Smart contracts

Smart contracts are self-executing contracts with their own state. On Sei, developers have the flexibility to use both EVM (Ethereum Virtual Machine) and CosmWasm for smart contract development, allowing for a broad range of decentralized applications (dApps) and interoperability between different blockchain ecosystems.

Optimistic Parallelization on Sei

Optimistic parallelization is a technique used by Sei to enhance the throughput and efficiency of both EVM and CosmWasm smart contract executions. This approach significantly increases the number of transactions that can be handled per second without compromising the developer experience by requiring them to define dependencies in smart contracts, making Sei backwards compatible with Ethereum.

Choosing Between EVM and CosmWasm for Smart Contract Development

When developing smart contracts on Sei, developers have the option to choose between using the EVM and CosmWasm. Each approach has its own benefits and trade-offs, and the best choice depends on the specific requirements and context of the project. Here, we will compare EVM and CosmWasm to help you make an informed decision.

EVM Smart Contracts

Overview : EVM is a decentralized computation engine that allows the execution of smart contracts on Ethereum and EVM-compatible blockchains. EVM contracts are typically written in Solidity, a language designed for Ethereum.

Benefits :

- Compatibility
- : EVM is by far the most widely supported and allows for the use of existing Ethereum tools, libraries, and wallets (such as MetaMask). This makes it seamless to port existing Ethereum dApps to the Sei blockchain.
- Developer Ecosystem
- : The large Ethereum developer community means extensive resources, tutorials, and libraries are available.

Considerations :

- Language
- : Solidity, while powerful, has a steeper learning curve compared to some other languages.
- Resource Consumption
- : EVM contracts can be gas-intensive, making optimization important.

Use Cases :

- Projects requiring compatibility with existing Ethereum infrastructure.
- Developers with a background in Ethereum development.
- Applications benefiting from Ethereum’s mature tooling and ecosystem.

Example Tools :

- MetaMask
- : Popular Ethereum wallet.* MetaMask(opens in a new tab)
- Hardhat
- : Development environment for Ethereum.* Hardhat Documentation(opens in a new tab)
- Foundry
- : Fast and portable toolkit for Ethereum development.* Foundry Documentation(opens in a new tab)
- Wagmi
- : React hooks library for Ethereum dApps.* Wagmi Documentation(opens in a new tab)
- Viem
- : A lightweight and flexible library for Ethereum development.* Viem Documentation(opens in a new tab)

CosmWasm Smart Contracts

Overview : CosmWasm is a smart contract platform built for the Cosmos ecosystem, enabling contracts to be written in WebAssembly (Wasm) languages, with Rust being the most commonly used.

Benefits :

- Language Flexibility
- : Supports multiple programming languages that compile to Wasm, though Rust is preferred for its performance and safety features.

- Modularity
- : CosmWasm’s architecture encourages modular contract design, making it easier to upgrade and maintain.
- Integration with Cosmos
- : Seamlessly integrates with other Cosmos chains, leveraging the extensive Cosmos ecosystem and its IBC (Inter-Blockchain Communication) protocol.

Considerations :

- Learning Curve
- : Developers need to be familiar with Rust or other Wasm-supported languages.
- Ecosystem
- : While growing, the CosmWasm ecosystem is smaller compared to Ethereum’s.

Use Cases :

- Projects requiring high performance and security.
- Applications benefiting from Cosmos features and interoperability.
- Developers with a background in Rust or other Wasm-supported languages.

Example Tools :

- CosmWasm
- : Platform for building smart contracts on Cosmos.*CosmWasm Documentation(opens in a new tab)
- CosmosKit
- : Library for connecting to Cosmos wallets.*CosmosKit Documentation(opens in a new tab)
- @sei-js: Typescript library for interacting with Sei.*@sei-js Documentation(opens in a new tab)
- CosmJS
- : JavaScript library for interacting with Cosmos blockchains.*CosmJS Documentation(opens in a new tab)

Interoperability

Interoperability is a key feature of the Sei blockchain, allowing EVM and CosmWasm contracts to interact seamlessly.

- EVM Precompile Contracts: Precompiles are smart contracts pre-bundled into the chain. Sei has many precompiles to enable EVM dApps and contracts to access native Cosmos functions, such as staking and executing CosmWasm contracts. The official precompiles can be found in@sei-js/evm(opens in a new tab)
- Pointer Contracts
- : These contracts act as intermediaries, enabling calls between EVM and CosmWasm contracts. This allows developers to leverage the strengths of both environments and create more versatile dApps.* Pointer Contracts Documentation

Best Practices

Security is paramount in smart contract development. Here are some best practices to ensure your contracts are secure:

- Code Audits
- : Regularly audit your code to identify and fix vulnerabilities.
- Use Established Libraries
- : Leverage well-tested libraries and frameworks to reduce the risk of introducing bugs.
- Write Quality Code
- : Adhere to best practices for smart contract development, such as proper error handling, using safe math libraries, and avoiding unnecessary complexity.
- Testing
- : Thoroughly test your contracts, including unit tests, integration tests, and security tests, to ensure they function as expected under various scenarios.
- Optimization:
- Using packages like rust-optimizer for CosmWasm contracts are critical to ensure gas fees remain low while using your smart contract.

Last updated onMay 27, 2024 Gas Frontend dApps