Sovrin Steward
Validator Preparation Guide

Purpose

The purpose of this document is to help you to set up a Validator node on Sovrin networks, as well as to set up a CLI machine for using the indy-cli. The CLI will be used now and in the future to post transactions to the networks as a Sovrin Steward.

Table of Contents

1. Introduction

This document will be periodically updated.

Before reading the contents of this document it is highly recommended that you have a good understanding of the Sovrin Governance Framework. Another resource is the Sovrin Steward FAQ. You will also need to join the Sovrin Team on Sovrin Foundation Slack for important communications regarding updates, news, and requests from the Network Operations team. Important channels include #stewards #steward-private and #validator-support..

1.1. High Level Overview

Sovrin Stewards operate the Validator nodes of the Sovrin decentralized identity network. All Stewards must be approved by the Sovrin Foundation Board of Trustees. The qualifications for becoming a Steward are described in the Steward Technical Policies and Steward Business Policies of the Governance Framework.

Having a Validator node will allow your organization to be part of what is called **consensus**. Consensus is a protocol spoken between all of the validator nodes on the network when they come together to agree upon modifying the ledger. In essence, they say, “This is what is currently on the ledger, we agree about the nature and order of these new items that should be added, and we agree that this is what the ledger will look like when we’re done.”

By design, as a Steward, you are only allowed to operate one Validator node per network.

1.2. Looking Forward: Observer Nodes

Validator nodes on Sovrin have practical limits; the distributed consensus at the heart of Sovrin slows down as more Validators are added.

As Sovrin grows, we plan for hundreds of nodes to be run by Stewards, with some handling writes and many more providing a hot-swappable read cache (like a CDN). These latter nodes will be called **observers**. A node's status as a Validator (handling writes) or observer (handling reads) may change over time, either statically or dynamically, as determined by the Technical Governance Board and Steward Council. The Foundation will likely continue to add different testing Networks for development and security testing.

1.3. Hyperledger Indy and Indy-SDK

All Stewards run the same codebase—open source Validator node software developed under the Hyperledger Indy project hosted and managed by the Linux Foundation. Not only does this enable Stewards (and everyone else in the ecosystem) to be confident in the code running the Sovrin network, but in the future it will enable all Sovrin Steward nodes (both Validators and observers) to monitor each other's performance and upvote or downvote their peers, dynamically maintaining the health of the network. (Note: this reputation system is not developed yet, and it will not undo Founding Steward status for a Steward.)

2. Preliminaries to the Set Up

Before you start this process, you'll need to gather a couple of things and make a few decisions.

As you proceed through these steps, you will be generating data that will be needed later. As you follow the instructions and obtain the following, store them for later use:

- Your Steward key seed
    - This is extremely important and it must be kept **safe and private**. It is used to generate the public / private key pair that you will use as a Steward to post transactions to the ledger.
- Your Steward distributed identity (DID)
    - This is public information that is generated by your Steward key seed. It is an identifier for your organization that will be written to the ledger by a Sovrin Trustee.
- Your Steward verification key (verkey)

    - This is public information that is generated by your Steward key seed. It will be written to the ledger by a Sovrin trustee along with your DID, and will be used to verify transactions that you write to the ledger
- The Validator IP Address for inter-node communications
    - This IP address must be configured for exclusive use of inter-node consensus traffic. Ideally, traffic to this address will be whitelisted in your firewall.
- The Validator node port
- The Validator IP Address for client connections
    - This IP address must be open for connections from anywhere, since clients around the world will need to be able to connect to your node freely.
- The Validator client port
- The Validator alias
- The Validator node seed
    - This is distinct from your Steward seed, and will generate public and private keys that your Validator will use for communications with other Validators. Like the Steward seed, it should be **kept secure**.
- The Validator Node Identifier
    - This is distinct from your Steward verkey. It is also public information that will be placed on the ledger, but is used as a public key by your Validator node, rather than by you, the Steward.
- The Validator BLS public key.
    - Used by the Validator to sign individual transactions that will be committed to the ledger. It is public information that will be written to the ledger.
- The Validator BLS key proof-of-possession (pop)
    - A cryptographic check against certain forgeries that can be done with BLS keys.

2.1. Two Machines

You'll need two machines: one is your Validator node and the other a CLI machine to run a CLI with which you will interact with the ledger. They can be actual physical machines, virtual machines, or a combination. The machine with the CLI can be turned on and off at your convenience (e.g., it could be a VM on a laptop); only the Validator node needs to be public and constantly up.

