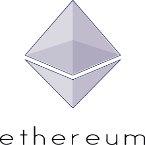
hands on lab instructions

****

Azure – Ethereum SmartContracts DevTest Lab 2

Blockchain as a Service

****

Marley Gray – Microsoft

Contents

[Azure Lab Overview 2](#_Toc443315086)

[**Test your Ðapp and SmartContracts** 3](#_Toc443315087)

[Advanced Lab – CryptoBond (Incomplete) 9](#_Toc443315088)

[**Create your Ðapp** 9](#_Toc443315089)

[**Add Ðapp styles** 11](#_Toc443315090)

[**Using ethereum packages** 11](#_Toc443315091)

[**Ðapp code structure** 12](#_Toc443315092)

[**Bundle your Ðapp** 12](#_Toc443315093)

# Azure Lab Overview

This lab will have you create your first SmartContract application.

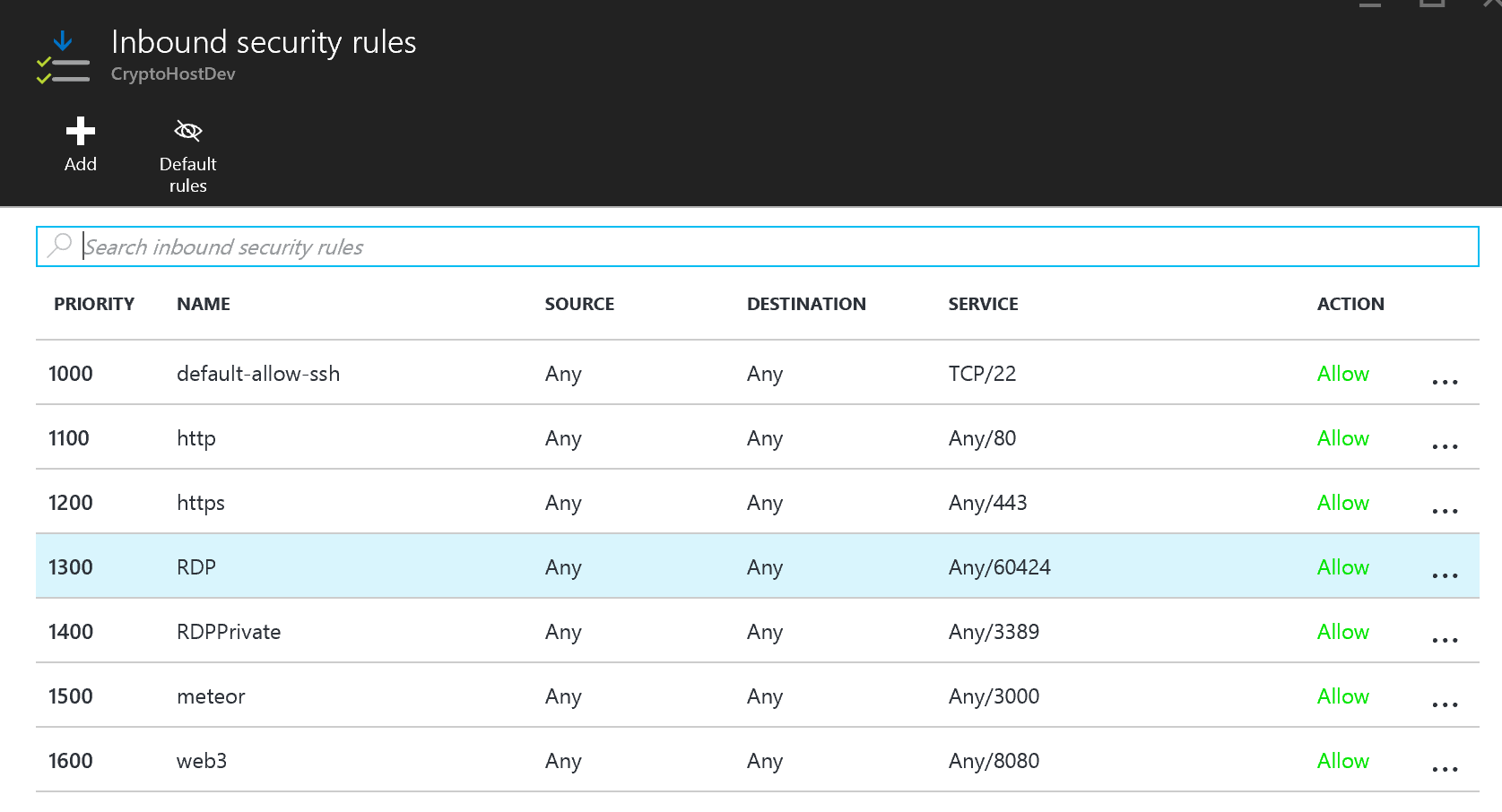
**Lab – Step by Step**

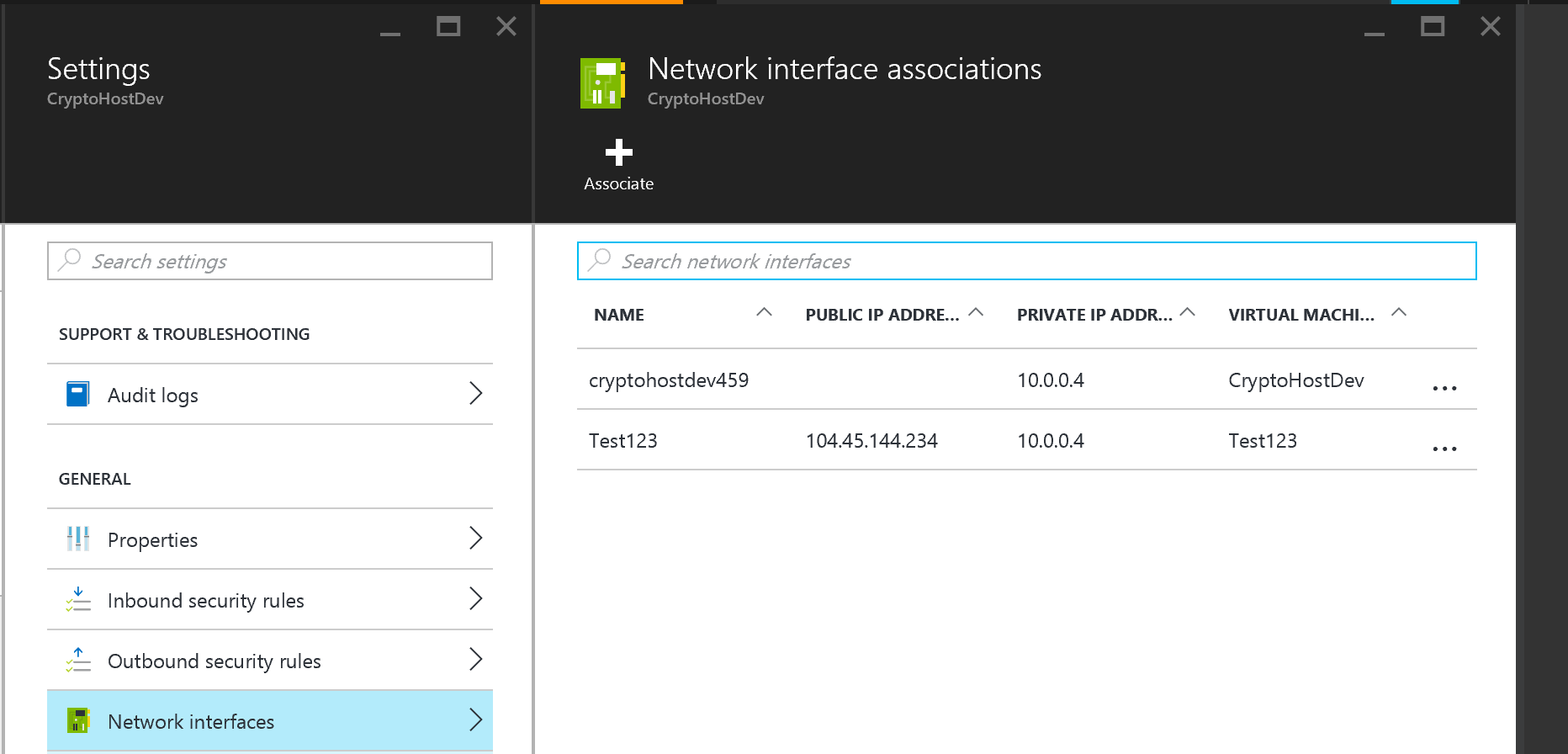
Participants in this lab will do the following:

* Install Meteor to build your application
* Write and Test your SmartContract
* Perform transactions with your SmartContract

**Lab – Dependencies**

This lab requires that certain ports be open on the Ethereum node accessible outside the private vnet in Azure. To do this create a Network Security Group similar to this one that has ports: 3000 and 8080 open and apply it to the network interface the node is operating on. In most cases this will be done for you ahead of time.

****



**Test your Ðapp and SmartContracts**

Install Meteor and our Sample Template Dapps

$ curl https://install.meteor.com/ | sh

$ sudo apt-get install git

$ git clone http://github.com/SilentCicero/meteor-dapp-cosmo.git

Stop your Geth Miner and restart it with these options and unlock the miner account:

$ geth –rpc –rpcaddr “0.0.0.0" --rpcport "8080" --rpccorsdomain "\*" --datadir "chains/devtest" --port "2402" --ipcapi "admin,db,eth,debug,miner,net,shh,txpool,personal,web3" --rpcapi "db,eth,net,web3" --networkid 1001201 –unlock=primary console

>personal.unlockAccount(eth.coinbase)

