Truffle Suite

Archived: This tutorial has been archived and may not work as expected; versions are out of date, methods and workflows may have changed. We leave these up for historical context and for any universally useful information contained. Use at your own risk!

Ethereum is perhaps best defined by its public network, a network where every transaction -- and all participants of each transaction -- are publicly available to anyone looking at its transaction history.

Truffle got its start building for the public blockchain: at the time of Ethereum's launch, the public blockchain was the only smart contract blockchain around. However since launch, technology has changed, most notably withQuorum , the permissioned blockchain initially developed by JP Morgan Chase.

Quorum is a version of Ethereum that adds new features on top of what Ethereum already provides. Specifically,Quorum adds the ability to create private blockchains between select participants, and more importantly adds transaction privacy on top of normal Ethereum transactions .

Transaction privacy presents a number of useful use cases, especially in the enterprise and banking worlds. For instance, large banks might want to take advantage of blockchain technologies like Ethereum but don't want their transactions to be publicly available to everyone as a matter of doing business. Quorum presents a useful alternative.

Let's use a quick example: Say Bob, Tom, and Alice all create a blockchain together, and Alice wants to send 20 TruffleCoin to Bob. But here's the kicker: she doesn't want Tom (or anyone else other than Bob) to know, because she cares about her privacy. Using Quorum, Alice could easily send a transaction where the transaction data is only available to her and Bob.

This tutorial represents Truffle's official support for Quorum. By the end of this tutorial you'll learn how to use both Truffle and Quorum together to build privacy-enabled dapps.

Requirements¶

This tutorial expects you to have some knowledge of Truffle, Ethereum, Quorum, and Solidity. For more information on these topics, please see the following links:

- Truffle documentation
- Ethereum overview
- Quorum overview
- anddocumentation
- Solidity documentation

You will primarily be using the command line for this tutorial. Please ensure you have basic familiarity with opening and using the command line provided by your operating system. Additionally, you will need the following software installed before proceeding:

- VirtualBox
- Vagrant
- Git

Getting Started¶

In this tutorial, we'll show you how to develop dapps for Quorum using Truffle and Quorum's7nodes example . The steps are as follows:

- Setting up your Quorum client
- Connecting Truffle to Quorum
- Deploying smart contracts on Quorum
- Using Quorum's privacy features to make transactions private
- Interacting with contracts privately

You'll see that developing for Quorum using Truffle isexactly like developing for the public Ethereum blockchain. Truffle supports Quorum out of the box, and the same strategies and methods for building Ethereum-enabled applications for the Ethereum public blockchain also apply to building dapps on Quorum.

Setting up your Quorum client¶

The Quorum client is a replacement for the Ethereum client. Using the Quorum client, you can set up a private blockchain that's only available to you and the people you allow to participate.

We're going to use a Quorum cluster of seven nodes (so seven Quorum clients) already set up and configured for us inside

a virtual machine. Youcould choose to install Quorum yourself bydownloading it directly and building it from source, but for this example, using the pre-configured cluster is much easier.

1. To set up the cluster, open a terminal and navigate to a directory where you'd like it installed. Here, we choseworkspace
2. :

mkdir workspacecd

workspace 1. Download the Quorum examples repository:

git clone https://github.com/jpmorganchase/quorum-examples 1. We'll use Vagrant to initialize the cluster's virtual machine. Note that this step could take a few minutes as it makes the virtual machine ready for use.

cd

quorum-examples vagrant up 1. Aftervagrant up 2. successfully completes, we'll want a way to access our newly minted virtual machine. Note that a virtual machine is like another computer running inside of your own, and so we'll need a way to access it in order to run commands within the machine. Luckilyvagrant 3. provides just such a feature. Type the following command:

vagrant ssh Note that after running this command, our command line changes to show a different prompt:

ubuntu@ubuntu-xenial:~ This designates that we're running commands inside the virtual machine.

