ZK Stack Quick Guide

What is ZK Stack?

ZK Stack is a modular framework for building sovereign ZK-powered Ethereum rollups (called Hyperchains); it is a modular, open-source framework that is both free and designed to build custom Hyperchains (ZK-powered L2s and L3s), based on the code of zkSync Era, the first Hyperchain.

At its core, the ZK Stack offers two key features: **sovereignty and seamless connectivity**. The creator (you) possesses full rights to the code and enjoys unrestricted autonomy to customize and shape many aspects of your chain. Hyperchains operate independently, but are interconnected by a network of *Hyperbridges*, enabling trustless, fast (within minutes), and cheap (cost of a single transaction) interoperability.

What are the ZK Stack modules today?

▼ zkEVM

Our execution machine: EVM compatible, tailored for ZK and including enhancements like Native Account Abstraction.

▼ System Contracts

Important contracts with admin powers (could be compared to kernel modules), which define the behaviour of the system. The bytecodes of these contracts are set upon genesis and can be changed only by conducting upgrades. You can read more about system contracts here.

While the zkEVM does not know about the bytecodes of system contracts per se, it does rely on some of these for behaving correctly (for instance, it uses AccountCodeStorage system contract's storage to retrieve bytecodes of deployed zkEVM contracts, etc).

Repo: https://github.com/matter-labs/era-system-contracts

▼ Virtual machine

The rust-based virtual machine that is responsible for transaction validation and execution (including logic for rollbacks, history, etc).

Repo: https://github.com/matter-labs/era-zk_evm

▼ Smart contracts

All the smart contracts needed for the rollup, both on L1 and L2.

Repo: https://github.com/matter-labs/era-contracts

▼ Server (and its sub-components)

Repo: https://github.com/matter-labs/zksync-era

All the backend services needed to run a Hyperchain, including:

- **RPC services**

  The main interface for users to interact with the server.

  - *HttpApi* - Public Web3 API running on HTTP.
  - *WsApi* - Public Web3 API (including PubSub) running on WebSocket server

- **ETH Operator**

  This module is responsible for interacting with the base layer, both as an observer, as well as executing transactions.

  - *EthWatcher* - Monitor the base layer for specific events, such as Deposits or System Upgrades.
  - *EthTxAggregator* - Aggregates L1 batches and prepare a base layer tx, such as `commit_blocks` , `prove_blocks` , `execute_blocks`.
  - *EthTxManager* - Sign and send base layer txs, prepared by `EthTxAggregator`, also responsible for resending txs if they are stuck for any reason (example, low gas price).

- **Sequencer**

  Component that takes a list of incoming transactions, and packs them into blocks and batches - so that each one fits within the multiple constraints that our proving system has. It takes these 'batches' and executes them on the zkEVM.

  - *Tree and TreeBackup* - maintains a local RocksDB with the complete storage tree, computing the hash of the latest
  - *StateKeeper* - component that executes the transactions and saves sealed blocks to the DB

▼ Portal - Wallet + Bridge

A dApp that allows you (and your users) to easily interact with your Hyperchain, including bridging assets from and to the base layer, sending them within your Hyperchain, checking historical transactions, and managing contacts. You can also enhance its capabilities by running the Block Explorer Indexer/API.

Repo: https://github.com/matter-labs/dapp-portal

▼ Modules being released soon

▼ Prover

The set of components that allows zkEVM instructions (and the state changes they cause) to be proven with zero knowledge cryptography.

- **Circuits**

  Based on the virtual machine zkEVM, but with added functionality that re-executes transactions generating special traces that are used to recreate a given computation as a set of arithmetic circuits, which are then used for zero-knowledge proofs.

- **Witness generator**

  Component that prepares the needed information for the proving step.

- **Circuit Synthesizer**

  Component responsible for synthesizing the circuits that will be used during the proving step.

- **Boojum**

  Our STARK approach to proving zkEVM instructions. This module contains:

  - *GPU implementation* - Can be run on consumer-grade GPUs with as only as 16GB of RAM.
  - *CPU implementation* **-** Mostly for testing purposes, but an alternative to enable anyone to generate proofs in case GPUs are not available.

- **Verifier**

  Solidity smart contract that is responsible to verify that the state changes that were committed are valid given some proof.

- **Housekeeper**

  Used for queuing/retrying failed witness generations and prover jobs, and cleaning up blobs from cloud storages.

▼ Block explorer

A module to allow exposing everything that is happening on your Hyperchain to your users/developers. It comes with 4 components:

- *Block Explorer Worker*: an indexer service for Hyperchain data. The purpose of the service is to read the data from the blockchain in real-time, transform it and fill in its database to be used by the API.
- *Block Explorer API*: a service that provides an HTTPS API for retrieving structured Hyperchain data. It must be connected to the block explorer worker database.
- *Block Explorer app:* The UI that allows user/devs to explore transactions, blocks, batches, contracts, tokens, and much more, on your Hyperchain.
- *Contact Verifier*: the service that receives smart contract verification requests, validates them, and provides the code/ABIs for verified contracts.

▼ Fee withdrawer

A component that automates the withdraw of the Hyperchain collected fees to an address on the base layer. This helps ensuring the ETH operator on the base layer side has a constant influx of the gas token to keep working properly.