> **Important: for security reasons, you must not use your Validator node as a CLI client. If you do, it could expose your Steward credentials needlessly.**

Your Validator **must run Ubuntu 16.04 (64-bit)** as this is the only version we have prebuilt packages for. This guide presupposes that your CLI machine will run in Ubuntu as well.

Your Validator node should have two NICs, each with associated IP addresses and ports. One NIC will be used for inter-validator communication, and the other for connections from clients, including Sovrin edge agents, as well as ssh and other connections you use for administration.

This two NIC approach is required as a defense against denial-of-service attacks, since the NIC for inter-validator communications should be behind a firewall that is configured to be much more restrictive of inbound connections than the client-facing NIC is.

It is currently possible to have just one NIC and IP address, as the transition for older Stewards to change to 2 NICs is ongoing. The inability to or delay of adding a second NIC will likely affect which network your node will be placed on. A resource that may help you to configure your node to use two NICs is in an appendix at the end of this document.

2.2. Validator Node Preliminary Information

Get the IP Addresses

Your Validator node will be the machine that will become a part of the Sovrin network. It should have two static, publicly accessible, world routable IP addresses. It should be configured so that outgoing TCP/IP connections are from these same address, as well as incoming connections.

Obtain IP addresses that meet this requirement.

Choose Port Numbers

The Validator node will also be required to have the following:

- **Node Port: TCP** - The Validators use this IP address and port combination to communicate with each other.
- **Client Port: TCP** - Clients use this IP address and port combination to communicate with the Validator node, to interact with the ledger.

By convention, please choose ports 9701 and 9702 for your Node and Client ports, respectively.

Choose an Alias:

Your Validator node will need to have an alias. This will be used later when we create a key for the node. It can be any convenient, unique name that you don't mind the world seeing. It need not reference your company name, however it should be distinguishable from the other Validator nodes on the network. Many Stewards choose a Validator alias that identifies their organization, for pride of their contribution to the cause of self-sovereign identity.

3. Setup and Configuration

You must perform the following instructions in order. If you require assistance, please contact support@sovrin.org or post to support in the Sovrin Foundation Slack. Some instructions must

be executed on the Validator node, and others on the CLI machine. The command line prompts in the instructions will help remind you which machine should be used for each command.

3.1. CLI Node Installation

3.1.1. Install the CLI

On the machine you've chosen for the CLI, open a terminal and run the following lines to install the indy-cli package.

In the CLI node:

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

3.1.2. Add an Acceptance Mechanism

To write to a Sovrin Ledger, you'll need to sign the Transaction Author Agreement. This Agreement is incorporated into the process of connecting to the node pool and requires an acceptance mechanism. For the Indy CLI, the default mechanism is "For Session" and the following instructions are required to be able to use "For Session" for your CLI:

Create a JSON Config file containing your taaAcceptanceMechanism. (You can also add plugins to this config file, but for now just set it up as basic as possible.)

This example cliconfig.json file contains the line that sets the AML:

```
{
        "taaAcceptanceMechanism": "for_session"
}
```

To start the indy-cli using your new config file, run the following:

```
ubuntu@cli$ indy-cli --config <path_to_cfg>/cliconfig.json
```

Now all of the appropriate transactions will have an "Agreement Accepted" authorization attached to them during this CLI session.

3.1.3. Obtain the Genesis Files

Obtain the genesis transaction files for the Sovrin Networks with the following steps. These files contain bootstrap information about some of the Validator nodes, which will be used by your CLI to connect to the networks.

If you are in the indy prompt, please exit:

```
indy> exit
```

Most Stewards will currently be onboarded to the BuilderNet. Obtain the genesis transaction file for it:

```
ubuntu@cli$ cd
```

```
ubuntu@cli:~ $ curl -O
https://raw.githubusercontent.com/sovrin-foundation/sovrin/master/sovrin/pool_transactions_builder_genesis
```

You will also want to obtain the genesis files for the StagingNet and MainNet, for the possibility of moving between networks:

```
ubuntu@cli:~ $ curl -O
https://raw.githubusercontent.com/sovrin-foundation/sovrin/stable/sovrin/pool_transactions_sandbox_genesis
ubuntu@cli:~ $ curl -O
https://raw.githubusercontent.com/sovrin-foundation/sovrin/stable/sovrin/pool_transactions_live_genesis
```

3.1.4. Generate the Steward Key

Next, generate a Steward key using the CLI machine you just installed. This will be comprised of a public and private key pair, generated from a seed. Knowing your seed will allow you to regenerate the key on demand. *To keep this secure, you will need to have a **very** secure seed that is not easy to guess.*

Generate a Seed

WARNING:

You want to guard your seed well. The seed will be used to generate your public (verification) key as well as your secret private key. If your seed falls into the wrong hands, someone could regenerate your private key, and take over your identity on the ledger. Keys can be rotated, which can stop some of the damage, but damage will still have been done.

