Infrachain

Developer documentation

Developer documentation

Introduction

The Public Sector Blockchain (PSB) was launched in 2019 to allow the Luxembourg government to develop a series of new blockchain applications for the public sector. The Infrachain Challenge is a hackathon that will enable the team to provide an innovative answer based on PSB. Valuable resources for developers are gathered in this document.

Chain specification

PSB is a private network based on the Ethereum protocol. All member nodes are running with Go Ethereum. The consensus Algorithm is Clique (Proof of Authority).

You can check the following resources:

- Ethereum documentation : https://ethereum.org/developers/#getting-started
- Geth command line option: https://github.com/ethereum/go-ethereum/wiki/Command-Line-Options

Setting up the development environment I

Each team can set up a development environment locally using the content shown in the "Hackathon2022" folder, then choose your distribution (MacOS or Other (2022)). Please refer to README for the instructions.

The genesis file (genesis.json) provided gives necessary information about the Public Sector Blockchain, including gas limit and target block time of 15 secondes.

You can use this genesis to create your local test environment during the event.

For this challenge, A minimized chain is available with a docker-compose. 2 mining nodes and a public node are started. The public node can be accessed at:

- HTTP-RPC server: localhost:8545/

Useful resources:

- JSON RPC API documentation: https://github.com/ethereum/wiki/wiki/JSON-RPC
- In case you prefer to you POSTMAN: https://documenter.getpostman.com/view/4117254/ethereum-json-rpc/RVu7CT5J

The authority nodes are interconnected to maintain the consensus and "public" node managed by chain participants are used to interact with the chain.

A Block explorer is also available: localhost:80 by default. You can change it in explorer-config-file/nginx.conf

Getting started

By the end of the challenge, teams will need to provide specific deliverables. Each team is free to choose languages and to use their favorite development tools. The following resources can help you choose:

- Languages: https://ethereum.org/developers/#language-specific-resources
- IDE: https://ethereum.org/developers/#integrated-development-environments-ides
- Developer tools: https://ethereum.org/developers/#developer-tools
- Docker Images: https://geth.ethereum.org/docs/install-and-build/installing-geth

Account

A Keystore is available in the folder with an account with many ethers
0x8f04152f99d639fe427aadc32d3d9648038e8df6
There is no password for this account (you can use the empty password file to unlock the account).

Private keys

You can use the following private keys to sign any transaction:

- node 1 privatekey:
  22aae6e36021acbf8d4a05a169d77919929d390dab212c609c319ea99c4dd298
- node 2 privatekey:
  0f7a99cc92f0d32b8ee2ed697e013acede5b2c12008d3e71fd2c58dac3e78a06

Smart Contract

Each team is responsible for writing and deploying its smart contracts. You can use remix to get your contract's bytecode.

Useful resources are:

- Ethereum documentation: https://ethereum.org/developers/#smart-contract-languages
- Solidity documentation: https://solidity.readthedocs.io/
- Remix documentation: https://remix-ide.readthedocs.io/en/latest/layout.html

Signing transaction

Each team is responsible for signing their transactions. The following snippet is an example that can help you write your own code.

```
var EthereumTx = require('ethereumjs-tx');
var keythereum = require("keythereum");
var fs = require('fs');

const signerAddr = ""; // public address of account that will sign transaction

var password = fs.readFileSync('password').toString().trim(); // retrieve password from password file.

const transaction = {
   "gasLimit": "0x47b760",        // constant, specified by genesis
   "gasPrice": "0x3B9ACA00",
   "value": "0x",                 // hex value of ether to transfer, empty when deploying a smart contract
   "from": signerAddr,
   "to": "",                      // destination address, empty when deploying a smart contract
   "nonce": "0x",                 // transaction count in hex, curl -X GET "infrachain-bc.intech-
dev.com:8081/accounts/{{accounts_from}}/getNonce"
   "data": "0x"                   // bytecode of smart contract
};

const keyObject = keythereum.importFromFile(signerAddr, "./");      // keystore path of the signerAddr
const privateKey = keythereum.recover(password, keyObject);

const tx = new EthereumTx(transaction);
tx.sign(privateKey);
console.log("0x"+tx.serialize().toString('hex'));     // print signTransaction to send
```

