Blockchain Development on Metis Part II

Fast Track Course - 2024

METIS

Program Details

(Tentative)

Module 1: Introduction to Blockchain and Ethereum

Module 2: Solidity and Smart Contract Development

Module 3: Advanced Solidity Concepts

Module 4: Development Tools

Module 5: Frontend Integration

Module 6: Introduction to Metis L2 Network

Module 7: Advanced Metis L2 Topics

Advanced Solidity

Functions, Visibility, and Error Handling

Functions are the executable units of code in Solidity. They can have different levels of visibility:

- public: can be called internally and externally.
- private: can only be called within the contract they are defined.
- internal: can only be accessed within the contract and derived contracts.
- external: can only be called from outside the contract.

Modifiers are used to change the behavior of functions. They can be used for checking conditions before executing the function.

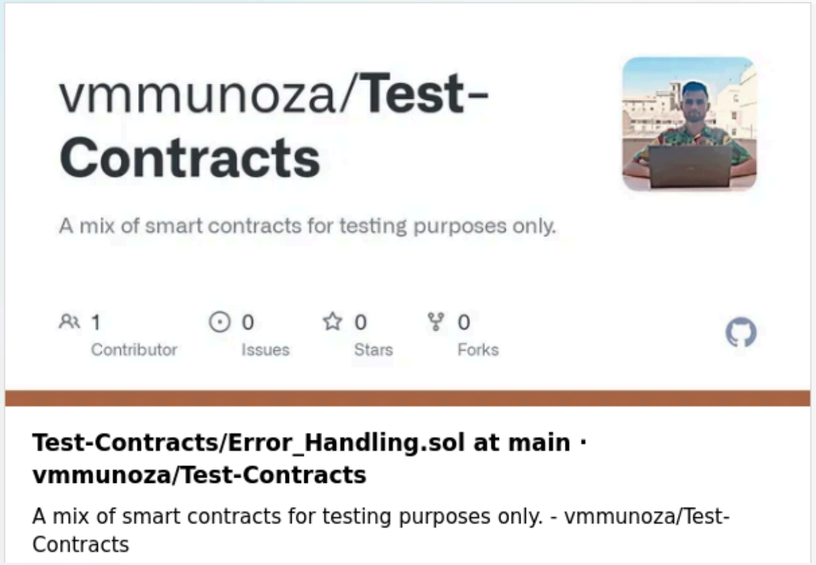
Events and Logging
Events are used to log information on the blockchain. They are useful for providing feedback and allowing off-chain applications to listen to specific events.

Error Handling and Exceptions
Solidity provides several ways to handle errors and exceptions:

- require: for conditions that must be met.
- assert: for internal errors and invariants.
- revert: to abort execution and revert state changes.

Advanced Solidity

Functions, Visibility, and Error Handling



- Functions, Visibility, and Modifiers:
  - incrementCount: A public function to increment the count.
  - resetCount: An external function to reset the count, restricted to the owner using the onlyOwner modifier.
  - getSpecialNumber: A private function that returns a special number, for internal use only.
  - internalFunction: An internal function that returns the owner's address.

- Events and Logging:
  - The CountIncremented event is emitted whenever the count is incremented, allowing external applications to listen for this event.

- Error Handling and Exceptions:
  - The errorHandlingExample function demonstrates error handling using require, assert, and revert. It ensures that the input value is greater than zero, checks for overflow using assert, and reverts if the result equals 42.

Advanced Solidity

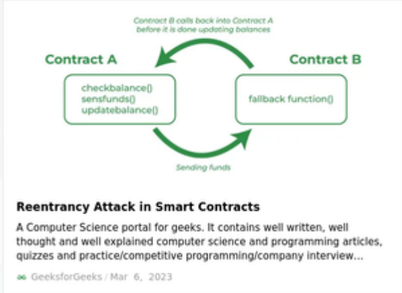
Vulnerabilities and Best Practices

Common Vulnerabilities

Vulnerability	Description	Example	Prevention
Reentrancy Attack	Occurs when a function makes an external call to another contract before updating its state. The called contract can re-enter the original function, potentially causing unexpected behavior.	A reentrant call in a withdrawal function can drain funds from the contract.	Use the `Checks-Effects-Interactions` pattern. Implement reentrancy guards using the `nonReentrant` modifier from OpenZeppelin's ReentrancyGuard.
Integer Overflow/Underflow	Happens when arithmetic operations exceed the maximum or minimum size of the integer type, causing unexpected results.	If `uint8 x = 255`, then `x + 1` results in `0`.	Use SafeMath library from OpenZeppelin to perform arithmetic operations safely.

Advanced Solidity

```solidity
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.22 <0.9.0;
/// @title A contract for demonstrate payable functions and addresses
/// @author Jitendra Kumar
/// @notice For now, this contract just show how payable functions and addresses can receive ether into the contract
contract Reentrancy {
    mapping(address => uint) public balance;

    function deposit() public payable {
        balance[msg.sender] += msg.value;
    }

    function withdraw() public payable {
        require(balance[msg.sender] >= msg.value);
        payable(msg.sender).transfer(msg.value);
        balance[msg.sender] -= msg.value;
    }
}
```



