[Telegram:@elonmusk1950m][Twitter:@elonmusk1950m][Email:elonmusk1950m@gmail.com]

2901.部署 ZKS Era 开发环境

● 项目介绍

ZkSync 是一个用于 Etherum 的扩展和隐私引擎。其目前的功能范围包括以太网络中 ETH 和 ERC20 令牌的低气体传输。本文档描述了可用于与 zkSync 交互的 JS 库。ZkSync 是基于 ZK Rollup 架构构建的。ZK Rollup 是一个 L2 缩放解决方案，其中所有资金都由主链上的一个智能契约持有，而计算和存储则在主链外执行。对于每个 Rollup 块，由主链契约生成并验证一个状态转换零知识证明(SNARK)。这个 SNARK 包含了对总结块中每一个事务的有效性的证明。此外，每个块的公共数据更新都通过主链网络在廉价的调用数据中发布。这种体系结构提供了以下保证：Rollup 验证器永远不会破坏国家或窃取资金(不同于 Sidechains)。即使验证器因为数据可用(不同于等离子体)而停止合作，用户仍然可以从汇总中检索资金。由于有效性证明，无论是用户还是其他任何一个可信方都不需要在线监视 Rollup 块，以防止欺诈。换句话说，ZK Rollup 严格继承了底层 L1 的安全保证。

● 相关新闻

PANews 2 月 16 日消息，基于 ZK Rollup 的以太坊二层网 zkSync 发推表示，zkSync Era 主网正式上线，将向所有开发者开放主网，需注册部署，并开放其整个代码库；同时 zkSync 2.0 更名为 zkSync Era（这里要和 zkSync Lite 做区分，后期都是与 zkSync Era 做交互）

zkSync 1.0 更名为 zkSync Lite。从测试网到主网经历了四个月，期间测试网上共有大约 900 万笔交易，部署了 3 万份合约，49.7 万个活跃地址。新路线图将很快发布，下一个目标是全面启动 Alpha。

● 相关地址

官网：https://zksync.io/

Zks Era 开发文档：https://era.zksync.io/docs/dev/

区块链浏览器：https://explorer.zksync.io/

桥：https://bridge.zksync.io/

交易所：https://izumi.finance

● 网络信息

主网网络信息

- ✧ 网络名称：zkSync Era Mainnet
- ✧ 远程过程调用地址：https://mainnet.era.zksync.io
- ✧ 链号：324
- ✧ 货币符号：ETH
- ✧ 区块浏览器网址：https://explorer.zksync.io/
- ✧ WebSocket 网址：wss://mainnet.era.zksync.io/ws

试网网络信息

- ✧ 网络名称：zkSync Era Testnet
- ✧ 远程过程调用地址：https://testnet.era.zksync.dev
- ✧ 链号：280
- ✧ 货币符号：ETH
- ✧ 区块浏览器网址：https://goerli.explorer.zksync.io/
- ✧ WebSocket 网址：wss://testnet.era.zksync.dev/ws

● 开发差异

1、开发工具

zksync era 部署合约和 ETH 等网络不同，不能直接使用 remix 进行部署，官方出的解决方案是使用 hardhat 插件。

2、Constructor

合约中的 constructor 需要传参进去，不能直接写入

2、时间戳

如果在合约中使用了 block.timestamp，编译器会生成一个警告。这是因为 block.timestamp 提供了当前区块的时间戳信息，但在智能合约中，时间戳的使用需要特别注意。如果时间戳对您的合约行为不重要，您可以选择忽略该警告。然而，如果您的合约依赖于时间戳以进行某些操作或决策，那么请务必仔细查看 Solidity 文档和相关的最佳实践，以确保您正确处理时间

戳，避免潜在的安全风险和不稳定性。同时，注意在不同以太坊网络上，时间戳的可用性和精度可能会有所不同，因此确保您的合约在不同网络上都能正确工作。

```
Warning: You are using 'block.timestamp' in your code, which might lead to unexpected behaviour.
'block.timestamp' actually refers to the timestamp of the whole batch that will be sent to L1.
We are planning to change this in the near future so that it returns the timestamp of the L2
block. A separate method will be introduced for accessing the timestamp of the L1 batch.
More information here:
https://github.com/zkSync-Community-Hub/zkync-developers/discussions/32
```

参考地址：https://github.com/zkSync-Community-Hub/zkync-developers/discussions/32

3、代币转出

使用 `call` 超过 `.send` 或 `.transfer`

避免使用 `payable(addr).send(x)` / `payable(addr).transfer(x)` ，因为 2300 Gas 津贴可能不足以满足此类调用，特别是当它涉及需要大量 L2 Gas 数据的状态更改时。相反，我们建议使用 `call` .

