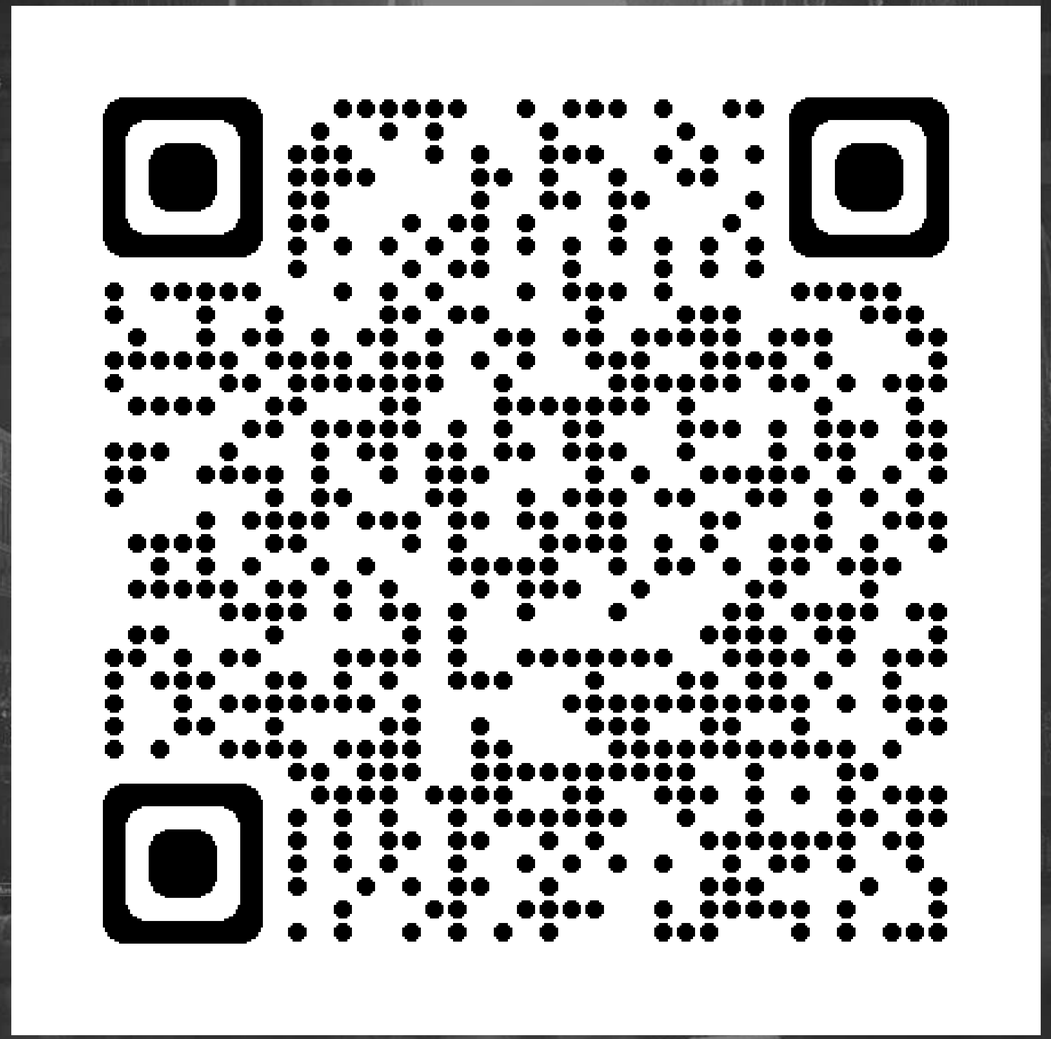
tinyurl.com/45p27hnf



GO TO THE REPO NOW ➔

Hedera dApp Days

Build on a Next Generation “Blockchain”

Ed Marquez – Swirlds Labs
Abi Castro – Swirlds Labs

/ed-marquez

@ed__marquez

hedera.com

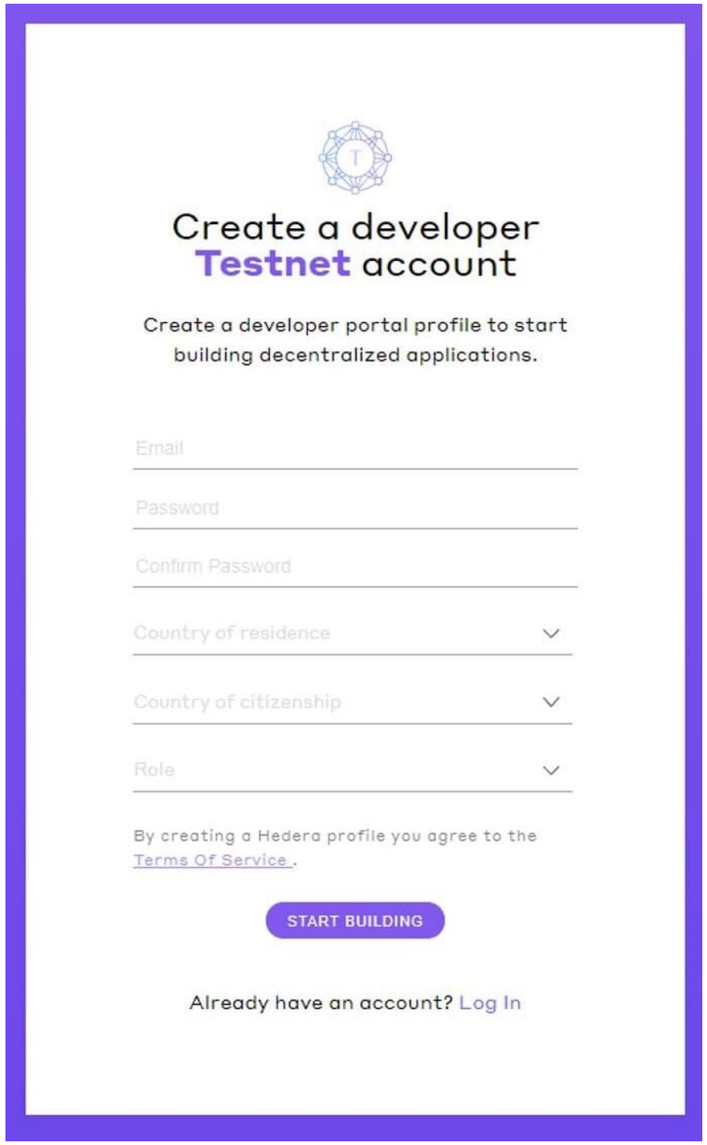
With dApp days, you will learn how to build and deploy applications on Hedera!

Section 1:
Introduction to
Web 3 and Hedera

Section 2:
Buidl on Hedera

Section 3:
Enter NFT Giveaway
and Next Steps

Get a testnet account! (https://portal.hedera.com/register)



