Portfele Ethereum i inteligentne kontrakty

W części 1 przedstawiłem Ethereum, kiedy omówiłem bitcoin, altcoiny i różne mechanizmy konsensusu. W szczególności omówiłem konsensus PoW Ethereum i sposób, w jaki wykorzystanie Ethereum umożliwia programistom tworzenie własnych inteligentnych kontraktów i tokenów. Wspomniałem, że tokeny Ethereum mogą być generowane jako żądania Ethereum o komentarz (ERC), takie jak ERC-20, ERC-223 lub ERC-777. W części 3 stworzyłeś własny blockchain, a ja omówiłem portfele i transakcje bitcoin. W tej części omówię bardziej szczegółowo Ethereum. Ethereum umożliwia tworzenie kodu (inteligentnych kontraktów) do obsługi funduszy przy użyciu technologii blockchain w celu przezwyciężenia przestojów i ingerencji osób trzecich. Platformę Ethereum przypisuje się głównie Vitalikowi Buterinowi i Gavinowi Woodowi. Według strony internetowej Ethereum definicja Ethereum jest następująca:

„Ethereum to zdecentralizowana platforma, która obsługuje inteligentne kontrakty: aplikacje, które działają dokładnie tak, jak zaprogramowano, bez możliwości przestojów, cenzury, oszustw lub ingerencji osób trzecich". -Ethereum.org

W poprzednich częściach można było przekazywać i przechowywać dane, takie jak przypadek użycia monet w kolorze bitcoin z parametrem OP_RETURN. Jest to przydatne, ponieważ możesz wygenerować skrót MD5 pliku i przechowywać go w sieci bitcoin. Przechowywany MD5 może być dokumentem, umową lub czymkolwiek, co chcesz. Jednak, jak zauważyłeś, bitcoin ogranicza się tylko do przechowywania informacji, a Ty nie możesz wchodzić w interakcję z danymi. W szczególności możesz przekazywać i przechowywać dane w sieci, ale nie możesz uruchomić kodu na swoim pliku, aby wykonać operacje na swoich danych. Ethereum rozwiązuje ten brak funkcjonalności, umożliwiając stworzenie inteligentnego kontraktu wykorzystującego moc blockchain.

Uwaga : Inteligentne kontrakty to programowalny kod używany do obsługi funduszy. Kod działa samodzielnie, bez potrzeby osób trzecich. Solidity to popularny język programowania Ethereum zorientowany na kontakt i może być używany do pisania inteligentnych kontraktów i wdrażania kodu na wielu łańcuchach bloków.

Sercem Ethereum jest maszyna wirtualna Ethereum (EVM). EVM to miejsce, w którym działają inteligentne kontrakty w Ethereum. Dobrym sposobem na zrozumienie EVM jest myślenie o EVM jako o rozproszonym globalnym komputerze, na którym można realizować inteligentne kontrakty.

Uwaga: EVM jest rozproszonym komputerem globalnym do uruchamiania dowolnego, algorytmicznego, złożonego kodu. Mówiąc prościej, EVM składa się ze wszystkich węzłów w sieci Ethereum połączonych jako pojedynczy konsensus i w stanie wziąć kod inteligentnej umowy, przetworzyć go i wykonać. EVM wykorzystuje 256 bitów jako podstawowy mechanizm konsensusu; może obsłużyć blok o pojemności 1 TB, a standardowy czas bloku to 15 sekund.

Zdecentralizowani programiści aplikacji piszą inteligentne kontrakty, a następnie uruchamiają kod na EVM za pomocą kodu front-end. EVM wykonuje kod w równoległych połączeniach na wszystkich połączonych węzłach Ethereum. Zapewnia to konsensus węzłów. Rozmiar łańcucha bloków Ethereum może wynosić nawet 1 TB w momencie pisania w porównaniu z wysokością bloku bitcoina, która jest ograniczona do 4 MB na blok. Dodatkowo bitcoin zajmuje około 10 minut, aby utworzyć nowy blok w porównaniu do 15 sekund na EVM. Chociaż korzystne jest, aby zdecentralizowany kod działał jako pojedynczy konsensus, istnieją również wady. Na przykład kod inteligentnej umowy jest wolniejszy i droższy niż tradycyjny komputer, ponieważ działa na wszystkich węzłach. Aby uruchomić górnika Ethereum, musisz uruchomić EVM z pełnym węzłem. Górnicy uruchamiają mechanizm konsensusu PoW, aby weryfikować transakcje, podobnie jak bitcoin. Podczas procesu wydobycia na każdym bloku

wydobywa się pięć monet. Tak jak widzieliście z NEO w rozdziale 1, górnicy Ethereum otrzymują wynagrodzenie za prowadzenie inteligentnych kontraktów za pomocą monet Ethereum, które zamieniają się w tak zwany gaz.

Uwaga: gaz Ethereum to ułamek tokena Ethereum. Gaz Ethereum jest zmieniany i wykorzystywany przez kontrakt do zapłaty górnikowi za jego wysiłki. Pomyśl o samochodzie. Do działania potrzebuje gazu, podobnie jak Ethereum. W przypadku braku gazu Ethereum nie można wykonać inteligentnego kontraktu.

Ponieważ Ethereum oferuje możliwość budowania ciekawych aplikacji, platforma została doceniona za swój potencjał i jest wykorzystywana w taki czy inny sposób przez Microsoft, Intel, Amazon, J.P. Morgan, a nawet rządy. Dzięki temu Ethereum stał się rozległym ekosystemem z wieloma opcjami do wyboru, które pomogą Ci łatwo tworzyć inteligentne kontrakty. Możesz wybierać spośród wielu narzędzi programistycznych, aplikacji komunikujących się z innymi narzędziami, najlepszych praktyk, infrastruktury, testowania, zabezpieczeń, narzędzi do monitorowania i wielu innych. Wybór narzędzi do użycia może być przytłaczający i mylący, zwłaszcza gdy wiele z nich jest nadal w fazie alfa, beta lub nie jest w pełni przetestowanych. Pamiętaj jednak, że do tej pory jesteś już wyposażony w dobrą podstawową wiedzę na temat technologii blockchain, w tym transakcji, portfeli i tego, jak to wszystko działa. Dodatkowo, łańcuch bloków, który opracowałeś w rozdziale 3, był oparty na JavaScript wykorzystującym Node.js, co jest podstawą wielu narzędzi Ethereum. Istnieją dwie listy, które polecam Ci dodać do zakładek, wymienione tutaj:

