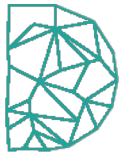


DECENTRALIZED
TRAINING SERIES

Day 2 - Session 1

# Advanced Features on Eth dApp Development

## Bootstrapping Session

Klitos Christodoulou, Ph.D.

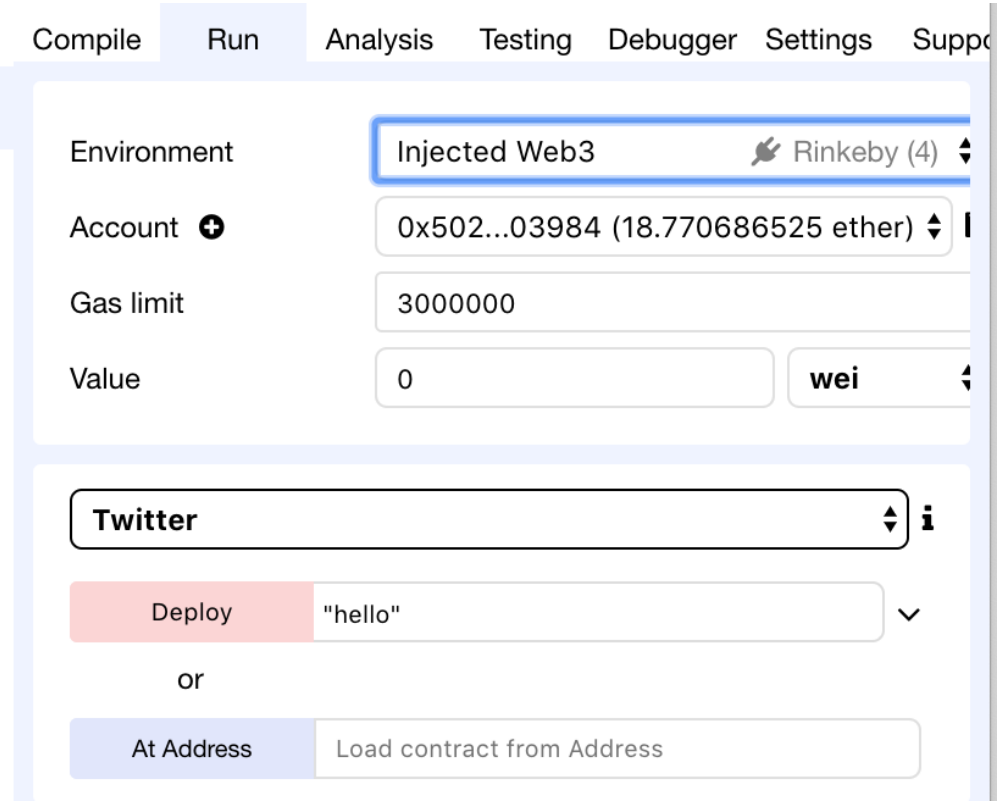
christodoulou.kl@unic.ac.cy

UNIC | Institute For the Future

CERTH CENTRE FOR RESEARCH & TECHNOLOGY HELLAS | iti Information Technologies Institute

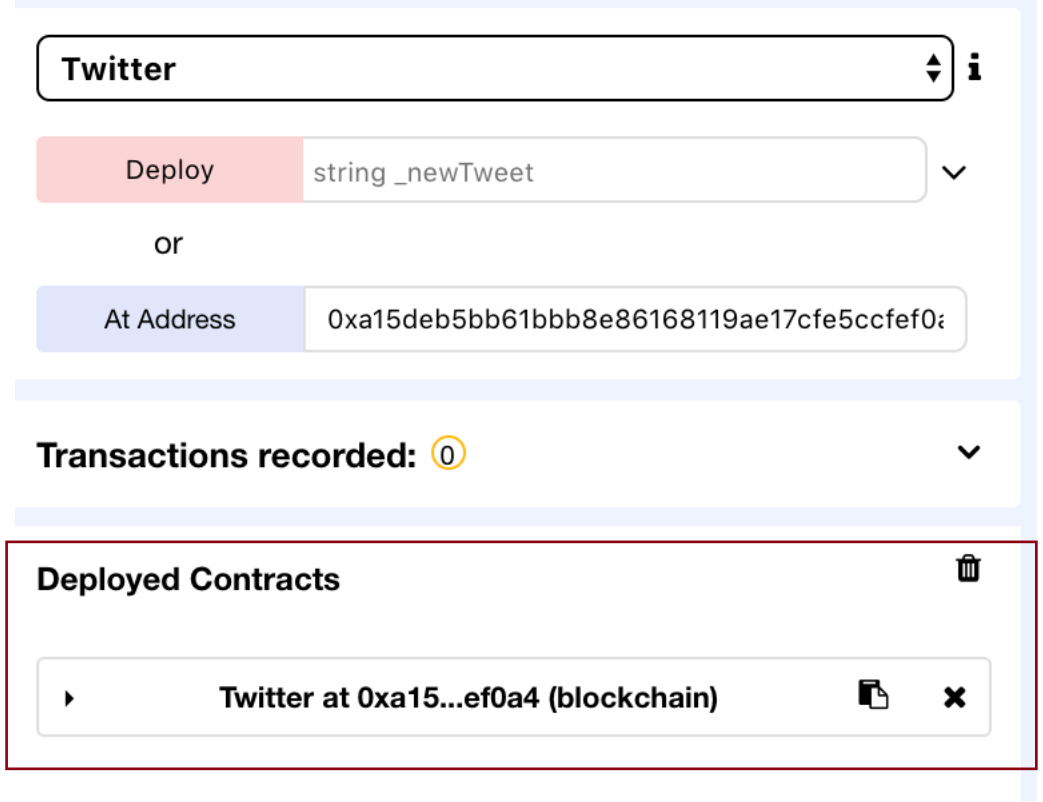
# Interact with contracts the easy way

- We have been using Remix thus far to experiment with a local node networks that existed in our browser. Remix can be used:
  - to interact with public test networks as well as the main net.
  - as a tool for interacting with deployed contracts.
- Select Environment -> Injected Web3
- This is basically an instance of Web3 that has been injected to the Webpage by the Metamask extension.
- It also has access to all of our Metamask Accounts



