ETSIT UPM

dit UPM

Blockchain: Desarrollo de Aplicaciones

Truffle - Introducción

BCDA 2018

Versión: 2018-10-18

Índice

- Introducción, documentación.
- Ganache
- Caso de Estudio #1: Smart Contract Contador
  - Crear un proyecto que usa un Smart Contract llamado Contador.
  - Usando Truffle, crear pruebas, scripts, cmds y una dapp.
- Caso de Estudio #2: Webpack
  - Añadir Webpack al caso de estudio #1
- Caso de Estudio #3: Introducción a React y Redux
  - Crear un ejemplo de una SPA con un nuevo contador.
    - El contador es diferente al de los casos de estudio anteriores: Sin Blockchain.
    - Partiendo de la configuración de Webpack del caso de estudio #2 y ampliarla para soportar react.
- Caso de Estudio #4: Smart Contract Contador con React y Redux. Y SIN Drizzle.
  - Unir todo lo visto en los casos de estudio anteriores.
- Caso de Estudio #5: SC Contador con React, Redux y CON Drizzle.
  - Añadir Drizzle al caso de estudio #4.

Introducción

- ¿Qué es Truffle?

  `https://truffleframework.com`

  `https://github.com/trufflesuite/truffle.git`

  - Entorno y librerías para desarrollar y usar contratos inteligentes:
    - compilar, desplegar, gestionar las redes, migraciones, pruebas, creación de scripts, interaccionar con las instancias desplegadas, etc...

Documentación

- Consultar la página de documentación para ver los detalles sobre Truffle

  `https://trufflefframework.com/docs`

  - Instalación

    ```
    $ npm install -g truffle
    ```

  - Crear un proyecto desde cero

    ```
    $ truffle init
    ```

    - Se crean los ficheros y directorios del proyecto.
      - `contracts/`, `migrations/`, `tests/`, `truffle.js`

  - Descargar un **box** (ejemplos y plantillas ya hechos)

    ```
    $ truffle unbox <nombre>
    ```

- Editar truffle.js para configurar las redes sobre las que se desplegarán los contratos.

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 7545,
      network_id: "*"
    }
  }
};
```

- Comandos
  - Compilar

    ```
    $ truffle compile
    ```

  - Migrar

    ```
    $ truffle migrate
    ```

  - Probar

    ```
    $ truffle test
    ```

  - Depurar

    ```
    $ truffle debug <transaccion_hash>
    ```

  - Ejecutar scripts

    ```
    $ truffle exec <path_fichero.js>
    ```

  - Lanzar un interprete Javascript:

    ```
    $ truffle console
    ```

    - podemos ejecutar los mismos comandos de truffle pero sin anteponer truffle:

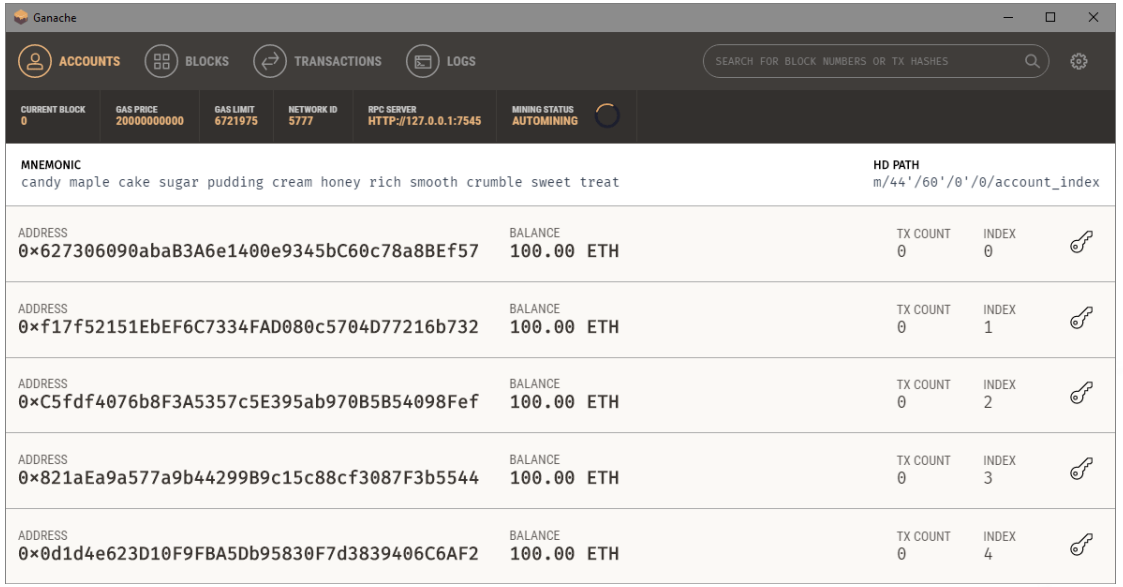
  - ...
    - Consultar la página **Command Reference** para ver todos los comandos disponibles y sus opciones.

Ganache

- Red Ethereum personal para desarrollo.
  - Generación de bloques instantáneo y no requiere gastar ethers.
  - Documentación:

    `https://trufflframework.com/docs/ganache/using`

  - Instalación:
    - **Ganache**
      - interfaz gráfica .
    - **ganache-cli**
      - línea de comandos.



- Existe un entorno de desarrollo que lanza internamente una red blockchain con la que interaccionaremos.
  - Se lanza el comando:

    ```
    $ truffle develop
    ```

  - La red blockchain interna crea automáticamente varias cuentas, con saldo, ...
  - Lanza un interprete de Javascript que ofrece un prompt para introducir comandos.
    - Y podemos invocar los mismos comandos de truffle ya vistos pero sin llamar a truffle.

- Demo: Ejemplo de uso del contrato Contador con truffle develop (igual con truffle console):
  - Ir al directorio donde está el proyecto truffle del caso de estudio #1
  - Ejecutar:

```
$ truffle develop
```

  - Invocar los siguiente comandos:

```
truffle(develop)> compile
truffle(develop)> migrate
truffle(develop)> migrate --reset --compile-all
truffle(develop)> Contador.deployed() ↩
    .then(instance => {↩
              instance.valor.call() ↩
              .then(v => console.log(v.toNumber()))
    })
truffle(develop)> Contador.deployed() ↩
    .then(instance => { ↩
        instance.incr().then(result => console.log(result))
    })
truffle(develop)> Contador.deployed() ↩
    .then(instance => {↩
              instance.valor() ↩
              .then(v => console.log(v.toNumber()))
    })
```

```
Contador.deployed().then(instance => {instance.valor.call().then(v => console.log(v.toNumber()))})
Contador.deployed().then(instance => {instance.incr().then(result => console.log(result))})
```

Artefactos

- Al compilar los contratos se crean unos ficheros JSON llamados artefactos que se guardan en el directorio `./build/contracts`.
- Los artefactos son ficheros que necesita Truffle para realizar el despliegue de los contratos en la red especificada.
  - La opción `--network` permite indicar la red con la que se desea trabajar.
- Cuando se ejecutan migraciones contra una determinada red, estos ficheros .json se actualizan con la información del despliegue en la red indicada.
- Cuando una aplicación carga esos artefactos, estos detectan a que red está conectado el cliente Ethereum para usar las instancias de los contratos adecuadas.