Complete Tutorial: Smart Contract Stake Delegation on Hyperliquid Testnet

Hyperliquid's dual-architecture enables high-performance staking with EVM compatibility

Hyperliquid represents a unique approach to blockchain architecture, combining a native high-performance execution layer (HyperCore) with full EVM compatibility (HyperEVM). Gitbook +2 This tutorial provides a comprehensive guide to implementing stake delegation smart contracts on Hyperliquid's testnet, leveraging the platform's **200,000+ transactions per second** capability and sub-second finality through its HyperBFT consensus mechanism. Hyperliquid Docs +2

The platform's staking system, launched in December 2024, operates through a delegated proof-of-stake model with **21 active validators on mainnet** and automatic reward compounding at approximately **2.37% APY**. Kiln Hyperliquid Docs Smart contracts can interact with this native staking infrastructure through specialized precompiles and system contracts, enabling sophisticated delegation strategies while maintaining the performance benefits of the native implementation.

Understanding Hyperliquid's technical architecture

Hyperliquid employs a dual-execution model that separates concerns between high-frequency trading operations and smart contract execution. **HyperCore handles perpetual futures, spot trading, and native staking** with optimized performance, while **HyperEVM provides full Ethereum compatibility** for decentralized applications. Gitbook +2 This architecture allows smart contracts to interact with native staking operations through precompiled contracts at specific addresses.

The consensus layer uses HyperBFT, a variant of HotStuff gitbook Hyperliquid Docs that achieves **0.1-second median block latency** with one-block finality. Hyperliquid Docs Blockhead Validators maintain the network through a stake-weighted selection process, with the top validators by total delegated stake forming the active set. The staking mechanism follows Ethereum's reward model, where rewards are inversely proportional to the square root of total staked HYPE, encouraging early participation while maintaining long-term sustainability. Kiln +2

The platform's unique dual-block architecture introduces two execution modes: small blocks with **2 million gas limits executing in 1 second**, and large blocks with **30 million gas limits executing in 1 minute.** Alchemy +3 This design optimizes for both high-frequency operations and complex contract deployments, requiring developers to understand when to switch between modes for optimal performance.

Setting up your Hyperliquid testnet environment

Network configuration and wallet setup

Begin by configuring your development environment for the Hyperliquid testnet. Add the network to MetaMask or your preferred wallet using these parameters:

```javascript
const testnetConfig = {
  chainId: 998,
  chainName: "HyperEVM Testnet",
  rpcUrls: ["https://rpc.hyperliquid-testnet.xyz/evm"],
  nativeCurrency: {
    name: "HYPE",
    symbol: "HYPE",
    decimals: 18
  },
  blockExplorerUrls: ["https://testnet.purrsec.com/"]
}
```

For production-grade development, consider using enterprise RPC providers like **QuickNode or Chainstack**, which offer dedicated nodes with archive data support and higher rate limits than the public endpoints' 100 requests per minute limitation. Chainstack Chainstack

Acquiring testnet resources

The official testnet faucet requires prior mainnet activity, providing **1,000 mock USDC every 4 hours** to addresses that have deposited USDC on mainnet. Chainstack +3 To obtain HYPE tokens for gas fees, claim mock USDC from the faucet at `https://app.hyperliquid-testnet.xyz/drip`, then trade it for HYPE on the testnet exchange. Transfer HYPE from HyperCore to HyperEVM by sending to the bridge address `0x2222222222222222222222222222222222222222`. Chainstack gitbook

Alternative faucets like Chainstack's service provide **1 HYPE every 24 hours** without mainnet requirements, though API key authentication is necessary. Chainstack For larger testing requirements, consider requesting tokens through the official Discord channel's developer support.