In the terminal, run the following to install a good random string generator, and then use it to generate your seed:

```
ubuntu@cli$ sudo apt install pwgen
ubuntu@cli$ pwgen -s 32 1
```

EXAMPLE:

```
ubuntu@cli$ pwgen -s 32 -1
ShahXae2ieG1uibeoraepa4eyu6mexei
```

IMPORTANT:

Keep this seed in a safe place, such as an encrypted password manager or other secure location designated by your organization. You will need it later in this guide, as well as in the future for other Steward interactions with the ledger.

Run the Indy CLI and generate key

Next we run the indy-cli command line CLI by entering:

```
ubuntu@cli$ indy-cli --config <path_to_cfg>/cliconfig.json
```

In the command line, enter the following to create your pool configuration and your wallet locally. In these instructions, we use "buildernet" for the pool name and "buildernet_wallet" for the wallet name, although you may use other names of your choosing, if desired. The encrypted wallet will be used to store important information on this machine, such as your public and private keys. When creating your wallet, you will need to provide a "key" that is any string desired. It will be the encryption key of your local wallet.

```
indy> pool create buildernet gen_txn_file=pool_transactions_builder_genesis
indy> wallet create buildernet_wallet key
```

Upon entering this command, you'll see a prompt to enter your wallet key. Enter the key and hit enter.

IMPORTANT:

To be able to retain your wallet and not re-create it when you need it in the future, keep this wallet key in a secure location as well.

Using the pool configuration and wallet you have created, connect to the pool and open the wallet:

```
indy> pool connect buildernet
```

When you connect to a Network with TAA enabled, you will be asked whether you want to view the Agreement. Type ‘y’ to accept to see the Agreement, then select ‘y’ again to accept the Agreement displayed. If you do not accept the agreement, then you will not be allowed to write to the Network.

```
indy> wallet open buildernet_wallet key
<enter the key>
```

Using the seed that you generated with pwgen, place your public and private keys into your wallet.

```
indy> did new seed
<enter the seed>
```

The result should look something like this:

```
Did "DIDDIDDIDDIDDIDDIDDID" has been created with "~VERKEYVERKEYVERKEYVERKEY"
verkey
```

IMPORTANT: Save the “DID” and “verkey” portions of this. They are not secret, but they will be used when you are prompted to supply your Steward verkey and DID.

3.2. Validator Node Installation

3.2.1. Perform Network Test

This test is to confirm that your Validator node can connect with external devices.

Note that the communication protocol used for both node and client connections is ZMQ. If your firewall uses deep packet inspection, be sure to allow this protocol bi-directionally.

The tests in this section are to assure that your node's networking is operational, and that firewalls will allow TCP traffic to and from your IP addresses and ports. The assumptions are that for this stage of testing, you will be able to reach both sets of IP address/port combinations from an arbitrary client, but that later you will implement rules on your firewall restricting access to your node (inter-validator) IP address/port.

3.2.1.1Test the node (inter-validator) connection to your Validator

Use netcat to listen on the "node" IP address and port on your Validator

IMPORTANT:
Many providers, such as AWS, use local, non-routable IP addresses on their nodes and then use NAT to translate these to public, routable IP addresses. If you are on such a system, use the **local address** to listen on, and the **public address** to ping with.

```
ubuntu@validator$  nc -l <node_ip_address> <node_port>
```

The above command will wait for a connection. On a system that can be used as a client, such as your CLI node, do a TCP ping of that IP address and port:

```
ubuntu@cli$  nc -vz <node_ip_address> <node_port>
```

If the test is successful, the ping will return a "succeeded" message and the commands on both nodes will exit.

3.2.1.2 Test the client (edge agent) connection to your Validator

Repeat the above test on your Validator and a test client, but using the Validator's "client" IP address and port.

Important: The “client” IP address referred to here is not the CLI machine’s IP address.
Reminder: The Validator node has a node IP address for communications with other Validators and a “client” IP address for communications with edge agents (anything outside the Sovrin Network of Validators).

On your Validator:

```
ubuntu@validator$  nc -l <client_ip_address> <client_port>
```

On your client:

```
ubuntu@cli$  nc -vz <client_ip_address> <client_port>
```

If the test is successful, the ping will return a "succeeded" message and the commands on both nodes will exit.