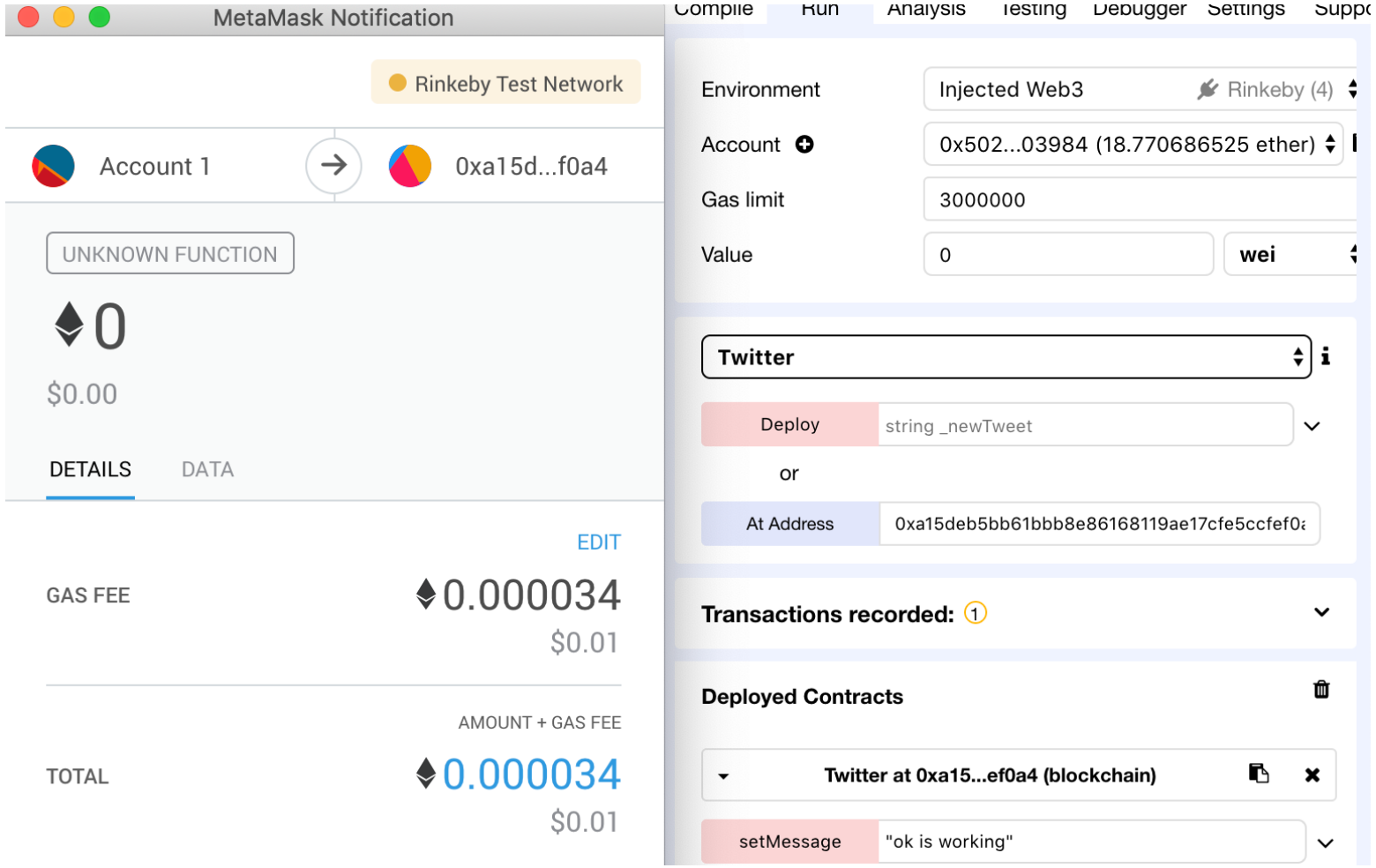
# Interact with contracts the easy way

- Copy the contract's deployed address as a result of the previous slide
  - https://rinkeby.etherscan.io/address/0xa15deb5bb61bbb8e86168119ae17cfe5ccfef0a4
- Go back to https://remix.ethereum.org
- Past the address of the previous Deployed contract into the "At Address" and then click the button.
- An instance of the Deployed contract will appear so that we can interact with it.
- However, any time we interact with the contract now we have to wait for a confirmation of the transaction :- )
- This is not instant ! We have to wait.



# Interact with contracts the easy way

- If you attempt to call the setMessage() this time.
- Metamask asks for a confirmation of a Transaction object that requires some amount of Ether to execute.
- Once Submitted a confirmation is created with a link to the actual Transaction
- https://rinkeby.etherscan.io/tx/0x4c461830863b0fb7382ad16443335aea52999c22ce3d94878ef022d7e9afd289



# Interact with contracts the easy way

Overview

## Transaction Information

[ This is a Rinkeby **Testnet** Transaction Only ]

| | |
|---|---|
| TxHash: | 0x4c461830863b0fb7382ad16443335aea52999c22ce3d94878ef022d7e9afd289 |
| TxReceipt Status: | Success |
| Block Height: | 3359534 (3 Block Confirmations) |
| TimeStamp: | 46 secs ago (Nov-18-2018 06:13:30 AM +UTC) |
| From: | 0x502579d20b2cf9dc4d4bbdb5c383743531003984 |
| To: | Contract 0xa15deb5bb61bbb8e86168119ae17cfe5ccfef0a4 |
| Value: | 0 Ether ($0.00) |
| Gas Limit: | 33549 |
| Gas Used By Transaction: | 33549 |
| Gas Price: | 0.000000001 Ether (1 Gwei) |