Development environment preparation

Install Foundry as your primary development framework, as it provides optimal compatibility with Hyperliquid's infrastructure:

```bash
```

```bash
# Install Foundry
curl -L https://foundry.paradigm.xyz | bash
foundryup

# Create new project
forge init hyperliquid-staking
cd hyperliquid-staking

# Configure for Hyperliquid
echo "HYPERLIQUID_TESTNET_RPC=https://rpc.hyperliquid-testnet.xyz/evm" >> .env
echo "PRIVATE_KEY=your_private_key_here" >> .env
```

Implementing the stake delegation smart contract

Core staking interface and precompiles

Hyperliquid's staking functionality is accessible through precompiled contracts and the CoreWriter system contract. Hyperliquid Docs gitbook Here's the comprehensive implementation of a stake delegation manager:

```solidity
```

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

interface IHyperCoreReader {
    function getValidatorInfo(address validator) external view returns (
        uint256 totalStake,
        uint256 commission,
        bool isActive,
        bool isJailed
    );

    function getDelegationInfo(address delegator, address validator)
        external view returns (uint256 amount, uint256 rewards);

    function getValidatorSet() external view returns (address[] memory);
}

interface ICoreWriter {
    function sendRawAction(bytes calldata data) external;
}

contract StakeDelegationManager {
    // Precompile addresses
    address constant CORE_READER = 0x0000000000000000000000000000000000000800;
    address constant CORE_WRITER = 0x3333333333333333333333333333333333333333;

    // Constants
    uint256 constant MIN_DELEGATION = 1e18; // 1 HYPE minimum
    uint256 constant DELEGATION_LOCKUP = 1 days;
    uint256 constant UNBONDING_PERIOD = 7 days;

    // State variables
    mapping(address => mapping(address => uint256)) public delegations;
    mapping(address => mapping(address => uint256)) public delegationTimestamp;
    mapping(address => uint256) public totalDelegated;

    // Events
    event DelegationInitiated(
        address indexed delegator,
        address indexed validator,
        uint256 amount,
        uint256 timestamp
    );

    event UndelegationQueued(
        address indexed delegator,
```

```solidity
    address indexed validator,
    uint256 amount,
    uint256 availableAt
);

// Errors
error InvalidValidator(address validator);
error InsufficientAmount(uint256 provided, uint256 required);
error ValidatorJailed(address validator);
error DelegationLocked(uint256 unlockTime);
error ExcessiveCommission(uint256 commission);

/**
 * @notice Delegate HYPE tokens to a validator
 * @param validator Address of the validator to delegate to
 * @param amount Amount of HYPE to delegate (in wei)
 */
function delegateToValidator(address validator, uint256 amount)
    external payable {

    if (amount < MIN_DELEGATION) {
        revert InsufficientAmount(amount, MIN_DELEGATION);
    }

    // Query validator status through precompile
    (
        uint256 totalStake,
        uint256 commission,
        bool isActive,
        bool isJailed
    ) = IHyperCoreReader(CORE_READER).getValidatorInfo(validator);

    if (!isActive) {
        revert InvalidValidator(validator);
    }

    if (isJailed) {
        revert ValidatorJailed(validator);
    }

    // Check commission (10000 = 100%)
    if (commission > 500) { // Max 5% commission
        revert ExcessiveCommission(commission);
    }

    // Encode delegation action for CoreWriter
    bytes memory action = _encodeDelegateAction(validator, amount, false);
```

```
        // Execute delegation through CoreWriter
        ICoreWriter(CORE_WRITER).sendRawAction(action);

        // Update state
        delegations[msg.sender][validator] += amount;
        delegationTimestamp[msg.sender][validator] = block.timestamp;
        totalDelegated[msg.sender] += amount;

        emit DelegationInitiated(msg.sender, validator, amount, block.timestamp);
    }

    /**
     * @notice Undelegate HYPE tokens from a validator
     * @param validator Address of the validator to undelegate from
     * @param amount Amount to undelegate
     */
    function undelegateFromValidator(address validator, uint256 amount)
        external {

        uint256 delegated = delegations[msg.sender][validator];
        if (amount > delegated) {
            revert InsufficientAmount(delegated, amount);
        }

        // Check lockup period (1 day minimum)
        uint256 delegationTime = delegationTimestamp[msg.sender][validator];
        if (block.timestamp < delegationTime + DELEGATION_LOCKUP) {
            revert DelegationLocked(delegationTime + DELEGATION_LOCKUP);
        }

        // Encode undelegation action
        bytes memory action = _encodeDelegateAction(validator, amount, true);

        // Execute undelegation
        ICoreWriter(CORE_WRITER).sendRawAction(action);

        // Update state
        delegations[msg.sender][validator] -= amount;
        totalDelegated[msg.sender] -= amount;

        uint256 availableAt = block.timestamp + UNBONDING_PERIOD;
        emit UndelegationQueued(msg.sender, validator, amount, availableAt);
    }