代替：

```solidity
payable(addr).send(x) // or
payable(addr).transfer(x)
```

使用：

```solidity
(bool s, ) = addr.call{value: x}("");
require(s);
```

这会将 `send` / `transfer` 功能转换为此处概述的潜在安全风险 `call` ，并避免潜在的安全风险。 。

> **⚠ 注意重入**
>
> 虽然与 or `.call` 相比提供了更大的灵活性，但开发人员应该意识到它不提供与/相同级别的重入保护。遵守检查-效果-交互模式等最佳实践和/或使用重入防护来保护您的合约免受重入攻击至关重要。即使在意外情况下，它也可以帮助确保 zkEVM 上的智能合约的稳健性和安全性。 `.send` `.transfer` `.call` `.transfer` `.send`

参考地址：https://era.zksync.io/docs/dev/building-on-zksync/best-practices.html#use-call-over-send-or-transfer

4、地址推导

对于 zkEVM 字节码，zkSync Era 使用与以太坊不同的地址派生方法。具体的公式可以在我们的SDK中找到，如下所示：

```ts
export function create2Address(sender: Address, bytecodeHash: BytesLike, salt: BytesLi
  const prefix = ethers.utils.keccak256(ethers.utils.toUtf8Bytes("zksyncCreate2"));
  const inputHash = ethers.utils.keccak256(input);
  const addressBytes = ethers.utils.keccak256(ethers.utils.concat([prefix, ethers.utils
  return ethers.utils.getAddress(addressBytes);
}

export function createAddress(sender: Address, senderNonce: BigNumberish) {
  const prefix = ethers.utils.keccak256(ethers.utils.toUtf8Bytes("zksyncCreate"));
  const addressBytes = ethers.utils
    .keccak256(ethers.utils.concat([prefix, ethers.utils.zeroPad(sender, 32), ethers.u
    .slice(26);

  return ethers.utils.getAddress(addressBytes);
}
```

由于字节码与以太坊不同，因为 zkSync 使用 EVM 的修改版本，因此从字节码哈希派生的地址也会不同。这意味着在以太坊和 zkSync 上部署的相同字节码将具有不同的地址，并且以太坊地址在 zkSync 上仍然可用且未使用。如果 zkEVM 与 EVM 达到奇偶校验，地址派生将更新以匹配以太坊，并且相同的字节码将在两条链上具有相同的地址，部署到 zkSync 上不同地址的字节码可以部署到相同的以太坊 - zkSync 上的匹配地址。

参考地址：https://era.zksync.io/docs/reference/architecture/differences-with-ethereum.html#create-create2

● 部署环境

系统环境：ubuntu 22.04

1、安装 zkSync CLI：

```
yarn add global zksync-cli@latest
```

2、通过运行以下命令搭建一个新项目

```
zksync-cli create-project greeter-example
```

这将创建一个新的 zkSync Era 项目，该项目 greeter-example 包含基本 Greeter 合约以及所有 zkSync 插件和配置。

3、进入项目目录

```
cd greeter-example
```

4、添加秘钥：

该项目使用该 dotenv 包来加载部署智能合约并与智能合约交互所需的私钥。该.env 文件包含在内.gitignore，因此不会上传到存储库。要配置您的私钥，请复制.env.example 文件，将副本重命名为.env，然后添加您的钱包私钥。

```
WALLET_PRIVATE_KEY=abcdef12345....
```

5、编译并部署 Greeter 合约

我们将所有智能合约的*.sol 文件存储在该 contracts 文件夹中。该 deploy 文件夹包含与部署相关的所有脚本。

1）包含的 contracts/Greeter.sol 合同有以下代码：

```
//SPDX-License-Identifier: Unlicense
pragma solidity ^0.8.8;

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;
    }
}
```