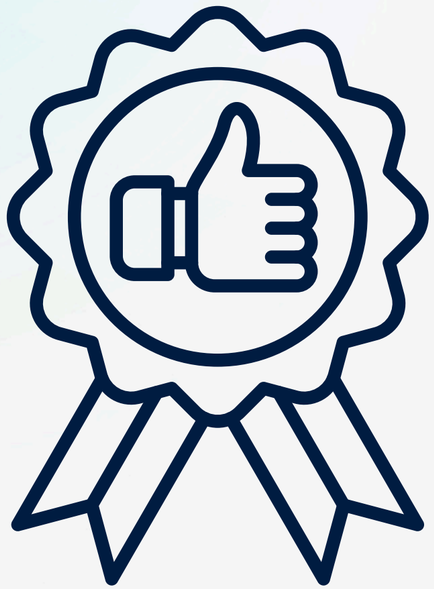

```solidity
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.22 <0.9.0;
/// @title A contract for demonstrate Reentrancy Attack
/// @author Jitendra Kumar
/// @notice For now, this contract just show how to protect Smart Contracts against a Reentrancy Attack
contract Reentrancy {

    mapping(address => uint) public balance;
    bool public reentrancyLock;

    function deposit() public payable {
        require(!reentrancyLock);
        reentrancyLock = true;
        balance[msg.sender] += msg.value;
        reentrancyLock = false;
    }

    function withdraw() public payable{
        require(balance[msg.sender] >= msg.value);
        payable(msg.sender).transfer(msg.value);
        balance[msg.sender] -= msg.value;
    }
}
```

reentrancylock: This boolean variable acts as a lock to prevent reentrancy. When a user initiates a deposit, the lock is set to true, ensuring that no other function can be executed until the current one completes. After the deposit operation, the lock is reset to false.

By using the reentrancylock, the contract ensures that no other function can be executed while a user's transaction is in progress.

Advanced Solidity

Vulnerabilities and Best Practices



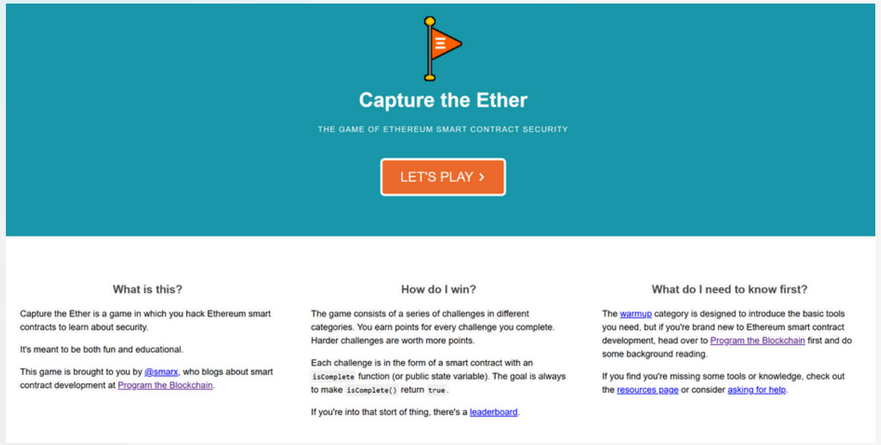
Best Practices

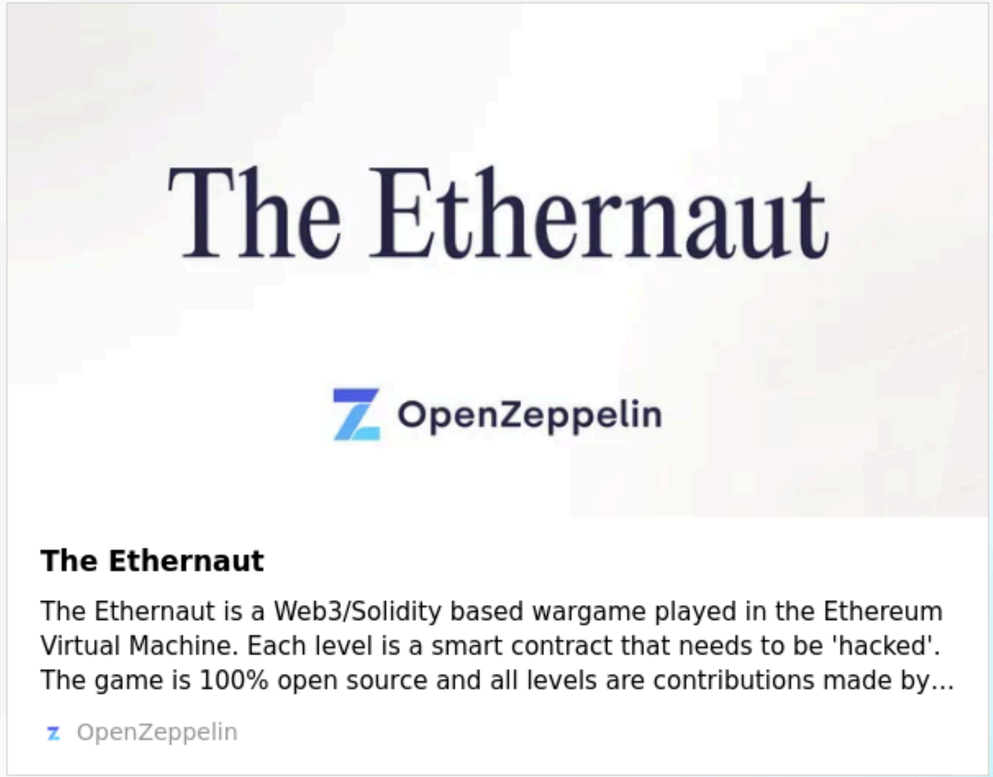
Best Practice	Description	Example	Benefits
Follow the Checks-Effects-Interactions Pattern	Ensure that all checks and state changes are performed before any external interactions.		Minimizes the risk of reentrancy attacks.
Use Established Libraries	Use established libraries for common functionalities.	OpenZeppelin provides secure implementations of common contracts and utilities.	Reduces the risk of introducing vulnerabilities.
Limit External Calls	Minimize the number of external calls your contract makes.		Reduces the attack surface for reentrancy and other attacks.
Implement Access Control	Restrict sensitive functions to authorized users.	Use `onlyOwner` modifier to restrict access.	Prevents unauthorized access and misuse of functions.
Regular Audits and Testing	Regularly audit your codebase and perform thorough testing, including unit tests and integration tests.		Identifies and fixes vulnerabilities before deployment.
Use SafeMath Library	Utilize libraries like OpenZeppelin's SafeMath for safe arithmetic operations.		Prevents integer overflow and underflow issues.