IMPORTANT:
If your system uses NAT, the same approach should be used as above.

3.2.1.3 Test the connection from your node to another Validator on the BuilderNet

One of the Validator nodes on the BuilderNet is named "FoundationBuilder", which has a node IP address and port of 50.112.53.5 and 9701, respectively. On your Validator, make sure that your node is able to connect to this node on BuilderNet by TCP pinging its node IP address and port:

```
ubuntu@validator$ nc -vz 50.112.53.5 9701
Connection to 50.112.53.5 9701 port [tcp/*] succeeded!
```

When the above three tests are successful, you may proceed.

3.2.2. Install the Validator Node

Continue on your Validator node machine.

> **Important: You must use a login user with sudo privileges (not root or indy) to run these commands, unless otherwise indicated.**

```
ubuntu@validator$ sudo apt-key adv --keyserver keyserver.ubuntu.com
--recv-keys CE7709D068DB5E88
ubuntu@validator$ sudo apt-get install -y software-properties-common
ubuntu@validator$ sudo add-apt-repository "deb https://repo.sovrin.org/deb
xenial stable"
ubuntu@validator$ sudo apt update
ubuntu@validator$ sudo apt upgrade -y
ubuntu@validator$ sudo apt install -y sovrin
```

3.2.3. Create the Key for the Validator Node

IMPORTANT:
Many providers, such as AWS, use local, non-routable IP addresses on their nodes and then use NAT to translate these to public, routable IP addresses. If you are on such a system, use the **local addresses** for the init_indy_node command.

Please run the following on the Validator before running init-indy-node.

1. In the /etc/indy/indy_config.py file, **change the Network name from "sandbox" to "net3"** (use sudo to edit the file or use `sudo sed -i -re "s/(NETWORK_NAME = ')\w+/\1net3/" /etc/indy/indy_config.py`) then run the following commands:

2. sudo -i -u indy mkdir /var/lib/indy/net3
3. cd /var/lib/indy/net3
4. sudo curl -o domain_transactions_genesis https://raw.githubusercontent.com/sovrin-foundation/sovrin/master/sovrin/domain_transactions_builder_genesis
5. sudo curl -o pool_transactions_genesis https://raw.githubusercontent.com/sovrin-foundation/sovrin/master/sovrin/pool_transactions_builder_genesis
6. Make sure that both genesis files are owned by indy:indy sudo chown indy:indy *

Enter the following where <ALIAS> is the alias you chose for your Validator node machine and <node ip>, <client IP>, <node port #> and <client port #> are the correct values for your Validator.

Note: the node IP and client IP addresses should be the LOCAL addresses for your node.

```
ubuntu@validator$ sudo -i -u indy init_indy_node <ALIAS> <node ip> <node port> <client ip> <client port>
```

You will see something like this (highlighting added):

```
Node-stack name is Node19
Client-stack name is Node19C
Generating keys for random seed b'FA7b1cc42Da11B8F4BC83990cECF63aD'
Init local keys for client-stack
Public key is
a9abcd497631de182bb6f767ffb4921cdf83ffdb20e9d22e252883b4fc34bf2f
Verification key is
3d604d22c4bbfd55508a5a7e0008847bdeccd98a41acd048b500030020629ee1
Init local keys for node-stack
Public key is
a9abcd497631de182bb6f767ffb4921cdf83ffdb20e9d22e252883b4fc34bf2f
Verification key is
bfede8c4581f03d16eb053450d103477c6e840e5682adc67dc948a177ab8bc9b
BLS Public key is
4kCWXzcEEzdh93rf3zhhDEeybLij7AwcE4NDewTf3LRdn8eoKBwufFcUyyvSJ4GfPpTQLuX6iHjQwnCCQx4sSpfnptCWzvFEdJnhNSt4tJMQ2EzjcL9ewRWi24QxAaCnwbm2BBGJXF7JjqFgMzGfuFXXHhGPX3UtdfAphrojk3A1sgq
Proof of possession for BLS key is
QqPuAnjnkYcE51H11Tub12i7Yri3ZLHmEYtJuaH1NFYKZBLi87SXgC3tMHxw3LMxErnbFwJCSdJKbTb2aCVmGzqXQtVWSpTVEQCsaSm4SUZLbzWVoHNQqDJASRYNbHH2CqpR2MtntA4YNb2WixNSZNXFSdHMbB1yMQ7XUcZqtGHhcb
```

Store the original command, the random seed, the verification key, the BLS public key, and the BLS key proof-of-possession (POP). These are the keys for your Validator node (not to be confused with the keys for you in your Steward role). The Validator verification key and BLS key (with its POP) are public, and will be published on the ledger.

