

Testing Smart Contracts with **Waffle**



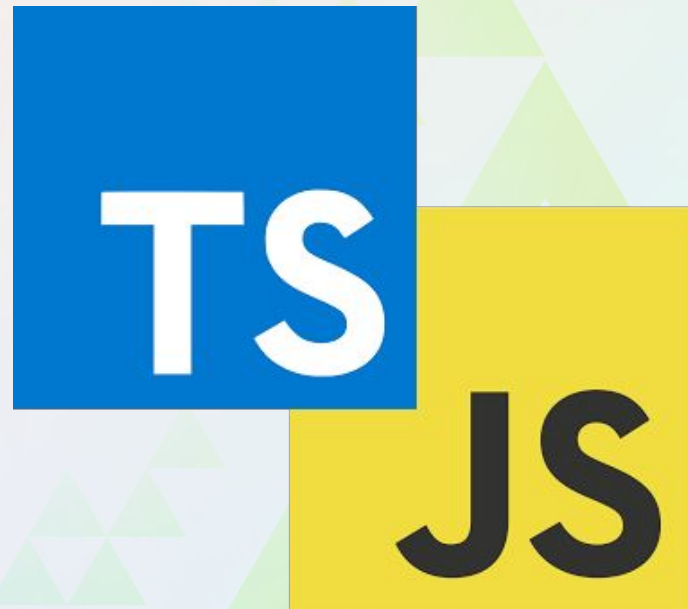


Let's download the repo

tinyurl.com/eth-waffle

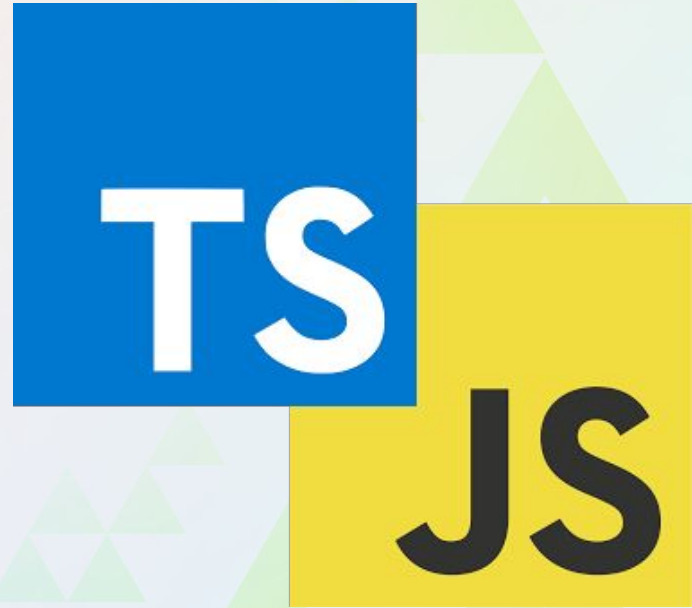


Two ways to test a smart contract...



Two ways to test a smart contract...

- Easy and intuitive
- Extremely flexible
- DApp native



Waffle's qualities

What makes Waffle **sweet & simple**?

- Minimalistic approach 🌿
- Blazing fast 🔥
- Friendly syntax 🐙
- Open source 🪨

Waffle's qualities

What makes Waffle **sweet & simple**?

- Minimalistic approach 🌀
- Blazing fast 🔥
- Friendly syntax 🐙
- Open source 🪨



Waffle



Hardhat

Waffle's functions

What Waffle **actually** does?

- Smart contract **compilation**
 - Vyper
 - Solidity
- Smart contract **deployment**
- Smart contract **testing**
 - Matchers
 - Fixtures
 - Smart contract mocks



Waffle's components

What is Waffle **made of?**

- TypeScript
 - TypeChain
- Mocha
- Chai
 - With custom matchers
- ethers.js



Smart Contract Compilation

```
() waffle.config.json > ...
```

You, 6 months ago | 1 author (You)

```
1 {  
2   "sourceDirectory": "./contracts",  
3   "outputDirectory": "./build",  
4   "nodeModulesDirectory": "./node_modules",  
5   "flattenOutputDirectory": "./flatten"  
6 }
```

Exchange.sol

→ **npx waffle**

1. Bytecode
2. ABI (App. Binary Interface)
3. Flattened code

Smart Contract Deployment

```
import BasicTokenMock from "build/BasicTokenMock.json";  
  
token = await deployContract(wallet, BasicTokenMock, [wallet.address, 1000]);
```

Smart Contract Deployment

```
import BasicTokenMock from "build/BasicTokenMock.json";
```

