03 Operations Guidelines

This document describes guidelines for installing and running the SSI Platform solution.

Confluence Page	Short Description	Owner
01 SSI Platform Runbook	Describes a compilation of procedures to run the EESDI solution.	[redacted]
02 Installation Guidelines	Describes guides for installing parts of the EESDI solution	[redacted]
03 Technical Environment (IBM Cloud)	Describes the Technical Environment in more details (focus on the IBM Cloud based components)	

Target Audience

The operations team responsible for installing and running the solution.

01 SSI Platform Runbook

The Runbook describes a compilation of routine procedures and operations that the system administrators / operators are performing to begin, stop, supervise, and debug the SSI Platform.

ID	Confluence PAge	Short Description	Owner
RB001	RB001 - Fixing issues with the Indy network	How to fix issues with the Indy network when transactions fail due to timeout.	[redacted]
RB002	RB002 - Adding a new steward node	How to set up a node from scratch and add it to the pool	[redacted]
RB003	RB003 - Restarting a node	How to restart a node and resync	[redacted]
RB004	RB004 - Creating and registering DIDs	How to create DIDs and manage the permissions that they have	[redacted]
RB005	RB005 - Bootstrapping a new Issuer	How to bootstrap a new issuer in the EESDI ledger.	[redacted]

RB001 - Fixing issues with the Indy network

Version	Date	Author	Comment	Status
0.1	??.0?.2021	[redacted]	Initial version	DRAFT
0.2	23.09.2021	[redacted]	Updated Format, removing wallets & dids included	IN REVIEW
1.0	23.09.2021	[redacted]	Changed status to approved	APPROVED

Table of Contents

Problem Description

If write transactions fail with a timeout, it is likely that the **ledger cannot find consensus**. A potential reason for this is that at some time, less than the minimum required number of nodes was available. For example, in a 5 node Indy network, one failure is tolerated while two failures cannot be compensated.

Solution

Restarting Indy-Nodes

To solve the problem we need to restart all nodes (see also RB003 - Restarting a node). For the native installation on Ubuntu 16, this is as easy as:

Restart all indy-nodes

```
systemctl stop indy-node && systemctl start indy-node
```

For dockerized nodes, restarting the docker container will likely work.

Alternative Approaches

Another possibility is killing the process and restarting it afterward. This approach has helped at least once so far.

Killing and restarting the process

```
# Look for the indy process to be killed
ps -aux | grep start_indy_node

# Kill the corresponding PID
kill -s KILL <PID>
```

There will likely be cases in which just restarting every node is not sufficient. In this case, one could try to sequentially stop each node, delete the ledger, and restart the node:

Stop each node, delete the ledger, and restart the node

```
systemctl stop indy-node
rm -rf /var/lib/indy/data
systemctl start indy-node
```

It may be the case that the ledger is stored somewhere else, the configuration that specifies the path to the ledger data is configured in /etc/indy /indy_config.py. Note that the catchup-process where the node fetches the ledger data from other nodes can take some time if there are already many transactions on the ledger. Also, simultaneously deleting the ledger data on too many nodes will probably imply that the ledger cannot be recovered and the data is lost. So, it is advisable to only make a "clean restart" with one node at a time and move on to the next node only once the catchup-process is finished.

Test Consensus Status

A convenient way to test whether the ledger can find consensus again is to conduct a transaction from the indy-cli:

Test whether the ledger finds consensus again

```
root@ssi-indy-network-node01:/var/lib/indy/Chancellory Network# indy-cli

# Create a wallet
indy> wallet create name=mywallet key=mykey
Wallet "name=mywallet" has been created

# Open the wallet
indy> wallet open name=mywallet key=mykey
Wallet "name=mywallet" has been opened

# Create a DID from seed (use the same seed that was used for creating the trustee DID, i.e., the seed used in
an aca-py with write permissions)
name=mywallet:indy> did new seed=<<TRUETEE-SEED>>
Did "QECRt1tsqU3W9NksPCw76b" has been created with "~GPehEsXwoqw5cqDdJyMZrq" verkey

# Switch to this DID
name=mywallet:indy> did use QECRt1tsqU3W9NksPCw76b
Did "QECRt1tsqU3W9NksPCw76b" has been set as active
```

Then, load and connect to the ledger from the pool_transaction_genesis file and register the new DID:

Test whether the ledger finds consensus again

```
# Load ledger from pool_transactions_genesis
name=mywallet:did(QEC...76b):indy> pool create myledger gen_txn_file="/var/lib/indy/Chancellory Network
/pool_transactions_genesis"
Pool config "myledger" has been created

# Connect to this ledger
name=mywallet:did(QEC...76b):indy> pool connect myledger
Pool "myledger" has been connected

# Register new DID (for examples, type ledger nym help)
pool(myledger):name=mywallet:did(QEC...76b):indy> ledger nym did=Ltwa6LtWHV2ZM5RNnV8kCo
verkey=~Lzrhaa2pbX7bqgzTrnrw6T
Nym request has been sent to Ledger.

# The response should look as follows:

Metadata:
+------------------------+------------------+---------------------+---------------------+
| From                   | Sequence Number  | Request ID          | Transaction time    |
+------------------------+------------------+---------------------+---------------------+
| QECRt1tsqU3W9NksPCw76b | 13               | 1623409656806699207 | 2021-06-11 11:07:36 |
+------------------------+------------------+---------------------+---------------------+
Data:
+------------------------+-------------------------+------+
| Did                    | Verkey                  | Role |
+------------------------+-------------------------+------+
| Ltwa6LtWHV2ZM5RNnV8kCo | ~Lzrhaa2pbX7bqgzTrnrw6T | -    |
+------------------------+-------------------------+------+
```

Removing Wallets & DIDs

In case a locally generated DID (i.e., stored in the wallet) should be removed there is no general command as to delete the entire wallet.

Delete wallet

```
# delete wallet
indy> wallet delete name=mywallet key=mykey
Wallet "name=mywallet" has been deleted
```

Similarly, a registered DID on the ledger can not be removed afterward. However, the roles and thus the entitled rights of a registered public DID can be changed seamlessly by another DID that possesses the permissions (i.e., by a Trustee or a Steward) using the ledger nym command. To do so the verkey parameter of the DID is omitted. This only works hierarchically from the top to bottom. For example, a Trustee can change the role of a Steward to an Endorser. As such, it is not possible to switch and use a DID with fewer rights (e.g., an Endorser) to grant an existing public DID with more rights (e.g., a Trustee).

Change role of a public DID

```
# Possible roles on the ledger (there is no difference in entering the role as string (be cautious about the
capital letters) or the IDs (no quotation marks required))
# – None (common USER)
# – “0” (TRUSTEE)
# – “2” (STEWARD)
# – “101” (ENDORSER)
# – “201” (NETWORK_MONITOR)

# change current role of a public DID to the disered role
name=mywallet:did(QEC...76b):indy> ledger nym did=Ltwa6LtWHV2ZM5RNnV8kCo role=NETWORK_MONITOR

# The output should look like this:

Nym request has been sent to Ledger.
Metadata:
+------------------------+-----------------+---------------------+---------------------+
| From                   | Sequence Number | Request ID          | Transaction time    |
+------------------------+-----------------+---------------------+---------------------+
| QECRt1tsqU3W9NksPCw76b |<sequence number>| <request ID>                 | <transaction time>  |
+------------------------+-----------------+---------------------+---------------------+
Data:
+------------------------+-----------------+
| Did                    | Role            |
+------------------------+-----------------+
| Ltwa6LtWHV2ZM5RNnV8kCo | NETWORK_MONITOR |
+------------------------+-----------------+
```

RB002 - Adding a new steward node

Version	Date	Author	Comment	Status
0.1	22.08.2021	[redacted]	Initial version	DRAFT
0.2	23.08.2021	[redacted]	Reworked version	IN REVIEW
0.21	06.09.2021	[redacted]	Reviewed	IN REVIEW
1.0	09.09.2021	[redacted]	Reviewed; minor editorial changes; tagged as approved	APPROVED

Table of Contents

This is a considerably more detailed description than https://hyperledger-indy.readthedocs.io/projects/node/en/latest/add-node.html.

Prepare Steward and Node Keys

Run the indy-cli

Install the indy-cli on an Ubuntu 16 server (alternatively, exec into a docker container that has the indy-cli installed, like `docker run -it docker.io/bcgovimages/von-image:node-1.12-3 indy-cli).`

More instructions on how to use the indy-cli are available at IG003 Indy Node Installation, Indy Network Setup.

