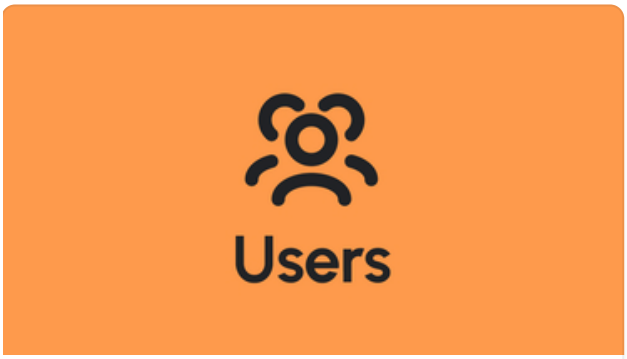
Sonic

Ask Sonic AI

Introduction

Sonic is the highest-performing EVM L1, combining speed, incentives, and world-class infrastructure, powering the next generation of DeFi applications. The chain provides 10,000 TPS and sub-second finality.

The S token is Sonic's native token, used for paying transaction fees, staking, running validators, and participating in governance.



Users

- Upgrade FTM to S
- Bridge to Sonic
- Earn Airdrop Points
- Explore Apps

Developers

- Deploy App
- Tooling and Infra
- Earn Airdrop Gems
- Fee Monetization

Network Information

- **Network name**: Sonic
- **RPC URL:** https://rpc.soniclabs.com
- **Explorer URL:** https://sonicscan.org
- **Chain ID**: 146
- **Currency symbol**: S

Sonic

Overview

Sonic is the highest-performing EVM L1, combining speed, incentives, and world-class infrastructure for DeFi. The chain provides 10,000 TPS and sub-second finality.

Developers and users on Sonic are supported by several incentive programs, including:

Fee Monetization

Developers earn 90% of the network fees their apps generate.

Innovator Fund

Up to 200 million S to onboard apps to Sonic and support new ventures.

Airdrop

~200 million S to incentivize users of both Opera and the Sonic chain.

Sonic delivers exceptional performance, enabling developers to scale their applications without limits while ensuring smooth user experiences.

- 10,000 Transactions per Second
- Sub-Second Finality
- EVM Compatible
- Solidity/Vyper Support

The native token of Sonic is S, which is used for transaction fees, staking, running validators, and participating in governance. Users holding FTM can upgrade to S on a 1:1 basis.

Furthermore, the Sonic Gateway provides developers and users with seamless access to vast liquidity through a secure bridge connected to Ethereum. With a unique fail-safe mechanism, it ensures user assets are protected in all circumstances.

S Token

The S token is the native token of Sonic. It has multiple roles within the network, such as paying for transaction fees, staking, running validators, and participating in governance.

- Upgrade Portal — Upgrade FTM to S.

Staking

You can stake S on Sonic on MySonic. Staking your S involves a 14-day waiting period if you choose to withdraw.

When staking your S, choose a reputable validator carefully. If your validator is penalized for misconduct or errors in their setup, it could impact your delegated S stake as well.

Tokenomics

At Sonic's launch, the total supply of S is 3.175 billion. As decided by multiple governance proposals, the additions below will gradually be implemented into the tokenomics of the S token.

Airdrop

An additional 6% of the 3.175 billion S will be minted six months after Sonic launches exclusively used for the airdrop program, rewarding both Fantom Opera and Sonic users and builders. The airdrop features an innovative burn mechanism that rewards active participation and gradually reduces the total supply of S tokens.

Ongoing Funding

Six months after Sonic launches, the network will mint additional S tokens to:

- Increase S adoption and global presence
- Grow the team and scale operations to drive increased adoption
- Implement robust marketing initiatives and DeFi onboarding campaigns

To fund this program, an additional 1.5% of S (47,625,000 tokens) will be minted annually for six years, starting six months after the mainnet launch.

However, to guard against inflation, the network will burn newly minted tokens not used during the year, ensuring that 100% of all newly minted tokens from this initiative are allocated toward network growth rather than being held by the treasury for later use.

For example, if Sonic Labs uses only 5,000,000 tokens in the first year, the network will burn the remaining 42,625,000 tokens.

Block Rewards

We are migrating block rewards from Fantom Opera to Sonic. As validators and stakers move to Sonic, Opera's block rewards will be reduced to zero, and the saved funds will be used to reward Sonic validators. The Sonic Foundation will maintain Opera validators for now.

To achieve a 3.5% APR for Sonic without causing further inflation in the first four years, we're reallocating the remaining FTM block rewards from Opera to Sonic. These rewards are technically part of the initial 3.175 billion S supply, but the circulating supply at launch will be ~2.88 billion tokens. The annual difference of 70 million tokens will be distributed

as validator rewards over the first four years, avoiding the need to mint new S tokens during this period for block rewards.

As a result, Opera's APR dropped to zero upon Sonic's launch. To preserve value and avoid new inflationary rewards at Sonic's inception, we will not mint new tokens for block rewards during the initial four years, as stated. After that period, S block rewards will resume by minting new tokens at a rate of 1.75% per year to reward validators.

Token Burn

We have three burn mechanisms in place that will decrease the emission of new S tokens.

Fee Monetization Burn

If a user submits a transaction on an app that isn't participating in FeeM, 50% of the transaction fee will be burned.

Airdrop Burn

Users who choose not to wait for the full 270-day maturation period for 75% of their airdrop will lose a portion of their S tokens, which will be burned.

Ongoing Funding Burn

From the 47,625,000 S tokens minted annually in the first six years of Sonic to fund growth, the network will burn any of the tokens not used during the year.

Validator Rewards

To help secure the Sonic network by running a validator and staking a certain amount of S, you can earn block rewards as well as transaction fees paid by users on Sonic.

Block Rewards

The target reward rate for validators on Sonic is 3.5% when 50% of the network is staked. The network mints tokens each epoch to provide this, except during Sonic's first four years when rewards stem from block rewards migrated from Opera, as outlined above.

The reward rate adjusts proportionally, e.g. if all S tokens are staked, the annual reward is 1.75%. Conversely, if only 25% of S tokens are staked, the annual reward is 7%.

Network Fees

Network fees are generated when users pay gas to interact with the network. Validators earn a percentage of these fees, which are distributed equally among all staked S tokens. For a detailed breakdown of how much a validator earns from gas fees, see here.

Ecosystem Vault

The Ecosystem Vault was initially launched on Fantom Opera to fuel the community ecosystem by sharing a percentage of total fees on the network with select apps in the community.

To extend this program to the Sonic chain, we'll revise the program to allocate quarterly disbursements from the Ecosystem Vault to the Sonic Community Council (SCC), an independently operated collective of ecosystem members who actively contribute to elevating the Sonic community via user-based programs, developer onboarding, and app support.

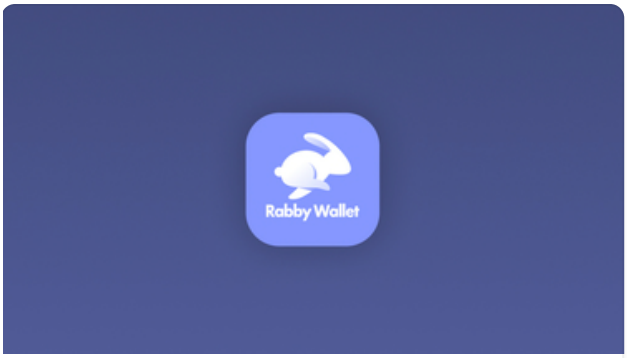
The amount will be decided at the discretion of the Sonic Foundation and reflect the SCC's previous quarter performance.

Wallets

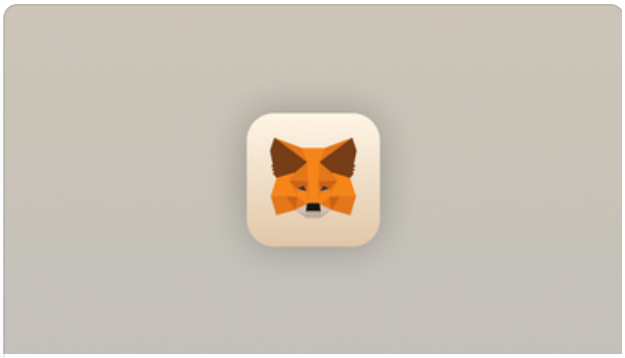
Sonic supports a range of popular wallets, both for users and institutions. Use the network information below to add Sonic to your wallet.

- **Network name**: Sonic
- **RPC URL:** https://rpc.soniclabs.com
- **Explorer URL:** https://sonicscan.org
- **Chain ID**: 146
- **Currency symbol**: S

User Wallets



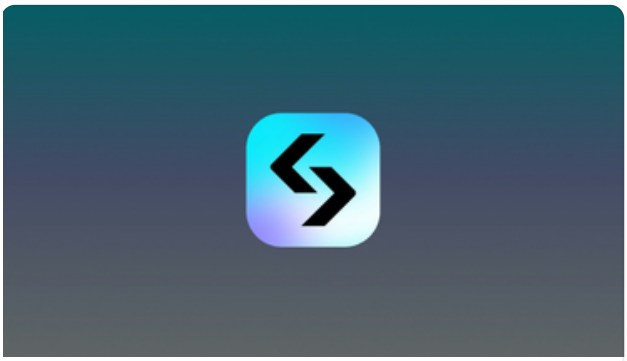
Rabby Wallet



MetaMask

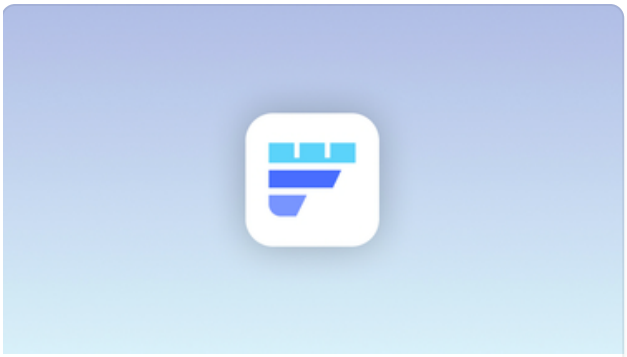
OKX Wallet

Trust Wallet

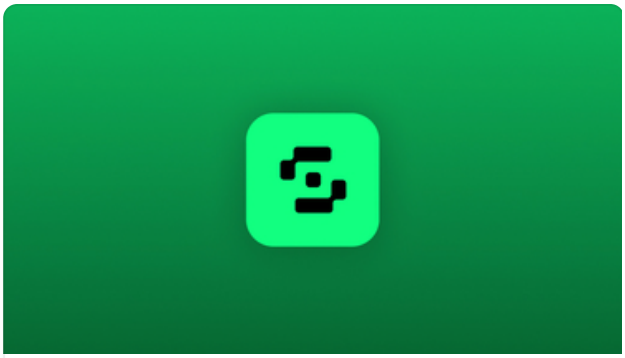


Bitget Wallet

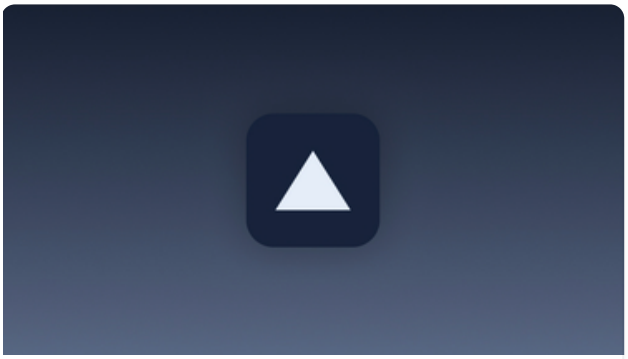
Institutional Wallets



Fordefi



Safe



Fireblocks

Sonic Gateway

Quick Overview

The Sonic Gateway is our native bridge that facilitates token transfers between Ethereum and Sonic. The bridging process consists of three steps:

1. **Deposit**
   Deposit your assets into the bridge, which takes ~15 minutes on Ethereum to achieve finalization and only ~1 second on Sonic.
2. **Heartbeat**
   After your deposit is confirmed, your assets will be bridged at the next heartbeat, which are intervals that bridge user assets in batches to ensure gas efficiency. A heartbeat occurs every ~10 minutes from Ethereum to Sonic and ~1 hour the other way. You can pay a Fast Lane fee to trigger an immediate heartbeat.
3. **Claim**
   Claim your bridged assets on the destination chain. That's it! You're now free to explore the Sonic ecosystem with your new assets.

Introduction

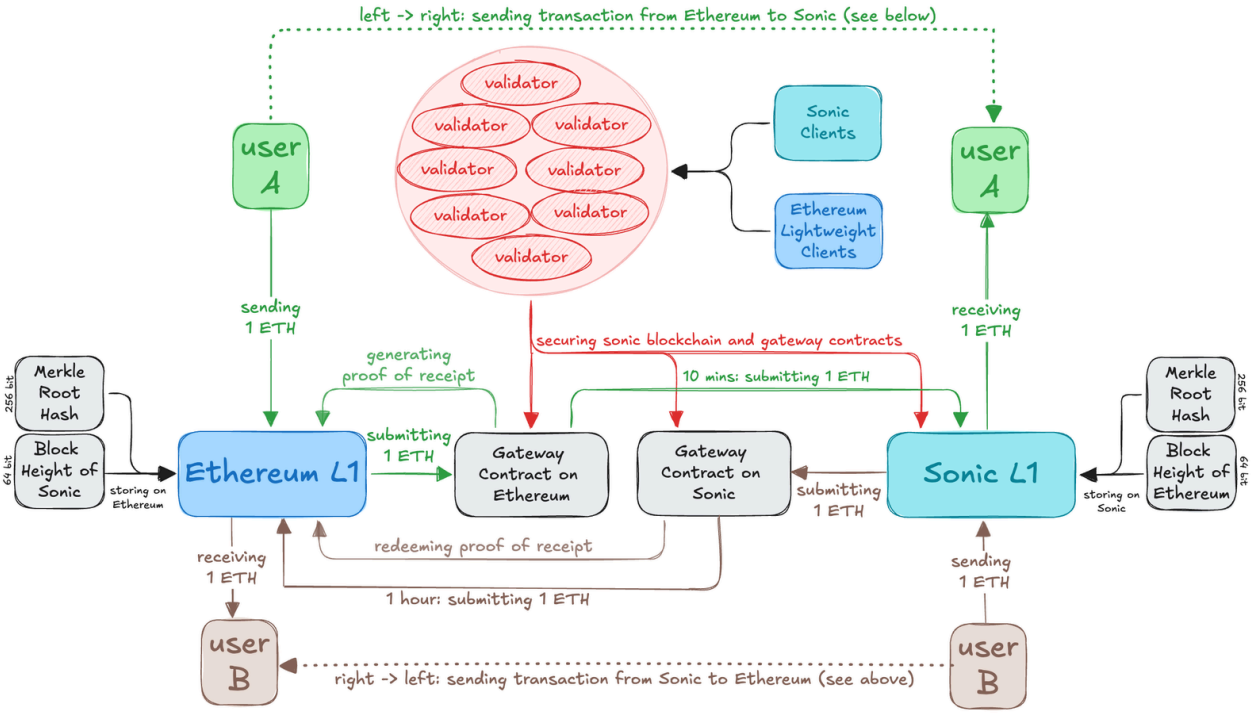
In today's evolving blockchain landscape, a native, secure bridge is critical for ecosystem health, ensuring true interoperability and preventing network isolation. Yet, many current solutions on both layer-1 and layer-2 compromise security and speed —resulting in over $2.5 billion lost to bridge hacks.