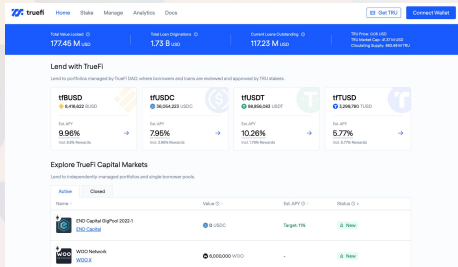
```
token = await deployContract(wallet, BasicTokenMock, [wallet.address, 1000]);
```

```
it('Call as another address', async () => {  
  const contractAsBob = contract.connect(bob);  
  contractAsBob.myFunction() // The sender is Bob.  
})
```

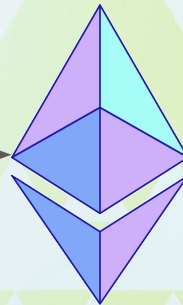


Interacting with the blockchain

Interacting with the Ethereum Network



JSON RPC



Ethereum APIs

- **Infura**
- **Alchemy**
- **Pokt**
- **Metamask** (*injected provider*)
- **In-Memory Emulated Networks** (*i.e. Ganache*)
- **Your own Ethereum node** 🎉

Providing access to Ethereum network with a single endpoint

Ethereum JS libraries

- **Ethers.js**
- **Web3.js**

Wrapping JSON-RPC complexities
with a nice JS interface

ethers.js

```
let privateKey = "0x012345678901234567890123456789012345678901234567890123";  
let wallet = new ethers.Wallet(privateKey);
```

// Connect a wallet to mainnet

```
let provider = ethers.getDefaultProvider();  
let walletWithProvider = new ethers.Wallet(privateKey, provider);
```

```
let balancePromise = wallet.getBalance();
```

```
balancePromise.then((balance) => {  
    console.log(balance);  
});
```

ethers.js

`contract.METHOD_NAME(...args [, overrides]) ⇒ Promise< TransactionResponse >`

Returns a [TransactionResponse](#) for the transaction after it is sent to the network. This requires the **Contract** has a signer.

```
// All overrides are optional
let overrides = {

  // The maximum units of gas for the transaction to use
  gasLimit: 23000,

  // The price (in wei) per unit of gas
  gasPrice: utils.parseUnits('9.0', 'gwei'),

  // The nonce to use in the transaction
  nonce: 123,

  // The amount to send with the transaction (i.e. msg.value)
  value: utils.parseEther('1.0'),

  // The chain ID (or network ID) to use
  chainId: 1
};

// Solidity: function someFunction(address addr) public
let tx = contract.someFunction(addr, overrides)
```



Now the fun begins...

Smart contract **testing**

Basic Testing

```
expect(await token.balanceOf(wallet.address)).to.equal(993);
```

```
expect(BigNumber.from(100)).to.be.within(BigNumber.from(99), BigNumber.from(101));  
expect(BigNumber.from(100)).to.be.closeTo(BigNumber.from(101), 10);
```

Events



```
await expect(token.transfer(walletTo.address, 7))  
  .to.emit(token, 'Transfer')  
  .withArgs(wallet.address, walletTo.address, 7);
```

```
await expect(tx)  
  .to.emit(contract, 'One')  
  .to.emit(contract, 'Two');
```


External Calls

```
await token.balanceOf(wallet.address)  
  
expect('balanceOf').to.be.calledOnContract(token);
```

```
await token.balanceOf(wallet.address)  
  
expect('balanceOf').to.be.calledOnContractWith(token, [wallet.address]);
```

Reverts

```
await expect(token.transfer(walletTo.address, 1007)).to.be.reverted;
```

```
await expect(token.transfer(walletTo.address, 1007))  
  .to.be.revertedWith('Insufficient funds');
```

```
await expect(token.checkRole('ADMIN'))  
  .to.be.revertedWith(/AccessControl: account .* is missing role .*/);
```

```
await expect(token.transfer(receiver, 100))  
  .to.be.revertedWith('InsufficientBalance')  
  .withArgs(0, 100);
```

Token Balances

```
await expect(() => token.transferFrom(wallet.address, walletTo.address, 200))  
  .to.changeTokenBalance(token, walletTo, 200);
```

```
await expect(() => token.transfer(walletTo.address, 200))  
  .to.changeTokenBalances(token, [wallet, walletTo], [-200, 200]);
```

Mock Contracts



```
import {deployMockContract} from '@ethereum-waffle/mock-contract';  
  
...  
  
const mockContract = await deployMockContract(wallet, contractAbi);
```

```
await mockContract.mock.<nameOfMethod>.returns(<value>)  
await mockContract.mock.<nameOfMethod>.withArgs(<arguments>).returns(<value>)
```

```
await mockContract.mock.<nameOfMethod>.reverts()  
await mockContract.mock.<nameOfMethod>.revertsWithReason(<reason>)  
await mockContract.mock.<nameOfMethod>.withArgs(<arguments>).reverts()  
await mockContract.mock.<nameOfMethod>.withArgs(<arguments>).revertsWithReason(<reason>)
```