- https://github.com/ConsenSys/ethereumdeveloper-tools-list

- https://github.com/ConsenSys/ethereumdeveloper-tools-list/blob/master/EcosystemResources.md

Zasoby te zawierają obszerną listę wszystkich narzędzi programistycznych i zasobów związanych z Ethereum. Omówienie wszystkich tych różnych narzędzi wykracza poza zakres tej książki, ale polecam przejrzenie tych narzędzi w pewnym momencie, jeśli skupisz się na rozwoju Ethereum, abyś mógł sam zdecydować, które narzędzie najlepiej pasuje do twojego projektu. W tym rozdziale skupię się na inteligentnych kontraktach Ethereum i uruchomieniu ich w sieci testowej, tak jak zrobiłeś to w poprzednich częściach dla bitcoina. Pokażę, jak skonfigurować narzędzia programistyczne i IDE oraz podam podstawowe informacje na temat wdrożenia Dapp mainnet

Symulowany klient pełnowęzłowy Ganache

Ganache (wcześniej znany jako ethereumjs-testrpc) umożliwia uruchomienie symulowanego, pełnowęzłowego klienta Ethereum na komputerze i interakcję z umową za pośrednictwem interfejsu wiersza polecenia. To narzędzie jest przydatne, ponieważ będziesz konfigurować sieć programistyczną i prywatną sieć testową, aby przetestować kod inteligentnej umowy. Tak jak widzieliśmy w poprzednim rozdziale, konfiguracja sieci testnet pozwala przetestować kod za pomocą pozorowanych pieniędzy przed przekazaniem kodu do sieci mainnet. Zdecydowałem się użyć Ganache w tej części, ponieważ jest on częścią pakietu programistycznego Truffle i dobrze integruje się z Truffle.

Zainstaluj Ganache

Aby rozpocząć, możesz zainstalować Ganache globalnie za pomocą npm i potwierdzić, że działa poprawnie, wywołując polecenie help.

> npm install -g ganache-cli

> ganache-cli help

Jeśli masz problemy z instalacją lub chcesz uzyskać więcej informacji na temat narzędzia, odwiedź stronę Ganache GitHub: https://github.com/trufflesuite/ganache-cli. Możesz również sprawdzić wersję CLI, uruchamiając to polecenie:

> ganache-cli -v

To polecenie wyświetla wersję. W chwili pisania tego tekstu Ganache CLI to wersja v6.4.3 (rdzeń ganache: 2.5.5).

Ganache CLI: Posłuchaj portu

Możesz uruchomić Ganache na swoim komputerze podczas opracowywania i debugowania swoich kontraktów. Aby to zrobić, skonfiguruj Ganache CLI w Terminalu, aby nasłuchiwał portu, który ustawisz w truffle.js w dalszej części

> ganache-cli -p 8584

Zauważ, że w tym momencie nic nie działa na porcie 8584, więc załóżmy, że będziesz konfigurować port 8584. Polecenie powinno wyprowadzić następujące polecenie:

Listening on 127.0.0.1:8584

Wtyczka IntelliJ IDEA dla solidności

W części 3 pobrałeś i wykorzystałeś WebStorm jako IDE do rozwoju swojego łańcucha bloków. WebStorm jest podzbiorem IntelliJ IDEA i zawiera wtyczkę do języka Solidity, która zapewnia łatwy sposób pisania umów. Zapewnia również podświetlenia i uzupełnianie kodu, aby ułatwić programowanie. Możesz użyć wcześniej zainstalowanej wersji WebStorm i po prostu dodać wtyczkę Solidity. Aby to zrobić, najpierw pobierz wtyczkę tutaj: https://plugins.jetbrains.com/plugin/9475-intellij-solidity. Aby zainstalować wtyczkę, wykonaj następujące kroki:

1. Wybierz WebStorm ➤ Preferencje (lub naciśnij polecenie + ,).

2. Wybierz Wtyczki.

3. Wyszukaj w „Wtyczkach” „Solidność”. To powie „Nie znaleziono wtyczek”. Z linkiem do „Szukaj w repozytoriach”. Kliknij link „Wyszukaj w repozytoriach”. Pojawi się wtyczka „Intellij-Solidity”. S

4. Zainstaluj obie wtyczki „Intellij-Solidity”: JĘZYKI i KONTROLA.

5. Kliknij IntelliJ-Solidity ➤ zainstaluj.

6. W sekcji Wtyczki wyszukaj Solidity Solhint. Pojawi się komunikat „Nie znaleziono wtyczek”. Z linkiem do „Szukaj w repozytoriach”. Kliknij link „Wyszukaj w repozytoriach”. Kliknij Solidity Solhint INSPECTION ➤, a następnie kliknij Zainstaluj.

7. Uruchom ponownie WebStorm.

Pamiętaj, że jeśli jesteś fanem programu Visual Studio, istnieje również rozszerzenie Solidity dla programu Visual Studio; Pamiętaj, że jak zawsze możesz użyć swojego ulubionego IDE, edytora tekstu, a nawet vima, aby napisać swój kod; nie ma potrzeby kupowania IDE.

Truffle

Będziesz używać Truffle, ponieważ jest to jedno z najpopularniejszych narzędzi i ma zintegrowane biblioteki, które pomagają przyspieszyć cykl rozwoju. Truffle Suite zawiera Truffle, Ganache i Drizzle.

„Truffle to środowisko programistyczne, platforma testowa i potok zasobów dla Ethereum, mające na celu ułatwienie życia programistom Ethereum”. —https://github.com/trufflesuite/truffle

Dokumentacja Truffle zawiera instrukcje instalacji, które można znaleźć na https://truffleframework.com/docs. Aby rozpocząć, otwórz nowe okno Terminala i zainstaluj Truffle globalnie na swoim komputerze. Następnie upewnij się, że jest poprawnie zainstalowany, uruchamiając polecenie pomocy, aby wyświetlić listę wszystkich dostępnych poleceń.

