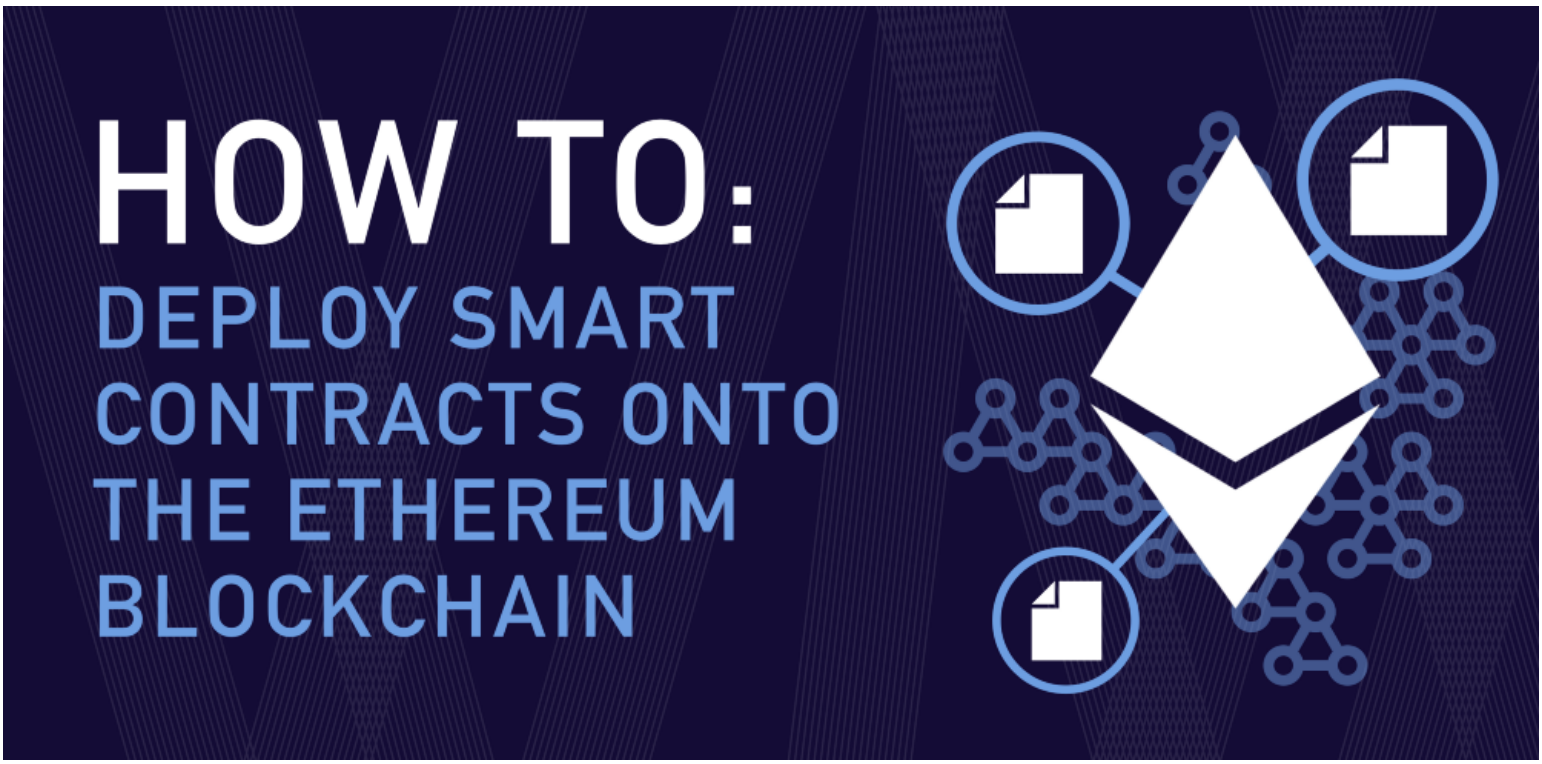
Mercury Protocol Follow
Dec 21, 2017 · 5 min read