JavaScript snippet to sign transaction

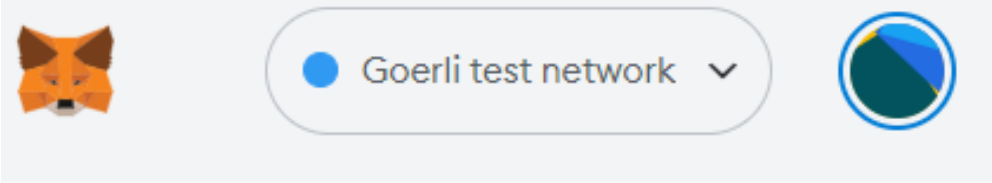
Setting up development environment II

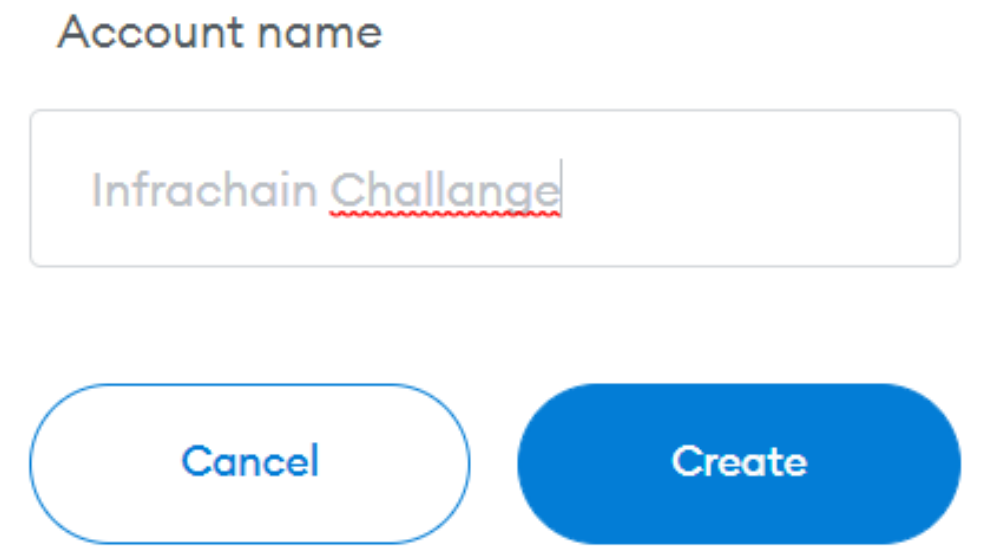
Since the PSC is an Ethereum-based blockchain, you can use other Ethereum test environments to develop your solution (DAap).

Goerli/Sepolia Test Network and Meta Mask.

To use **Goerli** Test Network, follow these steps:

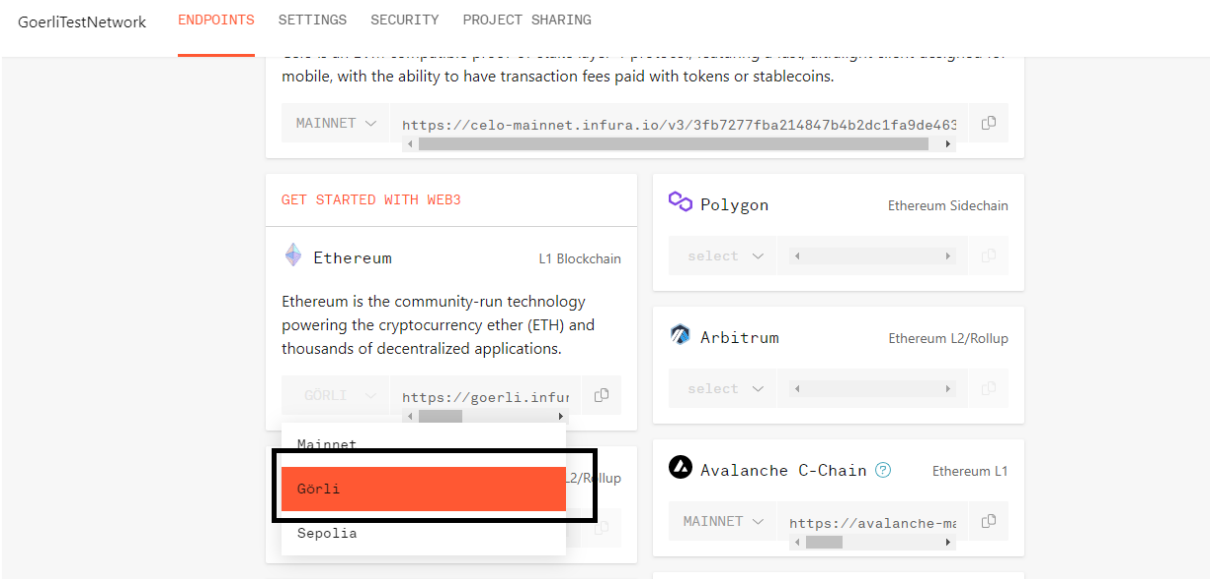
a) Download and Install Metamask (extension in google chrome), https://metamask.io/
b) Create an account in **Goerli** (or Sepolia) Test Network





c) Add ether to your account with token: https://goerli-faucet.mudit.blog/ or alternatively you can use https://goerlifaucet.com/
d) Block/Transaction explorer: https://goerli.etherscan.io/
e) Use https://remix.ethereum.org/ to develop and deploys smart contracts.
f) Confirm transaction via Metamask

For JSOR-RPC (Web3) provider you can use Infura:



```
var Web3 = require('web3')
var url = 'https://goerli.infura.io/v3/3fb7277fba214847b4b2dc1fa9de463c'
var web3 = new Web3(url)

//print all Web3 methods. ...
console.log(web3)
```

You can use the provided link, or you can create your own Web3 project with Infura and provide transaction.

Good Luck!