> npm install -g truffle

+ truffle@5.0.14

> truffle help

Truffle v5.0.14 - a development framework for Ethereum

Usage: truffle <command> [options]

Twórz swoje inteligentne kontrakty

Aby rozpocząć, utwórzmy folder i zainicjuj kreatora Truffle, aby wygenerować cały kod potrzebny do rozpoczęcia. W Terminalu wpisz:

> mkdir MySmartContract && cd $_

> truffle init

Te polecenia tworzą folder o nazwie MySmartContract i zmieniają lokalizację katalogu na nowy projekt; następnie polecenie init truffle inicjuje projekt. Następnie otwórz WebStorm i otwórz utworzony projekt, wybierając Plik ➤ Otwórz. Przejdź do katalogu projektu MySmartContract i kliknij Otwórz. WebStorm otworzy projekt. EVM obsługuje wiele języków programowania, takich jak Solidity, JavaScript, GO, C++, Python, Java, Ruby, Web Assembly, Rust i Haskell. W tej sekcji będziesz używać Solidity, ponieważ jest to najpopularniejszy język programowania Ethereum dla inteligentnych kontraktów w momencie pisania. Solidność jest oparta na ECMAScript i ma wpływ na JavaScript, C++ i Python. Solidity ma tę zaletę, że możesz wdrażać transakcje inteligentnych kontraktów na innych różnych platformach blockchain oprócz Ethereum, takich jak Ethereum Classic, Tendermint, ErisDB i Counterparty. Solidity używa rozszerzenia pliku .sol; w rzeczywistości, jeśli sprawdzisz folder kontraktów swojego projektu, znajdziesz plik o nazwie Migrations.sol. Ten plik został wygenerowany automatycznie podczas inicjalizacji kreatora Truffle. Pliki migracji ułatwiają wdrażanie kontraktów w sieci Ethereum. W miarę postępu projektu będziesz tworzyć nowe pliki migracji.

Połącz Truffle z siecią Ganache

Następnie dostosujesz swoje środowisko, wywołując swoją sieć, opracowując i ustawiając adres URL i portu. Jak pamiętasz, korzystasz już z Ganache i zaprogramowałeś swoją sieć do nasłuchiwania na 127.0.0.1, port 8584. Użyjesz tych ustawień do wdrażania swoich kontraktów w sieci blockchain Ethereum. Aby rozpocząć, otwórz MySmartContract/truffle-config.js i wewnątrz obiektu sieciowego dodaj obiekt programistyczny z tymi ustawieniami konfiguracyjnymi:

module.exports = {

networks: {

development: {

host: "127.0.0.1",

port: 8584,

network_id: "∗",

gas: 4712388,

gasPrice: 100000000000

}

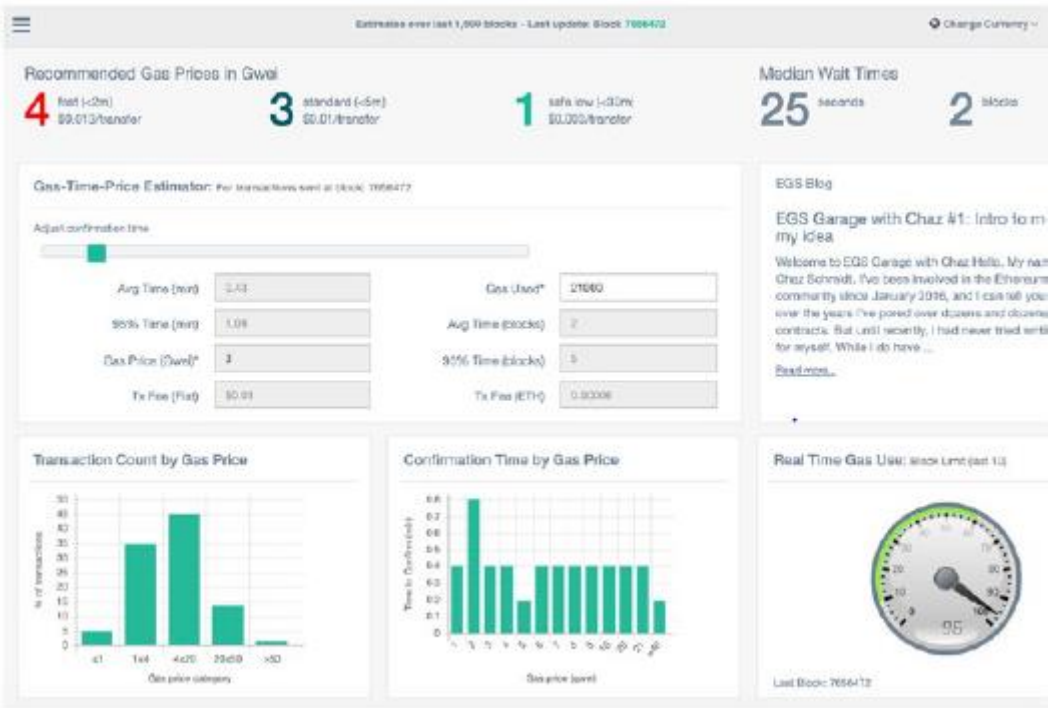
}

Ustawiasz hosta, port i identyfikator sieci, a także gaz i Parametry gasPrice.

– gaz: jest to limit gazu używany do wdrożeń. Wartość domyślna to 4712388.

– gasPrice: jest to cena gazu używana do wdrożeń. Wartość domyślna to 100000000000 (100 Shannon).

