NEO Blockchain i inteligentne kontrakty

W Części 1 omówiłem mechanizm konsensusu blockchain NEO proof of stake (PoS). W rozdziale 2 utworzyłeś węzeł księgowy NEO na AWS Ubuntu i nauczyłeś się, jak poprosić o certyfikat urzędu konsensusu i zostać wybrany na księgowego. W tej części rozszerzę się o blockchain NEO, i dowiesz się jak skonfigurować lokalne środowisko, wykonywać operacje w portfelach NEO, tworzyć inteligentne kontrakty (NeoContracts) i publikować. Omówię wysokopoziomową architekturę blockchain NEO oraz jak skonfigurować lokalne środowisko, utworzyć lokalny łańcuch testowy, tworzyć projekty „Hello, World" zarówno w C#, jak i Pythonie, publikować te inteligentne kontrakty i poznawać kryteria aby porównać Ethereum z EOS z NEO. Jak widać, zrozumienie inteligentnych kontraktów, blockchainu i procesu publikowania jest podobne między projektami, a pokrycie trzech projektów wystarczy, aby zrozumieć, jak pracować z pozostałymi 40 projektami dostępnymi do pisania inteligentnych kontraktów, które są obecnie tam.

Architektura Blockchain wysokiego poziomu NEO

NEO zostało założone w 2014 roku pod nazwą AntShares przez Da Hongfei i Erika Zhanga, a następnie zostało otwarte na GitHub w czerwcu 2015 roku pod nazwą NEO. Mechanizm konsensusu NEO nazywa się Byzantine Fault Tolerant (dBFT), co jest zmodyfikowanym PoS. Ten rodzaj mechanizmu sprawia, że NEO jest skalowalnym blockchainem. Węzły księgowe są losowo wybierane do walidacji transakcji i mogą obsługiwać do 10 000 transakcji na sekundę.

„NEO to projekt blockchain o charakterze non-profit, oparty na społeczności. Wykorzystuje technologię blockchain i tożsamość cyfrową do digitalizacji zasobów i automatyzacji zarządzania zasobami cyfrowymi za pomocą inteligentnych kontraktów. Wykorzystując sieć rozproszoną, ma na celu stworzenie „inteligentnej gospodarki". "-Neo.org

Transakcje NEO to opłaty za pomocą tokenów gazowych NEO. Blok genezy NEO obejmuje 100 milionów NEO. Połowa została sprzedana wczesnym inwestorom, a połowa została zablokowana w tokenach inteligentnych kontraktów NEO. Każdego roku 15 milionów tokenów NEO jest odblokowywanych do wykorzystania przez zespół programistów NEO do finansowania celów rozwoju. NEO pobiera opłaty za transakcje, a także transakcje związane z inteligentną umową. Jeśli chodzi o języki programowania, inteligentne kontrakty NEO obsługują kompilator NeoVM (NEO Universal Lightweight Virtual Machine), Microsoft.net, Java, Kotlin, Go i Python. Oto kilka godnych uwagi funkcji programistycznych NEO:

- NEO może tworzyć inteligentne tokeny kontraktowe zbudowane w standardzie komunikacyjnym (NEP5). Te tokeny są w stanie komunikować się z innymi tokenami NEO.

- Inteligentne kontrakty mogą komunikować się z innymi łańcuchami bloków (ta funkcja nazywa się NeoX).

- NEO może przekazywać informacje za pośrednictwem protokołu udostępniania plików (zwanego NeoFS).

- Wykorzystuje mechanizm kryptograficzny oparty na sieci o nazwie Quantum-Safe (NeoQS).

Infrastruktura „inteligentnej gospodarki" NEO (wyjaśnię tę koncepcję w następnej sekcji) umożliwia inteligentnym kontraktom obsługę aplikacji front-endowych i integrację z innymi inteligentnymi kontraktami i innymi blockchainami za pośrednictwem otwartego API. Otwarte API NEO pozwala na integrację danych ze źródeł zewnętrznych. Rdzeń NeoVM to pole wdrażania (pole przerywane). Jak

widać, dane zewnętrzne z silnikiem wykonawczym (zielona skrzynka) umożliwiają interakcję inteligentnych kontraktów i wykonywanie operacji. Następnie dane mogą być przechowywane w księdze rozproszonej NEO.

„Mamy nadzieję, że platforma może być używana w różnych scenariuszach front-end, takich jak portfel zasobów cyfrowych, forum, głosowanie, zarządzanie profilami i aplikacje mobilne. Platforma posiada również otwarte API, które można wykorzystać do integracji z innymi systemami." -Da Hongfei, Zhao Chen założyciel NEO

Czym jest inteligentna ekonomia NEO?

NEO ukuł termin smart economy, który wyjaśnia wizję NEO. Ta wizja polega na zmianie istniejącego rynku z tradycyjnej gospodarki na gospodarkę inteligentną z mocą zdecentralizowanego łańcucha bloków. Aby osiągnąć ten cel, NEO integruje zasoby cyfrowe, tożsamości cyfrowe i inteligentne kontrakty na swojej platformie.

Uwaga: wizja inteligentnej gospodarki NEO ma na celu zmianę sposobu działania istniejących rynków, z tradycyjnej gospodarki na „inteligentną gospodarkę" z mocą zdecentralizowanego łańcucha bloków. Osiąga się to poprzez integrację zasobów cyfrowych, tożsamości cyfrowych i inteligentnych kontraktów.

