Issue certificate
certbot certonly \
--manual \
--preferred-challenges=dns \
--email schill@hey.com \
--server https://acme-v02.api.letsencrypt.org/directory \
--agree-tos \
-d janschill.de -d *.janschill.de
```

This command shows that a certificate for the 'janschill.de' domain is created, but also for the wildcard '*.janschill.de' domain. It means any string in front of the 'janschill.de' domain, separated by a period, is allowed and will have a valid certificate.

- 'certonly' obtains or renews a certificate, but does not install it (it does not edit any of the server's configuration – this will be done manually in the next step).
- 'manual' will make the process of obtaining the certificate interactive.
- 'preferred-challenges=dns' this is a challenge that needs to be successfully completed in order to get certificates. Let's Encrypt will not allow HTTP-01 challenges for wildcard certificates. Therefore, DNS is set for the DNS-01 challenge ([Challenge types](https://letsencrypt.org/docs/challenge-types/)).
- 'email' which will be used to send important notifications.
- 'server' the address 'certbot' will connect to.
- 'agree-tos' agree to the server's Subscriber Agreement.

DNS-01 challenge asks to prove the control of the DNS for the specified domain. This is done by placing a TXT record with a defined value under the domain name. Let's Encrypt will then verify the key and value (TXT record) by querying the DNS system. Make sure the certificate directory has the correct permissions set.

"For historical reasons, the containing directories are created with permissions of 0700 meaning that certificates are accessible only to servers that run as the root user. If you will never downgrade to an older version of Certbot, then you can safely fix this using chmod 0755 /etc/letsencrypt/live,archive" [8]

Listing 1.2. Set permissions.

```
chmod -R 755 /etc/letsencrypt/live
```

Why are digital certificates needed in the first place? The Solid specifications say that: "A data pod SHOULD use TLS connections through the https URI scheme in order to secure the communication between clients and servers" [5]. Therefore, the NSS makes it mandatory to provide the location of a valid certificate when started.

Reverse Proxy A reverse proxy allows a server to run multiple services on the same port. A reverse proxy receives the initial request on the host and port and then forwards it to the configured local service on the machine. Solid has WebID-TLS implemented as one of its authentication mechanisms. A reverse proxy – when not configured correctly – does not permit the usage of this, as the client when performing the handshake with the server also [sends its certificate](https://blog.cloudflare.com/introducing-tls-client-auth/#handshakeswithtlsclientauth), which means with the usage of a reverse proxy that performs the handshake, the certificate is not sent to the Solid server, denying the possibility of authenticating properly. A solution is the correct configuration of the reverse proxy [This document](https://github.com/solid/node-solid-server/wiki/Running-Solid-behind-a-reverse-proxy) introduces this issue and a few solutions to it. Therefore, the same Nginx configuration with necessary steps to set up can be found [here](https://solidproject.org/for-developers/pod-server/nginx):

1. Open the default configuration after installing Nginx

[24] `https://en.wikipedia.org/wiki/Wildcard_certificate`

Listing 1.3. Installing and editing Nginx.

```
sudo apt update
sudo apt install nginx
vi /etc/nginx/sites-available/default
```

2. Configuration for the reverse proxy

Listing 1.4. Nginx configuration file.

```
server {
  listen 0.0.0.0:80;
  listen [::]:80;
  server_name janschill.de;
  server_tokens off;

  return 301 https://$http_host$request_uri;

  access_log /var/log/nginx/solid_access.log;
  error_log /var/log/nginx/solid_error.log;
}

server {
  listen *:443 ssl;
  listen [::]:443 ssl;
  server_name janschill.de;
  server_tokens off;

  access_log /var/log/nginx/solid_ssl_access.log;
  error_log /var/log/nginx/solid_ssl_error.log;

  ssl_certificate /etc/letsencrypt/live/janschill.de/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/janschill.de/privkey.pem;

  root /var/www/janschill.de;

  add_header Strict-Transport-Security "max-age=31536000; includeSubDomains";

  location / {
    proxy_pass https://localhost:8443;

    gzip off;
    proxy_redirect off;

    proxy_read_timeout 300;
    proxy_connect_timeout 300;
    proxy_redirect off;

    proxy_http_version 1.1;