The Sonic Gateway is a revolutionary, secure bridge between Ethereum and Sonic that offers:

- **Security**: The Gateway includes a fail-safe mechanism that safeguards user assets. If the Gateway experiences prolonged failure (14 consecutive days), users can recover their bridged funds on Ethereum.
- **Speed**: Asset bridging is processed in intervals called "heartbeats" to ensure gas efficiency — every 10 minutes from Ethereum to Sonic and hourly in reverse.

At launch, the Sonic Gateway will only support bridging four tokens from Ethereum — USDC, EURC, WETH, and FTM. Our roadmap includes adding more tokens and introducing a permissionless mechanism for anyone to add new tokens for bridging.

While Sonic is not an L2, we are active participants on Ethereum as we spend ETH through the Sonic Gateway contracts.

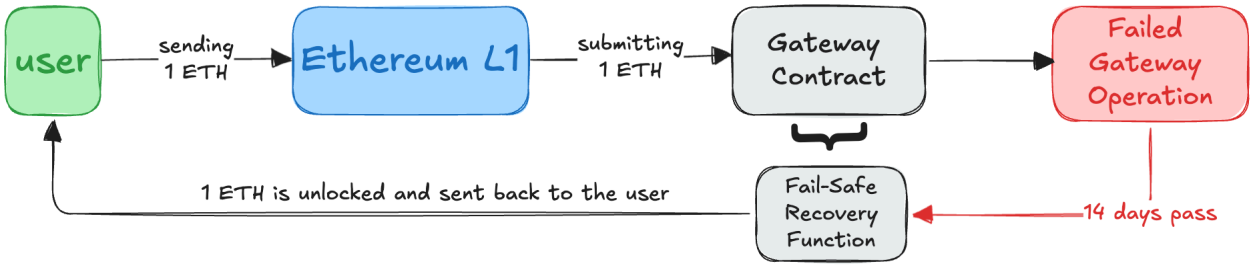


Credit: Pavel Paramonov

Fail-Safe Mechanism

The Sonic Gateway includes a fail-safe mechanism that allows users to retrieve bridged assets on the original chain if the Gateway experiences a failure. In the highly unlikely event that the Gateway or the Sonic chain is down for 14 consecutive days, users are able to reclaim their bridged assets on Ethereum.

The 14-day period is immutable and cannot be altered by Sonic Labs or any third party after deployment. Importantly, this period is not intended as a contest period but rather as an essential feature that ensures users retain custody of their bridged funds on the originating chain.



Credit: Pavel Paramonov

Fast Lane

Assets bridged through the Sonic Gateway are processed in intervals called "heartbeats", ensuring gas efficiency by bundling bridging transactions together. For assets moving from Ethereum to Sonic, these heartbeats occur every 10 minutes, while Sonic to Ethereum heartbeats occur every hour.

During each interval, all queued transactions are processed simultaneously. While this system reduces costs, it may introduce waiting periods for users needing their assets bridged immediately. To address this, Fast Lane allows users to bypass the wait for a small fee and have their bridge transaction processed instantly.

Fast Lane works by adding an additional heartbeat to the Gateway. This means all other queued assets waiting to be bridged are also processed immediately, effectively accelerating the entire network. By using Fast Lane, users not only avoid delays and seize timely opportunities but also contribute to the broader ecosystem's efficiency, ensuring faster bridging for everyone involved.

Looking Into the Future

By enabling canonical access to native assets from other layer-1 platforms, the Gateway fosters a secure and thriving economy on the Sonic network.

Users can directly access these canonical assets on Sonic while maintaining asset security. The Sonic Gateway thus provides safe access to high-demand assets that natively exist outside the Sonic network.

Frequently Asked Questions

Does the Sonic Gateway also secure third-party bridges?

No, only assets bridged through the Sonic Gateway are secured by the fail-safe mechanism. Additionally, if a user spends the tokens bridged through the Sonic Gateway, e.g. by swapping to another token, these tokens may not be recoverable. In the future, Sonic Labs may provide specific recovery mechanisms for some financial protocols on Sonic.

Does Sonic rely on the state of Ethereum?

No, Sonic operates independently as its own layer-1 platform and does not rely on Ethereum's state. Instead, the Sonic Gateway uses Merkle proofs on both Ethereum and Sonic to verify the rightful owners of bridged assets. These assets can be redeemed on Ethereum by their rightful ownership regardless of the Sonic Gateway's status.

Does Sonic store data on Ethereum?

L2 platforms store their transactions on Ethereum, using methods like blobs, which allow these transactions to be reconstructed and contested. This is crucial for resolving any security issues that may arise after the transaction is submitted by the user (e.g. if a transaction is challenged during the 7-day period). However, this approach leads to substantial storage demands on Ethereum, costing millions annually.

Our approach is more efficient as we store only a Merkle root hash (256-bit) and the block heights of both blockchains (2x64-bit) at regular intervals (e.g.

hourly) through the Sonic Gateway, which uses Sonic's own trusted validator set. This significantly reduces data storage, requiring only about 1 KB per day (or 407 KB per year).

Note: We do not retain previous hashes; each new update through the Sonic Gateway overwrites the existing hash and block heights in the Gateway's smart contracts. The full historical states and transactions remain accessible via an untrusted channel as an archive node.

Build on Sonic

Sonic offers developers robust infrastructure with 10,000 transactions per second and sub-second finality, ensuring your apps are both fast and scalable.

With full EVM compatibility and support for Solidity and Vyper, Sonic seamlessly integrates with common developer tools such as Chainlink, Safe, Pyth, Alchemy, and more. Additionally, Sonic provides the incentives, such as Fee Monetization, necessary to innovate and thrive in the ecosystem.

Test your contracts on the Blaze testnet first (see Getting Started), then deploy to the mainnet when ready. Dive in and explore how you can leverage Sonic's powerful features to bring your ideas to life.

To meet other builders and receive support, join the official Sonic builders group.

Getting Started

Sonic offers developers two distinct networks to build and deploy their apps. First, the Sonic Blaze testnet serves as a dedicated environment where you can rigorously test your smart contract code and deployments, ensuring everything functions correctly using faucet tokens.

Once your contracts are thoroughly vetted on the Blaze testnet, you can confidently deploy your live apps to the Sonic mainnet, the production environment where end users will interact with your apps.

The following are the network details:

Sonic Mainnet

- **Network name**: Sonic
- **RPC URL:** https://rpc.soniclabs.com
- **Explorer URL:** https://sonicscan.org
- **Chain ID**: 146
- **Currency symbol**: S

Blaze Testnet

- **Network name**: Sonic Blaze Testnet
- **RPC URL:** https://rpc.blaze.soniclabs.com
- **Explorer URL:** https://testnet.sonicscan.org
- **Chain ID**: 57054
- **Currency symbol**: S
- **Faucet:** https://testnet.soniclabs.com/account

To meet other builders and receive support, join the official Sonic builders group.

Deploy Contracts

At the software level, deploying to Sonic is the same as deploying to any other EVM network.

The only difference is which network you connect to. Use https://rpc.blaze.soniclabs.com as the connection endpoint for the Blaze testnet or https://rpc.soniclabs.com for the mainnet.

For the Blaze testnet, you can use the Sonic Blaze dashboard to obtain an initial amount of S to execute transactions on the testnet.

Here are example configurations for Hardhat to deploy on the Sonic mainnet or Blaze testnet:

Sonic Mainnet

```
require("@nomicfoundation/hardhat-toolbox");

// Replace this private key with your Sonic account private key
const SONIC_PRIVATE_KEY = "YOUR SONIC TEST ACCOUNT PRIVATE KEY";

module.exports = {
  solidity: "0.8.26",
  networks: {
    sonic: {
      url: "https://rpc.soniclabs.com",
      accounts: [SONIC_PRIVATE_KEY]
    }
  }
};
```

Blaze Testnet

```
require("@nomicfoundation/hardhat-toolbox");

// Replace this private key with your Sonic account private key
const SONIC_PRIVATE_KEY = "YOUR SONIC TEST ACCOUNT PRIVATE KEY";

module.exports = {
  solidity: "0.8.26",
  networks: {
    sonic: {
      url: "https://rpc.blaze.soniclabs.com",
      accounts: [SONIC_PRIVATE_KEY]
    }
  }
};
```

To deploy, execute `npx hardhat run scripts/deploy.js --network sonic`.

Please note that the **Sonic Blaze testnet** is a testing playground designed to showcase technology capabilities. The data stored on the network might eventually be deleted, with or without notice.

Verify Contracts

Verifying your smart contract makes its source code publicly visible and auditable on the block explorer, creating transparency and trust. Here are the recommended methods to verify contracts on the Sonic mainnet explorer and the Sonic Blaze testnet explorer.

Method 1. Hardhat Verification (*Recommended*)

The most streamlined way to verify contracts is using Hardhat with hardhat-toolbox:

1. Install Hardhat toolbox:

```
npm install --save-dev @nomicfoundation/hardhat-toolbox
```

2. Configure `hardhat.config.js`:

```
require("@nomicfoundation/hardhat-toolbox");

module.exports = {
  solidity: "0.8.26",
  networks: {
    sonic: {
      url: "https://rpc.soniclabs.com",
      chainId: 146,
      accounts: [SONIC_PRIVATE_KEY]
    },
    sonicTestnet: {
      url: "https://rpc.blaze.soniclabs.com",
      chainId: 57054,
      accounts: [SONIC_PRIVATE_KEY]
    }
  },
  etherscan: {
    apiKey: {
      sonic: "YOUR_SONICSCAN_API_KEY",
      sonicTestnet: "YOUR_SONICSCAN_TESTNET_API_KEY"
    },
    customChains: [
      {
        network: "sonic",
        chainId: 146,
        urls: {
          apiURL: "https://api.sonicscan.org/api",
          browserURL: "https://sonicscan.org"
        }
      },
      {
        network: "sonicTestnet",
        chainId: 57054,
        urls: {
          apiURL: "https://api-testnet.sonicscan.org/api",
          browserURL: "https://testnet.sonicscan.org"
        }
      }
    ]
  }
};
```

3. Store your SonicScan API key in a `.env` file:

```
API_KEY=your_sonicscan_api_key
```

4. Verify your contract:

```
# For mainnet
npx hardhat verify --network sonic DEPLOYED_CONTRACT_ADDRESS [CONSTRUCTOR

# For testnet
npx hardhat verify --network sonicTestnet DEPLOYED_CONTRACT_ADDRESS [CONS
```

Method 2: Programmatic Verification

For automated deployments, you can verify contracts programmatically in your deployment scripts:

```
async function main() {
  // Deploy contract
  const Contract = await ethers.getContractFactory("YourContract");
  const contract = await Contract.deploy(constructorArg1, constructorArg2
  await contract.waitForDeployment();

  console.log("Contract deployed to:", await contract.getAddress());

  // Wait for some block confirmations
  await new Promise(resolve => setTimeout(resolve, 30000));

  // Verify the contract
  await hre.run("verify:verify", {
    address: await contract.getAddress(),
    constructorArguments: [constructorArg1, constructorArg2]
  });
}
```

Method 3: Manual Verification

If automated methods fail, you can verify manually through the explorer interface:

1. Go to the Sonic explorer (or the testnet explorer)
2. Navigate to your contract address
3. Click the *Contract* tab and *Verify & Publish*

4. Fill in the verification details:
    - Contract address
    - Compiler type (single file recommended)
    - Compiler version (must match deployment)
    - Open-source license
    - Optimization settings (if used during deployment)
5. If your contract has constructor arguments:
    - Generate ABI-encoded arguments at e.g. HashEx
    - Paste them in the Constructor Arguments field
6. Complete the captcha and submit

Method 4: Flattened Source

For contracts with complex dependencies that fail standard verification:

1. Install Hardhat flattener:

```
npm install --save-dev hardhat-flattener
```

2. Flatten your contract:

```
npx hardhat flatten contracts/YourContract.sol > flattened.sol
```

3. Clean up the flattened file:
    - Keep only one SPDX license identifier
    - Keep only one pragma statement
    - Use this file for manual verification

Troubleshooting

Common verification issues to check:

- Compiler version must match deployment exactly
- Optimization settings must match deployment
- Constructor arguments must be correctly ABI-encoded
- Library addresses must be provided if used
- Source code must match deployed bytecode exactly
- Flattened files should not have duplicate SPDX/pragma statements

Tooling and Infra

Explore the tooling and infrastructure platforms currently available on Sonic below. This list is regularly updated as new projects integrate the Sonic chain.

RPCs

- Sonic Labs
  - **https://rpc.soniclabs.com**
  - wss://rpc.soniclabs.com

- Ankr
  - https://rpc.ankr.com/sonic_mainnet
    wss://rpc.ankr.com/sonic_mainnet

- Alchemy
  - Sign up for a private RPC.

- dRPC
  - https://sonic.drpc.org
    wss://sonic.drpc.org

- thirdweb
  - https://146.rpc.thirdweb.com/${THIRDWEB_API_KEY}

Subgraphs

- Alchemy
  - Sign up on the website (*free for Sonic users*).

- Sentio
- The Graph
  - Follow the official guide.
- SQD
  - https://v2.archive.subsquid.io/network/sonic-mainnet

Oracles

- API3
  - 0x709944a48cAf83535e43471680fDA4905FB3920a
- Band Protocol
  - 0x506085050Ea5494Fe4b89Dd5BEa659F506F470Cc
- Chainlink (Data Streams)
- Chainlink (Data Feeds)
- Pyth Network (Price Feed)
  - 0x2880aB155794e7179c9eE2e38200202908C17B43
- RedStone
- Stork Network
  - 0xacC0a0cF13571d30B4b8637996F5D6D774d4fd62

Interoperability

- LayerZero
  - 0xb6319cC6c8c27A8F5dAF0dD3DF91EA35C4720dd7

Development Tools

- Sentio Explorer
- Tenderly
- thirdweb

Automation/VRF

- Gelato
  - Follow the official documentation.

- Pyth Network (Entropy)
  - 0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320

Contract Addresses

Below is a list of all important contract addresses relevant to the Sonic network.

