

Mastering Identus: A Developer Handbook

Jon Bauer, Roberto Carvajal

2024-04-05

Table of contents

Welcome	1
Copyright	3
License	5
Book Content:	5
Applications:	5
Dedication	7
Preface	9
Section I	11
Introduction	13
SSI Basics	15
Identus Concepts	17
Section II - Getting Started	21
Installation - Local Environment	23
Introduction	23
Identus Releases Overview	24

Table of contents

Pre-requisites	24
Git	24
Docker	25
WSL	25
Before We Run The Agent	25
Atala Community Projects	26
Exploring The Repository	26
Running The Cloud Agent	27
Environment Variables	27
Section III - Building	31
Project Overview	33
Wallets	35
DIDs	37
DID Documents	39
Connections	41
Verifiable Credentials	43
Overview	43
Formats	45
Schemas	45
Issuing	45
Updating	45
Revoking	46
DIDComm	47
Overview	47
Sending Messages	47
Sending Files	47

Table of contents

Verification	49
Plugins	51
 Section IV - Deploy	 53
Installation - Production Environment	55
Mediator	57
Overview	57
Why use a Mediator?	58
Set up a Mediator	58
Maintenance	59
 Section V - Addendum	 61
Trust Registries	63
Continuing on Your Journey	65
 Appendices	 67
Errata	69
Glossary	71

Welcome

Welcome to the book!

Copyright

IdentusBook.com

Copyright ©2024 Jon Bauer and Roberto Carvajal

All rights reserved

Please see our License for more information about fair use.

License

Book Content:

The content of the book (text, printed code, and images) are licensed under Creative Commons Attribution-NonCommercial-ShareAlike 4.0

The intent here is to allow anyone to use or excerpt from the book as long as they credit the source. In general we are very open to having our work shared non-commercially. We are copyrighting this book to protect the ability to version and revise the work officially, and to keep open the door for a print version if the opportunity arises and/or makes sense.

If you'd like to commercially reproduce contents from this book, please contact us.

Applications:

The example code that accompanies this book is licensed under a MIT License

This allows anyone to freely take, modify, or redistribute the code as long as they also assign the MIT license to it.

Dedication

Preface

Explains who the authors are and why we were motivated to write this book.

Section I

Introduction

This is a developer-centric book about creating and launching Self-Sovereign applications with Identus. We aim to show the reader how to configure, build and deploy a complex idea from scratch.

This is not a book about Self-Sovereign Identity. There are already great resources available on that topic. If you're new to the idea of SSI or Identus, we recommend the excellent resources listed below as a pre-requisite to this text.

- Self-Sovereign Identity by Alex Preukschat, Drummond Reed, et al. This is the definitive book on SSI and it's ecosystem of topics.
- Atala PRISM Documentation

It should be noted that Identus is still new and at the time of writing this book, there are very few best practices, in fact, very little practices at all. A handful of adventurous developers have been building on the platform and sharing their experiences, and we hope to share our own learnings in hopes of magnifying that knowledge and helping developers skip the common pitfalls and bring their ideas to market.

We hope this text will be accessible to anyone who's curious about what's

We hope that newcomers will be able to use this text to skip common pitfalls and misunderstandings, and bring their ideas to market faster.

We're glad you could join us :)

SSI Basics

[SSI Roles/Conceptual Diagram (Triangle of Trust)]

Issuer:

The entity that issues a Verifiable Credential to a Holder. This could be a bank, government agency or anyone that accepts responsibility for making credentials. For example a governmental agency can issue a passport to a citizen, or a gym can issue a membership to a member. The type of agency is not important, only that they issue a credential. This role accepts responsibility for having issued that credential and should look to establish trust and reputation with Holders and Verifiers. The Issuer's DID will be listed in the Issuer (iss:) field of a Verifiable Credential and can be inspected by anyone wishing to know the origin of the credential.

Holder:

A Holder is simply any person or wallet that holds a Verifiable Credential. Verifiable Credentials can represent anything, a gym membership, an accomplishment like a University degree, or a permission set, such as a login or authentication role.