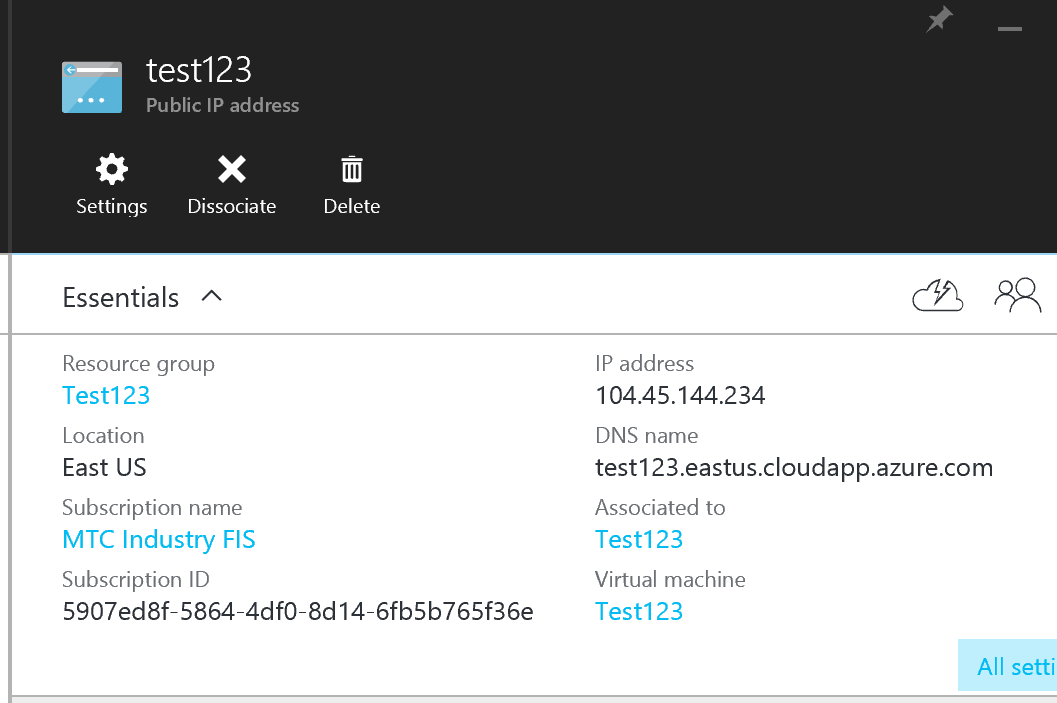
>miner.start(1)

Start Cosmo (Dapp tool) using Meteor

$ cd meteor-dapp-cosmo/app’

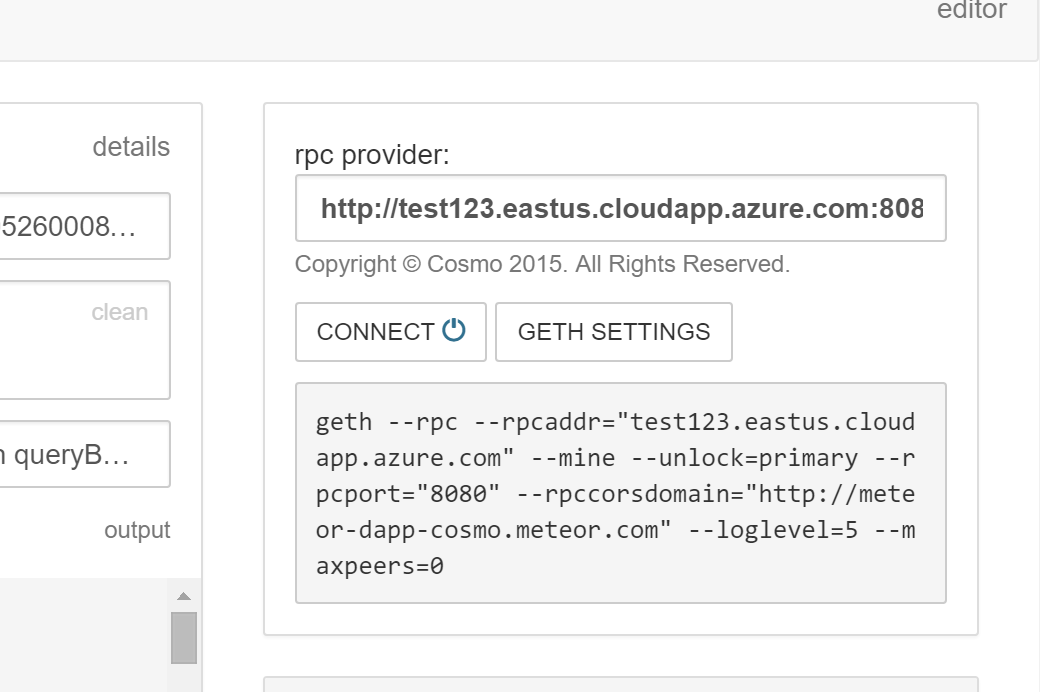
$ meteor

Go to the Cosmo Dashboard and deploy the Coin SmartContract. First you will need to get your DNS host address from the Azure portal:

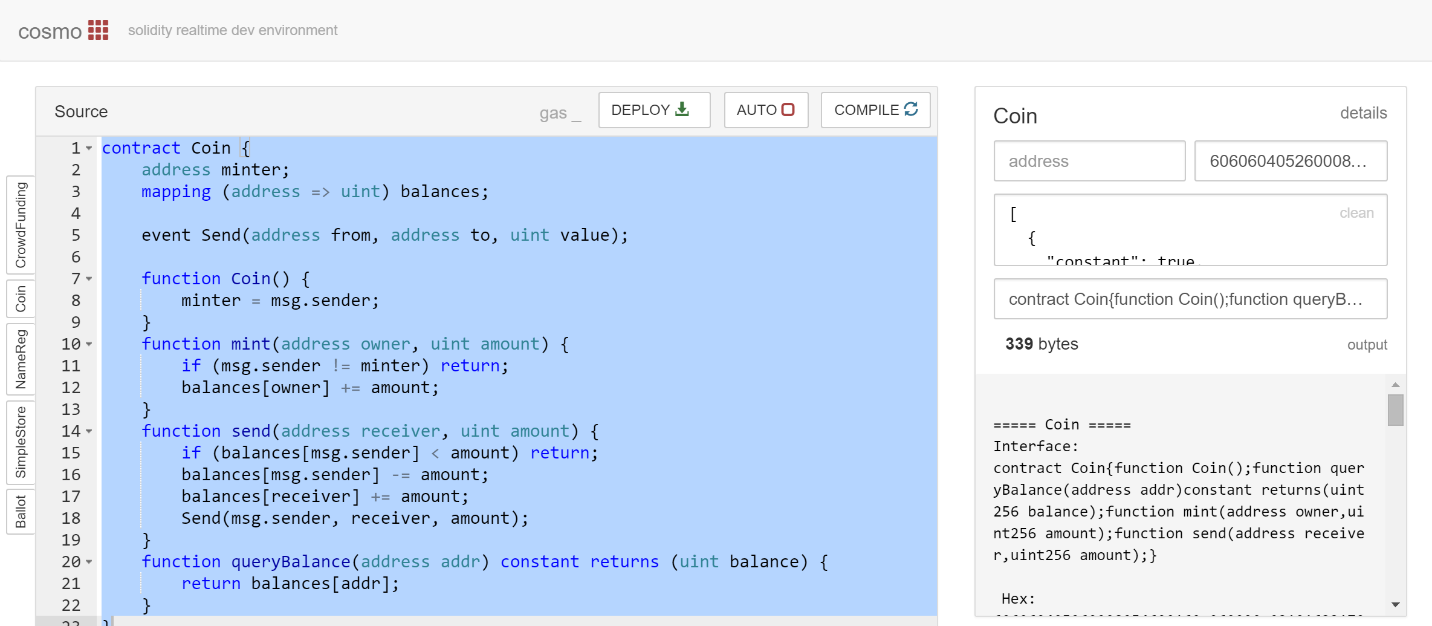


Then open up a browser and go to your URL: the above would be <http://test123.eastus.cloudapp.azure.com:3000>

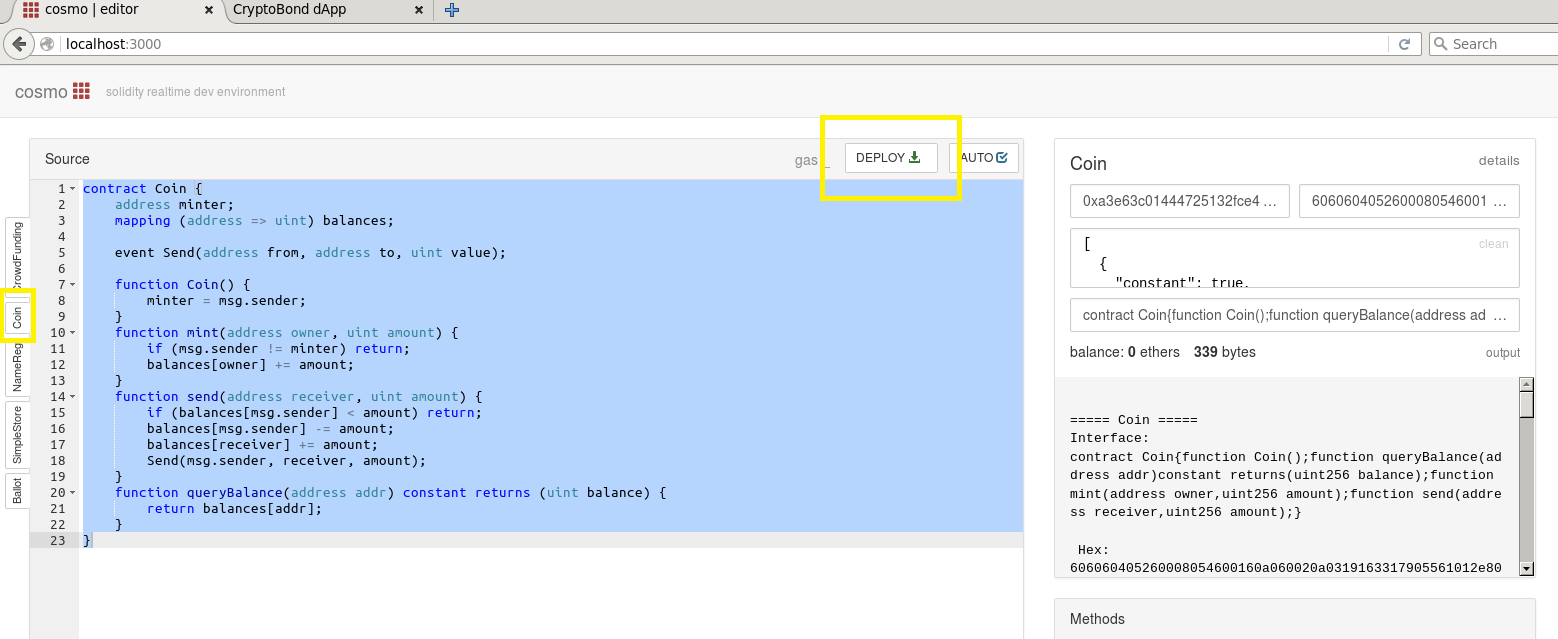
Next set your RPC Provider in the browser to the Web3 API on your node:



Select the Coin Tab on the Left Hand Side of the browser client:



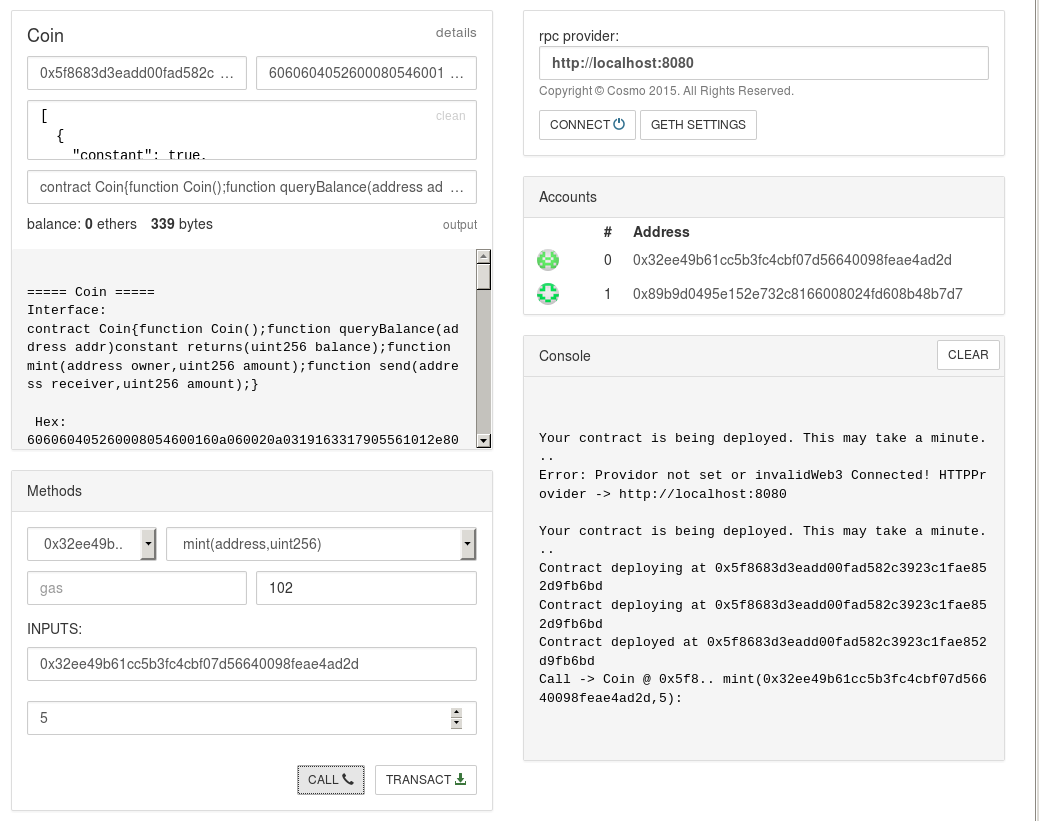
Select Deploy



Once the Coin is deployed go to your Geth terminal and create a new account to transact with:

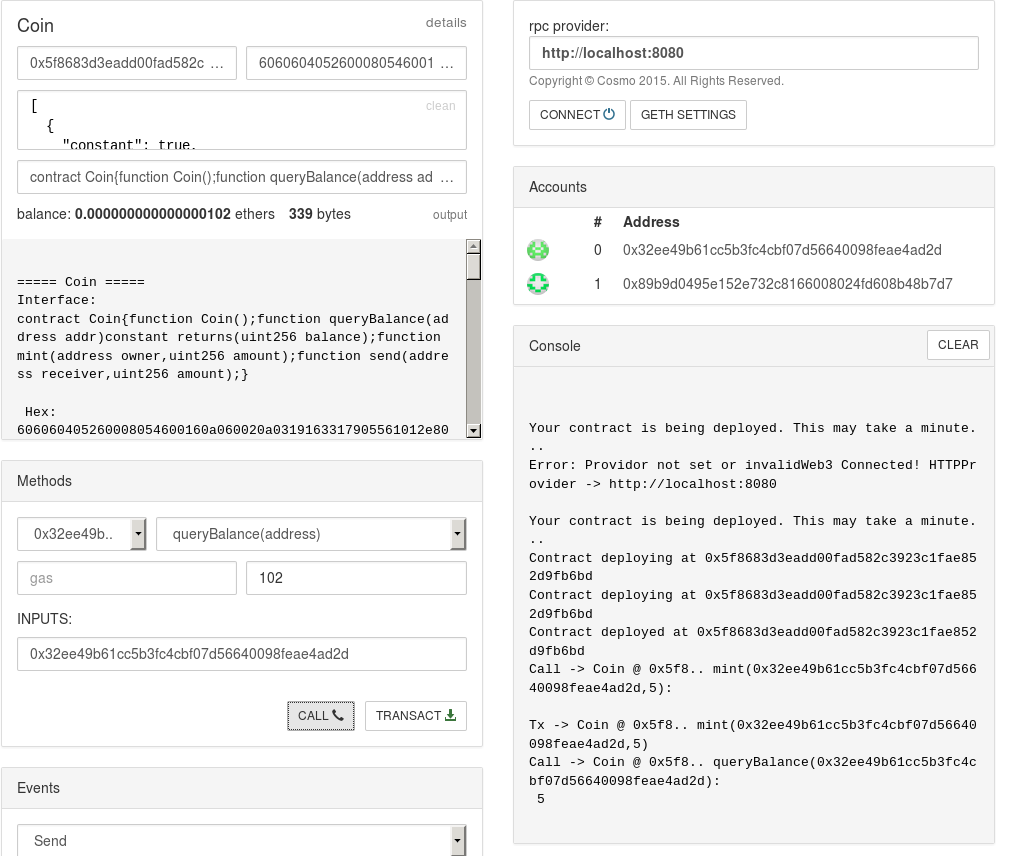
> personal.newAccount()

Now to Mint some of your CryptoCurrency which back to the browser and select the Mint method:

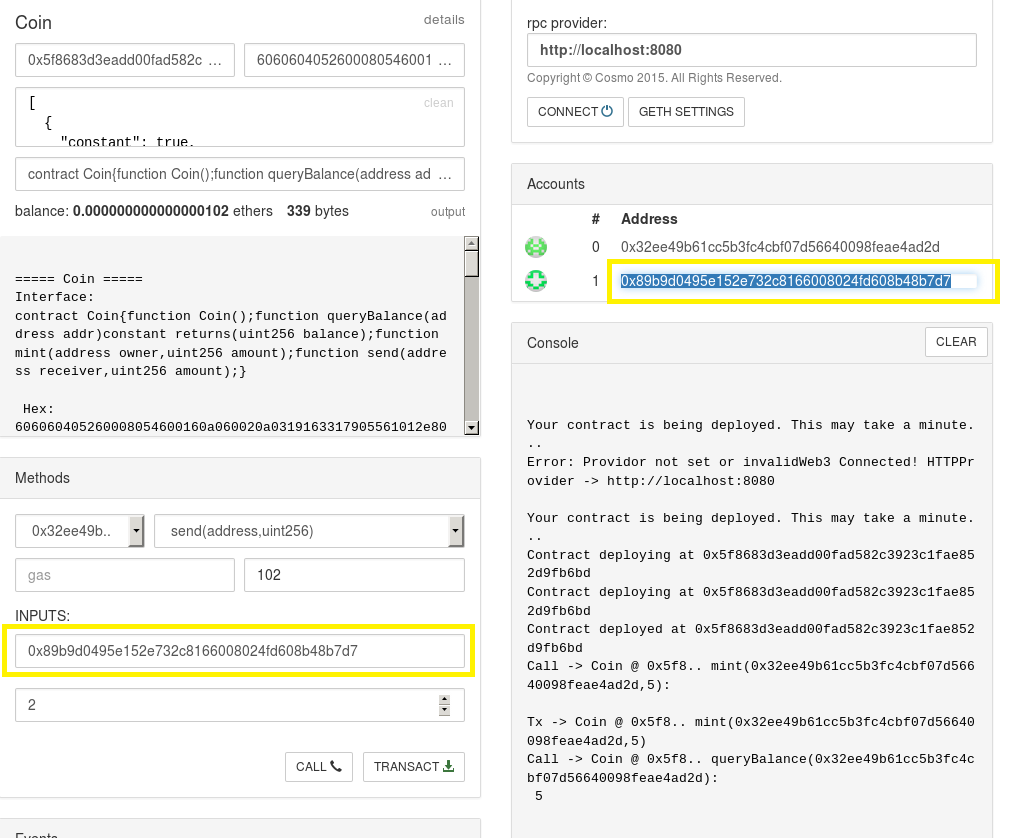


Make sure the from address is the unlocked miner address (eth.coinbase) and to deposit it to that same address. Select Call and Transact to place the transaction on the blockchain.