Advanced Solidity

Vulnerabilities and Best Practices

https://capturetheether.com





Development Tools

Feature	Foundry	Hardhat	Truffle
Overview	Fast, modular Ethereum development framework focused on Solidity testing.	Flexible, plugin-based development environment for Ethereum.	Comprehensive Ethereum development framework with an integrated suite of tools.
Key Features	- High-performance	- Extensible with plugins	- Integrated development suite
	- Tests written in Solidity	- Built-in testing environment	- Built-in testing and deployment
	- Advanced testing (e.g., fuzz testing)	- Network management capabilities	- Network management
Testing Framework	Built-in advanced testing tools	Mocha/Chai	Mocha/Chai
Plugin System	Growing collection of plugins	Extensive plugin system	Extensive plugin system
Performance	High	Moderate	Moderate
User Interface	Command line interface	Command line interface	Command line interface and graphical
Debugger	Advanced debugging tools	Built-in Solidity debugger	Built-in debugger
Script Execution	Supports custom scripts in Solidity	Supports custom scripts in JavaScript/TypeScript	Supports custom scripts in JavaScript
Ecosystem Integration	Integrated with Forge, Cast, Anvil	Integrated with Ethers.js, Waffle	Integrated with Ganache

Development Tools

Deprecated Dev

Old Dev

New Dev

Hardhat and Deployment on Metis

Hands-on: Cookbook Interactive Library

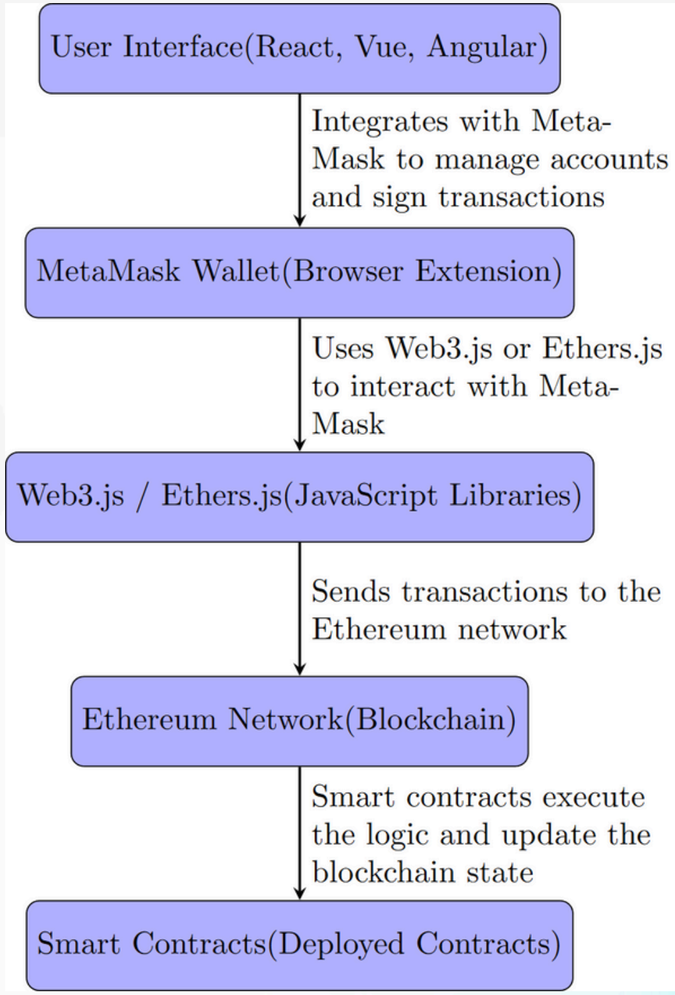
Find any Smart Contract

The easiest way for web3 developers to find documented smart contracts, solidity libraries, and discover protocols. Easily integrate with ChainIDE, Remix, Hardhat and Foundry.