Verifier:

A Verifier is any person or wallet that performs a verification on a Verifiable Credential or any of its referenced entities. A Verifier might perform a check on cryptographic elements of a VC, or make sure that the Issuer DID belongs to the expected entity. Verifiers usually only care to double check that the Verifiable Credential is legitimate and that the included

SSI Basics

claims meet their expectations, for example when a bartender checks the age of a patron before serving them alcohol.

Trust Registry:

When Verifiers need to know who a DID belongs to, there needs to be a way to look up that information. A Trust Registry is a mapping between DIDs and the entities they represent. You can think of this like a phone book for DIDs. Trust Registries need to be trusted themselves, and can be as broad or as specific as they need to be. For example, an Real Estate specific Trust Registry that lists real estate agents can be a way to validate that a particular agent is an accepted member of that industry.

Identus Concepts

[Identus Application Architecture Diagram]

Identus is made up of several open source components. Each could be used or forked separately but they are designed to work well together.

PRISM Node:

PRISM Node implements the `did:prism` methods and is an interface to multiple VDR (Verifiable Data Registries). The node can resolve PRISM DIDs and write transactions to a blockchain or database. PRISM Node is expected to be online at all times.

Cloud Agent:

Written in Scala, the Cloud Agent runs on a server and communicates with clients and peers via a REST API. It is a critical component of an Identus application, able to manage identity wallets and their associated operations, as well as issue Verifiable Credentials. The Cloud Agent is expected to be online at all times.

Edge Agent:

Edge Agents give agent capabilities to clients like Websites and mobile apps. They can never be assumed to be online at any given time, and therefore rely on sending and receiving all communications through an online proxy, the Mediator.

DIDComm:

Identus Concepts

DIDComm is a private, secure, and interoperable protocol for communication between decentralized identities. Identus supports DIDCommV2 and allows peers to pass messages between each other, proxied by the Mediator. Messages contain a **to:** and **from:** DID

Mediator:

Mediators act as middlemen between Peer DIDs. In order for any agent to send a message to any other agent, it must know the **to** and **from** DIDs of each message. The sender and recipient together make up a cryptographic connection called a **DIDPair**. Mediators maintain queues of messages for each **DIDPair**. If an Edge Agent is offline, the Mediator will hold incoming messages for them until the agent is back online and able to receive them. Mediators can deliver messages when polled, or push via Websockets. Mediators are expected to be online at all times and be highly available.

Since instantiation of Identus Edge Agents requires a Mediator, there are several publicly available Mediator services which make development simple.

- PRISM Mediator
- RootsID Mediator
- Blocktrust Mediator

While extremely helpful during development, these are not recommended for production Identus deployments as they have no uptime guarantee and will not scale past a small number of concurrent users. We will discuss how to run your own Mediator in Chapter

Building Blocks:

Identus separates the handling of important SSI operations into separate, focused libraries.

Apollo: Apollo is a cryptographic primitives toolbox, which Identus uses to ensure data integrity, authenticity, and confidentiality.

Castor: Castor enables creation, management, and resolution of DIDs.

Pollux: Pollux handles all Verifiable Credential operations.

Mercury: Mercury is an interface to the DIDCommV2 protocol, allowing the sending and receiving of messages between DIDs.

More info on each of the Building Blocks can be found in the Docs

Section II - Getting Started

Installation - Local Environment

Introduction

Hyperledger Indentus, previously known as Atala PRISM, is distributed across various repositories. These repositories group together different building blocks to provide the necessary functionality for fulfilling each of the essential roles in Self-Sovereign Identity (SSI), as introduced in Indentus Concepts and SSI Basics. Throughout this book, we will detail the setup of each component.