Koncepcja inteligentnej gospodarki NEO składa się z integracji następujących trzech komponentów:

- Zasoby cyfrowe NEO: Zasoby te zawierają dane elektroniczne i można je programować. Umieszczenie zasobów cyfrowych w łańcuchu bloków zapewnia korzyści wynikające z łańcuchów bloków PoS, takie jak decentralizacja, zaufanie, identyfikowalność i przejrzystość. Blockchain NEO umożliwia użytkownikom rejestrację, handel i transfer różnych rodzajów aktywów. Aktywa fizyczne uzyskują cyfryzację poprzez tożsamość cyfrową; następnie te zasoby cyfrowe mogą być chronione prawem poprzez walidację. W przypadku ICO rejestracja zasobu cyfrowego kosztuje 5000 gazu. Następnie obowiązuje opłata za odnowienie w wysokości 5000 gazu rocznie.

- Tożsamość cyfrowa NEO: jest to digitalizacja tożsamości osób, organizacji lub innych podmiotów. Tożsamość cyfrowa NEO jest oparta na implementacji standardowej infrastruktury klucza publicznego (PKI) X.509, która obsługuje również sieć zaufania certyfikatów punktowych

- Inteligentne kontrakty NEO: Inteligentne kontrakty w NEO nazywane są NeoContracts i obsługują języki C#, VB.NET, F#, Java, Kotlin i Python. Obsługa tych języków zapewnia korzyści związane z zaawansowanym rozwojem, debugowaniem i kompilacją w środowiskach IDE Visual Studio, Eclipse i WebStorm. NeoVM jest zbudowany z myślą o skalowalności.

Konfigurowanie lokalnego środowiska

Jak wspomniano, NEO obsługuje języki programowania na poziomie korporacyjnym, takie jak C#, VB.NET, F# Java, Kotlin i Python. Ten wybór języków programowania daje NEO przewagę w budowaniu NeoContracts, ponieważ możesz korzystać ze środowiska IDE Visual Studio 2017, które oferuje narzędzia dla przedsiębiorstw do rozwoju. W tym rozdziale będę korzystał z następujących narzędzi .NET:

- Visual Studio 2017 IDE: Aby kontynuować, zainstaluj Visual Studio (VS) Community Edition for Mac.

- .NET Core: Aby kontynuować, zainstaluj .NET Core, aby móc publikować pliki bibliotek DLL.

Oprócz .NET potrzebne są następujące narzędzia:

- Xcode 10.1: Potrzebujesz Xcode 10.2 dla narzędzi i bibliotek, które będziesz instalować.

- Docker: Docker to popularne narzędzie do tworzenia kontenerów i integracji oprogramowania. Będziesz używać Dockera w swojej prywatnej sieci do uruchomienia całego łańcucha bloków NEO, aby symulować cztery węzły konsensusu w jednym, lekkim kontenerze Dockera.

- neo-kompilator: kompilator NEO jest potrzebny do przekształcenia kodu w plik .avm, który można wdrożyć w łańcuchu bloków NEO.

- neo-cli: zainstalujesz i użyjesz narzędzi wiersza poleceń NEO dla portfeli, operacji i wywołań RPC do NEO API.

Teraz, gdy już wiesz, czego potrzebujesz, zacznijmy.

Xcode 10.2

W chwili pisania tego tekstu potrzebujesz Xcode w wersji co najmniej 10.1 dla narzędzi i bibliotek potrzebnych dla NEO. Najnowszy Xcode w momencie pisania to Xcode 10.2.1. Możesz sprawdzić, czy masz już zainstalowany Xcode za pomocą wiersza poleceń.

> xcodebuild --version

Xcode 10.1

Build version 10B61

To polecenie wyświetli wersję, jeśli zainstalowano Xcode

Zainstaluj środowisko IDE programu Visual Studio 2017

Następnie pobierz i zainstaluj najnowszą wersję programu Visual Studio (VS) Wersja Community dla komputerów Mac. Na przyszłość, aby odinstalować część lub całość VS, postępuj zgodnie z instrukcjami tutaj: https://docs.microsoft.com/en-us/visualstudio/ mac/odinstaluj#net-core-script. Kompletny VS 2017 zajmuje dużo miejsca na dysku; jednak nie potrzebujesz wszystkich pobranych pakietów. Do opracowania NeoContracts potrzebujesz tylko skoroszytów Xamarin, więc pobieraj tylko to, co jest potrzebne. Podczas procesu instalacji kreator daje możliwość wyboru platform i narzędzi do zainstalowania. Wybierz skoroszyty Xamarin, klikając pole wyboru, a następnie kliknij przycisk Instaluj

Zainstaluj .NET Core

Będziesz instalować .NET Core, dzięki czemu będziesz mógł publikować pliki bibliotek DLL za pomocą wiersza poleceń. Zostanie to zrobione za pomocą polecenia dotnet publish.

Będziesz pobierał zarówno: Build apps - SDK v2.2.101, jak i Run aplikacje - Runtime v2.2.0. Aby potwierdzić, że instalacja przebiegła prawidłowo, uruchom polecenie dotnet --version.

> dotnet --version

2.2.101

To polecenie wyświetli wersję dotnet, która w czasie pisanie to było 2.2.101. Jeśli zestaw SDK nie jest zainstalowany, otrzymasz następujący komunikat o błędzie:

Did you mean to run dotnet SDK commands? Please install dotnet SDK from: http://go.microsoft.com/fwlink/?LinkID=798306&clcid= 0x409

Możesz również wyprowadzić informacje o swojej maszynie za pomocą polecenia info.

