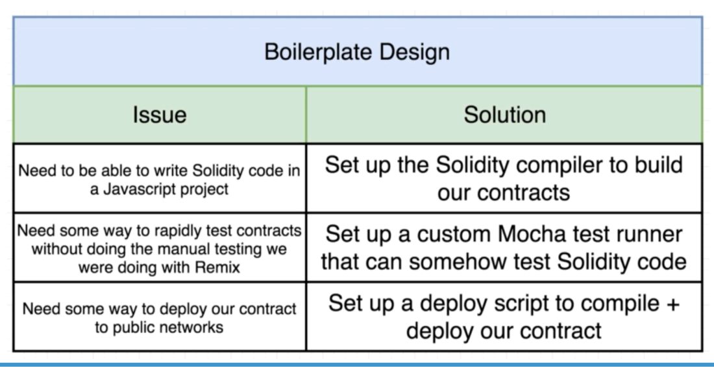
Ethereum and Solidity - The Complete Developer's Guide

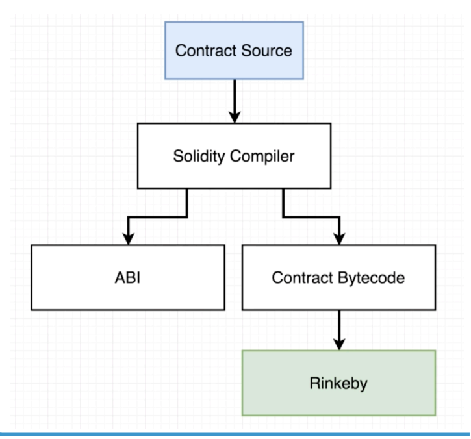
Section 2. Smart Contracts with Solidity

- Boilerplate Requirements

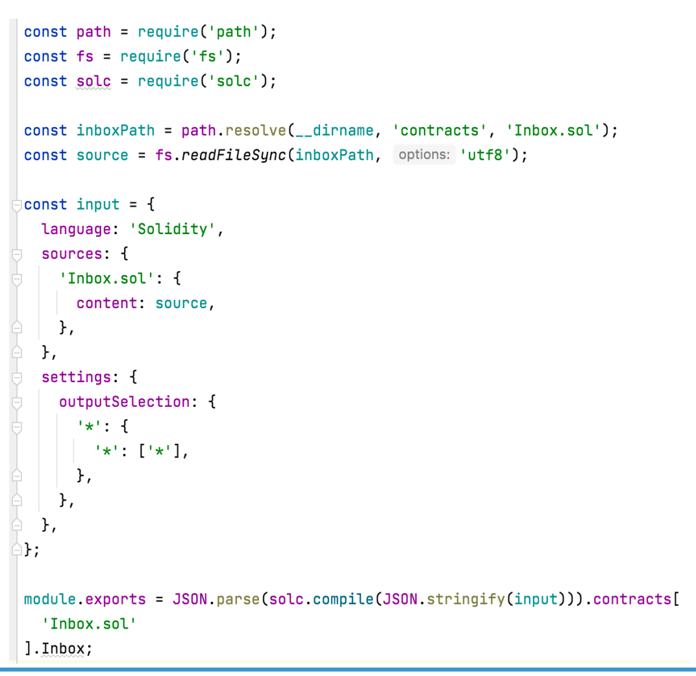


Boilerplate Design	
Issue	Solution
Need to be able to write Solidity code in a Javascript project	Set up the Solidity compiler to build our contracts
Need some way to rapidly test contracts without doing the manual testing we were doing with Remix	Set up a custom Mocha test runner that can somehow test Solidity code
Need some way to deploy our contract to public networks	Set up a deploy script to compile + deploy our contract

  - We will use Node.js, npm



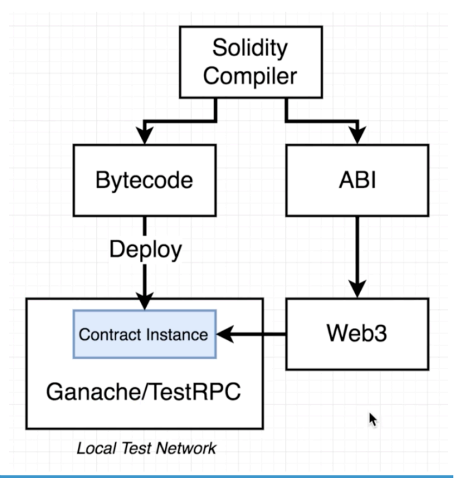
- 1. Compiling contract
  - /contracts/ContractName.sol
  - script 'compile.js'