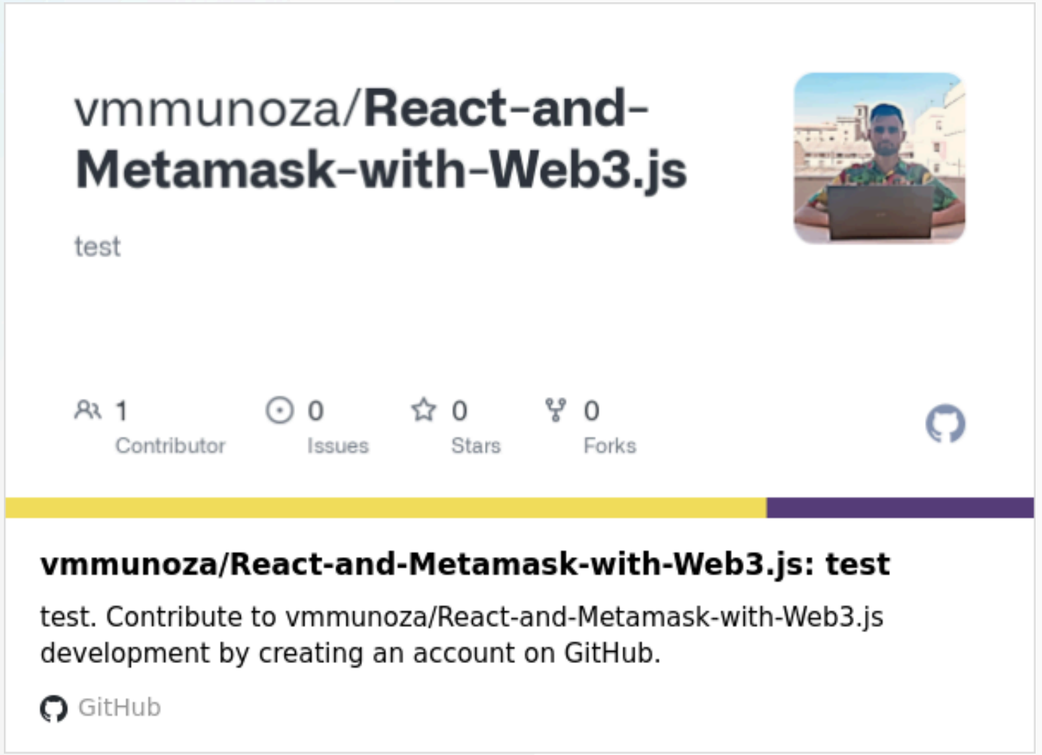
cookbook.dev

Front-End and Wallet Interaction

- **User Interface: The front end built using frameworks like React, Vue, or Angular.**
- **MetaMask Wallet: The browser extension used to manage blockchain accounts and sign transactions.**
- **Web3.js / Ethers.js: JavaScript libraries to interact with the Ethereum blockchain and MetaMask.**
- **Ethereum Network: The blockchain where transactions are sent and processed.**
- **Smart Contracts: The deployed contracts on the Ethereum network that execute logic and update the state.**



Front-End and Wallet Interaction

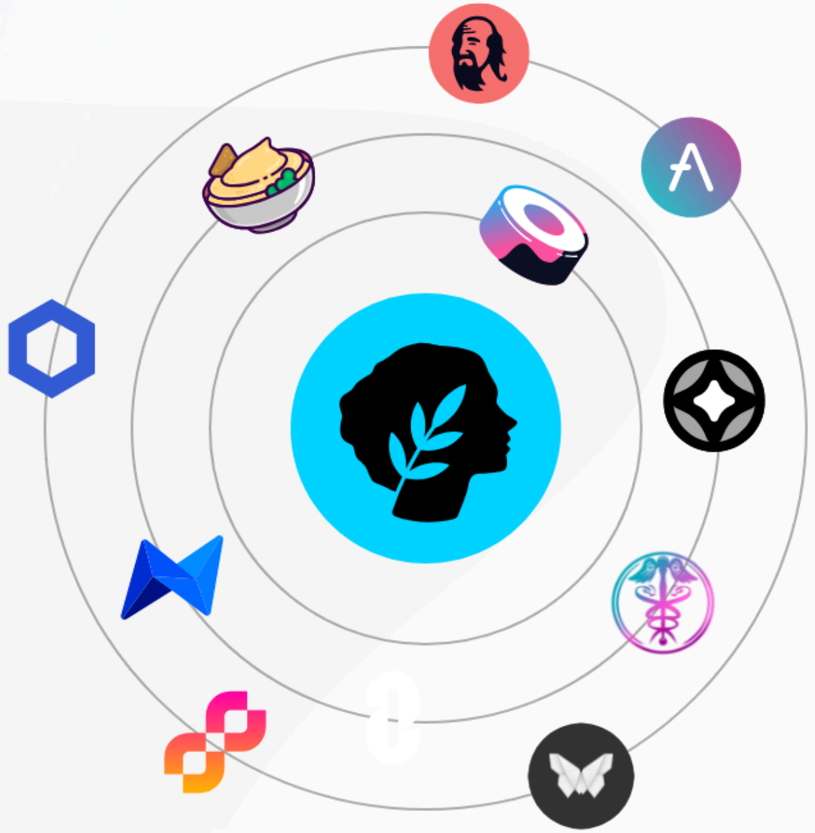


Homework

Gain hands-on experience with deploying a smart contract on the Metis network and creating a decentralized application (dApp) that interacts with it!

Using Hardhat, you will write and deploy a simple storage contract. Then, leveraging React and Web3.js, you will build a user interface that allows users to connect their MetaMask wallet, read the stored value from the blockchain, and update it.

Building on Metis L2



Rollups

What are Rollups?
Rollups are Layer 2 (L2) scaling solutions that bundle multiple transactions into a single batch, which is then submitted to the Ethereum mainnet.

Types of Rollups:

- Optimistic Rollups: Assume transactions are valid and only perform computation if fraud is detected through challenge periods.
- ZK-Rollups (Zero-Knowledge Rollups): Use zero-knowledge proofs to validate transactions off-chain and submit proof to the mainnet for verification.

How Do Rollups Work?
Transaction Batching: Rollups collect and bundle multiple transactions off-chain.

Data Compression: Transactions are compressed and only essential data is posted to the mainnet.

Verification: Depending on the type, either through fraud proofs (Optimistic) or cryptographic proofs (ZK-Rollups).