In the case of docker, make sure that the seed is stored in a password manager like 1Password or KeePass, a company vault, or similar software, as the keypair that is created from the seed will be removed when the container stops.

Install pre-requisites

```
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys CE7709D068DB5E88
add-apt-repository "deb https://repo.sovrin.org/sdk/deb xenial stable"
apt update
apt install indy-cli
```

Transfer (e.g. scp, wget) the `pool_transactions_genesis` from a running node to the server.
Example: `/root/pool_transactions_genesis`

Register the new Steward

Use the indy-cli to add a new Steward (a Steward may add a single node to the ledger). Only Trustees can add Stewards.

Add Steward to the Ledger

```
indy-cli

# create and open a wallet; the key should be replaced by a safe passphrase if the wallet is persisted
wallet create name="stewardwallet" key="stewardwallet"
wallet open name="stewardwallet" key="stewardwallet"

# import the Trustee Seed (or open a wallet that contains the trustee DID)
did new seed=[TRUSTEE_SEED]

# create a new for the Steward (this can also happen in another wallet, what is needed is the Steward DID and
verkey
did new seed=[STEWARD_SEED]

# switch to the trustee DID to prepare for adding the Steward
did use [TRUSTEE_DID]

# connect to the ledger (this is where the pool_transactions_genesis is required)
name="stewardwallet":did(TRUSTEE_DID):indy> pool create pool1 gen_txn_file=/root/pool_transactions_genesis
name="stewardwallet":did(TRUSTEE_DID):indy> pool connect pool1

# register the Steward on the ledger. If there is a timeout, the ledger is currently not finding consensus and
some nodes need to be restarted
pool(pool1):name="stewardwallet":did(TRUSTEE_DID):indy> ledger nym did=[DID] verkey=[VERKEY] role=STEWARD
```

Add and start the new node

Now, run the new node by following the instructions in IG003 Indy Node Installation, Indy Network Setup. Make sure that the node_ip (default:9701) and client_ip (default:9702) ports are available from the open internet. Also, make sure that the node's IP address and the node_ip and client_ip ports are added to the other node's IP tables, so they are not blocked by the firewall before they reach the indy node.

If the node crashes, this is okay (in fact, it will likely connect with all of the other nodes but be rejected, which can be seen in the other nodes' logs), but it is important that the logs show that it creates cryptographic material from the seed (see IG003 Indy Node Installation, Indy Network Setup, **Install Indy Node Step-By-Step**, Lines 29 - 39). This cryptographic material (VERIFICATION_KEY, BLS_KEY; BLS_POP (proof of possession)) is needed for registering the node on the ledger.

Now, it is time to register the new node and its cryptographic material on the ledger in a transaction.

Add New Node to the Ledger

```
# Open the indy-cli
indy-cli

# create and open a wallet; the key should be replaced by a safe passphrase if the wallet is persisted
wallet create name="stewardwallet" key="stewardwallet"
wallet open name="stewardwallet" key="stewardwallet"

# switch to the stewardDID to prepare for adding the validator
did use [STEWARD_DID]

# add the new node to the ledger
ledger node target=<<NEW NODE VERKEY>> client_port=9702 client_ip=<<NODE IP ADDRESS>> alias=<<NODE NAME>>
node_ip=<<NODE IP ADDRESS>> node_port=9701 services=VALIDATOR blskey=<<NEW NODE BLSKEY>> blskey_pop=<<NEW NODE
BLSKEY PROOF OF POSSESSION (POP)
```

If the response from the ledger is a success, the logs of the nodes that are already running should now switch to "hungry" (if they were not before) and watch out for the newly added node.

```
2021-08-23 06:56:07,086|INFO|motor.py|Node2 changing status from started to started_hungry
```

Now it is time to restart the new newly added node and check whether it can connect with the other nodes. There should be logs for catching up with all of the previous transactions on the new node, and the existing nodes should switch back from hungry to non_hungry.

2021-08-23 06:57:18,971|INFO|seeder_service.py|Node2 received catchup request: CATCHUP_REQ{'ledgerId': 3, 'seqNoStart': 1, 'seqNoEnd': 400023, 'catchupTill': 400023} from Node4

