Лабораторна робота №1
«Розгортання систем Ethereum та криптовалют»
Абдуллаєва Есміра, Товстенко Артем, Пашинський Максим ФБ-31мп

Мета роботи: Отримання навичок налаштування платформ виконання смарт-контрактів та криптовалют.

Завдання: Провести налаштування обраної системи та виконати тестові операції в системі Ethereum.

Теоретична частина

В Ethereum, система заснована на взаємодії між елементами, званими "акаунтами", кожен з яких оснащений унікальною 20-байтовою адресою. Ці акаунти взаємодіють між собою через обмін вартістю та даними, що веде до змін у загальному стані системи. Опис акаунта включає в себе наступні частини:

- Nonce - це числовий лічильник, що забезпечує унікальність кожної транзакції, запобігаючи її повторному використанню.
- Баланс ефіру - відображає кількість коштів на акаунті в данний момент.
- Код контракту (за наявності) - містить виконуваний код акаунта-контракту.
- Сховище даних - внутрішнє сховище акаунта, що є порожнім за умовчанням.

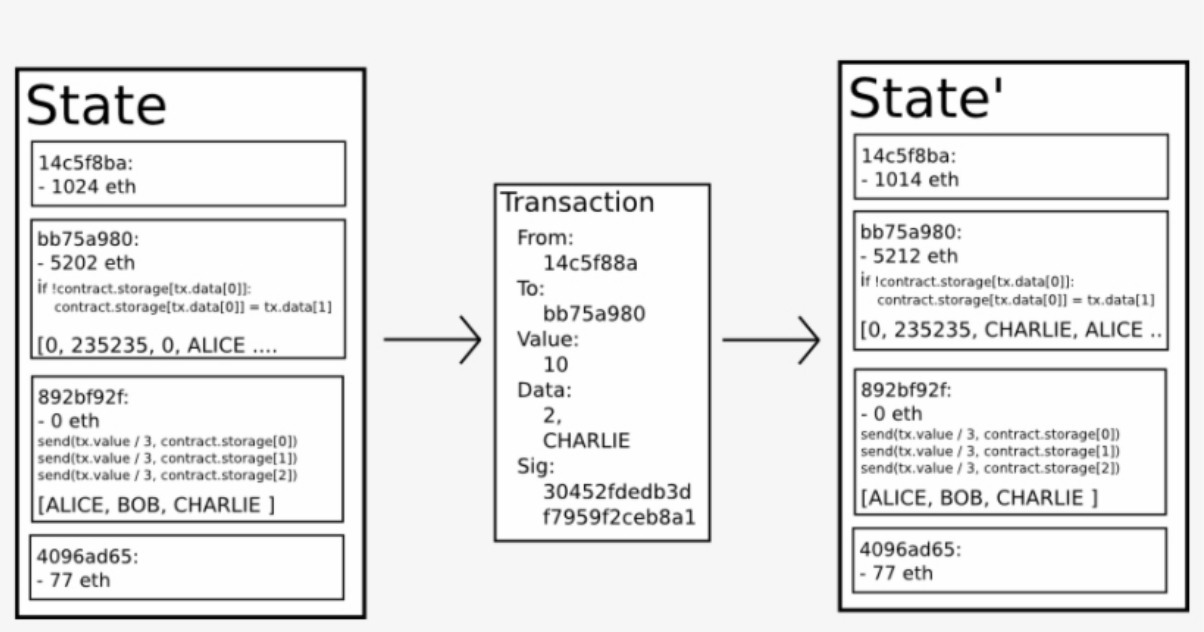
Транзакції в Ethereum є підписаними наборами даних, що виконують інструкції між акаунтами і включають:

- Адресу одержувача - вказує, кому надсилається повідомлення або кошти.
- Підпис відправника - автентифікує особу, що ініціювала транзакцію.
- Кількість ефіру для переказу - сума коштів, що має бути передана.
- Додаткові дані (необов'язково) - містить інформацію за бажанням відправника.
- STARTGAS і GASPRICE - параметри, що визначають ліміт газу для виконання транзакції та ціну газу відповідно.

Контракти у Ethereum можуть ініціювати відправлення "повідомлень" до інших контрактів, що є віртуальними інструкціями, які не підлягають серіалізації і діють лише в межах виконавчого середовища Ethereum. Структура повідомлення включає:

- Відправника (неявний) - акаунт, що ініціює повідомлення.
- Одержувача - акаунт-контракт, до якого надсилається повідомлення.
- Кількість ефіру, що передається з повідомленням.
- Додаткові дані (необов'язково) - інформація, що передається разом з повідомленням.
- STARTGAS - ліміт газу для виконання повідомлення.