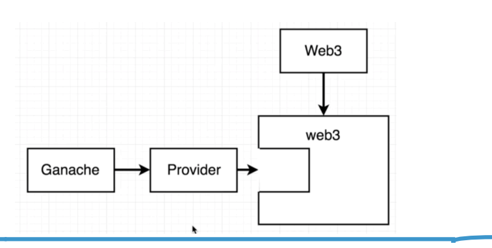
```
const path = require('path');
const fs = require('fs');
const solc = require('solc');

const inboxPath = path.resolve(__dirname, 'contracts', 'Inbox.sol');
const source = fs.readFileSync(inboxPath,  options: 'utf8');

const input = {
  language: 'Solidity',
  sources: {
    'Inbox.sol': {
      content: source,
    },
  },
  settings: {
    outputSelection: {
      '*': {
        '*': ['*'],
      },
    },
  },
};

module.exports = JSON.parse(solc.compile(JSON.stringify(input))).contracts[
  'Inbox.sol'
].Inbox;
```

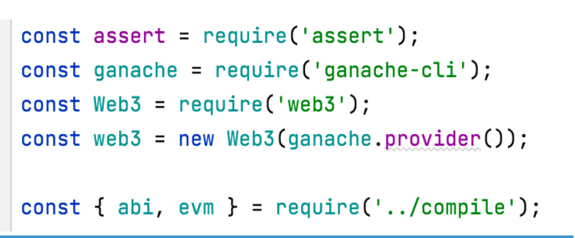
  - ⚠️ export Interface and Bytecode!

- 2. Testing contract locally
  - script 'test/ContractName.test.js'

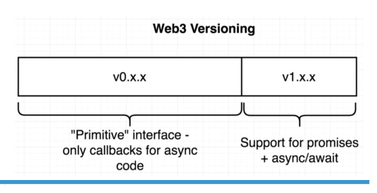


  - Initialization of test network
    - 👍 We run test Ethereum network (Ganache) locally

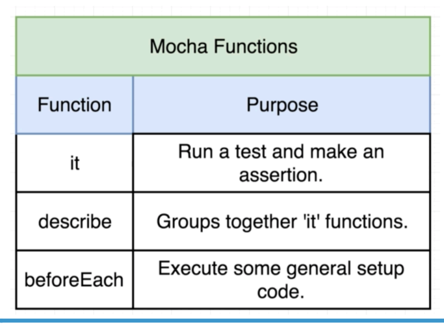




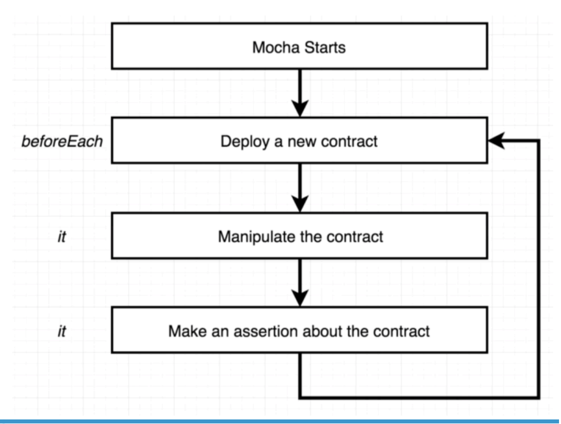
```
const assert = require('assert');
const ganache = require('ganache-cli');
const Web3 = require('web3');
const web3 = new Web3(ganache.provider());

const { abi, evm } = require('../compile');
```


Web3 Versioning
v0.x.x
v1.x.x
"Primitive" interface - only callbacks for async code
Support for promises + async/await

