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 ADD or GET request (nym, schema, claim or attribute). The perf\_add\_requests.py and perf\_get\_requests.py call requests\_builder.py to build all requests into files and requests\_sender.py to read all requests from that files, then submit them to the ledger.

### config.json

This file contains some necessary configuration to run the script. Now, this just contain path to you pool genesis file location.

Template:

{

“pool\_genesis\_file”: [path\_to\_you\_genesis\_file]

}

### perf\_runner.py

There is several parameters possible with this script, the main parameters are:

“-a”: to show that you want to submit ADD request (runner will call perf\_add\_requests).

“-g”: to show that you want to submit GET request (runner will call perf\_get\_requests).

“-k”: kind of request (schema, nym, claim, attribute).

“-d”: the directory you want to store request information in case you want to submit ADD request or the place you want to collect request information in case you want to submit GET request.

“-c”: the number of clients you want to simulate.

“-s”: number of thread of each client.

“-n”: number of ADD request you want to submit (in case you want to submit GET request, this parameter will be ignored).

**Example**

If you want create two clients, each client will submit 2000 ADD SCHEMA requests and there is two threads per client and you want to store request information in /home/your\_account/info, the command should be like:

python3.6 perf\_runner.py -a -k schema -d /home/your\_account/info -c 2 -s 2 -n 2000

### 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 information when sending ADD request.

“-k”: kind of request (schema, nym, claim, attribute).

The script will create the directory in the location it is running from. The directory will store text files with all the requests information (did, attributes, …) 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 requests information to lookup are located.

You will need to run perf\_add\_requests.py to generate the requests information 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 will be created by client).

### requests\_sender.py

This script will receive the list files of requests from “requests\_builder.py”, create threads base on number of file and submit those requests to ledger. Then, delete the requests file that completely submitted to ledger.

### utils.py

This file contains all utility function that need for all scripts above.