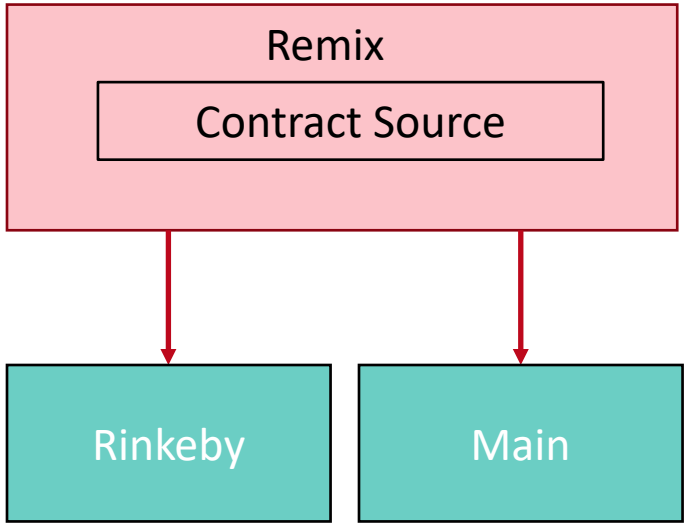
https://rinkeby.etherscan.io/address/0xa15deb5bb61bbb8e86168119ae17cfe5ccfef0a4

# Node JS - Versioning

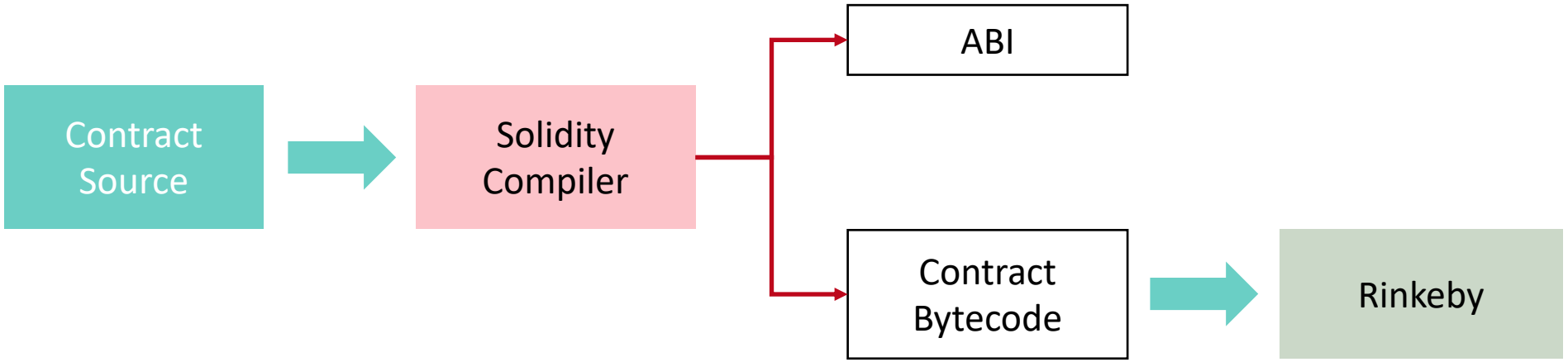
- For this Workshop we will need an up-to-date version of Node JS.
- Please take this opportunity to make sure you're running ***at least*** **version 8.0.0** of Node JS.
  - Check your current version by running the command `node -v` at your command line environment.
- If you are running an older version, you can easily update it by navigating to: https://nodejs.org/en/download/

# Our Simple Twitter Contract

```
1  pragma solidity ^0.4.25;
2
3  contract Twitter {
4
5      //Variables declaration
6      string public tweetMessage;
7
8      //constructor
9      constructor(string _newTweet) public {
10         tweetMessage = _newTweet;
11     }
12
13     //Methods
14     function setMessage(string _m) public {
15         tweetMessage = _m;
16     }
17
18     function getMessage() public view returns(string) {
19         return tweetMessage;
20     }
21
22 }//end contract
```

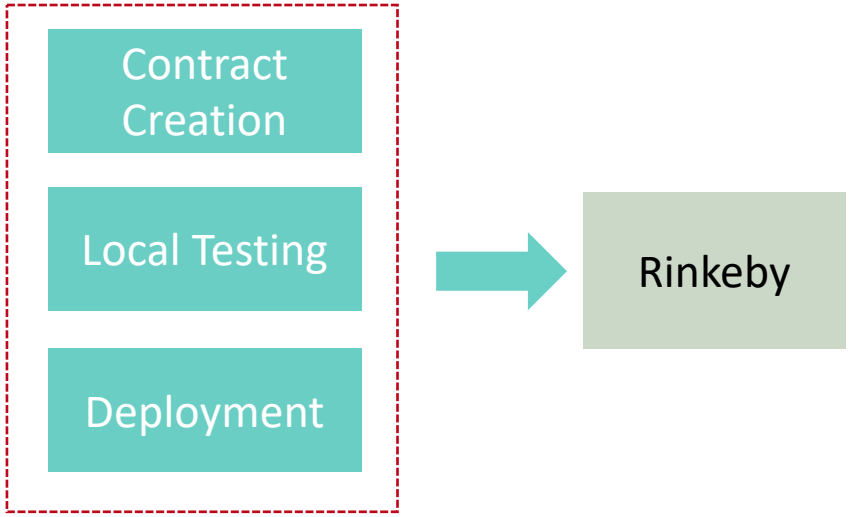


# Abstract Process for Deployment



# Journey to an actual Network

- However, how do things work behind the scenes.
- Let us setup a ground process for Deploying contracts based on a Custom Node.