Функцію переходу стану Ethereum



Функцію переходу стану в *Ethereum, APPLY(S, TX) -> S'*, можна визначити як процес, у якому поточний стан блокчейна S змінюється на новий стан S' в результаті застосування транзакції TX. Цей процес включає кілька ключових кроків, які забезпечують валідацію, виконання транзакцій, і зміну стану відповідно до правил Ethereum. Ось основні аспекти функції APPLY:

1. **Перевірка транзакції**

Перед тим як транзакція буде застосована, вона перевіряється на відповідність певним умовам:

- Підпис транзакції повинен бути дійсним, а відправник ідентифікований.
- Nonce акаунта-відправника повинен відповідати очікуваному, що запобігає подвійній витраті та повторенню транзакцій.
- Баланс відправника повинен бути достатнім для покриття вартості переказу ефіру та комісій за газ.

2. **Розрахунок комісії за газ**

Визначається сума газу, що витрачається на виконання транзакції, і розраховується комісія, множачи витрачений газ на ціну газу, вказану в транзакції.

3. **Виконання транзакції**

Транзакція виконується згідно з її типом:

- Переказ ефіру між акаунтами.
- Виклик контракту, який може включати виконання коду контракту та зміну даних контракту.

4. **Зміна стану**

Стан S змінюється на S' через:

- Оновлення балансу акаунтів (відправника і одержувача).
- Зміни в стані контрактів, якщо відбувся виклик контракту.
- Оновлення nonce акаунта-відправника.

5. **Завершення транзакції**

Після застосування транзакції видається квитанція транзакції, що містить інформацію про виконання, включно з використаним газом та будь-якими змінами в стані.

Ця модель забезпечує консистентність і відтворюваність стану в Ethereum, дозволяючи всім учасникам мережі переходити від одного загального стану до іншого в результаті виконання транзакцій.

Газ в Ethereum

Концепція газу в Ethereum відіграє ключову роль у підтримці економічного балансу всієї системи, встановлюючи ціни на виконання операцій залежно від їхньої складності та потреби в обчислювальних ресурсах. Це забезпечує, що вартість обробки транзакцій адекватно відображає витрачені на це ресурси, регулюючись механізмом пропозиції та попиту на ринку.

Що стосується структури мережі:

1. Bootnode — це вузол для первинної ініціації, що використовується для виявлення інших учасників мережі. Він працює на порті 30303, і нові вузли приєднуються до мережі, спочатку з'єднуючись із цим bootnode.
2. JSON-RPC endpoint — цей вузол надає доступ до API JSON-RPC через HTTP на порту 8545. Порт 8545 цього вузла публікується для зовнішніх взаємодій з цією приватною блокчейн-мережею.
3. Майнер — цей вузол відповідає за майнінг, процес створення нового блоку в блокчейні. Коли вузол-майнер успішно здійснює майнінг нового блоку, він отримує винагороду на сконфігурований акаунт.

Genesis block

Genesis блок є фундаментом для будь-якого блокчейна Ethereum, включаючи приватні мережі. Він визначає початковий стан мережі і містить важливі параметри конфігурації. Роз'яснення кожного компонента у genesis блоці:

Розділ **"config"** містить конфігурації, які визначають протокольні правила мережі.

- **"chainId":** Унікальний ідентифікатор ланцюга (9876 у вашому прикладі), який допомагає запобігти повторним витратам в мережах Ethereum через механізм replay protection.
- **"homesteadBlock":** Вказує номер блоку, з якого починається активація Homestead, другої великої версії протоколу Ethereum. Тут встановлено 0, що означає її активацію з самого початку.
- **"eip150Block", "eip155Block", "eip158Block":** Номери блоків для активації відповідних Ethereum Improvement Proposals (EIPs), кожен з яких вносить зміни у функціонування мережі. Значення 0 означає, що ці удосконалення застосовуються з першого блоку.
- **"byzantiumBlock", "constantinopleBlock", "petersburgBlock":** Вказують на активацію відповідних оновлень мережі, починаючи з першого блоку.
- **"ethash":** Визначає алгоритм консенсусу Ethash, який використовується для доведення роботи (Proof of Work). Пустий об'єкт {} означає використання стандартних налаштувань Ethash.