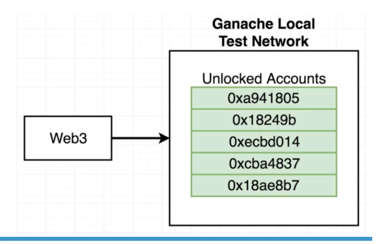
  - Testing with Mocha principles

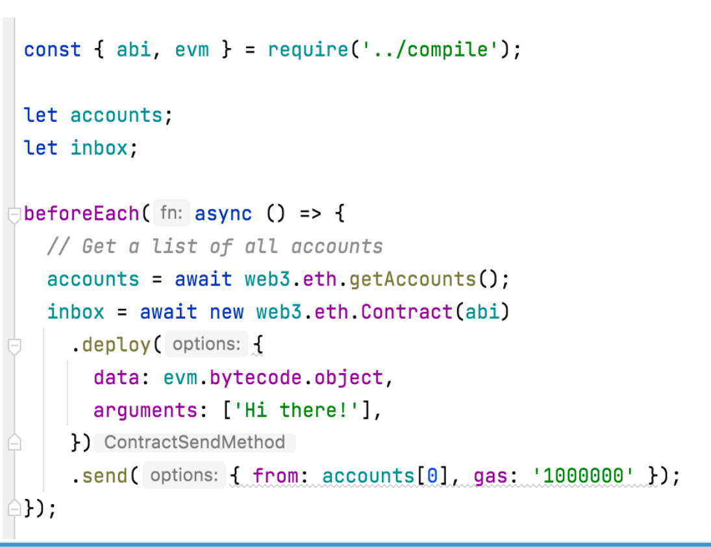


Mocha Functions	
Function	Purpose
it	Run a test and make an assertion.
describe	Groups together 'it' functions.
beforeEach	Execute some general setup code.

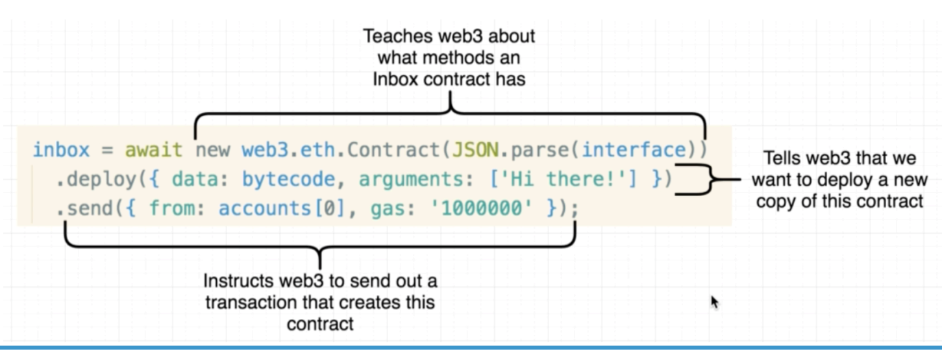


  - Deploying compiled contract on test network using test account
    - 👍 Ganache already has unlocked test accounts without keys - useful for testing!





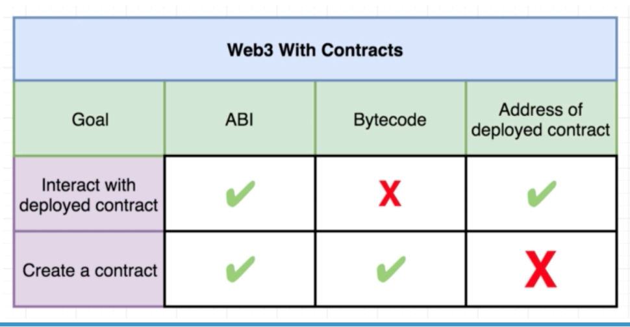
```
const { abi, evm } = require('../compile');

let accounts;
let inbox;

beforeEach( fn: async () => {
  // Get a list of all accounts
  accounts = await web3.eth.getAccounts();
  inbox = await new web3.eth.Contract(abi)
    .deploy( options: {
      data: evm.bytecode.object,
      arguments: ['Hi there!'],
    }) ContractSendMethod
    .send( options: { from: accounts[0], gas: '1000000' });
});
```



```
inbox = await new web3.eth.Contract(JSON.parse(interface))
  .deploy({ data: bytecode, arguments: ['Hi there!'] })
  .send({ from: accounts[0], gas: '1000000' });
```