Mock Contracts

```
it('returns false if the wallet has less than 1000000 coins', async () => {  
  const {contract, mockERC20} = await setup();  
  await mockERC20.mock.balanceOf.returns(utils.parseEther('999999'));  
  expect(await contract.check()).to.be.equal(false);  
});  
  
it('returns true if the wallet has more than 1000000 coins', async () => {  
  const {contract, mockERC20} = await setup();  
  await mockERC20.mock.balanceOf.returns(utils.parseEther('1000001'));  
  expect(await contract.check()).to.equal(true);  
});
```


Fixtures



```
import {expect} from 'chai';
import {loadFixture, deployContract} from 'ethereum-waffle';
import BasicTokenMock from './build/BasicTokenMock';

describe('Fixtures', () => {
  async function fixture([wallet, other], provider) {
    const token = await deployContract(wallet, BasicTokenMock, [
      wallet.address, 1000
    ]);
    return {token, wallet, other};
  }

  it('Assigns initial balance', async () => {
    const {token, wallet} = await loadFixture(fixture);
    expect(await token.balanceOf(wallet.address)).to.equal(1000);
  });

  it('Transfer adds amount to destination account', async () => {
    const {token, other} = await loadFixture(fixture);
    await token.transfer(other.address, 7);
    expect(await token.balanceOf(other.address)).to.equal(7);
  });
});
```


First full setup

```
import {expect, use} from 'chai';
import {Contract} from 'ethers';
import {deployContract, MockProvider, solidity} from 'ethereum-waffle';
import Template from '../build/Template.json';

use(solidity);

describe('Template', () => {
  const provider = new MockProvider()
  const [alice] = provider.getWallets();
  let contract: Contract;

  beforeEach(async () => {
    contract = await deployContract(alice, Template, []);
  });

  it('Deploys correctly and has an address', async () => {
    expect(contract.address).to.be.properAddress
  });

  it('Calls a function', async () => {
    await contract.myFunction();
  });

  it('Transfers ether to the contract', async () => {
    expect(await provider.getBalance(contract.address)).to.eq(0)
    await contract.myFunction({value: 123});
    expect(await provider.getBalance(contract.address)).to.eq(123)
  });
});
```



Let's code!

Difficulty tracks

Beginner

Use already done smart contract code and work on adding tests



Advanced

Start with an empty contract and create smart contract and tests altogether in using Test Driven Development

- Write a failing test
- Implement contract logic
- Check if the test passes
- Refactor
- Repeat



Task 1

Write a **smart contract**

- Split transferred ETH in half and send to two addresses
- Revert if 0 ETH was sent
- Refund remainder

Write **tests**

- Test balances before and after split
 - Check returning the remainder
- Test if contract reverts on 0

Task 2

Write a **smart contract**

- Add proper event when split happens
- Add an event signalling that a non-zero remainder was returned
- Only owner of the contract is allowed to use splitting function
- Use dynamic array of addresses

Write **tests**

- Test the events
- Test owner restrictions
- Test dynamic argument behavior

Let's code!

Task 1

ethereum-waffle.readthedocs.io



- Split ether
- Revert on zero
- Return remainder

Task 2

- Add events
- Use dynamic array
- Add owner

Difficulty tracks

- Focus on tests
- Test-drive smart contract

tinyurl.com/eth-waffle



```
import {expect, use} from 'chai';
import {Contract} from 'ethers';
import {deployContract, MockProvider, solidity} from 'ethereum-waffle';
import Template from '../build/Template.json';
```

```
use(solidity);
```

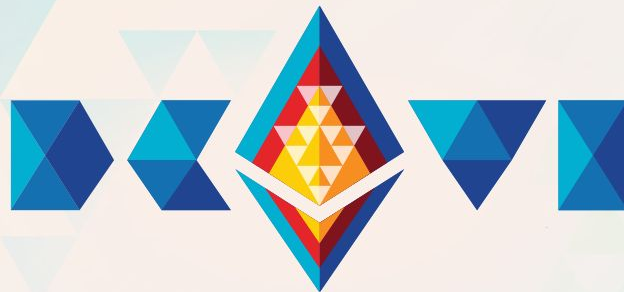
```
describe('Template', () => {
  const provider = new MockProvider()
  const [alice] = provider.getWallets();
  let contract: Contract;

  beforeEach(async () => {
    contract = await deployContract(alice, Template, []);
  });

  it('Deploys correctly and has an address', async () => {
    expect(contract.address).to.be.properAddress
  });

  it('Calls a function', async () => {
    await contract.myFunction();
  });

  it('Transfers ether to the contract', async () => {
    expect(await provider.getBalance(contract.address)).to.eq(0)
    await contract.myFunction({value: 123});
    expect(await provider.getBalance(contract.address)).to.eq(123)
  });
});
```

Thank you!



Daniel Izdebski
@DatSpodo



Bartek Rutkowski
@barrutko



Przemek Rząd
@przemek_rzad



Justyna Broniszewska
@jusbroni