The initial component to set up is our Cloud Agent. This agent is responsible for creating and publishing DID Documents into a Verifiable Data Registry (VDR), issuing Verifiable Credentials, and, depending on the configuration, even providing Identity Wallets to multiple users through a multi-tenancy setup. For now, our focus will be on setting up the Cloud Agent to run locally in development mode, supporting only a single tenant. This step is crucial for learning the basics and getting started. As we progress and build our example application, we will deploy the Cloud Agent in development mode first, then set it up to connect to Cardano **testnet** as our VDR on pre-production mode and, finally, in production mode with multi-tenancy support connected to Cardano **mainnet**. This will be a gradual process as we need to familiarize on each stage of the Cloud Agent.

Identus Releases Overview

Identus is built upon multiple interdependent building blocks, including the Cloud Agent, Wallet SDK, Mediator, and Apollo Crypto Library. To ensure compatibility among these components, it is crucial to identify the correct building block versions that are compatible between them. For this purpose, a dedicated repository named `atala-releases` is available. This repository provides comprehensive documentation and a compatibility table for each Identus Release. We will be using Identus v2.12 as our selected release because it is the latest at the time of writing (May 2024).

Pre-requisites

Git

GitHub is currently the primary platform for hosting repositories. As of this writing, projects are transitioning from Atala PRISM's original repositories to Hyperledger ones, with the Cloud Agent being the first to migrate. For more details, you can read the press release [here](#).

If you're using a UNIX-based system (such as OS X or Linux), you likely already have `git` installed. If not, you can download the installer from [Git downloads](#). Additionally, various GUI clients are available for those who prefer a graphical interface.

To clone the Cloud Agent repository, first go to the Releases page and identify the tagged release corresponding to the Identus release you are targeting (e.g. `cloud-agent-v1.33.0` is part of Identus v2.12 release), then clone the repository with this command:

```
git clone --depth 1 --branch cloud-agent-v1.33.0 https://github.com/hyperledger
```

i Note

Using `--depth 1` will skip the history of changes in the repository up until the point of the tag, this is optional.

Docker

The Cloud Agent and Mediator are distributed as Docker containers, which is the recommended method for starting and stopping the various components required to run the cloud infrastructure.

To begin, install Docker Desktop, which provides everything you need to get started.

WSL

For Windows users, please refer to [How to install Linux on Windows with WSL](#).

i Note

Windows is the least tested environment, the community has already found some issues and workarounds on how to get the Cloud Agent working. We will try to always include instructions regarding this use case.

Before We Run The Agent

Once you have cloned the `identus-cloud-agent` repository and Docker is up and running you can jump right ahead and run the agent, but before we do that if you are not yet familiar with the community projects or

Installation - Local Environment

the structure of the agent itself, we recommend you to spend a little time exploring the following information, this is optional and you can skip it.

Atala Community Projects

There is a growing list of community repositories that aim to provide some extra functionality, mostly maintained by official developers and community members on their spare time. At present time there are three Atala Community Projects:

- **Pluto Encrypted:** Implementation of Pluto storage engine with encryption support.
- **Edge Agent SDK Demos:** Browser and Node versions of Edge Agent SDK integrated with Pluto Encrypted.
- **Identus Test:** Shell script helper that will checkout a particular Identus release and compatible components.

Exploring The Repository

There are two fundamental directories inside the repository if you are an end user.

1. **docs** Where all the latest *technical documentation* will be available, this includes the Architecture Decision Records (ADR), general insights, guides to deploy, examples and tutorials about how to handle VC Schemas, Connections, secrets, etc. We will do our best to explain in detail all this procedures as we build our example app.
2. **infrastructure** This directory holds the agent's Docker file and related scripts to run the agent in different modes such as `dev`, `local` or `multi`. **The way to change the agent setup is by customizing environmental variables trough the Docker file**, so we really advise you to get familiar with the `shared` directory content,

because that's the base for every other mode, in essence, every mode is a customization of the shared Docker file.

Our first mode to explore and the simplest one should be `local` mode, which by default will run a single agent as a single-tenant, meaning that this instance will control only a single Identity Wallet that will be automatically created and seeded upon the first start of the agent.