Rollups

A rollup-centric ethereum roadmap

What would a rollup-centric ethereum roadmap look like? Last week the Optimism team announced the...

✨ Fellowship of Ethereum Magicians / Oct 2, 2020

Rollup-Centric Roadmap (2024 version) (mike+stokes version)...

Rollup-Centric Roadmap (2024 version) (mike+stokes version) (From The Vault)...

HackMD

What is Metis?

OPTIMISTIC ROLLUPS SCALING ETHEREUM

Metis is an EVM-equivalent Optimistic Rollup built to scale Ethereum and onboard real-world use cases, besides enhancing existing ones.

Metis offers fast transactions, low gas fees, and allows developers to build on an easy-to-code blockchain.

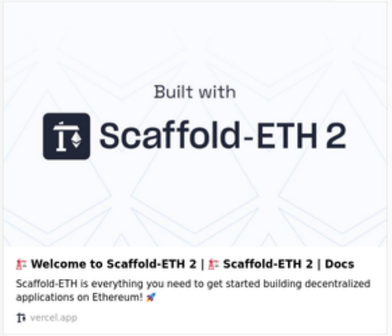
Users, developers, and enterprises can benefit from enhanced security, enhanced capital efficiency, and progressive decentralization through our sequencer technology.

METIS

Boilerplate

METIS

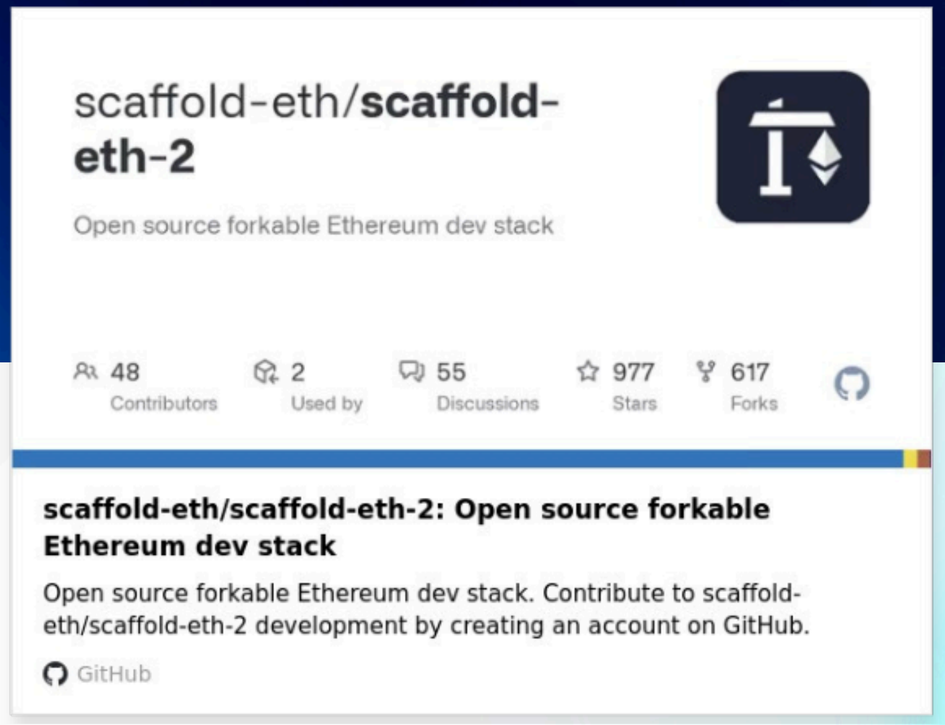
Pre-configured template or starting point for building decentralized applications (dApps) on a specific blockchain network.