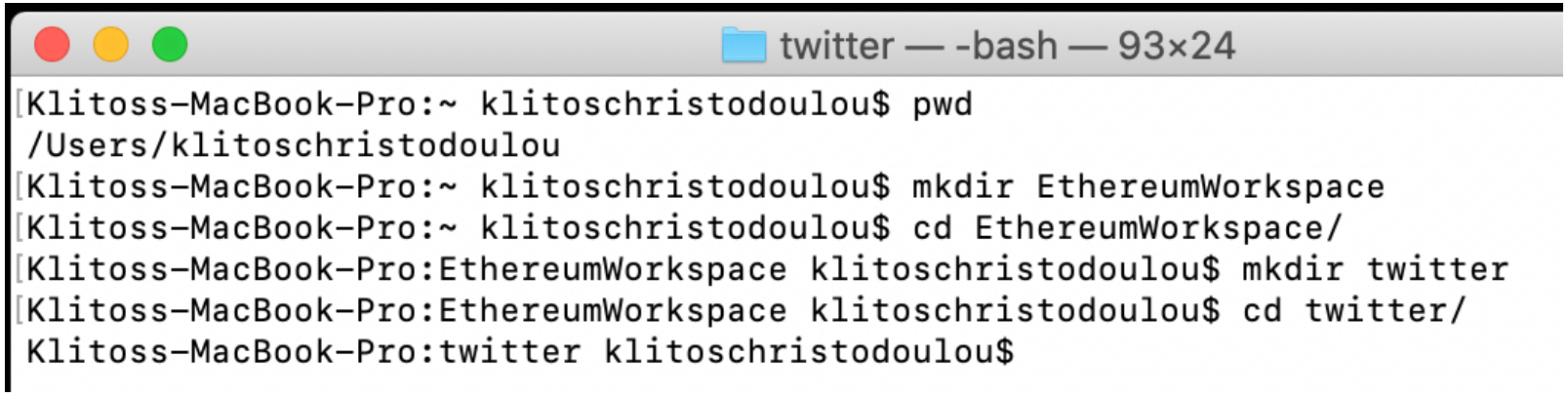



- Contract deployment based on a Custom Node.

# Journey to an actual Network

- Why are we deploying with our own Node?
  - We need to have control over the code using our own editor
  - We might need to setup version control for our project (especially for large scale project development – e.g., Git)
  - Get away from the Browser environment
- Remix IDE is great for quick development and Testing but we do not need to have any constraints thus we are creating our own process to Deployment using a Custom Node
- Other potential needs that we need to take into account:
  1. Write Solidity code that can be accessed from a Javascript like environment
     - ➢ Thus we need to setup the Solidity Compiler
  2. Rapidly test our Contracts
     - ➢ Setup a Custom Mocha Test Runner
  3. We need a way to deploy our Contracts to public Networks
     - ➢ Create a Script to both Compile & Deploy!

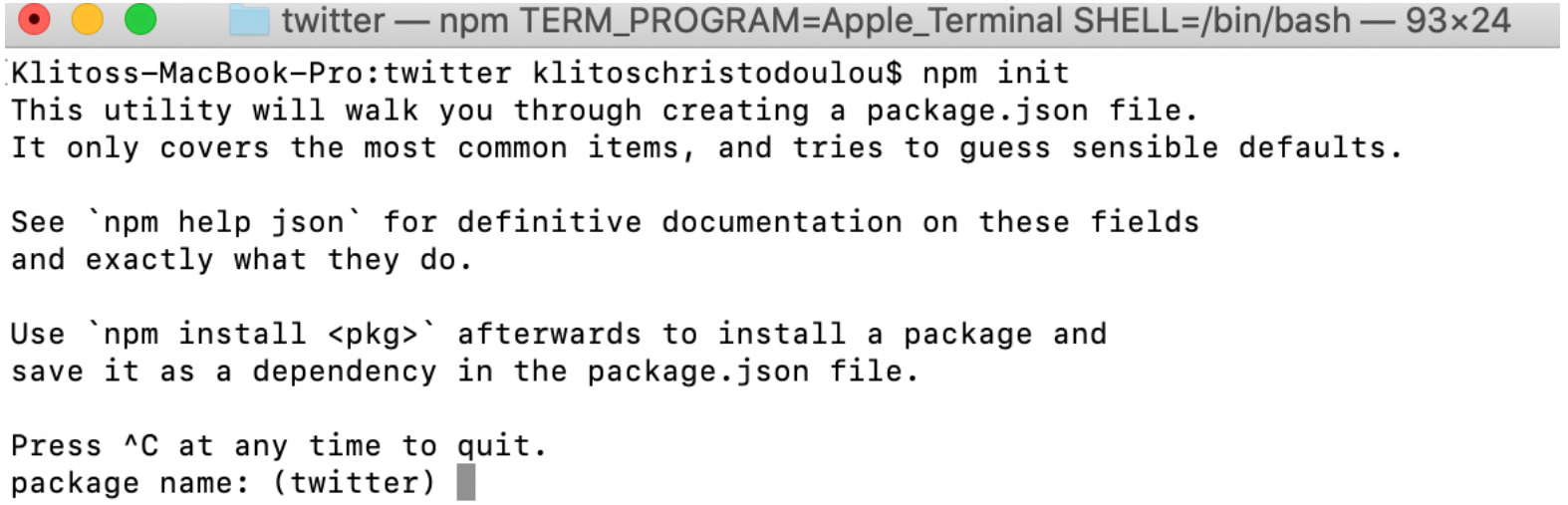
# Folders Setup




- On Windows:
  - Open your “Documents” folder
  - Create “New Folder” -> name it “EthereumWorkspace”
  - Inside the new dir create a subfolder that describes the name of our current project e.g., “twitter”
  - Switch to that directory

# Generate a new NodeJS Project

- Open the Terminal:
  - Make sure that you are on the “twitter” dir
  - Then generate a new NodeJS Project




  - Just Click Enter to proceed! Skip the warnings!