Sonic Mainnet

Tokens:

- Wrapped S (wS)
  - 0x039e2fB66102314Ce7b64Ce5Ce3E5183bc94aD38
- Wrapped Ether (WETH)
  - 0x50c42dEAcD8Fc9773493ED674b675bE577f2634b
- USDC (Bridged)
  - 0x29219dd400f2Bf60E5a23d13Be72B486D4038894
- EURC (Bridged)
  - 0xe715cba7b5ccb33790cebff1436809d36cb17e57
- USDT (Bridged)
  - 0x6047828dc181963ba44974801ff68e538da5eaf9

Core Contracts:

- SFC
  - 0xFC00FACE0000000000000000000000000000
- Multicall3
  - 0xcA11bde05977b3631167028862bE2a173976CA11
- FTM to S Upgrade Portal (Opera)
  - 0x3561607590e28e0848ba3B67074C676D6D1C9953
- FTM to S Upgrade Portal (Sonic)
  - 0x3561607590e28e0848ba3B67074C676D6D1C9953

Gateway Infrastructure (on Sonic):

- MPTProofVerifier
  - 0xD2f1e904dAf7446686F8057b7dfeb068c75D29A9

- StateOracle
  - 0x836664B0c0CB29B7877bCcF94159CC996528F2C3
- ValidatorsRegistry
  - 0x12727D4169a42A9b5E3ECB11A6d2c95553d3f447
- MessageBus
  - 0xB5B371B75f9850dDD6CCB6C436DB54972a925308
- TokenPairs
  - 0x134E4c207aD5A13549DE1eBF8D43c1f49b00ba94
  - Implementation: 0xE34e6851a4a3763e1d27aa7ac5980d2D33C2d315
- Bridge
  - 0x9Ef7629F9B930168b76283AdD7120777b3c895b3
  - Implementation: 0x0B3fe0c10C050270a9bc34271987989B6CF2107C
- UpdateManager
  - 0x1D3c99DA3CEF5C26f02a86dC7D685efa40176bb7
  - Implementation: 0x1071405A4736535C545580064039A235827ee6D4

Gateway Infrastructure (on Ethereum):

- MPTProofVerifier
  - 0x921B147a90Ef738BBb7c2c89D88ea9d8Af3e9306
- StateOracle
  - 0xB7e8CC3F5FeA12443136f0cc13D81F109B2dEd7f
- ValidatorsRegistry
  - 0x72965045A6691E5A74299D1e878f303264D4D910
- TokenPairs
  - 0xf2b1510c2709072C88C5b14db90Ec3b6297193e4
  - Implementation: 0x0c40Ae1c82401EA741953D3f026ADc07BE9e7943
- TokenDeposit
  - 0xa1E2481a9CD0Cb0447EeB1cbc26F1b3fff3bec20
  - Implementation: 0x4cbd824685F1E21B119F230B54d65C5a7D2a5330
- DirectExitAdministrator

- 0x7390251Bf35AA7eA7C196fc4750bd5d6c5918329

- UpdateManager
  - 0xB0bECf0fBfE431D42bA0FbD8dFBFbB0DCFd62Da4
  - Implementation: 0x13bd43A6BE5795D4A4E3Efc4baC21Cd36Ae9e68A

Blaze Testnet

Tokens:

- Wrapped S (wS)
  - 0x039e2fB66102314Ce7b64Ce5Ce3E5183bc94aD38
- Wrapped Ether (wETH)
  - 0x309C92261178fA0CF748A855e90Ae73FDb79EBc7

Contracts:

- Multicall3
  - 0xcA11bde05977b3631167028862bE2a173976CA11

Network Parameters

```
{
  "Name": "sonic",
  "NetworkID": 146,
  "Dag": {
    "MaxParents": 12,
    "MaxFreeParents": 6,
    "MaxExtraData": 128
  },
  "Emitter": {
    "Interval": 100000000,
    "StallThreshold": 30000000000,
    "StalledInterval": 60000000000
  },
  "Epochs": {
    "MaxEpochGas": 15000000000,
    "MaxEpochDuration": 600000000000
  },
  "Blocks": {
    "MaxBlockGas": 1000000000,
    "MaxEmptyBlockSkipPeriod": 3000000000
  },
  "Economy": {
    "BlockMissedSlack": 50,
    "Gas": {
      "MaxEventGas": 100000000,
      "EventGas": 28000,
      "ParentGas": 2400,
      "ExtraDataGas": 25,
      "BlockVotesBaseGas": 1024,
      "BlockVoteGas": 512,
      "EpochVoteGas": 1536,
      "MisbehaviourProofGas": 71536
    },
    "MinGasPrice": 0,
    "MinBaseFee": 1000000000,
    "ShortGasPower": {
      "AllocPerSec": 2000000000,
      "MaxAllocPeriod": 5000000000,
      "StartupAllocPeriod": 1000000000,
      "MinStartupGas": 560000
    },
    "LongGasPower": {
      "AllocPerSec": 1000000000,
      "MaxAllocPeriod": 5000000000,
      "StartupAllocPeriod": 1000000000,
      "MinStartupGas": 560000
    }
  },
```

```
  "Upgrades": {
    "Berlin": true,
    "London": true,
    "Llr": false,
    "Sonic": true
  }
}
```

Programmatic Gateway

Sonic Bridge: Programmatic Usage Guide

Contract Addresses

```
// Ethereum (L1)
const ETH_CONTRACTS = {
    TOKEN_DEPOSIT: "0xa1E2481a9CD0Cb0447EeB1cbc26F1b3fff3bec20",
    TOKEN_PAIRS: "0xf2b1510c2709072C88C5b14db90Ec3b6297193e4",
    STATE_ORACLE: "0xB7e8CC3F5FeA12443136f0cc13D81F109B2dEd7f"
};

// Sonic (L2)
const SONIC_CONTRACTS = {
    BRIDGE: "0x9Ef7629F9B930168b76283AdD7120777b3c895b3",
    TOKEN_PAIRS: "0x134E4c207aD5A13549DE1eBF8D43c1f49b00ba94",
    STATE_ORACLE: "0x836664B0c0CB29B7877bCcF94159CC996528F2C3"
};
```

Setup

```
// Network RPC endpoints
const ETHEREUM_RPC = "https://eth-mainnet.g.alchemy.com/v2/YOUR_KEY";
const SONIC_RPC = "https://rpc.soniclabs.com";

// Initialize providers
const ethProvider = new ethers.providers.JsonRpcProvider(ETHEREUM_RPC);
const sonicProvider = new ethers.providers.JsonRpcProvider(SONIC_RPC);

// Initialize signer with your private key
const PRIVATE_KEY = "your-private-key";
const ethSigner = new ethers.Wallet(PRIVATE_KEY, ethProvider);
const sonicSigner = new ethers.Wallet(PRIVATE_KEY, sonicProvider);
```

Bridge Operations

1. Ethereum to Sonic Transfer

```
async function bridgeToSonic(tokenAddress, amount) {
    // 1. Check if token is supported
    const tokenPairs = new ethers.Contract(ETH_CONTRACTS.TOKEN_PAIRS, TOK
    const mintedToken = await tokenPairs.originalToMinted(tokenAddress);
    if (mintedToken === ethers.constants.AddressZero) {
        throw new Error("Token not supported");
    }

    // 2. Approve token spending
    const token = new ethers.Contract(tokenAddress, ERC20_ABI, ethSigner)
    const approveTx = await token.approve(ETH_CONTRACTS.TOKEN_DEPOSIT, am
    await approveTx.wait();

    // 3. Deposit tokens
    const deposit = new ethers.Contract(ETH_CONTRACTS.TOKEN_DEPOSIT, TOKE
    const tx = await deposit.deposit(Date.now(), tokenAddress, amount);
    const receipt = await tx.wait();

    return {
        transactionHash: receipt.transactionHash,
        mintedToken,
        blockNumber: receipt.blockNumber,
        depositId: receipt.events.find(e => e.event === 'Deposit').args.i
    };
}
```

2. Claim Tokens on Sonic

```
async function waitForStateUpdate(depositBlockNumber) {
    const stateOracle = new ethers.Contract(SONIC_CONTRACTS.STATE_ORACLE,

    while (true) {
        const currentBlockNum = await stateOracle.lastBlockNum();
        if (currentBlockNum >= depositBlockNumber) {
            return;
        }
        await new Promise(resolve => setTimeout(resolve, 30000)); // Chec
    }
}

async function generateProof(depositId) {
    // Generate storage slot for deposit
    const storageSlot = ethers.utils.keccak256(
        ethers.utils.defaultAbiCoder.encode(['uint256', 'uint8'], [deposi
    );

    // Get proof from Ethereum node
    const proof = await ethProvider.send("eth_getProof", [
        ETH_CONTRACTS.TOKEN_DEPOSIT,
        [storageSlot],
        "latest"
    ]);

    // Encode proof in required format
    return ethers.utils.RLP.encode([
        ethers.utils.RLP.encode(proof.accountProof),
        ethers.utils.RLP.encode(proof.storageProof[0].proof)
    ]);
}

async function claimOnSonic(depositTxHash, depositBlockNumber, depositId)
    // 1. Wait for state oracle update
    console.log("Waiting for state oracle update...");
    await waitForStateUpdate(depositBlockNumber);

    // 2. Generate proof
    console.log("Generating proof...");
    const proof = await generateProof(depositId);

    // 3. Claim tokens with proof
    const bridge = new ethers.Contract(SONIC_CONTRACTS.BRIDGE, BRIDGE_ABI
    const tx = await bridge.claim(depositTxHash, proof);
    const receipt = await tx.wait();

    return receipt.transactionHash;
}
```

3. Sonic to Ethereum Transfer

```
async function bridgeToEthereum(tokenAddress, amount) {
    // 1. Check if token is supported
    const tokenPairs = new ethers.Contract(SONIC_CONTRACTS.TOKEN_PAIRS, T
    const originalToken = await tokenPairs.mintedToOriginal(tokenAddress)
    if (originalToken === ethers.constants.AddressZero) {
        throw new Error("Token not supported");
    }

    // 2. Initiate withdrawal
    const bridge = new ethers.Contract(SONIC_CONTRACTS.BRIDGE, BRIDGE_ABI
    const tx = await bridge.withdraw(Date.now(), originalToken, amount);
    const receipt = await tx.wait();

    return {
        transactionHash: receipt.transactionHash,
        blockNumber: receipt.blockNumber,
        withdrawalId: receipt.events.find(e => e.event === 'Withdrawal').
    };
}
```

4. Claim Tokens on Ethereum

```
async function waitForEthStateUpdate(withdrawalBlockNumber) {
    const stateOracle = new ethers.Contract(ETH_CONTRACTS.STATE_ORACLE, S

    while (true) {
        const currentBlockNum = await stateOracle.lastBlockNum();
        if (currentBlockNum >= withdrawalBlockNumber) {
            return;
        }
        await new Promise(resolve => setTimeout(resolve, 30000)); // Chec
    }
}

async function generateWithdrawalProof(withdrawalId) {
    // Generate storage slot for withdrawal
    const storageSlot = ethers.utils.keccak256(
        ethers.utils.defaultAbiCoder.encode(['uint256', 'uint8'], [withdra
    );

    // Get proof from Sonic node
    const proof = await sonicProvider.send("eth_getProof", [
        SONIC_CONTRACTS.BRIDGE,
        [storageSlot],
        "latest"
    ]);

    // Encode proof in required format
    return ethers.utils.RLP.encode([
        ethers.utils.RLP.encode(proof.accountProof),
        ethers.utils.RLP.encode(proof.storageProof[0].proof)
    ]);
}

async function claimOnEthereum(withdrawalTxHash, withdrawalBlockNumber, w
    // 1. Wait for state oracle update
    console.log("Waiting for state oracle update...");
    await waitForEthStateUpdate(withdrawalBlockNumber);

    // 2. Generate proof
    console.log("Generating proof...");
    const proof = await generateWithdrawalProof(withdrawalId);

    // 3. Claim tokens with proof
    const deposit = new ethers.Contract(ETH_CONTRACTS.TOKEN_DEPOSIT, TOKE
    const tx = await deposit.claim(withdrawalTxHash, proof);
    const receipt = await tx.wait();

    return receipt.transactionHash;
}
```

Complete Example

```
async function bridgeUSDC() {
    try {
        // USDC details
        const USDC_ADDRESS = "0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48"
        const amount = ethers.utils.parseUnits("100", 6); // USDC has 6 d

        // 1. Bridge USDC to Sonic
        console.log("Initiating bridge to Sonic...");
        const deposit = await bridgeToSonic(USDC_ADDRESS, amount);
        console.log(`Deposit successful: ${deposit.transactionHash}`);

        // 2. Claim USDC on Sonic
        console.log("Waiting for state update and claiming on Sonic...");
        const claimTx = await claimOnSonic(deposit.transactionHash, depos
        console.log(`Claim successful: ${claimTx}`);

        // Later: Bridge back to Ethereum
        console.log("Initiating bridge back to Ethereum...");
        const withdrawal = await bridgeToEthereum(deposit.mintedToken, am
        console.log(`Withdrawal initiated: ${withdrawal.transactionHash}`

        // Claim on Ethereum
        console.log("Waiting for state update and claiming on Ethereum...
        const finalClaim = await claimOnEthereum(
            withdrawal.transactionHash,
            withdrawal.blockNumber,
            withdrawal.withdrawalId
        );
        console.log(`Final claim successful: ${finalClaim}`);
    } catch (error) {
        console.error("Bridge operation failed:", error.message);
        throw error;
    }
}
```

Required ABIs

```
const STATE_ORACLE_ABI = [
    "function lastBlockNum() external view returns (uint256)",
    "function lastState() external view returns (bytes32)"
];

const ERC20_ABI = [
    "function approve(address spender, uint256 amount) external returns (
    "function allowance(address owner, address spender) external view ret
];

const TOKEN_PAIRS_ABI = [
    "function originalToMinted(address) external view returns (address)",
    "function mintedToOriginal(address) external view returns (address)"
];

const TOKEN_DEPOSIT_ABI = [
    "function deposit(uint256 nonce, address token, uint256 amount) exter
    "function claim(bytes32 txHash, bytes calldata proof) external"
];

const BRIDGE_ABI = [
    "function withdraw(uint256 nonce, address token, uint256 amount) exte
    "function claim(bytes32 txHash, bytes calldata proof) external"
];
```

Important Notes

1. **State Updates**
   - Ethereum → Sonic: Monitor StateOracle.lastBlockNum until it's >= deposit block
   - Sonic → Ethereum: Monitor StateOracle.lastBlockNum until it's >= withdrawal block
2. **Proofs**
   - Required for all claim operations
   - Generated using eth_getProof RPC call with correct storage slots
   - Must be RLP encoded in format: `RLP.encode([RLP.encode(accountProof), RLP.encode(storageProof)])`
   - Storage slots are calculated using:
     - Deposits: `keccak256(abi.encode(depositId, uint8(7)))`
     - Withdrawals: `keccak256(abi.encode(withdrawalId, uint8(1)))`

3. **Gas Fees**
    - Keep extra ETH for gas on both networks
    - Claim operations typically cost more gas due to proof verification
4. **Security**
    - Never share private keys
    - Always verify contract addresses
    - Test with small amounts first
    - Use the same private key for both networks
5. **Monitoring**
    - Monitor transaction status on both networks
    - Keep transaction hashes for reference
    - Verify successful claims before proceeding
    - Monitor StateOracle updates for claim timing

Learn Solidity

Sonic Labs has partnered with HackQuest to offer a learning track for anyone who is eager to learn Solidity on Sonic, whether they are a beginner or advanced. Start learning now.