"gasLimit": Встановлює максимальний ліміт газу для блоку, тобто максимальну кількість газу, яка може бути використана в усіх транзакціях в цьому блоку. У нашому прикладі ліміт становить 12,000,000.

"difficulty": Визначає складність першого блоку, що є мірою скільки обчислень потрібно для знаходження відповідного хешу. Значення "1" робить це дуже простим, сприяючи швидкому майнінгу блоку в приватних мережах.

"alloc": Дозволяє вказати початковий розподіл ефіру між акаунтами в мережі. У вашому випадку цей об'єкт пустий {}, означаючи, що ніякі початкові баланси не встановлені. Це можна використовувати для пре-майнінгу ефіру на певні акаунти.

Практична частина

Підготовка:

1. Створення genesis.json файлу.
   Кожен блокчейн починається з генезису (першого) блоку. Блок genesis налаштовується за допомогою файлу genesis.json для приватної мережі.
2. Створення .env файлу.

```
NETWORK_ID=2345
ACCOUNT_PASSWORD=test-password
```

3. Створення ключ вузла та еноди для початкового вузла:

```
esmira23 blockchain23-24/lab1 on  main [!?] via  desktop-linux
❯ bootnode -genkey bootnode.key

esmira23 blockchain23-24/lab1 on  main [!?] via  desktop-linux
❯ bootnode -nodekeyhex c4732f8d7b8ac69c00abfcab308439d735adf73ad50fd77ecc63dee285c4433b -writeaddress
b32f6428f4678b84e8d1cb98fc76c8a05cebeaea9c3b406f0599bdde5dbe5d15cc04b3947a5243cc65c53f9819974d2e565ef5709662b5decd4efb2750d92214
```

Dockerfile

Dockerfile створює образ для запуску вузла Ethereum використовуючи офіційний образ клієнта Go Ethereum (ethereum/client-go) версії v1.10.1. Він призначений для ініціалізації приватного блокчейну Ethereum з певним genesis блоком і створення нового акаунта.

```
FROM ethereum/client-go:v1.10.1

ARG ACCOUNT_PASSWORD

COPY genesis.json .

RUN geth init ./genesis.json \
    && rm -f ~/.ethereum/geth/nodekey \
    && echo ${ACCOUNT_PASSWORD} > ./password.txt \
    && geth account new --password ./password.txt \
    && rm -f ./password.txt

ENTRYPOINT ["geth"]
```

Docker-compose file

docker-compose файл створений для розгортання приватної мережі Ethereum, що складається з трьох вузлів: одного bootnode та двох вузлів-майнерів. Конфігурація дозволяє легко запустити і управляти цією мережею в ізольованому середовищі за допомогою Docker.

Bootnode

```
  mybootnode:
      hostname: mybootnode
      env_file:
        - .env
      build:
        context: .
        args:
          - ACCOUNT_PASSWORD=${ACCOUNT_PASSWORD}
      command:
        --
nodekeyhex="c4732f8d7b8ac69c00abfcab308439d735adf73ad50fd77ecc63dee285c4433b"
        --nodiscover
        --ipcdisable
        --networkid=${NETWORK_ID}
        --netrestrict="172.16.254.0/28"
      networks:
        priv-eth-net:
```

- **nodekeyhex** використовується для задання фіксованого ключа вузла для bootnode, дозволяючи заздалегідь визначити його enode адресу. Це необхідно для конфігурації з'єднань з іншими вузлами мережі.
- **nodiscover** забезпечує, що цей вузол не шукатиме інші вузли самостійно, оскільки він призначений для прямого підключення до нього інших учасників мережі.
- **ipcdisable** вимикає міжпроцесне спілкування, оптимізуючи вузол для його ролі як bootnode, що не вимагає високої взаємодії з іншими процесами.
- **networkid** встановлює унікальний ідентифікатор для мережі, що дозволяє вузлам ідентифікувати і приєднуватися до правильної приватної мережі і уникає конфліктів із головною мережею Ethereum.
- **netrestrict** обмежує мережеві з'єднання вузла, дозволяючи приймати з'єднання лише з певного діапазону IP-адрес, заданого у форматі CIDR, що забезпечує додатковий рівень ізоляції і безпеки.