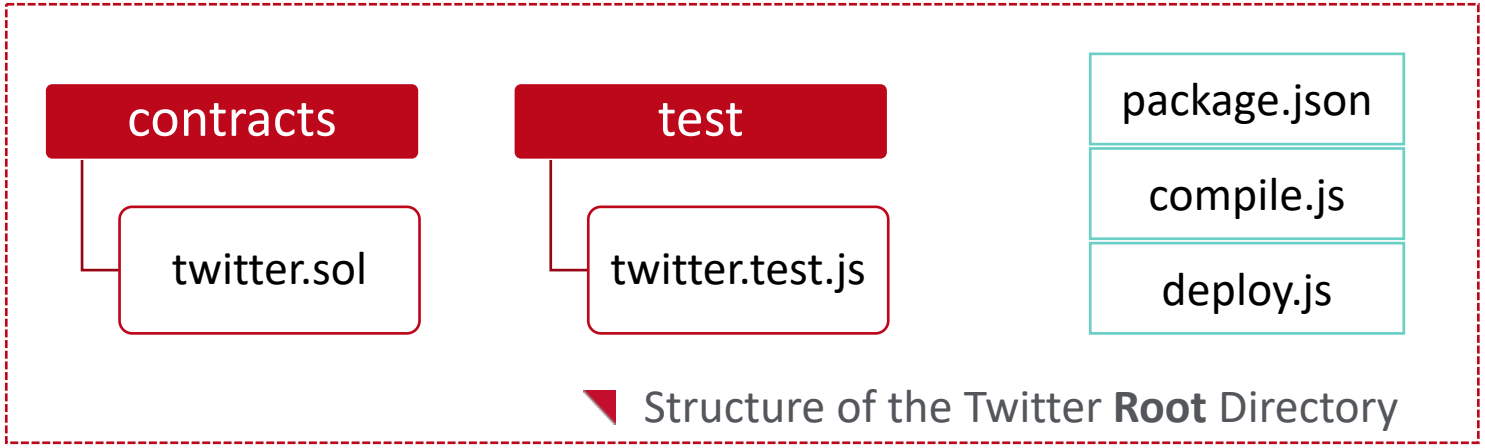
# Generate a new NodeJS Project


```
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /Users/klitoschristodoulou/EthereumWorkspace/twitter/package.json:

{
  "name": "twitter",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this OK? (yes)
Klitoss-MacBook-Pro:twitter klitoschristodoulou$
```

- A new .json file is created in our directory.

# Structure of the Twitter Dir

- With in the current folder, create two subfolders
  - mkdir contract
  - mkdir test




- Structure of the Twitter **Root** Directory

# Compiling Solidity Contracts

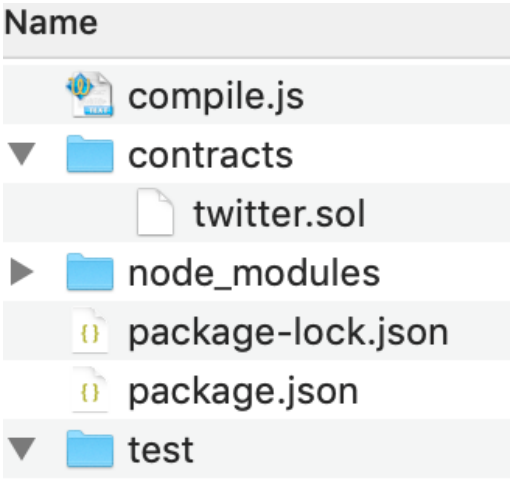
- Open any text editor you are familiar with
  - Navigate to the DecentralizedTrainingSeries and download

    https://github.com/DecentralizedTrainingSeries/contracts/blob/master/twitter.sol
  - Make sure that you save that .sol file in the "**Contracts**" directory
- Create a new file with the text editor and name it "**compile.js**"
  - We are working first with that file because both the Deploy and Testing processes are expecting that we will have generated the Bytecode for our contract
  - For this step we are going to make use of the **Solidity Compiler** :- )
    - **ABI:** is the Javascript interpretation layer of what the contract is.
    - **Contract Bytecode:** What we actually Deploy to the EVM.

# Install the Solidity Compiler

- To install the Solidity Compiler module
  - In terminal: `npm install --save solc`

```
twitter — -bash — 93×24
[Klitoss-MacBook-Pro:twitter klitoschristodoulou$ pwd
 /Users/klitoschristodoulou/EthereumWorkspace/twitter
[Klitoss-MacBook-Pro:twitter klitoschristodoulou$ ls -l
 total 8
 drwxr-xr-x  3 klitoschristodoulou  staff   96 17 Nov 13:13 contracts
 -rw-r--r--  1 klitoschristodoulou  staff  203 17 Nov 12:41 package.json
 drwxr-xr-x  2 klitoschristodoulou  staff   64 17 Nov 13:13 test
[Klitoss-MacBook-Pro:twitter klitoschristodoulou$ clear

 Klitoss-MacBook-Pro:twitter klitoschristodoulou$ npm install --save solc
```



# Compiling Solidity Contracts

- With the text editor open "**compile.js**"
- Start off by requiring two standard modules (we do not need to npm install them)
  - path – build a path to the .sol file (cross platform compatibility)
  - fs – to read the raw source code from the .sol file
- Type the following:

```
1   const path = require('path');
2   const fs = require('fs');
3   const solc = require('solc'); //Solidity compiler
4
5   //Create the path that points directly to our .sol
6   // arg1 = __dirname = set to the working dir
7   // arg2 = contracts folder
8   // arg3 = twitter.sol
9   const inboxPath = path.resolve(__dirname, 'contracts', 'Twitter.sol');
10
11  //Read the source code from our .sol contract using the fs module
12  // arg2 = specify the encoding
13  const source = fs.readFileSync(inboxPath, 'utf8');
```

# Compiling Solidity Contracts: Introduce solc

- With in the "**compile.js**" introduce the `solc` module

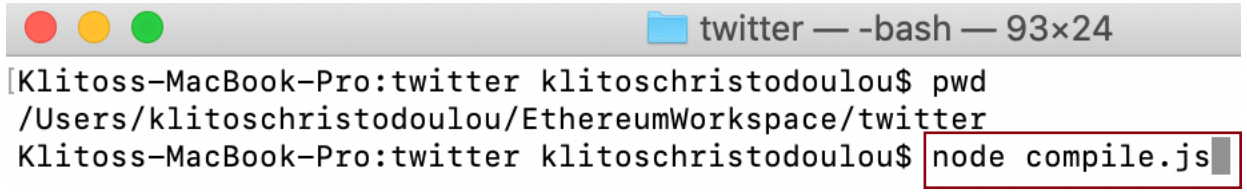
```
1  const path = require('path');
2  const fs = require('fs');
3  const solc = require('solc'); //Solidity compiler
4
5  //Create the path that points directly to our .sol
6  // arg1 = __dirname = set to the working dir
7  // arg2 = contracts folder
8  // arg3 = twitter.sol
9  const inboxPath = path.resolve(__dirname, 'contracts', 'Twitter.sol');
10
11 //Read the source code from our .sol contract using the fs module
12 // arg2 = specify the encoding
13 const source = fs.readFileSync(inboxPath, 'utf8');
14
15 //Write the compile statement
16 //arg1 = the raw source code
17 //arg2 = the number of contracts we would like to compile
18 module.exports = solc.compile(source, 1).contracts[':Twitter'];
```