APPLICATION PREVIEW

The web evolves continuously and now it is becoming more decentralized

Pre-1970

Analog Era

Paper documentation

1970

Digital Era

Mainframes & computers

- Speed

2000

Internet Era

Interconnected computers

- Speed
- Access & collaboration

2009

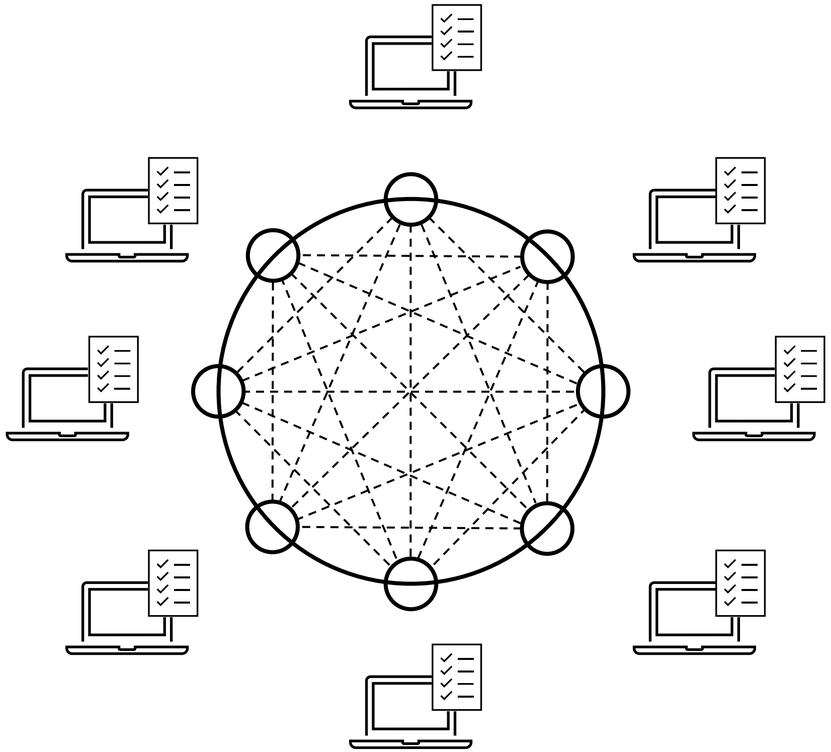
Trust Era

Public blockchain & distributed ledgers

- Speed
- Access & collaboration
- Trust

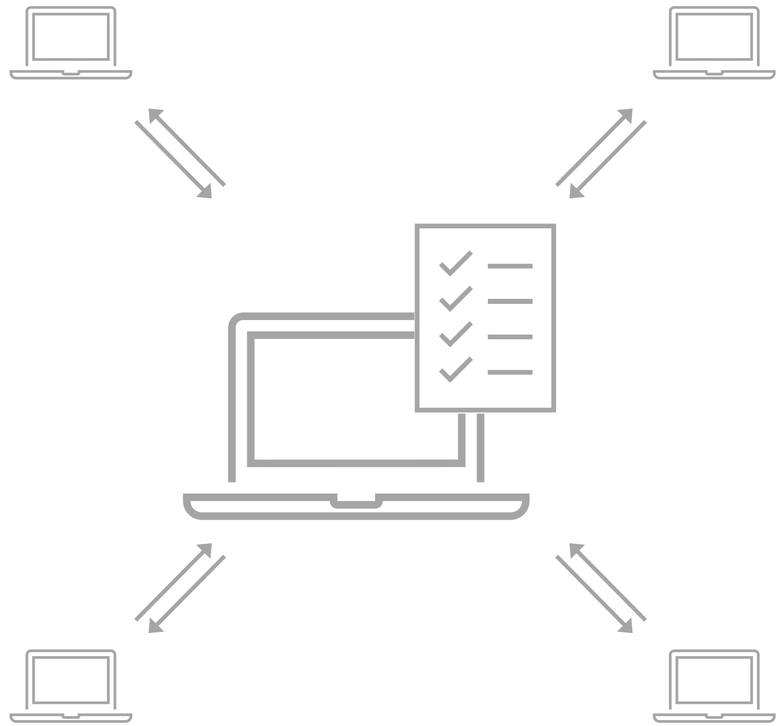
Distributed ledgers are a key component of Web 3.0 because of their qualities

DISTRIBUTED LEDGER



Entire network records and validates each transaction

CENTRALIZED LEDGER



Single authority verifies, records, and executes transactions

Distributed ledgers are a key component of Web 3.0 because of their qualities

No central point of failure to attack

Reliant on strong cryptography to prove data integrity and tamper resistance

Rules for the ledger are determined by a governing concept of some sort

Require a consensus mechanism to determine the rules for adding new transactions to the state of the ledger

Hedera is a third-generation public distributed ledger

2009

Bitcoin

Computational trust

Decentralized trust and programmable money

2014

Ethereum

Programability

Programmable smart contracts and digital assets

2018

Hedera

Trust at web scale

Architected from the ground up for new and existing applications running at scale

Hedera is a third-generation public distributed ledger

	1ST GENERATION	2ND GENERATION	3RD GENERATION
	Bitcoin Blockchain	Ethereum Blockchain	Hedera™
TRANSACTIONS PER SECOND	3+ TPS	12+ TPS	10,000+ TPS
AVERAGE TRANSACTION FEE	$22.57 USD	$19.55 USD	$0.0001 USD
AVERAGE TRANSACTION CONFIRMATION	10-60 MINUTES	10-20 SECONDS	3-5 SECONDS (W/ FINALITY)

- Cryptocurrency transactions. For Hedera, range shown for transactions not requiring a transaction record, but can receive a transaction receipt.
- Avg. Bitcoin tx fee from 6/26/20 - 9/24/20 from https://blockchair.com/bitcoin/charts/average-transaction-fee-usd?interval=3m
- Avg. Ethereum tx fee from 6/26/20 - 9/24/20 from https://blockchair.com/ethereum/charts/average-transaction-fee-usd?interval=3m

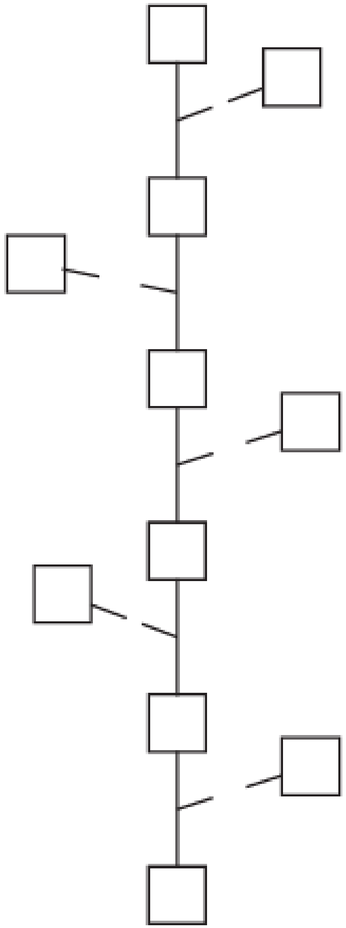
3rd Generation Public Ledgers

	polygon	Algorand		
Transactions per second	6,500* (claimed)	1,000† (approximate)	12+	**10,000+**
Average Fee (USD)	$0.0020 (variable)	0.001 Algo (fixed)	$19.55 (variable)	**$0.0001^ (fixed)**
Time to Confirmation	5 - 10 seconds (leader block creation)	5 seconds (leader block creation)	10 - 10 seconds (leader block creation)	**3-5 seconds (with finality)**
Energy use per transaction	90+ kWh	0.00534 kWh	102+ kWh	**0.00017 kWh**

THE HBAR FOUNDATION

* Actual TPS closer to 53 over a rolling 30 day period - https://thecoinweekly.com/not-so-fast-are-eth-killers-as-speedy-as-they-claim
^ Hedera Transaction fees - https://hedera.com/fees
† https://www.algorand.com/resources/algorand-announcements/algorand-2021-performance

TRADITIONAL BLOCKCHAINS



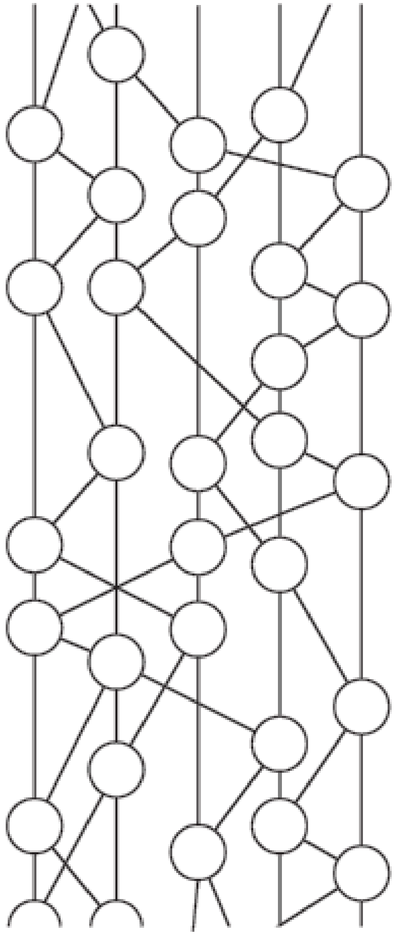
Proof-of-Work (PoW) typically has heavy electrical/computational requirements