Ustawiasz wartości domyślne, które możesz osiągnąć również pomijając tagi gas i gasPrice; jednak w przypadku sieci mainnet na żywo, w momencie pisania tego tekstu, zalecam ustawienie ceny gazu na poziomie 21 000, która jest rozsądną wartością. Sprawdź stację benzynową ETH (https://ethgasstation.info/), aby dowiedzieć się, jaka powinna być wartość gasPrice, jak pokazano na rysunku



Jak widać, w momencie pisania tego tekstu zapłacenie fiat w wysokości 0,014 USD zapewnia standardowy czas transakcji 5,6. Skonfigurowałeś tylko środowisko programistyczne; jednak, gdy przenosisz swój kod z programowania do publicznej sieci testowej, a następnie produkcyjnej, możesz dodać więcej środowisk do pliku truffle-config.js.

Inteligentny kontrakt „Witaj świecie”

Jak wspomniano, inteligentne kontrakty to obiekty kont w blockchainie Ethereum; możesz pisać funkcje do interakcji z innymi umowami, wysyłać monety, podejmować decyzje i przechowywać dane. Ogólnie rzecz biorąc, umowy są budowane tak, aby były zdecentralizowane; należy jednak pamiętać, że można je zaprogramować za pomocą regulowanej opcji, dzięki czemu są scentralizowane. Na przykład dolar Ethereum Gemini ma możliwość zamrożenia transakcji, a nawet ich odwrócenia, a inne monety mogą zostać zbudowane przez właściciela z funkcją samozniszczenia. Zaczniesz od stworzenia prostej umowy „Hello, World". To jest minimalny kod, a intencją tutaj nie jest tworzenie niczego użytecznego, ale pomoc w zrozumieniu, jak stworzyć inteligentny kontrakt. W Terminalu, w lokalizacji projektu, utwórz nowy kontrakt i nazwij go HelloWorldContract za pomocą polecenia truffle.

> truffle create contract HelloWorldContract

Jeśli CLI działał poprawnie i bez błędów, nie wyświetla danych byle jakich. Następnie otwórz utworzoną umowę; pojawi się pod kontraktami/HelloWorldContract.sol. Jak widzisz, kreator Truffle stworzył dla Ciebie umowę. Ta pierwsza inteligentna umowa jest minimalnym przykładem pracy; po prostu przechowuje wiadomość i umożliwia jej odzyskanie przez wywołanie głównej funkcji. Zastąp istniejący kod w contract/HelloWorldContract.sol następującym poniżej;

```
pragma solidity ^0.5.0;

contract HelloWorldContract {

string greeting;

constructor() public {

greeting = 'Hello World';

}

function greet() public view returns (string memory) {

return greeting;

}

}
```

Jak widać, skrypty Solidity są podobne do JavaScript lub C++ i są łatwe do odczytania. Pierwsza linia kodu to wersja kompilatora Solidity; będziesz używać 0.5.0. W konstruktorze HelloWorldContract ustawiasz zmienną powitania na „Hello World". Główną funkcją jest powitanie(). Po wywołaniu funkcji main możesz pobrać wartość zmiennej powitania.

Inteligentny kontrakt „MD5SmartContract"

Teraz stworzysz drugą umowę, która jest bardziej praktyczna. Ten kontakt pozwoli Ci przechowywać hash MD5, który zapisałeś w poprzedniej części, ale tym razem będziesz mógł z nim wchodzić w interakcję zamiast przechowywać dane MD5 na blockchainie. W Terminalu na poziomie projektu utwórz nową umowę o nazwie MD5SmartContract za pomocą polecenia truffle.

> truffle create contract MD5SmartContract

Next, open the contract you created called contracts/

RegisterContract.sol. You will be running the following contact:

```
pragma solidity ^0.5.0;

contract MD5SmartContract {

bytes32 public signature;

event signEvent(bytes32 signature);

constructor() public {

}

function sign(string memory document) public {

signature = sha256(bytes(document));

emit signEvent(signature);

}

}
```

Kod tworzy zmienną sygnaturę wywołania. Wtedy twoja główna funkcja podpisuje twój dokument. Przekazujesz dokument MD5 i za pomocą SHA256 podpisujesz dokument. Tworzysz zdarzenie, które zostanie wysłane po podpisaniu dokumentu.

Uwaga: Secure Hash Algorithm (SHA) to jedna z wielu kryptograficznych funkcji skrótu. Funkcja skrótu kryptograficznego działa jako podpis dla tekstu lub danych; jest jednokierunkowy i nie można go odszyfrować. Wygenerowany skrót SHA256 ma stały rozmiar, 256 bitów (32 bajty) i jest prawie unikalny.

Twórz pliki migracji Truffle na potrzeby wdrażania inteligentnych kontraktów

Jak wspomniano, pliki migracji Truffle pomagają wdrożyć umowy w sieci Ethereum. Utworzysz plik migracji dla swojego wdrożenia. Aby to zrobić, utwórz nowy plik wdrożenia; nazwij to 2_deploy_contracts.js i umieść plik tutaj: migrations/2_deploy_contracts. js. Możesz wskazać utworzony kod inteligentnej umowy w następujący sposób:

```
const HelloWorldContract = artifacts.

require("HelloWorldContract.sol");

module.exports = function(deployer) {

deployer.deploy(HelloWorldContract);

};
```

Utwórz kolejny plik wdrożenia o nazwie 3_deploy_contracts.js i umieść plik tutaj: migrations/3_deploy_contracts.js.

```
const MD5SmartContract = artifacts.require("MD5SmartContract.sol");

module.exports = function(deployer) {

deployer.deploy(MD5SmartContract);
```

};

W tym momencie Twój projekt zawiera dwa inteligentne kontrakty i pliki migracji. Inną techniką dla leniwych programistów jest użycie kreatora tworzenia Truffle do wygenerowania pliku migracji.

> truffle create migration deploy_my_contract

To polecenie automatycznie generuje plik migracji.

Skompiluj swój inteligentny kontrakt z Truffle

W osobnym oknie Terminala uruchomisz Truffle, aby skompilować inteligentną umowę. Polecenie kompilacji zamienia kod Solidity na kod bajtowy, który może być interpretowany przez EVM. Na razie Ganache symuluje EVM.