Better to log out to the console what the compiler is doing:

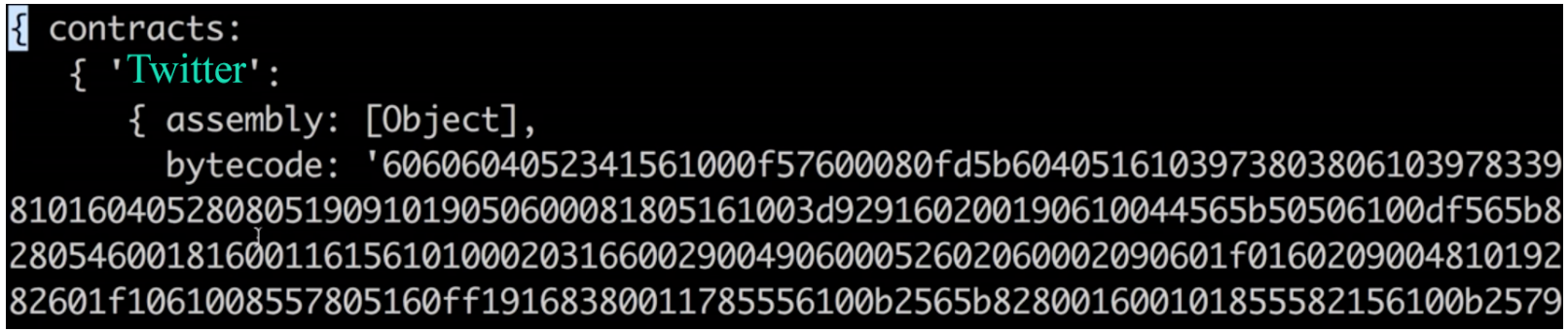
```
console.log(solc.compile(source, 1));
```

# Run the compile.js

- Run the **compile.js** through the terminal using the node command

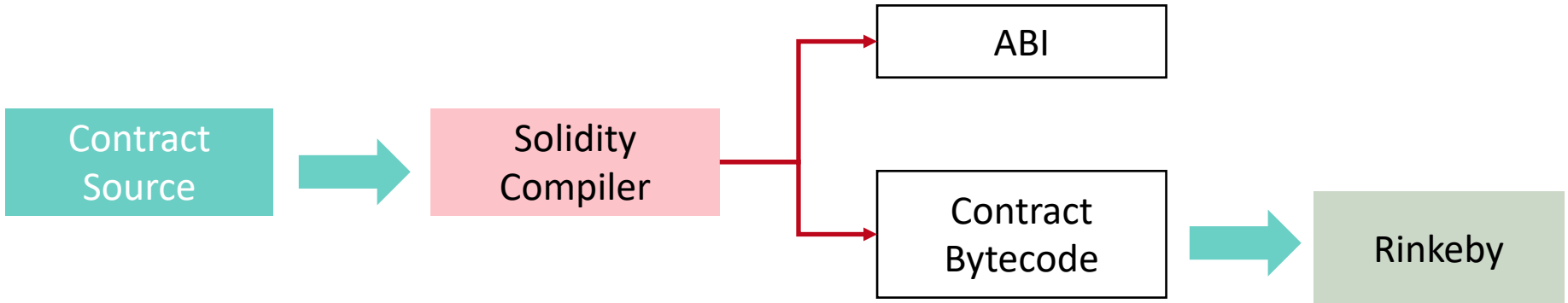



- If everything went well you will see something similar to the following :- )

