Deploying and Verifying Contracts

Deploy with Foundry or Hardhat, verify source code, use proxies safely and follow a launch checklist.

VERSION	LAST UPDATED	STATUS
0.1	November 17, 2026	Draft

1. Environments

Neo Smart Chain provides a public testnet and a mainnet. Always deploy to the testnet first and run your full test plan there before you deploy to mainnet.

Environment	Chain ID	RPC URL	Explorer
Testnet	[testnet chain ID]	[testnet RPC URL]	[testnet explorer URL]
Mainnet	[mainnet chain ID]	[mainnet RPC URL]	[mainnet explorer URL]

Store these values in environment variables so the same scripts work on both networks.

2. Deploying with Foundry

Foundry deploys through scripts. Use an encrypted keystore, or a hardware wallet for mainnet, instead of a private key in the command line.

```
# Encrypted keystore
cast wallet import deployer --interactive

# Simulate first, without sending anything
forge script script/Deploy.s.sol:Deploy --rpc-url $NSC_RPC_URL --account deployer

# Broadcast
forge script script/Deploy.s.sol:Deploy \
--rpc-url $NSC_RPC_URL --account deployer --broadcast

# Hardware wallet (mainnet)
forge script script/Deploy.s.sol:Deploy \
--rpc-url $NSC_RPC_URL --ledger --broadcast
```

Foundry records each broadcast in the broadcast folder, including addresses and transaction hashes. Commit it so your deployment history is part of your repository.

3. Deploying with Hardhat

Add the network to hardhat.config.ts and keep the compiler settings in line with the supported EVM version.

```
import { HardhatUserConfig } from 'hardhat/config'
import '@nomicfoundation/hardhat-toolbox'

const config: HardhatUserConfig = {
solidity: {
version: '0.8.28',
settings: {
evmVersion: 'paris',
optimizer: { enabled: true, runs: 200 },
},
},
networks: {
neo: {
url: process.env.NSC_RPC_URL ?? '',
chainId: Number(process.env.NSC_CHAIN_ID),
accounts: process.env.DEPLOYER_KEY ? [process.env.DEPLOYER_KEY] : [],
},
},
}

export default config
```

```
import { ethers } from 'hardhat'

async function main() {
const [deployer] = await ethers.getSigners()
const token = await ethers.deployContract('NeoToken', [
deployer.address,
```

```
ethers.parseEther('1000000'),
])
await token.waitForDeployment()
console.log('NeoToken deployed to', await token.getAddress())
}

main().catch((error) => {
console.error(error)
process.exitCode = 1
})
```

```
npx hardhat run scripts/deploy.ts --network neo
```

> Use a private key in an environment variable only for the testnet. For mainnet, deploy from a hardware wallet or a multisig, and never commit key files.

4. Verifying source code

Verify every contract. A verified contract shows its source, ABI and read and write tabs on the explorer, and registry listings require it.

```
forge verify-contract <CONTRACT_ADDRESS> src/NeoToken.sol:NeoToken \
--chain-id $NSC_CHAIN_ID \
--verifier $NSC_VERIFIER \
--verifier-url $NSC_VERIFIER_URL \
--constructor-args $(cast abi-encode \
"constructor(address,uint256)" $OWNER $(cast to-wei 1000000))

# Hardhat
npx hardhat verify --network neo <CONTRACT_ADDRESS> <OWNER> 1000000000000000000000000
```

Verification fails when the compiler version, optimizer settings, EVM version, libraries or constructor arguments differ from the deployment. See the troubleshooting guide for the most common causes.

Explorer verification method and API: [verifier details].

5. Upgradeable contracts

Upgradeable contracts let you fix bugs after launch, but they add trust assumptions. Holders must trust whoever controls the upgrade. Use them only when you need them, and disclose the upgrade authority in your documentation.

- Use OpenZeppelin upgradeable contracts with a UUPS or transparent proxy that follows ERC-1967.

- Replace constructors with an initializer function and protect it so it can only run once.
- Never reorder or remove existing storage variables in a new version.
- Put the upgrade right behind a multisig with a time delay. Do not leave it on a single key.
- Run the OpenZeppelin upgrade safety checks, or forge-based storage layout comparisons, before every upgrade.

6. Mainnet launch checklist

1. All tests pass, including fuzz and invariant tests, and coverage covers every public function.
2. The contract has been deployed and exercised end to end on the testnet.
3. An independent audit is complete and every high and medium finding is fixed or documented.
4. Admin keys sit behind a multisig, and the signers are documented.
5. Compiler version, optimizer settings and EVM version match the network and are pinned.
6. The deployer address is funded with enough native currency for gas, and nothing else.
7. The contract is verified on the explorer immediately after deployment.
8. Addresses and transaction hashes are recorded in your repository and in your token documentation.
9. Monitoring and alerts are set up for large transfers and admin actions.
10. A security contact and, where possible, a bug bounty are published.

This document is for information only and is not financial, legal or tax advice. Highlighted items in square brackets are still to be confirmed by the issuer. The latest version is always available at neosmartchain.com/developers/deploy