> truffle compile

Kod bajtowy umowy można zobaczyć w pliku JSON, który można znaleźć tutaj:

build/contracts/HelloWorldContract.json i build/contracts/MD5SmartContract.json. Poszukaj pokazanego tutaj tagu kodu bajtowego:

"bytecode": "0x6080604052348015610010576000080fd5b5061031...",

Uwaga: pamiętaj, że najlepiej byłoby ręcznie usunąć kontrakty / plik ∗.json przed ponowną kompilacją. Zapewni to skompilowanie najnowszego kodu, ponieważ interfejs wiersza polecenia nie zawsze od razu rozpoznaje zmiany.

Wdróż inteligentny kontrakt w swojej sieci programistycznej

Teraz, gdy masz skompilowany kod bajtowy z inteligentnego kontraktu, możesz przeprowadzić migrację kodu bajtowego do środowiska programistycznego, aby uruchomić polecenie migracji, aby przełączyć się na sieć ustawioną w pliku truffle.js.

> truffle migrate --network development

Uruchomienie tego polecenia zwróci odpowiedź pokazaną na rysunku

Pokazuje to, że trzy pliki migracji zostały pomyślnie wdrożone w sieci. Masz pomyślne wdrożenie dla każdego kontraktu. Należy pamiętać, że flaga --reset jest przydatna podczas zmiany kodu, ponieważ trzeba ponownie skompilować i ponownie wdrożyć.

> truffle migrate –reset

Konsola Ttruffle

Teraz, gdy Twoja umowa została wdrożona w Twojej sieci programistycznej, możesz komunikować się ze swoją inteligentną umową za pośrednictwem Truffle CLI. Aby to zrobić, możesz otworzyć konsolę i połączyć ją z siecią programistyczną.

> truffle console --network development

Po uruchomieniu polecenia konsoli terminal pokazuje, że jesteś w trybie programowania Truffle CLI.

truffle(development)>

Aby wyjść z trybu CLI, kliknij dwukrotnie Control+C lub wpisz .exit w konsoli

Wejdź w interakcję ze swoim inteligentnym kontraktem za pośrednictwem Truffle CLI

Ustawiasz dwie zmienne, cześć i podpisz, dla swoich inteligentnych kontraktów, aby móc z nimi wchodzić w interakcje.

truffle(development)> HelloWorldContract.deployed().then(_app

=> { hello = _app })

undefined

truffle(development)> MD5SmartContract.deployed().then(_app =>

{ doc = _app })

undefined

Aby wchodzić w interakcję z umową HelloWorldContract, możesz wywołać główną funkcję publiczną, którą utworzyłeś, ponieważ odsłoniłeś funkcję hello.

truffle(development)> hello.greet()

'Hello World'

Podobnie możesz wchodzić w interakcję z umową MD5SmartContract.sol. Przekażesz ten sam skrót MD5, który wygenerowałeś w części 3 (634ef85e038cea45bd20900fc97e09dc) i wywołasz główną funkcję o nazwie sign. Ta funkcja wygeneruje skrót SHA256.

truffle(development)> doc.sign('634ef85e038cea45bd20900fc97e09dc')

Teraz możesz potwierdzić, że masz skrót SHA256, wywłując funkcję podpisu

truffle(development)> doc.signature()

'0x7869cd540ff8c3b2635ec87251f361e21ad3c72fbc2f79897b9816

bec54b0a48'

Kompiluj z Remiksem

Do tej pory używałeś narzędzi Truffle, sieci Ganache i WebStorm IDE do tworzenia, kompilowania, wdrażania i interakcji z umową; istnieje jednak jeszcze prostszy sposób. Remix oferuje internetowe IDE, które może zrobić to samo, co WebStorm i Truffle. Aby zobaczyć tę pracę, przejdź do strony Remix: https://remix.ethereum.org. Wklej kod inteligentnej umowy „Hello, World" z przykładu. Upewnij się, że prawy panel jest ustawiony na poprawny kompilator; będziesz korzystać z „Bieżącej wersji:0.4.22". Następnie kliknij „Rozpocznij kompilację (Ctrl-S). Utwórz nowy folder w swoim projekcie i nazwij go remiksem; następnie utwórz plik i nazwij go HelloWorldContract.js. Kliknij przycisk Szczegóły w Remix Online IDE, a następnie skopiuj i wklej zawartość WEB3DEPLOY do utworzonego pliku HelloWorldRemix.js.

Uwaga : web3.js to Ethereum JavaScript API; jego biblioteki pozwalają na interakcję z węzłem Ethereum za pośrednictwem połączenia HTTP lub IPC. Kod WEB3DEPLOY można wdrożyć na węźle lokalnym lub zdalnym.

Prywatny blockchain Ethereum z Geth

Nawiązałeś interakcję z inteligentną umową na lokalnym komputerze. Następnie wskazane jest uruchomienie pełnego węzła i przetestowanie inteligentnego kontraktu na blockchainie testnetowym; to testuje go w bardziej realistycznym środowisku. Geth oferuje pełny węzeł Ethereum zaimplementowany w Go, który można uruchomić lokalnie. Ta prywatna sieć testowa pozwoli Ci opracować i przetestować Twój obecny inteligentny kontrakt w oderwaniu od prawdziwego blockchaina Ethereum. Aby rozpocząć, najpierw zainstaluj Geth za pomocą Brew.

> brew tap ethereum/ethereum

> brew install ethereum

Aby upewnić się, że instalacja przebiegła prawidłowo, uruchom polecenie --version dla bieżącej wersji Geth .

> geth version

Version: 1.8.27-stable

Zainicjowany prywatny łańcuch bloków Geth

Teraz, gdy masz zainstalowany Geth, utworzysz swój pierwszy blok lub blok 0, który nazywa się blokiem genezy. Utwórz plik o nazwie genesis_block.json i umieść go w katalogu głównym projektu. Na razie po prostu wklej dostarczony JSON, ale pamiętaj, że możesz wygenerować niestandardowy blok genezy za pomocą skryptu Pythona, który znajdziesz tutaj: https://blog.ethereum.org/2015/07/27/final-steps/.