The `multi` mode essentially runs 3 different `local` agents but each is assigned a particular role such as `issuer`, `holder` and `verifier`. This is useful in order try test more complex interactions between independent actors.

Finally the `dev` mode is meant to be used for development and provides an easy way to modify the Cloud Agent source code, it does not rely on the pre-built Docker images that the `local` mode fetches and run. We will not use this mode at all trough the book but feel free to explore this option if you would like to make contributions to the Cloud Agent in the future.

Running The Cloud Agent

Environment Variables

Inside `infrastructure/local` directory, you will find three important files, `run.sh` and `stop.sh` scripts and the `.env` file.

Our `local` environment file should look like this

```
AGENT_VERSION=1.33.0
PRISM_NODE_VERSION=2.2.1
VAULT_DEV_ROOT_TOKEN_ID=root
```

Installation - Local Environment

This will tell Docker which versions of the Cloud Agent and PRISM Node to run, plus a default value for the `VAULT_DEV_ROOT_TOKEN_ID`, this value corresponds to the HashiCorp Token ID. HashiCorp is a secrets storage engine and it will become relevant later on when we need to prepare the agent to run in `prepod` and `production` modes, for now the `local` mode will ignore this value because by default it will use a local `postgres` database for its secret storage engine.

The `run.sh` script options:

```
./run.sh --help
Run an instance of the ATALA bulding-block stack locally

Syntax: run.sh [-n/--name NAME|-p/--port PORT|-b/--background|-e/--env|-w/--wait]
options:
-n/--name           Name of this instance - defaults to dev.
-p/--port           Port to run this instance on - defaults to 80.
-b/--background     Run in docker-compose daemon mode in the background.
-e/--env            Provide your own .env file with versions.
-w/--wait           Wait until all containers are healthy (only in the background)
--network           Specify a docker network to run containers on.
--webhook           Specify webhook URL for agent events
--webhook-api-key   Specify api key to secure webhook if required
--debug            Run additional services for debug using docker-compose
-h/--help          Print this help text.
```

For our first interaction with the agent all we have to do is to call the `run` script. If you have any conflicts with the port 80 already in use you can pass `--port 8080` or any other available port that you would like to use.

So, from the root of the repository you can run:

Running The Cloud Agent

```
./infrastructure/local/run.sh
```

This will take a while the first time as Docker will fetch the required container images and get them running. To check the status of the Cloud Agent you can use `curl` or open a browser window at same endpoint URL (make sure to replace port 80 if you changed it in the previous step):

```
curl http://127.0.0.1:80/cloud-agent/_system/health  
{"version":"1.33.0"}
```

The `version` should match the version of the Cloud Agent defined in the `.envfile`.

To stop the agent, run:

```
./infrastructure/local/stop.sh
```

Congratulations! you have successfully setup the agent in `local` mode. Next we will explore our Docker file in detail and interact with our agent using the REST API.

Section III - Building

Project Overview

This is project overview of the example app, which can issue and verify educational credentials.

Wallets

A deep dive on what an Identus Identity wallet is and represents. We will discuss seed phrases and how they work.

We will talk about creating and recovering Identus wallets.

Tips for using:

- Pluto Encrypted
- Keycloak for OAuth / OpenIDConnect

DIDs

We'll do a deep dive on DIDs and did:prism specifically

- did:prism methods
- canonical vs longform DID

DID Documents

DIDDocuments:

- Resolvers
- Controllers

Connections

We will discuss the concept of Connections, PeerDIDs, and what it means to connect with another Peer, what's happening under the hood and different ways to handle the process in your application.

Also discussed:

- Managing Invites
 - OOB
 - Manual Accept
 - Auto Accept
 - DIDLess Connections (Atala Roadmap)

Verifiable Credentials

Overview

Verifiable Credentials are an integral part of Self-Sovereign Identity, allowing individuals to keep and control how their personal information is shared.

