Quickstart: hello world! hello zkSync

On This Page * [Prerequisites *][Fund your wallet *][Create the project *][Compile the contract *][Deploy and verify *][Interact with the contract *][Takeaways *][Next steps *]

#

Quickstart: hello world! hello zkSync

Time to complete: <5 minutes

This guide shows you how to deploy and interact with a smart contract on zkSync in less than 5 minutes. It will help you get familiar with the zkSync development toolbox.

This is what we're going to do:

- Fund your wallet with zkSync testnet ETH.
- Usezksync-cli
- to scaffold a new project.
- Build a smart contract that stores a greeting message and deploy it to zkSync Era testnet.
- Run a script to retrieve and update the greeting message usingzksync-ethers
- .

#

Prerequisites

- Make sure your machine satisfies thesystem requirementsopen in new window
- .
- Download and installNode.jsopen in new window
- .
- Use theyarn
- ornpm
- package manager. We recommend usingyarn
- . To installyarn
- , follow theYarn installation guideopen in new window
- .
- Learn how toget your private key from your Metamask walletopen in new window
- as we'll use it to deploy a contract.

#

Fund your wallet

You can get testnet ETH directly into zkSync testnet using thefollowing faucet provided by LearnWeb3open in new window.

Another option is toget SepoliaETH from any of the following faucetsand bridge it to zkSync Sepolia testnet using thezkSync bridgeopen in new window.

You can check the balance of your account in thezkSync Sepolia exploreropen in new window.

#

Create the project

Project available in Atlas IDE

This entire tutorial can be run in under a minute using Atlas. Atlas is a smart contract IDE that lets you write, deploy, and interact with contracts from your browser.

Open this project in Atlasopen in new window. Run the following command in your terminal to create a new project using zkSync CLI.

npx zksync-cli create hello-zksync It will give you options for different types of projects but for this tutorial choose the the following:

? Whattype of projectdo you want to create? Contracts ? Ethereum framework: Ethers v6 ? Template: Hardhat + Solidity ? Private key of the wallet responsiblefor deploying contracts(optional)

? Package manager:yarn Info

The private key of your wallet will be included in the.env file of the project and won't be pushed to GitHub. The project structure is pretty straight forward:

- hardhat.config.ts
- contains the general configuration for Hardhat and the zkSync plugins, which are already imported and setup.
- /contracts
- contains smart contracts.zksync-cli
- provides common examples like an ERC20, an NFT, and the Greeter contract that we'll use later on.
- /deploy
- contains the deployment scripts.

For this tutorial we'll focus on the/contracts/Greeter.sol contract:

//SPDX-License-Identifier: Unlicense pragma solidity ^ 0.8.0 ; contract Greeter { string private greeting; constructor (string memory _greeting) { greeting= _greeting; } function greet ( ) public view returns (string memory) { return greeting; } function setGreeting (string memory _greeting) public { greeting= _greeting; } } As you can see, it a simple Solidity contract with two methods to read a message,greet() , and modify it,setGreeting() .

Takeaway

zkSync is EVM compatible. You can write smart contracts with Solidity or Vyper and use existing popular libraries like OpenZeppelin.

#

Compile the contract

Smart contracts deployed to zkSync must be compiled using our custom compilers:

- zksolc
- for Solidity contracts.
- zkvyper
- for Vyper contracts.

As this is a Solidity project, it already has thehardhat-zksync-solc plugin installed and configured so there's nothing you need to setup. To compile the contracts in the project, run the following command:

yarn npm [yarn compile npm run compile] You'll get the following output:

Compiling contractsfor zkSync Era with zksolc v1.3.21 and solc v0.8.17 Compiling46 Solidity files Successfully compiled46 Solidity files ✨ Donein 21 .55s. The compiled artifacts will be located in the/artifacts-zk folder. These artifacts are similar to the ones generated by the Solidity compiler. For example, the ABI of the Greeter contract will be located in/artifacts-zk/contracts/Greeter.sol/Greeter.json .

Takeaway

Smart contracts deployed to zkSync must be compiled usingzksolc orzkvyper as they generate a custom bytecode compatible with zkSync's ZKEVM. The configuration for thezksolc compiler is located in thezksolc section of thehardhat.config.ts file. You can find more info about the compiler settings in thehardhat-zksync-solc plugin and thecompiler section of the ZK Stack documentation.

