Introduction to Truffle

What is Truffle?

- Truffle is a development environment, testing framework and asset pipeline for Ethereum
- Most widely used development framework in the Ethereum community
- Can use it to build and deploy DApps for testing purposes

Who should use it ?

Anyone who wants to dive into Ethereum development and needs a framework so they can better organize their DApp development assets and not have to worry about manually setting up a test environment.

Features of Truffle

- Built-in smart contract compilation, linking, deployment
- Automated contract testing for rapid development.
- Scriptable, extensible deployment & migrations framework.
- Network management for deploying to any number of public & private networks.
- Interactive console for direct contract communication.
- Configurable build pipeline with support for tight integration
- Automated contract testing with Mocha and Chai.

Installing Truffle

Requirements

1. NodeJS v10.x
2. Ubuntu 16.04

Truffle also requires that you have a running Ethereum client which supports the standard JSON RPC API

Install

```
npm install -g truffle
```

Creating a Project

- To use most Truffle commands, you need to run them against an existing Truffle project.

- We can create a bare project template OR can use Truffle Boxes, which are example applications and project templates.

- Check https://truffleframework.com/boxes for the available boxes.

- To start with, first create a directory for your project:

  - **mkdir Directory Name (Example: mkdir truffleDemo)**

  - **cd Directory Name (Example: cd truffleDemo)**

Creating a Project

- To download ("unbox") a box, use:
  - **truffle unbox <box-name>**
- To create a bare truffle project, use:
  - **truffle init**
- You can use optional
  - **--force** to initialize the project in the current directory even if it contains other files or directories
  - You should be careful while using this as this may potentially overwrite the contents of the directory.

Creating a Project

- Once this command has been completed, you will get a project structure with the following items:

- **contracts/**: Directory for Solidity contracts

- **migrations/**: Directory for scriptable deployment files

- **test/**: Directory for test files for testing your application and contracts

- **truffle-config.js**: Truffle configuration file

Creating Smart Contracts

- You can automate the creation of smart contracts using truffle.

  To create the smart contract, type and execute:

  truffle create contract <<ContractName>>

  Example: truffle create contract MyFirstContract

Compiling Smart Contracts

- All of your contracts are located in your project's contracts/ directory
- Truffle project (created through truffle init), you're given a single Migrations.sol
- If you're using a Truffle Box, you will have multiple files here.
- To compile a Truffle project, change to the root of the directory where the project is located and then type and run:
  - truffle compile

Compiling Smart Contracts

- When you first run compile command, all the smart contracts will be compiled.
- Upon subsequent runs, truffle will compile only those contracts which are modified since last compile.
- After successful compilation, the output / artifact will be created in the **build/contracts/** directory.

Warning: *These artifacts are integral to the inner workings of Truffle, and they play an important part in the successful deployment of your application. You should not edit these files as they'll be overwritten by contract compilation and deployment.*

Create Migration File

- We can create Migrations file using Truffle
- To create a migrations file, type and run the following:

truffle create migration <<ContractName>>

Example: truffle create migration MyFirstContract

Running Migrations

- Migrations are JavaScript files that help you deploy contracts to the Ethereum network.

- Responsible for staging the deployment tasks.

- As your project evolves, you'll create new migration scripts to further this evolution on the blockchain

To run the migrations, use the following command:

```
truffle migrate
```

Running Migrations

- truffle migrate will run all migrations located within your project's migrations directory

- If your migrations were previously run successfully, truffle migrate will start execution from the last migration that was run, running only newly created migrations

- If no new migrations exists, truffle migrate won't perform any action

- You can use the **--reset** option to run all your migrations from the beginning

Running Migrations

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

module.exports = function(deployer) {
  // deployment steps
  deployer.deploy(FirstContract);
};
```

- **artifacts.require()** - At the beginning of the migration, we tell Truffle which contracts we'd like to interact with via the artifacts.require() method
- The name specified should match the name of the contract definition within that source file
- Do not pass the name of the source file, as files can contain more than one contract.

Running Migrations

- **module.exports**

- All migrations must export a function via the module.exports syntax
- The function exported by each migration should accept a deployer object as its first parameter
- The deployer object is your main interface for staging deployment tasks
- Truffle requires you to have a Migrations contract in order to use the Migrations feature
- You will also receive this contract by default when creating a new project with truffle init.
- You must deploy this contract inside your first migration in order to take advantage of the Migrations feature

Running Migrations

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

module.exports = function(deployer) {
  // Deploy the Migrations contract as our only task
  deployer.deploy(Migrations);
};
```

deployer:

- Our migration files will use the deployer to stage deployment tasks

- We can write deployment tasks synchronously and they'll be executed in the correct order

Running Migrations

- Alternatively, each function on the deployer can be used as a Promise, to queue up deployment tasks that depend on the execution of the previous task

```
// Deploy A, then deploy B, passing in A's
newly deployed address

deployer.deploy(A).then(function() {
  return deployer.deploy(B, A.address);
});
```

Network Considerations

- It is possible to run deployment steps conditionally based on the network being deployed to

- To conditionally stage deployment steps, write your migrations so that they accept a second parameter, called network

```
module.exports = function(deployer,
network) {
  if (network == "live") {
    // Do something specific to the network
named "live".
  } else {
    // Perform a different step otherwise.
  }
}
```

Testing Contracts

- Truffle comes standard with an automated testing framework to make testing your contracts a breeze

- This framework lets you write simple and manageable tests in two different ways:

  (a) In **Javascript**, for exercising your contracts from the outside world, just like your application - using **Mocha** and **Chai**

  (b) In **Solidity**, for exercising your contracts in advanced, bare-to-the-metal scenarios.

Creating Test File

- Create a test file in Truffle:

truffle create test <<ContractName>>

Example: truffle create test <<MyFirstContract>>

Run Test

- All test files should be located in the **./test** directory.
- Truffle will only run test files with the following file extensions: **.js, .ts, .es, .es6**, and **.jsx**, and **.sol**
- All other files are ignored.
- To run all tests, simply run:

```
truffle test
```

- Alternatively, you can specify a path to a specific file you want to run

```
truffle test ./path/to/test/file.js
```

THANK YOU