AGENDA

- Introductions
- Session Goals
- Decentralized Applications (aka DApps) 101
- Introducing the Truffle Suite
- Exploring the DApp Development Lifecycle
- Hands On
- Next Steps & Q&A
ABOUT ME

- Head of Training & Ecosystem Engineering @ Truffle
- 12+ years in software development (across a number of paradigms)
- Decentralized FTW
ABOUT YOU

- Heard of Truffle before?
- Used any tools in the Truffle Suite?
- Used Metamask (or an equivalent wallet)?
- Deployed a smart contract to a public network?
SESSION GOALS

- Better understanding of the (decentralized) paradigm
- Appreciation of the development lifecycle
- Demystification of the tools used to build your own DApps
DECENTRALIZED APPLICATIONS 101
KEY CHARACTERISTICS

Written using **smart contracts** that store both the data and business logic 📜

Often have a **shared governance model** (via a DAO or equivalent) 📅

Applications that are "unstoppable and uncensorable" 💪

Can intrinsically store, or reference, tangible **digital assets** (no gatekeepers) 🡿

**TRUST**

Between zero / semi-trusted parties 🌟
DEVELOPMENT CONSIDERATIONS

- New things you’ll now need to consider...

1. USER IDENTITY
   - Fully "owned" by the user

2. INTERACTION / UX
   - Transaction fees, latency, etc.

3. GOVERNANCE
   - Upgradability, collectively owned by users
   - Vested in the service

4. SECURITY / AUDITS
   - Now a lot more at stake

5. COST OF STORAGE
   - More expensive, other options
APPLICATION ARCHITECTURE / WEB 2.0 VS WEB 3.0

Browser, Mobile App, etc

API

Business Logic (node.js, ruby, python, etc)

DB

DApp Browser

JSON-RPC

Web3.js

Business Logic (solidity)

Global State
SO WHAT CAN YOU BUILD?

- Types of apps and services we're seeing emerge (not definitive)...

Digital Assets, Exchanges, etc
- ERC20, ERC721, etc
- No middleman
- Digital twins

Decentralized Finance
- Transparent
- Programmable
- Accessible

Governance & Identity
- DAOs
- Immutable identity
- Government 2.0

New Ways of Operating
- Supply Chains
- Healthcare
- Gaming
- Efficiency between silos
INTRODUCING THE TRUFFLE SUITE
TRUFFLE SUITE OVERVIEW

“Gets developers from idea to dapp as comfortably as possible”
TRUFFLE SUITE OVERVIEW

- A **complete blockchain environment** (accounts, node / miners, programming interface) enabling you to model, build, iterate, etc.
- Over 7m aggregate downloads
- OSS @ [https://github.com/trufflesuite](https://github.com/trufflesuite)
TRUFFLE CLI

- Command-line tool that covers the full contract development lifecycle to make your life easier. Examples commands...

- "unbox" Download and setup a Truffle box
- "compile" Generate all the artifacts required to test, deploy, etc
- "test" Write and run tests to ensure code quality
- "debug" Step through the txns run against your contracts
- "migrate" Deploy contracts to any EVM-based network
• Zero-configuration local blockchain environment
• Built for development (workflow, testing, etc)
• Comes in a number of “delicious” flavors
• Enables the simulation of existing networks (via forking)
TEAMS

- Blockchain operations for everyone
- Built for open source and enterprise
- Features include ganache sandboxes, continuous integration, visual deployments, contract monitoring, visual debugging, etc
- Designed for the following
  - Developers
  - Operations Management
  - Systems Administrators
  - Product Managers
EXPLORING THE DAPP DEVELOPMENT LIFECYCLE
THE DECENTRALIZED DEVELOPMENT LIFECYCLE

1. Define Project Goals, Use Cases
2. Architectural / Technical Design
3. Development (and in-house testing, etc)
4. 3rd Party Audit
5. Bug Bounty (Gitcoin, HackerOne, etc)
6. Launch

Do I need a blockchain / decentralized ledger? 😐
VS MORE “TRADITIONAL” DEVELOPMENT?