> dotnet –info

Zainstaluj Docker

Następnie zainstalujesz Docker. Docker jest potrzebny do utworzenia kontenera, którego będziesz używać do tworzenia lokalnego łańcucha bloków.

- Pobierz Docker stąd: https://download.docker.com/mac/beta/Docker.dmg

- Instrukcja instalacji: https://runnable.com/docker/install-docker-on-macos

Po pobraniu i zainstalowaniu Dockera kliknij dwukrotnie Docker z menu Aplikacje, aby uruchomić Docker. Zobaczysz ikonę Docker w górnym menu na swoim komputerze. Możesz sprawdzić, czy jest poprawnie zainstalowany, wpisując docker w wierszu poleceń; zawiera listę poleceń platformy Docker.

> docker

Uruchom docker ps, aby wyświetlić działające kontenery, aby upewnić się, że nie dostaniesz wszelkich komunikaty o błędach.

> docker ps

Jeśli Docker nie działa, otrzymasz następujący komunikat:

Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?

Po prostu otwórz Docker, jeśli otrzymasz tę wiadomość. Dodatkowo, jeśli Twój kontener nie działa, ale został już utworzony, możesz użyć opcji flagi -a (wszystkie) i znajdź identyfikator kontenera.

> docker ps –a

List containers

Następnie, gdy masz identyfikator kontenera, możesz go uruchomić.

> docker start [CONTAINER ID]

Na razie nie zobaczysz żadnej listy kontenerów, ponieważ nie utworzyłeś jeszcze swoich kontenerów.

Pobierz NeoCompiler i wygeneruj neon.dll

Aby utworzyć NeoContract, musisz wygenerować plik .avm. Aby to zrobić, musisz utworzyć plik neon.dll, aby móc wygenerować inteligentny kontrakt. Aby rozpocząć, sklonujesz neo-kompilator na pulpit, a następnie wygenerujesz plik neon.dll.

> cd ~/Desktop

> git clone https://github.com/neo-project/neo-compiler

> cd ~/Desktop/neo-compiler/neon/

Aby opublikować samodzielny plik .avm, musisz ustawić identyfikator środowiska wykonawczego. Możesz ustawić identyfikator środowiska uruchomieniowego neon.csproj na poprawny system operacyjny. Ponieważ używam tutaj komputera Mac, a nie komputera PC, muszę zmienić plik neon.csproj. Aby kontynuować, najpierw zrób kopię oryginału.

> cp neon.csproj neon.csproj.backup

Używam vima, ale możesz użyć swojego ulubionego edytora.

> vim neon.csproj

Po otwarciu pliku zastąp następującą konfigurację, która ustawia platformę docelową.

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<Copyright>2016-2017 The Neo Project</Copyright>

<AssemblyTitle>Neo.Compiler.MSIL</AssemblyTitle>

<Version>2.3.1.1</Version>

<Authors>The Neo Project</Authors>

<TargetFramework>netcoreapp2.0</TargetFramework>

<PlatformTarget>anycpu</PlatformTarget>

<AssemblyName>neon</AssemblyName>

<OutputType>Exe</OutputType>

<PackageId>Neo.Compiler.MSIL</PackageId>

<RuntimeIdentifiers>osx.10.12-x64</RuntimeIdentifiers>

<RootNamespace>Neo.Compiler</RootNamespace>

<Company>The Neo Project</Company>

<Product>Neo.Compiler.MSIL</Product>

<Description>Neo.Compiler.MSIL</Description>

</PropertyGroup>

<PropertyGroup Condition="'$(Configuration)|$(Platform)'==

'Release|AnyCPU'">

<DefineConstants>RELEASE;NETCOREAPP1_0</DefineConstants>

<DebugType>none</DebugType>

<DebugSymbols>False</DebugSymbols>

<AllowUnsafeBlocks>true</AllowUnsafeBlocks>

</PropertyGroup>

<PropertyGroup Condition="'$(Configuration)|$(Platform)'==

'Debug|AnyCPU'">

<AllowUnsafeBlocks>true</AllowUnsafeBlocks>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Mono.Cecil" Version="0.10.0" />

<PackageReference Include="Neo.VM" Version="2.3.0" />

</ItemGroup>

</Project>

Teraz opublikuj wskazując na identyfikator środowiska uruchomieniowego osx.10.11-x64 przez przekazanie parametru ustawienia RuntimeIdentifier.

> dotnet publish -r osx.10.11-x64

Kompilator utworzył twój plik neon.dll tutaj:

bin/Debug/netcoreapp2.0/osx.10.11-x64/publish/neon.dll

neo-cli do generowania węzła NEO

Następnie chcesz utworzyć węzeł NEO wypełnienia. Aby wygenerować pełny węzeł NEO, istnieją dwie opcje pełnego węzła.

- neo-gui: Może być używane zarówno przez programistów, jak i użytkowników NEO. Może być używany do wykonywania podstawowych operacji użytkownik-klient, takich jak zarządzanie portfelami, ale także publikowanie inteligentnych kontraktów. Posiada wizualny interfejs użytkownika. Jednak w momencie pisania działa tylko w systemie Windows.

- neo-cli: Zapewnia zewnętrzne API do podstawowych operacji portfela. Pomaga również innym węzłom utrzymać konsensus z siecią i generować nowe bloki.

