Лабораторная работа 2

28 октября 2018 г.

“Сначала учите науку программирования и всю теорию. Далее выработаете свой программистский стиль. Затем забудьте все и просто программируйте. — George Carrette”

1 Введение

Цель: изучить работу Ethereum, изучить Ganache

2 Основные понятия

- Etheruim (эфир) - платформа для создания децентрализованных онлайн-сервисов на базе блокчейна
- Smart Contract (умный контракт) - компьютерный алгоритм, предназначенный для заключения и поддержания коммерческих контрактов в технологии блокчейн.

Для выполнения лабораторной работы, необходимо воспользоватся виртуальной машиной, на которой установлена Ubuntu 18.04 и весь необходимый набор ПО.

Теорию можно прочесть в файле Theory.pdf.

(Тут будет инфа о пути к тестовому проекту)

3 Подготовка окружения

Все необходимые зависимости уже установлены на виртуальную машину. Далее будут представлены необходимые программы и пакеты для выполнения лабораторной работы.

3.1 Ganache

Ganache - это блокчейн для разработки Ethereum, которую вы можете использовать для развертывания контрактов, разработки приложений и запуска тестов. Он создает виртуальную цепочку Ethereum и генерирует некоторые поддельные учетные записи, которые мы будем использовать во время разработки (Рис. 1).

Для запуска Ganache, найдите значек в меню приложений. Сразу после запука будут установлены значения по умолчанию, контролирующие количество поддельных учетных записей и сумму ETH на их счетах.