A Verifiable Credential (VC) is a digital statement made by an **Issuer** about a **Subject**. This statement is cryptographically secured and can be verified by a third party without the need for the **Verifier** to directly contact the **Issuer**. Verifiable Credentials are used to represent information such as identity documents, academic records, professional certifications, and other forms of credentials that traditionally exist in paper form. Coupled with other technology such as Self-Sovereign Identity, Verifiable Credentials can unlock novel and exciting use cases.

Components of a Verifiable Credential:

- **Issuer:** The entity that creates and signs the credential. This could be an organization, institution, government entity or individual.
- **Holder:** The entity or individual to whom the credential is issued to and who can present proof to a **Verifier**.
- **Verifier:** The entity or individual that checks the authenticity and validity of the credential through requesting proof from a **Holder**.
- **Subject:** The entity or individual about which the claims are made. In many cases, the **Holder** and the **Subject** are the same entity.

Verifiable Credentials

- **Claims:** Statements about the **Subject**, such as “Alice has an Educational Credential from Vienna University.”
- **Proof:** Cryptographic evidence, using Digital Signatures, that the credential is authentic and has not been tampered with.
- **Metadata:** Additional contextual information which may have content or application specific meaning, like expiration date or credential description.

How Verifiable Credentials Work:

- **Issuance:** The issuer creates a credential containing claims about the subject, the subject DID (public key) and signs it with their private key.
- **Storage:** The holder receives the credential and stores it in a digital wallet.
- **Presentation:** When required, the holder presents the credential to a verifier. Selective Disclosure can be used to reveal only relevant context about a claim, and not all user data.
- **Verification:** The verifier checks the credential’s authenticity by validating the issuer’s digital signature and ensuring the credential has not been tampered with.

Benefits of Verifiable Credentials:

- **Interoperability:** VCs follow standard formats, making them compatible across different systems and platforms.
- **Privacy:** Holders can share only the necessary information, protecting their privacy.
- **Security:** Cryptographic techniques ensure the integrity and authenticity of credentials.
- **Decentralization:** VCs do not rely on a central authority for verification, reducing single points of failure.

Use Cases:

- **Digital Identity:** Proof of identity for accessing services.
- **Education:** Digital diplomas and certificates.
- **Healthcare:** Vaccination records and medical certificates.
- **Employment:** Professional qualifications and work experience.

Formats

- Formats (just mention types, details later in Issuing)
 - W3C
 - JWT
 - SD-JWT
 - AnonCreds

Schemas

- Schemas
- Publishing your Schema

Issuing

- JWT
- SD-JWT (Atala Roadmap Q2)
- AnonCreds

Updating

- Updating (re-binding)

Revoking

- Revoking

DIDComm

Overview

DIDComm v2 (*Decentralized Identifier Communication version 2*) is a set of communication protocols designed to enable secure, private, and interoperable messaging between DIDs. These protocols have been adopted throughout Identus to standardize many important interactions, like sending and receiving encrypted messages between a mediator its peers.

DIDComm Messaging messages are encrypted JSON Web Message (JWM) envelopes. They have a standard structure including headers, a body, and optional attachments, which are encrypted and signed to ensure security and integrity.

For details on how messages are sent and received in Identus, see Mediators

Sending Messages

Sending Files

Verification

Verification

- Presenting Proof
- Presentation Policies / Verification Policies
- Selective Disclosure with AnonCreds

Plugins

This is a placeholder for the topic of Plugins. Identus plugins have not been officially announced yet but are on the IOG roadmap for 2024, Q4. If there is any information by the time we publish we will write about them.

Section IV - Deploy

Installation - Production Environment

In this chapter we will discuss how to prepare a production environment for an Idenus application.

We will discuss:

- Hardware recommendations
- Production configuration and security
- Lock down Docker / Postgres (default password hole, etc)
- SSL
- Multi Tenancy
- Keycloak
- Testnet / Preprod env
- Connecting to Mainnet
- Set up Cardano Wallet
- Key Management with HashiCorp
- Connecting to Cardano
- Running dbSync

Mediator

Overview

A Self-Sovereign Identity Mediator is a service that enables private and secure connections between peers. Its primary function is to relay encrypted messages between DIDs using DIDComm V2 protocols. Since each participant wallet is Self-Sovereign, Mediators help route messages to their intended recipient without any one wallet knowing the details of the other. Encrypted messages can be sent to peers even if they are not online, and the Mediator will manage a queue for the offline party until they are able to connect and retrieve them.