How To: Deploy Smart Contracts Onto The Ethereum Blockchain



Dev highlights of this week

- Implemented analytics of rewards in preparation for switching to mainnet
- Integrated the transaction status on iOS and Android client (pending release)
- Completed POC to reduce latency for message delivery

- Released Android update (version 3.2.2.440) with minor improvements and bug fixes
- Investigating cross-platform development using React Native
- Continuing research on ways to reduce or eliminate Ethereum gas costs including Brave, Raiden, 0x, and IOTA
- Interviewing candidates that will help us scale through 2018

. . .

How To: Deploy Smart Contracts Onto The Ethereum Blockchain

The power of Ethereum comes from its ability to host and execute smart contracts. Smart contracts are pieces of code that can be deployed onto the Ethereum blockchain and live there in perpetuity (unless coded to self-destruct), without a governing body maintaining control over them. Smart contracts are **trustless** by nature. No party is required to **trust** another party . to complete a transaction. Instead, every party interacts with the same smart contract that contains an agreed upon set of rules in the form of code, that will run without the possibility of any third party manipulation.

Deploying smart contracts can be a bit tricky if you've never done it before. This guide aims to ease that learning curve and walk you through the low-level deployment approach, as well as provide some higher level alternatives. Smart contracts are written in the Solidity programming language. This guide is written for Mac.

Prerequisites

You need to have the Solidity compiler (solc) installed. The easiest way to do this is with npm.

1. Download Node.js and npm here

2. Now install solc with

```
npm install -g solc
```

You also need to have Geth (go-ethereum) installed. The easiest way is with Homebrew.

1. Install homebrew

2. Now install geth

```
brew tap ethereum/ethereum
brew install ethereum
```

Write Your Smart Contract

Here is our smart contract example. It is a simple contract with just a constructor and two functions. It does nothing useful, but hopefully it'll make you smile! Copy this code to a plain text file in its own folder and name it `getSchwifty.sol`. Latest `solc` compiler was using solidity version 0.4.19 at the time of this writing.

```
pragma solidity 0.4.19;

contract getSchwifty {
  string internal constant seeWhatYouGot = "I want to see w
  string internal contestResult;

  // Contract constructor that takes a string param
  function getSchwifty(string _contestResult) public {
    contestResult = _contestResult;
  }

  function showMeWhatYouGot() external pure returns (string
    return seeWhatYouGot;
  }

  function enterContest(string _contestEntry) external view
    // completely disregard the _contestEntry ;)
    return contestResult;
  }
}
```

getSchwifty.sol hosted with ❤️ by GitHub view raw

Compile Your Contract

Now we need to compile our solidity code into an interface (abi) and bytecode (bin) that can be deployed onto the blockchain. Open a terminal window and navigate to the directory that contains `getSchwifty.sol`.

1. Compile abi and bin

```
solcjs --abi getSchwifty.sol
solcjs --bin getSchwifty.sol
```

Don't worry about the compiler warning. We already know that `_contestEntry` is unused.

2. Now display the contents of the compiled files

```
more getSchwifty_sol_getSchwifty.abi
more getSchwifty_sol_getSchwifty.bin
```

We will use these outputs in the next section, so keep this terminal window handy.