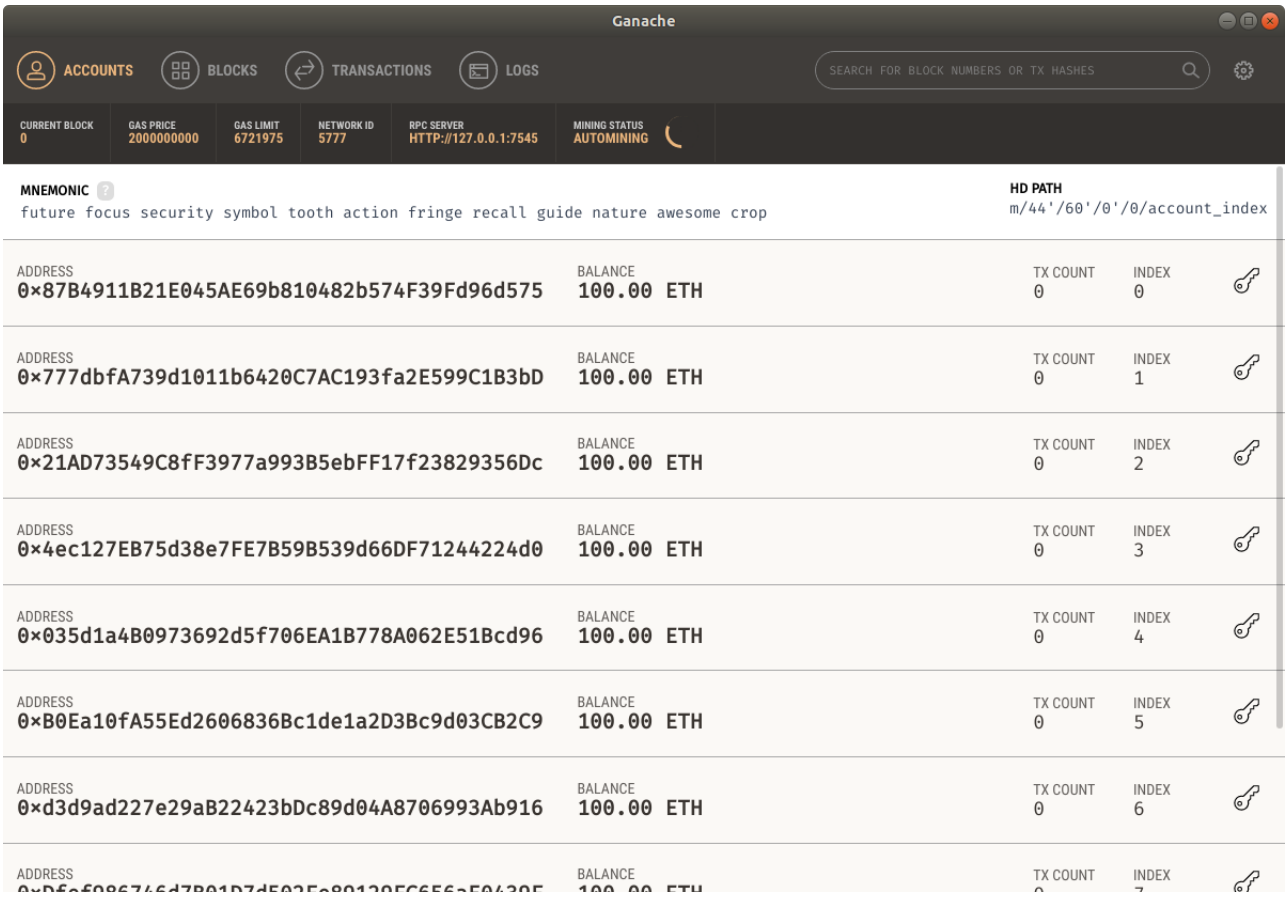

Рис. 1: Интерфейс Ganache.

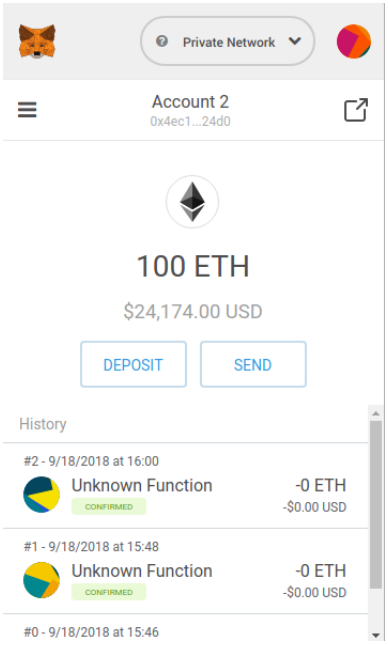

Рис. 2: Интерфейс MetaMask.

3.2 Metamask

Metamask - это криптовалютный кошелек, который встраивается в браузер Google Chrome и он нужен для упрощения передачи Эфира (Эфириум, Ethereum, ETH) или токенов ERC-20 в сети Эфириума. Мы будем использовать его для тестирования смарт-контрактов (Рис. 2).

3.3 Remix IDE

Remix - это мощный инструмент с открытым исходным кодом, который помогает писать контракты Solidity прямо из браузера. Написанна на Javascript, Remix поддерживает как использование в браузере, так и локально (Рис. 3). Remix также поддерживает тестирование, отладку и развертывание смарт-контрактов и многое другое.

На данный момент, она в некоторх моментах подтармаживает, и при работе на вирутальной машине, это вызывает дискомфорт, по этому советуется пользоватся VSCode, которая предустановлена на ВМ.

4 Создание проекта

Для выполнения лабораторной работы, воспользуемся уже готовым проектом, в нем будут необходимые файлы для фронтенда и т.п. Для скачивания тестового проекта, перейдите в каталог, куда будет скачан проект и выполнить:

```
1 $ truffle unbox pet-shop
```

Листинг 1: Скачивание проекта pet-shop

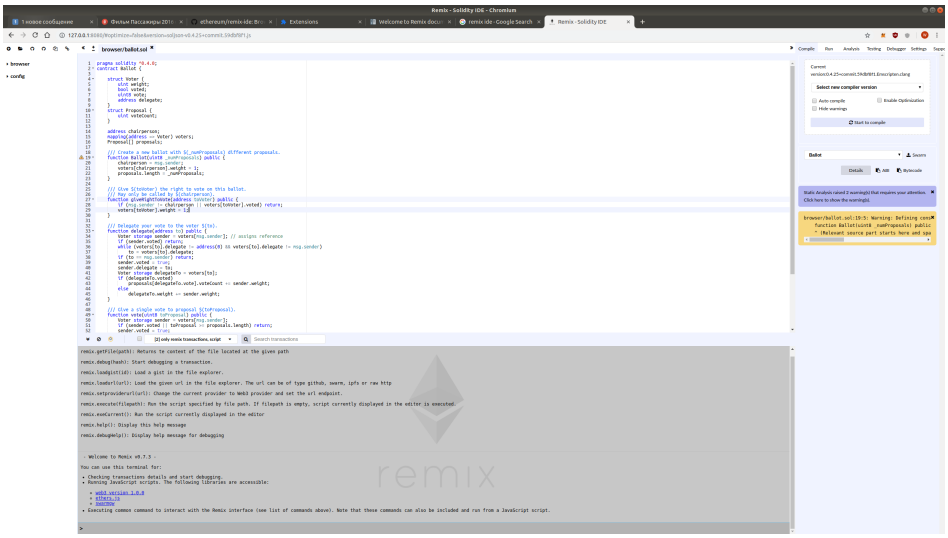

Рис. 3: Интерфейс Remix IDE.