New Node Catchup Logs

```
2021-08-23 06:57:12,798|INFO|motor.py|Node4 changing status from stopped to starting
2021-08-23 06:57:12,808|INFO|zstack.py|Node4 will bind its listener at 0.0.0.0:9701
2021-08-23 06:57:12,809|INFO|stacks.py|CONNECTION: Node4 listening for other nodes at 0.0.0.0:9701
2021-08-23 06:57:12,809|INFO|zstack.py|Node4C will bind its listener at 0.0.0.0:9702
2021-08-23 06:57:12,827|INFO|node.py|Node4 first time running...
2021-08-23 06:57:12,827|INFO|node.py|Node4 processed 0 Ordered batches for instance 0 before starting catch up
2021-08-23 06:57:12,828|INFO|ordering_service.py|Node4:0 reverted 0 batches before starting catch up
2021-08-23 06:57:12,828|INFO|node_leecher_service.py|Node4:NodeLeecherService starting catchup (is_initial=True)
2021-08-23 06:57:12,828|INFO|node_leecher_service.py|Node4:NodeLeecherService transitioning from Idle to
PreSyncingPool
2021-08-23 06:57:12,828|INFO|cons_proof_service.py|Node4:ConsProofService:0 starts
2021-08-23 06:57:12,829|INFO|kit_zstack.py|CONNECTION: Node4 found the following missing connections: Node3,
Node2
2021-08-23 06:57:12,830|INFO|zstack.py|CONNECTION: Node4 looking for Node3 at 192.168.1.205:9701
2021-08-23 06:57:12,831|INFO|zstack.py|CONNECTION: Node4 looking for Node2 at 192.168.1.204:9701
2021-08-23 06:57:12,831|INFO|zstack.py|CONNECTION: Node4 looking for Node1 at 192.168.1.203:9701
2021-08-23 06:57:12,909|INFO|seeder_service.py|Node4 received ledger status: LEDGER_STATUS{'ledgerId': 0,
'txnSeqNo': 3, 'viewNo': None, 'ppSeqNo': None, 'merkleRoot': 'FDSyDSEzDhiJ8J5AHJasquEY9WSaL3p6Msn83EYrYxyT',
'protocolVersion': 2} from Node3
2021-08-23 06:57:12,910|INFO|cons_proof_service.py|Node4:ConsProofService:0 comparing its ledger 0 of size 3
with 3
2021-08-23 06:57:12,910|INFO|cons_proof_service.py|Node4:ConsProofService:0 comparing its ledger 0 of size 3
with 3
2021-08-23 06:57:12,910|INFO|cons_proof_service.py|Node4:ConsProofService:0 deciding on the basis of CPs
{'Node3': None} and f 0
2021-08-23 06:57:12,911|INFO|cons_proof_service.py|Node4:ConsProofService:0 found proof by Node3 null
2021-08-23 06:57:12,924|NOTIFICATION|keep_in_touch.py|Node4's connections changed from set() to {'Node1',
'Node2', 'Node3'}
2021-08-23 06:57:12,924|NOTIFICATION|keep_in_touch.py|CONNECTION: Node4 now connected to Node3
2021-08-23 06:57:12,925|INFO|motor.py|Node4 changing status from starting to started_hungry
2021-08-23 06:57:12,927|INFO|seeder_service.py|Node4 received ledger status: LEDGER_STATUS{'ledgerId': 0,
'txnSeqNo': 3, 'viewNo': None, 'ppSeqNo': None, 'merkleRoot': 'FDSyDSEzDhiJ8J5AHJasquEY9WSaL3p6Msn83EYrYxyT',
'protocolVersion': 2} from Node2
2021-08-23 06:57:12,927|INFO|cons_proof_service.py|Node4:ConsProofService:0 comparing its ledger 0 of size 3
with 3
2021-08-23 06:57:12,928|INFO|cons_proof_service.py|Node4:ConsProofService:0 comparing its ledger 0 of size 3
with 3
2021-08-23 06:57:12,929|INFO|cons_proof_service.py|Node4:ConsProofService:0 finished with consistency proof None
2021-08-23 06:57:12,929|INFO|catchup_rep_service.py|Node4:CatchupRepService:0 started catching up with
LedgerCatchupStart(ledger_id=0, catchup_till=None, nodes_ledger_sizes={})
2021-08-23 06:57:12,929|INFO|catchup_rep_service.py|CATCH-UP: Node4:CatchupRepService:0 completed catching up
ledger 0, caught up 0 in total
2021-08-23 06:57:12,930|INFO|node_leecher_service.py|Node4:NodeLeecherService transitioning from PreSyncingPool
to SyncingAudit
2021-08-23 06:57:12,930|INFO|cons_proof_service.py|Node4:ConsProofService:3 starts
2021-08-23 06:57:12,930|INFO|cons_proof_service.py|Node4:ConsProofService:3 asking for ledger status of ledger 3
2021-08-23 06:57:12,932|NOTIFICATION|keep_in_touch.py|Node4's connections changed from {'Node3'} to {'Node3',
'Node2'}
2021-08-23 06:57:12,932|NOTIFICATION|keep_in_touch.py|CONNECTION: Node4 now connected to Node2
[...]
2021-08-23 06:57:12,932|INFO|motor.py|Node4 changing status from started_hungry to started
2021-08-23 06:57:12,947|INFO|seeder_service.py|Node4 received ledger status: LEDGER_STATUS{'ledgerId': 3,
'txnSeqNo': 400023, 'viewNo': None, 'ppSeqNo': None, 'merkleRoot':
'Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8', 'protocolVersion': 2} from Node3
2021-08-23 06:57:12,948|INFO|cons_proof_service.py|Node4:ConsProofService:3 comparing its ledger 3 of size 0
with 400023
2021-08-23 06:57:12,948|INFO|seeder_service.py|Node4 received ledger status: LEDGER_STATUS{'ledgerId': 3,
'txnSeqNo': 400023, 'viewNo': None, 'ppSeqNo': None, 'merkleRoot':
'Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8', 'protocolVersion': 2} from Node2
2021-08-23 06:57:12,949|INFO|cons_proof_service.py|Node4:ConsProofService:3 comparing its ledger 3 of size 0
with 400023
2021-08-23 06:57:12,963|INFO|cons_proof_service.py|Node4:ConsProofService:3 received consistency proof:
CONSISTENCY_PROOF{'ledgerId': 3, 'seqNoStart': 0, 'seqNoEnd': 400023, 'viewNo': 0, 'ppSeqNo': 0,
'oldMerkleRoot': 'Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8', 'newMerkleRoot':
'Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8', 'hashes': ['Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8']}
from Node3
2021-08-23 06:57:12,963|INFO|cons_proof_service.py|Node4:ConsProofService:3 deciding on the basis of CPs
{'Node3': CONSISTENCY_PROOF{'ledgerId': 3, 'seqNoStart': 0, 'seqNoEnd': 400023, 'viewNo': 0, 'ppSeqNo': 0,
'oldMerkleRoot': 'Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8', 'newMerkleRoot':
'Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8', 'hashes': ['Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8']}}
and f 0
2021-08-23 06:57:12,964|INFO|cons_proof_service.py|Node4:ConsProofService:3 deciding on the basis of CPs
{'Node3': CONSISTENCY_PROOF{'ledgerId': 3, 'seqNoStart': 0, 'seqNoEnd': 400023, 'viewNo': 0, 'ppSeqNo': 0,
'oldMerkleRoot': 'Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8', 'newMerkleRoot':
'Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8', 'hashes': ['Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8']}}
and f 0
2021-08-23 06:57:12,964|INFO|cons_proof_service.py|Node4:ConsProofService:3 finished with consistency proof
CONSISTENCY_PROOF{'ledgerId': 3, 'seqNoStart': 0, 'seqNoEnd': 400023, 'viewNo': 0, 'ppSeqNo': 0,
'oldMerkleRoot': 'Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8', 'newMerkleRoot':
'Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8', 'hashes': ('Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8',)}
2021-08-23 06:57:12,965|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 started catching up with
LedgerCatchupStart(ledger_id=3, catchup_till=CatchupTill(start_size=0, final_size=400023,
final_hash='Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8'), nodes_ledger_sizes={'Node3': 400023})
2021-08-23 06:57:12,966|INFO|cons_proof_service.py|Node4:ConsProofService:3 ignoring CONSISTENCY_PROOF
{'ledgerId': 3, 'seqNoStart': 0, 'seqNoEnd': 400023, 'viewNo': 0, 'ppSeqNo': 0, 'oldMerkleRoot':
'Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8', 'newMerkleRoot':
'Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8', 'hashes': ['Hbf86JquHDVoxPhHJd3HXHRiuQQcSGqmK5EaXq8pVVN8']}
since it is not gathering consistency proofs
2021-08-23 06:57:18,970|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 requesting 400023 missing
transactions after timeout
2021-08-23 06:57:18,971|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 still missing 400023 transactions
after looking at receivedCatchUpReplies
2021-08-23 06:57:18,971|WARNING|catchup_rep_service.py|Node4:CatchupRepService:3 no eligible nodes found
containing transactions from 1 to 400023,trying all eligible nodes ['Node2'] as a last resort,last data on
available txns was {'Node3': 400023}
2021-08-23 06:57:24,977|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 requesting 400023 missing
transactions after timeout
2021-08-23 06:57:24,977|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 still missing 400023 transactions
after looking at receivedCatchUpReplies
2021-08-23 06:57:30,984|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 requesting 400023 missing
transactions after timeout
2021-08-23 06:57:30,984|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 still missing 400023 transactions
after looking at receivedCatchUpReplies
2021-08-23 06:57:30,985|WARNING|catchup_rep_service.py|Node4:CatchupRepService:3 no eligible nodes found
containing transactions from 1 to 400023,trying all eligible nodes ['Node2'] as a last resort,last data on
available txns was {'Node3': 400023}
2021-08-23 06:57:36,990|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 requesting 400023 missing
transactions after timeout
2021-08-23 06:57:36,990|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 still missing 400023 transactions
after looking at receivedCatchUpReplies
2021-08-23 06:57:43,000|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 requesting 400023 missing
transactions after timeout
2021-08-23 06:57:43,001|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 still missing 400023 transactions
after looking at receivedCatchUpReplies
2021-08-23 06:57:43,001|WARNING|catchup_rep_service.py|Node4:CatchupRepService:3 no eligible nodes found
containing transactions from 1 to 400023,trying all eligible nodes ['Node2'] as a last resort,last data on
available txns was {'Node3': 400023}
2021-08-23 06:57:49,011|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 requesting 400023 missing
transactions after timeout
2021-08-23 06:57:49,011|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 still missing 400023 transactions
after looking at receivedCatchUpReplies
2021-08-23 06:57:55,014|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 requesting 400023 missing
transactions after timeout
2021-08-23 06:57:55,015|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 still missing 400023 transactions
after looking at receivedCatchUpReplies
2021-08-23 06:57:55,015|WARNING|catchup_rep_service.py|Node4:CatchupRepService:3 no eligible nodes found
containing transactions from 1 to 400023,trying all eligible nodes ['Node2'] as a last resort,last data on
available txns was {'Node3': 400023}
2021-08-23 06:58:01,021|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 requesting 400023 missing
transactions after timeout
2021-08-23 06:58:01,021|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 still missing 400023 transactions
after looking at receivedCatchUpReplies
2021-08-23 06:58:04,908|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 found 196 interesting
transactions in the catchup from Node3
2021-08-23 06:58:04,908|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 merged catchups, there are 196 of
them now, from 399828 to 400023
2021-08-23 06:58:04,908|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 processed 0 catchup replies with
sequence numbers []
2021-08-23 06:58:04,909|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 found 195 interesting
```

```
transactions in the catchup from Node3
2021-08-23 06:58:04,909|INFO|catchup_rep_service.py|Node4:CatchupRepService:3 merged catchups, there are 391 of
them now, from 399633 to 400023
...
2021-08-23 07:38:21,820|INFO|catchup_rep_service.py|CATCH-UP: Node1:CatchupRepService:0 completed catching up
ledger 0, caught up 400023 in total
```

On the other nodes, important log entries that indicate that they connected with the new node are:

```
2021-08-23 06:57:12,845|NOTIFICATION|keep_in_touch.py|Node2's connections changed from {'Node1, Node3'} to
{'Node1', 'Node3', 'Node4'}
2021-08-23 06:57:12,846|NOTIFICATION|keep_in_touch.py|CONNECTION: Node2 now connected to Node4 ()
```

Finally, the node has been properly added.

Troubleshooting:

- Adding the Steward fails. If there is a timeout, the indy ledger probably cannot find consensus, and a sequential restart of all nodes may be necessary
- Make sure that the new node (IP_addresss, ports) is added in the IP_Tables or security settings of all the other nodes. This is not an indy-specific setting but standard security measures (firewall)
- The node immediately crashes when using systemctl (re-)start. This may be a permission issue. Try to run start_indy_node without systemctl by calling start_indy_node with the corresponding parameters.

