Dick Olsson

Random thoughts on blockchains, cycling in London and other unrelated things

Menu

Building dapps on Ethereum – part 5: Ethereum Name Service and Swarm

Free software July 20, 2017 5 Minutes

A common usability problem with cryptographic systems like blockchains is that accounts, smart contracts and content on the blockchain are addressed with hashes like `0x7eF963588706a8d39D481634eB46f5c54A04c584`. These addresses are easy for machines to securely verify but hard for humans to type and remember. And anything

that's difficult for humans creates vulnerability vectors, like phishing attacks by using an address that looks similar but is owned by a malicious person.

The Ethereum Name Service (ENS) attempts to reduce these usability and security problems by allowing one to use an address like `dickolsson.eth` instead of `0x7eF963588706a8d39D481634eB46f5c54A04c584`. It's very similar to the Domain Name Service (DNS) that let you use `dickolsson.com` instead of `46.232.178.79`.

In this blog post we'll learn how to deploy ENS locally and use ENS names to address content on Swarm which I covered in part 4 of this series. Note, we will not use ENS to address wallet accounts or smart contracts (at least not right now). Throughout this blog post series we will use sample code from the Iron Doers project which is a quite simple concept briefly described in a practical example of using blockchains and the project's whitepaper.

How does Ethereum Name Service work?

ENS is beautifully simple. There are three main concepts that all are represented as smart contracts on the blockchain. They are (1) the ENS root (2) many registrars (3) many resolvers.

It should be mentioned that ENS is still in its infancy (as with most blockchain things). There's gaps to fill and improvements to be made, but the system is already peer-reviewed by the Ethereum community and in use on its public network.

1. The ENS root

The ENS root is a very simple smart contract that mainly keep track of two things (1) registrars responsible for specific top-level domains (TLD) and (2) what resolver to use for a particular domain.

The ENS root is currently maintained as democratic autonomous organisation (DAO) where at least 4 of the 7 trustees need to agree on changes to the ENS root (such as

adding additional registrars). Read more about the ENS' plans to further democratise and decentralise the decision making on their website.

2. Registrars

ENS registrars are very similar to DNS registrars, i.e. they are the entity (a smart contract for Ethereum) responsible for managing specific top-level domains (TLD), like the `.eth` TLD on the public Ethereum network.

3. Resolvers

ENS resolvers are smart contracts responsible for the actual translation of hashes to human-readable names.

One ENS name can resolve to a few different things at once. The two most important things for dapps are (1) addresses for wallets or smart contracts (2) content hashes on the Swarm platform. The latter is what we'll focus on in this blog post.

Installing ENS locally

In order to properly test and develop a dapp using ENS names you have to deploy all required ENS contracts locally and configure things like Swarm and your Ethereum browser to use the ENS root for your local blockchain. The ENS system was defined in the Ethereum Improvement Proposal #137 and we will use existing implementations of this standard to help with our deployment.

The Ethereum Foundation is hosting a Github repository with the ENS reference implementations that we'll use:

1. ENS root abstract interface
2. ENS root contract
3. First-in-first-served (FIFS) registrar contract
4. Public resolver contract

The ENS root and resolver contracts above are very similar to the implementation that's deployed on the public Ethereum network (i.e. the `.LLL` contracts). The registrar contract for `.eth` on the public network is implemented like a "Vickrey auction", as opposed to FIFS (which'd be a bit chaotic on the public net). But for local development FIFS is good.