Start Your Geth Node

Fire up your geth node by opening a new terminal window and running the below command. If you have a folder you normally start `geth` from navigate there, otherwise your getSchwifty project folder is perfectly fine.

```
geth --rinkeby console 2>> ./rinkebyEth.log
```

This will start a node pointing at the **Rinkeby** test network. To run a node pointing to the main Ethereum network, simply omit the `--rinkeby` flag. This will also save network logs to the file `rinkebyEth.log` which you can view with `tail -f rinkebyEth.log`. A new file with this name will be created if none exists. Unless otherwise specified, this command will use the default Ethereum data directory, which on Mac is `/Users/<your_user>/Library/Ethereum/rinkeby`

Output should look like this:

```
Welcome to the Geth JavaScript console!
```

```
instance: Geth/v1.7.3-stable-4bb3c89d/darwin-amd64/go1.8.3
coinbase: 0xae13d41d66af28380c7af6d825ab557eb271ffff
at block: 1449394 (Wed, 20 Dec 2017 14:52:09 PST)
datadir: /Users/<your_user>/Library/Ethereum/rinkeby
modules: admin:1.0 clique:1.0 debug:1.0 eth:1.0 miner:1.0
net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0
```

```
>
```

Your `coinbase` must have Eth. If you want to deploy on Rinkeby Testnet (recommended for testing), make sure you have an account with Rinkeby Eth. You can fund a Rinkeby account with the Rinkeby Faucet. If you would prefer to create your own test network and deploy contracts there, follow the steps in our post, How To: Create Your Own Private Ethereum Blockchain.

If this is your first time starting Geth or you haven't started it in a while, it may take a while for the blockchain to sync. You can check syncing progress with `eth.syncing`. If syncing is complete, this will return `false`, otherwise you must wait until `currentBlock` matches `highestBlock`.

Deploy Your Compiled Contract From Geth

1. Unlock your account

```
> personal.unlockAccount(eth.coinbase)
Unlock account 0xabc123....
Passphrase: <ENTER_PASSPHRASE>
```

2. Set up your bytecode and abi variables

```
> var getSchwifty = eth.contract(<CONTENTS_OF_ABI_FILE>)
> var bytecode = '0x<CONTENTS_OF_BIN_FILE>'
```

Replace `<CONTENTS_OF_ABI_FILE>` and `<CONTENTS_OF_BIN_FILE>` with the outputs of the `more` commands that should still be open in another terminal window.

3. Deploy your contract!

```
> var deploy = {from:eth.coinbase, data:bytecode, gas:
2000000}
> var getSchwiftyPartialInstance =
getSchwifty.new("DISQUALIFIED!", deploy)
```

Pass the `new` function all parameters your contract constructor expects in a comma separated list. The final parameter should always be the `deploy` variable you declared in the previous line.

4. Interact with your contract

Wait about a minute for your contract to be mined on the network, then type `getSchwiftyPartialInstance`. If the `address` field is filled in, your contract is live and ready to go! Lets store a reference to the deployed contract.

```
> var getSchwiftyInstance =
getSchwifty.at(getSchwiftyPartialInstance.address)
```

You can now type `getSchwiftyInstance.<FUNCTION_IN_CONTRACT>` to interact with it. You can also just type `getSchwiftyInstance` to display information about the deployed contract.

```
> getSchwiftyInstance.showMeWhatYouGot()
> getSchwiftyInstance.enterContest("bleep bleep bloop!")
```

Note: If you are deploying to your own private Ethereum blockchain, someone on the network needs to be mining for the contract to be deployed.

Wrapping Up

Congrats! You've successfully deployed your first smart contract onto the Ethereum network, a fundamental step in the world of Ethereum blockchain development.

This is a relatively hands on way of deploying contracts, but there are tools that speed this process up quite a bit. I personally use Truffle to deploy most of my contracts (they refer to the deployment process as "migrations"), and once they're deployed I add them to the Ethereum wallet/browser Mist to interact with them. You could even deploy contracts directly from Mist, or write a Python script to do the heavy lifting for you. As you can see there are plenty of ways to deploy and interact with smart contracts, it just takes some experimentation to figure out which method works best for your workflow.

Hope you enjoyed the writeup, drop a comment below if you have any questions. Happy Holidays!

Connect

Slack
Telegram
Twitter
Reddit
Facebook
LinkedIn
GitHub

Learn more about the Mercury Protocol
Read the Mercury Protocol whitepaper
Follow +mercuryprotocol on Dust