The random seed should be protected from disclosure.

3.3. Run the Technical Verification Script

Download this script, upload it to your Validator node, and set the execution flag on it:

```
ubuntu@validator$ cd ~
ubuntu@validator$ curl -O
https://raw.githubusercontent.com/sovrin-foundation/steward-tools/master/steward_tech_check.py

ubuntu@validator$ chmod +x steward_tech_check.py
```

Execute it, answering the questions that it asks. There are no wrong answers; please be honest. Questions that can be answered by scripting are automatically completed for you.

```
ubuntu@validator$ sudo ./steward_tech_check.py
```

After the script completes, copy the output beginning at '== Results for "A Steward MUST" ==', and paste it into an email addressed to support@sovrin.org then send it.

3.4. Provide Information to Trustees

At this point you should have the following data available:

- Your Steward verkey and DID
- The Validator ‘node IP address’
- The Validator ‘client IP address’
- The Validator ‘node port’
- The Validator ‘client port’
- The Validator alias
- The Validator verkey
- The BLS key

Please go to the Steward Validator Registration form and provide the requested information.

Note: You are done with the first part of the onboarding. The Sovrin Foundation staff will contact you to set up the rest.

3.5. Add Node to a Pool

After your data is submitted via the Steward Validator Registration form, a Sovrin Trustee will put your Steward public key into the ledger. You will receive notification when your DID and verkey have been added to the ledger. You will be asked to work together with a Sovrin TechOps engineer to complete the final steps to onboard your node onto the network. Please be prepared to suggest times to do this together online.

IMPORTANT:
DO NOT proceed further with this document until your DID and verkey (the public key) is on the ledger.

3.5.1. Configuration

After you have been informed that your public key has been placed onto the ledger of the Sovrin BuilderNet, you may complete the configuration steps to activate your Validator node on that network.

Make Sure Your Version Is Current

In some cases, some time may have passed before reaching this point. You should ensure that you have the current version of indy software installed before proceeding. On the Validator node, execute the following.

In the Validator node:

```
ubuntu@validator$ sudo apt update
ubuntu@validator$ apt-cache policy sovrin
```

If your installed version is not the newest, contact support@sovrin.org to make sure that what you have is compatible with the Sovrin BuilderNet. If needed, upgrade your version:

```
ubuntu@validator$ sudo apt install sovrin=<version_number>
```

On the CLI machine, execute the following.

In the CLI machine:

```
ubuntu@cli$ sudo apt update
ubuntu@cli$ sudo apt upgrade indy-cli
```

Add Validator Node to Ledger

On your CLI machine, if you are not still on the indy-cli prompt, you will need to return to it. To get back to where you were, type `indy-cli --config <path_to_cfg>/cliconfig.json`, connect to the network pool, designate the wallet to use (using the same wallet key as before), and enter the DID that was returned earlier, when you typed '`did new seed`' (then enter your seed) for your Steward user:

```
ubuntu@cli$ indy-cli --config <path_to_cfg>/cliconfig.json
indy> pool connect buildernet
indy> wallet open buildernet_wallet key=<wallet_key>
indy> did use <your_steward_DID>
```

- *Note: You may need to create a new wallet and run "`did new seed`" then enter `<your_steward_seed>` instead, if you did not save your wallet or forgot your wallet key.*

If the connection is successful, enter the following, substituting the correct data as appropriate. An example will follow to be more clear.

- *Suggestion: Edit this in a text editor first, then copy and paste it into the Indy CLI. Some editors will insert 'smart quotes' in place of regular ones. This will cause the command to fail.*

```
indy> ledger node target=<node_identifier>
node_ip=<validator_node_ip_address> node_port=<node_port>
client_ip=<validator_client_ip_address> client_port=<client_port>
alias=<validator_alias> services=VALIDATOR blskey=<validator_bls_key>
blskey_pop=<validator_bls_key_pop>
```

IMPORTANT:
Many providers, such as AWS, use local, non-routable IP addresses on their nodes and then use NAT to translate these to public, routable IP addresses. If you are on such a system, use the **routable public addresses** for the ledger node command.