1. While we're at the above virtual machine prompt, navigate to the example we want to run:

ubuntu@ubuntu-xenial:~ cd

quorum-examples/7nodes/ 1. Our Quorum client is nearly ready to go. We need to run two more commands within the virtual machine. The first one creates seven Quorum nodes we can use to simulate a real Quorum deployment. The second one starts up those seven nodes. While the first command only needs to be run one, you should run the second command anytime you restart the virtual machine.

ubuntu@ubuntu-xenial:~/quorum-examples/7nodes ./raft-init.sh ubuntu@ubuntu-xenial:~/quorum-examples/7nodes ./raft-start.sh Success! We now have seven Quorum nodes set up that we can use to represent seven different actors on our private network.

Connecting Truffle to Quorum¶

To set up Truffle, we're going to start by creating a bare Truffle project, without any contracts or code.

1. We want to leave the old command line window alone and let the Quorum example run without bothering it, so open a second terminal.
2. In the new terminal, navigate to your workspace and create a new directory for your Truffle project.

cd

workspace mkdir myproject 1. Next, navigate to the new directory and initialize the bare Truffle project:

cd

myproject truffle init If you look at the contents of themyproject directory, you'll notice directories were created for you. See theTruffle documentation for more information about Truffle's project structure.

1. Before moving onto code, we need to configure Truffle to point to our running Quorum client. For this example, we'll edit ourdevelopment
2. network configuration withintruffle-config.js
3. to point to the first node available in the 7nodes example:

// File: truffle-config.js (edited for 7nodes example) module . exports

=

{

networks :

{

development :

{

host :

"127.0.0.1" ,

port :

22000 ,

// was 8545

network_id :

"*" ,

// Match any network id

gasPrice :

0 ,

gas :

4500000 ,

type :

"quorum" ,

// needed for Truffle to support Quorum

},

}, }; Note that we changed the port that Truffle normally connects to (we changed it to22000). Because of the magic of VirtualBox and Vagrant, the node running inside the virtual machine is made available to us via local ports, so connecting via127.0.0.1 and22000 will work just fine.

Note: The seven Quorum clients respond on ports 22000 (node 1) through 22006 (node 7).

Now that we have Truffle set up, we can move onto code.

Deploying smart contracts on Quorum¶

We won't spend too much time talking about writing or deploying contracts in Truffle since we haveample documentation , however we do want to show how deploying contracts applies to Quorum.

1. First, copy the following contract into a new file, calledSimpleStorage.sol
2. . Place it in yourcontracts/
3. directory:

// File: ./contracts/SimpleStorage.sol pragma solidity

^ 0.4.17 ; contract

SimpleStorage

{

uint

public storedData ;

constructor (uint

initVal)

public

{

storedData

initVal;

}

function

set (uint

x)

public

{

storedData

x;

}

function

get ()

view public

returns

(uint

retVal)

{

return

storedData;

} } 1. Ensure the contract compiles using the commandtruffle compile 2. within your project directory.

truffle compile 1. Next, create a new migration called2_deploy_simplestorage.js 2. within yourmigrations/ 3. directory. Note that this migration is just like any other you'd create for Truffle, but there's one important difference. Let's see if you can catch it.

// File: ./migrations/2_deploy_simplestorage.js var

SimpleStorage

=

artifacts . require ("SimpleStorage"); module . exports

=

function

(deployer)

{

// Pass 42 to the contract as the first constructor parameter

deployer . deploy (SimpleStorage ,

42 ,

{

privateFor :

["ROAZBWtSacxXQrOe3FGAqJDyJjFePR5ce4TSIzmJ0Bc="],

}); }; If you guessed the difference wasprivateFor , you got it!privateFor is an extra transaction parameter added to Quorum that specifies thatthe transaction you're making -- in this case a contract deployment --is private for a specific account, identified by the given public key . For this transaction, the public key we chose represents node 7. Since we previously configured Truffle to connect to node 1, this transaction will deploy a contract from node 1, making the transaction private between node 1 and node 7.

1. Now it's time to deploy your contracts. Run thetruffle migrate
2. command and watch your contracts be successfully deployed:

truffle migrate You will see output that looks like this. Your transaction IDs will be different though.

Using network 'development' .