Upon completing the course outlined below, graduates will receive a Sonic Developer Certificate.

1 **Phase 1**

Web3 Basics.

2 **Phase 2**

Ecosystem Overview

3 **Phase 3**

Basic Solidity Syntax

4 **Phase 4**

Advanced Solidity Syntax

5 **Phase 5**

Build With Guided Projects

6 **Phase 6**

Practical Frameworks — Foundry

7 **Phase 7**

Smart Contract Security

8 **Phase 8**

Explore More Resources on Sonic

Node Deployment

Nodes are interconnected devices, usually software run on servers, that maintain the Sonic network by storing a full or partial copy of the chain, validating transactions, and ensuring consensus to keep the system secure and decentralized.

Visit the pages below to learn how to deploy archive and validator nodes on Sonic.

Archive Node

Stores the entire history of the chain since the genesis block but does not validate transactions or create new blocks.

Validator Node

Validates transactions and creates new blocks to securely power and maintain the integrity of the network.

Archive Node

Archive nodes store the entire history of the Sonic chain, including all historical states, transactions, and blocks since the genesis block.

These nodes handle historical data requests, useful for chain explorers or apps that require historical chain information. However, they do not validate transactions or create new blocks, which is the role of validator nodes.

To run an archive node on the Sonic mainnet or Blaze testnet, follow the steps below.

1. Build Sonic Client and Tooling

The minimal configuration for a Sonic archive node is a Linux server with **4 vCPU, 32 GB of RAM**, and **local SSD storage**. We recommend at least **8 vCPU** and **64 GB of RAM**, but **128GB of RAM** is generally preferable for high-demand nodes.

You will need **≈1TB of free local SSD** space to achieve good performance and speed. The configuration details depend on your specific use case.

The IOPS throughput and random access latency of the state DB persistent storage determine the performance of Sonic. For a smooth installation and fast response time, we recommend a **local NVMe** or a **local SSD** drive.

A remote block device, e.g. AWS Elastic Block Store (EBS), does not provide the required latency and IOPS performance.

> The system firewall must allow TCP and UDP connections from/to port 5050.

Building the Sonic binary requires the essential software compilation tools and the Go language version 1.22 or newer to be available. Please refer to your Linux distribution manuals to obtain the development tools installed on your system.

Build Sonic Client

Download the Sonic source code from the following GitHub repository.

```
git clone https://github.com/0xsoniclabs/Sonic.git
```

Switch to the most recent Sonic release.

```
git fetch --tags && git checkout -b v2.0.1 tags/v2.0.1
```

Build the Sonic binary using the provided configuration.

```
make all
```

Transfer the new binaries to the bin folder for system-wide access (optional).

```
sudo cp build/sonic* /usr/local/bin/
```

2. Prime Sonic State DB

Download the most recent network archive genesis file for the **Sonic mainnet or Blaze testnet**.

```
wget https://genesis.soniclabs.com/sonic-mainnet/genesis/sonic.g
```

The genesis file will be used to prime your local state database and will allow you to join the network and synchronize with it. Please check the downloaded genesis file using the provided checksum.

```
wget https://genesis.soniclabs.com/sonic-mainnet/genesis/sonic.g.md5
md5sum --check sonic.g.md5
```


The expected output is `sonic.g: OK`.

Prime Sonic Database

Use the sonictool app (created during the building process as build/sonictool) to prime a validated archive state database for the Sonic client. Start the genesis expansion.

```
GOMEMLIMIT=50GiB sonictool --datadir <datadir> --cache 12000 genesis <gen
```

The last step of the genesis processing is the state validation. Please double-check that the output log contains the following messages with details about the verified state:

Sonic Mainnet

```
StateDB imported successfully, stateRoot matches module=gossip-store :
```

Blaze Testnet

```
StateDB imported successfully, stateRoot matches module=gossip-store :
```

3. Synchronize Sonic Node With Network

With the Sonic app created and the database primed in the previous step, you are ready to start the node and synchronize its state with the network.

- Use the node database path (`<datadir>`) from the previous step.
- Use ≈90% of the RAM as the GOMEMLIMIT value.
- Use ≈12 GiB of the RAM as the --cache value.

Additional starting flags may need to be added to start Web3 RPC and WebSocket interfaces on the node. Add the following flags to enable the Web3 HTTP interface. Adjust your listening IP address, port, CORS, and list of enabled APIs for your specific needs.

```
--http --http.addr=0.0.0.0 --http.port=18545
--http.corsdomain=* --http.vhosts=*
--http.api=eth,web3,net,ftm,txpool,abft,dag,trace,debug
```

Add the following flags to enable the WebSockets interface. Adjust your listening IP address, port, origin, and list of enabled APIs for your specific needs.

```
--ws --ws.addr=0.0.0.0 --ws.port=18546 --ws.origins=*
--ws.api=eth,web3,net,ftm,txpool,abft,dag,trace,debug
```

Please obtain and explicitly specify the external IP address of your node as a starting flag to help with the inter-node peer-to-peer connectivity.

Now, start your node with the sonicd application.

```
GOMEMLIMIT=50GiB sonicd --datadir <datadir> --cache 12000 --nat extip:<pul
```

Validator Node

Validator nodes are critical to the Sonic chain, responsible for validating transactions and creating new blocks in accordance with the consensus protocol. These nodes ensure the integrity and security of the network by verifying data, participating in block creation, and maintaining the chain's state.

Unlike archive nodes, validator nodes focus on real-time operations rather than storing extensive historical data or responding to general API requests.

To run a validator node on the Sonic mainnet or Blaze testnet, follow the steps below.

Steps to Run a Validator

1. Launch a Server Instance
2. Install Required Development Tools
3. Build the Sonic Node Software
4. Prime the Sonic Database
5. Synchronize With the Network
6. Create a Validator Wallet
7. Create a Validator Signing Key
8. Register Your Validator
9. Run Your Validator Node
10. Validator Name and Logo
11. Next Steps

Validator Parameters

- Minimum self-stake amount: 500,000 S
- Maximum validator size: 15x self-stake
- Rewards: ~6% APR initially, plus 15% of delegators' rewards and network fees

The minimum self-stake on Sonic is set at 500,000 S to ensure security during the network's early months, with plans to gradually reduce it.

Network Resources

- Mainnet RPC: https://rpc.soniclabs.com
- Testnet RPC: https://rpc.blaze.soniclabs.com
- WebSocket Endpoint: wss://rpc.soniclabs.com
- Mainnet Explorer: https://sonicscan.org
- Testnet Explorer: https://testnet.sonicscan.org
- SFC Contract Address: 0xFC00FACE0000000000000000000000000000000000
- STI Contract (Validators' Information): 0xFf4cD89f549432c312c497628748d4d76AC180f6
- Genesis Files: https://genesis.soniclabs.com

1. Launch a Server Instance

You can run your validator node on dedicated hardware (bare metal) or use a cloud provider. We recommend choosing one of the established providers, such as Google GCP or Amazon AWS.

Node Hardware Specifications

- At least 4 vCPUs and 32 GB of RAM
- At least 1 Gbps redundant backbone connectivity for stable and uninterrupted network traffic
- At least 1 TB of local SSD/NVMe storage. Remote block devices (e.g., AWS EBS) typically do not provide the required latency and random access performance. Google GCP N2D instances with local SSD or AWS i3en.xlarge with instance storage are good options.
- Ubuntu LTS Server (64-bit) or a similar Linux distribution is recommended.

Required Storage Capacity

- 1 TB of local SSD/NVMe is sufficient for a validator database.

Network Settings

A Sonic node requires both **TCP** and **UDP** network traffic allowed on port 5050 by default. You can use `--port <port>` to customize this if needed.

2. Install Required Development Tools

You need the essential build tools and the latest Go compiler, version 1.22 or newer, to build the Sonic client and its bundled tools. Example (Ubuntu/Debian):

```
sudo apt-get update
sudo apt-get install -y build-essential git
```

Install Go language compiler (Ubuntu/Debian):

```
sudo rm -rf /usr/local/go
wget https://go.dev/dl/go1.23.4.linux-amd64.tar.gz
sudo tar -xzf go1.23.4.linux-amd64.tar.gz -C /usr/local/
rm -f go1.23.4.linux-amd64.tar.gz

sudo tee /etc/profile.d/golang.sh > /dev/null <<EOT
export GOROOT=/usr/local/go
export GOPATH=\$HOME/go
export PATH=\$PATH:\$GOROOT/bin:\$GOPATH/bin
EOT

source /etc/profile.d/golang.sh
```

3. Build the Sonic Node Software

Check the latest Sonic release and adjust commands if necessary:

```
git clone https://github.com/0xsoniclabs/Sonic.git
cd Sonic
git fetch --tags && git checkout -b v2.0.1 tags/v2.0.1
make all
```

You can confirm the Sonic release by running:

```
build/sonicd version
```

Optionally, copy the binaries to a system-wide location:

```
sudo mv build/sonic* /usr/local/bin/
```

4. Prime the Sonic Database

To participate in the Sonic network, you need a valid state database. The fastest way is to use a genesis file from the official repository. Download the appropriate genesis file. For the mainnet, for example:

```
wget https://genesis.soniclabs.com/sonic-mainnet/genesis/sonic.g
```

Use `sonictool` to prime the state database. Adjust `GOMEMLIMIT` and `--cache` according to your available RAM size. For the most common cases, use 12GiB as `--cache` and ~90% of RAM as `GOMEMLIMIT`.

```
GOMEMLIMIT=54GiB sonictool --datadir ~/.sonic \
    --cache 12000 genesis \
    --mode validator sonic.g
```

After processing, you should see a confirmation of a successfully imported state.

5. Synchronize With the Network

Now that your database is primed, start the node to synchronize it with the network. Ensure your firewall is configured correctly.

```
GOMEMLIMIT=54GiB sonicd --datadir ~/.sonic --cache 12000 --mode validator
```

The Sonic node will connect to the network and advance its state by processing new blocks. Once fully synchronized, the "age" of new blocks should be only a few seconds.

6. Create a Validator Wallet

Your Sonic node is now synchronizing. Next, create a wallet to identify and control your validator account. This wallet must hold the minimum amount you are going to use as the self-stake (500,000 S).

We recommend creating the wallet securely (e.g. a hardware wallet). If you choose to create it locally using sonictool:

```
sonictool --datadir ~/.sonic account new
```

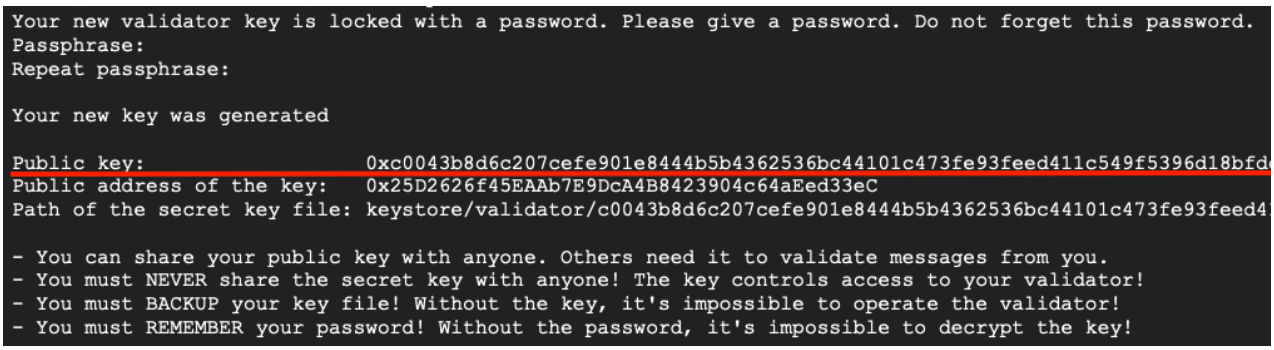
Important: Keep your wallet and keys secure. Never share them.

7. Create a Validator Signing Key

A Sonic validator node signs consensus messages. Your node needs a validator signing key to participate. Create it on the server:

```
sonictool --datadir ~/.sonic validator new
```

Follow the prompts and set a secure password. Note the public key (starts with 0xc00). This key will be used during registration.



Important: Back up your signing key. Although it cannot move funds, misuse could lead to penalties.

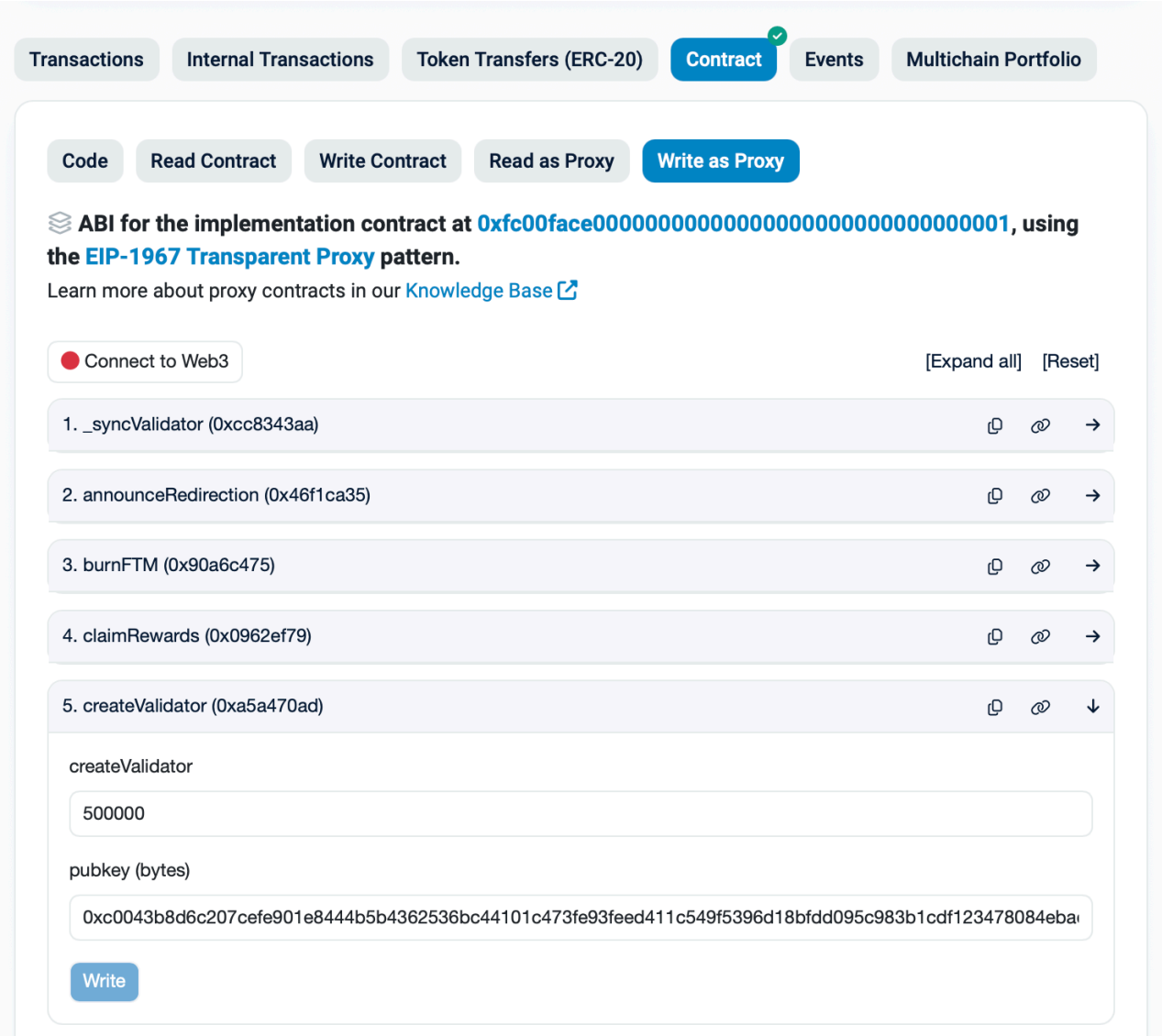
8. Register Your Validator

The network must recognize your validator key. Register by invoking the SFC (Staking and Consensus) contract:

- SFC Contract Address: 0xFC00FACE0000000000000000000000000000000

Call createValidator with your validator public key and at least the minimum required stake (500,000 S). Sign this transaction with your validator wallet. After confirmation, query your validator ID using the getValidatorID function. The easiest way would be to open the Sonic explorer and navigate to the SFC address.