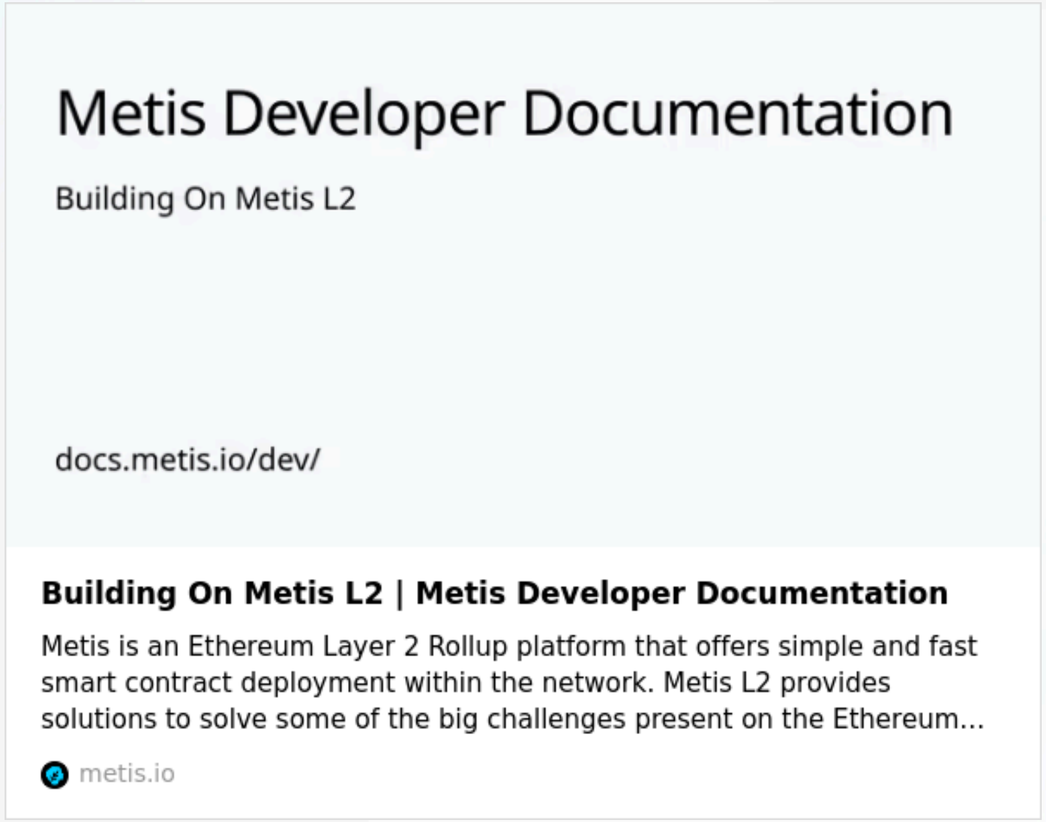
Boilerplate

METIS

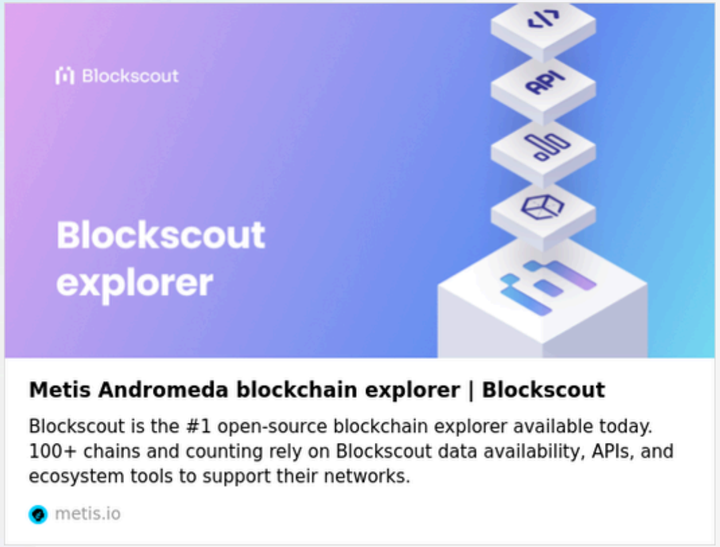
Prototype to Production with Scaffold ETH-2



Metis Documentation



Metis Explorers



Metis Andromeda blockchain explorer | Blockscout

Blockscout is the #1 open-source blockchain explorer available today. 100+ chains and counting rely on Blockscout data availability, APIs, and ecosystem tools to support their networks.

metis.io

Metis

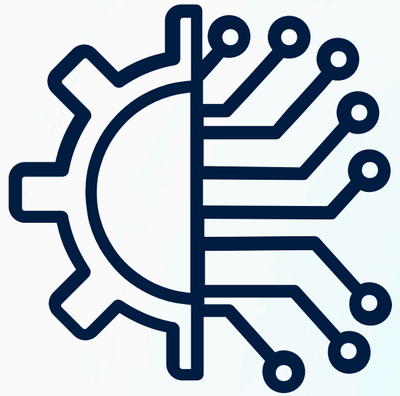
Metis Explorer allows you to explore and search for transactions, addresses, tokens, prices and other activities taking place on Metis

Metis Blockchain Explorer /

Advanced Tools to build on Metis L2

Advanced dApp Development Tools

- Indexers
- Oracles
- Subgraphs



Advanced Tools to build on Metis L2

Indexers

Indexers are tools that organize and provide efficient access to blockchain data by indexing various types of data on the blockchain.

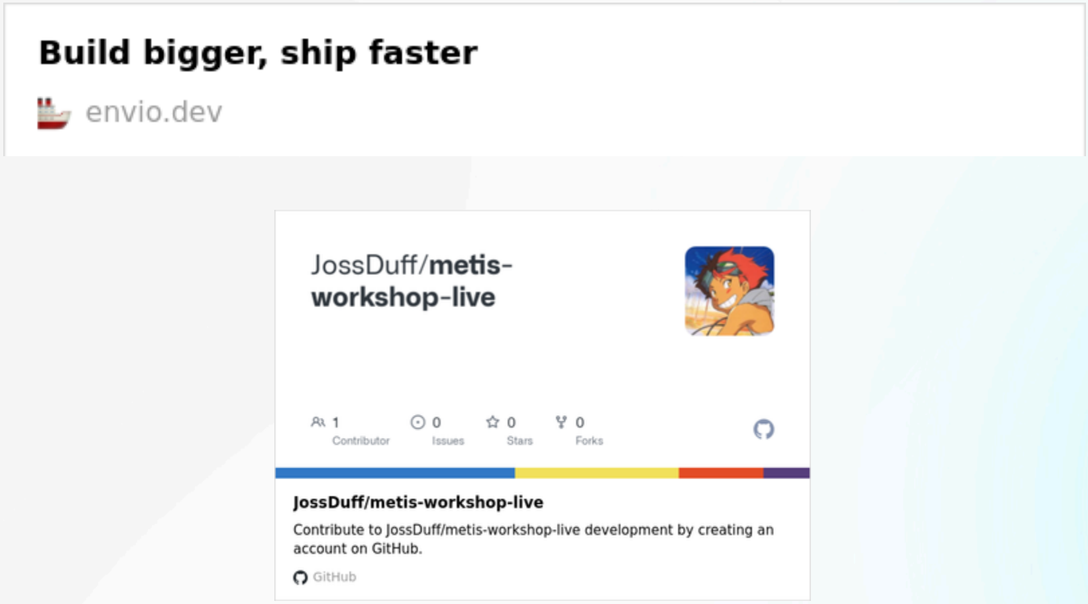
They are essential for improving the performance and usability of decentralized applications (dApps) by enabling quick and reliable data retrieval. Without indexers, querying the blockchain directly can be slow and computationally expensive.