Start by copying all four `.sol` contracts that was linked above into your `contracts/` directory. Next, we need a deployment script that will (1) deploy the ENS root contract (2) deploy the FIFS registry contract for the `.test` TLD (3) deploy the public resolver contract (4) register the `irondoers.test` domain (5) tell the ENS root to use the public resolver for the `irondoers.test` domain.

```
1  const ENS = artifacts.require("./ENS.sol");
2  const FIFSRegistrar = artifacts.require('./FIFSRegistrar.sol');
3  const PublicResolver = artifacts.require('./PublicResolver.sol');
4  const namehash = require('../node_modules/eth-ens-namehash');
5
6  const Web3 = require('web3');
7  const web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
8
9  module.exports = function(deployer, network, accounts) {
10   var domain = 'irondoers';
11   var tld = 'test';
12
13   var rootNode = {
14     namehash: namehash(tld),
15     sha3: web3.sha3(tld)
16   };
17
18   deployer.deploy(ENS)
19     .then(function() {
20       return deployer.deploy(FIFSRegistrar, ENS.address, rootNode.namehash);
21     })
22     .then(function() {
23       return ENS.at(ENS.address).setSubnodeOwner('0x0', rootNode.sha3, FIFSRegistrar.
```

```
    })
    .then(function() {
      return deployer.deploy(PublicResolver, ENS.address);
    })
    .then(function() {
      return FIFSRegistrar.at(FIFSRegistrar.address).register(web3.sha3(domain), web3
    })
    .then(function() {
      return ENS.at(ENS.address).setResolver(namehash(domain + '.' + tld), PublicReso
    });
};
```

3_local_ens.js hosted with ❤️ by GitHub view raw

Next steps are to start your local blockchain, set up accounts, start mining, compiling and migrating all contracts to the blockchain. For more detailed instructions and commands for doing these steps see see part 4 of this series.

After migrating your contracts you essentially have a full ENS system running locally, nice! Take note of the various contract addresses because you'll need them to interact with ENS later on!

```
$ truffle migrate --reset

Using network 'development'.
Running migration: 1_initial_migration.js
  Replacing Migrations...
  Migrations: 0x80a6e733991d7068e9683106f6a2c935d51462f5
Saving successful migration to network...
Running migration: 2_deploy_contracts.js
  Replacing IronDoers...
```

```
  IronDoers: 0x4e87c83e70ad3917d1e3b7b39915e90e837f9aa4
Saving successful migration to network...
Running migration: 3_local_ens.js
  Replacing ENS...
  ENS: 0x3b5cc4bb784f805d1e7670695231f25f941f129a
  Replacing FIFSRegistrar...
  FIFSRegistrar: 0x5e6edd79c7917fc38fcd5b89ee15a5fdb5e93ef8
  Replacing PublicResolver...
  PublicResolver: 0xe9a51737bbef33f5e14e26ff433860267b225de4
Saving successful migration to network...
```

Using ENS with Swarm

With ENS deployed on our local blockchain it's time to start addressing some Swarm content! But first we need some content on Swarm. The only thing we need to add is a pointer to our ENS address. Replace with your own addresses accordingly and don't forget to start your local `geth` node and mining before starting Swarm!

```
$ swarm --bzzaccount MAINACCOUNT \
  --ens-addr 0x3b5cc4bb784f805d1e7670695231f25f941f129a \
  --datadir ./chain/ \
  --ens-api ./chain/geth.ipc \
  --verbosity 4 \
  --maxpeers 0
```

Then upload some content to Swarm and take note of the content hash of the upload:

```
$ swarm --recursive --defaultpath=www/index.html up www/
c777819a8eaa98d61615d94be49cd3bfaa95a2ccbe0225a3c3cf354777f33
c0f
```

Addressing content

In order to address content we need to interact with the ENS contracts. At the moment there are little-to-none convenient tooling for doing this, so we will interact with our contracts directly from the `geth` console! Start by attaching to the console in a new terminal tab.

```
$ geth attach ipc:./chain/geth.ipc
```

Next, we will use a utility script that you can find in the example repository. Put this script in your repository's `scripts/` directory as well. This utility script defines the Application Binary Interface (ABI) for each contract along with the `namehash` function that we need to translate human-readable names into the format that ENS understands.