The contract is validated and you can interact with it using SonicScan and a connected Web3 wallet. The control validator account can be imported into your Web3 wallet either using the generated JSON key store or as your hardware wallet. With the account open, you can sign the createValidator transaction call specifying the amount of stake and the public key obtained in the previous step.



9. Run Your Validator Node

Stop the currently running node:

```
pkill sonicd
```

Wait for it to gracefully shut down. Never force-terminate it, as it could corrupt the database. Restart your node in validator mode, providing the validator ID, public key, and password file:

```
GOMEMLIMIT=50GiB sonicd \
    --datadir ~/.sonic \
    --cache 12000 \
    --mode validator \
    --validator.id <your_validator_ID> \
    --validator.pubkey <your_public_key> \
    --validator.password <path_to_password_file>
```

Your validator node will now participate in consensus and produce blocks.

10. Validator Name and Logo

Create a config file in `JSON` format that contains the following parameters (you can also leave parameters empty):

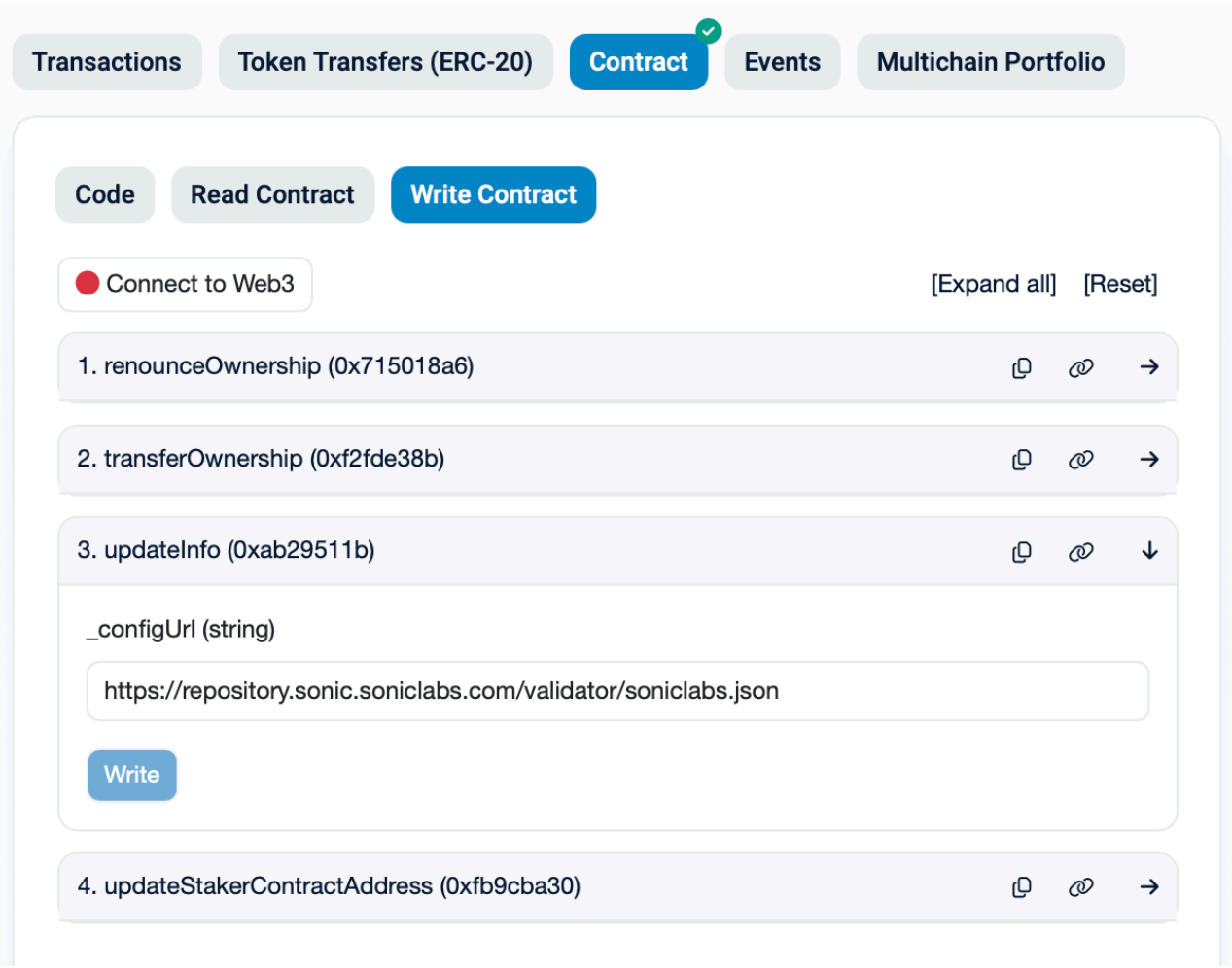
```
{
  "name": "VALIDATOR_NAME", /* Name of the validator */
  "logoUrl": "LOGO_URL", /* Validator logo (PNG|JPEG|SVG 100x100) startin
  "website": "WEBSITE_URL", /* Website icon on the right */
  "contact": "CONTACT_URL" /* Contact icon on the right */
}

/* It could look something like this 👇 */

{
  "name": "SonicLabs",
  "logoUrl": "https://repository.sonic.soniclabs.com/validator/sonic.svg"
  "website": "https://www.soniclabs.com",
  "contact": "https://www.soniclabs.com/contact"
}
```

Host it somewhere so it is publicly accessible, such as in this example. Make sure anybody can download the JSON file using a browser and that the hosting site supports HTTPS. A 100x100 logo size is sufficient.

Now you need to insert the JSON URL into the STI contract.

1. Go to SonicScan and open the STI contract.
2. Hit *Contract -> Write Contract*.

3. Find the function *updateInfo* and paste the JSON URL.
4. Connect using your validator account Web3 wallet (the one you used to register).
5. Sign and execute the update.



11. Next Steps

Validator/Name Logo

You can set a validator name and logo on the explorer. Check official repositories for instructions.

The STI contract on Sonic: 0xFf4cD89f549432c312c497628748d4d76AC180f6

Community and Help

Join Sonic community channels, follow updates, and use your stake to participate in network governance. If you have issues, the community can provide guidance.

Monitoring and Health

If your node goes offline, you stop earning rewards. Extended downtime (e.g. more than 5 days) may result in suspension, requiring you to withdraw and start over. Regular monitoring, maintenance, and backups are essential.

Congratulations! You are now running a Sonic validator node.

FAQ

Check out the most frequently asked questions about Sonic and its S token.

S and FTM Tokens

How do I upgrade my FTM to S?

FTM holders on Fantom Opera can upgrade their tokens to S on a 1:1 basis using the upgrade portal on MySonic. For the first 90 days after Sonic's mainnet launch, you can freely swap between FTM and S through this portal. After that, upgrades will be from FTM to S only.

If you hold FTM anywhere other than Opera, follow the Sonic Upgrade Handbook.

Where can I buy or sell S?

You can acquire S on decentralized exchanges (DEXs) on Sonic and most centralized exchanges (CEXs).

What if I have staked FTM?

If you have staked FTM, you can withdraw, and following a 24-hour waiting period, you can claim your FTM and upgrade to S on our upgrade portal on a 1:1 basis.

Will the S token have any inflation?

The S token will not experience inflation during the first six months after Sonic's launch. Following that period, the network will mint S tokens per previously approved governance votes to enable an airdrop, ecosystem

growth, and validator rewards. To ensure efficient use of any additional tokens, a number of new burn mechanisms and improvements are also outlined in the votes.

How does the FTM to S upgrade work?

For the first six months following Sonic's launch, you'll have the option to upgrade FTM to S on a 1:1 basis through the upgrade portal. For the first 90 days after Sonic's mainnet launch, you can freely swap between FTM and S through this portal. After that, upgrades will be from FTM to S only.

What happens if I don't upgrade my FTM to S?

If you choose to hold your FTM on the Opera network and do not upgrade it to S, no immediate changes will occur. The Opera network is expected to continue operating as normal for the foreseeable future, and you'll still have the option to upgrade your FTM to S at a later time if desired.

However, to participate in the Sonic network — including governance, transactions, and more — you'll need the S token.

Is there an S airdrop?

Yes. We're planning an airdrop of 190,500,000 S to incentivize users of both Opera and our new Sonic chain.

Sonic Chain

Is Sonic EVM compatible?

Yes. The Sonic chain is fully EVM compatible, allowing any app developed for

other EVM chains to be deployed on Sonic without changes in their code.

Can apps on Sonic be coded with Solidity and Vyper?

Yes. Since Sonic is EVM compatible, apps can be coded with both Solidity and Vyper.

How many transactions per second (TPS) can Sonic handle?

Sonic can process around 10,000 TPS with complete 1-block finality.

Is there a Sonic testnet?

Yes. You can immediately access the Sonic testnet to deploy your apps and experience the speed of Sonic.

Why create Sonic instead of continuing with Opera?

The technological advancements we achieved with our Sonic technology could not be fully integrated into Opera through a simple soft-fork upgrade. Therefore, we decided to launch an entirely new network with a new token, allowing us to usher in the next era of L1 blockchain innovation.

The benefits include:

- 10x faster node synchronization compared to Opera.
- 66% reduction in validator node costs compared to Opera.
- Live-pruning capabilities for nodes.
- Much smaller database size.
- Up to 96% reduction in costs for operating large-scale RPC nodes.
- Sub-second transaction finality.

- Fully compatible with the Sonic Gateway.

How do I migrate my app from Opera to Sonic?

Since Sonic is fully EVM compatible, migrating your app from Opera to Sonic should be straightforward. You can deploy your existing contracts on Sonic without any changes. Follow our official migration guide.

Is there an incentive program for developers to build on Sonic?

Yes. Sonic Labs is running a massive incentive program, which includes the Fee Monetization program that rewards developers with up to 90% of the fees their apps generate, and the Innovator Fund that offers up to 200,000,000 S to expedite the immediate adoption of apps to Sonic and support new ventures.

What are the tooling/infrastructure apps available on Sonic?

Check out all the tooling and infrastructure in our app explorer on MySonic.

I'm a developer who wants to launch on Sonic, how do I start?

Get started here.

Opera Chain

What will happen to the Opera chain?

Following the launch of Sonic, the Opera network will continue to be supported for FTM holders. However, Sonic Labs will focus entirely on our Sonic chain.

Will the current apps on Opera be available on Sonic?

Yes. Many of the leading apps on Opera are available on Sonic, alongside a significant number of fresh, groundbreaking apps.

Funding

Sonic Airdrop

Visit the points dashboard to track your points.

The S airdrop will distribute 190,500,000 S tokens to incentivize users of both Opera and the Sonic chain. The airdrop will be distributed using two main approaches, Sonic Points and Sonic Gems.

Sonic Points

User-focused airdrop points, earned by interacting with various tokens and apps on Sonic.

Sonic Gems

Developer-focused airdrop points, earned by driving user engagement and innovation on Sonic.

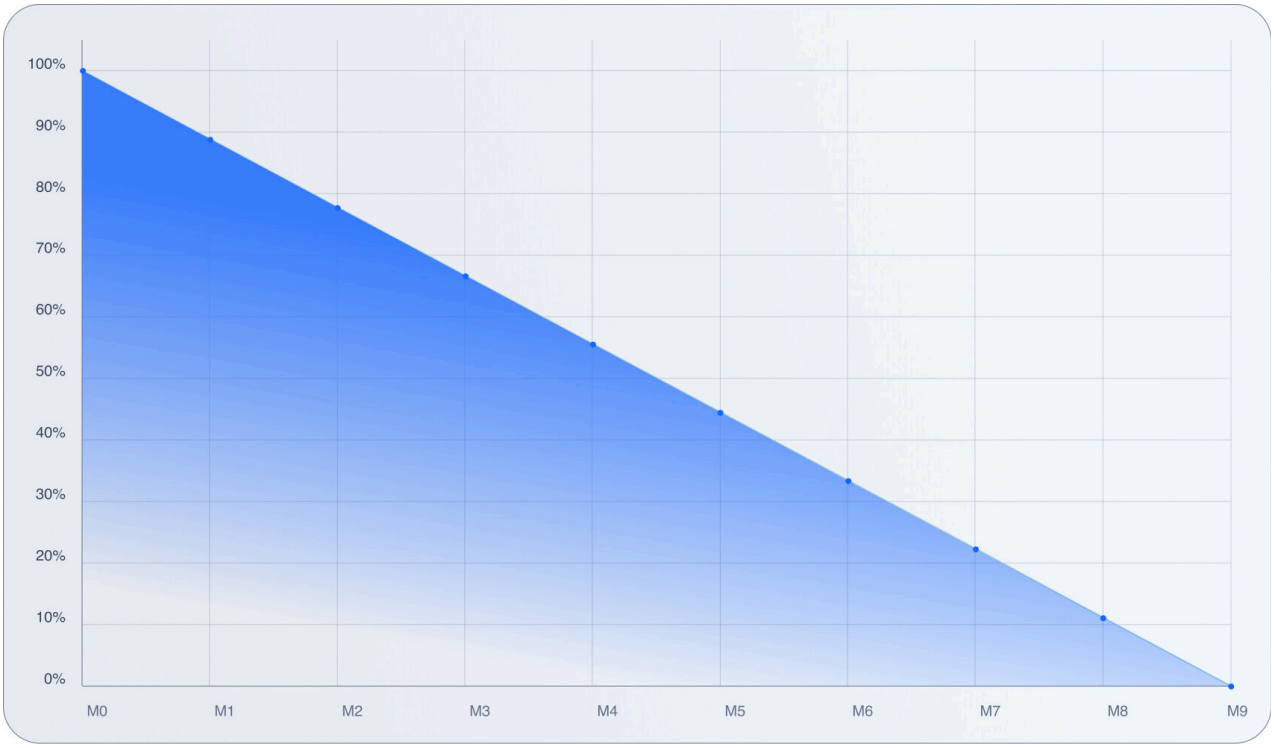
The airdrop will also reward historical activity on Opera, participation on Sonic Arcade, and minters of the exclusive Shard NFT.