    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-Ssl on;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
  }
}
```

As per default the logs for the server will the written in ‘/var/log/nginx/solid_*.log‘ files.

An additional interesting part of the configuration is that it sets the ‘Strict-Transport-Security‘ header. This header instructs the user’s browser to use HTTP Strict Transport Security (HSTS) – meaning that it should use HTTPS for every request. This is beneficial as requests that are addressed to http://janschill.de, or just janschill.de will usually connect on HTTP to the server and then get redirected. This leaves an open window for a man-in-the-middle attack. HSTS solves this by instructing the browser upon first visit to always use HTTPS when connecting to janschill.de. HSTS is not perfect, as it still needs one initial request to even be able to cache the ‘Strict-Transport-Security‘ header ([Source](https://www.nginx.com/blog/http-strict-transport-security-hsts-and-nginx/)).

3. Restart the Nginx server

Listing 1.5. Restart Nginx server.

```
systemctl restart nginx
```

Node Solid Server 1. Install ‘npm‘ and ‘solid-server‘

Listing 1.6. Install NSS.

```
sudo apt update
sudo apt install nodejs npm
npm install -g solid-server
```

2. Initialize ‘solid-server‘

Listing 1.7. Configure NSS.

```
solid init
# Path to the folder you want to serve. Default is (./data)
/var/www/janschill.de/data
# SSL port to run on. Default is (8443)
8443
# Solid server uri (with protocol, hostname and port)
https://janschill.de
# Enable WebID authentication
Yes
# Serve Solid on URL path
/
# Path to the config directory (for example: /etc/solid-server) (./config)
/var/www/janschill.de/config
# Path to the config file (for example: ./config.json) (./config.json)
/var/www/janschill.de/config.json
# Path to the server metadata db directory (for users/apps etc) (./.db)
/var/www/janschill.de/.db
# Path to the SSL private key in PEM format
/etc/letsencrypt/live/janschill.de/privkey.pem
# Path to the SSL certificate key in PEM format
/etc/letsencrypt/live/janschill.de/fullchain.pem
# Enable multi-user mode
Yes
# Do you want to set up an email service (y/N)
N
# A name for your server (not required)
janschill.de
# A description of your server (not required)

# A logo (not required)

# Do you want to enforce Terms & Conditions for your service (y/N)
N
# Do you want to disable password strength checking (y/N)
N
# The support email you provide for your users (not required)
```

It is important that the configuration directories exist and have correct user permissions.

Listing 1.8. Configure directories for file creation from NSS.

```
# Create directories
mkdir -p /var/www/janschill.de/config
mkdir /var/www/janschill.de/data
mkdir /var/www/janschill.de/.db
# Give permission
chown -R 1000:1000 /var/www/janschill.de/
```

Within the directory start up the server.

Listing 1.9. Start Solid server.