W tym przypadku instaluję na komputerze Mac, więc będziesz używać neo-cli do zarządzania portfelem za pomocą wiersza poleceń. Jednak dobrze jest wiedzieć, że możesz w ten sposób zainstalować neo-gui i stworzyć wirtualny komputer.

neo-cli

W przypadku neo-cli musisz zainstalować pakiet LevelDB, ponieważ jest to zależność. Jak pamiętasz, zainstalowałeś już LevelDB w Rozdziale 3 przez Homebrew. Jeśli wcześniej nie instalowałeś LevelDB, oto polecenie ponownie:

> brew install leveldb

Alternatywnie możesz sprawdzić, czy go masz i zaktualizować.

> brew upgrade leveldb

Następnie sklonuj neo-cli na swój pulpit.

> cd ~/Desktop

> git clone https://github.com/neo-project/neo-cli

Teraz możesz użyć dotnet do publikowania neo-cli z kodu źródłowego, którego używasz.

> cd neo-cli

> dotnet restore

> dotnet publish -c Release

Plik .dll powinien zostać utworzony w folderze Release.

Aby uruchomić plik .dll, użyj dotnet i lokalizacji pliku DLL, który uruchamia terminal wiersza polecenia NEO.

> cd bin/Release/netcoreapp2.1/

> dotnet neo-cli.dll.

neo-cli obsługuje również wtyczki. Na przykład możesz włączyć dzienniki w neo-cli z dziennikami aplikacji lub poprawić bezpieczeństwo w węzłach RPC za pomocą zabezpieczeń RPC

Utwórz lokalną prywatną sieć testową NEO

Możesz uruchamiać swoje NeoContracts w publicznych sieciach testowych, tak jak robiłeś to z innymi blockchainami; jednak o wiele lepiej jest uruchomić własną prywatną sieć testową, aby mieć nad nią pełną kontrolę. Prywatna sieć testowa może znajdować się w chmurze, ale będziesz musiał zapłacić za dostawcę usług, więc lepiej, jeśli skonfigurujesz swoją sieć testową na swoim lokalnym urządzeniu. Jak wynika z dokumentacji, narzędzia dla NEO zostały stworzone przede wszystkim dla użytkowników komputerów PC. Jednak dzięki narzędziom opracowanym przez społeczność Miasta Zion, prowadzenie prywatnego łańcucha jest możliwe na dowolnej platformie z Dockerem i Pythonem. Kroki, które należy podjąć, aby uruchomić lokalną prywatną sieć testową NEO, są następujące:

1. Zainstaluj neo-python: Pozwala to na uruchomienie pełnego węzła NEO i interakcję z blockchainem.

2. Utwórz neo-privatenet-docker: Pozwala to na uruchomienie całego łańcucha bloków NEO z czterema węzłami konsensusu w jednym, lekkim kontenerze Docker.

3. Utwórz portfel NEO: łączy się z prywatną siecią i tworzy portfel.

4. Twierdzenie: początkowo jest to 100 000 000 NEO.

5. Bootstrap the testnet: synchronizuje sieć.

Python 3.6

neo-python wymaga Pythona 3.6 lub nowszego. Mac wychodzi z pudełka z Pythonem i możesz sprawdzić, czy masz zainstalowany python3 za pomocą polecenia --version.

> python3 --Version

Python 3.6.x

Jeśli używasz poprzedniej wersji Pythona i musisz zainstalować/ponownie zainstalować Pythona, wykonaj następujące kroki:

> python3 --version

Następnie zainstaluj Pythona z Brew.

> brew install --ignore-dependencies

https://raw.githubusercontent.com/Homebrew/homebrew-core/f2a764ef944b1080be64bd88dca9a1d80130c558/Formuła/python.rb

Teraz zmień wersje Pythona.

> brew switch python 3.7.0

> brew switch python 3.6.5_1

Jeśli nie masz zainstalowanego pip, uruchom to:

> curl -O https://bootstrap.pypa.io/get-pip.py

> sudo python get-pip.py

> pip

Zainstaluj neo-pythona

Następnie sklonuj neopytona z Miasta Syjon i sprawdź gałąź deweloperską.

> cd ~/Desktop

> git clone https://github.com/CityOfZion/neo-python.git

> cd neo-python

> git checkout development

Możesz utworzyć środowisko wirtualne za pomocą języka Python 3.6, a następnie uruchomić skrypt aktywacji.

> python3.6 -m venv venv

> source venv/bin/activate

Upewnij się, że masz najnowszą wersję pip, uruchamiając to polecenie:

(venv)> pip install --upgrade pip

Teraz możesz zainstalować pakiet w formie edytowalnej.

(venv)> pip install -e.

Aby potwierdzić, że instalacja przebiegła pomyślnie, uruchom --version dowództwo.

> np-prompt --version

neo-python v0.8.3-dev

Teraz możesz otworzyć bash NEO za pomocą polecenia np-prompt. Aby wyjść z bash, uruchom polecenie exit.

> np-prompt

neo>exit

Zainstaluj neo-privatenet-docker

Zainstalowałeś już Docker, więc teraz możesz stworzyć kontener Docker, który utworzy cztery węzły NEO, aby stworzyć prywatną sieć testową. Śmiało i zainstaluj kontener Docker na pulpicie i skompiluj pliki, jak pokazano tutaj:

> cd ~/Desktop

> git clone https://github.com/CityOfZion/neo-privatenet-docker.

git

> cd neo-privatenet-docker

>./docker_build.sh

Po zbudowaniu obrazu możesz uruchomić sieć prywatną w następujący sposób:

>./docker_build.sh

Pomyślnie zbudowano #numer kompilacji

Uwaga : Jeśli Docker wymaga ponownego uruchomienia lub nie jest uruchomiony, uruchom następujące polecenie:

> ./docker_run.sh

Uruchom sieć i zgłoś początkowy NEO i gaz

Następnie uruchomisz swoją prywatną sieć, utworzysz portfel i odbierzesz początkowy NEO i 40 gazu. Odbywa się to poprzez uruchomienie skryptu docker_run_and_create_wallet.sh.