Distribution

The first season of the S airdrop becomes claimable six months after Sonic's launch, around June 2025. At that time, 25% of your allocation is immediately available, while the remaining 75% vests over 270 days as NFT positions.

A linear decay mechanism applies to the vested portion, allowing you to claim early at the cost of a penalty that burns a portion of your tokens. This approach prevents sudden surges in circulating supply by burning tokens from users who choose to claim early. It also incentivizes recipients to stay active on-chain while waiting for an optimal time to claim the remainder of their allocation.

The chart below illustrates how many tokens will be burned based on when users choose to claim their vested airdrop allocation.



Burn amount of the vested airdrop allocation if claimed early

Burn Amount and Examples

Day	Burn Amount
0–30	100%–88.9%
31–60	88.9%–77.8%
61–90	77.8%–66.7%
91–120	66.7%–55.6%
121–150	55.6%–44.4%
151–180	44.4%–33.4%
180–210	33.4%–22.3%
211–240	22.3%–11.2%
241–270	11.2%–0%

Example A

Andrew earns 1,000 S in the Sonic airdrop. Andrew can claim his 25% liquid allocation immediately and receive 250 S in his wallet. 90 days pass and Andrew returns to the claim portal, deciding he would like the rest of his airdrop. As 90 days have passed by, he can claim 249.75 S tokens (33.3 of the remaining 750 token allocation) but must burn 500.25 S (66.7% of his locked allocation).

Example B

Bob earns 10,000 S in the Sonic airdrop. Bob can claim his 25% liquid allocation immediately and receive 2,500 S. Bob is excited to use his S in the ecosystem and decides he would like all of his eligible airdrop allocation after day 30. Bob claims again (the final 75% portion), receiving 1,110 S and burning 6,667.5 S (88.9% of the final portion).

For users who prefer not to wait or participate in the airdrop, there will be a speculative NFT marketplace where they can trade their airdrop allocation.

Data Collection

We are working with OpenBlock and Sentio to manage and oversee our points and airdrop designs through data-backed decision-making.

OpenBlock's data-driven incentive modeling platform powers over $2B in annual incentive spend and has been a leading provider of rewards design and efficacy analysis for leading protocols in the space including EigenLayer, Lido, Linea, Mode, Arbitrum, Solana, Sui, and many others. To initiate and sustain its ecosystem, Sonic will utilize OpenBlock's incentive modeling frameworks, ensuring continuous and balanced growth for all actors in the system.

Sentio is an industry-leading indexing service that aids in streamlining continuous point tracking, helping retrieve historical states of all user positions to update user points upon every interaction. With its high-performance database infrastructure, engineered to

handle queries at unprecedented speeds, Sentio enables the onboarding of tens of thousands of new users easily and efficiently.

Legal Provisions

As part of the Sonic Foundation's research and commitment to this airdrop, we have sought legal opinions to ensure its success.

The program will likely need to exclude any sanctioned country or person including all those who are citizens or residents of or residing in (the "Restricted Countries"): Belarus, Burundi, Central African Republic, Congo, Cuba, DPRK (North Korea), Guinea, Guinea-Bissau, Iran, Iraq, Lebanon, Libya, Mali, Myanmar (Burma), Republic of South Sudan, Russia, Somalia, Sudan, Syria, United States of America, Ukraine, the Crimea, Donetsk, and Luhansk regions of Ukraine, Venezuela, Yemen, or Zimbabwe.

These potential provisions may include:

- Geoblocking: Implementing geo-blocking measures for all restricted countries
- Self-declaration: Requiring participants to say they are not from one of the restricted countries
- Discretionary exclusion: The Sonic Foundation reserves the right to exclude any wallet address at its sole discretion

There will be a comprehensive set of terms and conditions provided at the program's launch.

Sonic Points

Visit the points dashboard to track your points.

Sonic Points are user-focused airdrop points that can be earned as part of the ~200 million S airdrop. Designed to boost liquidity on Sonic and strengthen its ecosystem, our points program positions Sonic as a premier hub for DeFi enthusiasts and users seeking to maximize the potential of their assets.

To earn Sonic Points, hold and use whitelisted assets across various DeFi apps. These points will be distributed over multiple seasons as NFT positions, ensuring long-term sustainability and preventing sudden supply shifts. The first season began with Sonic's launch and will conclude in June 2025.

How To Earn Points

1. Passive Points — Hold Assets

Users can earn passive points by holding whitelisted assets directly in their Web3 wallets, such as Rabby or MetaMask, including hardware wallets. Assets held on centralized exchanges are not eligible.

Please note that WETH, scUSD, scETH, scBTC, LBTC, SolvBTC, and SolvBTC.BBN earn activity points only, not passive points.

2. Activity Points — Deploy Assets on Apps

By deploying whitelisted assets as liquidity on participating apps, users will earn activity points, which provide 2x the amount of points compared to simply holding assets passively. A list of participating apps is available on the points dashboard.

3. App Points (Gems) — Earn Further Allocation

The S airdrop includes a developer-focused portion, where apps compete for an airdrop allocation known as Sonic Gems. Apps can redeem these Gems for S tokens, which they can then distribute to their users however they want.

To decide how these S tokens are distributed to their users, each app will run its own independent points program, entirely at its discretion. The app may consider various things, such as the amount of liquidity a user has deployed, the duration of deployment, and the specific pools to which a user has added liquidity.

As a user, you will be earning passive and activity points regardless. Your goal is to identify which app has the points program that will offer the highest return for your liquidity. The challenge lies in maximizing your overall rewards by combining the yield earned from providing liquidity with the points earned from the app's points program.

Whitelisted Assets

To qualify for the S airdrop, you must hold or use the whitelisted assets listed in the table below. The multipliers are applied to the passive or activity points you earn.

Asset	Multiplier
scUSD, stkscUSD, wstkscUSD	6x (Boosted)
USDC.e	5x (Boosted)
s, wS, stS, OS, wOS, scETH, stkscETH, wstkscETH, scBTC, stkscBTC, wstkscBTC	4x (Boosted)
WETH, LBTC, SolvBTC, SolvBTC.BBN, x33	2x (Boosted)

Please note that WETH, scUSD, scETH, scBTC, LBTC, SolvBTC, and SolvBTC.BBN earn activity points only, not passive points.

Some whitelisted assets are boosted in the first 3 months after Sonic's launch to encourage their usage and attract more liquidity to Sonic. After this period, we may choose to extend or reduce boosts.

Whitelisted assets and their multipliers are subject to change. S tokens staked through MySonic are not eligible for points. Users who wish to stake can instead use stS by Beets, a liquid staking token.

Learn more about scUSD/scETH by Rings. Sonic Labs makes no guarantee to the safety or peg of third-party assets, such as Rings. Use them at your own risk.

Airdrop Points Dashboard

The Sonic points dashboard is a comprehensive platform where users can:

- Check earned points
- Check the list of participating apps
- Get whitelisted assets through a simple interface
- Generate a referral code and share with friends to earn extra points
- View a leaderboard that displays the points and Gems earned by users and apps

Sonic Arcade Points

The Sonic Arcade was a digital playground on the Sonic testnet that featured three play-to-earn games — Plinko, Mines, and Wheel — each offering airdrop points.

Arcade players will receive their points at the end of the S airdrop's first season.

Terms and Conditions

View the terms and conditions for the Sonic points program.

Sonic Gems

Apply for your app to earn Gems.

Sonic Gems are developer-focused airdrop points designed to reward apps for driving user engagement and innovation. These Gems can be redeemed for S tokens, which apps can then distribute as rewards to their users.

This system helps apps grow by driving consistent user activity and maintaining long-term engagement. Gems are distributed across multiple seasons, ensuring a dynamic and sustainable reward structure. The first season is set to end around June 2025.

What are Sonic Gems?

Sonic Gems are airdrop points exclusively designed for apps. Each season, a fixed supply of Gems is allocated to apps based on performance metrics. Apps can track their progress through the points dashboard.

To distribute the S tokens earned through Gems to their users, apps must manage the process independently. For example, an app could:

- Mint a new token representing its share of Gems.
- Maintain an internal record of user balances.
- Use the third-party Merkl program
- Join Sonic Builders on Telegram for further support

Unlike Sonic Points, which are airdrop points designed for users, Gems empower apps to claim liquid S tokens instead of vested NFT airdrop position. Once the S tokens are

claimed, it’s the app’s responsibility to determine how they’re distributed to their users.

While there’s no strict requirement for apps to share a specific percentage of their claimed S tokens with their users, the design of Gems incentivizes generosity. Apps that share a larger portion of their claimed S with their communities are rewarded more favorably compared to those that do not.

Gems Season 1

A total of 1,680,000 Gems will be distributed during season 1. Out of this, 262,500 Gems are pre-allocated to Sonic Boom winners. The chart below shows the number of Gems allocated to each tier in Sonic Boom.

Category	Gems Per Winner
Emerald	13,125
Sapphire	8,750
Ruby	4,375

The remaining 1,417,500 Gems will be available for any app to earn throughout the season — whether they’re Sonic Boom winners or not. At the end of season 1, all eligible apps will be able to claim S tokens based on the number of Gems they have earned.

Distribution of Gems

Sonic Gems are distributed by considering factors such as category type, Sonic exclusivity, and effective reward distribution, promoting fairness and incentivizing active participation.

The competitive PvP nature and fixed supply of Gems mean that an app's Gem balance may fluctuate daily, influenced by the performance of other apps on the platform.

Below are the key criteria that will determine an app's share of Gems in season 1.

Category

Apps are assessed across several weighted categories, with each app assigned a weight based on its primary category. For season 1, the specific weights are detailed below. If an app falls into multiple categories, the weight of its dominant category will be applied.

Category	Weight
Bridges	5
CLOB DEX	4
Lending Markets / CDPs	4
Fixed Yield / Yield Tokenization	4
Spot DEX	4
GambleFi	3
Perps	3
Derivatives	3
Yield	2
Gaming	2
Tooling, Misc, and Payment	1

Sonic Native

Apps are assigned different weights depending on their level of exclusivity to Sonic:

- **Weight 2**: Exclusively available on Sonic
- **Weight 1**: Primarily on Sonic but accessible elsewhere
- **Weight 0.5**: Available across multiple chains

Note: An app's Sonic-native weight cannot be upgraded during a season. However, if an app takes actions that reduce its Sonic nativeness, its weight will be reduced immediately and remain in effect for the following season as well.

Point Score

Point score is determined by calculating the total amount of Sonic Points that an app has generated for its users. This score is then divided by the total points generated across all eligible apps.

$$\text{Point Score} = \frac{\text{Sonic Points Generated By App}}{\text{Total Sonic Points Generated}}$$

Incentive

> Only applicable from season 2 onwards.

This assesses how effectively an app distributes its claimed S to its users. An app's incentive weight is determined by the percentage of its claimed S that was distributed to its users during the previous season.

$$\text{Incentive Weight} = \frac{\text{S Distributed to Users}}{\text{S Claimed}}$$

For example, if an app distributed 100% of its claimed S to its users, it'll receive a weight of 1 in the next season, while distributing only 80% would give it a weight of 0.8.

While there's no requirement for apps to distribute a specific amount of their claimed S to users, it's mandatory for all apps to publicly disclose the percentage they intend to share with their communities.

This transparency allows users to make informed decisions about allocating their capital. Any instances of false communication or misuse of claimed S will result in blacklisting for subsequent seasons.

Final Gems Calculation

Apps will receive a pro-rata share of Sonic Gems based on their final weights, determined by the calculations below.

Step 1 — Gem Score

$$\text{Gem Score} = \text{Category Weight} \times \text{Sonic Native Weight} \times \text{Incentive Weight (N/A in Season 1)}$$

Step 2 — Point Score

$$\text{Point Score} = \frac{\text{Sonic Points Generated By App}}{\text{Total Sonic Points Generated}}$$

Step 3 — Final Score

$$\text{Final Score} = \text{Gem Score} \times (1 + \text{Point Score})$$

Step 4 — Share of Gems

$$\text{Share of Sonic Gems} = \frac{\text{App Final Score}}{\text{Total Final Score}} \times \text{Total Gems}$$

Gems Revocation Policy

The following actions by the app can cause their Sonic Gems to be revoked.

1. **Incentivizing Project Tokens or NFTs with Gems**
   Allocating Gems as rewards for activities like holding, staking, or providing liquidity (LPing) for a project's token or NFT. For apps that have a voting mechanism to direct emissions, Gems can be used as vote incentives for any pool other than those that contain the project's token.
2. **Suspicious Distribution Practices**
   Distributing large quantities of Gems non-transparently, such as allocating them disproportionately to insiders or KOLs without clear disclosure.

3. **Misrepresenting Gem Redistribution**
   Providing false information about the amount of claimed S being distributed to users during any season.

> Users are encouraged to report any suspicious activity or malpractice to Sonic Labs.

Sonic Boom

Sonic Boom was a bounty program that rewarded developers with points toward our airdrop for building innovative applications on Sonic.

Up to 30 winning projects received an allocation of Sonic Gems, which are used to claim $S in our airdrop that projects can distribute to users as rewards for using their apps. This approach helps projects bootstrap and sustain user activity by encouraging continued use and interaction.

> With Sonic Boom having concluded, check out the winners here.

Sonic Gems

Sonic Gems are developer-focused airdrop points. As a groundbreaking incentive mechanism for developers within the Sonic ecosystem, Gems reward apps for driving user engagement and innovation based on their performance on Sonic.

These Gems can be redeemed for $S tokens, which apps can then distribute as rewards to their users. This system empowers apps to kickstart growth and maintain long-term user activity by encouraging consistent interaction and participation.

37.5% of the total airdrop is allocated to Gems. The Sonic Boom program was the first initiative with which we distributed a portion of these Gems.

Allocation

We allocated a substantial amount of Sonic Gems to the winning apps of this program, recognizing up to 10 winners each in three tiers:

🏆 **Emerald**
Huge amount of Gems.

🥈 **Sapphire**
Medium amount of Gems.

🥉 **Ruby**
Small amount of Gems.

App Categories

The Sonic Boom program covered a wide range of categories, encouraging diverse and innovative applications. Multiple winners were selected in each category.

Here are the key categories:

🏦 **Lending**
Simple lending markets, isolated lending markets.

