IOTA & Python — A Beginning

John Grant
Jul 28, 2017 · 3 min read

Intro

> *IOTA is a revolutionary new transactional settlement and data integrity layer for the Internet of Things. It's based on a new distributed ledger architecture, the Tangle, which overcomes the inefficiencies of current Blockchain designs and introduces a new way of reaching consensus in a decentralized peer-to-peer system. For the first time ever, through IOTA people can transfer money without any fees. This means that even infinitesimally small nanopayments can be made through IOTA. — From the docs*

Sounds pretty cool! And I think it has potential to overcome some existing blockchain issues. I wanted to try out some development using Python and IOTA Python. It took a bit of work getting started but once you know the steps it's pretty easy so I thought I'd share what I did.

Setup Dev Environment

It's always good practise to develop using a virtual environment so I started in a clean dir and ran:

```
virtualenv env
```

And then activated the environment:

```
source env/bin/activate
```

Install PyOTA

PyOTA is the official Python library for the IOTA Core and implements the official IOTA API.

In your activated dev environment install the pyota library using:

```
pip install pyota
```

Full Nodes, Light Clients, Adapters & Routing

An IOTA full node connects to the tangle to share data with other nodes, it also takes care of Proof Of Work (PoW). It's possible to interface with a running node through port 14265 using API calls.

Some resource restricted devices will not be able to act as a full node. These can run a light client and connect to a remote IRI instance which takes care of some of the more intensive overhead. Using a light client also makes getting started easier as you don't have to worry about connecting to neighbours and static IP addressing.

PyOTA doesn't yet have PoW integrated, so requests like attachToTangle need to be offloaded to a full IOTA node. Normally you would want a full node to sync with the network which means you need to pair with neighbours but because I don't have a static IP address this is tricky.

To overcome this it's possible to use PyOTAs Adapter Routing Wrapper. Adapters are responsible for sending requests to a node and returning the response. The RoutingWrapper allows you to route API requests to different nodes depending on the command name.

So now PoW commands can be routed to a locally running full node which doesn't have to be synced because everything else can be routed to a remote IRI. Very handy!

Install Local Full Node

Next I installed the IRI — the IOTA Reference Implementation. It's a complete IOTA node so can take care of PoW. As mentioned in the last section I'm not worried about syncing this node so don't need to worry about finding neighbours.

The source code for the IRI can be found here. I installed it by downloading an official release of a jar file, in my case iri-1.2.4.jar, from this location.

Once downloaded it can be started using:

```
java -jar iri-1.2.3.jar -p 14265
```

Send Example

Now we can bring it all together by using the extended API call, send_transfer, to make a transaction.

```
# coding=utf-8
"""
Simple example using the RoutingWrapper to route API requests to
different nodes.
See:
https://github.com/iotaledger/documentation/blob/iota.lib.py/1.2.
x/source/includes/_adapters.md#routingwrapper
"""
from iota import *
from iota.adapter.wrappers import RoutingWrapper

api =\
 Iota(
 # Send PoW requests to local node.
 # All other requests go to light wallet node.
 RoutingWrapper('http://service.iotasupport.com:14265')
 .add_route('attachToTangle', 'http://localhost:14265'),

# Seed used for cryptographic functions.
 seed = b'SEED9GOES9HERE'
 )

# Example of sending a transfer using the adapter.
bundle = api.send_transfer(
 depth = 100,
 transfers = [
 ProposedTransaction(
 # Recipient of the transfer.
 address =
 Address(
 #b'TESTVALUE9DONTUSEINPRODUCTION99999FBFFTG'
 #b'QFWEHEL9KCAFXBJBXGE9HID9XCOHFIDABHDG9AHDR'
 ),
```

```
# Amount of IOTA to transfer.
 # This value may be zero.
 value = 1,

# Optional tag to attach to the transfer.
 tag = Tag(b'ADAPT'),

# Optional message to include with the transfer.
 message = TryteString.from_string('Hello!'),
 ),
 ],
)
```

Python Cryptocurrency Technology Software Development