ETHEREUM ALARM CLOCK

MAIN SITE BLOG DOCS

Introduction to the Python-Ethereum ecosystem

FEBRUARY 22, 2016

Introduction to the Python-Ethereum ecosystem

This post is targeted at developers who are interested in getting started developing on Ethereum using python.

Is python the right choice?

It's important to know what you are planning to build because Python may not be the best choice for certain projects.

If you are planning on building a user facing application that will run in a browser then Python may not be the right choice for you. DApps that run in the browser are likely to benifit from a javascript toolchain so you may be better off looking into Embark or Truffle.

One of the powerful features of a DApp that is written as pure HTML/JS/CSS is that it can be completely serverless. Choosing python as part of your web toolchain may anchor your application in the web2 world.

Outside of the browser however, Python and Ethereum work very well together.

Base Layer Tooling

The `pyethereum` library by Vitalik Buterin has been the base for most of the tooling that I've written in the Python ecosystem. If what you are looking to write deals with low level EVM interactions then this library is a great place to start.

Interacting with the blockchain

When you want to actually interact with the blockchain from python you'll probably want to use JSON-RPC. There are a few python client implementations to choose from.

- ethereum-rpc-client
- ethereum-ipc-client

These two libraries provide a client for interacting with the JSON-RPC service over either HTTP or an IPC Socket respectively. They can both act as drop-in replacements for each other as they expose the same API over a different transport layer.

Interacting with Contracts

To interact with contracts on the blockchain, you'll need to encode and decode the inputs and outputs according to the Ethereum Contract ABI. There are low level tools available for doing this using either ethereum-abi-utils. This library provides the abi encoding and decoding functionality available from within the `pyethereum` library as a standalone library with fewer dependencies.

This method of interacting with contracts is a bit clumsy and verbose, so you may want to take a look at the `ethereum-contract` library. It comes with a python class that can be used to represent an ethereum contract that has

callable methods for each of the contract methods which are exposed via the contract ABI.

Testing

Lots of people have used the `ethereum.tester` module that is included within `pyethereum` to write tests. This module exposes a python based EVM which works great for testing EVM interactions within your python code.

For a slightly higher level tool, you can use the `ethereum-tester-client`] (https://pypi.python.org/pypi/ethereum-tester-client). This library exposes a drop-in replacement for either the RPC or IPC based clients which interacts directly with the `ethereum.tester` EVM. This client can also be used with the `ethereum-contract` library to test your contract code.

Populus ties it all together

All of these tools serve as a foundation for `populus`. Populus is a python based framework focused on contract development and testing. Populus's command line interface provides tools for compiling, testing, and deploying your contracts.

Pull Requests welcome

All of these tools are open source and available for use today. Please feel free to reach out to me directly, ideally either in a Gitter channel, or via github issue if you have any problems with any of these tools. And as with any of my projects, pull requests are welcome. Please ensure you inclued the obligatory cute animal picture with any pull requests.

‹ Newer Older ›

Ethereum Alarm Clock