```
# Change directory
cd /var/www/janschill.de
# Start server
solid start
```

Difficulties In the beginning, the thought of using Docker seemed tempting. Installing all dependencies in isolated environments gives the benefit of having all configurations as code. A Dockerfile holds all commands that are needed to set up an Nginx reverse proxy for example. Because this setup needs multiple running services (Nginx reverse proxy, certification issuing, the Solid server) that all need to communicate to each other, the Docker configuration can get easily out of control and not offer a one-click solution anymore. Docker Compose tackles this problem by offering a configuration file to easily define how these different services/container should be connected. To not reinvent the wheel and spend too much time on configuring for example an Nginx reverse proxy, well-established Docker images can be used. Existing solutions exist and can be used to set up an NSS. Unfortunately, problems occurred when the Docker images were tried, for example the wildcard certificates were not distributed correctly. Due to time constraints and the additional overhead of dealing with these extra issues, Docker was abandoned.

7.2 Solid Clients

7.3 Conclusion

The NSS is a decent foundation to get started in the realm of Solid. Setting up the server and using it is straightforward. It has been running with occasional usage on the domain for two months without problems.

No major bugs were discovered in the process so far, but it must be said the server never got pressured into a heavy load.

The CSS has not been used for any personal experiments so far. It promises a lot for the future of Solid in open-source. The architecture and quality of code seem to be well-thought-out. Defining a clear goal in the beginning and making considerations in the architecture of the implementations, having access to people that developed on the NSS to acquire learned lessons, working with a *most complete* specification and Test Suite to allow constant testing against the specification make this an opportune candidate for future work.

The ESS is an interesting product, as it is the first professionally and closed-source server currently available for production usage. Inrupt offers their customers

The Solid ecosystem is vibrant and in full motion. Server implementations are being worked on, Solid applications are starting to appear here and there and everything seems to move into a direction where everything will come together. Even though it might seem no perfect solution exists, there is still potential as the idea of Solid allows interoperability as one of the key concepts. This means if a Solid application in the form of a proof of concept is developed while using the CSS as a data pod, everything from the data pod on CSS could be easily migrated to another server and the applications would still work. It would of course be naive to just assume a flawless migration from one implementation to another, as different server implementations will never be 100% equal and theory should be validated by actual practice. Therefore, claims about a flawless migration between servers is much safer to be done after testing. This is why the specifications are so important and only the future will tell how well-defined and robust they are in the current stage, but a foundation can definitely be laid.

NSS internals are better mastered at this point in time for this report
a. because it is the oldest,
b. because it is open to and successfully passing the tests and
c. because it is used in practice by the janschill pod, which is hosted there.
Hence, like with all current implementations, one should follow the specs' evolution, seek advice in the Solid chat in Gitter and be ready to contribute and or change solutions as things evolve rapidly now.

8 Conclusion

Acronyms

ACL access control list. 1, 3, 5
ACP acccess control policies. 5

CSS Community Solid Server. 2, 6, 10

ESS Enterprise Solid Server. 2, 6, 10

HTTP Hypertext Transfer Protocol. 3, 4

IDP identity provider. 1, 2, 4, 5
IRI Internationalized Resource Identifier. 1

JS JavaScript. 6

LDP Linked Data Platform. 3, 4
LDPR Linked Data Platform Resource. 3

MIT Massachusetts Institute of Technology. 5

NSS Node Solid Server. 2, 5–10

OIDC OpenID Connect. 4

POC proof of concept. 3

RDF Resource Description Framework. 1–3

SLA service level agreement. 6
Solid OIDC Solid OpenID Connect. 2, 4, 5
SolidOS Solid operating system. 2
SSO single sign-on. 1, 4

TS TypeScript. 6

URI Uniform Resource Identifier. 1, 3, 4

WAC Web Access Control. 3, 5, 6

References

[1] Tim Berners-Lee. *Longer Biography*. 2020. URL: https://www.w3.org/People/Berners-Lee/Longer.html. (Accessed: 11.12.2020).
[2] CERN. *CERN Authorization Service*. 2020. URL: https://auth.docs.cern.ch. (Accessed: 11.12.2020).
[3] Lars Holm Nielsen. *InvenioRDM: a turn-key open source research data management platform*. 2019. URL: https://inveniosoftware.org/blog/2019-04-29-rdm/. (Accessed: 11.12.2020).
[4] CERN. *Indico*. 2020. URL: https://getindico.org. (Accessed: 11.12.2020).
[5] Tim Berners-Lee et al. *The Solid Ecosystem*. Tech. rep. W3C Solid Community Group, Dec. 2020.
[6] Ashok Malhotra, Steve Speicher, and John Arwe. *Linked Data Platform 1.0*. W3C Recommendation. https://www.w3.org/TR/2015/REC-ldp-20150226/. W3C, Feb. 2015.
[7] Adam Migus and Ricky White. *SOLID-OIDC*. Tech. rep. W3C, Dec. 2020.
[8] Certbot. *Where are my certificates?* 2018. URL: https://certbot.eff.org/docs/using.html#where-are-my-certificates. (Accessed: 11.12.2020).