    /**
     * @notice Encode delegation/undelegation action for CoreWriter
     */
    function _encodeDelegateAction(
        address validator,
        uint256 amount,
        bool isUndelegate
    ) private pure returns (bytes memory) {
        bytes memory encodedParams = abi.encode(validator, amount, isUndelegate);
        bytes memory action = new bytes(4 + encodedParams.length);

        // Protocol version and action ID
        action[0] = 0x01; // Version 1
        action[1] = 0x00;
        action[2] = 0x00;
        action[3] = 0x03; // Action ID 3 = token delegate

        // Copy encoded parameters
        for (uint256 i = 0; i < encodedParams.length; i++) {
            action[4 + i] = encodedParams[i];
        }

        return action;
    }
}
```

Advanced multi-validator delegation strategy

For sophisticated delegation strategies across multiple validators, implement this advanced contract:

```
solidity
```

```
contract MultiValidatorDelegation {
    struct ValidatorAllocation {
        address validator;
        uint256 weight; // Basis points (10000 = 100%)
        uint256 maxCommission; // Maximum acceptable commission
        bool autoRebalance;
    }

    struct DelegationStrategy {
        ValidatorAllocation[] allocations;
        uint256 rebalanceThreshold; // Percentage deviation to trigger rebalance
        uint256 lastRebalance;
    }

    mapping(address => DelegationStrategy) public strategies;

    /**
     * @notice Set delegation strategy with automatic rebalancing
     */
    function setDelegationStrategy(
        ValidatorAllocation[] calldata allocations,
        uint256 rebalanceThreshold
    ) external {
        require(allocations.length > 0 && allocations.length <= 10,
            "Invalid allocation count");

        uint256 totalWeight = 0;
        for (uint i = 0; i < allocations.length; i++) {
            totalWeight += allocations[i].weight;

            // Validate validator
            (,uint256 commission, bool isActive,) =
                IHyperCoreReader(CORE_READER).getValidatorInfo(
                    allocations[i].validator
                );

            require(isActive, "Validator not active");
            require(commission <= allocations[i].maxCommission,
                "Commission exceeds maximum");
        }

        require(totalWeight == 10000, "Weights must sum to 100%");

        // Clear existing strategy
        delete strategies[msg.sender].allocations;
        // Set new strategy
        DelegationStrategy storage strategy = strategies[msg.sender];
        for (uint i = 0; i < allocations.length; i++) {
            strategy.allocations.push(allocations[i]);
        }
        strategy.rebalanceThreshold = rebalanceThreshold;
        strategy.lastRebalance = block.timestamp;
    }

    /**
     * @notice Execute delegation according to strategy
     */
    function executeDelegation() external payable {
        DelegationStrategy storage strategy = strategies[msg.sender];
        require(strategy.allocations.length > 0, "No strategy set");

        uint256 totalAmount = msg.value;

        for (uint i = 0; i < strategy.allocations.length; i++) {
            uint256 allocation = (totalAmount * strategy.allocations[i].weight)
                / 10000;

            if (allocation > 0) {
                _delegateToValidator(
                    strategy.allocations[i].validator,
                    allocation
                );
            }
        }
    }
}
```

Deploying to Hyperliquid testnet

Switching to big blocks for deployment

Contract deployment requires switching to big blocks due to gas requirements. (medium) Execute this command before deployment: (LayerZero)

```bash
```

```
# Using LayerZero SDK to switch to big blocks
npx @layerzerolabs/hyperliquid-composer set-block \
  --size big \
  --network testnet \
  --private-key $PRIVATE_KEY

# Wait for confirmation (approximately 1 minute)
sleep 60
```

Deployment script using Foundry

Create a deployment script at `script/Deploy.s.sol`:

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "forge-std/Script.sol";
import "../src/StakeDelegationManager.sol";

contract DeployScript is Script {
  function run() external {
    uint256 deployerPrivateKey = vm.envUint("PRIVATE_KEY");

    vm.startBroadcast(deployerPrivateKey);

    StakeDelegationManager manager = new StakeDelegationManager();

    console.log("StakeDelegationManager deployed at:",
      address(manager));

    vm.stopBroadcast();
  }
}
```