Miner

```
  miner-1:
      hostname: miner-1
      env_file:
        - .env
      build:
        context: .
        args:
          - ACCOUNT_PASSWORD=${ACCOUNT_PASSWORD}
      command:
        --
bootnodes="enode://b32f6428f4678b84e8d1cb98fc76c8a05cebeaea9c3b406f0599bdde5dbe5d15
cc04b3947a5243cc65c53f9819974d2e565ef5709662b5decd4efb2750d92214@mybootnode:30303"
        --mine
        --miner.threads=1
        --networkid=${NETWORK_ID}
        --netrestrict="172.16.254.0/28"
      networks:
        priv-eth-net:
```

- **mine** активізує процес майнінгу на даному вузлі.
- **miner.threads** дозволяє вказати кількість потоків процесора, які будуть задіяні для процесу майнінгу.
- **miner.etherbase** встановлює адресу акаунта для зарахування винагород за блоки, що змайновані. У цьому випадку, адреса не вказана, тож винагороди за майнінг автоматично спрямовуються на акаунт, створений за промовчанням.

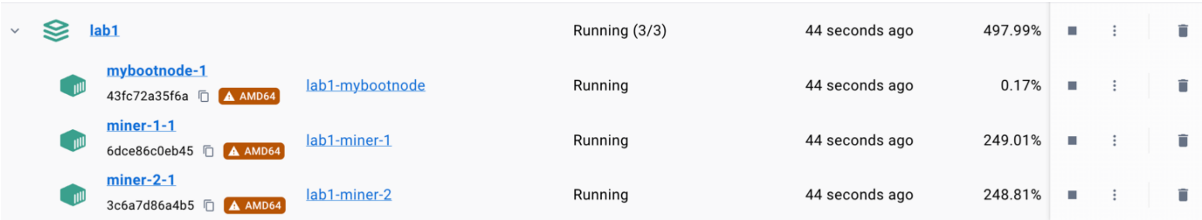
Network configuration

```
networks:
  priv-eth-net:
    driver: bridge
    ipam:
      config:
        - subnet: 172.16.254.0/28
```

Запуск контейнерів

```
esmira23 blockchain23-24/lab1 on  main [!?] via  desktop-linux
❯ docker-compose build
[+] Building 1.3s (17/21)                                                                                    docker:desktop-linux
 => [miner-1 internal] load .dockerignore                                                                                   0.0s
 => => transferring context: 2B                                                                                             0.0s
 => [miner-1 internal] load build definition from Dockerfile                                                                0.0s
 => => transferring dockerfile: 338B                                                                                        0.0s
 => [miner-2 internal] load metadata for docker.io/ethereum/client-go:v1.10.1                                               1.2s
 => [mybootnode internal] load .dockerignore                                                                                0.0s
```

```
esmira23 blockchain23-24/lab1 on  main [!?] via  desktop-linux
❯ docker-compose up
[+] Running 7/5
 ✔ Network lab1_priv-eth-net                                                                                      Created   0.0s
 ✔ Container lab1-miner-2-1                                                                                       Created   0.0s
 ✔ Container lab1-miner-1-1                                                                                       Created   0.0s
 ✔ Container lab1-mybootnode-1                                                                                    Created   0.0s
 ! miner-1 The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platform was requested   0.0s
 ! mybootnode The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platform was requested   0.0s
 ! miner-2 The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platform was requested   0.0s
Attaching to lab1-miner-1-1, lab1-miner-2-1, lab1-mybootnode-1
lab1-miner-2-1     | INFO [03-26|19:44:51.466] Maximum peer count                       ETH=50 LES=0 total=50
lab1-miner-2-1     | INFO [03-26|19:44:51.469] Smartcard socket not found, disabling    err="stat /run/pcscd/pcscd.comm: no such file or directory"
lab1-miner-1-1     | INFO [03-26|19:44:51.469] Maximum peer count                       ETH=50 LES=0 total=50
lab1-miner-1-1     | INFO [03-26|19:44:51.472] Smartcard socket not found, disabling    err="stat /run/pcscd/pcscd.comm: no such file or directory"
lab1-mybootnode-1  | INFO [03-26|19:44:51.475] Maximum peer count                       ETH=50 LES=0 total=50
lab1-mybootnode-1  | INFO [03-26|19:44:51.478] Smartcard socket not found, disabling    err="stat /run/pcscd/pcscd.comm: no such file or directory"
```