Użyjesz tego skryptu i ustawisz niski poziom trudności 1000 i limit gazu 1000000, aby ułatwić wydobycie i niskie opłaty za gaz; jednak możesz swobodnie dostosowywać się do potrzeb we własnych eksperymentach. Zobacz genesis_block.json.

```
{

"config": {

"chainId": 1,

"homesteadBlock": 0,

"eip155Block": 0,

"eip158Block": 0

},

"difficulty": "0x1000",

"gasLimit": "0x1000000",

"alloc": {

"0x44dc998cbc1c7504bec0a96af4a9aef6606a768a":

{"balance": "0x133700000000000000000"}

}

}
```

Następnie utworzysz swoją prywatną sieć testową. W Terminalu uruchom to polecenie:

```
> geth --identity "MyTestNet" --nodiscover --networkid 1999

--datadir testnet-blockchain init genesis_block.json
```

Będziesz potrzebować konta dla swojego testnet-blockchain; użyj polecenia konto. Wybierz proste hasło, ponieważ prowadzisz lokalną sieć testową, ale w sieci głównej musisz pamiętać o bezpieczeństwie; tutaj wybieram hasło 123.

```
> geth account new --datadir testnet-blockchain

Passphrase: 123

Repeat passphrase: 123

Address: { a8eceb3e2dd7af9c6fdb12edd8a7e84290932c2d}
```

Jak widać, po wybraniu hasła otrzymałeś adres portfela. Możesz porównać swoje wyniki z moimi

Konsola Getha

Teraz, gdy masz już ustawione konto i łańcuch testnet-blockchain, możesz otworzyć konsolę Geth, aby wchodzić w interakcje z łańcuchem.

> geth --identity "MyTestNet" --datadir testnet-blockchain

--nodiscover --networkid 1999 konsola 2>> geth.log

Zauważ, że użyłem parametru 2>> geth.log, aby wyprowadzić logi do aniestandardowa lokalizacja pliku. Po uruchomieniu konsoli Geth możesz uruchomić eth. polecenie synchronizacji, aby sprawdzić bieżący blok, który jest synchronizowany. W takim przypadku zwróci false, ponieważ nie ma nic do synchronizacji; zaczynasz od bloku 0 w sieci lokalnej. Otrzymasz wyskakujące powiadomienie z pytaniem: „Czy chcesz, aby aplikacja „geth" akceptowała przychodzące połączenia sieciowe? Kliknięcie Odmów może ograniczyć działanie aplikacji. To ustawienie można zmienić w panelu Zapora w preferencjach Bezpieczeństwo i prywatność". Wybierz „Zezwól". Następnie w terminalu Geth uruchom polecenie synchronizacji.

geth> eth.syncing

false

Jeśli uruchomisz polecenie eth.blockNumber, zwróci ono zero, tak jak ty, nie wydobyli jeszcze żadnych bloków.

geth> eth.blockNumber

0

Kopiuj Ethereum dla swojej prywatnej sieci testowej

Następnie możesz potwierdzić, że masz saldo na swoim koncie za pomocą polecenie getBalance.

geth> eth.getBalance(eth.accounts[0])

0

Za pomocą polecenia eth.accounts otrzymasz nowe konto, które utworzmy.

geth> eth.accounts

["0xa2a6d8fe7e39645613e74fe19c79071ee52009ba"]

Możesz generować lub wydobywać monety eterowe w utworzonym przez siebie prywatnym łańcuchu Ethereum. Niezależnie od tego, musisz wiedzieć, jak wydobywać monety, ponieważ będziesz potrzebować transakcji, które będą uwzględniane w wydobywanych blokach podczas testowania kodu. Aby rozpocząć wydobywanie, po prostu uruchom polecenie miner.start.

geth> miner.start()

zero

Podobnie, aby zatrzymać kopanie, po prostu uruchom polecenie miner.stop.

> miner.stop()

zero

Jeśli pozwolisz na uruchomienie wydobycia, wydobędziesz kilka bloków, więc po sprawdzeniu numeru bloku zobaczysz wyniki, a także fundusze.

geth> eth.block.Number

1672

geth> eth.getBalance(eth.accounts[0])

8.36e+21

Wdróż Remix w Geth

Teraz, gdy Twój węzeł jest zsynchronizowany i wiesz, jak kopać, możesz wdrożyć swoje kontrakty w sieci testowej. Najpierw musisz odblokować swoje główne konto Geth, aby móc z niego korzystać. Upewnij się, że Twoje konto posiada saldo; w przeciwnym razie nie będziesz w stanie wdrożyć swojej umowy w sieci. W Geth odblokuj swoje konto za pomocą hasła, aby Geth mogło z niego korzystać.

geth> personal.unlockAccount(eth.accounts[0], "123", 24∗3600)

TRUE

Użyłem hasła 123, ale musisz zmienić je na swoje hasło, jeśli użyłeś innego hasła. Następnie załaduj skrypt web3.js, który wygenerowałeś w Remix.

> loadScript("remix/HelloWorldContract.js")

null [object Object]

true

Wydobywanie następnego bloku i uwzględnienie tego kontraktu zajmuje kilka sekund; po wydobyciu otrzymasz następujący komunikat:

Contract mined! address: 0x9905f1663f1b808d52dca42ce26e0d264

8f8be07 transactionHash: 0x66b80787eb3eae16c9535a1bd86ff1a

623c1914ac9ffc2addde74655aed09157

Jeśli nie widzisz tego komunikatu, upewnij się, że wydobywasz.

geth> miner.start()

Gdy kontakt zostanie wykopany, w Terminalu otrzymasz następującą wiadomość, która zawiera adres i hash transakcji:

Contract mined! address: 0xe49da16551c5c5735de46e07e8ab9e

713310a13b transactionHash: 0x36d3ec593f63280ca6aae1b079bfb6

f00eea719468e04960643c23f39cbef5b3

Wyślij Truffle do Getha

Podobnie, aby wdrożyć skrypt kontraktu web3.js za pośrednictwem Truffle, należy uruchomić polecenie migrate --reset. Flaga --reset mówi Truffle, aby uruchomił wszystkie migracje od początku. Upewnij się, że używasz .exit do wyjścia z konsoli Truffle przed uruchomieniem polecenia migracji. Truffle automatycznie skompiluje Twoją umowę.

truffle(development)> .exit

> truffle migrate –reset

Korzystanie z „rozwoju” sieci.