Advanced Tools to build on Metis L2

Indexers

Use Cases

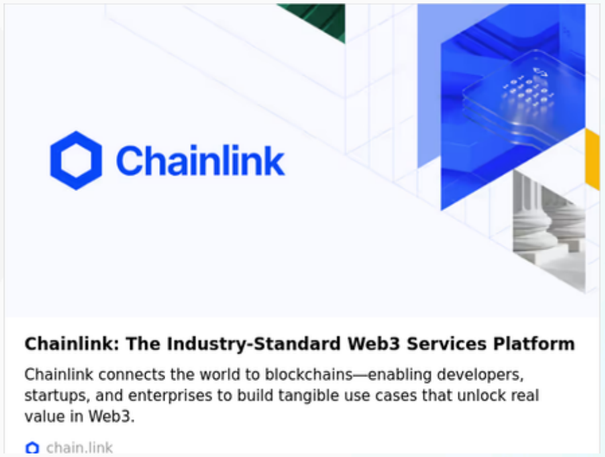
- Querying Transaction History: Retrieve a user's transaction history efficiently without directly querying the blockchain.
- Fetching User Balances: Access current and historical balances of users quickly.
- Monitoring Contract Events: Listen to and process events emitted by smart contracts in real-time.



Advanced Tools to build on Metis L2

Oracles

- Oracles are services that provide external data to smart contracts, enabling them to interact with real-world information that is not natively available on the blockchain.
- Purpose: Oracles are essential for bridging the gap between on-chain and off-chain data, allowing smart contracts to execute based on real-world events and information.

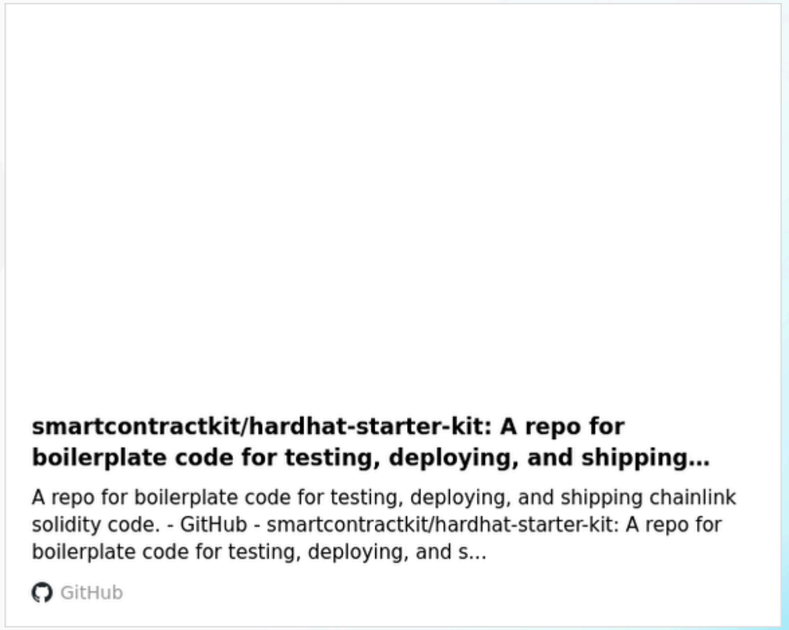


Advanced Tools to build on Metis L2

Oracles

Use Cases

- Price Feeds: Providing real-time price data for assets to decentralized finance (DeFi) applications.
- Weather Data: Supplying weather information for insurance contracts.
- Sports Scores: Feeding sports scores for betting platforms.
- APIs Integration: Connecting to any external API to fetch data required by smart contracts.



Advanced Tools to build on Metis L2

Subgraphs

- Subgraphs are custom APIs that query and organize blockchain data, making it easily accessible for decentralized applications (dApps).
- Purpose: Subgraphs simplify and optimize the process of fetching complex blockchain data, enhancing the performance and user experience of dApps.

https://www.ormilabs.xyz

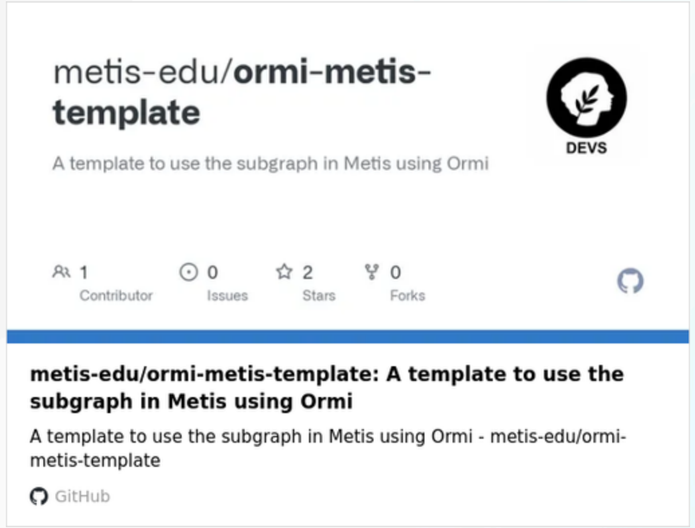
Advanced Tools to build on Metis L2

Subgraphs

Use Cases
Data Visualization: Efficiently fetch and display transaction histories, user balances, and other on-chain data.