Execute deployment:

```bash
```

```
# Deploy to testnet
forge script script/Deploy.s.sol:DeployScript \
  --rpc-url $HYPERLIQUID_TESTNET_RPC \
  --broadcast \
  --verify \
  --gas-price 1000000000

# Switch back to small blocks after deployment
npx @layerzerolabs/hyperliquid-composer set-block \
  --size small \
  --network testnet \
  --private-key $PRIVATE_KEY
```

Testing and validation procedures

Comprehensive test suite implementation

Create thorough tests at `test/StakeDelegation.t.sol`:

```
solidity
```

```
contract StakeDelegationTest is Test {
    StakeDelegationManager manager;
    address validator = 0x1234567890123456789012345678901234567890;
    address user = address(0x1);

    function setUp() public {
        // Fork testnet for testing
        vm.createSelectFork("https://rpc.hyperliquid-testnet.xyz/evm");

        manager = new StakeDelegationManager();
        vm.deal(user, 100 ether); // Fund test user
    }

    function testDelegation() public {
        vm.startPrank(user);

        // Test successful delegation
        manager.delegateToValidator{value: 10 ether}(validator, 10 ether);

        assertEq(manager.delegations(user, validator), 10 ether);
        assertEq(manager.totalDelegated(user), 10 ether);

        vm.stopPrank();
    }

    function testDelegationLockup() public {
        vm.startPrank(user);

        manager.delegateToValidator{value: 10 ether}(validator, 10 ether);

        // Attempt immediate undelegation (should fail)
        vm.expectRevert();
        manager.undelegateFromValidator(validator, 10 ether);

        // Fast forward past lockup period
        vm.warp(block.timestamp + 1 days + 1);

        // Now undelegation should succeed
        manager.undelegateFromValidator(validator, 10 ether);

        vm.stopPrank();
    }
}
```

Monitoring and debugging tools

Monitor your deployed contracts using these endpoints and tools:

```javascript
// Query delegation status via API
const queryDelegations = async (address) => {
  const response = await fetch('https://api.hyperliquid-testnet.xyz/info', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      type: 'delegations',
      user: address
    })
  });
  return response.json();
};

// Monitor validator performance
const getValidatorMetrics = async (validator) => {
  const response = await fetch('https://api.hyperliquid-testnet.xyz/info', {
    method: 'POST',
    body: JSON.stringify({
      type: 'validatorSummary',
      validator: validator
    })
  });
  return response.json();
};
```

Validator selection and delegation strategies

Evaluating validator performance metrics

When selecting validators for delegation, consider these **critical performance indicators**: uptime percentage (target >99.9%), commission rates (typically 1-5%), jailing history, and total stake distribution. Imperator The testnet's top 50 validators form the active set, compared to mainnet's 21, Panews providing more options for diversification during testing. Medium

Foundation validators offer **0% commission but raise centralization concerns**, currently controlling approximately 81% of staked shares on mainnet. Panews Independent validators like Imperator, Bharvest, and Validation Cloud provide alternative options with proven track records and reasonable commission structures. Imperator

Implementing automated validator selection

```solidity
contract ValidatorSelector {
    struct ValidatorScore {
        address validator;
        uint256 score;
        uint256 commission;
        uint256 totalStake;
    }

    function selectOptimalValidators(uint256 count)
        public view returns (address[] memory) {

        address[] memory allValidators = IHyperCoreReader(CORE_READER)
            .getValidatorSet();

        ValidatorScore[] memory scores = new ValidatorScore[](
            allValidators.length
        );

        for (uint i = 0; i < allValidators.length; i++) {
            (uint256 stake, uint256 commission, bool isActive, bool isJailed) =
                IHyperCoreReader(CORE_READER).getValidatorInfo(
                    allValidators[i]
                );

            if (isActive && !isJailed) {
                // Score based on commission (lower is better)
                // and decentralization (lower stake is better)
                scores[i].validator = allValidators[i];
                scores[i].commission = commission;
                scores[i].totalStake = stake;
                scores[i].score = _calculateScore(commission, stake);
            }
        }

        // Sort and return top validators
        return _selectTopValidators(scores, count);
    }
}
```

Error handling and security best practices

Common error scenarios and mitigation

The platform's unique architecture introduces specific error conditions requiring careful handling. **Invalid precompile calls consume all provided gas** without reverting, necessitating defensive programming patterns. Hyperliquid Docs gitbook The dual-block system can cause transaction failures if attempting complex operations in small blocks, requiring dynamic block-size detection.