После выполнения данной команды будет скачан проект в виде следующих файлов:

- contracts: Смарт-контракты. В папке уже есть контракт для миграций в блокчейн.
- migrations: Миграции. Эти миграции похожи на используемые в web фромеворках для изменения состояния базы данных.
- node modules: зависимости NodeJS.
- src: клиентские приложения.
- test: папка с тестами.
- truffle.js: файл с конфигурацией для truffle

4.1 Создание первого смарт-контракта

Для создания смарт-контракта необходимо создать файл с расширением .sol в папке contracts. Пример:

```
pragma solidity 0.4.2;
contract Election {
  // Read/write candidate
  string public candidate;
  // Constructor
  function Election () public {
    candidate = "Candidate 1";
  }
}
```

Листинг 2: contracts/Election.sol

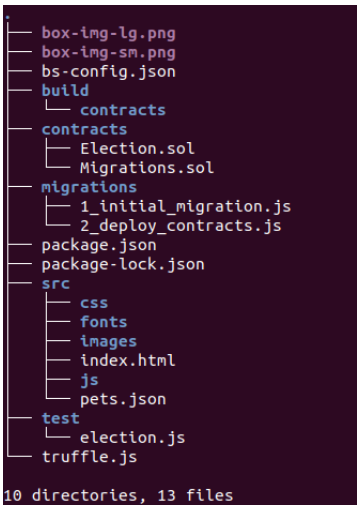

Рис. 4: Дерево проекта.

Файл начинается с объявления версии solidity с помощью pragma. Затем объявляем смарт-контракт с ключевым словом contract, за которым следует название контракта. Затем мы объявляем переменную состояния, которая будет хранить значение имени кандидата. Переменные состояния позволяют нам записывать данные в блокчейн. Мы объявили, что эта переменная будет строкой, и мы установили ее видимость public. Поскольку она является public, solidity генерирует функцию getter, которая позволит получить доступ к этой переменной за пределами контракта.

Затем мы создаем функцию-конструктор, которая будет вызываться при каждом разворачивании смарт-контракта в цепочку. Здесь мы установим значение переменной состояния кандидата, которая будет сохранена в блокчейне после миграции. Обратите внимание, что функция конструктора имеет то же имя, что и смарт-контракт. Так solidity знает, что функция является конструктором.

Теперь, когда мы создали основу для смарт-контракта, давайте посмотрим, можно ли его развернуть в блокчейн. Для этого нам нужно создать новый файл в каталоге миграции. Из корня проекта создайте новый файл из командной строки следующим образом:

```
1 var Election = artifacts.require("./Election.sol");
2
3   module.exports = function(deployer) {
4   deployer.deploy(Election);
5 };
```

Листинг 3: contracts/Election.sol

После добавления новой миграции, ее нужно применить выполнив:

```
1 $ truffle migrate
```

В консоли должно быть что то подобное:

```
1 Compiling ./contracts/Election.sol...
```

```
Compiling ./contracts/Migrations.sol...
^----------------------^

Writing artifacts to ./build/contracts

Using network 'development'.

Running migration: 1_initial_migration.js
Deploying Migrations...
... 0x8c07a54c548ed49128b8fb2d17fa071b118019b6590536efd1fc2f0d49d3a850
Migrations: 0x24a632b41f13d45b189670ed3730ef610cb2bd29
Saving successful migration to network...
... 0x8c3023da63417b2debfcdeedefb78ca379aced324ac8d688335f7ce0dcfd37bc
Saving artifacts...
Running migration: 2_deploy_contracts.js
Deploying Election...
... 0xd53622e3735d4284f19b2b751913f251983e3f4163abb35daf1bb30a199c17df
Election: 0x850643f300b2516518d4c7b7f093e107957fcc17
Saving successful migration to network...
... 0xbafe0c2afb2fe2639500e8ba03e7b2fd13cda795cdabdad8f585bef2ab14cf59
Saving artifacts...
```

Можно протестировать в консоли, как работает наш модуль:

```
$ truffle console
```

Создадим инстанс смарт-контракта и попробуем считать имя первого кандидата:

```
>> Election.deployed().then(function(instance) { app = instance })
```

Сдесь Election это имя переменной которая была создана в файле миграций. Мы получили инстанс контракта из фунции deploted().

После этого, можно считать переменные:

```
>> app.candidate()
  // => 'Candidate 1'
```

Вот и все, первый смарт-контракт готов!) Congratulations!

Список литературы

[1] Как работает Эфириум (Ethereum)? https://habr.com/post/407583/