No guaranteed consensus, just a “good enough” probability

Inefficient - “stale” blocks are pruned

Can be difficult to scale beyond ~1,000 tps

HASHGRAPH



Hashgraph is an **open-source** consensus algorithm and data structure

Uses a directed acyclic graph (DAG) and novel inventions

- Gossip about gossip
- Virtual voting

Hedera currently supports 10,000+ cryptocurrency transactions per second

Finality within 3-5 seconds

TRADITIONAL BLOCKCHAINS

HASHGRAPH

END USERS

INDIVIDUALS

ENTERPRISES

GOVERNMENTS

DEVICES

APPLICATION USE CASES

DATA INTEGRITY

TOKENIZATION

DECENTRALIZED IDENTITY

SUPPLY CHAIN

PAYMENTS

CONSENSUS SERVICE

TOKEN SERVICE

SMART CONTRACTS

PRIMARY NETWORK SERVICES

HEDERA GOVERNING COUNCIL

HASHGRAPH CONSENSUS

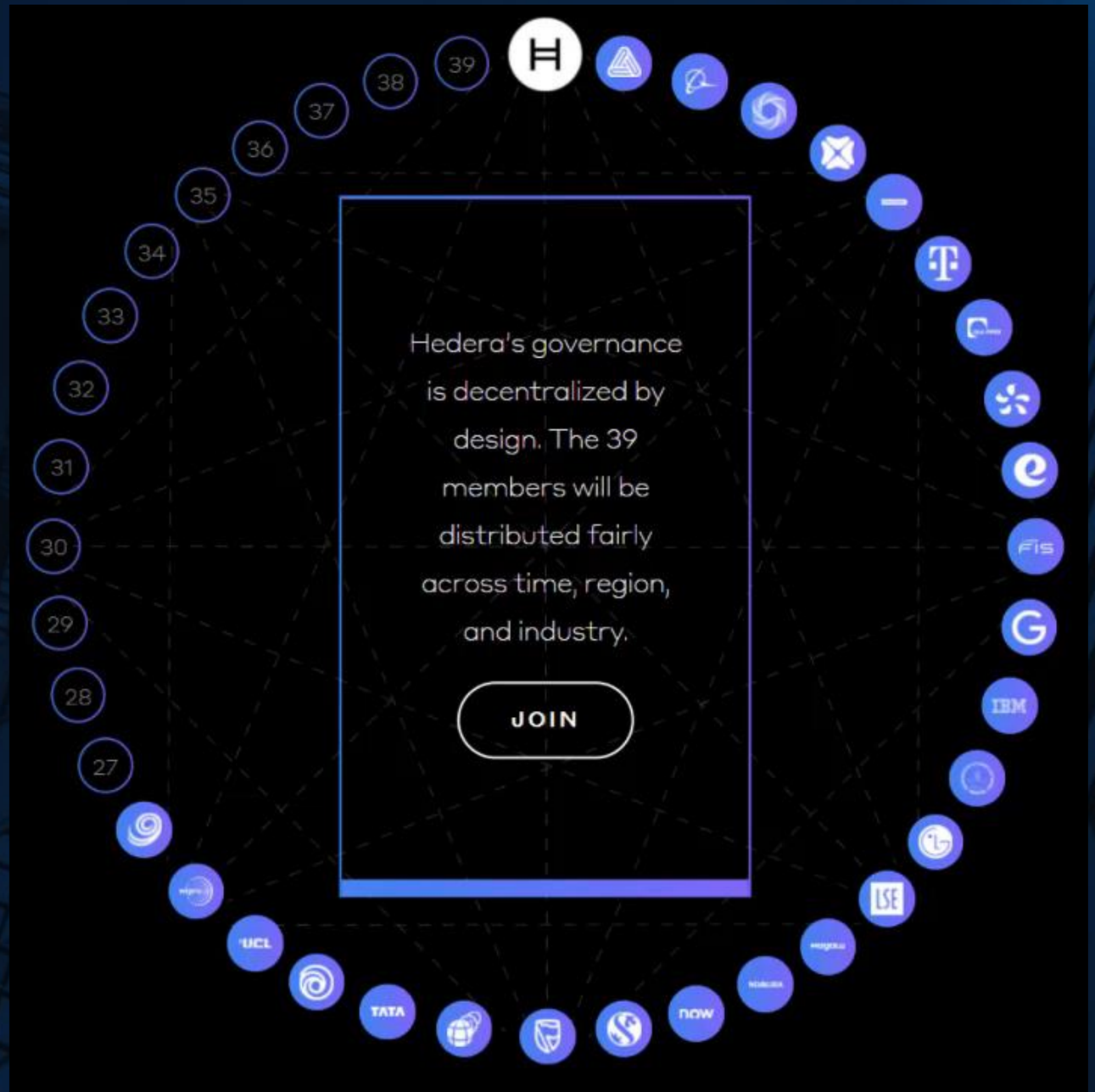
HEDERA MAINNET & MIRROR NODES

Hedera Governing Council

Up to **39** leading global organizations

20+ unique industries across **5** continents, touching all major markets

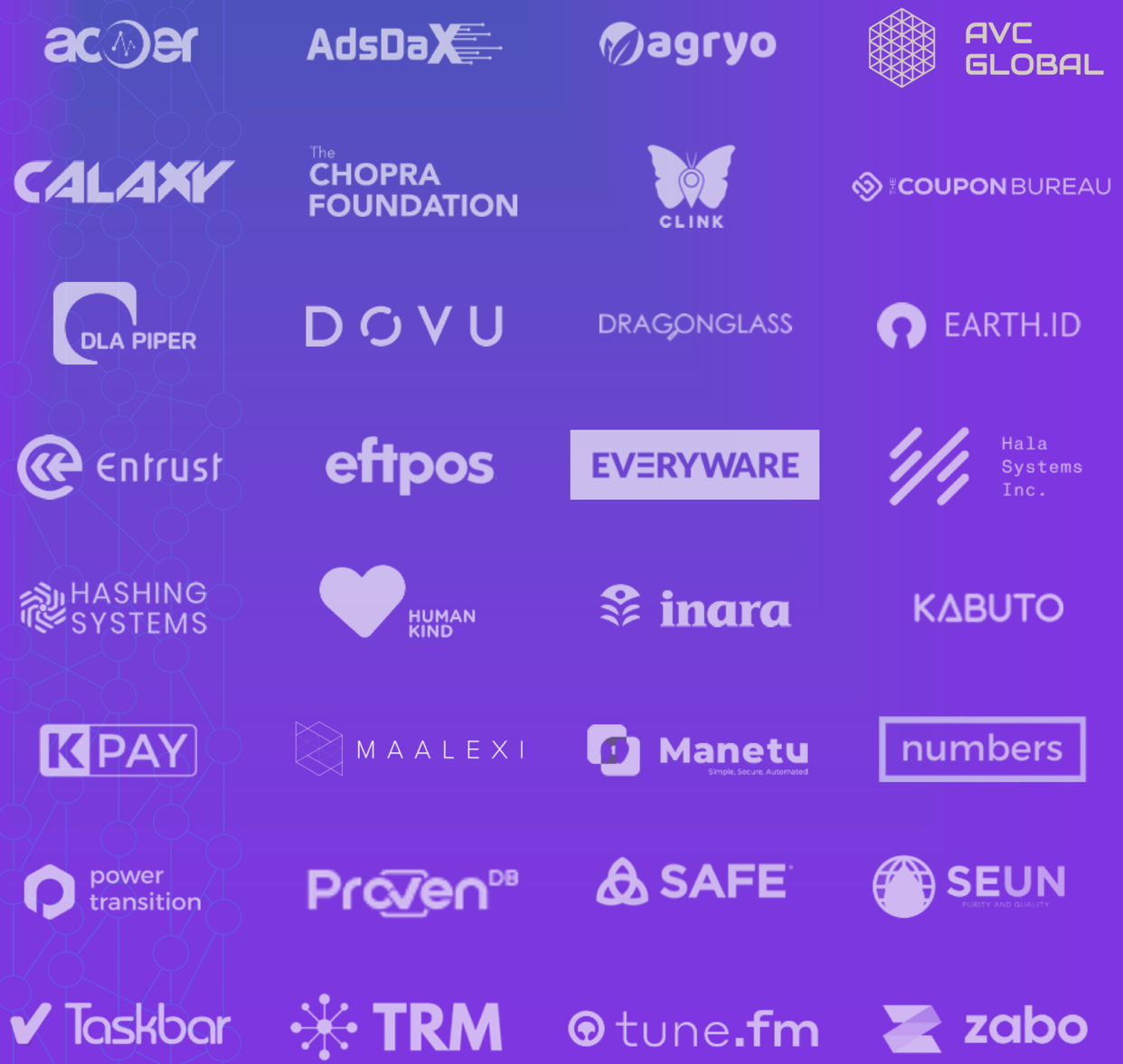
3-year maximum term, with up to 2 consecutive terms



2.6% influence per member (equal vote)

7 committees: Membership, Marketing, Finance, Audit, Technical Steering/Product Pricing, Legal & Regulatory

Every member required to run a network node



hedera.com

14,000+ DEVELOPERS

Attending global hackathons, meetups, and active in Discord.

100+ APPLICATIONS

In production on Hedera Mainnet, since open access on Sept. 2019

>7,000,000+ TXS/DAY

Far surpassing the daily transaction volume of Ethereum.

>2,350,000,000+ TXS TO DATE

Most used public DLT surpassing total transaction volume of Ethereum.

>830,000+ TOTAL ACCOUNTS ON MAINNET

66,700 accounts created in April 2022 alone.

With dApp days, you will learn how to build and deploy applications on Hedera!

Section 1:
Introduction to
Web 3 and Hedera

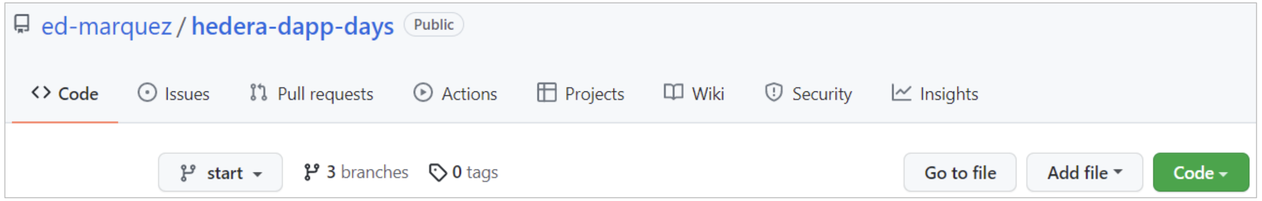
Section 2:
Buidl on Hedera

Section 3:
Enter NFT Giveaway
and Next Steps


LET’S BUIDL A COOL DAPP ON HEDERA!