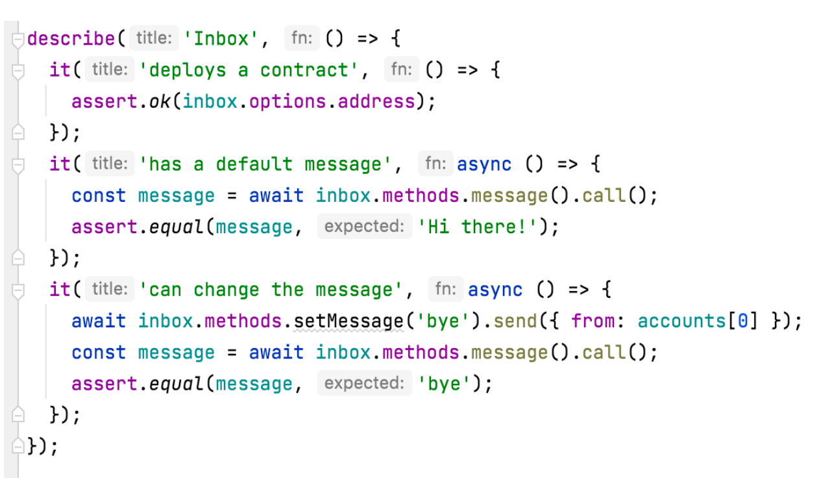
Teaches web3 about what methods an Inbox contract has
Tells web3 that we want to deploy a new copy of this contract
Instructs web3 to send out a transaction that creates this contract

  - Two goals of using Web3



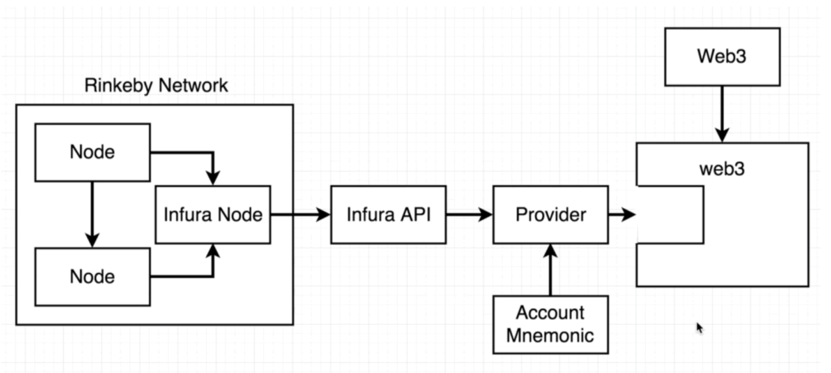
Web3 With Contracts			
Goal	ABI	Bytecode	Address of deployed contract
Interact with deployed contract	✔	X	✔
Create a contract	✔	✔	X

  - Testing contract methods
    - Two types of messages
      - Reading data — call()
      - Changing data (sending transaction) — send({from: account})

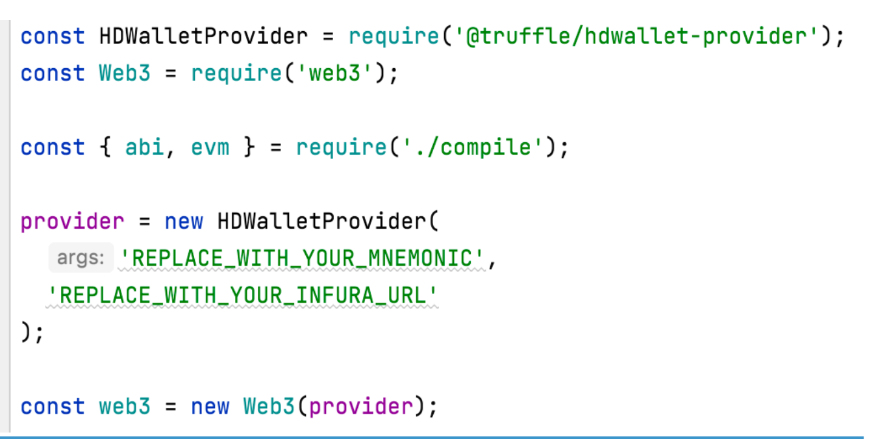


```
describe( title: 'Inbox',  fn: () => {
  it( title: 'deploys a contract',  fn: () => {
    assert.ok(inbox.options.address);
  });
  it( title: 'has a default message',  fn: async () => {
    const message = await inbox.methods.message().call();
    assert.equal(message,  expected: 'Hi there!');
  });
  it( title: 'can change the message',  fn: async () => {
    await inbox.methods.setMessage('bye').send({ from: accounts[0] });
    const message = await inbox.methods.message().call();
    assert.equal(message,  expected: 'bye');
  });
});
```