Implement comprehensive error handling for validator jailing events, which temporarily remove validators from the active set. Hyperliquid Docs Monitor commission changes, as validators can increase rates up to 1% without restriction, Hyperliquid Docs potentially affecting delegation returns. Imperator The 7-day unbonding period creates liquidity constraints requiring careful management in production systems. Hyperliquid Docs

Security implementation patterns

solidity

```
contract SecureStakingManager {
    using ReentrancyGuard for *;

    // Rate limiting
    mapping(address => uint256) lastAction;
    uint256 constant ACTION_COOLDOWN = 1 minutes;

    modifier rateLimited() {
        require(block.timestamp >= lastAction[msg.sender] + ACTION_COOLDOWN,
            "Action rate limited");
        lastAction[msg.sender] = block.timestamp;
        _;
    }

    // Safe precompile interaction
    function safePrecompileCall(address precompile, bytes memory data)
        internal view returns (bool success, bytes memory result) {

        // Gas limit for precompile calls
        uint256 gasLimit = 50000;

        assembly {
            success := staticcall(
                gasLimit,
                precompile,
                add(data, 0x20),
                mload(data),
                0,
                0
            )

            let size := returndatasize()
            result := mload(0x40)
            mstore(result, size)
            returndatacopy(add(result, 0x20), 0, size)
            mstore(0x40, add(result, add(0x20, size)))
        }
    }
}
```

Recent updates and platform evolution (2024-2025)

December 2024 staking launch

The staking mechanism launched in December 2024 introduced **automatic daily reward compounding** and the ability to delegate to multiple validators simultaneously. Hyperliquid Docs Kiln The initial reward rate of approximately 2.37% APY adjusts dynamically based on total staked supply, following Ethereum's inverse square root model. Kiln Hyperliquid Docs

HyperEVM mainnet deployment (2025)

The 2025 HyperEVM mainnet launch enables **full smart contract functionality** with native integration to HyperCore's order books and staking system. DeFi Planet The CoreWriter contract allows HyperEVM applications to execute actions on HyperCore, creating unprecedented composability between DeFi primitives and high-performance trading infrastructure. X Medium

Recent statistics demonstrate the platform's growth trajectory: **$450 billion cumulative trading volume**, over 200,000 active users, and processing the majority of on-chain perpetual futures volume. DeFi Prime Medium The introduction of USDhl, a native fiat-backed stablecoin, further enhances the ecosystem's capabilities for stake delegation rewards and automated strategies.

Production deployment recommendations

Before mainnet deployment, conduct extensive testing on the testnet with realistic transaction volumes and delegation amounts. Implement comprehensive monitoring using tools like Chainstack's dedicated nodes or QuickNode's enterprise infrastructure to ensure reliable access during high-traffic periods. Chainstack Chainstack

Consider implementing **gradual delegation strategies** to minimize risk, starting with small amounts across multiple validators before scaling up. Maintain detailed logs of all delegation activities for audit purposes, and implement automated alerting for validator performance degradation or commission changes.

The platform's rapid evolution necessitates staying current with updates through official channels: the GitHub repository for technical specifications, Discord for community support, and the official documentation for API changes. Hyperliquid Docs Regular security audits and participation in the bug bounty program help ensure contract robustness as the ecosystem matures. X

Conclusion

Hyperliquid's innovative architecture enables sophisticated stake delegation strategies through smart contracts while maintaining the performance characteristics of native implementations. The combination of HyperCore's optimized execution and HyperEVM's flexibility creates unique opportunities for developers to build advanced staking applications that were previously impossible on traditional blockchain platforms. Gitbook +3

Success in implementing stake delegation on Hyperliquid requires understanding the platform's dual-block architecture, mastering the interaction between HyperEVM and HyperCore through precompiles, and carefully managing the various time-based constraints inherent in the staking mechanism. LayerZero As the ecosystem continues to evolve rapidly, maintaining adaptability and following best practices ensures your implementations remain secure, efficient, and aligned with the platform's high-performance ethos.