BUIDL A DAPP ON HEDERA (1/4)

1. Go to the hedera-dapp-days repository in GitHub

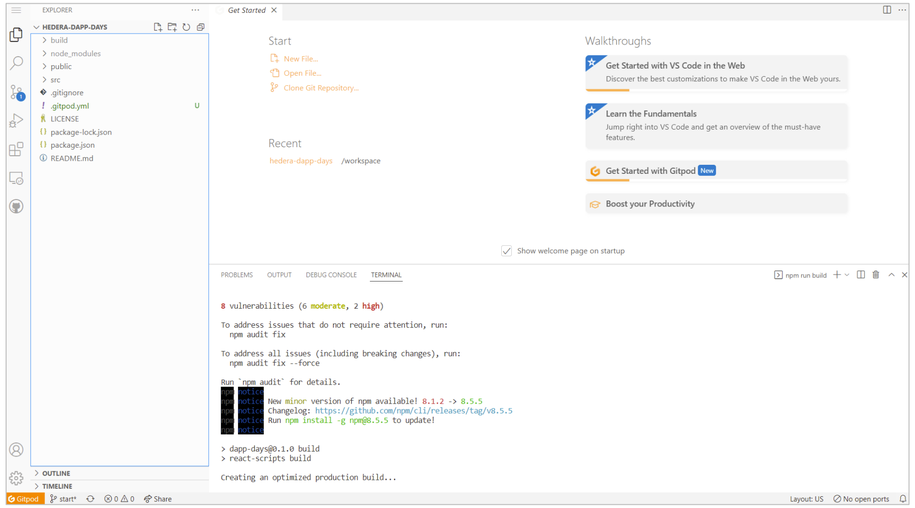


2. **Fork** the repository…

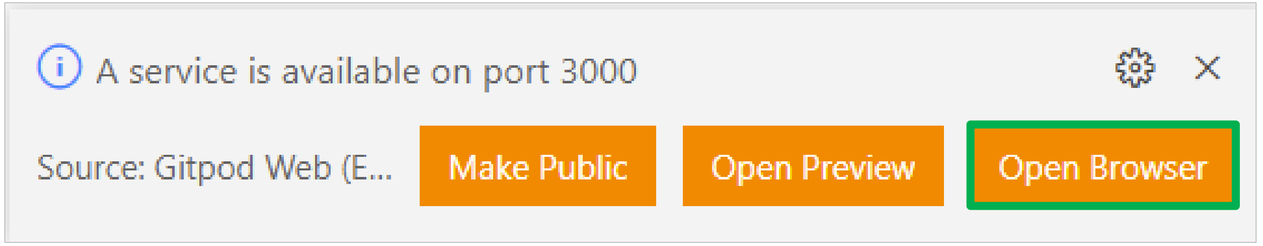
… and **open** your fork in Gitpod using a link that looks as follows:

gitpod.io#https://github.com/<GITHUB_UN>/hedera-dapp-days/tree/start

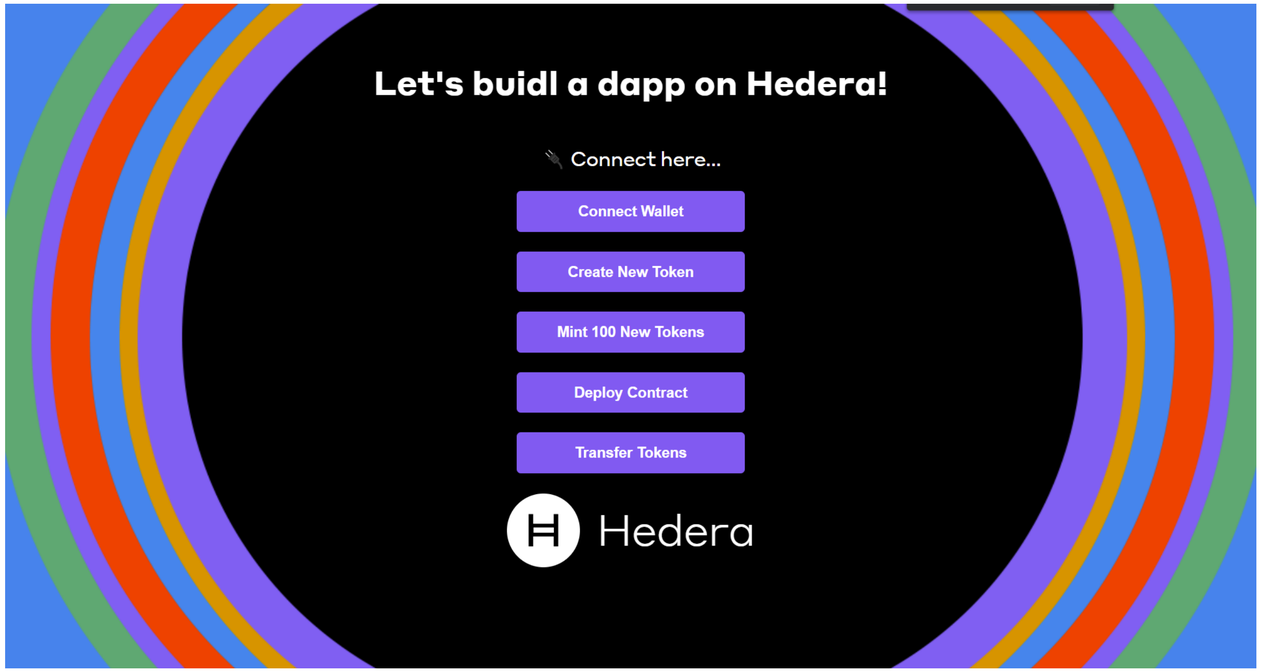
Where *<GITHUB_UN>* is your GitHub username



As Gitpod loads, you will see this prompt. Click ***Open Browser***



After that, you will see the template application in a new tab

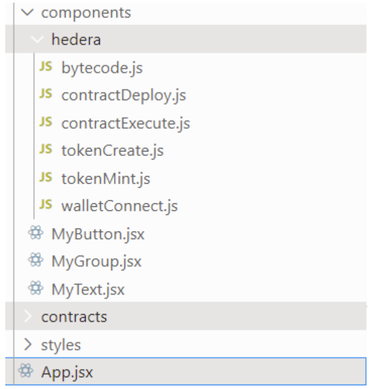


BUIDL A DAPP ON HEDERA (2/4)

3. If you're not too familiar with React, explore the project in Gitpod. Most of the relevant files for dAppDays are in the ***src*** directory

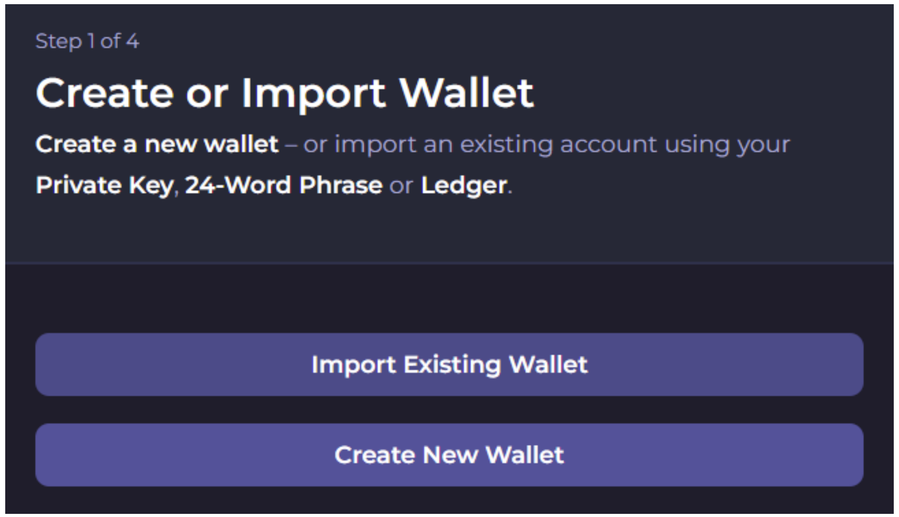
Files and directories of interest include:

- ***App.jsx***
- ***src/components***
- ***src/components/hedera***
- ***src/contracts***



4. Open the HashPack wallet extension to…

- Import your existing Hedera testnet account credentials, OR to..
- Create a new testnet account from HashPack



5. Code the Hedera components and the Solidity contract. All files you will code have boilerplate code as a starting point. Add the following:

5a. walletConnect.js

```
async function walletConnectFcn() {
    console.log(`\n=======================================`);
    console.log("- Connecting wallet...");

    let saveData = {
        topic: "",
        pairingString: "",
        privateKey: "",
        pairedWalletData: null,
        pairedAccounts: [],
    };
    let appMetadata = {
        name: "Hedera dApp Days",
        description: "Let's buidl a dapp on Hedera",
        icon: "https://raw.githubusercontent.com/ed-marquez/hedera-dapp-days/testing/src/assets/hederaLogo.png",
    };
    let hashconnect = new HashConnect();

    // First init and store the pairing private key for later (this is NOT your account private key)
    const initData = await hashconnect.init(appMetadata);
    saveData.privateKey = initData.privKey;
    console.log(`- Private key for pairing: ${saveData.privateKey}`);

    // Then connect, storing the new topic for later
    const state = await hashconnect.connect();
    saveData.topic = state.topic;
    console.log(`- Pairing topic is: ${saveData.topic}`);

    // Generate a pairing string, which you can display and generate a QR code from
    saveData.pairingString = hashconnect.generatePairingString(state, "testnet", false);

    // Find any supported local wallets
    hashconnect.findLocalWallets();
    hashconnect.connectToLocalWallet(saveData.pairingString);

    return [hashconnect, saveData];
}
```

BUIDL A DAPP ON HEDERA (3/4)