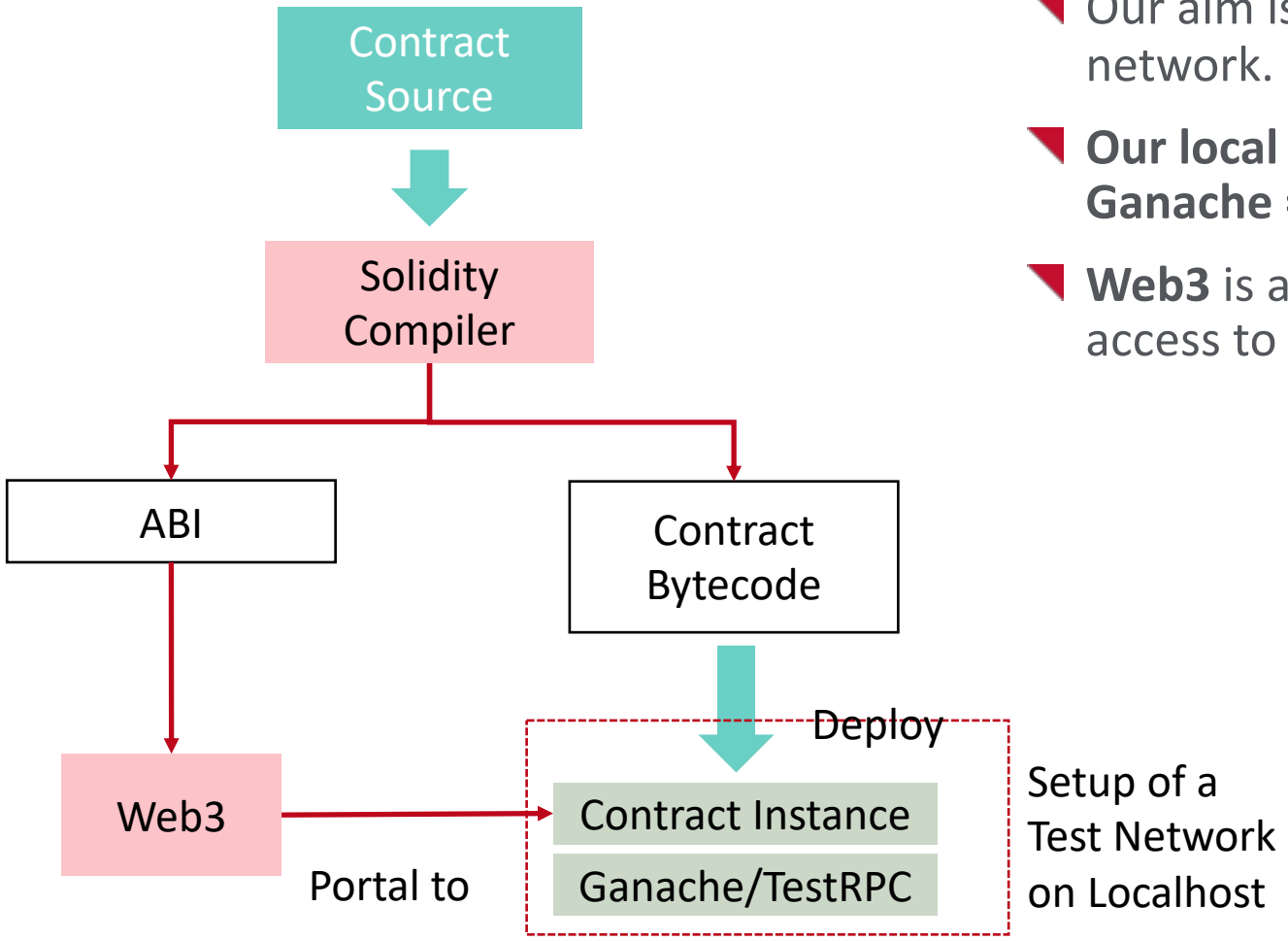


# Run the compile.js

- As you can observe from the previous compilation stage the outcome described **Bytecode**
- **Bytecode:** is the actual Bytecode that is interpretable by the Ethereum Network, the code that we will deploy to the Network.
- If you scroll down the actual outcome of the compile stage, you will locate another interesting property i.e., **interface**
- **Interface:** Describes our contract's **ABI**
  - **The communication layer between Solidity and JavaScript!**
  - **It is just a list of the Functions that exists in our contract (arguments, return values etc.)**



# Testing Setup for Solidity Contracts with NodeJS




- Our aim is to deploy the Bytecode to a local network.
- **Our local network will be created with the Ganache = TestRPC library.**
- **Web3** is a library we use to get programmatic access to a deployed contract (our portal).

# Installing Web3 & Ganache

- In our project directory
  1. Install out Mocha (Test Library) and test local network (Ganache)

```
$ npm install --save mocha ganache-cli
```

  2. Then, install web3

```
$ npm install --save web3@1.0.0-beta.35
```

- Test the Web3 version installed:
  - $npm ls web3

- Next, navigate to our “Test” folder within our project’s directory
  - Create a new file with the text editor name it “Twitter.test.js”
  - Or download from Git

<Set Github LInk>

# Required Modules for Testing

- assert -> is a build in library that is build on Node JS
  - Used to make assertions about tests
- Ganache -> we need to declare it on to be able to use our local network
- Web3 -> we will use the library to make instances of Web3 – calling the constructor
  - We will need it to create an instance of Web3
  - const web3 = new Web3(ganache.provider());

# Troubleshooting for NodeJS

- **On MacOS**
  - **Sudo access and then**
    - xcode-select --install
    - sudo xcodebuild -license accept
    - npm install -g npm@latest
- **Alternatives install solcjs globally**
  - npm install -g solc
  - **Test within terminal:** solcjs --help
- **Node Js – Issues with compile.js**
  - rm –rf node_modules (from the current folder)
  - npm install solc@0.4.19 (try to install an older version of the Compiler)

# Troubleshooting for NodeJS

- **On MacOS**

**Uninstall node and npm**

- Removed nvm

  $brew uninstall nvm

- Remove node,

$sudo rm –rf /usr/local/{lib/node{,/.npm,_modules},bin,share/man}/{npm*,node*,man1/node*}

- Try installing an older version of node@6.9.2

  $brew install node@6

# Troubleshooting for Web3

- **Web3**
  - **If you are using Windows, you need to install some dependencies as Admin:**
    - npm install --global --production windows-build-tools
  - **You should be fine with MacOS or Linux**

| | **Web3 Tip** |
|---|---|
|  | ▪ Make sure that you are using Web3 v.1.x.x or later |