Sieć na bieżąco.

Teraz możesz ponownie otworzyć konsolę programistyczną.

> truffle console --network development

truffle(development)> HelloWorldContract.deployed().then(_app

=> { hello = _app })

undefined

truffle(development)> hello.greet() 'Hello World''

Umowa zostanie ponownie wdrożona i możesz ponownie wejść w interakcję z umową.

Przydatne polecenia w Getha

Możesz zatrzymać proces Geth, naciskając Command + C, a następnie wyjść, lub możesz zatrzymać proces przez aux, aby sprawdzić, czy są otwarte jakieś procesy. Możesz też użyć polecenia killall, aby zatrzymać proces.

> ps aux | grep getha

> killall -HUP geth

W dowolnym momencie możesz uruchomić flagę pomocy, aby uzyskać listę poleceń.

> geth – help

Aby uzyskać listę oczekujących transakcji, uruchom następujące polecenie:

> geth --identity "MyTestNet" --datadir testnet-blockchain

--nodiscover --networkid 1999 konsola 2>> geth.log

geth> eth.pendingTransactions

Aby usunąć lokalnie zsynchronizowane dane łańcucha bloków z publicznej sieci testowej, użyj tego:

geth> geth removedb

Aby usunąć prywatne dane testowej sieci blockchain, użyj tego:

geth> geth removeb --datadir test-net-blockchain

Aby szybciej zsynchronizować łańcuch bloków, użyj flagi --fast, aby wykonać szybką synchronizację Ethereum. Pamiętaj, że za pomocą tego polecenia nie zachowasz danych z przeszłości. Flaga pamięci podręcznej ustawia limit pamięci podręcznej.

geth> geth --fast --cache=1024

Podłącz portfel Mist Ethereum do swojej sieci prywatnej

Przydałby się portfel, aby połączyć się z siecią prywatną. W tym właśnie pomaga Mgła. Możesz połączyć swój prywatny blockchain z Mist i przeprowadzać transakcje, przeprowadzając realistyczne transakcje tak, jakby ludzie korzystali z twoich umów. Aby rozpocząć, pobierz Mist tutaj: https://github.com/ethereum/mist/releases. W przypadku komputerów Mac plik do pobrania nosi nazwę Mist-macosx-0-11-1.dmg w momencie pisania tego tekstu. Pamiętaj, że możesz również osiągnąć te same wyniki za pomocą portfela Ethereum, który możesz pobrać z tego samego adresu URL. Następnie uruchomisz Mist i połączysz go ze swoim testnetowym blockchainem. W wierszu poleceń wskaż lokalizację Mist i bazę danych geth.ipc.

> /Applications/Mist.app/Contents/MacOS/Mist --rpc /[projectlocation]/MySmartContract/testnet-blockchain/geth.ipc.

Mgła otwiera się i pokazuje Twoje aktywne konto i saldo

Inni do interakcji z Twoim inteligentnym kontraktem

Po opublikowaniu umowy każdy może używać interfejsu binarnego adresu i aplikacji (ABI) do łączenia się i interakcji z umową. Możesz rozpocząć i wchodzić w interakcję z umową tak, jakby dana osoba korzystała z niej przed opublikowaniem jej w mainnet.

Mist to aplikacja komputerowa, której można używać do testowania. Aby obejrzeć kontrakt, w aplikacji Mist kliknij link Kontrakty w prawym górnym rogu, a następnie kliknij Obejrzyj kontrakt. Aby inni mogli zrealizować twój kontrakt, potrzebują dwóch rzeczy.

• Adres umowy

• Interfejs binarny aplikacji

Uwaga : A n ABI opisuje funkcje kontraktu. Ten opis jest potrzebny, aby wiedzieć, jak wywołać funkcję. Pomyśl o tym jak o instrukcji obsługi.

Możesz pobrać adres umowy, jak pokazano tutaj:

truffle(development)>var hello = HelloWorldContract.deployed().

then(_app => { hello = _app })

truffle(development)>hello.address

'0x0b4f69f88390bc8cec93e730128a5e5c5dffd56c'

Podobnie możesz pobrać ABI kontraktu za pomocą tego polecenia:

truffle(development)>JSON.stringify(hello.abi)

'[{"inputs":[],"payable":false,"stateMutability":"nonpayable","

type":"constructor","signature" '[{"inputs":[],"payable":false,

"stateMutability":"nonpayable","type":

"constructor","signature":"constructor"},{"constant":true,

"inputs":[],"name":"greet","outputs":[{"name":"","type":"string"}],

"payable":false,"stateMutability":"view","type":"function",

"signature":"0xcfae3217"}]'

Następnie przekaż adres umowy i ABI w Mist

Zauważ, że pomijasz pojedynczy cytat z ABI i adresu przed wklejeniem go do Mgły. Teraz kliknij OK, a zobaczysz swoją umowę na liście obserwowanych umów.

Masz swoją umowę w Mist i możesz z nią wchodzić w interakcje, wysyłać środki, słuchać wydarzeń i korzystać z funkcji, tak jak użytkownicy będą wchodzić w interakcje z Twoją umową w sieci mainnet.

Metamaska

Podobnie jak w przypadku Mist, innym sposobem interakcji z umowami, nawet bez pobierania aplikacji komputerowej, jest przeglądarka Chrome lub Firefox z wtyczką o nazwie MetaMask. Podobnie jak w przypadku Mist, możesz użyć MetaMask, aby połączyć swój kontrakt z siecią mainnet, publiczną siecią testową i lokalnym łańcuchem bloków (takim jak ten, który utworzyłeś za pomocą Ganache), a nawet możesz połączyć się z Truffle Develop. Aby rozpocząć, pobierz wtyczkę MetaMask dla przeglądarki Chrome lub Firefox.