Перехід до терміналу одного вузла майнера та виконання команди geth attach, щоб отримати доступ до консолі Javascript.

```
esmira23 ~ took 16s
❯ docker exec -it 501c4e4b65df /bin/sh
/ # geth attach
Welcome to the Geth JavaScript console!

instance: Geth/v1.10.1-stable-c2d2f4ed/linux-amd64/go1.16
coinbase: 0x9491fba28eac8b21c077baac63556580d26417bb
at block: 2 (Tue Mar 26 2024 20:13:12 GMT+0000 (UTC))
 datadir: /root/.ethereum
 modules: admin:1.0 debug:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0
```

Перевірка enode вузла:

```
> admin.nodeInfo.enode
"enode://13b735298ee7e3e115c28523702e363949205f68521a3147c1e760d0081ba9943ed0d72045d5f08fe06e2657f685ba3346f253c8429bea4242cce4e4a53bf2c7@127.0.0.1:30303"
```

Спочатку, перевірка балансу першого рахунку.

```
> eth.getBalance(eth.accounts[0])
10000000000000000000
```

Створення нового акаунту та його розблокування.

```
> personal.newAccount()
Passphrase:
Repeat passphrase:
"0x3873ef1bb9d85e0c58fa03aba3bd46e8f65aab4c"
> eth.accounts
["0x9491fba28eac8b21c077baac63556580d26417bb", "0x3873ef1bb9d85e0c58fa03aba3bd46e8f65aab4c"]
> eth.getBalance(eth.accounts[1])
0
> personal.unlockAccount(eth.accounts[0], "test-password")
true
```

Проведення транзакцій:

```
> eth.sendTransaction({from:eth.accounts[0], to:"0x3873ef1bb9d85e0c58fa03aba3bd46e8f65aab4c", value: web3.toWei(1, "ether")})
"0x2258ec3e41abf0ec15bdc6c9203776fb976378bf96ae3c444575ee2caec8464b"
> eth.getBalance(eth.accounts[1])
8000000000000000000
```

Інформація про транзакцію та блок:

```
> eth.getTransaction("0x2258ec3e41abf0ec15bdc6c9203776fb976378bf96ae3c444575ee2caec8464b")
{
  blockHash: "0x728245ad845a4637df0c4370de7722d3702bc70cf337c86ef217a8baeee0ab62",
  blockNumber: 18,
  from: "0x9491fba28eac8b21c077baac63556580d26417bb",
  gas: 21000,
  gasPrice: 1000000000,
  hash: "0x2258ec3e41abf0ec15bdc6c9203776fb976378bf96ae3c444575ee2caec8464b",
  input: "0x",
  nonce: 3,
  r: "0x5648a6bddff3832aa8dba6b909bdd9fd2d1871aaa83be89652577f59ca2ea6d0",
  s: "0x49ae3935161a7312beaaa2645405151c50ba2af22085a97b4cb4bfdf9e9817c",
  to: "0x3873ef1bb9d85e0c58fa03aba3bd46e8f65aab4c",
  transactionIndex: 0,
  type: "0x0",
  v: "0x4d4b",
  value: 1000000000000000000
}
> eth.getBlock("0x728245ad845a4637df0c4370de7722d3702bc70cf337c86ef217a8baeee0ab62")
{
  difficulty: 131712,
  extraData: "0xd683010a0184676574688676f312e3136856c696e7578",
  gasLimit: 11790831,
  gasUsed: 21000,
  hash: "0x728245ad845a4637df0c4370de7722d3702bc70cf337c86ef217a8baeee0ab62",
  logsBloom: "0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000",
  miner: "0x9491fba28eac8b21c077baac63556580d26417bb",
  mixHash: "0xc91397540d453f96b9f6d090302b82525d9547d5ece27c6923fc354b93c62ca0",
  nonce: "0x1621cfd576349615",
  number: 18,
  parentHash: "0x3cb66a6d13a0c87e57e06be2eb8abdd0cbfc9298735890333cf767daa54cd122",
  receiptsRoot: "0xc2cd6858dd9c95ee24cc3fa28fbd27fa7a563a28eb57c7f13c6f55a64b0253a0",
  sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
  size: 648,
  stateRoot: "0x07935c5cab1aca92ae02946d32389d6f831bdf3c2c8c8333eca2b010807f08c4",
  timestamp: 1711484506,
  totalDifficulty: 2363713,
  transactions: ["0x2258ec3e41abf0ec15bdc6c9203776fb976378bf96ae3c444575ee2caec8464b"],
  transactionsRoot: "0x3d3869972f693e71209536e19f3ec7e703ccf53383fd9a708dbf2eb6d7311910",
  uncles: []
}
```