● Potentially a lot at stake (e.g. assets of tangible / significant value) stored within the contracts

● While there are ways to update deployed contracts...
  ○ You have to have planned for this in advance (updatability patterns)
  ○ It might already be too late (depending on the vulnerability)

● Likely you’ll want to include some form of decentralized governance

● Testing and securing need to be factored in from the start
DEVELOPMENT (AND IN-HOUSE TESTING)

- Added emphasis placed on the following...
  - Documentation
  - Coverage
  - Analysis
  - Linting
  - Process (e.g. freeze before audit)
- Plenty of tools to assist with the above
  - E.g. Truffle, EthLint, MythX, Slither, Manticore
- Wide range of existing 3rd party libraries and frameworks that can (and should) be leveraged...e.g. OpenZeppelin
3RD PARTY LIBRARY EXAMPLE: OPENZEPPELIN

- Open: [https://openzeppelin.com/contracts](https://openzeppelin.com/contracts)
- "OpenZeppelin Contracts helps you minimize risk by using battle-tested libraries of smart contracts for Ethereum and other blockchains. It includes the most used implementations of ERC standards."
- Benefits...save you re-inventing the wheel and...
  - Emphasis on security
  - Modular
  - Strong community
- Contracts types include access control, tokens, crowdsales, utilities, math, payments, cryptography, etc
SECURITY AUDITS

- Systematic assessment of your code’s security, safety, etc (*with a particular emphasis on identifying subtle vulnerabilities*)
- Do you need one?
  - What’s at risk?
  - Relative complexity of code?
  - Ease of recovery from an incident?
- Also an opportunity for the team to...
  - Learn from experts
  - Identify gaps in process
- Phases
  - Initial audit (1-4 weeks)
  - Mitigations (2-3 weeks)
“Bounties are offered to developers in exchange for their expertise in resolving bugs and disclosing security vulnerabilities.”

- Popular Bug Bounty platforms...
  - HackerOne
  - Gitcoin

- Submit code to repo / deployed contracts to a Testnet
HANDS ON
HANDS ON

- Installation
- Hello World
- Truffle Boxes
- MetaCoin
HANDS ON - HELLO WORLD

● What?
  ○ A simple Dapp that facilitates the storage of a string (“Hello World”) on-chain
  ○ Subsequent retrieval (setter) and updating (getter) of that string

● Note
  ○ Slides are provided if you want to follow along after the lecture
HANDS ON - INSTALLING TRUFFLE CLI
**EXERCISE - INSTALLING TRUFFLE CLI**

- Install Node.js ([https://nodejs.org](https://nodejs.org))
  - Install using NVM if you want to switch Node.js version
  - Ideally we want lts/dubnium (although lts/erbium should now be all good too)
- Install Truffle globally using NPM

```
> npm install -g truffle
...
> truffle version
```
HANDS ON - DAPP HELLO WORLD WITH TRUFFLE
EXERCISE - CREATE A PROJECT

- Create and enter your project directory
  
  > mkdir truffle-hello-world  
  > cd truffle-hello-world

- Tell truffle to initialize the project directory
  
  > truffle init

- Note that there are lots more templates / scaffolds that we’ll be exploring later...
EXERCISE - TRUFFLE PROJECT STRUCTURE

- Open the project in your IDE (Code, Atom, Sublime, etc) and you should see following:
EXERCISE - CREATING A NEW CONTRACT

- Contracts are created / stored in the `contracts` directory
- Create a new contract via the following command (or via the IDE):

  ```
  > truffle create contract HelloWorld
  ```

- This will also scaffold a basic Solidity contract with a constructor:

  ```
  > cat contracts/HelloWorld.sol
  ```
EXERCISE - COMPILING CONTRACTS

● In the **contracts** directory paste the contents of the following:
  ○ [https://pastebin.com/ziEfNLnA](https://pastebin.com/ziEfNLnA)

```
//SPDX-License-Identifier: MIT
pragma solidity >= 0.5.0 < 0.7.0;

contract HelloWorld {
    string public x;

    function setX(string memory newX) public {
        x = newX;
    }

    function getX() public view returns (string memory) {
        return x;
    }
}
```
**EXERCISE - COMPILe THE CONTRACT**

- When you are ready to build your contracts run:

  ```
  > truffle compile
  ```

- Note that if you see an error related to a mismatch in compiler version we can specify the appropriate version in the `compilers` section of `truffle-config.js` and Truffle will pull down the correct version.

- Truffle will compile your contracts and create contract artifacts in the `build/contracts` directory.