RB003 - Restarting a node

Version	Date	Author	Comment	Status
0.1	23.08.2021	[redacted]	Initial version	DRAFT
0.2	23.09.2021	[redacted]	Updated Format, extracted from RB001 - Fixing issues with the Indy Network	IN REVIEW

Table of Contents

Restarting the ledger

The following code shows how to restart the nodes and ledger:

Restart all indy-nodes

```
# Restarting the ledger
systemctl stop indy-node && systemctl start indy-node
```

Deleting the data and restarting

When running the node in a docker container, the node data should likely be persisted. If we need a restart just because of a crash or because consensus fails, a simple restart will do. If there are major conflicts, a "clean restart" with an empty ledger and a catchup will be needed; in this case, the node data in the mounted directory needs to be deleted manually.

Stop each node, delete the ledger, and restart the node

```
# Stopping the node:
systemctl stop indy-node
cat /etc/indy/indy_config.py

# find ledger directory:

LEDGER_DIR='/var/lib/indy'
NETWORK_NAME=Chancellory\ Network

# delete the data:
rm -Rf $LEDGER_DIR/$NETWORK_NAME/data/

# restarting the node:
systemctl start indy-node
```

RB004 - Creating and registering DIDs

There are several ways of creating and managing DIDs:

- Use the Web-UI / Block explorer of an Indy network
- Use the aca-py to create a new DID (/wallet/create) or to register it (/ledger/register)
- Use the indy-cli

The indy-cli is recommended as it offers the broadest functionality. The Web-UI should not be used when it runs on another server, as the seed should not be shared with any other party

The indy-cli can be started in several ways:

- With a docker container that has the indy-cli installed, such as `docker run -it docker.io/bcgovimages/von-image:node-1.12-3 indy-cli`
- With a native installation, which will likely only work on an Ubuntu 16 server (see also here):

Installation of the indy-cli

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys CE7709D068DB5E88
sudo add-apt-repository "deb https://repo.sovrin.org/sdk/deb xenial stable"
sudo apt-get update
sudo apt-get install -y indy-cli
```

A DID can be created from random entropy or from a seed, i.e., known entropy. A wallet that contains a DID and the corresponding private key can be copied and moved to another server, BUT: if the DID should be recreated, for example for running a node or using it for an aca-py deployment, the seed must be (1) specified and (2) memorized, e.g., by storing it in a vault.

The basic operations with the indy-cli are as follows. Note that the wallet_key needs to be remembered to unlock the wallet at a later stage.

```
# Create wallet.
$ wallet create <WALLET_NAME> key=<WALLET_KEY>
### Example
$ wallet create test-wallet key=test1234
### Should print a similar output
Wallet "test-wallet" has been created

# Open wallet
$ wallet open <WALLET_NAME> key=<WALLET_KEY>
### Example
$ wallet open test-wallet key=test1234
### Should print a similar output
$ Wallet "test-wallet" has been opened

# Generate DID (Repeat for Trustee and Steward and store the seeds secretly)
$ did new seed=<SEED-PHRASE>

# If the seed is not required because the keys should not be recreated anywhere else, you can also use
$ did new
# but it is advisable to use a seed in most cases

### Example
$ did new seed=test0000000000000000000000000000
### Should print a similar output
$ Did "9735PT7g6A3AgLRMJG1rU6" has been created with "~QNRBSja8p9RV49nrKBz2i9" verkey

# List DIDs
$ did list
### Should print a similar output
```

```
+------------------------+-------------------------+----------+
| Did                    | Verkey                  | Metadata |
+------------------------+-------------------------+----------+
| 9735PT7g6A3AgLRMJG1rU6 | ~QNRBSja8p9RV49nrKBz2i9 | -        |
+------------------------+-------------------------+----------+
| SqhakTa8LR3oU6hPd2maFb | ~PGsmJAmvGXHFEfco7Ts3Zr | -        |
+------------------------+-------------------------+----------+

# This is enough if a DID needs to be used for setting up a new ledger, when collecting the trustees' DIDs and
verkeys.
# If the DID is supposed to be additionally registered on an existing ledger, proceed as follows

# Create a pool
$ mywallet:indy> pool create mypool gen_txn_file=/var/lib/indy/my-net/pool_transactions_genesis
$ mywallet:indy> Pool config "mypool" has been created

# Connect with pool based on a pool_transactions_genesis that is stored in the container (copied, curled, or
mounted)
$ mywallet:indy> pool connect mypool
$ Pool "mypool" has been connected

# Register a DID with one of the roles STEWARD, TRUSTEE, ENDORSER, or NETWORK_MONITOR

# First, pick a DID that has sufficient authorizations, such as a trustee or steward DID
$ pool(pool1):mywallet:indy> did new seed=000000000000000000000000Steward1
$ Did "Th7MpTaRZVRYnPiabds81Y" has been created with "~7TYfekw4GUagBnBVCqPjiC" verkey

$ pool(pool1):mywallet:indy> did use Th7MpTaRZVRYnPiabds81Y
$ Did "Th7MpTaRZVRYnPiabds81Y" has been set as active

# Create the DID that should be registered (if not yet existent)
$ pool(pool1):mywallet:did(Th7...81Y):indy> did new
$ Did "QXHFuTpdkPSKKQXvb1zmxd" has been created with "~AvK2GMPL3vvaGmK29cvQS" verkey

#
$ pool(pool1):mywallet:did(Th7...81Y):indy> ledger nym did=QXHFuTpdkPSKKQXvb1zmxd verkey=~AvK2GMPL3vvaGmK29cvQS
role=ENDORSER
$ Nym request has been sent to Ledger.
$ Metadata:
$ +------------------------+-----------------+---------------------+---------------------+
$ | From                   | Sequence Number | Request ID          | Transaction time    |
$ +------------------------+-----------------+---------------------+---------------------+
$ | Th7MpTaRZVRYnPiabds81Y | 4661            | 1629668575050605460 | 2021-08-22 21:42:55 |
$ +------------------------+-----------------+---------------------+---------------------+
$ Data:
$ +------------------------+------------------------+----------+
$ | Did                    | Verkey                 | Role     |
$ +------------------------+------------------------+----------+
$ | QXHFuTpdkPSKKQXvb1zmxd | ~AvK2GMPL3vvaGmK29cvQS | ENDORSER |
$ +------------------------+------------------------+----------+

# This indicates that the DID was successfully registered with the intended role.
```

RB005 - Bootstrapping a new Issuer

Version	Date	Author	Comment	Status
0.1	02.09.2021	[redacted]	Initial version	DRAFT
0.2	03.09.2021	[redacted]	Ownership transfered to Lukas Willburger	DRAFT
0.3	15.09.2021	[redacted]	Verbalization of sequence diagram	DRAFT
0.4	23.09.2021	[redacted]	Changed structure	IN REVIEW

Table of Contents

Introduction

When adding a new issuer to the SSI architecture, there are multiple steps that need to be conducted by an Indy ledger Trustee and an administrator of the new issuer. This document describes the steps that are necessary to onboard and bootstrap a new issuer. On a high level, the steps are as follows:

- The issuer sets up a Tails Server and a Cloud Agent with the ledger information (transactions genesis file or URL)
- The new issuer creates cryptographic material (DID, keypair) and asks a trustee to register it on the blockchain
- Upon successful registration, the new issuer creates a new credential definition and revocation registry (automatically done by the bootstrapping script in the docker container)

Bootstrapping a new issuer

Given you installed the required docker containers onto your system (aca-py agent, corresponding wallet (= postgres) and Tails Server) and configured their network properly so that they are able to reach each other, you can proceed with the following steps.

The administrator of the new issuer first asks a Trustee on the Indy network for ledger configurations (hence requesting pool_transaction_genesis file or genesis_url). If you work with the file-based version, you need to make sure to reference it correctly in your Docker configuration and mount it into your container. If you use the url-based version, you just need to alter the corresponding Docker environment variable.

The issuer admin then creates a random alphanumerical seed with 32 characters:

Create random alphanumerical string with 32 characters