Example:

```
indy> ledger node target=4Tn3wZMNCvhSTXPcLinQDnHyj56DTLQtL61ki4jo2Loc
node_ip=18.136.178.42 client_ip=18.136.178.42 node_port=9701 client_port=9702
services=VALIDATOR alias=Node19
blskey=4kCWXzcEEzdh93rf3zhhDEeybLij7AwcE4NDewTf3LRdn8eoKBwufFcUyyvSJ4GfPpTQLu
X6iHjQwnCCQx4sSpfnptCWzvFEdJnhNSt4tJMQ2EzjcL9ewRWi24QxAaCnwbm2BBGJXF7JjqFgMzG
fuFXXHhGPX3UtdfAphrojk3A1sgq
```

```
blskey_pop=QqPuAnjnkYcE51H11Tub12i7Yri3ZLHmEYtJuaH1NFYKZBLi87SXgC3tMHxw3LMxEr
nbFwJCSdJKbTb2aCVmGzqXQtVWSpTVEQCsaSm4SUZLbzWVoHNQqDJASRYNbHH2CqpR2MtntA4YNb2
WixNSZNXFSdHMbB1yMQ7XUcZqtGHhcb
```

- *Suggestion: Save this command. You will use it again if you later move to another Sovrin Network.*

3.5.2. Enable the Service

In the Validator node:

Return to the Validator node machine.

Start the Validator service:

```
ubuntu@validator$ sudo systemctl start indy-node
```

Verify the start:

```
ubuntu@validator$ sudo systemctl status indy-node.service
```

Enable the service so that it will auto-restart when your node reboots:

```
ubuntu@validator$ sudo systemctl enable indy-node.service
```

3.6. See if the Node Is Working

If the setup is successful, your Validator node now connects to the Validator pool.

In the Validator node:

```
ubuntu@validator$ sudo validator-info
```

If your node is configured properly, you should see several nodes being selected as the primary or its backups, as in this example:

```
                England (1)
                Singapore (3)
                Virginia (4)
                RFCU (5)
                Canada (0)
                Korea (2)
```

Note: A ledger with a lot of transactions on it, like what often exists on the BuilderNet, can take a lot of time to sync up a new Validator node. If you don't get the right results for this test right away, try it again in a few minutes.

To check that messages and connections are occurring normally you can run the following commands to follow the log file:
In the Validator node:

```
ubuntu@validator$ sudo tail -f /var/log/indy/net3/<validator_alias>.log
```

Appendix A - Moving to Another Sovrin Network

Sovrin's BuilderNet is a key network maintained by the Sovrin Foundation. At some time, you may be asked to move your node to another of the Sovrin networks, such as the *MainNet or StagingNet*. As with the BuilderNet, a Sovrin Trustee must place your keys on any network that you will join with your Validator. **After you are informed that your keys are on the new ledger,** you may proceed with these steps. It is required to be on a call with someone from support@sovrin.org while performing these steps.
In the sections below, it is assumed that the network you are moving to is the MainNet, but the instructions can be adapted to moving to any other Sovrin network, or to moving back to BuilderNet if that becomes desirable in the future.

1.0 Remove your node from your current network

From your indy-cli turn off your validator services:

a. On your **CLI**, join your pool, open your wallet, and activate your Steward DID as explained earlier in this document. Then run the following command:

```
ledger node target=<validator_identifier> alias=<node_alias>
services=
```

1.1 Configure the Validator for another network, such as MainNet

On your Validator node, turn off the indy services:

```
ubuntu@validator$ sudo systemctl stop indy-node indy-node-control
```

Configure your Validator node to connect to the MainNet by using this command to change a line in the indy_config.py file:

```
ubuntu@validator$ sudo -i -u indy sed -i "s/'net3'/'live'/"
/etc/indy/indy_config.py
```

When you ran init_indy_node before, it auto-generated a seed that you were told to save securely. Run the node initialization script again, this time adding the seed that was auto-generated when you ran it before. The output should contain the same public keys that you saw before.

REMINDER: the following command requires you to use the local IP addresses if they are different from the public IP addresses (for example, they are different in AWS).

```
ubuntu@validator$ sudo -i -u indy init_indy_node <ALIAS> <node ip> <node port
#> <client ip> <client port#> <seed>
```

Finally, make sure that the MainNet network genesis files are in place on your Validator node. You should see "pool_transactions_genesis" and "domain_transactions_genesis" files in the /var/lib/indy/live/ directory. If they are not there, you will need to stop here and request them from support@sovrin.org before proceeding.

```
ubuntu@validator$ sudo su - indy
indy@validator$ ls /var/lib/indy/live/*genesis
indy@validator$ exit
```

Desired result:

```
/var/lib/indy/live/domain_transactions_genesis
/var/lib/indy/live/pool_transactions_genesis
```