🎨 **NFT**
NFT lending markets, points trading markets.

🛠️ **Tooling**
Vote incentive markets, liquid locker protocols.

💳 **Payments**
Debit cards supporting S and USDC.

💵 **Stablecoins**
CDP stablecoins, yield-bearing stablecoins, RWA-backed stablecoins.

📈 **Yield**
Fixed yield, yield aggregators, cross-chain yield tokenization.

🎮 **Gaming**
Prediction markets, single-click games, gamified NFT games.

💱 **Exchanges**
Perpetual markets, options markets, AMMs, CLOB exchanges.

🤖 **DePIN & AI**
Wireless networks, energy grids, computation, storage.

✊ **Community Movements**
Brand ambassador campaigns, user-engagement campaigns.

Bonus Opportunities

We also offered extra rewards for Sonic builders who incorporated the following into their projects:

Fee Monetization

Participate in our Fee Monetization program, which could become a significant revenue stream for your app.

Safe

Use SAFE's tools, such as multi-signature security, backup and recovery protocols, and other DeFi compatibility features.

Sonic Native

Build your app exclusively on Sonic.

Assessment Criteria

Applications were judged on uniqueness, usability, and potential to drive user adoption on Sonic.

- **Concept**: Is the idea captivating and valuable to users?
- **Execution**: Is the project executed with excellence?
- **Impact**: What is the broader impact and sustainability of the project?
- **Points Strategy**: How effectively will the application use Sonic Gems to bootstrap success?

For inquiries or further information, contact our support team. Make sure to read our Terms and Conditions.

Winners

Below are the winners of each tier of the Sonic Boom bounty program.

Emerald Tier

Silo	Gravity Finance
Vicuna Finance	MarginZero
Yearn	Plus.Bet
Sonic Market	Rings
XPress	Soneta

Sapphire Tier

Shadow Exchange	RabbitX
Vertex	Magpie
Metropolis	Equalizer
SwapX	Stable Jack
Beethoven X	CrossCurve

Ruby Tier

SuperSonic	Avalon
Degen Express	Stability
Stryke	Lever by ZeroLend
Sacra: Falling of Myrd	Estfor Kingdom
MachFi	Lynx

Fee Monetization

Fee Monetization (FeeM) on Sonic empowers developers by allowing them to earn 90% of the network fees generated by their apps, creating a sustainable revenue stream. This model enables developers to focus on scaling their projects and growing their user base—without the constant need for fundraising or external financing.

Inspired by Web2 ad-revenue models popularized by platforms like YouTube, FeeM ensures that developers directly benefit from the traffic they drive to Sonic. By prioritizing builder rewards, Sonic sets itself apart from traditional blockchains that prioritize value extraction over ecosystem growth.

FeeM also challenges the extractive "app chain" model by eliminating high costs, infrastructure constraints, and interoperability issues. On Sonic, builders retain the majority of their app's network fees while gaining open access to world-class infrastructure, fostering a more sustainable, scalable, and developer-friendly blockchain ecosystem.

How To Participate in FeeM

Any app on the Sonic network is eligible to participate in Fee Monetization. To apply, developers can message SamHarc on Telegram to have their app added to the program.

Once accepted, developers can actively claim their rewards on the Fee Monetization dashboard.

Transaction Breakdown and Fee Structure

The transaction fee breakdown is as follows, which includes an innovative burn mechanism.

Transactions on Non-FeeM Apps

If a user submits a transaction on an app that isn't participating in FeeM, 50% of the transaction fee will be burned, and the remaining amount will be tipped to the Ecosystem Vault and validators.

Transactions on FeeM Apps

If a user submits a transaction on an app that's participating in FeeM, up to 90% of the transaction fee will be given to the app's developer, and the remaining amount will be tipped to validators.

The table below outlines the difference in transaction fee distribution between an app that does not participate in FeeM and one that does.

Type	Sonic (Non-FeeM TX)	Sonic (FeeM TX)
Burn	50%	0%
Validator Reward	45%	10%
Ecosystem Vault	5%	0%
Fee Monetization	0%	90%

Example

- **Scenario**: 50% of transactions are from apps participating in FeeM, and the remaining 50% are non-FeeM transactions.
- **Average Target Cost**: $0.01 per transaction.
- **Network Capability**: Sonic network can handle up to 900M+ transactions per day.
- **Projection**: Achieving 10 million transactions per day would result in:
  - ~$0.1 million inflow daily
  - ~$36.5 million inflow yearly
  - ~$9.125 million burned yearly
  - ~$10.095 million paid to validators yearly
  - ~$16.425 million paid to Sonic developers yearly

This projection utilizes a fraction of the network's capabilities.

Double-Spend Dilemma

In Fee Monetization, double counting is prevented by accurately tracking gas consumption within the virtual machine. The system traces all internal calls in a transaction and splits the reward based on the gas each sub-operation consumes.

This ensures that the sum of rewards across different projects never exceeds the total transaction fee. An example is given below.

- A trade consumes 100,000 units of gas with a total FeeM reward of 0.017 S.
- Inside this transaction, there are operations related to two projects: Project A and Project B.
- Project A, the DEX aggregator, is responsible for consuming 37,000 units of gas, while Project B, the liquidity pool on the DEX with which the aggregator interacts, uses 63,000 units of gas.
- The total reward is split based on the gas consumption:
    - Project A receives 37% of the reward (0.00629 S).
    - Project B receives 63% of the reward (0.01071 S).
- The total reward still adds up to 100% of the 0.017 S FeeM reward, ensuring no over-distribution.

Innovator Fund

Apply for funding from the Innovator Fund.

The Sonic Labs Innovator Fund offers up to 200,000,000 S from the Sonic Foundation treasury to expedite the immediate adoption of apps to the Sonic chain and support new innovative ventures.

Allocations

Funding from the Innovator Fund is being used to secure the best infrastructure and applications for Sonic, ensuring builders have the tools and capabilities to thrive within today's ever-challenging marketplace.

We're actively engaging with dozens of apps across the industry and top-tier infrastructure providers across on-chain tooling, compliance, native assets, cross-chain technology, wallets, indexes, strategic Web2 partnerships, and more.

To do so, we'll work closely with strategic angel investors such as Michael (Curve), Stani (Aave), Robert (Compound), Tarun (Gauntlet), and Sam (FRAX), as well as our venture partners Hashed, Signum Capital, and UOB Venture Management.

Sonic & Sodas

Apply to Sonic & Sodas with your event.

The Sonic & Sodas program empowers our community to host developer-focused networking events around the world, funded by Sonic Labs.

Think of Sonic & Sodas events as grassroots gatherings where collaboration, networking, and innovation thrive, all within the Sonic ecosystem. You host the event, and we provide the funding. Our aim is to cultivate local communities, particularly in tech hubs and around major conferences.

How It Works

The process for organizing a Sonic & Sodas event is detailed below.

1 Submit Event Proposal

Got an idea for an event? Awesome! Just submit an application or drop into our Telegram chat first for any questions. Tell us how many people you're expecting, where it's happening, and any specific support you need.

2 Plan Event

Once your proposal is approved, start planning. Choose a time that works best — whether it's breakfast, lunch, or dinner. Select a venue that's easily accessible and fosters a welcoming atmosphere for networking.

3 Get Support

We're all in to make your Sonic & Sodas a hit. We'll boost your event with marketing support, provide promotional materials, and offer educational content tailored to our ecosystem. We'll also send stickers and swag whenever we can.

4 Be Reimbursed

Once your event's over, submit your reimbursement request through Request Finance. We'll get your payment to you in crypto within a week.

5 Be Consistent

Hosting Sonic & Sodas regularly can significantly strengthen the developer community as consistent meetups enable ongoing collaboration and innovation. As your events grow, you may be able to become a community ambassador.

Got questions? Join our Telegram chat.

MIGRATION

Overview

Sonic is an EVM blockchain featuring the native S token, which is used for transaction fees, staking, running validators, and participating in governance.

Sonic is migrating from the Fantom chain and its FTM token. Users holding FTM can now upgrade to S.

Users

Users holding FTM:
If you're a user holding FTM wanting to upgrade to the S token, follow the instructions on the Sonic Upgrade Handbook.

Users holding app tokens:
If you're a user holding an app token on Opera, follow the instructions on the App Token Migration page.

Developers

If you're an Opera app developer wanting to migrate to Sonic, follow the instructions on the App Token Migration page.

Validators

If you run a validator node on Opera and you choose to migrate, please follow these steps:

- Un-delegate your stake first.
- Wait at least 2 epochs (≈15 minutes) before you shut down your node.
- Turn off the node and back up your validator key securely.

Migration Overview

Here is an overview of the migration timeline and associated events:

Event	Period
Sonic launch	December 18, 2024
Migration period	Launch onwards
Two-way swap between FTM and S	First 90 days
One-way swap (FTM to S only)	After 90 days

Opera will continue to operate, but all future development focus will shift to Sonic.

Migration FAQ

App Token Migration

This page outlines token migration options for apps from Fantom Opera to the Sonic chain.

It provides comprehensive guidance for both token holders and project owners regarding their migration options.

App Token Holders

Instructions for Token Holders

The official Sonic Labs upgrade portal will facilitate the FTM to S upgrade on a 1:1 basis.

For the first 90 days after Sonic's launch, you'll be able to swap between the two tokens freely. After this period, swaps will only be possible in one direction, from FTM to S.

Official Migration Contract Timeline

Phase	Duration	Conversion Type	Notes
Initial period	90 days	Two-way FTM ↔ S	Full flexibility
Final phase	Indefinite	One-way FTM → S	No reverse conversion

CEX-Held FTM

If you hold FTM on a centralized exchange, we're collaborating with major exchanges to enable users to upgrade FTM to S directly on their platforms. Stay tuned for updates from the CEX where you hold your FTM.

Alternatively, you can withdraw your FTM to Fantom Opera and upgrade to S through our upgrade portal.

Staked FTM

Users will be able to unlock their staked FTM and bridge immediately to Sonic following a 24-hour withdrawal waiting period. The new Sonic staking system features:

Parameter	Requirement
Withdrawal period	14 days
Minimum stake	1 S

LP Positions

If you hold liquidity positions, you'll need to:

1. Break existing LP positions on Fantom
2. Migrate each token individually following their specific processes
3. Recreate your positions on Sonic

Other Token Holders

Contact your token's project team to learn their specific migration plan. Each project will choose between migration options such as:

- Bridge migration through LayerZero
- Airdrop migration based on snapshots
- Hybrid approach combining both methods

Project Developers

Instructions for Project Owners

Projects have several options to migrate tokens from Fantom to Sonic. There is no one-size-fits-all solution — choose based on your token's setup and requirements.

Option 1: Bridge Migration

Day-one token bridging will be available through LayerZero at the Sonic launch.

Provider	Implementation	User Interface	Documentation
LayerZero	Technical OFT setup	Stargate	OFT Standard

Limitations

- Burnt LP remains on Fantom
- If LP was sent to SCC, it can be bridged and recreated
- Original token contract stays on Fantom (generally not an issue)

Option 2: Airdrop Migration

Deploy new token contract on Sonic with snapshot-based airdrop distribution.

Special Cases Requirements

Case	Handling Required	Solution
Multisigs	Yes	Different addresses on Sonic
LP tokens	Yes	Snapshot by LP weights
Burnt tokens	Optional	Can redistribute on Sonic
Initial LP	Yes	Additional funds for pairing

Advantages

- Works for burnt tokens and LPs
- No bridge needed

Limitations

- Special handling for LP and multisig holders

- More preparation work needed
- Recreation of initial burnt LP requires additional funds

Option 3: Hybrid (Beethoven X Approach)

1. Deploy new token on Sonic
2. Enable bridging of original tokens
3. Create migration contract for 1:1 swaps
4. Allow two-way conversion initially
5. Switch to one-way later

Example Implementation

- New contract: sonicBeets
- Users bridge fantomBeets to Sonic
- Migration contract handles 1:1 exchange
- Two-way conversion for limited time
- Mirrors official FTM → S migration process

Migration FAQ ›

Migration FAQ

Below are the most frequently asked questions specifically regarding the migration process from Fantom Opera to Sonic.

What if I hold FTM somewhere other than Opera?

Check out the Sonic Upgrade Handbook.

Does the upgrade portal handle app token migrations?

No, the official upgrade portal only supports FTM to S migration on Opera. Each project must implement its own migration solution.

What happens to my LP positions during migration?

LP positions must be broken, tokens migrated individually, and positions re-created on Sonic.

How long will two-way migration be available?

Two-way migration between FTM and S will be available for 90 days following launch, after which it will switch to one-way from FTM to S.

What happens if I don't migrate during the two-way period?

You'll still be able to upgrade from FTM to S but not back to FTM from S.

Will my transaction history migrate?

No, transaction history remains on the original chain. Sonic is a fresh chain with no history.

How do I know the migration method used by a token I'm holding?

Check your project's announcements or contact the team directly.

Technology

Overview

The Sonic chain provides developers with exceptional scalability and storage capabilities while delivering a fast and seamless user experience.

Sonic achieves 10,000 transactions per second with sub-second finality for immediate, irreversible transactions. There is zero risk of rollbacks or reorganizations, making every transaction final and tamper-proof. Sonic's cutting-edge storage system ensures efficient data management.

Learn more about the Sonic technology stack:

- Consensus
- Database Storage
- Proof of Stake

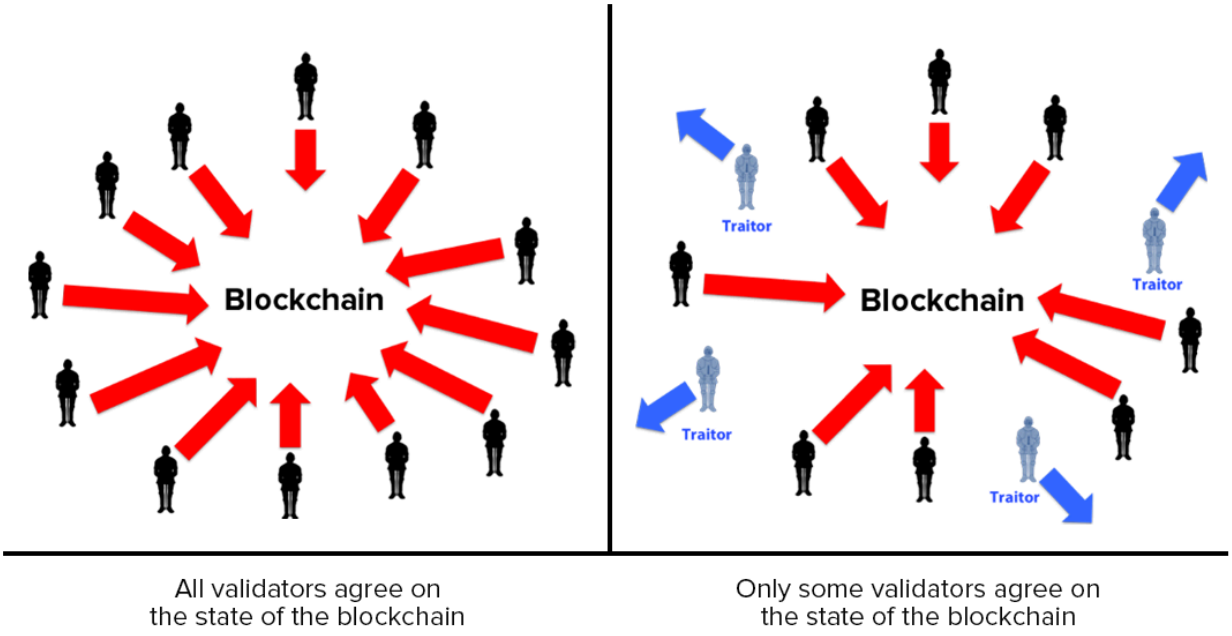
Consensus

Consensus in a decentralized system is not just a process but a cornerstone of the system. Its mechanism guarantees a consistent and secure blockchain among all participants and ensures the system's integrity and reliability. Consensus ensures that transactions are consistently and securely validated and added to the blockchain.

