ZkSync

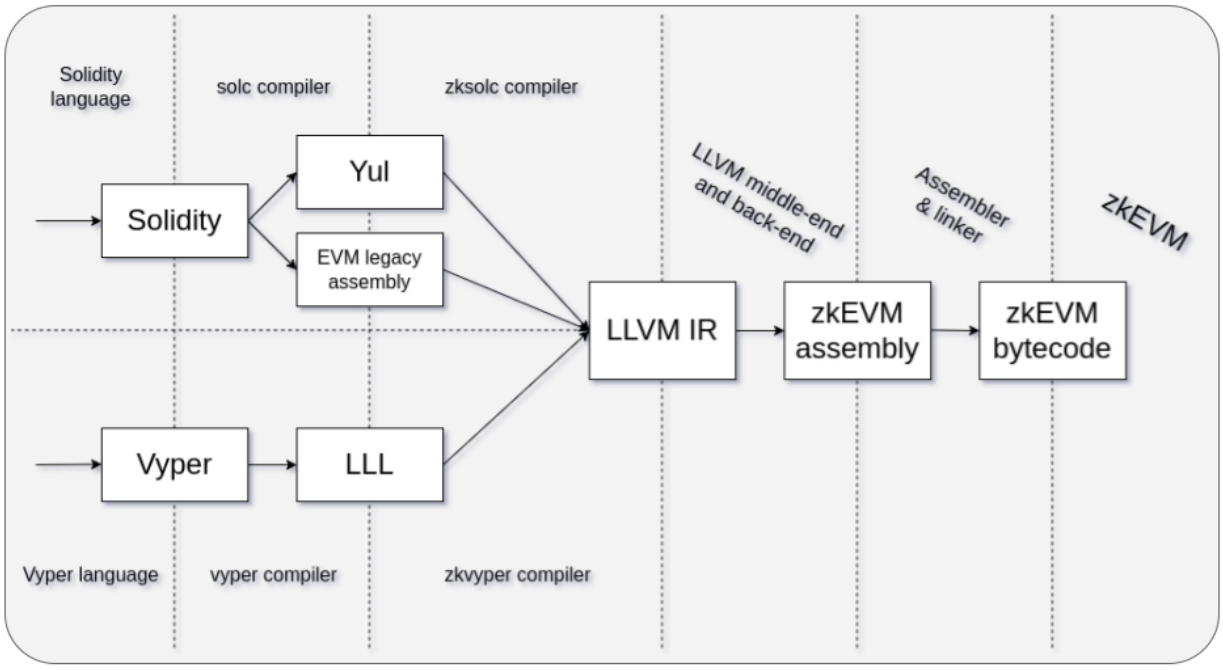
Created by	CJ
Created time	@July 18, 2023 4:16 PM
Status	New
Tags	

Introduction

ZkSync is a layer2 of Ethereum, ranked the #3 on **l2beat**. It's a ZK Rollup which provides volition support for users. I encourage everyone to learn ZkSync from **zkSync 2.0 for Builders** so that you can have an overview, after this video you can read the official docs to learn more about details.

Compilation

ZkSync uses a VM different than EVM, and was previously implementing its own contract language - Zinc but later decided to be **Solidity first.** The following is the flow of compilation:



As you can see from the graph, ZkSync needs two compilers:

- solc compiler
- zksolc compiler

The latter was introduced for converting **Yul** to **LLVM IR** and is the root cause of incompatibility with existing hardhat toolchains. ZkSync paid lots of effort on the hardhat toolchains but it's still not fully ready for production usage

Integration

To work with ZkSync we will be needing the following toolchains, all of these have been integrated seamlessly via PR#393

- solidity ≥ 0.8.17

- zksolc ≥ 1.3.9
- @matterlabs/hardhat-zksync-deploy
- @matterlabs/hardhat-zksync-solc
- zksync-web3

