



Smart Contract Development & Testing with Truffle

University of Missouri – St. Louis



Our Development Environment

Truffle

- Truffle will be our main development environment.
- It can compile, deploy, test, and interact with smart contracts.
- Installation:

```
> npm install -g truffle
```

Ganache

- Ganache creates a development blockchain running on localhost.
- User-friendly graphical interface
- Installation:
 - DL latest release from <https://github.com/trufflesuite/ganache/releases>

```
> tar -xzf ganache-*.tar.gz  
> cd ganache-*  
> npm install && npm run build-<OS>  
> ./out/make/ganache-*.AppImage
```



More on Ganache

- Ganache creates 10 accounts, with 100.00 ETH in each accounts' wallet.
- It *automines*, meaning it only consumes CPU resources when there are pending blocks.
- You can view accounts, logs, past transactions, and past blocks. (TX data is encrypted)
- Defaults:
 - hostname: 127.0.0.1
 - port-number: 7545
 - network-id: 5777



Truffle

- In a new directory, run `truffle init`.
- This creates two directories, *migrations* & *contracts*, and a `truffle.js` config file.
 - *contracts* contains `Migrations.sol`, a contract required for deploying your smart contract.
 - *migrations* contains `1_initial_migrations.js`, which deploys the required Migrations contract.



More on Truffle

- For each contract, we need a *migration* file to allow deployment of our contract.
- Migrations occur on `truffle deploy` or `truffle migrate`
- Here's an example migration for a contract called `MyCoin`:

```
var MyCoin = artifacts.require("MyCoin");

module.exports = function(deployer) {
  deployer.deploy(MyCoin);
};
```

`./migrations/2_deploy_mycoin.js`



Truffle Console

- Truffle's console is where many of our interactions will take place. Start with `truffle console`.
- We can interact directly or run scripts.
- Common commands are:
 - `build, compile, deploy/ migrate, test, exec`
- Here is a complete list of commands for Truffle:
<http://truffleframework.com/docs/advanced/commands>



Contract Abstractions

- *Contract abstractions* allow interaction with your smart contracts using JavaScript.
- Assume we have a contract called `MyCoin` already deployed.
- In Truffle console, we can retrieve an abstraction of `MyCoin`, called instance, with:

```
> MyCoin.deployed().then(function(instance) {  
  console.log(instance);  
});  
TruffleContract:{  
  // - address: "0xa9f441a487754e6b27ba044a5a8eb2eec77f6b92"  
  // - allEvents: ()  
  // - getBalance: ()  
  // - transfer: ()  
}
```



Interacting with Smart Contracts

- Assume the contract is called **MyCoin** and owner address is `0x42`
 - MyCoin sets owner initial balance to 1,000,000 and has two functions: **`transfer(_to, _value)`** and **`getBalance(_address)`**
- To get the deployed contract's address, in truffle console run `MyCoin.address`
- To retrieve your (owner's) balance:

```
> var balance;  
  
> MyCoin.deployed().then(function(instance) { return  
instance.getBalance.call("0x42");  
}).then(function(value) { balance = value; });  
  
> balance.toString(10);  
    '1000000'
```




Interacting with Smart Contracts

- MyCoin has a new user at address `0x23` who pays for \$10 worth of MyCoin. *Suppose MyCoin is valued at ~\$0.01*
- The owner now sends the new user 1000 MyCoin *tokens*.

```
> MyCoin.deployed().then(function(instance) { return
instance.transfer("0x23", 1000, {from: "0x42"});
}).then(function(result)
{ console.log(result.tx); });
0x???????????????????????????????? // the transaction id
```

- The new user now checks his balance.

```
> var balance;
> MyCoin.deployed().then(function(instance) { return
instance.getBalance.call("0x23"); }).then(function(value)
{ balance = value; });
> balance.toString(10);
'1000'
```



Smart Contracts: Calls vs Transactions

- Calls:
 - Free (no gas cost)
 - No changes to network
 - Processed immediately
 - Expose a return value
 - Transactions:
 - Cost gas
 - Change state of network
 - Processed in next block
 - Returns only TX info
-

Assume we have an abstraction of MyCoin called **instance**.

- `instance.getBalance.call("0x")`
- `instance.transfer("0x", 1, {from:"0x"})`



Writing Tests with Truffle

- Truffle has a Javascript testing framework based on Mocha.
- This is a test which checks whether the constructor successfully gave 1,000,000 tokens to the owner's account.

`./test/mytest.js`

```
var MyCoin = artifacts.require("MyCoin");

contract('MyCoin', function(accounts) {
  it("should give owner an initial balance of 1M.", function() {
    return MyCoin.deployed().then(function(instance) {
      return instance.getBalance.call(accounts[0]);
    }).then(function(balance) {
      assert.equal(balance.toString(10), 1000000, "TestFailed");
    });
  });
});
```

- Truffle testing happens in a *clean-room* environment.
 - This means it makes no permanent changes to the blockchain.



Writing External Scripts for Truffle

- Truffle console has many environment variables for use. `MyCoin.deployed()` won't work in an external script.
- We must use `MyCoin.at(_contractAddress)`
- Scripts can be executed with `truffle exec ./scripts/<script>.js`
- The following script checks the balance of owner's account.

```
var MyCoin = artifacts.require("MyCoin");
var contractAddress = "0x99";
var owner = "0x42";

module.exports = function(callback) {
  var instance = MyCoin.at(contractAddress);
  var balance = instance.getBalance.call(owner);
  balance.then(function(result) {
    console.log(result.toString(10));
  });
};
```

`./scripts/getownerbalance.js`



Resources

- http://truffleframework.com/docs/getting_started/contracts
- <http://truffleframework.com/ganache/>
- <https://github.com/mochajs/mocha/wiki/Shared-Behaviours>
- <https://mochajs.org/#getting-started>