#

Deploy and verify

The project also contains a script to deploy and verify the contract in/deploy/deploy.ts . Under the hood, this script useshardhat-zksync-deploy andhardhat-zksync-verify for deployment and contract verification.

import { deployContract} from "./utils" ; // An example of a basic deploy script // It will deploy a Greeter contract to selected network // as well as verify it on Block Explorer if possible for the network export default async function ( ) { const contractArtifactName= "Greeter" ; const constructorArguments= ["Hi there!"] ; await deployContract (contractArtifactName, constructorArguments) ; } To execute it, just run:

yarn npm [yarn deploy npm run deploy] You'll get the following output:

Starting deployment process of"Greeter" .. . Estimated deployment cost:0.0000648863 ETH"Greeter" was successfully deployed: - Contract address: 0x0BaF96A7f137B05d0D35b76d59B16c86C1791D8D - Contract source: contracts/Greeter.sol:Greeter - Encoded constructor arguments: 0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000000948692074686572652100000000000000

Requesting contract verification.. . Your verification ID is:1781 Congratulations! You just deployed a smart contract to zkSync Sepolia testnet. You can find it in thezkSync Sepolia exploreropen in new window by searching the contract address.

In addition, the deployment script verified the contract automatically so you can see the source code in the contract tab of the block explorer.

#

Interact with the contract

The project also comes with a script to interact with the contract in/deploy/interact.ts . Add the address of the Greeter contract you just deployed in theCONTRACT_ADDRESS variable inside the/deploy/interact.ts file:

import * as hrefrom "hardhat" ; import { getWallet} from "./utils" ; import { ethers} from "ethers" ; // Address of the contract to interact with const CONTRACT_ADDRESS = "" ; if (! CONTRACT_ADDRESS) throw " Provide address of the contract to interact with!" ; // An example of a script to interact with the contract export default async function ( ) { console . log (Running script to interact with contract { CONTRACT_ADDRESS }) ; // Load compiled contract info const contractArtifact= await hre. artifacts. readArtifact ("Greeter") ; // Initialize contract instance for interaction const contract= new ethers . Contract (CONTRACT_ADDRESS , contractArtifact. abi, getWallet ( ) // Interact with the contract on behalf of this wallet) ; // Run contract read function const response= await contract. greet ( ) ; console . log (Current message is: { response}) ; // Run contract write function const transaction= await contract. setGreeting ("Hello people!") ; console . log (Transaction hash of setting new message: { transaction. hash}) ; // Wait until transaction is processed await transaction. wait ( ) ; // Read message after transaction console . log (The message now is: { await contract. greet ( ) }) ; } As you can see, we're simply usingethers to interact with our contract. zkSync is EVM compatible so you can use existing tools and libraries likeHardhat ,ethers ,web3.js , and users can use their existing wallets like Metamask, Rabby or Zerion.

Takeaway

Existing libraries likeethers andweb3.js can be used to interact with smart contracts deployed on zkSync. To execute the/deploy/interact.ts script, run:

yarn npm [yarn interact npm run interact] You'll get the following output:

Running script to interact with contract 0x0BaF96A7f137B05d0D35b76d59B16c86C1791D8D Current message is: Hi there! Transactionhash of setting new message: 0x7a534857cfcd6df7e3eaf40a79a2c88f2e36bb60ce6f2399345e5431362b04eb The message now is: Hello people! ✨ Donein 4 .13s. Congratulations! You've retrieved and updated the message on theGreeter contract. You can see the transactionin the block exploreropen in new windowby searching the transaction hash.

#

Takeaways

- zkSync is EVM compatible and you can write smart contracts in Solidity or Vyper, use Hardhat, libraries like Ethers and Web3.js, or wallets like Metamask and Rabby.
- zkSync CLI provides a quick way to scaffold different types of projects thanks to its multiple templates.
- Contracts deployed to zkSync are compiled usingzksolc
- orzkvyper
- as they generate a special bytecode for zkSync's ZKEVM.

#

Next steps

This was your first step towards becoming a zkSync developer. Here is what you can do next:

- Create a frontend for this contract following theFrontend quickstart
- .
- Join ourdeveloper community in GitHubopen in new window
- , where you can ask questions and help other developers.
- Read theSecurity and best practices
- to keep you apps secure.
- Learn about thedifferences between Ethereum and zkSync
- .
- If you have a project, check out ourmigration guide
- .

[ ]Edit this pageopen in new windowLast update: Contributors: [[Antonio ,][Nicolas Villanueva ,][Samui68 ,][albicodes]]

Next Interact with zkSync Era