Load Testing Using Libindy

Running Performance and Stability Tests

**Simple Version**

1. Setup
2. Installing Libindy
3. Executing the Scripts

# Files and Folders

I have been creating a "perf" directory on the machines running libindy, but you can create any directory you want.

Place all the python scripts in your directory (like perf).

perf/

- perf\_runner.py

- perf\_add\_requests.py

- perf\_get\_requests.py

- perf\_cleanup.py

- requests\_sender.py

- requests\_builder.py

- config.json (contains the pool\_genesis\_file path)

# Setup

Create a test pool.

Create one or more machines from which you will run libindy to simulate the client connections.

Install Libindy (see below)

Copy test scripts into a directory on the libindy machines.

Run scripts (see below)

# Installing Libindy

### First (Add repos)

sudo apt-key adv --keyserver [keyserver.ubuntu.com](http://keyserver.ubuntu.com/) --recv-keys BD33704C

sudo apt-key adv --keyserver [keyserver.ubuntu.com](http://keyserver.ubuntu.com/) --recv-keys 68DB5E88

sudo add-apt-repository "deb [https://repo.sovrin.org/](https://repo.evernym.com/libindy_crypto)deb xenial stable"

sudo add-apt-repository "deb<https://repo.sovrin.org/sdk/deb> xenial stable"

sudo add-apt-repository ppa:jonathonf/python-3.6

sudo apt update

### Second (Install packages)

##### Python 3.6

sudo apt install python3.6 -y

DEB Packages including Libindy

sudo DEBIAN\_FRONTEND=noninteractive apt-get install -y debsigs debsig-verify apt-transport-https python-pip python3-pip python3.5-dev python3.6 libsodium18 libsqlite0 libindy-crypto libindy

##### Libindy Python Wrappers

sudo pip3 install python3-indy

sudo pip3 install --upgrade python3-indy

**NOTE**

Make sure you have the indy files in the python3.6 directory (/usr/local/lib/python3.6/dist-packages/indy). If you do not you may need to copy or symlink the /indy directory in python3.5 to python3.6

/usr/local/lib/python3.5/dist-packages/indy to /usr/local/lib/python3.6/dist-packages/

# Understanding the scripts

The script "Perf\_runner.py" is the script used to run all the automation. Perf\_runner.py calls either Perf\_add\_requests.py or Perf\_get\_requests.py to execute either adding or getting nyms. The Perf\_add\_requests.py and Perf\_get\_requests.py call requests\_builder.py to build all requests in to files and requests\_sender.py to read all requests from that files, then sending them to the ledger.

At the end of the test run Perf\_runner.py calls Perf\_cleanup.py to wrap up the test.

### Perf\_runner.py

This script calls the other scripts.

About line 29 you should see Options class that contains all parameters and the meaning.

Three important parameters are "clients", “thread\_num” and "txns".

**Variable "clients"** is the number of simulated clients you want to run. Each client will run all the txns transactions.

**Variable "thread\_num"** is the number of threads that will run on your machine.

**Variable "txns"** is the number of ADD (nym, schema, claim, etc.) transactions you want to have 1 client execute.

**Example**

The following setting will create 2,000 ADD requests from one libindy machine if running. Each machine will run with 4 threads.

clients=40

thread\_num=4

txns=50

### Perf\_add\_requests.py

There are several parameters possible with this script. The main parameters are:

“-n”: the number of transactions to run.

“-s”: the number of threads.

“-d”: the directory you want to store requests info when sending add request.

The script will create the directory in the location it is running from. The directory will store text files with all the DIDs, wallet handle, pool handle create in that test run. The text files are used by perf\_get\_requests.py to run GET request lookups.

### perf\_get\_requests.py

There are several parameters possible with this script. The main parameters are “-s” and “-d” for the number of threads to run and the location where the files containing the DIDs to lookup are located.

You will need to run perf\_add\_requests.py to generate the DIDs to lookup before running perf\_get\_requests.py.

### perf\_cleanup.py

This script just removes the .indy/pool and .indy/wallet directories. There is an issue with libindy where it will not run if there is already a pool with the same name created. To work with this issue we clean up the pool and wallets so each run is clean.

### requests\_builder.py

For each of clients, this script will build the requests based on the number of transactions. Next, the requests\_builder writes all requests into the files (the number of files based on the number of threads).

Example:

python3.6 perf\_runner.py -a -k nym -d [path\_to\_save\_request\_info] -c 2 -s 2 -n 500

This cmd will create two clients, each of them will create two threads and send 50 nym requests to the ledger. All nym info will be saved in [path\_to\_save\_request\_info]\nym

The total requests is 500 x 2 = 1000

### requests\_sender.py

For each of clients, this script will create the threads and send all get nym requests that was stored in [path\_to\_get\_request\_info] to the ledger.

Example:

python3.6 perf\_runner.py -g -k nym -d [path\_to\_get\_request\_info] -c 2 -s 2

This command will create two clients, each of them create two threads and send all get nym requests that was stored in [path\_to\_get\_request\_info] to the ledger.