- These artifact files will be used later to make it easy to programmatically interact with your contracts.
EXERCISE - DEPLOYING & INTERACTING WITH YOUR CONTRACTS
EXERCISE - USING TRUFFLE DEVELOP

- Truffle has a built-in personal blockchain (based on ganache) that can be used for testing
- Note that it's completely local to your system and does not interact with any public Ethereum network
- Accessed via the following command:

```bash
> truffle develop
```
EXERCISE - USING TRUFFLE DEVELOP

- Creates ten temporary accounts (and their associated private keys) that can be used when interacting with the blockchain...

Accounts:
(0) 0x8128880dc48cde7e471ef6b99d3877357bb3f01
(1) 0x12b6971f6eb35dd138a03bd6cbdf9fc9b9a87d7e
(2) 0xe17634217e02b89552765bed11661c666e8d7a11
(3) 0x15b309b5fbc634afbof61f065f4fbbf82aba203
(4) 0x036548fd3a6d2d38a5d72eb2f689d3e053c00d9
(5) 0x9dc4c654f382c2716288caalbfbc0cb96077855
(6) 0x4ea836185f15eb492647a2e611abe9ba4c62f9e
(7) 0xda2c638069e6b761d7e1ab6880c18875bdcfbc1
(8) 0x6a3f70f2100fb84fd2b0a3767469eba4247a3d7c
(9) 0x22e416e72c1ac78f55dee28a48f9437f05ea68eb
EXERCISE - DEPLOYING YOUR CONTRACTS

- Write migration files and place them in the `migrations` directory
- Modify `truffle-config.js` to include the configuration for the network to which you want to deploy (note that `truffle develop` will automatically detect)
- Initiate the migration with the following command:

```bash
truffle(develop)> migrate
```
EXERCISE - ADDING A MIGRATION SCRIPT

- At the moment we’re only migrating the Migrations contract.
- To also migrate `HelloWorld`, we’ll need to add an additional script.
- In the `migrations` directory, create the following: `2Deploy_contracts.js`
- Copy and paste the contents of `1_initial_migration.js` and specify the `HelloWorld` contract as follows:

```javascript
var HelloWorld = artifacts.require("./HelloWorld.sol");

module.exports = function(deployer) {
  deployer.deploy(HelloWorld);
};
```
EXERCISE - INTERACTING WITH THE CONTRACT

- There’s a LOT that can be done from the console (as it mounts a web3.js instance), but for now, we’ll just use it to interact with our `HelloWorld` contract.
- Try the following:

```javascript
truffle(develop)> let instance = await HelloWorld.deployed()
truffle(develop)> instance.setX('Hello World')
truffle(develop)> instance.getX()
```
EXERCISE - INTERACTING WITH THE CONTRACT

- Try updating the contract return string...”Hello <Your Name>”
- Migrate again...what happens?
- You’ll need to use a --reset to force an update

```
truffle(develop)> migrate --reset
```

- Note that we’ll be addressing development workflow / lifecycle in the next class(es)
TRUFFLE BOXES
TRUFFLE BOXES - OVERVIEW🎁

- Boilerplates for both learning and kick starting new projects (e.g. sample contracts, front-ends, complete sample DApps)
- 3 flavors...
  - Official
  - Partner
  - Community
- Full list at [https://www.trufflesuite.com/boxes](https://www.trufflesuite.com/boxes)
- Moving towards a monthly release cadence (Aave, RSK, etc)

> truffle unbox <box-name>
TRUFFLE BOXES - THEMES

- **Getting Started**
  - MetaCoin
  - Drizzle

- **Tokenization**
  - Etherplate
  - Cheshire
  - TutorialToken

- **Front-end focused**
  - React
  - Drizzle-vue-box
  - AngularTruffleDApp
HANDS ON - TRUFFLE TEAMS + METACOIN
EXERCISE - METACOIN BOX

● Unbox the Truffle Metacoin Box
● Review in the context of Truffle Teams...
  ○ Build
  ○ Sandbox
  ○ Deploy
  ○ Send a Transaction
  ○ Debug
SUMMARY & NEXT STEPS
SUMMARY

- Explore Decentralized Applications (aka DApps) through a developer lens
- Introduced the Truffle Suite
- Explored the DApp Development Lifecycle
- Got a little Hands On
NEXT STEPS

● Get in touch :) Questions, feedback, slides, etc...
  ○ kevin@trufflesuite.com

● Contributing...
  ○ https://github.com/trufflesuite

● TruffleCon 2020
  ○ https://www.trufflesuite.com/trufflecon2020
Q&A