# Deployment to a Public Test Network

Infura

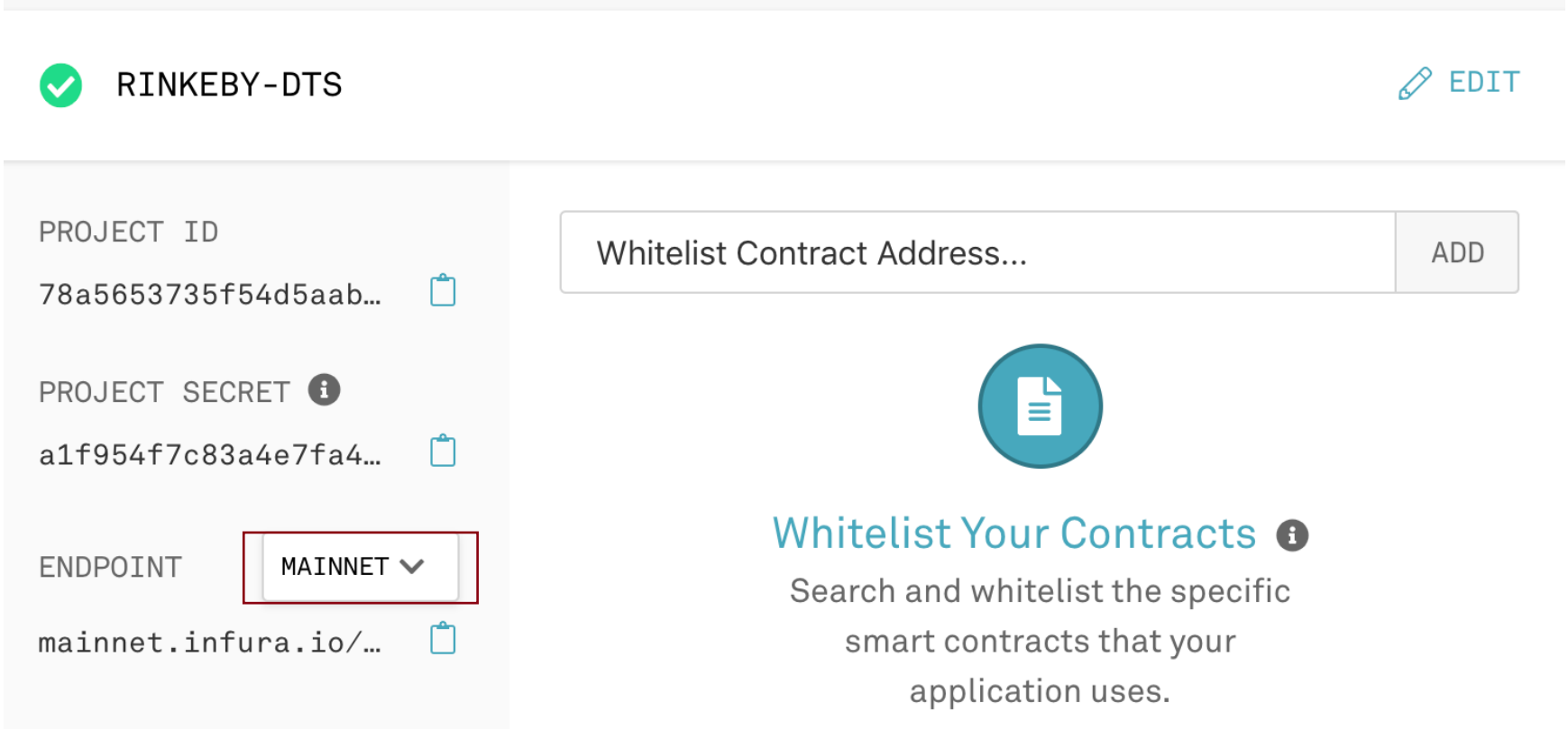
# Deployment with Infura

- Rather than using a local Ethereum Node on our local machine we are using a Node provided by a service provider to the Rinkeby test net.
- **Infura** is a public API that we can sign up and gives us access to a Node that is connected to the Rinkeby network by Infura
  - **Infura** is like our portal to the Rinkeby network
  - **Infura** has nodes to all the public Test Networks and the Main Network as well
  - It provides an easy way to deploy and accessing the Network
  - If we haven't been using **Infura** we would have to host our own Node on our local machine which includes has some complications
- To make our lives easier we are going to use **Infura** during the case-study as well.

# Sign up to the Infura API

- Navigate to https://infura.io/ and aign-up for an Account.
- Complete the form provided and then verify your email account that is linked with the Infura account.
  - Just remember your log-in details we are using them later to access to the key API needs to connect to our Infura account
- The Infura account has nothing to do with your real ETH wallet, just provides access to the tools and API for testing and deploying Smart Contracts
- You will be prompted to take a tour, please go ahead and SKIP that.
- Then click “CREATE NEW PROJECT” and enter any name, it does not have to relate to the name of the Contract, let as call it e.g., “Rinkeby-dts”

# Sign up to the Infura API




- Under the ENDPOINT drop-down choose the Rinkeby Network and copy the Link.

# Using NodeJS to create a provider to the Infura Net

- Navigate to our main project root directory
  - $npm install --save truffle-hdwallet-provider@0.0.3
- Ganache provides us some test Wallet accounts to do our experiments towards our local network
- For infura we need to create this provider account manually
  - Specify the network we are using
  - Specify the account we are unlocking
- Make sure you are on the main project root directory
  - Create a new script call it “deploy.js”
  - Download the “deploy.js” from Git and copy it in our project’s root directory

  https://github.com/DecentralizedTrainingSeries/contracts/blob/master/deploy.js
- Open up the terminal and run it: $node deploy.js

# Recap

- Remix is a great tool for quick deployment and experimentation.
- However, in a real word applications we usually tend to write the code for the contract quickly but spend time for testing and building the Web application that provides the portal for interacting with it.
  - Also, Remix does not provide any version control which might be important for larger scale projects (involving more people).
- So just to recap what we have seen:
  - We wrote our simple contract
  - Then we move into writing the “compile.js” script for compiling the contract to the actual bytecode, using the solidity compiler through nodejs.
  - We then started discussing on testing using Web3 (our portal) and Ganache (a local test network) using a contracts object
    - e.g., `twitter.methods.<method>.call() // for getter methods`
    - `twitter.methods.<method>.send() // for setter methods`

    https://github.com/DecentralizedTrainingSeries/contracts/blob/master/Twitter.test.js

# Recap

- We also experimented by creating a “deploy.js” script

  https://github.com/DecentralizedTrainingSeries/contracts/blob/master/deploy.js

- We have used the **HDWalletProvider** module to connect to some public Network and unlock an account so that we can use it on that Network
  - Remember we needed our seed from Metamask
  - Also we needed the link of the Infura node (our provider) - this is a requirement of the Web3
- The rest part of the “deploy.js” is similar to the previous test file we created.

- Good Luck with the Case Study!

# Troubleshooting for Infura

- Solving issues that may appear during the implementation of the “deploy.js”
  - npm uninstall ethereumjs-wallet
  - npm uninstall truffle-hdwallet-provider
  - npm install ethereumjs-wallet@0.6.0 --save
  - npm install --save truffle-hdwallet-provider@0.0.3

- Solving issues that might appear during Node install with python
  - npm config set python /usr/bin/python

# Decentralized Training Series Workshop: Certified Ethereum Specialist

19-20 November 2018, Athens, Greece

UNIC | Institute For the Future

CERTH CENTRE FOR RESEARCH & TECHNOLOGY HELLAS | iti Information Technologies Institute