After adding this dependencies, we need to add the following configs to `hardhat.config.ts`

```
zksolc: {
  version: "1.3.9",
  compilerSource: "binary",
  settings: {
    libraries: ZK_LIBRARIES // Here it should contain all non-inlinable libraries' address
  }
},
networks: {
    ....
    zksync: {
      chainId: CHAINS_ID[eEthereumNetwork.zksync],
      url: NETWORKS_RPC_URL[eEthereumNetwork.zksync],
      accounts: DEPLOYER,
      ethNetwork: NETWORKS_RPC_URL[eEthereumNetwork.mainnet],
      zksync: true,
      verifyURL: ETHERSCAN_APIS[eEthereumNetwork.zksync],
    },
    zksyncGoerli: {
      chainId: CHAINS_ID[eEthereumNetwork.zksyncGoerli],
      url: NETWORKS_RPC_URL[eEthereumNetwork.zksyncGoerli],
      accounts: DEPLOYER,
      ethNetwork: NETWORKS_RPC_URL[eEthereumNetwork.goerli],
      zksync: true,
      verifyURL: ETHERSCAN_APIS[eEthereumNetwork.zksyncGoerli],
    },
    ....
}
```

then since ZkSync uses a different `Signer` type we will need to be compatible on this while getting the deployer:

```
export const getFirstSigner = async () => {
  if (DRE.network.zksync) {
    return new zk.Wallet(
      last(accounts)!.privateKey,
      new zk.Provider((DRE.network.config as HttpNetworkConfig).url), // L2
      new ethers.providers.JsonRpcProvider(
        (DRE.network.config as HttpNetworkConfig).ethNetwork // L1
      )
    );
  } else {
    if (!RPC_URL) {
      return first(await getEthersSigners())!;
    }

    const {paraSpaceAdminAddress} = await getParaSpaceAdmins();
    return (
      await impersonateAddress(IMPERSONATE_ADDRESS || paraSpaceAdminAddress)
    ).signer;
  }
};
```

finally because of ZkSync uses `Deployer` to loadArtifact and deploy, we also need to be compatible on this

```
export const getContractFactory = async (
  name: string,
  libraries?: Libraries
) => {
  const signer = await getFirstSigner();
  if (DRE.network.zksync) {
    const deployer = new Deployer(DRE, signer as zk.Wallet);
    const artifact = await deployer.loadArtifact(name);
    const factoryDeps = await deployer.extractFactoryDeps(artifact);
    return {
```

```
      artifact,
      factory: new zk.ContractFactory(
        artifact.abi,
        artifact.bytecode,
        signer as zk.Signer
      ),
      customData: {
        factoryDeps,
        feeToken: zk.utils.ETH_ADDRESS,
      },
    };
  } else {
    const artifact = await DRE.artifacts.readArtifact(name);
    if (libraries) {
      artifact.bytecode = linkLibraries(
        artifact,
        normalizeLibraryAddresses(libraries)
      );
    }
    return {
      artifact,
      factory: new ContractFactory(artifact.abi, artifact.bytecode, signer),
      customData: undefined,
    };
  }
};
```

Incompatibility

- SELFDESTRUCT op code is not supported
- payable is removed, `msg.value` by default equals to zero
- zksync bytecode is different with EVM bytecode (different VM)
- zksync bytecode contains only creationCode，no more runtimeCode
- zksync account now has two nonces: `deployment nonce` and `transaction nonce`
- zksync supports AA out of box and each address can be considered as contract address

▼ **!!deployment process!!**

1. create a contract by calling `ContractDeployer` system contract by providing bytecode hash and constructor arguments
2. bytecode pre-image should be provided via `custom_data.factory_deps` , if this contract creates other contracts, all contracts' bytecode should be put inside `custom_data.factory_deps`
3. bytecode hash must be **Known** on L1

▼ **!!compile time library linking!!**

1. `inlineable libraries` should be ok, no change needed
2. `non-inlineable libraries` ' address should be put inside `hardhat.config.ts` before compiling
3. if library relies on other libraries, we need to compile from the bottom to top so that each library can contain correct bytecode

- **!!no cache support for compilation!!**
- **compile time is very long**
- zksync rpc server is centralized and not fully compatible with ETH RPC