Running migration: 1_initial_migration.js Deploying Migrations... Migrations: 0x721650d027d87cd247a3a776c4b6170bf1e5b936 Saving successful migration to network... Saving artifacts... Running migration: 2_deploy_simplestorage.js Deploying SimpleStorage... SimpleStorage: 0x10ae69385c79ef3eb815ac008a7013d6878f1d38 Saving successful migration to network... Saving artifacts... Now that the contract's deployed, it's off to the races.

Using Quorum's privacy features to make transactions private¶

We originally configured Truffle to point our development environment to the first of the seven nodes provided by the example. You can think of the first node as "us", as if we were developing a dapp for a private network that's used by multiple other parties. Since the 7nodes example provides us seven nodes to work with, we can tell Truffle about the other nodes so we can "be" someone else, and ensure the contract we deployed was private.

1. To add a new network configuration, edit yourtruffle-config.js
2. file again and add additional network configuration, choosing a network name that best describes the network connection you're adding. In this case, we're going to add a connection to node 4 (nodefour
3.) and node 7 (nodeseven
4.):

// File: truffle-config.js module . exports

=

{

networks :

{

development :

{

host :

"127.0.0.1" ,

port :

22000 ,

// was 8545

network_id :

"*" ,

// Match any network id

gasPrice :

0 ,

gas :

4500000 ,

type :

```
"quorum" ,
// needed for Truffle to support Quorum
},
nodefour :
{
host :
"127.0.0.1" ,
port :
22003 ,
network_id :
"*" ,
// Match any network id
gasPrice :
0 ,
gas :
4500000 ,
type :
"quorum" ,
// needed for Truffle to support Quorum
},
nodeseven :
{
host :
"127.0.0.1" ,
port :
22006 ,
network_id :
"*" ,
// Match any network id
gasPrice :
0 ,
gas :
4500000 ,
type :
"quorum" ,
// needed for Truffle to support Quorum
},
```

}, }; Like before, VirtualBox and Vagrant are making these nodes available to us through local ports, so these configurations can look the same as ourdevelopment configuration except with a different ports specified.

1. Now that our configuration is set up, we can use the Truffle console to interact with our deployed contract. First let's be "us" (node 1), configured via ourdevelopment
2. configuration. The easiest way to do this is to launch theTruffle console
3. , which lets us interact with our deployed contracts directly.

truffle console You will see a new prompt:

truffle(development)

1. Here, we'll get the deployed instance of the SimpleStorage contract and then get the integer value we specified on deployment. Enter the following command:

truffle (development)

SimpleStorage . deployed (). then (function

(instance)

{

return

instance . get ();

}); You'll see the following response:

{

[String :

'42']

s :

1 ,

e :

1 ,

c :

[

42

]

} Note that Truffle's contract abstractions use Promises to interact with Ethereum. This can be a little cumbersome on the console as it requires a few extra key strokes to get things done, but within your application it makes control flow a lot smoother. Additionally, take a look at the output we received: We got42 back, but as an object. This is because Ethereum can represent larger numbers than those natively represented by JavaScript, and so we need an abstraction in order to interact with them.

1. Now let's try accessing the SimpleStorage contract as node four. To do this, quit out of the console (usingCtrl + C
2. /Command + C
3.), and then launch the console again, but this time specifying the connection to node 4 instead of node 1:

truffle console --network nodefour You'll see a new prompt:

truffle(nodefour)

1. Run the same command as above to get the integer value from the SimpleStorage contract:

truffle (nodefour)

SimpleStorage . deployed (). then (function

(instance)

{

return

instance . get ();

}); You'll see the following response:

{

[String :

'0']

s :

1 ,

e :

0 ,

c :

[

0

]

} You'll notice we got0 back instead. This is because the accounts represented by node 4 weren't privy to this contract.

1. Lastly we can try with node 7, whichwas
2. privy to this contract. Quit and relaunch the console again:

truffle console --network nodeseven You'll see a new prompt:

truffle(nodeseven)

1. Run the same command as above to get the integer value from the SimpleStorage contract:

truffle (nodeseven)

SimpleStorage . deployed (). then (function

(instance)

{

return

instance . get ();

}); You'll see the following response:

{

[String :

'42']

s :

1 ,

e :

1 ,

c :

[

42

]

} As you can see, we get42 ! This shows that we can deploy contracts that are only available to our desired parties.