> ./docker_run_and_create_wallet.sh

Po zakończeniu procesu możesz uzyskać potwierdzenie dwóch utworzonych plików.

- neo-privnet.wallet: Ten plik to portfel, którego możesz używać z neo-pythonem.

- neo-privnet.wif: Ten plik jest kluczem prywatnym WIF, który można zaimportować do innych klientów, takich jak neo-gui.

Pliki te dają dostęp do portfela zawierającego NEO i gaz dla Twojej sieci prywatnej. Skrypt automatycznie pobrał dla ciebie NEO i gaz. Możesz sprawdzić Docker i zobaczyć działający kontener neo-privnet.

> docker ps

Uruchamianie sieci testowej

Teraz, gdy masz uruchomioną prywatną sieć testową, musisz załadować bazę danych blockchain testnetu. To synchronizuje sieć i odbywa się przez uruchomienie np-bootstrap. Może to chwilę potrwać; po zakończeniu otrzymasz potwierdzenie.

> np-bootstrap -n

confirm

Pomyślnie pobrano łańcuch ładowania początkowego! Zwróć uwagę, że używasz flagi –n, aby otrzymywać powiadomienia z bazy danych.

Uruchom NEO Bash

Teraz, gdy masz już swój prywatny kontener testnet działający z czterema węzłami i załadujesz bazę danych testnet, możesz uruchomić bash neo-cli, wywołując polecenie prompt.py.

> cd ~/Desktop/neo-python/neo/bin

> python3.6 prompt.py –p

Po uruchomieniu tego polecenia otworzy się NEO bash i możesz użyć polecenia state, aby wyświetlić informacje o łańcuchu bloków

neo> state

neo-cli oferuje dostęp do wielu wywołań RPC za pośrednictwem NEO API; jednak portfel musi być otwarty, aby uruchomić te polecenia. Możesz otworzyć swój portfel za pomocą polecenia portfela i lokalizacji pliku. To polecenie poprosi o hasło do portfela. Jako hasło użyj coz.

neo> wallet open ~/Desktop/neo-privatenet-docker/neo-privnet.

wallet

password: coz

Następnie odbuduj portfel i wywołaj polecenie portfela. Zobaczysz dostępne fałszywe monety testowe NEO i NeoGas.

neo> wallet rebuild

neo> wallet

Aby zamknąć portfel i wyjść z bash, użyj polecenia zamknięcia portfela close i exit.

neo> wallet close

neo> exit

Udało Ci się stworzyć prywatny blockchain NEO działający na testnet ze 100 milionami NEO i 40.0 NeoGasem twierdził monety, które możesz użyć do rozwoju.

Potencjalne problemy podczas instalacji

NEO czasami ma ochotę gonić za ruchomym celem. W rzeczywistości jest prawdopodobne, że zanim zaczniesz korzystać z instrukcji zawartych w tej książce, kod nie będzie działał zgodnie z oczekiwaniami z powodu zmian w NEO. Ponadto podczas instalacji mogą wystąpić pewne potencjalne problemy

Czysta baza danych

Jeśli chcesz wyczyścić bazę danych neo-python w celu ponownego uruchomienia i synchronizacji, uruchom następujące polecenie:

> rm -rf ~/.neopython/Chains/privnet*

b’Corruption Message

Jeśli otrzymujesz komunikat „b'Corruption: uszkodzona skompresowana zawartość bloku”, musisz ponownie zainstalować LevelDB.

> brew reinstall leveldb

Uruchom ponownie Docker

Warto wiedzieć, jak zrestartować Docker w przypadku konieczności ponownego uruchomienia komputera, uaktualnienia wersji Dockera lub uaktualnienia plików kontenera. Aby ponownie

uruchomić Docker, wybierz Docker z górnego menu i kliknij Uruchom ponownie. Stan zostanie usunięty (cały „stary" blockchain zniknie), a także należy usunąć Chains/privnet z neo-pythona i wszelkich utworzonych portfeli privnet.

> rm ~/Desktop/neo-privatenet-docker/*.wallet

> rm ~/Desktop/neo-privatenet-docker/*.wif

> rm -rf ~/.neopython/Chains/privnet*

> docker ps

NEO „Witaj świecie"

Masz skonfigurowane lokalne prywatne środowisko testnetowe i narzędzia NEO na swoim komputerze, więc teraz jesteś gotowy do rozwoju swojego projektu NeoContract. Możesz programować w różnych językach, a proces jest podobny. Pokażę Ci kod w C# jak również w Pythonie. Zachowałem kod do prostego, działającego przykładu „Hello, World", ale kiedy jesteś w stanie dojść do tego punktu, możesz eksperymentować z różnymi funkcjami, które oferuje NEO. Wykonaj następujące kroki, aby utworzyć i opublikować swój kod:

1. Budowanie struktury NeoContract: Wygeneruj plik Neo.SmartContract.Framework.dll.

2. Stwórz projekt NEO „Hello, World": Stwórz swój projekt kontraktowy #C.

3. Zakoduj inteligentny kontrakt NEO „Hello, World" w C#: Zakoduj swój minimalistyczny przykład w C#.

4. Zakoduj inteligentny kontrakt NEO „Hello, World" w Pythonie: Zakoduj swój minimalistyczny przykład w Pythonie.

5. Opublikuj: Opublikuj swój kontrakt w prywatnym łańcuchu testnetowym.

Budowanie ram NeoContract: Neo. SmartContract.Framework.dll

Pierwszym krokiem jest utworzenie pliku zawierającego kod frameworku NeoContract, który musisz dołączyć do swojego NeoContract, aby uzyskać dostęp do funkcji NEO. Aby zbudować NeoKontrakt, pobierz i zainstaluj pakiet rozwojowy NEO. Umieścisz te narzędzia na pulpicie, aby mieć do nich łatwy dostęp. Pamiętaj, że zawsze możesz przenieść pliki w lepsze miejsce później. Przejdź do pulpitu i sklonuj projekt neo-devpack-dotnet.

> cd ~/Desktop

> git clone https://github.com/neo-project/neo-devpack-dotnet

Następnie uruchom plik neo-devpack-dotnet.sln, klikając go dwukrotnie lub uruchom polecenie Otwórz terminal.

> open neo-devpack-dotnet.sln

VS otwiera się i powinieneś spodziewać się trzech komunikatów o błędach. Kliknij OK, aby odrzucić te komunikaty, ponieważ te błędy nie wpłyną na budowanie projektu.W lewym oknie możesz zobaczyć kartę Rozwiązanie. Rozwiń „neo-devpack-dotnet (master)", jeśli nie jest rozwinięty. Następnie kliknij prawym przyciskiem myszy Neo.Smartcontract.Framework i wybierz Build Neo. Smartcontract.Framework. Po zakończeniu kompilacji w górnym środkowym oknie danych

wyjściowych programu VS pojawi się komunikat „Konfiguracja pomyślna”. Możesz również znaleźć Neo. Plik Smartcontract.Framework.dll tutaj:

> cat ~/Desktop/neo-devpack-dotnet/Neo.SmartContract.Framework/

bin/Debug/netstandard1.6/Neo.SmartContract.Framework.dll

Plik .dll to plik języka .NET Intermediate Language (IL), który zostanie uwzględniony w bibliotece, aby mieć dostęp do kodu platformy NeoContract. Neo.SmartContract.Framework nie obsługuje pełnego zestawu funkcji języka C# ze względu na różnice między plikiem NeoVM a plikiem C# IL.

Utwórz projekt NEO „Hello, World”

Teraz, gdy plik Neo.Smartcontract.Framework.dll jest gotowy do użycia, możesz utworzyć swój projekt i dołączyć framework NEO jako zależność. Aby rozpocząć, otwórz program Visual Studio. Wybierz Plik -> Nowe rozwiązanie&helip; -> Otwiera się kreator nowego projektu. W menu po lewej stronie wybierz Biblioteka -> Biblioteka standardowa .NET. Następnie wybierz .NET Standard 2,0 dla wersji .NET Core, a następnie kliknij przycisk Dalej. Kreator konfiguracji otworzy się z nowym oknem projektu. Wywołaj projekt hello_contract, pozostaw ustawienia domyślne i kliknij przycisk Utwórz. Po utworzeniu projektu należy dołączyć plik Neo.Smartcontract.Framework.dll jako zależność. Aby to zrobić, kliknij prawym przyciskiem myszy folder Zależności w menu Rozwiązanie, a następnie kliknij polecenie Edytuj odwołania. W oknie Edytuj odwołania przejdź do karty Zespół .NET. Wybierz Przeglądaj i dodaj plik Neo.Smartcontract.Framework.dll znajdujący się tutaj:

~/Desktop/neo-devpack-dotnet/Neo.SmartContract.Framework/bin/

Debug/netstandard1.6/Neo.SmartContract.Framework.dll

Następnie kliknij Otwórz. Zaznacz pole wyboru Neo.SmartContract.Framework.dll i kliknij przycisk OK.

Kodowanie inteligentnego kontraktu NEO „Hello, World” w C#

W tej sekcji będziesz używać C# do tworzenia inteligentnego kontraktu NEO „Hello, World” w .NET. NeoVM jest bardziej kompaktowy; możesz skompilować tylko ograniczone funkcje C#/dotnet do pliku AVM. W przykładach wykorzystany zostanie przykład „Hello, World” podany w przykładach NEO.

```
using Neo.SmartContract.Framework;

using Neo.SmartContract.Framework.Services.Neo;

public class Class1: SmartContract

{

public static void Main()

{

Storage.Put(Storage.CurrentContext, "Hello", "World");

}

}
```

Po napisaniu kodu wybierz Build z górnego menu, a następnie Build

Wszystkie (lub Command + B), aby skompilować kod Class1.cs. Plik biblioteki .dll został utworzony w folderze bin/Debug/netstandard2.0/. Użyjesz tego pliku .dll z neo-kompilatorem i przekonwertujesz plik .dll na plik AVM. Po skompilowaniu pliku DLL tworzony jest plik hello_contract.dll:

~/Projects/hello_contract/hello_contract/obj/Debug/netstandard2.0/hello_contract.dll

Uwaga : Framework NeoContract generuje kod bajtowy NeoVM. Kod jest zapisywany w formacie pliku AVM. Plik *.avm może być wdrożony na blockchainie NEO.