5b. *tokenCreate.js*

```
async function tokenCreateFcn(walletData, accountId) {
    console.log(`\n=====================================`);
    console.log(`- Creating HTS token...`);

    const hashconnect = walletData[0];
    const saveData = walletData[1];
    const provider = hashconnect.getProvider("testnet", saveData.topic, accountId);
    const signer = hashconnect.getSigner(provider);

    const url = `https://testnet.mirrornode.hedera.com/api/v1/accounts?account.id=${accountId}`;
    const mirrorQuery = await axios(url);
    const supplyKey = PublicKey.fromString(mirrorQuery.data.accounts[0].key.key);

    const tokenCreateTx = await new TokenCreateTransaction()
        .setTokenName("dAppDayToken")
        .setTokenSymbol("DDT")
        .setTreasuryAccountId(accountId)
        .setAutoRenewAccountId(accountId)
        .setAutoRenewPeriod(7776000)
        .setInitialSupply(400)
        .setDecimals(0)
        .setSupplyKey(supplyKey)
        .freezeWithSigner(signer);
    const tokenCreateSubmit = await tokenCreateTx.executeWithSigner(signer);
    const tokenCreateRx = await provider.getTransactionReceipt(tokenCreateSubmit.transactionId);
    const tId = tokenCreateRx.tokenId;
    const supply = tokenCreateTx._initialSupply.low;
    console.log(`- Created HTS token with ID: ${tId}`);

    return [tId, supply, tokenCreateSubmit.transactionId];
}
```

5c. *tokenMint.js*

```
async function tokenMintFcn(walletData, accountId, tId) {
    console.log(`\n=====================================`);
    const amount = 100;
    console.log(`- Minting ${amount} tokens...`);

    const hashconnect = walletData[0];
    const saveData = walletData[1];
    const provider = hashconnect.getProvider("testnet", saveData.topic, accountId);
    const signer = hashconnect.getSigner(provider);

    const tokenMintTx = await new TokenMintTransaction()
        .setTokenId(tId)
        .setAmount(amount)
        .freezeWithSigner(signer);
    const tokenMintSubmit = await tokenMintTx.executeWithSigner(signer);
    const tokenCreateRx = await provider.getTransactionReceipt(tokenMintSubmit.transactionId);
    const supply = tokenCreateRx.totalSupply;
    console.log(`- Tokens minted. New supply is ${supply}`);

    return [supply, tokenMintSubmit.transactionId];
}
```

5d. *contracts ➔ AssoTransHTS.sol*

```
contract AssoTransHTS is HederaTokenService {

    address tokenAddress;

    constructor(address _tokenAddress) public {
        tokenAddress = _tokenAddress;
     }

    function tokenAssoTrans(int64 _amount) external {
        int response1 = HederaTokenService.associateToken(address(this), tokenAddress);

        int response2 = HederaTokenService.transferToken(tokenAddress, msg.sender, address(this), _amount);
    }
}
```

***…and** contractDeploy.js*

```
async function contractDeployFcn(walletData, accountId, tokenId) {
    console.log(`\n=====================================`);
    console.log(`- Deploying smart contract on Hedera...`);

    const hashconnect = walletData[0];
    const saveData = walletData[1];
    const provider = hashconnect.getProvider("testnet", saveData.topic, accountId);
    const signer = hashconnect.getSigner(provider);

    //Create a file on Hedera and store the hex-encoded bytecode
    const fileCreateTx = await new FileCreateTransaction()
        .setContents(bytecode)
        .freezeWithSigner(signer);
    const fileSubmit = await fileCreateTx.executeWithSigner(signer);
    const fileCreateRx = await provider.getTransactionReceipt(fileSubmit.transactionId);
    const bytecodeFileId = fileCreateRx.fileId;
    console.log(`- The smart contract bytecode file ID is: ${bytecodeFileId}`);

    // Create the smart contract
    const contractCreateTx = await new ContractCreateTransaction()
        .setBytecodeFileId(bytecodeFileId)
        .setGas(3000000)
        .setConstructorParameters(
            new ContractFunctionParameters().addAddress(tokenId.toSolidityAddress())
        )
        .freezeWithSigner(signer);
    const contractCreateSubmit = await contractCreateTx.executeWithSigner(signer);
    const contractCreateRx = await provider.getTransactionReceipt(contractCreateSubmit.transactionId);
    const cId = contractCreateRx.contractId;
    const contractAddress = cId.toSolidityAddress();
    console.log(`- The smart contract ID is: ${cId}`);
    console.log(`- The smart contract ID in Solidity format is: ${contractAddress} \n`);

    return [cId, contractCreateSubmit.transactionId];
}
```

BUIDL A DAPP ON HEDERA (4/4)

5e. *contractExecute.js*

```
async function contractExecuteFcn(walletData, accountId, tokenId, contractId) {
    console.log(`\n=====================================`);
    console.log(`- Executing the smart contract...`);

    const hashconnect = walletData[0];
    const saveData = walletData[1];
    const provider = hashconnect.getProvider("testnet", saveData.topic, accountId);
    const signer = hashconnect.getSigner(provider);

    //Execute a contract function (transfer)
    const contractExecTx = await new ContractExecuteTransaction()
        .setContractId(contractId)
        .setGas(3000000)
        .setFunction("tokenAssoTrans", new ContractFunctionParameters().addInt64(50))
        .freezeWithSigner(signer);
    const contractExecSign = await contractExecTx.signWithSigner(signer);
    const contractExecSubmit = await contractExecSign.executeWithSigner(signer);
    const contractExecRx = await provider.getTransactionReceipt(contractExecSubmit.transactionId);
    console.log(`- Token transfer from Operator to contract: ${contractExecRx.status.toString()}`);

    const bCheck = await signer.getAccountBalance();
    console.log(
        `- Operator balance: ${bCheck.tokens._map.get(tokenId.toString())} units of token ${tokenId}`
    );

    return contractExecSubmit.transactionId;
}
```

6. Next steps **for you…**

- Add more functionality to your application. Here are a few ideas:
  - Send *hbar* to and from the contract
  - Create tokens directly via the contract (*hint: HTS precompiles*)
  - Exchange *hbar* and HTS tokens via the contract
  - Add error handling to the React application

- Try libraries and tools to speed up your development:
  - Run a local Hedera local network for testing
  - Use hether.js / ether.js / web3.js with Hedera

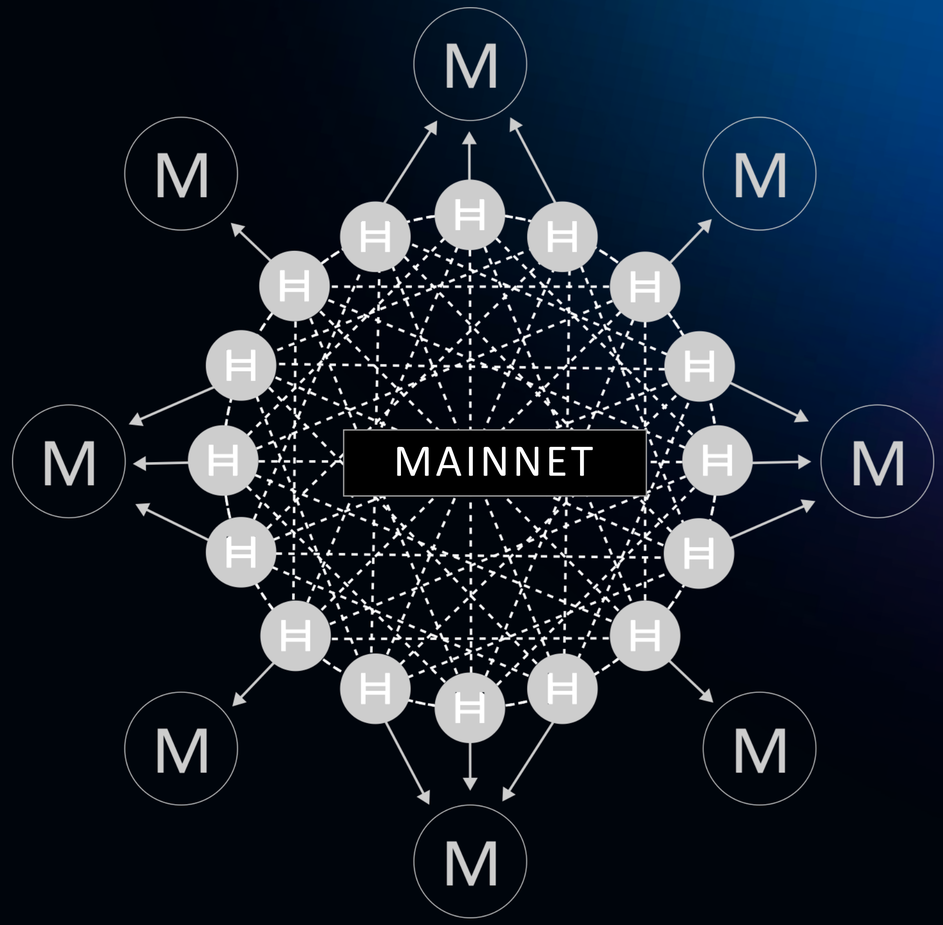
- Check out the documentation and try:
  - Examples
  - Tutorials
  - Demo applications
  - SDK implementations

- Join another Hedera dApp Day

HEDERA MAINNET & MIRRORNET

MAINNET

- Can submit HAPI (Hedera API) transactions to the Hedera network
- Contributes to consensus on transactions
- Creates events on the Hedera network
- Requires HBAR cryptocurrency payment for transactions & queries