Kliknij ikonę MetaMask, a następnie przycisk Kontynuuj. Następnie wybierz hasło, zaakceptuj warunki, zapisz tajną frazę kopii zapasowej i utwórz konto. Teraz, gdy konto jest utworzone, masz opcję w górnej liście rozwijanej sieci, z którą chcesz się połączyć. Jak być może pamiętasz, zaprogramowałeś wartości truffle.js, aby ustawić sieć na porcie localhost 8584, który odpowiada sieci domyślnej; jednak możesz ustawić niestandardowe RPC lub połączyć się z testnetem lub mainnetem. Aby uzyskać więcej informacji na temat łączenia Truffle z Metamaskiem, odwiedź dokumentację dotyczącą frameworka Truffle.

Publiczna sieć testowa

Teraz, gdy jesteś w stanie uruchomić swój inteligentny kontrakt w sieci programistycznej, możesz wykonać dodatkowy krok przed przejściem do mainnetu. Możesz uruchomić swoją umowę w publicznej sieci testowej.

Synchronizacja bloków

Istnieją trzy dobrze znane sieci testowe: Ropsten, Kovan i Rinkeby. Możesz zaprogramować Geth, aby łączył się z siecią testową z flagą --testnet, która połączy się z publiczną siecią testnetową (Ropsten).

> geth --testnet --syncmode "fast" --cache=512 console

Tak jak poprzednio, flaga rpc jest potrzebna do akceptowania połączeń Geth RPC i aby Truffle mógł połączyć się z Geth. Ustawiasz również szybką synchronizację i ograniczasz rozmiar pamięci podręcznej do 512. To polecenie obejmuje uruchomienie konsoli Geth. Aby sprawdzić stan polecenia synchronizacji, użyj tego:

geth> eth.syncing

{

currentBlock: 1011878,

highestBlock: 3569550,

knownStates: 2058862,

pulledStates: 2056745,

startingBlock: 968873

}

Po zakończeniu polecenie synchronizacji zwróci false. Pamiętaj, że w momencie pisania istnieją miliony wpisów stanu i 3 569 550 bloków, co może zająć godziny w zależności od szybkości połączenia. Wartość currentBlock to bieżący blok pobierany z całkowitej liczby bloków (najwyższy blok). Dzięki temu możesz zorientować się, jak długo potrwa pobieranie. Jak pamiętasz, możesz sprawdzić bieżący numer bloku, który jest synchronizowany, uruchamiając polecenie eth.blockNumber w konsoli Geth, a także sprawdzić saldo na swoim koncie, aby sprawdzić, czy zostało jeszcze zaktualizowane.

> eth.blockNumber

> eth.getBalance(eth.accounts[0])

Kran publicznej sieci testowej

Oprócz monet testnetowych możesz uzyskać dodatkowe monety testnetowe za pomocą kranu, tak jak w przypadku bitcoina. Wejdź na https://faucet.ropsten.be/ i poproś o monety na adres swojego portfela ustawiony w Mist.

Sieć główna Ethereum

Z konsoli Ganache można było publikować w publicznej sieci testowej (Ropsten). Następnym krokiem jest opublikowanie w sieci głównej Ethereum. Aby to zrobić, zrestartujesz Geth i tym razem połączysz się z siecią główną.

> geth --fast --cache=512

Podobnie jak w przypadku publicznej sieci testowej, będziesz musiał poczekać na synchronizację Getha. Po zakończeniu synchronizacji możesz wywołać polecenie migracji Truffle, aby wdrożyć, i tak jak poprzednio, Twoje konto musi mieć monety ethernetowe.

> truffle migrate –reset

Polecane narzędzia dla inteligentnych kontraktów

W tym rozdziale omówiłem Ganache, Solidity, IntelliJ, Truffle, Geth, Remix i MetaMask; warto jednak wspomnieć o innych narzędziach.

- Solium: roztwór do czyszczenia kodu solidności

- conteract.io: Interakcja z inteligentnymi kontraktami

- Populus: Ramy programistyczne dla inteligentnych kontraktów Ethereum

- Parzystość: lekki węzeł Ethereum

- Mżawka: Front-endowe rozwiązanie dapp

Streszczenie

Omówiłem, jak wykorzystać Ganache do symulacji pełnowęzłowego klienta Ethereum. Zainstalowałeś Ganache, a kiedy już byłeś w stanie połączyć się z Ganache CLI, mogłeś stworzyć sieć i nasłuchiwać portu. Nauczyłeś się, jak korzystać z wtyczki IntelliJ IDEA dla Solidity, aby łatwo tworzyć inteligentne kontrakty z autouzupełnianiem i podświetleniami. Dowiedziałeś się również o pakiecie Truffle Suite i tworzeniu własnych inteligentnych kontraktów za pomocą kreatora wiersza poleceń. Połączyłeś Truffle

z siecią Ganache, a następnie utworzyłeś inteligentną umowę „Hello, World” oraz inteligentną umowę „MD5SmartContract”. Po utworzeniu kontraktów można było przeprowadzić migrację inteligentnej umowy, korzystając z procesu wdrażania Truffle. Skompilowałeś i wdrożyłeś kod inteligentnej umowy z Truffle w swojej sieci programistycznej. Następnie użyłeś konsoli Truffle do interakcji z inteligentną umową za pośrednictwem Truffle CLI. Następnie utworzyłeś prywatny blockchain Ethereum za pomocą Getha i zainicjowałeś blockchain. Korzystasz z konsoli Geth i wydobywasz udawane Ethereum w swojej prywatnej sieci testowej Geth. Następnie wdrożyłeś swój Remix web3.js w utworzonej prywatnej sieci testowej Geth, a także wdrożyłeś swoje kontrakty Truffle. Ponadto przyjrzałeś się kilku przydatnym komendom Geth, które pomogą ci podczas opracowywania inteligentnych kontraktów. Połączyłeś swój portfel Mist Ethereum z utworzoną przez siebie prywatną siecią Geth i byłeś w stanie wchodzić w interakcję z inteligentną umową. Udało Ci się użyć MetaMask w przeglądarce jako zamiennika klienta stacjonarnego. Gdy już byłeś w stanie zobaczyć, jak twoja umowa działa, ustawiłeś publiczną sieć testową i zsynchronizowałeś bloki, a także pobrałeś monety za pomocą kranu. Na koniec dowiedziałeś się, jak przenieść swój kod do głównej sieci Ethereum.