```
cat /dev/urandom | env LC_CTYPE=C tr -dc 'a-zA-Z0-9' | fold -w 32 | head -n 1
```

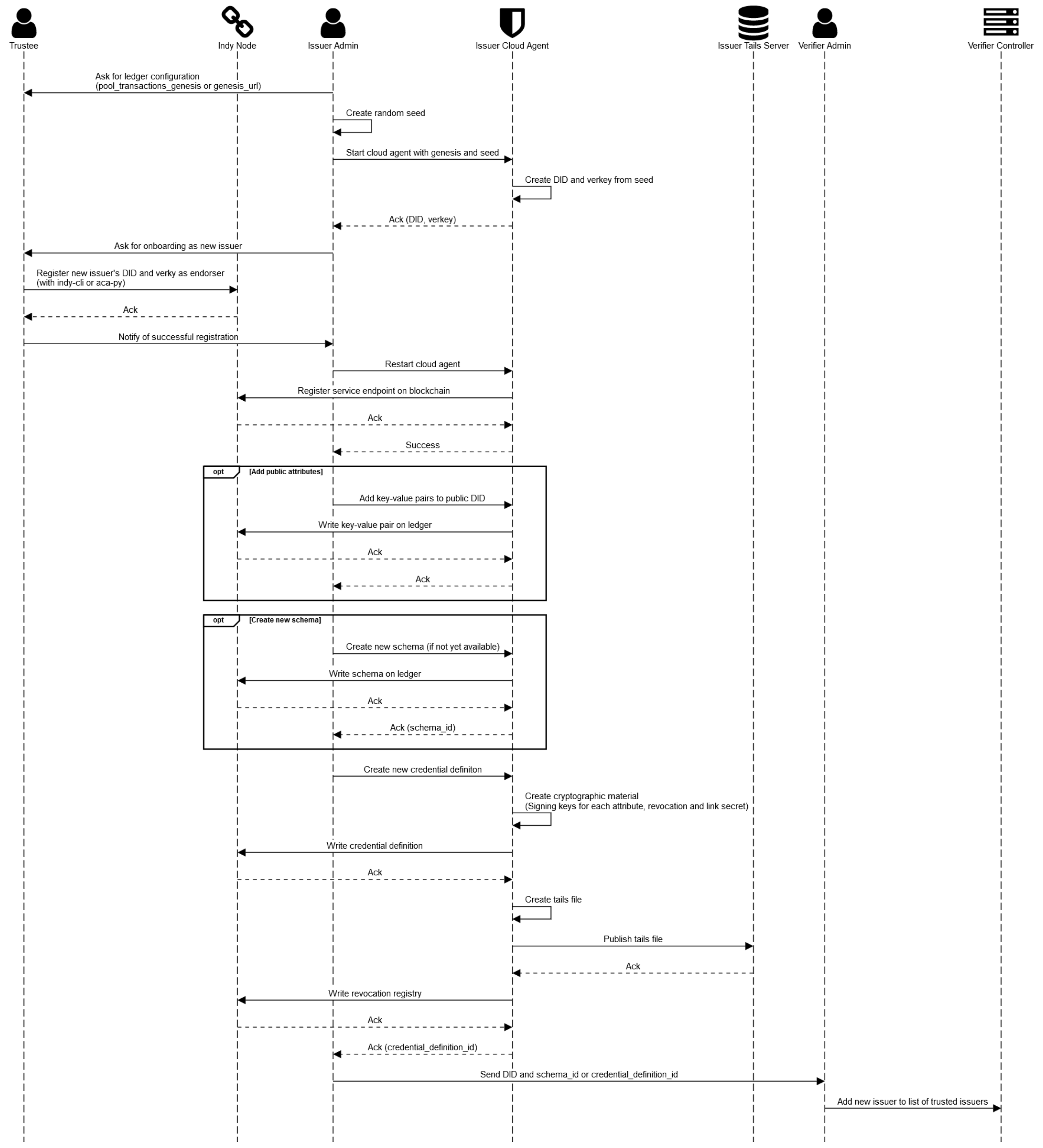
After that, you can proceed with registering a new DID over the indy CLI with the freshly generated seed as described here.

Please note: Whenever the wallet gets fully deleted, you are also going to lose your credential definition tied to your agent. As you cannot use the same name for the credential definition twice, you may need to alter it in the corresponding docker environment variable.

Appendix

Sequence diagram

For a good overview, the steps are illustrated in a sequence diagram.


Trustee
Indy Node
Issuer Admin
Issuer Cloud Agent
Issuer Tails Server
Verifier Admin
Verifier Controller
Ask for ledger configuration (pool_transactions_genesis or genesis_url)
Create random seed
Start cloud agent with genesis and seed
Create DID and verkey from seed
Ack (DID, verkey)
Ask for onboarding as new issuer
Register new issuer's DID and verky as endorser (with indy-cli or aca-py)
Ack
Notify of successful registration
Restart cloud agent
Register service endpoint on blockchain
Ack
Success
opt [Add public attributes]
Add key-value pairs to public DID
Write key-value pair on ledger
Ack
Ack
opt [Create new schema]
Create new schema (if not yet available)
Write schema on ledger
Ack
Ack (schema_id)
Create new credential definiton
Create cryptographic material (Signing keys for each attribute, revocation and link secret)
Write credential definition
Ack
Create tails file
Publish tails file
Ack
Write revocation registry
Ack
Ack (credential_definition_id)
Send DID and schema_id or credential_definition_id
Add new issuer to list of trusted issuers

Bootstrapping Process

Registration/Onboarding (just once)

The administrator of the new issuer first asks a Trustee on the Indy network for ledger configurations (hence requesting pool_transaction_genesis file or genesis_url). The issuer admin creates a random alphanumerical seed with 32 characters and starts the cloud agents with the received genesis file and freshly created seed.

The issuer's cloud agent creates a DID and a verkey, resulting from the seed, and sends an acknowledging message to the issuer admin. The issuer admin then asks the Trustee for permission to be onboarded as a new issuer in the network. The Trustee registers the new issuers's DID and verkey as an endorser (with indy-cli or aca-py) and notifies the issuer admin about the successful registration to the network. The issuer admin restarts the cloud agent that registers the service endpoint on the blockchain.

Creating a credential definition (automatically)

The issuer admin adds the key-value pairs to the public DID via the agent that writes it to the ledger. To create a schema the issuer instructs the agent to write a new schema on the ledger (if not yet available), returning the schema_id. The issuer's cloud agent writes a corresponding (based on the given schema) credential definition to the ledger and creates a tails file. This tail file is then published to the issuer's tails server. In addition, the cloud agent writes the revocation registry to the ledger if required. Eventually, the issuer admin sends the DID and schema_id (or credential_definition_id) to a verifier admin who adds the new issuer to the list of trusted issuers.

RB006 Node Installation & Configuration Checklist

Version	Date	Author	Comment	Status
0.1	13 Sep 2021	[redacted]	Initial version	IN REVIEW

Table of Contents

Checklist / Best Practices

The following checklist compiles a rough overview of best practices for installing and configuring an indy node and ACA-Py agent. Run this checklist to ensure you have installed and configured your indy node / ACA-Py instance correctly.

General

- [ ] Consider an API Gateway component, frontloading the verification/issuing controller to govern, monitor, load-balance and restrict the API endpoints
- [ ] Consider configuring a Web Application Firewall (WAF) to lock down publicly exposed REST endpoints.
- [ ] Consider operating multiple instances of the verification/issuing controllers to increase availability.
- [ ] Use a strong password as an API key for verification/issuing controller and ACA-Py. (see also SEC-DEV-010, SEC-DEV-011, SEC-DEV-012 Security Requirements)
- [ ] Limit access to the production environment.
- [ ] Encrypt attached storage.
- [ ] Backup SEEDs into a secure vault. (e.g. 1Password)
- [ ] Ensure that the SEEDs is randomly generated (32 characters) (Do not use e.g. `test0000000000000000000000000000`)

Indy Node

- [ ] Limit the number of connections and whitelist EESDI Pilot Network Nodes. (IG003 Indy Node Installation, Indy Network Setup#PostProcessing)
- [ ] Ensure that the public IP address of the new node is whitelisted by the other node operators. (Update the Node Information)
- [x] Only publicly expose ports 9701 and 9702.
- [ ] Setup Node monitoring. (Monitoring#CheckscriptConsensus-State)
- [ ] Consider operating two or more nodes ideally in multiple data centers to increase availability.

ACA-Py

- [ ] Use Postgresql for ACA-Py (do not publicly expose the postgresql)
- [ ] Configure ACA-Py, verification/issuing controller with HTTPS-only.
- [ ] Only publicly expose the inbound port of the ACA-Py Installation (admin port should only be reachable internally).

EESDI Pilot Network Nodes

The list of IPs of the different Pilot and Test Nodes can be found here Node Information. This list should be maintained by the network operators. However, to be on the very safe side the real values can be extracted from the blockchain explorer on the node transactions, see the following list for the latest node transactions.

- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]

02 Installation Guidelines

The Installation Guidelines describes the procedures of installing particular solution components required to run parts of the SSI Platform Solution.

ID	Title	Short Description	Owner
IG001	IG001 ACA-PY Installation and Setup, Testnetwork Configuration	Describes how to install and configure the ACA-PY Cloud agent so that it can connect to the test network.	[redacted]
IG002	IG002 Indy Node Monitor Installation	Describes how to install and configure the Indy Node Monitor	[redacted]
IG003	IG003 Indy Node Installation, Indy Network Setup	Describes how to install and start an Indy node and how to set up a new Indy ledger	[redacted]
IG004	IG004 Tails Server Installation	Describes how to install and start a tails server	[redacted]

IG001 ACA-PY Installation and Setup, Testnetwork Configuration

Version	Date	Author	Comment	Status
0.1	29.07.2021	[redacted]	Initial version	DRAFT
0.2	30.07.2021	[redacted]	Proof-reading, minor wording / formatting, incorporating feedback from participants	DRAFT
0.3	24.08.2021	[redacted]	Revised version, updating a few links and editorial improvements	IN REVIEW
0.4	02.09.2021	[redacted]	Ownership transferred to Lukas Willburger	IN REVIEW
1.0	03.09.2021	[redacted]	Approved	APPROVED
1.1	23.09.2021	[redacted]	Added operator guidelines	DRAFT

Table of Contents

Installation

Recommended option: Start aca-py using the docker image that can be built with the Aries Cloudagent in Python Github repository. The recommended version is 0.6.0 (git checkout 0.6.0).

Other options:

- Install dependencies natively (mainly, libindy), e.g., in a native Ubuntu-16-Installation (see IG003 Indy Node Installation, Indy Network Setup) and then run pip install aries-cloudagent=0.6.0.
- Use the verifier docker image provided by IBM (note: the environment variables have another name in this case).

Configuration

Run aca-py start --help for an overview of command-line arguments. The most important ones are briefly described in the following. Each command line parameter can also be set via an environment variable, which can be looked up with aca-py start --help, too.

Parameter	Description
`--auto-provision`	Set flag if the aca-py provision was not previously executed to set up the wallet with the agent's initial key pair and DID
`--seed [$AGENT_SEED]`	The 32 character seed that specifies the entropy from which the key pair and DID are derived deterministically
`--webhook-url "http://$AGENT_WEBHOOK_IP_ADDRESS:$AGENT_WEBHOOK_PORT#$AGENT_WEBHOOK_APIKEY"`	The IP address and port or url to which webhooks are sent. Starting from version 0.6.0, with #, an API key can be specified to enable security on the controller that receives the webhook. Can be left out as long as the aca-py does not communicate with a controller.

`--genesis-url $AGENT_GENESIS_URL`	The URL to the genesis file describes the indy network that the agent should send requests to. Alternatively, you can specify --genesis-file << path to the pool_transactions_genesis file >>
`--wallet-type indy --wallet-local-did`	The flag for an agent that verifies credentials (not including the Basis-ID in the production ledger). Such agents do not require to have a public DID register on-chain. Alternatively, remove this flag to let the agent start in public DID mode, i.e., it will check whether it is registered on-chain and be able to write to the ledger (credential definitions, etc). If the agent is started in public DID mode but its DID is not registered on the ledger yet, it will crash.
`--wallet-name $AGENT_WALLET_NAME`	Custom name for the wallet in which the agent stores keypairs, DIDs, credentials, etc.
`--wallet-key $AGENT_WALLET_KEY`	Key to decrypt the contents of the wallet
`--admin-api-key $AGENT_API_KEY`	API key for the admin API. To disable API key security, set --admin-insecure-mode
`--outbound-transport http`	
`--admin 0.0.0.0 $AGENT_PORT_ADMIN`	The IP address and port (or URL) under which the agent's Admin API can be reached. This is relevant for a controller API that triggers the agent to create connections, credentials, or proof requests
`--inbound-transport http 0.0.0.0 $AGENT_PORT_INBOUND`	
`-e http://$AGENT_IP_ADDRESS:$AGENT_PORT_INBOUND`	The port to which the endpoint is bound. This is the URL address that the connection invitation contains, so the mobile wallet can send an HTTP request to the agent to connect. The IP address and port (or URL) under which the agent can be reached from other agents (typically exposed to the internet). The port must coincide with the one specified in http_inbound_transport
`-l $AGENT_NAME`	The name of the agent displayed in the Swagger UI and in communication with another wallet

`--auto-accept-requests`
 	
`--auto-respond-credential-proposal`
 	
`--auto-respond-credential-offer`
 	
`--auto-respond-credential-request`
 	
`--auto-verify-presentation`
	Auto flags to proceed within a workflow without explicitly needing to trigger the Admin API

`--log-level $AGENT_LOG_LEVEL`	The agent's log level, e.g., DEBUG, INFO, ...

Issuers may additionally use the parameters --tails-server-base-url to specify the url to their indy-tails-server. If the internal address for uploading in the VPN is different from the external IP address for downloading, one can additionally specify --tails-server-upload-url. More details on the tails-server can be found at IG004 Tails Server Installation.

By default, aca-py uses SQLite for storage, which can become problematic when multiple aca-py need to access the same database. For running the aca-py with Postgres, include the following flags, where POSTGRES_URL should be the endpoint of a running Postgres application:

```
--wallet-storage-type postgres
--wallet-storage-config "{\"url\":\"$POSTGRES_URL\", \"wallet_scheme\":\"DatabasePerWallet\"}" \
--wallet-storage-creds "{\"account\":\"$POSTGRES_ACCOUNT_NAME\",\"password\":\"$POSTGRES_PASSWORD\",\"
admin_account\":\"$POSTGRES_ADMIN_ACCOUNT_NAME\",\"admin_password\":\"$POSTGRES_ADMIN_PASSWORD\"}"
```

Sample start configuration

Minimum (hopefully) running example: run on a server with IP_ADDRESS that can be reached from the mobile phone (e.g., private IP address in the home network)

Prerequisites: git installed, docker post-installation permissions configured

How to run aca-py

```
git clone https://github.com/hyperledger/aries-cloudagent-python

cd aries-cloudagent-python

PORTS="8001 8000" ./scripts/run_docker start \
        --auto-provision \
        --genesis-url https://raw.githubusercontent.com/My-DIGI-ID/Ledger-Genesis-Files/main/Test
/pool_transactions_genesis \
        --inbound-transport http 0.0.0.0 8000 \
        --outbound-transport http \
        --admin 0.0.0.0 8001 \
        --endpoint http://$IP_ADDRESS:8000 \
        --label Test-Agent \
        --wallet-name test \
        --wallet-key test \
        --wallet-type indy \
        --wallet-local-did \
        --seed 0000000000000000000000000000Test \
        --admin-insecure-mode \
        --auto-accept-invites \
        --auto-accept-requests \
        --auto-ping-connection \
        --auto-respond-messages \
        --auto-respond-credential-offer \
        --auto-respond-credential-proposal \
        --auto-respond-credential-offer \
        --auto-respond-credential-request \
        --auto-store-credential \
        --log-level DEBUG
```

Tip: If you are using a development ledger, for example, the one provided by von (for instructions, see IG003 Indy Node Installation, Indy Network Setup#Developmentnetwork), you can use the seed 000000000000000000000000Steward1 and remove the --wallet-local-did to obtain immediate authorizations of the aca-py. For the test and pilot ledger, you need to have your DID and Verkey that the agent creates registered by a trustee, for example, IBM, and remove the --wallet-local-did once the onboarding is finished - otherwise, the aca-py will crash.

Testing

Now, you can inspect the agent's Swagger UI at IP_ADDRESS:8001/api/doc

Establish Agent-to-Wallet connection

The typical workflow for end-to-end testing will be:

1. Install the *ID Wallet* app from the App Store (Android / iOS) (alternatively, if not customized features like getting a Base-ID are required, you can also use the LISSI wallet, the esatus wallet or the Trinsic wallet)
2. Receive a verifiabe credential relevant for your use (make sure that it is issued on the Indy ledger that is set in the ID Wallet app).
3. Establish a connection between the agent and the wallet.
   - POST /connections/create-invitation with body {}. The result will look similar to this:

Example connection invitation response body