Event Monitoring: Track and respond to smart contract events in real-time.

Analytics: Perform detailed analytics on blockchain data for insights and reporting.



Advanced Tools to build on Metis L2

Feature	Oracles	Subgraphs	Indexers
Purpose	Provide external data to smart contracts	Custom APIs for querying and organizing blockchain data	Efficient retrieval and organization of blockchain data
Example	Chainlink	ORMI	ENVIO
Primary Use	Real-time price feeds, weather data	Data visualization, event monitoring	Transaction history, user balances
Data Source	External APIs and data providers	On-chain data	On-chain data
Key Benefit	Enables interaction with real-world data	Simplifies complex data queries	Enhances data access performance
Complexity	Medium	Medium	Medium
Example Integration	Fetch ETH/USD price for DeFi	Display user transaction history	Real-time updates for dApp

The Metis Decentralized Sequencer

Benefits

For the Network

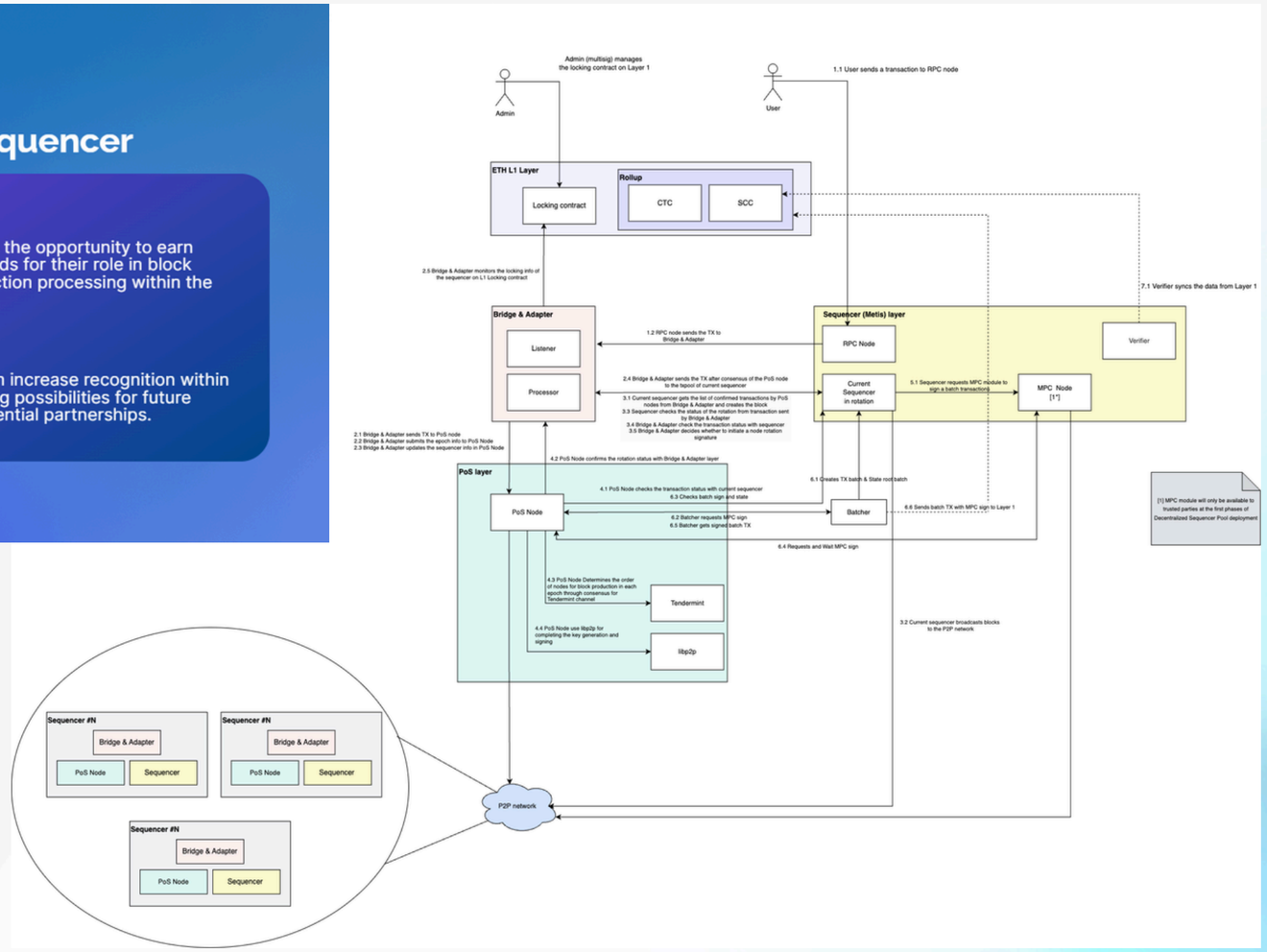
Decentralization
By participating in the consensus process, Sequencers play an essential role in maintaining the network's decentralization, transparency and stability.

Security
With multiple nodes or entities participating in sequencing, it becomes more challenging for malicious actors to control or compromise the order of transactions.

For the Sequencer

Rewards
Sequencer nodes have the opportunity to earn METIS tokens as rewards for their role in block production and transaction processing within the network.

Community
Active participation can increase recognition within the community, creating possibilities for future collaborations and potential partnerships.



END OF PART TWO :)

Connect with the Devs Here

Join Us on Telegram

@metis_dev

METIS

JOIN THE WEB3 ECONOMY

metis.io