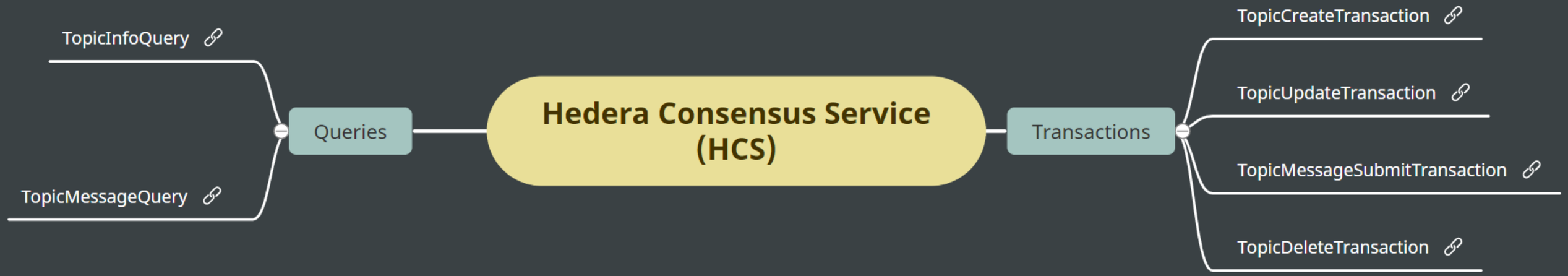


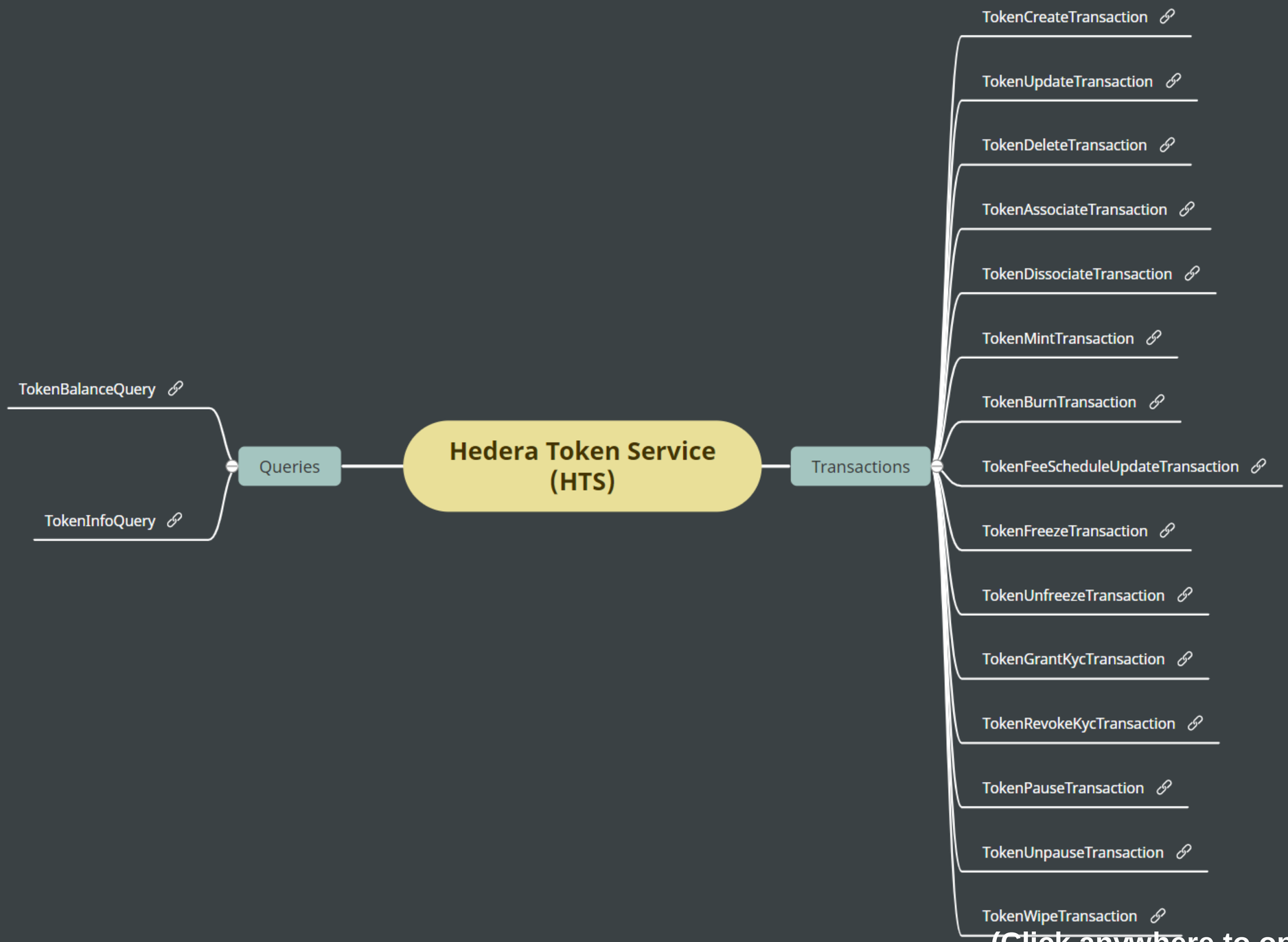
MIRRORNET

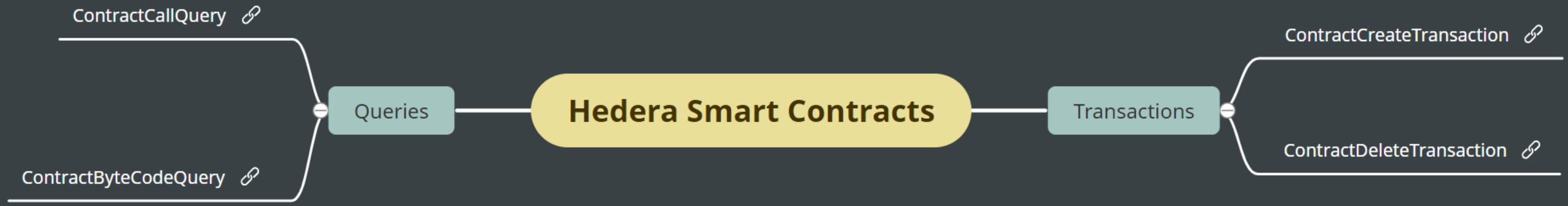
- Maintains a history of some or all the Hedera network state and ledger of transactions
- Value-added services (managed read-only node, etc.)
- Enables analytical insight into an application's state / transactions
- Publish and subscribe capabilities