In the console we'll first define some handy variables, namely our main account and an instance of the `PublicResolver` contract. Replace with your own addresses accordingly.

```
> loadScript('./scripts/ensutils.js');
> var accounts = personal.listAccounts;
> var resolver = PublicResolver.at('0xe9a51737bbef33f5e14e26f
```

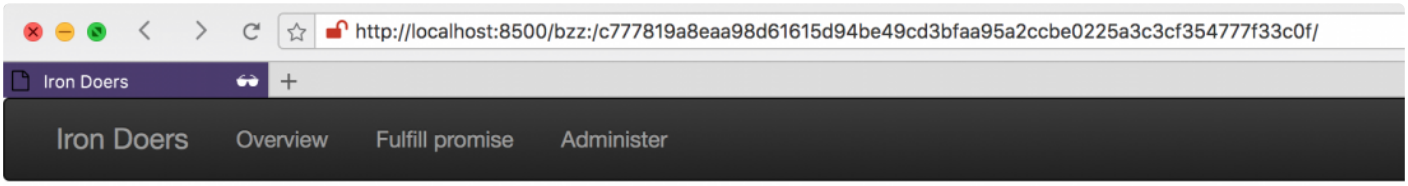
```
f433860267b225de4');
```

Once we have these handy variables defined we can start interacting with ENS! Let's start by setting the Swarm content hash for the `irondoers.test` domain which we registered in our deployment script earlier. *Important* — note the `0x` prefix on the Swarm hash that we got from the upload output earlier.

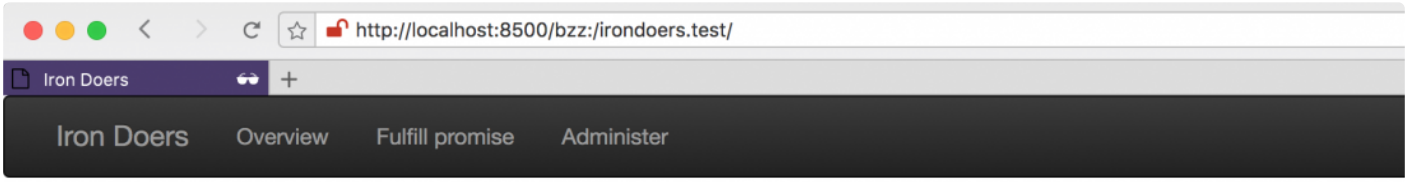
```
> resolver.setContent(namehash('irondoers.test'), '0xc777819a
8eaa98d61615d94be49cd3bfaa95a2ccbe0225a3c3cf354777f33c0f', {f
rom: accounts[0]});
```

Now you just need to wait for this Ethereum transaction to be mined (which can take a few minutes) and then you should be able to access your Swarm content with the domain name instead of a hash!

When accessing our dapp through the HTTP gateway, this is what it looked like before we had a ENS name registered:

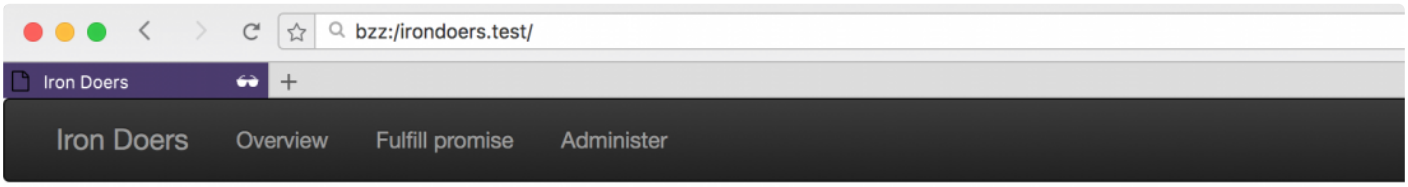


This is what it looks like with the HTTP gateway, but after setting up our ENS name:



This is what it'll look like when browsers understand the Swarm protocol directly

without needing a HTTP gateway:



Share this:

Twitter Facebook LinkedIn 1 Reddit Email Telegram WhatsApp

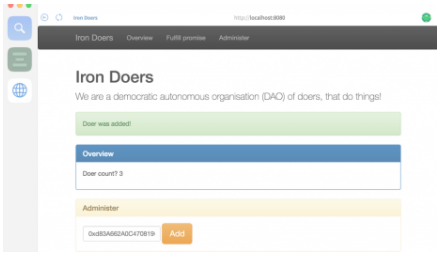
Like this:

Loading...