NOTE: When creating a new network (not when switching to an existing network) you will need to download the domain and pool genesis files to the newly named directory. For example:

cd /var/lib/indy/net3

```
curl -O
https://raw.githubusercontent.com/sovrin-foundation/sovrin/master/sovrin/domain_transactions_
builder_genesis
curl -O
https://raw.githubusercontent.com/sovrin-foundation/sovrin/master/sovrin/pool_transactions_buil
der_genesis
```

1.2 Add node attributes to the MainNet ledger

In your CLI node

As you did for the BuilderNet, in your CLI create metadata for the MainNet, and create a wallet to use with it:

```
ubuntu@cli$ indy-cli
indy> pool create mainnet gen_txn_file=pool_transactions_live_genesis
indy> wallet create mainnet_wallet key=<wallet_key>
```

Now, establish your Steward credentials, and connect to the MainNet network:

```
indy> pool connect mainnet
indy> wallet open mainnet_wallet key=<wallet_key>
indy> did new seed=<your_steward_seed>
indy> did use <your_steward_DID>
```

As you did for the BuilderNet network, put your node attributes onto the ledger of the MainNet. Use the same transaction here that you used there. For example:

```
indy@live> ledger node target=4Tn3wZMNCvhSTXPcLinQDnHyj56DTLQtL61ki4jo2Loc
node_ip=18.136.178.42 client_ip=18.136.178.42 node_port=9701 client_port=9702
blskey=4kCWXzcEEzdh93rf3zhhDEeybLij7AwcE4NDewTf3LRdn8eoKBwufFcUyyvSJ4GfPpTQLu
X6iHjQwnCCQx4sSpfnptCWzvFEdJnhNSt4tJMQ2EzjcL9ewRWi24QxAaCnwbm2BBGJXF7JjqFgMzG
fuFXXHhGPX3UtdfAphrojk3A1sgq
blskey_pop=QqPuAnjnkYcE51H11Tub12i7Yri3ZLHmEYtJuaH1NFYKZBLi87SXgC3tMHxw3LMxEr
nbFwJCSdJKbTb2aCVmGzqXQtVWSpTVEQCsaSm4SUZLbzWVoHNQqDJASRYNbHH2CqpR2MtntA4YNb2
WixNSZNXFSdHMbB1yMQ7XUcZqtGHhcb services=VALIDATOR alias=Node19
```

1.3 Start the indy-node Service

In your Validator node:

```
ubuntu@validator$ sudo systemctl start indy-node
```

Verify the start:

```
ubuntu@validator$ systemctl status indy-node
```

1.4 Verify Operation of the Validator

Repeat the steps of section 3.6 *for the MainNet network* to determine that the Validator is operating properly:

```
ubuntu@validator$  sudo validator-info -v | grep Primary
```

Note: A ledger with a lot of transactions on it can take a lot of time to sync up on a new Validator node. If you don't get the right results for this test right away, try it again after several minutes.

Appendix B - Configuring Your Dual-Homed (2 NIC) Node

First some caveats and warnings. These are notes based on setting up 2 NICs on an AWS VM. It might be possible to adapt them for other environments as well, particularly the "Configure Network Interfaces in Ubuntu" section.

WARNING:
When you are doing network configuration, it is very possible to put your VM into a state where you are no longer able to log into it over the network. This may be difficult or impossible to recover from. Be very careful. If you have questions, doubts, or just need help, reach out **prior** to following these instructions.

Initial networking steps in an AWS console

Create security group "validator client"

- Port 22 for ssh
- Port 9702 for Validator client connections

Create security group "validator inter-node"

- Port 9701 for Validator inter-node connections
- Initially set up your Validator IP address to accept connections from anywhere, but later modify it as follows to only allow connections from specific IP addresses.
    - Whitelisted IP addresses
        - To generate whitelist, run on Validator: "current_validators.py --writeJson | node_address_list.py --outFormat aws"

Setup Validator instance

1. Provision VM
    a. Use security group "validator client" for the default network interface
    b. make note of the instance ID when completed
2. Add and configure a 2nd network interface in AWS.
    a. On EC2 left side menu - Network & Security -> Network Interfaces -> Create Network Interface
        i. Subnet -> Select a different subnet in the same zone as your instance
        ii. Private IP -> auto assign
        iii. Security groups -> validator inter-node
    b. On the main screen, select the new interface and click the Attach button

i. Find and select the instance ID (recorded in step 1)