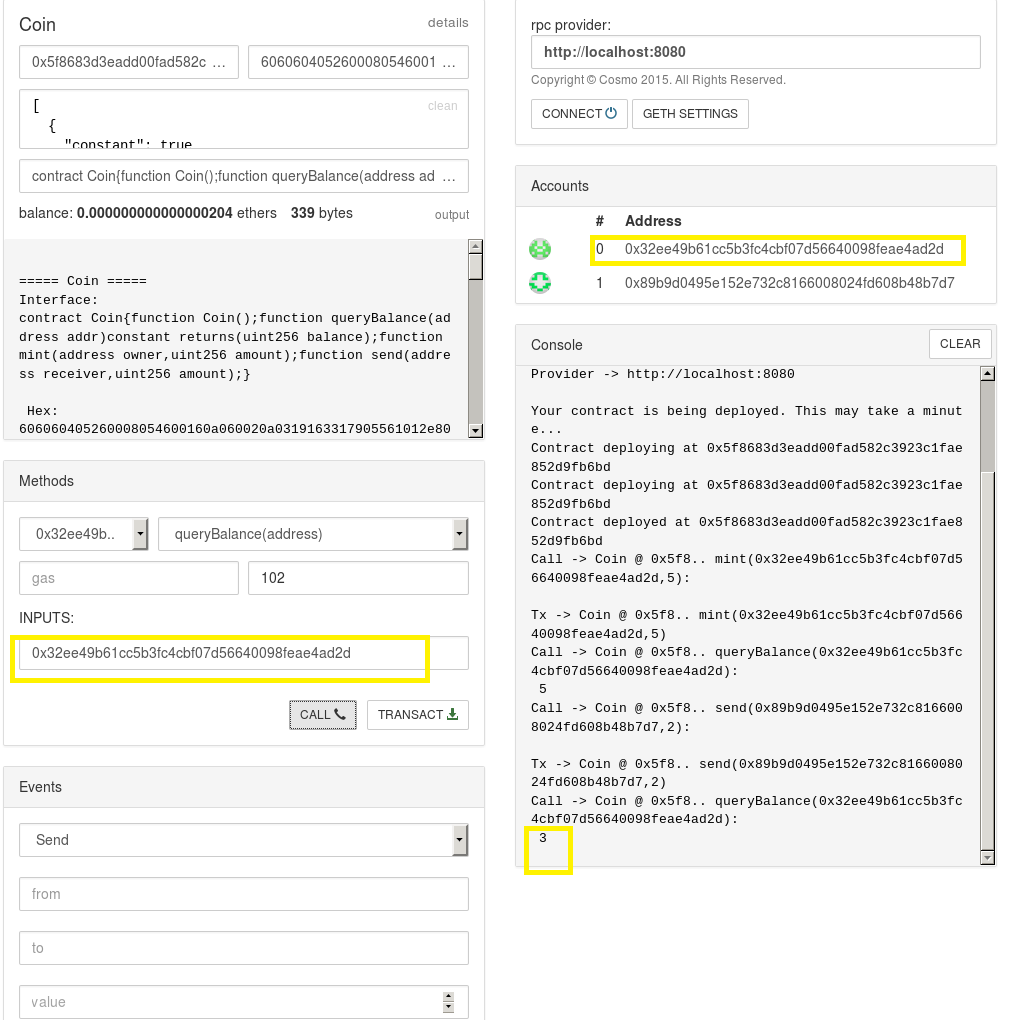
Now check your balance, by using the queryBalance method:



Note you now have 5 Coin. Now select the send() method to send some coin to the other address on the blockchain.



Select Call and Transact. Now re-check your balance…you should have 3 Coin now:



Complete! You have now successfully completed your first SmartContract Dapp on Ethereum.

# Advanced Lab – CryptoBond (Incomplete)

Then create an app:

$ meteor create myCryptoBond

$ cd myCryptoBond

Next add some packages:

$ meteor add ethereum:web3

$ meteor add ethereum:dapp-styles

$ meteor add ethereum:elements

$ meteor add ethereum:accounts

$ meteor add ethereum:blocks

I recommend also to add the following packages:

* [ethereum:dapp-styles](https://atmospherejs.com/ethereum/dapp-styles) - The LESS/CSS framework which gives your dapp a nice Mist-consistent look.
* [ethereum:tools](https://atmospherejs.com/ethereum/tools) - This package gives you the EthTools object with a set of formatting an conversion functions and template helpers for ether.
* [ethereum:elements](https://atmospherejs.com/ethereum/elements) - A set of interface elements specifically made for ethereum, see this [Demo](http://ethereum-elements.meteor.com/) for more.
* [ethereum:accounts](https://atmospherejs.com/ethereum/accounts) - Gives you the reactive EthAccounts collection with all current available ethereum accounts, where balances will be automatically updated.
* [ethereum:blocks](https://atmospherejs.com/ethereum/blocks) - Gives you the reactive EthBlocks collection with the latest 50 blocks. To get the lastest block use EthBlocks.latest (It will also have the latest default gasPrice)
* [frozeman:template-var](https://atmospherejs.com/frozeman/template-var) - Gives you the TemplateVar object, that allows you to set reactive variables, which are template instance specific. See the [readme](https://atmospherejs.com/frozeman/template-var) for more.
* [frozeman:persistent-minimongo2](https://atmospherejs.com/frozeman/persistent-minimongo2) - Allows you to auto persist your minimongo collection in local storage

**Create your Ðapp**

**A short excursion into Meteors folder structure**

Meteor doesn't force you to have a specific folder structure, though some folders have specificls meaning and will be treated differently when bundling/running your application.