2)使用以下命令编译合约：

```
yarn hardhat compile
```

3）zkSync -CLI 还提供了一个部署脚本/deploy/deploy-greeter.ts：

```
import { Wallet, utils } from "zksync-web3";
```

```
2.  import * as ethers from "ethers";
3.  import { HardhatRuntimeEnvironment } from "hardhat/types";
4.  import { Deployer } from "@matterlabs/hardhat-zksync-deploy";
5.
6.  // load env file
7.  import dotenv from "dotenv";
8.  dotenv.config();
9.
10. // load wallet private key from env file
11. const PRIVATE_KEY = process.env.WALLET_PRIVATE_KEY || "";
12.
13. if (!PRIVATE_KEY) throw "  Private key not detected! Add it to the .env file!";
14.
15. // An example of a deploy script that will deploy and call a simple contract.
16. export default async function (hre: HardhatRuntimeEnvironment) {
17.   console.log(`Running deploy script for the Greeter contract`);
18.
19.   // Initialize the wallet.
20.   const wallet = new Wallet(PRIVATE_KEY);
21.
22.   // Create deployer object and load the artifact of the contract you want to deploy.
23.   const deployer = new Deployer(hre, wallet);
24.   const artifact = await deployer.loadArtifact("Greeter");
25.
26.   // Estimate contract deployment fee
27.   const greeting = "Hi there!";
28.   const deploymentFee = await deployer.estimateDeployFee(artifact, [greeting]);
29.
30.   //   OPTIONAL: You can skip this block if your account already has funds in L2
31.   // Deposit funds to L2
32.   // const depositHandle = await deployer.zkWallet.deposit({
33.   //   to: deployer.zkWallet.address,
34.   //   token: utils.ETH_ADDRESS,
35.   //   amount: deploymentFee.mul(2),
36.   // });
37.   // // Wait until the deposit is processed on zkSync
38.   // await depositHandle.wait();
39.
40.   // Deploy this contract. The returned object will be of a `Contract` type, similarly to ones in
      `ethers`.
41.   // `greeting` is an argument for contract constructor.
42.   const parsedFee = ethers.utils.formatEther(deploymentFee.toString());
43.   console.log(`The deployment is estimated to cost ${parsedFee} ETH`);
44.
```

```
    const greeterContract = await deployer.deploy(artifact, [greeting]);

    //obtain the Constructor Arguments
    console.log("Constructor args:" + greeterContract.interface.encodeDeploy([greeting]));

    // Show the contract info.
    const contractAddress = greeterContract.address;
    console.log(`${artifact.contractName} was deployed to ${contractAddress}`);

    // verify contract for testnet & mainnet
    if (process.env.NODE_ENV != "test") {
      // Contract MUST be fully qualified name (e.g. path/sourceName:contractName)
      const contractFullyQualifedName = "contracts/Greeter.sol:Greeter";

      // Verify contract programmatically
      const verificationId = await hre.run("verify:verify", {
        address: contractAddress,
        contract: contractFullyQualifedName,
        constructorArguments: [greeting],
        bytecode: artifact.bytecode,
      });
    } else {
      console.log(`Contract not verified, deployed locally.`);
    }
  }
```

4）使用以下命令运行部署脚本：

```
yarn hardhat deploy-zksync --script deploy-greeter.ts
```

5）你应该看到这样的东西：

```
Running deploy script for the Greeter contract
The deployment is estimated to cost 0.0265726735 ETH
constructor args:0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000094869207468657265210000000000000000000000000000000000000000000000000000000000
Greeter was deployed to 0xE84774C41F096Ba5BafA1439cEE787D9dD1A6b72
Your verification ID is: 26642
Contract successfully verified on zkSync block explorer!
contracts/Greeter.sol:Greeter verified! VerificationId: 26642
```

恭喜！**您已将智能合约部署并验证到 zkSync Era 测试网**

现在访问 zkSync 区块浏览器并搜索合约地址以确认部署。

如果在部署的过程中遇到问题，可以联系@elonmusk1950m