3. Note the Network Interface ID of each network interface
    a. On EC2 left side menu - INSTANCES -> Instances
    b. Select your instance
    c. At the bottom of the screen select the description tab and scroll down to ‘Network interfaces’
    d. Click on each interface and then record the ‘Interface ID’ and the ‘Private IP Address’ for later use.
4. Create 2 Elastic IP’s, 1 for each NIC, and associate them with the network interfaces
    a. On EC2 left side menu - Network & Security ->Elastic IPs
        i. Click Allocate New Address
            1. Give your new addresses appropriate names so that you can identify them later. (i.e. BuilderNet Client and BuilderNet Inter-Node)
            2. I used Amazon IP addresses, but you can use your own if you like
            3. Repeat steps 1 and 2 to create a second Elastic IP
        ii. For each new Elastic IP do the following:
            1. Select one of the Elastic IP’s you just created
            2. Click Actions -> Associate address
                a. Resource type -> ‘Network interface’
                b. Network Interface -> <use one of the network interface IDs noted in previous step >
                c. Private IP -> (there should only be one option and it should match the internal IP address of the chosen interface)
                d. Leave checkbox empty (this might not matter)
                e. Click “Associate”
            3. Make sure you do this for both interfaces of your instance

Configure Network Interfaces in Ubuntu

1. Disable automatic network management by AWS. (These steps are for AWS users only and will keep AWS from overwriting your settings) Run the following from the Ubuntu command line:
    a. sudo su -
    b. echo 'network: {config: disabled}' > /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg

WARNING: The following steps use the common network interface names eth0 and eth1. You **must** substitute the interface device names used by your system or your instance will lose its network connection and you might not be able to reattach to it.

2. Run the following steps from the Ubuntu command line:
3. > ip a
    a. Record the interface device names and their local IP addresses for later use.
4. > route -n

   a. Record the Gateway for later use.
5. > cd /etc/network/interfaces.d
6. > vim 50-cloud-init.cfg
   a. Cut the existing eth0 lines from this file in preparation for moving them to a new file in this same directory.
   b. Example 50-cloud-init.cfg now looks like:

      auto lo
      iface lo inet loopback

7. > vim eth0.cfg (use <interface name>.cfg if your interface name is not eth0)
   a. Paste the eth0 lines cut from the 50-cloud-init.cfg file and add the following lines, indented 3 spaces:

      up ip route add default via <Gateway> dev <interface name> tab 1
      up ip rule add from <local IP addr of <interface name>>/32 tab 1
      up ip rule add to <local IP addr of <interface name>>/32 tab 1
      up ip route flush cache

   b. Example eth0.cfg

      auto eth0
      iface eth0 inet dhcp
        up ip route add default via 172.31.32.1 dev eth0 tab 1
        up ip rule add from 172.31.33.147/32 tab 1
        up ip rule add to 172.31.33.147/32 tab 1
        up ip route flush cache

8. Repeat step 7 but for the second network interface. The simplest way to do that is probably:
   a. > cp eth0.cfg eth1.cfg
   b. > vi eth1.cfg
      i. Replace all instances of eth0 with eth1
      ii. Change <local IP addr> to the one corresponding to eth1
      iii. Change ‘tab 1’ to ‘tab 2’
   c. Example eth1.cfg

      auto eth1
      iface eth1 inet dhcp
        up ip route add default via 172.31.32.1 dev eth1 tab 2
        up ip rule add from 172.31.35.63/32 tab 2
        up ip rule add to 172.31.35.63/32 tab 2
        up ip route flush cache

   d. > ifup eth1
      i. Check to make sure eth1 came up and is working properly. If the eth0 interface becomes unusable, you should then be able to log in through eth1 to fix it.
9. Reboot your machine

Tests

If the configuration is working, you should be able to connect a "listener" process to the IP address and port for the client connections. Then from a different, client machine you should be able to reach that port on that IP address, firewalls permitting. You should also be able to do the same thing for the node IP address and port. Netcat is ubiquitous and convenient for these tests.

On the Validator:
nc -l <client IP address> < client port>
On the client machine:
nc -v -z <client IP address> <client port>
Expected result:
Success!

On the Validator:
nc -l <node IP address> < node port>
On the client machine:
nc -v -z <node IP address> <node port>
Expected result:
Success!

Other combinations should fail or not return. Note that in AWS, the netcat commands executed on the Validator should use the private IP address, and the netcat commands executed on the client should use the public IP (Elastic) address.

Finally, remember to later modify firewalls to allow and deny traffic:
On client IP address, allow
Port 22 from your home network(s)
Port 9702 (or whatever you have configured for clients) from anywhere
On node IP address, allow
Port 9701 (or whatever you have configured for inter-validator) from whitelist of other Validators