Think of a Mediator like a set of Post Office mailboxes. Each mailbox has a mailing address (*a DID*). Each mailbox can hold an stack of incoming mail Encrypted DIDComm V2 messages for its owner. Each mailbox also keeps a list of cryptographic Connections with whom its owner can send or receive mail with. In this way, Identus Mediators facilitate all types of important transactions, including routing both messages, Verifiable Credential issuance, and Credential Exchange.

What's important is that the Mediator, while messages are routed through it, is not a centralized actor. Mediators can not know the contents of any message, only what DIDs can talk to other DIDs and what encrypted messages they have yet to read.

Why use a Mediator?

Why do we need a Mediator in the first place? If participants are Self-Sovereign, then why not connect an edge wallet directly to another edge wallet?

SSI participants are connected through various types of devices and network conditions. The Cloud Agent, is expected to be online and available all the time, as it's hosted in a remote server farm somewhere. However users who keep their wallets on mobile devices, or laptop computers, are only online sporadically, and can't be expected to have a reliable connection to the Internet at all times. This is where Mediators earn their keep. If User A were to send a message to User B, who happens to be offline at the moment, the message could not be delivered, plus User A would also have to know sensitive data about how to contact User B. If User B changes devices, they would have to tell User A, and all other peer Connections. If User A instead, sends the message through the Mediator, it can be held securely and retrieved the next time User B connects to the Internet. Mediators provide reliability between Connections, without compromising privacy or security.

Set up a Mediator

We will teach the reader how to

- Install, configure, and Run your own Mediator
- How to manage Websockets
- Performance Tuning

Maintenance

Mastering Identus means maintaining your application once it's launched.

In this chapter we will cover:

- Observability
 - Manage nodes / Memory
 - Performance Testing
 - Analytics with BlockTrust Analytics
- Upgrading Agents
 - How to minimize downtime
- Hashicorp
 - Key Management
 - Key rotation

Section V - Addendum

Trust Registries

Overview of the concept of a Trust Registry

- What role they play in SSI
- Trust Over IP Trust Registry Spec
- Highlight any real world examples if they exist by book completion

Continuing on Your Journey

Information about becoming a Contributor to Identus

- How to contribute to the source code or documentation

Information about how to get involved in the SSI community outside of Identus

Trust over IP (ToIP)

Trust Over IP (ToIP) comprises over 300 organizations from diverse industries collaborating to advance Decentralized Identity through ideas and software. According to ToIP, “We develop tools and specifications to help communities of any size use digital networks to build and strengthen trust between participants.”

The ToIP Stack, published in 2019, describes how the four layers (DIDs, DIDComm Protocol, Data Exchange Protocols, and Application Ecosystems) form a secure, scalable, and interoperable digital trust framework. Introduction to ToIP, Version 2.0, released in 2021, is an excellent resource for anyone catching up on the Digital Identity problem and solution. Membership options include free Contributor accounts and paid memberships, providing access to ToIP’s resources and Slack channels for collaboration.

To join Trust Over IP, you need to create a free Linux Foundation account and then complete your ToIP application [here](#).

Continuing on Your Journey

All members agree to open, non-competitive participation, so please have your legal team review all associated documents during the onboarding process.

For further details, refer to The ToIP Stack and Overview.

Decentralized Identity Foundation (DIF)

Appendices

Errata

Errata goes here

We will list any bugs that may have been “printed” in certain editions of the book.

Glossary

Glossary or Index here

This will reference key concepts and Identus terms and where they are mentioned in the book, allowing someone to look up a term and know what section it is referenced in.

TODO: First draft will just lay down the glossary terms, we want to add detailed descriptions later.

VDR Verifiable Data Registry