Interacting with contracts privately¶

So far, we've shown you how to deploy contracts that are private within your migrations. When building a dapp on Quorum, it'd also be helpful to learn how to makeall transactions private.

Truffle uses its@truffle/contract contract abstraction wherever contracts are used in JavaScript. When you interacted withSimpleStorage in the console above, for instance, you were using a@truffle/contract contract abstraction. These abstractions are also used within your migrations, your JavaScript-based unit tests, as well as executing external scripts with Truffle.

Truffle's contract abstraction allow you to make a transaction against any function available on the contract. It does so by evaluating the functions of the contract and making them available to JavaScript. To see these transactions in action, we're going to use an advanced feature of Truffle that lets us execute external scripts within our Truffle environment.

1. Creating a file calledsampletx.js
2. and save it in the root of your project (the same directory as yourtruffle-config.js
3. file). Then fill it with this code:

```
var

SimpleStorage

=

artifacts . require ( "SimpleStorage" ); module . exports

=

function

( done )

{

console . log ( "Getting deployed version of SimpleStorage..." );

SimpleStorage . deployed ()

. then ( function

( instance )

{

console . log ( "Setting value to 65..." );

return

instance . set ( 65 ,

{

privateFor :

[ "ROAZBWtSacxXQrOe3FGAqJDyJjFePR5ce4TSIzmJ0Bc=" ],

});

})

. then ( function

( result )

{

console . log ( "Transaction:" ,

result . tx );
```

console . log ("Finished!");

done ();

})

. catch (function

(e)

{

console . log (e);

done ();

}); }; This code does two things: First, it asks Truffle to get our contract abstraction for the SimpleStorage contract. Then, it finds the deployed contract and sets the value managed by SimpleStorage to65 , using the contract'sset() function. As with the migration we wrote previously, theprivateFor parameter can be appended within an object at the end of the transaction to tell Quorum that this transaction is private between the sender and the account represented by the given public key.

1. Run this code usingtruffle exec
2. :

truffle exec

sampletx.js Your output should look something like this (your transaction ID will be different though):

Using network 'development' .

Getting deployed version of SimpleStorage... Setting value to 65 ... Transaction: 0x0a7a661e657f5a706b0c39b4f197038ef0c3e77abc9970a623327c6f48ca9aff Finished! 1. We can now use the Truffle console, like before, to check the results of this transaction. Let's see the value as node one:

truffle console truffle (development)

SimpleStorage . deployed (). then (function

(instance)

{

return

instance . get ();

}); The response will be:

{

[String :

'65']

s :

1 ,

e :

1 ,

c :

[

65

]

} We got65 ! Now let's do node 4 (not privy to the transaction):

truffle console --network nodefour truffle (nodefour)

SimpleStorage . deployed (). then (function

(instance)

{

return

instance . get ();

}); The response will be:

{

[String :

'0']

s :

1 ,

e :

0 ,

c :

[

0

]

} We got zero, as expected. Now let's try node 7:

truffle console --network nodeseven truffle (nodeseven)

SimpleStorage . deployed (). then (function

(instance)

{

return

instance . get ();

}); The response will be:

{

[String :

'65']

s :

1 ,

e :

1 ,

c :

[

65

]

} And we got65 again, as we should expect. This is how we can use Truffle's contract abstractions to make private transactions with Quorum.

Is that all Truffle can do?¶

Absolutely not! What we've shown you today is everything that makes building dapps for Quorum different than building dapps for the public Ethereum network. And what you'll see is it's not different at all: The only difference is adding theprivateFor parameter for deployments and transactions you'd like to keep private. The rest is the same!

In fact, now that you have the basics, you can exploreall our other resources for building dapps with Truffle, includingtutorial ,writing advanced deployment scripts ,unit testing (with Solidity too), and much more.

By building with Truffle, you now have access to not only the best development tools and technologies (like Quorum), but you also have access to the largest Ethereum developer community around. Don't hesitate todrop us a line on Twitter orget help from fellow Trufflers in our community GitHub Discussions channel . There's always someone available to answer any questions you have.

Cheers, and happy coding!