It's a critical element for effectively thwarting attempts by malicious actors to manipulate the network or its data. Sonic uses Asynchronous Byzantine Fault Tolerance in combination with directed acyclic graphs to achieve consensus.

Practical Byzantine Fault Tolerance

Practical Byzantine Fault Tolerance (PBFT) is a consensus mechanism designed to enable decentralized systems to function correctly in the presence of malicious or faulty nodes. It is named after the Byzantine generals' problem, which is an idea that illustrates the difficulties of achieving consensus in a decentralized system when some of the participants may be acting in bad faith.



In a PBFT system, nodes in a network communicate with each other to reach a consensus on the system's state, even when malicious actors are involved. To achieve

this, they send messages back and forth that contain information about the system's state and the actions they propose.

Each node verifies the message it receives, and if it determines the message is valid, it sends a message to all the other nodes to indicate its agreement. In the context of cryptocurrencies, the message with which all nodes must agree is the blockchain, a ledger that stores a history of transactions.

Before the invention of cryptocurrencies, the major flaw with PBFT systems was their susceptibility to Sybil attacks. If an attacker controlled a sufficient number of nodes, they could control the entire system; there needed to be a deterrent to launch many nodes. Bitcoin first solved this problem with proof-of-work, forcing nodes to invest considerable energy to partake in the consensus.

Since then, many new solutions have been developed, such as proof-of-stake, which forces nodes to deposit tokens with monetary value, which Sonic uses.

Hence, Practical Byzantine Fault Tolerance (PBFT) is a mechanism to achieve consensus. It forms a functioning decentralized system when coupled with proof-of-work or proof-of-stake to deter participants from messing with the network. However, Sonic has decided to innovate on this mechanism by using **Asynchronous Byzantine Fault Tolerance**.

Asynchronous Byzantine Fault Tolerance

With Asynchronous Byzantine Fault Tolerance (ABFT), nodes can reach consensus independently and are not required to exchange final blocks sequentially to confirm transactions. At the same time, they exchange blocks, which is required to achieve consensus, and this is done asynchronously. Each node verifies transactions independently and is not required to incorporate blocks created by other miners or validators in sequential order.

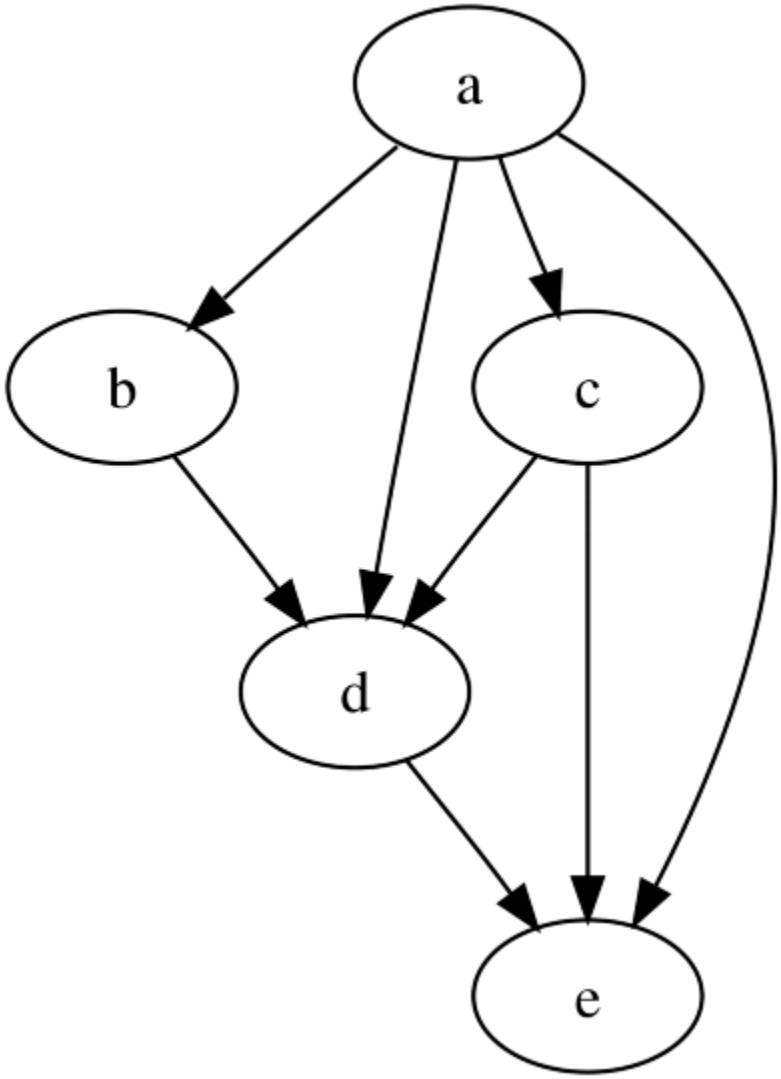
This is opposed to PBFT systems, such as Bitcoin, in which the majority of nodes must agree to a block before it becomes final, which they must then sequentially order into their blockchain record. This slows down the network during high traffic; more on this in the consensus mechanism section further below.

Now that we have a basic understanding of Byzantine fault tolerance, let's delve into the second part of Sonic's consensus mechanism, directed acyclic graphs.

Directed Acyclic Graphs

A graph is a non-linear data structure used to represent objects, called vertices, and the connections between them, called edges. A **directed graph** dictates that all its edges, the connections between objects, only flow in a certain direction. An **acyclic graph** does not contain any cycles, which makes it impossible to follow a sequence of edges and return to the starting point. As such, a **directed acyclic graph (DAG)** only flows in a certain direction and never repeats or cycles.

The diagram below is an example of a directed acyclic graph. Each oval is a vertex, and the lines connecting them are edges. The vertices only connect in one direction, downwards, and never repeat.

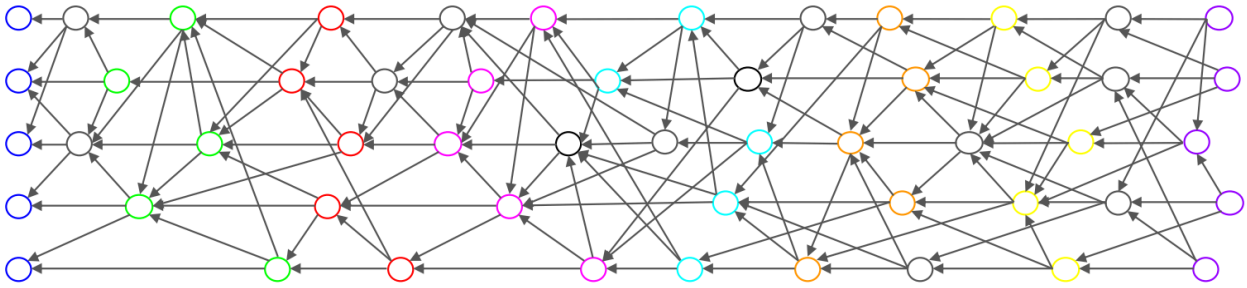


In our consensus algorithm, an event containing transactions is represented by a vertex in a DAG, and edges represent the relationships between the events. The edges may represent the dependencies between events indicating the order in which they were added to the DAG.

Events can be created and added to the DAG concurrently. The blocks do not need to be added in a specific order, which enables the system to achieve faster transaction times. It is not limited by the requirement to incorporate blocks sequentially, as is the case with many of the biggest blockchains currently available.

Sonic's Consensus Mechanism

Sonic uses a proof-of-stake, DAG-based, ABFT consensus mechanism. In this mechanism, each validator has its own local block DAG and batches incoming transactions into event blocks, which they add to their DAG as vertices — each event block is a vertex in the validator's DAG that is full of transactions.

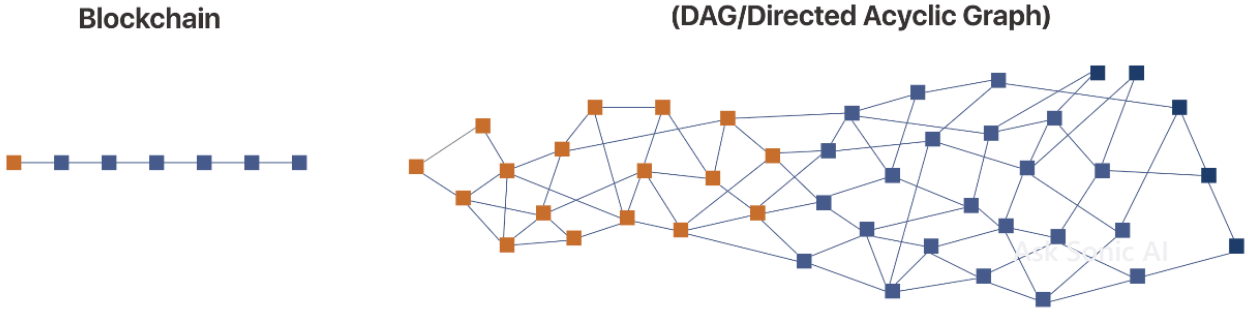


A validator's block DAG

Before creating a new event block, a validator must first validate all transactions in its current event block and part of the ones it has received from other nodes; these are the event blocks it has received during the asynchronous exchange of event blocks explained in the section above. The new event block then is communicated with other nodes through the same asynchronous event communication.

During this communication, nodes share their own event blocks, and the ones they received from other nodes, with other validators that incorporate them in their own local DAGs. Consequently, this spreads all information through the network. The process is asynchronous as the event blocks shared between validators are not required to be sequential.

Unlike most blockchains, this DAG-based approach does not force validators to work on the current block that is being produced, which places restrictions on transaction speed and finality. Validators are free to create their own event blocks that contain transactions and share these with other validators on the network asynchronously, creating a non-linear record of transactions. This increases transaction speed and efficiency.



Source: Central Blockchain Council of America

As an event block is sent and propagated across validators, it becomes a root event block once the majority of validators have received and agreed upon it. This root event block will eventually be ordered and included in the *main chain,* which is a blockchain that contains the final consensus among all event blocks that have become root event blocks.

Every validator stores and updates a copy of the *main chain*, which provides quick access to previous transaction history to process new event blocks more efficiently. As such, Sonic's consensus mechanism combines a DAG-based approach that allows validators to confirm transactions asynchronously, which greatly increases speed, with a final blockchain that orders and stores all final transactions immutably and indefinitely.

Currently, the process of submitting a transaction and having it added to the Sonic *main chain* through the consensus mechanism takes approximately 1-2 seconds. This involves the following steps:

1. A user submits a transaction
2. A validator node batches the transaction into a new event block
3. The event block becomes a root event block once the majority of nodes have received it
4. The root event block is ordered and finalized into the *main chain* as a block

When a user explores Sonic through a block explorer, they view the final blocks on the Sonic *main chain*. Event block generation and exchange in validators' DAGs is an internal process only and is not visible to end users.

To better understand the technical aspects of Sonic's consensus mechanism, review the paper on Fantom Opera's consensus mechanism, as Sonic's design is a continuation of it:

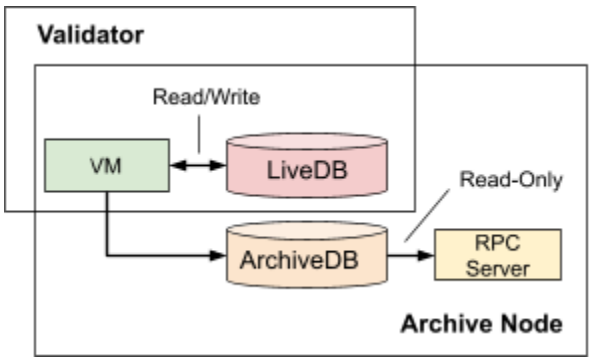
Lachesis: Scalable Asynchronous BFT on DAG Streams.

Database Storage

Sonic uses database storage to store its world state, which includes account information, virtual machine bytecode, smart contract storage, etc. This database has a feature called live pruning, which removes historical data automatically, reducing storage needs for validators as the blockchain grows.

Previously, pruning required validator nodes to go offline, risking financial and operational issues for them. Now, validators can use live pruning without going offline, ensuring continuous operation and saving on disk space and costs by discarding historical data in real-time.

Live pruning works by splitting the database into two types: LiveDB and ArchiveDB. The LiveDB contains the world state of the current block only, whereas the ArchiveDB contains the world states of all historical blocks. Validators use only LiveDB, while archive nodes have both LiveDB and ArchiveDB to handle historical data requests through the RPC interface.



Sonic's database storage uses efficient tree-like or hierarchical structures, which simplifies data retrieval. Importantly, it still provides cryptographic signatures for a world state and archive capabilities using an incremental version of a prefix algorithm. Additionally, it utilizes a native disk format instead of storing the world state indirectly through key-value stores like LevelDB or PebbleDB.

Proof of Stake

The Sonic chain is secured using a proof-of-stake (PoS) mechanism.

In PoS on Sonic, validators must lock their S (Sonic's native token); if they act maliciously in the network, they lose their tokens. Validators are incentivized to act in the network's best interest as their own funds are at stake. Since validators do not need to perform computations, this approach is a much more energy-efficient alternative to proof-of-work for achieving resistance to Sybil attacks.

A Sybil attack is an attack where a malicious actor runs a large number of validators to allow them an unsafe amount of influence over the network. PoS makes it costly to set up these validators and allows the network to punish validators for malicious behavior, increasing the costs of attacks.

Sonic requires nodes to lock up at least 500,000 S to validate transactions and produce blocks.

Audits

Sonic has undergone extensive security audits across its core components to ensure the highest level of security for our users and developers.

Sonic Gateway

The Sonic Gateway is our native bridge between Ethereum and Sonic.

- OpenZeppelin
- Quantstamp
- Certora

FTM to S Bridge

The FTM to S bridge enables users to upgrade their FTM on Fantom Opera to S on Sonic.

- OpenZeppelin

Special Fee Contract (SFC)

The SFC manages staking and validator operations on Sonic.

- Quantstamp