Getting Started

ZkSyncGoerli

1. install packages

```
yarn
```

2. touch .env

```
NETWORK=zksyncGoerli
ALCHEMY_KEY=v2H0jMbFK2BAlezbtzdYwdm9P_p38yOZ
ETHERSCAN_VERIFICATION=false
ETHERSCAN_VERIFICATION_CONTRACTS=*
MOCHA_JOBS=1
DB_PATH=deployed-contracts.json
DEPLOYER_MNEMONIC=...
```

3. build with no `zk-libraries.json`

```
make build
```

4. deploy libraries

```
make deploy-all-libraries
```

5. deploy

```
make deploy
```

Local Setup

1. clone https://github.com/matter-labs/local-setup

```
git clone https://github.com/matter-labs/local-setup
```

2. run local testnet

```
cd local-setup
./start.sh
```

3. open another terminal and go to paraspace-core, replace `hardhat.config.ts#221` by the following

```
ethNetwork: "http://127.0.0.1:8545",
```

4. modify .env

```
RPC_URL=http://127.0.0.1:3050
DEPLOYER_PRIVATE_KEY=0x7726827caac94a7f9e1b160f7ea819f172f7b6f9d2a97f992c38edeab82d4110
```

4. go back to normal process

Debugging

ZkSync's error message is a nightmare, it often looks like the following, `cast` will not work with zksync so we cannot simulate the transaction locally.

error.trace

I find it’s useful to debug via **debug_traceTransaction** rpc

```
curl -X POST  -H "Content-Type: application/json"  \
--data '{"jsonrpc":"2.0", "id":2, "method": "debug_traceTransaction", "params": ["0xe31f3c948f89066a131af16cdf7a0b459a0049ca6f1c4bb7bab285d
"https://zksync2-testnet.zksync.dev" > trace.json
```

then you can open `trace.json` and search the field with value non-null

```
{
  "type": "Call",
  "from": "0x0000000000000000000000000000000000008001",
  "to": "0x0000000000000000000000000000000000008004",
  "gas": "0xb0c9a0",
  "gasUsed": "0x37b",
  "value": "0x0",
  "output": "0x",
  "input": "0xe516761e0000000000000000000000000000000000000000000000000000000000000001000000000000000000000000000000000000000000000000000000
  "error": null,
  "revertReason": "Unknown revert reason",
  "calls": []
},
```

sometimes it tells you why, sometimes no, we can try again for a different trace on local-setup

error.trace2

```
"error": null,
      "from": "0x36615cf349d7f6344891b1e7ca7c72883f5dc049",
      "gas": "0xb59f3b",
      "gasUsed": "0x1081",
      "input": "0x13ad9cab000000000000000000000000512899a8dd96919721c5aacc2f01e9c6743d9337000000000000000000000000d0a217a9fcc538c971cff2a48
      "output": "0x",
      "revertReason": "Unknown revert reason",
      "to": "0x6f580854224a6ec4757ff9c99a8a4af3882f6e24",
      "type": "Call",
      "value": "0x0"
    }
  ],
  "error": null,
  "from": "0x0000000000000000000000000000000000008001",
  "gas": "0xb89008",
  "gasUsed": "0x1f9d",
  "input": "0xdf9c1589ab27573b667b629a20b24aa188941b729ef447b277cfb48cf7b0ced41bcf09c2ab116ca50abe068b3e6f3554b0e9b9f0d95bbd9eed17bbbc959e5
  "output": "0x",
  "revertReason": "Unknown revert reason",
  "to": "0x36615cf349d7f6344891b1e7ca7c72883f5dc049",
  "type": "Call",
  "value": "0x0"
},
```

OK still unknown revert reason, then **only god and ZkSync team knows why**

Bad Technical Support

26 June was the last day on which ZkSync team replied to us (ParaSpace), we asked for help multiple times via different methods:

- github issues

- telegram group

but ZkSync team doesn't really provide any helpful instruction, so we kept trying and it finally succeeded on local-setup

But, yes there is a but, it doesn't work with ZkSync goerli yet