- 3. Deploying and testing on public test network

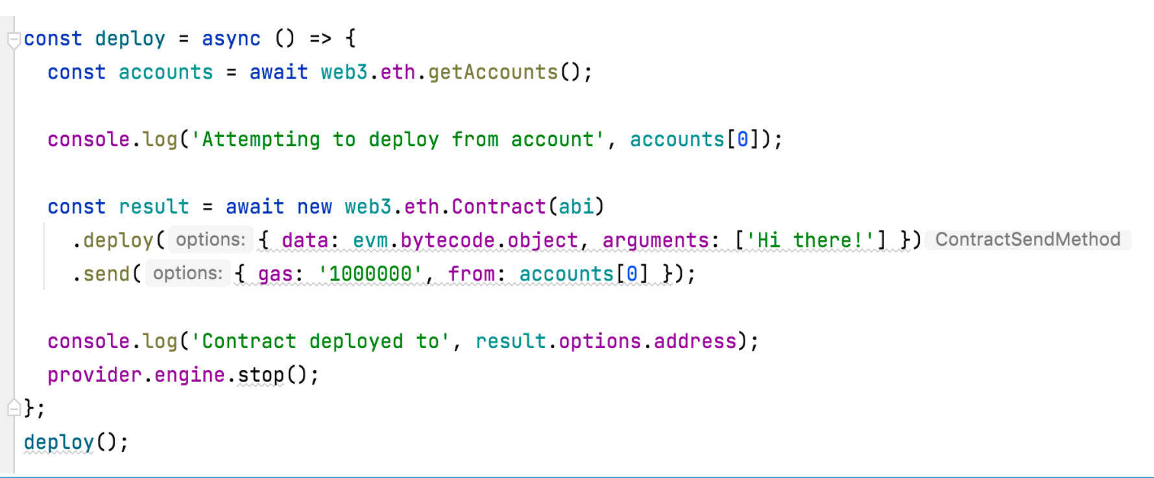


  - Deployment with Infura overview
    - ⚠️ We need to have account with Ether on it!
      - 👍 We can use our test Account Mnemonic created on Rinkeby using MetaMask
        - 🚨 Use only TEST metamask account!
    - ⚠️ We need to connect to some specific Node
      - We can run our node locally
        - 👎 But it is too complex
      - 👍 We can use Infura's nodes
        - Create project on infura.io
          - Copy Rinkeby endpoint
  - script deploy.js
  - Connect to public network using Infura and test account mnemonic



```
const HDWalletProvider = require('@truffle/hdwallet-provider');
const Web3 = require('web3');

const { abi, evm } = require('./compile');

provider = new HDWalletProvider(
   args: 'REPLACE_WITH_YOUR_MNEMONIC',
  'REPLACE_WITH_YOUR_INFURA_URL'
);

const web3 = new Web3(provider);
```

  - Deploying contract



```
const deploy = async () => {
  const accounts = await web3.eth.getAccounts();

  console.log('Attempting to deploy from account', accounts[0]);

  const result = await new web3.eth.Contract(abi)
    .deploy( options: { data: evm.bytecode.object, arguments: ['Hi there!'] }) ContractSendMethod
    .send( options: { gas: '1000000', from: accounts[0] });

  console.log('Contract deployed to', result.options.address);
  provider.engine.stop();
};
deploy();
```

  - Interacting with deployed contract
    - http://remix.ethereum.org/
    - Use Environment="Injected Web3"
      - Connect to MetaMask
    - Load contract "At Address"
    - Now can send messages
      - ⚠️ It requires to spend some Ether - MetaMask asks for confrmation