Kodowanie inteligentnego kontraktu NEO „Hello, World" w Pythonie

Podobnie jak w #C, możesz wygenerować minimalistyczny kod Pythona, aby wydrukować „Hello, World". Możesz użyć IDE Eclipse (https://www.eclipse.org/ide/) lub dowolnego edytora. Te instrukcje będą używać vima. Utwórz plik o nazwie sample1.py.

> vim ~/Desktop/smartContracts/sample1.py

Wpisz następujący kod, aby wydrukować „Hello World".

```
def Main():

print("Hello World")

return True
```

Aby zamknąć i zapisać plik, wpisz :wq w vim.

Kompilowanie inteligentnych kontraktów do formatu .avm

Teraz, gdy masz już dwa pliki o nazwach sample1.py i hello_contract.dll, następnym krokiem jest skompilowanie tych plików do plików maszyn wirtualnych NEO (.avm), które wdrożysz na łańcuchu bloków NEO. Zacznijmy od skompilowania pliku hello_contract.dll. Zmień katalog na plik DLL.

> cd ~/Desktop/neo-compiler/neon/bin/Debug/netcoreapp2.0/

osx.10.11-x64/publish

Copy Neo.SmartContract.Framework.dll.

> cp ~/Projects/hello_contract/hello_contract/bin/Debug/

netstandard2.0/Neo.SmartContract.Framework.dll ~/Projects/

hello_contract/hello_contract/obj/Debug/netstandard2.0

Teraz możesz użyć narzędzia dotnet core, aby opublikować swoją bibliotekę DLL w pliku AVM,

> dotnet neon.dll ~/Projects/hello_contract/hello_contract/obj/

Debug/netstandard2.0/hello_contract.dll

Możesz zobaczyć plik kodu bajtowego AVM za pomocą polecenia ls.

> ls ~/Projects/hello_contract/hello_contract/obj/Debug/

netstandard2.0/∗.avm

hello_contract.avm

Podobnie możesz skompilować plik Python sample1.py do AVM. W NEO bash, użyj polecenia sc build.

> cd ~/Desktop/neo-python/neo/bin

> python3.6 prompt.py –p

neo> sc build ~/Desktop/smartContracts/sample1.py

Saved output to ~/Desktop/smartContracts/sample1.avm

Opublikuj inteligentny kontrakt w prywatnej sieci testowej

Następnym krokiem jest wdrożenie plików AVM do prywatnego łańcucha testowego NEO. Nie musisz pamiętać wszystkich opcji. Możesz wywołać polecenie z flagą pomocy, aby zobaczyć opcje.

neo> sc deploy help

Deploy a smart contract (.avm) file to the blockchain

Usage: sc deploy {path} {storage} {dynamic_invoke} {payable}

{params} (returntype)

path - path to the desired Python (.py) file

storage - boolean input to determine if smart contract

requires storage

dynamic_invoke - boolean input to determine if smart contract

requires dynamic invoke

payable - boolean input to determine if smart contract

is payable

params - input parameter types of the smart contract

returntype - (Optional) the return type of the smart

contract output

For more information about parameter types see

https://neo-python.readthedocs.io/en/latest/

data-types.html#contractparametertypes

Następnie ustaw storage, dynamic_invoke i payable jako false oraz ustaw params i returntype jako 01.

neo> sc deploy ~/Desktop/smartContracts/sample1.avm False False False 01 01

NEO prosi o nazwę kontraktu; nazwijmy umowę helloWorld. Pozostaw puste pola wersji, autora, adresu e-mail i opisu oraz wprowadź hasło do portfela, aby zapłacić za umowę.

Publikowanie w Mainnet

Aby opublikować w sieci mainnet, możesz użyć tego samego procesu, co w sieci testowej; po prostu załaduj do sieci głównej.

Bootstrapping do Mainnet

Aby załadować do blockchaina mainnet, po prostu uruchom np-bootstrap z flagą -m (jest blisko 10 GB). Możesz również skorzystać z bazy danych powiadomień w sieci mainnet.

> np-podpowiedź –m -n

Instalowanie klienta neo-gui

Łatwiejszym podejściem jest ustawienie i opublikowanie NeoContract przez neo-gui. Musisz skonfigurować maszynę wirtualną na PC, ale wdrażanie plików AVM to pestka. Postępuj zgodnie z tymi instrukcjami:

https://docs.neo.org/en-us/sc/quickstart/deploy-invoke.html

https://docs.neo.org/en-us/node/gui/install.html

Ethereum vs. EOS vs. NEO: Smart Contracts Developer Perspective Showdown

W tym momencie omówiłem trzy główne łańcuchy bloków do tworzenia inteligentnych kontraktów i trudno ich nie porównywać. Jest jednak tak wiele czynników, które należy wziąć pod uwagę przy porównywaniu tych trzech łańcuchów bloków. Ponadto w chwili pisania tego tekstu istnieje ponad 40 projektów blockchain, z których można wybierać do wdrażania inteligentnych kontraktów. Każdy projekt ma zalety i wady, a omówienie ich wszystkich wykracza poza zakres tej książki. Zamiast tego skoncentruję się na konkretnych kryteriach, aby pomóc Ci zrozumieć, jakie czynniki należy wziąć pod uwagę przy wyborze platformy spośród trzech, które omówiłem do tej pory. Istnieje organizacja, która próbuje ocenić te różne łańcuchy bloków; nazywa się to Chińskim Centrum Rozwoju Przemysłu Informacyjnego (CCID). CCID wykorzystuje wkład profesorów i badaczy z najbardziej prestiżowych instytucji edukacyjnych w Chinach, w tym Tsinghua i Beijing University, aby uwzględnić funkcje, wskaźniki adopcji i wiele innych wskaźników w celu uszeregowania każdego łańcucha bloków. Jednak te oceny często się zmieniają i powinieneś sprawdzić najnowsze oceny blockchain na stronie internetowej: http://special.ccidnet.com/pub-bc-eval/index.shtml. Co więcej, określenie, jakiego łańcucha bloków użyć do publikowania inteligentnych kontraktów, powinno uwzględniać więcej czynników, takich jak umiejętności zespołu, finansowanie, liczba potrzebnych transakcji, liczba potrzebnych kont, portfele, giełdy i wiele innych. Innym ważnym wskaźnikiem, który należy wziąć pod uwagę przy określaniu kondycji łańcucha bloków, jest przyjęcie przez użytkowników i programistów. Aktualną liczbę dappów dla różnych platform inteligentnych kontraktów można znaleźć, sprawdzając te witryny:

- EOS: https://dappradar.com/eos-dapps

- Ethereum: https://dappradar.com/dapps

- NEO: http://ndapp.org/

Przeglądając listę dappów, pamiętaj, że chociaż w chwili pisania tego tekstu na Dappradar.com znajduje się 6050 dappów, jest tylko 106 938 użytkowników, co wskazuje, że niewiele dappów jest używanych i nie ma jeszcze masowej adaptacji. Dodatkowo zauważ, że to porównanie jest prawdziwe w momencie pisania i jest oparte na mojej opinii. Powinieneś przeprowadzić własne badania i należytą

staranność przed wyborem idealnego łańcucha bloków, który będzie odpowiadał Twoim potrzebom inteligentnego kontraktu.

- Największym plusem Ethereum jest to, że była to pierwsza i najpopularniejsza platforma inteligentnych kontraktów i ma najwięcej programistów, narzędzi innych firm, wsparcia, dokumentacji i społeczności wsparcia. Największym minusem jest kwestia skalowalności Ethereum przy użyciu PoW; tam jest hard fork w czasie pisania, aby zaradzić temu minusowi i przenieść Ethereum do PoS. Kolejną wadą jest koszt 200 gazu na bajt kodu źródłowego; jest to kosztowne, jeśli Twój kod nie jest zoptymalizowany, zwłaszcza, że musisz stale publikować swój kod. Wreszcie, obsługa mniej popularnych języków programowania, takich jak Solidity, nie jest idealna.

- Zaletą EOS jest jego skalowalność i możliwość wykonywania milionów transakcji na sekundę bez zmian, a także szybsze wykonywanie kodu przy użyciu WASM. EOS obsługuje C i C++, a rzeczywisty blockchain zakodowany w C++ daje mu przewagę, ponieważ C ma większą bazę programistów niż Solidity. Jednak EOS ma przed sobą długą drogę jeśli chodzi o przyjęcie, zapewnienie finansowania w wysokości 1 miliarda USD może być przydatne dla firm i osób fizycznych z odpowiednim pomysłem. Jego wysokie oceny i świetne funkcje nie wystarczą, aby zastąpić Ethereum w dominacji, za którą się podaje. Tylko czas powie.

- NEO obsługuje główne języki programowania (C#, VB.NET, Java i Python), co daje mu dużą przewagę, ponieważ duża liczba programistów może kodować z mniejszą krzywą uczenia się. Dodatkowo sprawna i tania obliczeniowo realizacja kontraktów jest zaletą; jednak NEO ma najmniejsze wsparcie społeczności spośród trzech platform, a sztywny 5000 NeoGas do rejestracji zasobów cyfrowych rocznie może być zabójcą dla wielu potencjalnych projektów.

Streszczenie

W tej części omówiłem blockchain NEO i NEOContracts. Przyjrzałeś się wysokopoziomowej architekturze blockchain NEO i dowiedziałeś się o inteligentnej ekonomii NEO. Ustawiasz środowisko lokalne i uaktualniono Xcode, zainstalowano środowisko IDE programu Visual Studio 2017 i zainstalowano .NET Core. Zainstalowałeś Dockera, więc możesz teraz tworzyć kontenery, pobrałeś neo-kompilator i wygenerowałeś neon.dll. Wreszcie zbudowałeś neo-cli, dzięki czemu możesz zarządzać swoim portfelem i uruchamiać inne operacje RPC. Następnie utworzyłeś lokalną prywatną sieć testową NEO, instalując neo-python i neo-privatenet-docker. Załadowałeś sieć testową i uruchomiłeś NEO bash, a następnie byłeś w stanie uruchomić swoją sieć i odebrać NEO i gaz. Dodatkowo omówiłem potencjalne problemy podczas instalacji narzędzi NEO. Następnie utworzyłeś dwa projekty „Hello, World”, jeden w C# i jeden w Pythonie, i byłeś w stanie skompilować te projekty do plików kodu bajtowego (AVM) maszyny wirtualnej NEO. Wziąłeś te pliki i nauczyłeś się, jak je publikować w blockchainie NEO testnet, a także w sieci głównej NEO. Na koniec porównałem Ethereum z EOS i NEO, aby pomóc Ci lepiej zrozumieć różnice między tymi platformami, a także na jakie kryteria należy zwrócić uwagę przy wyborze platformy dla swoich inteligentnych kontraktów.