Folders with specific treatment

* client - files in a folder called client will only be loaded by the client part of your app and as we are building a Ðapp, thats where most of our files go.
* lib - files in folders called lib will load before other files in the same folder. This is an ideal place your init files, libraries, or ethereum specifc files.
* public - a folder called public contains assets meteor will make available on the root of your webserver (or later bundled Ðapp)
* There are a few more specifc folders like server, tests, packages, etc. If you want to get to know them take a look at the [Meteor docs](http://docs.meteor.com/#/full/structuringyourapp)

So to build a Ðapp we ideally create the following folder structure in our myDapp folder:

- myDapp

- client

- lib

- myDapp.html

- myDapp.js

- myDapp.css

- public

**Note** The community provides also Meteor Ðapp Boilerplates like this on from Nick Dodson: <https://github.com/SilentCicero/meteor-dapp-boilerplate>

**Connect your Ðapp**

To connect our dapp we need to start geth with the right CORS headers in another terminal:

$ geth --rpc --rpccorsdomain "http://localhost:3000"

We also need to set the provider. Ideally we create a file in our lib folder called init.js and add the following line:

if(typeof web3 === 'undefined')

web3 = new Web3(new Web3.providers.HttpProvider('http://localhost:8545'));

**Run your Ðapp**

Now we can run our Ðapp by simply running:

$ meteor

If we go to http://localhost:3000, we should see a website appear and if we open the browser console we can use the web3 object to query the geth node:

> web3.eth.accounts

['0xfff2b43a7433ddf50bb82227ed519cd6b142d382']

**Add Ðapp styles**

If you want your Ðapp to nicely fit later into Mist and have follow the official look use the [dapp-styles css css/less framework](https://atmospherejs.com/ethereum/dapp-styles).

Note that they are under heavy development and the class names and elements may change.

To add it simple add the following packages to your Ðapp:

$ meteor add less

$ meteor add ethereum:dapp-styles

Now rename you myDapp.css to myDapp.less and add the following line inside:

// libs

@import '{ethereum:dapp-styles}/dapp-styles.less';

Now you can use all dapp-styles classes and also overwrite all variables of the framework. You can find them [in the repo](https://github.com/ethereum/dapp-styles/blob/master/constants.import.less). Overwrite them by copying them to your myDapp.less file and set different values.

**Using ethereum packages**

To make your life as a Ðapp developer easier we provide some packages that help you build Ðapps faster.

If you add the recommended packages above you should have the [ethereum:tools](https://atmospherejs.com/ethereum/tools), [accounts](https://atmospherejs.com/ethereum/accounts) and [ethereum:blocks](https://atmospherejs.com/ethereum/blocks) packages available.

These 3 packages give you the EthTools, EthAccounts and Ethblocks objects, which give you formatter functions, a collection with the accounts from web3.eth.accounts (with auto updated balance) and a collection of the last 50 blocks.

Most of these functions are reactive so they should make building interfaces a breeze.

**Example usage**

If you look into you myDapp.html you will find the hello template. Just add a helper called {{currentBlock}} some where between the <template name="hello">..</template> tags.

Now open the myDapp.js and add after the counter: function.. the currentBlock helper:

Template.elements.helpers({

counter: function () {

...

},

currentBlock: function(){

return EthBlocks.latest.number;

}

});

Then initialize EthBlocks by adding EthBlocks.init(); after Session.setDefault('counter', 0);

If you now check your Ðapp in the browser you should see the latest block number, which will increase once you mine.

For more examples please checkout the packages readmes and the [*demo*](http://ethereum-elements.meteor.com/) ([*source*](https://github.com/frozeman/meteor-ethereum-elements-demo)) for more.

**Ðapp code structure**

This tutorial won't go into building apps with Meteor. For this please refer to the [*Meteor's tutorials*](https://www.meteor.com/tutorials/blaze/creating-an-app), [*A list of good resources*](https://www.meteor.com/tools/resources), [*EventMinded*](https://www.eventedmind.com/) (payed tutorials) or books like [*Building Single-page Web Apps with Meteor*](https://www.packtpub.com/web-development/building-single-page-web-apps-meteor) or [*Discover Meteor*](http://discovermeteor.com/).

TODO Short:

* put ethereum related stuff into client/lib/ethereum/somefile.js
* use myCollection.observe({added: func, changed: func, removed: func}) to communicate to ethereum, keep ethereum logic out of your app as much as possible. This way you just write and read from your reactive collections and the observe functions will handle the rest (e.g. sendTransactions)
* Filters etc will add logs etc to your collections. So you keep all the callback mess out of your app logic.

For an example see the [Ethereum-Wallet](https://github.com/ethereum/meteor-dapp-wallet).

**Bundle your Ðapp**

To bundle your Ðapp into a local standalone file use [meteor-build-client](https://github.com/frozeman/meteor-build-client):

$ npm install -g meteor-build-client

$ cd myDapp

$ meteor-build-client ../build --path ""

This will put your Ðapps static files into the build folder, above your myDapp folder.

The last option --path will make the linking of all files relative, allowing you to start the app by simply clicking the build/index.html.

Be aware that when running your app on the file:// protocol, you won't be able to use client side routing, due to web security. Later in mist you will be able to use client side routing, as dapps are severed over the eth:// protocol.