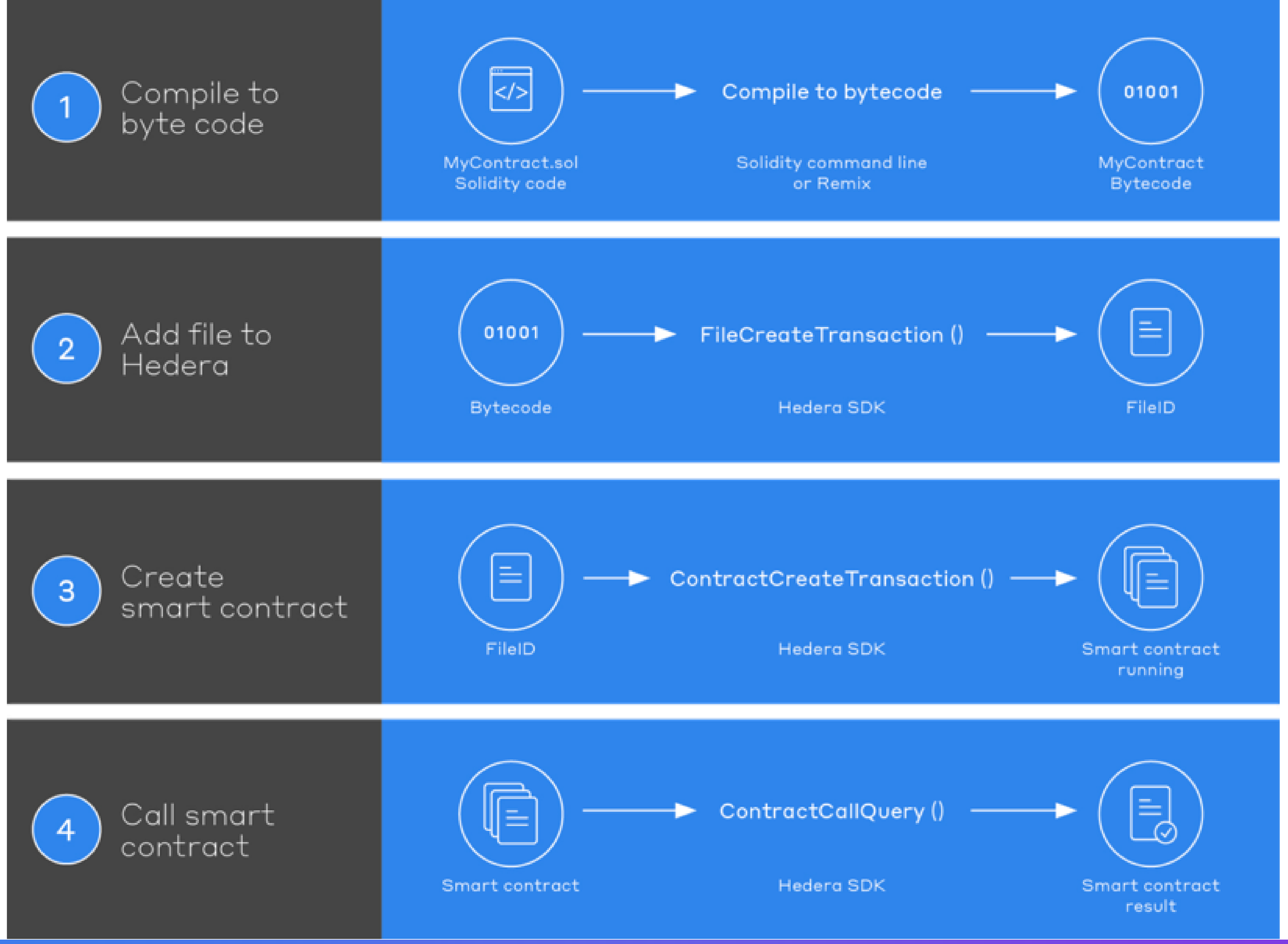
hedera.com

Hedera™ Hashgraph








1
Compile to byte code
MyContract.sol
Solidity code
Compile to bytecode
Solidity command line or Remix
01001
MyContract Bytecode
2
Add file to Hedera
01001
Bytecode
FileCreateTransaction ()
Hedera SDK
FileID
3
Create smart contract
FileID
ContractCreateTransaction ()
Hedera SDK
Smart contract running
4
Call smart contract
Smart contract
ContractCallQuery ()
Hedera SDK
Smart contract result

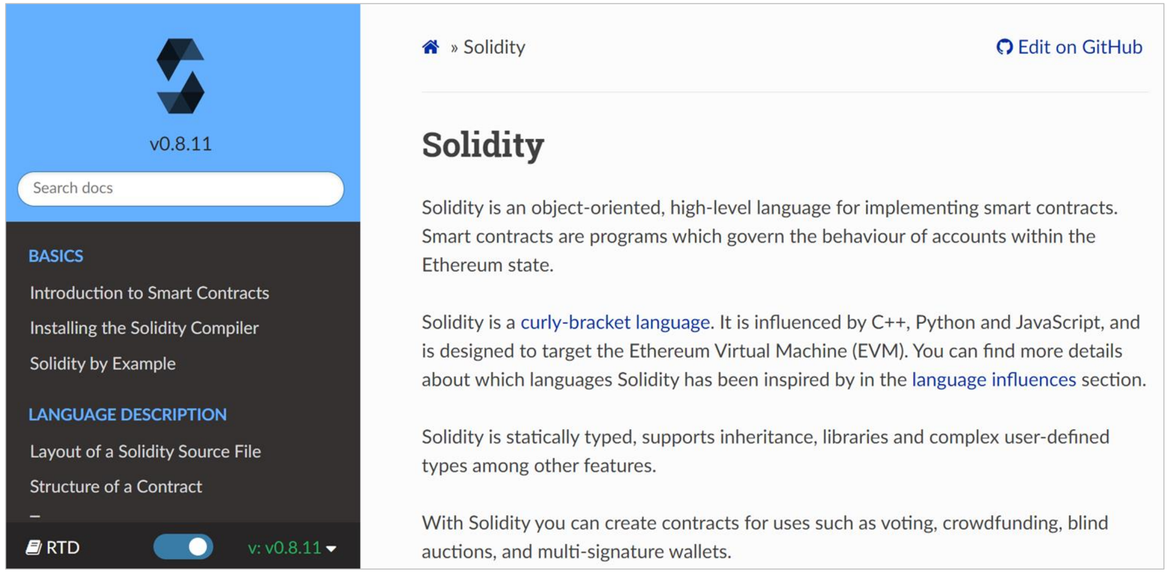
Solidity is a contract-oriented, high-level programming language

Contracts and inheritance similar to OOP libraries

Contracts are compiled and output:

- Bytecode (EVM instructions)
- ABI definition (contract's interface e.g., inputs, return values, etc.)

Solidity documentation



Bytecode


```
"data": {
    "bytecode": {
        "functionDebugData": {},
        "generatedSources": [],
        "linkReferences": {},
        "object": "6080604052348015610010576000
        "opcodes": "PUSH1 0x80 PUSH1 0x40 MSTO
        "sourceMap": "75:335:0:-:0;;;;;;;;;;;;
```


ABI Definition


```
[
    {
        "inputs": [
            {
                "internalType": "string",
                "name": "_name",
                "type": "string"
            },
            {
                "internalType": "uint256",
                "name": "_mobileNumber",
                "type": "uint256"
            }
        ],
        "name": "addMobileNumber",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "string",
                "name": "_name",
                "type": "string"
            }
        ],
        "name": "getMobileNumber",
        "outputs": [
            {
```


Ethereum virtual machine (EVM) executes compiled smart contract (bytecode)

Executing smart contract functions costs **gas**, which is the unit of account for execution costs

Gas reflects the cost necessary to pay for computing resources consumed by execution of smart contract

Each instruction in EVM and persistent storage write cost gas

Gas is measured and paid during the execution of each contract

Hedera processes up to:

- 15 million gas/second network-wide
- 4 million gas/second per contract call

Total Gas (non-Hedera Service transaction) = Intrinsic Gas + EVM Operation Gas

Total Gas (Hedera Service transaction) = Intrinsic Gas + EVM Operation Gas + Hedera Service Gas

Contracts in Solidity are like classes in OOP and they contain data and **functions**

Persistent data is stored in state variables

Functions can modify these variables

Contract execution is paid with gas

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract Storage {
    uint256 number;

    function store(uint256 num) public {
        number = num;
    }

    function retrieve() public view returns (uint256){
        return number;
    }
}
```

Functions can have different types of visibility:

- Public
- Private
- External
- Internal

Function with a uint256 parameter and no return value:

```
function store(uint256 num) public {
    number = num;
}
```

Function with no parameter and a with a return value:

```
function retrieve() public view returns (uint256){
    return number;
}
```

https://solidity-by-example.org/

Solidity is a statically typed language, so the **type** of each variable (state and local) must be specified

Address: Holds a 20-byte value

Integers: Signed and unsigned integers of various sizes

- *uint* and *int* are aliases for uint256 and int256, respectively

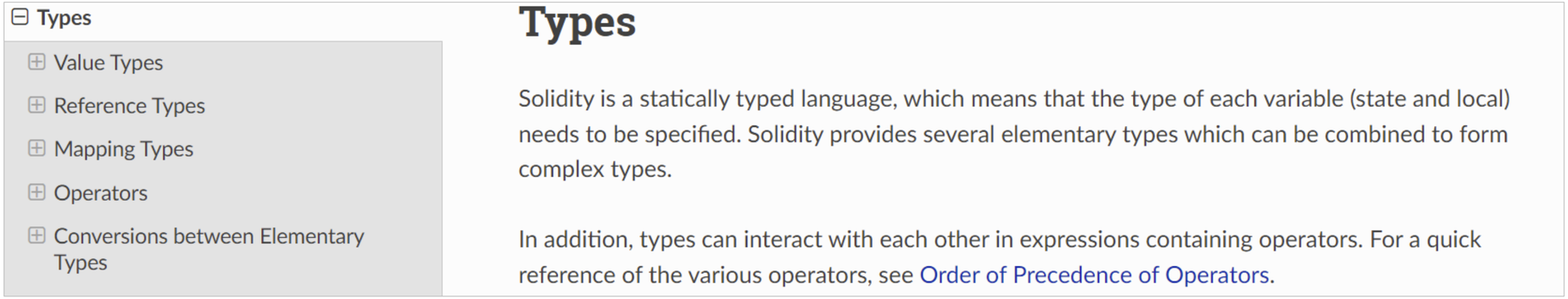
Mappings: Declared using the syntax

`mapping(KeyType => ValueType) VariableName`

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract MyFirstContract {

mapping (string => uint) public phoneNumbers;

function addMobileNumber(string memory _name, uint _mobileNumber) public {
        phoneNumbers[_name] = _mobileNumber;
    }

function getMobileNumber(string memory _name) public view returns (uint) {
        return phoneNumbers[_name];
    }

}
```

Other types include:

- String
- Booleans
- Byte arrays
- Enums



Read more about types in Solidity

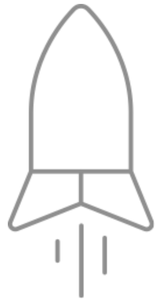
With dApp days, you will learn how to build and deploy applications on Hedera!