```
{
   "connection_id": "56aa43c4-4f80-4da1-917a-16bcf65187ec",
   "invitation": {
      "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/connections/1.0/invitation",
      "@id": "308f7cfe-d89b-4b11-8389-85f9849b7124",
      "label": "Test-Agent",
      "serviceEndpoint": "http://IP_ADDRESS:8000",
      "recipientKeys": [
         "2hT4PiSLaCSgvUWUPAyegxQwsbRcisbU3HhXUAPuZqdY"
      ]
   },
   "invitation_url": "http://IP_ADDRESS:8000?
c_i=eyJAdHlwZSI6ICJkaWQ6c292OkJ6Q2JzTlloTXJqSGlxWkRUVUFTSGc7c3BlYy9jb25uZWN0aW9..."// base-64 encoded
value of invitation (sub-json)
}
```

4. Create a QR-Code from the invitation_url (e.g., https://www.qrcode-monkey.com/) and scan it with the wallet.
5. Inspect GET /connections or GET /connections/{connection_id} and check whether the status of the connection under consideration is "active".

Send Proof Request

- **Send a proof request to the wallet:**
  - *POST /present-proof/send-request* with body

Example proof request input body

```
{
   "connection_id": CONNECTION_ID,
   "proof_request": {
      "name": "Test proof request",
      "requested_predicates": {

      },
      "requested_attributes": {
         "TestProp": {
            "name": One of the attribute names from the Basis-ID schema, e.g., FirstName,
            "non_revoked": {
               "from": 0,
               "to": 1612475174 // UNIX TIMESTAMP IN SECONDS AS INTEGER (NOT STRING)
            },
            "restrictions": [
               {
                  "cred_def_id": "MGfd8JjWRoiXMm2YGL4SGj:3:CL:43:Basis-ID Testnetzwerk" //CRED_DEF_ID of
BASE-ID (alternatively, SCHEMA_ID AND/OR ISSSUER_DID)
               }
            ]
         }
      },
      "version": "0.1",
      "nonce": "1234567890"
   },
   "comment": "string"
}
```

- Inspect the result of the verification with *GET /present-proof/records* or *GET /present-proof/{PRES_EXCHANGE_ID}*, specifically, whether **state=verified AND verified=true** (state indicates whether the proof has been checked cryptographically already, and verified whether it was correct) and retrieve the value of the attribute that was asked for, e.g., FirstName.
- For performance reasons in production: *DELETE /present-proof/{PRES_EXCHANGE_ID}* after persisting the required attributes and verification state in a database.

- For **connectionless proofs** that are a bit more intricate but have several benefits, see ART001 Verify SSI Credentials.

Guidelines

From an operator perspective, certain deployment criteria have to be met, in order to ensure a high enough aca-py performance for production use cases.

- [ ] Given a Kubernetes/Container-based deployment, ensure that the aca-py agents have enough system resources: 250mCPU requested and a limit of 750mCPU as well as 256MB RAM requested and a limit of 750MB are advised
- [ ] Always use Postgres as the wallet backend in production environments. If possible, set higher I/O operation speeds for the underlying block storage.
- [ ] Scale the aca-py horizontally to no more than about 20-22 instances. After that, performance will not increase any further.
- [ ] Configure a load balancer/service in front of those aca-py instances, so that requests get processed in a round-robin manor. If an aca-py instance gets to many requests, it will get unresponsive pretty soon
- [ ] Monitor aca-py agents system metrics in order to scale depending on the system usage. Note: readiness and liveness probes are exposed by the aca-py but not as integrated in the system, so use those with caution
- [ ] Start a cron-job at night to delete orphaned entities from the aca-py (for those entities see the development paragraph below).

FAQs / Troubleshooting

- The aca-py crashes with an error indicating that it could not find its DID on the ledger. In this case, either have the agent's DID registered by a TRUSTEE (e.g., IBM) or make sure that the --wallet-local-did flag is set when starting the agent.
- The aca-py crashes with a pool timeout. In this case, it cannot connect to the ledger. Probably, this is an issue with the local security, as ports 9701 and 9702 need to be open not only fo http but for any traffic (the indy-client runs with ZMQ, https://zeromq.org/.
- The wallet cannot connect to the agent when scanning the QR-code generated from the invitation. Make sure that the URL can be reached, e.g., by pasting the invitation-url in another device's or the mobile phone's browser.

IG002 Indy Node Monitor Installation

Version	Date	Author	Comment	Status
0.1	20.08.2021	[redacted]	Initial version	DRAFT
0.2	24.08.2021	[redacted]	Revised version	REVIEW

Table of Contents

Node Monitor Installation

The Indy Node Monitor repository on GitHub contains all necessary code to run the monitoring tool. The only two prerequisites to run it include git and docker. Docker is used to run an Indy VDR based on the bcgovimages/von-image.

1. To use the script, the pool transactions genesis (either provided as a file or via URL) and a seed are needed as described here. Normally, using any network Trustee seed should work fine. Nevertheless, it is advisable to create a new DID (RB004 - Creating and registering DIDs) with just the NETWORK_MONITOR role for security reasons on the ledger that needs to be monitored.
2. Make sure git and docker are installed: sudo apt install git && sudo apt install docker.io
3. Clone the Indy Node Monitor repository: git clone https://github.com/hyperledger/indy-node-monitor.git
4. Change into the right directory: cd indy-node-monitor/fetch-validator-status
5. Run the monitoring script:
   a. ./run.sh --genesis-url=<URL> --seed=<SEED> or
   b. ./run.sh --genesis-file=<FILE_PATH> --seed=<SEED>
   c. alternatively, you can set the genesis URL or file of the Indy ledger that is supposed to be monitored via environment variables: export GENESIS_URL=<URL> or export GENESIS_FILE=<FILE_PATH>
6. If you are planning on using the file-based approach, you have to add a volume from your local machine to the container. This means editing the run.sh script: add *-v <LOCAL_PATH>:/home/indy/pool_transactions_genesis* after line 37. Then you can *export GENESIS_FILE=/home/indy/pool_transactions_genesis* for setting the right genesis file path and running the run.sh script.

Troubleshooting

In case your first run of ./run.sh just gives you a short JSON file without any useful information and containing exceptions (e.g., an element is non-iterable), it's most probably due to a timeout we experienced in testing the indy node monitor. When querying the validator information, journalctl is also checked for exceptions on each node. As the corresponding command in the indy-sdk and hence the indy node monitor implementation has a 5-second timeout, it fails for large journalctl logs (which by experience can have a size of several GB). To work around that, it is advisable to clean those logs from time to time or set a standard time of e.g. two days, such that older entries in the journal will be cleaned continuously: *journalctl --vacuum-time=2d.*

IG003 Indy Node Installation, Indy Network Setup

Version	Date	Author	Comment	Status
0.1	24.08.2021	[redacted]	Initial version, taken largely from Indy Node Hosting in the hotel-checkin use case	DRAFT
0.2	29.08.2021	[redacted]	Reviewed	IN REVIEW
0.3	02.09.2021	[redacted]	Small adjustments; ownership transferred to Lukas Willburger	IN REVIEW
1.0	10.09.2021	[redacted]	fixed public IP to 0.0.0.0	APPROVED

Table of Contents

Indy Node Installation

Hardware and OS Requirements for an Indy node

- Ubuntu (Xenial) 16.04 LTS
- >= 2 vCPUs
- >= 8 GB RAM
- >= 100 GB Festplattenspeicher
- external (routable) IP-address

Other settings

- Default Ports
  - 9701: Indy Interledger-Communication
  - 9702: Client
- Recommended indy-node version: 1.12.4

Native installation

Install Indy Node Step-By-Step

```
# add sovrin source for indy-node and install indy-node in the provided version

sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys CE7709D068DB5E88
sudo bash -c 'echo "deb https://repo.sovrin.org/deb xenial stable" >> /etc/apt/sources.list'     # if indy-node
installation fails, make sure there is a new line in sources.list for the sovrin repository
sudo apt-get update
sudo apt-get install indy-node=1.12.4     # (or another version if agreed on)


# set local NETWORK_NAME in the node config

sudo vi /etc/indy/indy_config.py
NETWORK_NAME = '<NEW_NETWORK_NAME>'


# initialize indy-node (generate cryptographic material for the node)

sudo -i -u indy init_indy_node <NODE_NAME> 0.0.0.0 9701 0.0.0.0 9702 <SEED-PHRASE>
```

```
# The logs that display the cryptographic material created should look similar to:

Node-stack name is Node0
Client-stack name is Node0C
Generating keys for provided seed da2857ba6e8550cd3d981e09ad262a44
Init local keys for client-stack
Public key is 62KCsA8AtDmbtMNj3fzrVqxNPMVjpJgbAsBCyRvs62QB
Verification key is AcdZwPDqm7k6NEUQuTPPBKcP2q3TLGBvMmfGiNHzb9A5
Init local keys for node-stack
Public key is 62KCsA8AtDmbtMNj3fzrVqxNPMVjpJgbAsBCyRvs62QB
Verification key is AcdZwPDqm7k6NEUQuTPPBKcP2q3TLGBvMmfGiNHzb9A5
BLS Public key is
39xnpJeZUph63UtKmWxNbJrd1fhCpArkTRBKwTcKDxTFMhotfaDWd8NPpZYULCJicx6N8ghpUi1uRbckGZWMhN9FXz1ZDgWSJx1kyQpkYDA465vP
iGWYae9CGS2nSfe9ySBNa78qtfvsZXpsLJtcX99aXFT79vdihxFcGne3cku8R3U
Proof of possession for BLS key is
RXzc4jEDeTkbcK6no5AB1LBdqwWyAaCtsQWWUPjQjqdAKYkiL1WuCxGCr8FYLCCRKDhHeGqDoeNitkLGtdTiRrJCEHmtTPSBUau1AagQnJApy5qW
HTEjsj7CHPKwVsJ5SPSxTAEU2rNX9nYXfiMLtG9R7GCsB86UkEqPHt8JYNMnTy


# copy the genesis files provided (see below) onto your system, move them to the required directory and provide
the indy user the needed privileges

sudo chown indy:indy pool_transactions_genesis
sudo chown indy:indy domain_transactions_genesis

sudo mv domain_transactions_genesis /var/lib/indy/<NETWORK_NAME>
sudo mv pool_transactions_genesis /var/lib/indy/<NETWORK_NAME>


# Go-Live: run the node

sudo systemctl start indy-node
sudo systemctl status indy-node
sudo systemctl enable indy-node

# You can find the node logs at /var/log/indy/<NETWORK_NAME>/<NODE_NAME>.log
```

Post Processing

Limiting the number of connections and whitelisting other nodes

```
# install tool to persist iptable rules
sudo apt install iptables-persistent


# switch to a bash with root privileges
sudo bash

# create new chain
iptables -N LOG_CONN_REJECT

# log-prefix
iptables -I LOG_CONN_REJECT -j LOG --log-prefix "connlimit: "

# Append a rule that finally rejects connection
iptables -I LOG_CONN_REJECT -p tcp -j REJECT --reject-with tcp-reset

# setting the connection limits
iptables -A INPUT -p tcp -m tcp --dport 9702 --tcp-flags FIN,SYN,RST,ACK SYN -m connlimit --connlimit-above 500
--connlimit-mask 0 --connlimit-saddr  -j LOG_CONN_REJECT

# add rule for the different nodes

# current_validators --writeJson | node_address_list
# For each node
iptables -I INPUT -i <INTERFACE> -p TCP -s <IPADDR> --sport 1024:65535 --dport <NODE-PORT> -m state --state NEW,
ESTABLISHED,RELATED -j ACCEPT


# Drop all other connections on node port
iptables -A INPUT -i <INTERFACE> -p TCP --dport <NODE-PORT> -m state --state NEW,ESTABLISHED,RELATED -j DROP

# save iptable rules
iptables-save > /etc/iptables/rules.v4

# exit sudo bash
exit
```

There is also a Readme for Hyperledger Indy that presents the steps in a similar way: https://hyperledger-indy.readthedocs.io/projects/node/en/latest/start-nodes.html

Docker installation

We recommend the native Ubuntu 16 setup, and an update for Ubuntu 18 / 20 is currently being developed in IDunion. For those who want to try native installation on Ubuntu 18 or 20 earlier, there is a bootstrapping script in the Distributed Ledger Performance Scan Github Repository that is maintained from time to time by [redacted]. It describes how to install Hyperledger Ursa and the Python packages for indy-node. However, it is not very stable.

However, alternatively, an indy node can also be run in a docker container; the corresponding image is bcgovimages/von-image:node-1.12-3.

There is no systemctl command for initializing the node, but it is possible to work with

```
init_indy_keys --name <NODE_NAME> --seed <NODE_SEED>
```

and

```
start_indy_node <NODE_NAME> <PUBLIC-IP> 9701 <PUBLIC-IP> 9702 <SEED-PHRASE>
```

(first the node IP and port, then the client IP and port; the node and client IP will usually be the same).

Indy Network Setup

Development network

For testing purposes, one can run a 4-node indy test ledger on a single server with the indy-von-network:

Run a Centralized Indy Testnet

```
git clone https://github.com/bcgov/von-network;
cd von-network
./manage build
./manage start <<PUBLIC_IP>>
```

Test, pilot, and production network

To set up a real, decentralized Indy network, proceed as follows:

- Each node operator needs to generate two new DIDs – one for the node (from the steward seed) and one for the trustee associated with the node operator. For this, it is recommended to use the indy-cli, as described in RB004 - Creating and registering DIDs.
- Each node operator initializes their node with the steward seed and collects the command line outputs, together with the trustee DID information, in a table.

NODE (Steward)	Alias	Node-IP	Node-IP ausgehend (falls abweichend)	Node Port	Client-IP	Client Port	Verification key	BLS Public key	BLS Key PoP (Proof of Possession)	TRUSTEE	DID (Trustee)	Verkey (Trustee)	STEWARD

	Node-Name	<<NODE_IP>>		9701	<<NODE_IP>>	9702	DF9XCzFE12...	2Vw6Zes81W...	Qwn4vN7Nx647FeH...		PuJU8cnSUr...	~9vNmWu7ftS7...	

- This public information needs to be collected from each node operator by some entity (so far, it has been esatus) and combined in a csv file. Using a Python script, which is available here, the entity builds the initial pool_transactions_genesis and the domain_transactions_genesis and distribute them to all node operators. The entity does not need to be trusted as each node operator should check the final pool_transactions_genesis and domain_transactions_genesis whether the information on their own node and DIDs is correct.
- The information collected by the entity could look as the table above, with one row for each node operator. It contains all the information relevant for all nodes.
- Also, all node operators need to whitelist all other nodes' IP addresses and ports on their node.
- Once this has been accomplished, the node operators copy the pool_transactions_genesis and the domain_transactions_genesis to their node and start it, as described in the section Indy Node Installation above.

Related Documents

- see also RB006 Node Installation & Configuration Checklist to verify your installation.

IG004 Tails Server Installation

Version	Date	Author	Comment	Status
0.1	24.08.2021	[redacted]	Initial version	DRAFT
0.2	02.09.2021	[redacted]	Adding docker-compose information, Ownership transferred to Lukas Willburger	DRAFT
1.0	03.09.2021	[redacted]	Minor editorial changes - approved.	APPROVED

Table of Contents

Installation and Startup

The indy tails server allows to publish tails-files that the ID wallet needs to download once on the first proof with the verifiable credential to prove non-revocation. The tails-server needs to be run by the issuer of the verifiable credential. For the reasons why the tails-server is used for publishing tails-files as opposed to Github or another file store, see ARC004 Publish Tails Files.

The indy tails server is available in a separate Github repository. Running the tails server is simple:

Install and run tails server

```
# Get the indy tails server repository
git clone https://github.com/bcgov/indy-tails-server

# Run the indy tails server (./manage calls docker-compose in the background)
cd indy-tails-server/docker && ./manage up
```

With the indy-tails-server repository, it is also possible to refer to the Dockerfile in docker-compose or with other orchestration tools.

Parameters

There are a few relevant environment variables that should be set (e.g., with docker compose):

Parameter	Description
`--storage-path` `--$TAILS_SERVER_STORAGE_PATH`	For persisting the tails files. The specified path (e.g., /home/indy/.indy_client) should be mounted in this case
`--host $TAILS_SERVER_IP_ADDRESS`	IP address to bind to (default: 0.0.0.0)
`--port $TAILS_SERVER_PORT`	Port (default: 6543)
`--log-level $TAILS_SERVER_LOG_LEVEL`	The log level (for instance, debug, info)

When running together with an aca-py instance for issuance, make sure that --tails-server-base-url is set to http://<TAILS_SERVER_HOST>:<TAILS_SERVER_PORT> (and potentially --tails-server-upload-url is set accordingly).