Related

[Building dapps on Ethereum – part 4: decentralised hosting using Swarm](#)
July 14, 2017
In "Free software"

[Building dapps on Ethereum - part 2: smart contracts](#)
June 21, 2017
In "Free software"



[Building dapps on Ethereum - part 3: user interface](#)
July 9, 2017
In "Free software"

Tagged: blockchain, ens, ethereum, how to build dapps

‹ Building dapps on Ethereum – part 4: decentralised hosting using Swarm

EVRF – Ethereum Vulnerability Reporting Framework – a proposal for less chaos ›

9 thoughts on “Building dapps on Ethereum – part 5: Ethereum Name Service and Swarm”

Kumar

July 20, 2017 at 3:03 pm

HI, I just added resolver to my ens domain, but its not resolving my wallet address

https://etherscan.io/enslookup?q=internext.eth

but below link resolves wallet address correctly..(someone else)

https://etherscan.io/enslookup?q=firefly.eth

what I’m missing..? pls help

arjunrajjain123

July 20, 2017 at 6:23 pm

Great blog post! I was wondering if you could perhaps also show an example of using Whisper.

Dick Olsson

July 21, 2017 at 6:18 am

@arjunrajjain123 Thank you! I plan to do some blog posts on how to write chatbots for Status.im. They use Whisper as the chat protocol. I think that'll be some really good posts about Status 🙂

Dick Olsson

July 21, 2017 at 6:35 am

@Kumar How did you add the resolver for your domain? It should be something like:

```
ENS
  .at(ENS.address)
  .setResolver(namehash('yourdomain.eth'), PublicRes
olver.address);
```

What function did you use to resolve the account? It should be something like this:

```
PublicResolver
  .at(PublicResolver.address)
  .setAddr(namehash('yourdomain.eth'), '0xyourwallet
', {from: '0xyourwallet');
```

> Note that you must use the `setAddr()` function instead of `setContent()` when you want to resolve a wallet address instead of Swarm content.

Daniel Ellison

July 21, 2017 at 2:42 pm

Excellent article, Dick! The community needs more clear guides like this. One very minor correction, though I think how you've explained it works for people trying to get familiar with ENS: The ENS root contract that was deployed is actually ENS.lll, not ENS.sol. This was done because of the smaller binary size created by the LLL compiler, resulting in more efficient–and less costly–function calls. Both implementations are equivalent, however, so people who use your guide aren't being led astray.

I'm looking forward to more excellent articles!

Dick Olsson

July 21, 2017 at 3:46 pm

@Daniel Thanks a lot! I edited one of the paragraphs and tried to clarify this. Thanks a lot for the pointer, I really appreciate it 🙂

Daniel Ellison

July 21, 2017 at 4:45 pm

No problem. As I said, it's a minor thing that doesn't really affect the people reading this article. Just a clarification: ENS.lll is the only contract of the three that

> is written in LLL. The other two are Solidity contracts.

Daniel Ellison

July 21, 2017 at 7:06 pm

A clarification to the clarification! 🙂 ENS.lll is the only contract of the three that is written in LLL and has been deployed to mainnet. There's an LLL version of the public resolver but it was never deployed. I forget the reason, to be honest.

> Pingback: Summary of the week from 17.07.2017 – Iron Blogger

Leave a Reply

Enter your comment here...