Section 1:
Introduction to
Web 3 and Hedera

Section 2:
Buidl on Hedera

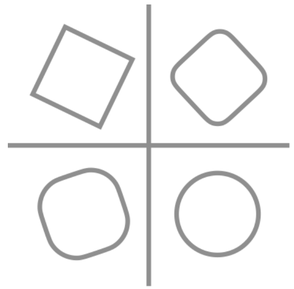
Section 3:
Enter NFT Giveaway
and Next Steps

Hedera helps you meet strict requirements in the following areas…

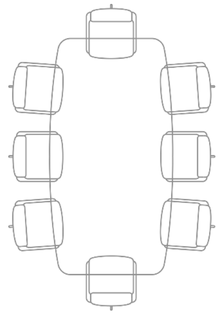


Performance

Security

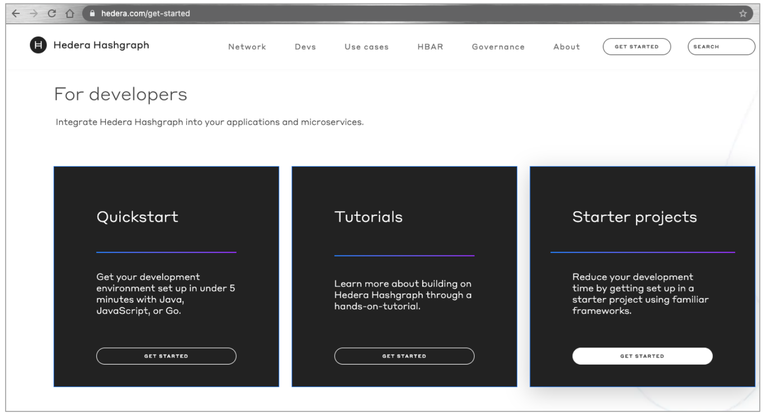


Stability

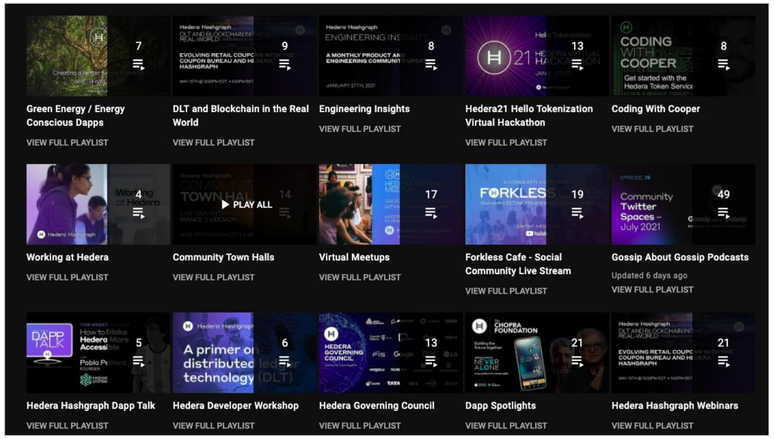


Governance

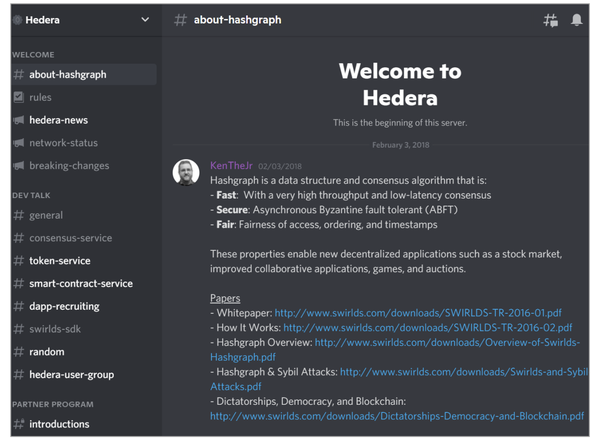
Continue learning with tutorials, videos, and conversations!



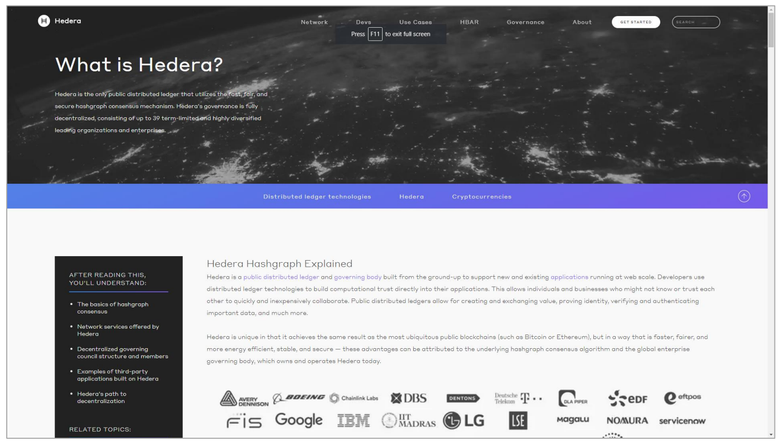
hedera.com/get-started



Hedera YouTube Channel



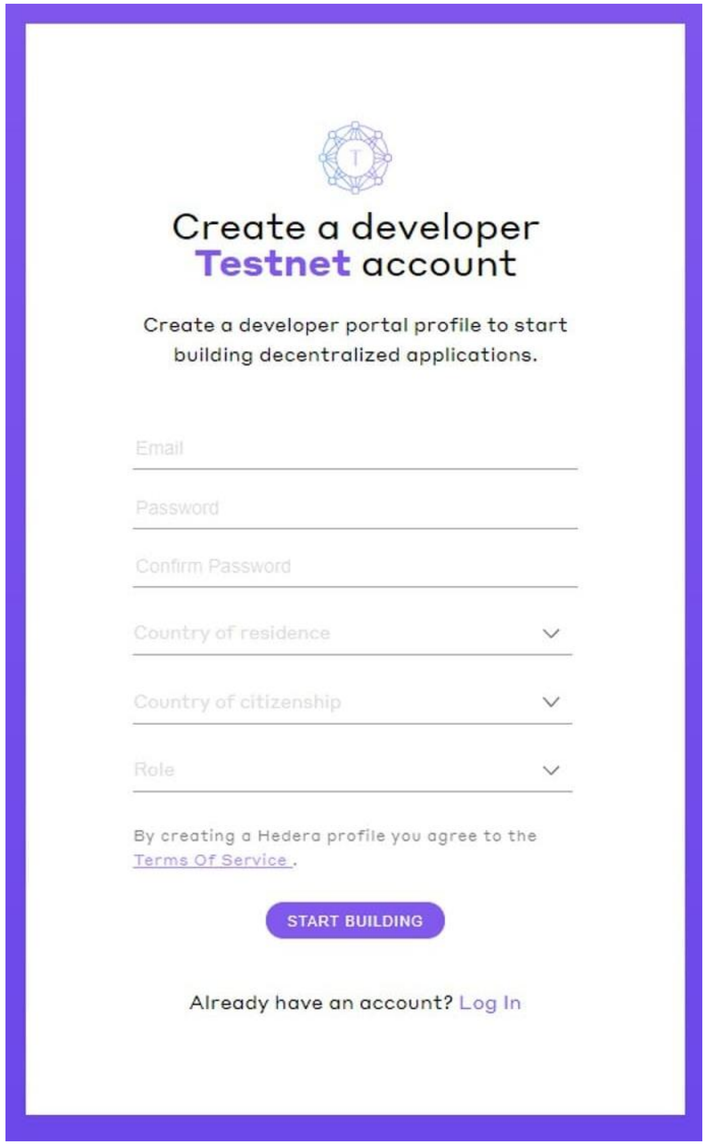
Join Developer Discord



Hedera Learning Center

Office Hours with Ed!

Learn more and start building web 3, but first get a testnet account!

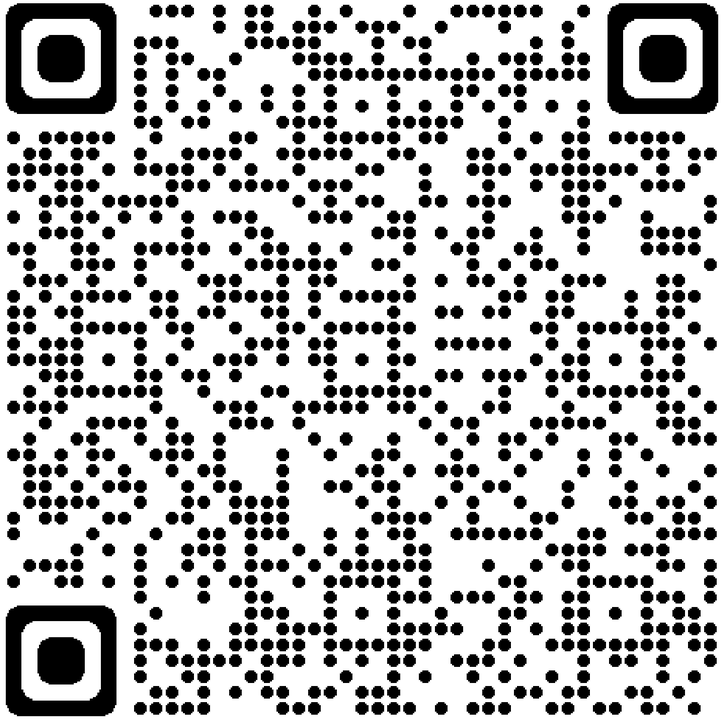


https://portal.hedera.com/register


NFT
GIVEAWAY!!!

FEEDBACK AND OPEN Q&A

tinyurl.com/dappdays-survey



FEEDBACK AND OPEN Q&A

Hedera™ Hashgraph

THE TRUST LAYER OF THE INTERNET

/ed-marquez

@ed__marquez

hedera.com