Future Modules

▼ Consensus node - PoA

An upgrade to the sequencer module that will allow anyone to run a decentralized network of sequencers. The sequencers will reach consensus using a proof-of-authority algorithm.

▼ Validium

Optional module where the data-availability is up to your choice. Only proofs are sent to the base layer, highly reducing overall costs, and enabling optional privacy for you Hyperchain. Initially this will only allow for sending data to any private server but later on there would be options for private DA layers.

▼ zkPorter

A sharded approach to scalability, where a set of accounts leverages off-chain data availability (a Validium), but still allows seamless interoperability with the regular rollup accounts. This is known as a Volition.

▼ Hyperbridge

The structure that allows fast messaging and asset bridging between hyperchains. It will start as a Shared Bridge on Ethereum so chains can register to it.

Getting Started with ZK Stack

ℹ️ This guide is a work in progress, and the initialization scripts will also go through relevant improvement throughout the next weeks.

⚠️ Requirements ⚠️

Make sure you have followed these instructions to set up dependencies on your machine (don't worry about the Environment part for now).

Deploying and running locally

1. Clone the repo and go to the root of it.

```
git clone https://github.com/matter-labs/zksync-era
```

2. Add ZKSYNC_HOME to your path (e.g. `~/.bash_profile`, `~/.zshrc`) - don't forget to source your profile again (or restart your terminal).

```
export ZKSYNC_HOME=/path/to/zksync/repo/you/cloned
export PATH=$ZKSYNC_HOME/bin:$PATH
```

3. Build latest version of zk tools by just running `zk` on the root of the project.
4. Lastly, start the wizard and follow instructions to set up and deploy your new Hyperchain by running `zk init-hyperchain`.
   a. Initially you want to `Configure new chain`.
   b. Give it a name and chain id.
   c. Select localhost (default matterlabs/geth) and follow the wizard.
      i. If you are doing this for the first time, several components needs to be compiled/built, so do not worry if it takes a few minutes. The console will show what is going on anyways.

Your Hyperchain is now deployed and configured. You can find all configuration in a new .env file created on /etc/env/<you_chain_name_slug>.env, and if you deployed test tokens, their addresses will be available at /etc/tokens/<the_l1_identifier>.json.

1. The wizard allows you to run a server in the end. If you chose not to, you can run it by executing `zk server --components "http_api,tree_lightweight,eth,data_fetcher,state_keeper,housekeeper "` .
2. You can now run transactions and start playing with your Hyperchain by using the RPC available at `http://localhost:3050`.
   a. Don't forget to deposit some ETH and fund your accounts on your Hyperchain. To do so follow these instructions.

Addendum

- With this default configuration, your Hyperchain is not running a prover, and has a DummyExecutor contract, which mainly “accepts” that a batch is executed without prove. This enables you to test it with much lower hardware requirements. The ability to run a Hyperchain with proving enable is still under development and will be available soon.
- If you make changes to any contract, you can always deploy a new Hyperchain to easily test those changes.
- If you configure your Hyperchain once, you don't need to do it again as the wizard allows you to use an existing config file.
- For now it is only possible to deploy a Hyperchain as an L2, but soon it will also work as L3s.
- When running the default matterlabs/geth, you have a set of rich wallets available to you. You can check them here.
- If you want to have a custom local L1, you must ensure you have a database for your Hyperchain, as well as the local RPC for your L1.
  - To run a Postgres 14 database for your Hyperchain, execute the following:

```
docker-compose up postgres -d
```

In case you don't want to use the docker Postgres database above but another one you already have locally, make sure its version is 14 and it is running and accepts connections at `postgres://postgres@localhost/zksync_local`. You can test with:

```
psql -h localhost -U postgres -d postgres -c 'SELECT 1;'
```

- If you face some issue compiling rust code (example `<jemalloc>: Error allocating TSD`), try removing the `rust-toolchain` file from the repo.

▼ Deploying to non-local environment

- **Database**

  Change the DATABASE_URL parameter in your Hyperchain env file to point to a remote instance, and make sure it accepts connections as described here.

- **Server**

  To run your server in a cloud environment, the first step is to build a docker image of it. To do so you should run:

  ```
  <Coming soon>
  ```

- **Prover**

  Not available at this moment

Using your Hyperchain

The first step to start interacting if to fund an account (or a few). This means you need some funds on the base layer:

▼ Base layer is the local geth node from @matterlabs/geth:latest

You will have a set of addresses that have 100 ETH each. You can find the list here: https://github.com/matter-labs/local-setup/blob/main/rich-wallets.json and use them to deposit into your Hyperchain.

▼ Base layer is an Ethereum network (eg, Goerli, Sepolia)

In this case you need to have an account on the base layer with ETH (which for sure you have given the deployer, governor and operator needed ETH go through the deployment process). You can use any of these wallets (or any other you have funds) to deposit into your Hyperchain.

Once you have the accounts with funds on the base layer, you can do a deposit (and any further interactions) in 3 ways:

▼ Using your Hyperchain RPC

Your server contains both a HTTPS as well as a WS services that are fully web3 compatible (and contains some extra ZK Stack functionalities). Know more about it here.

By default your server is available at http://localhost:3050 - but if you deployed the server into some cloud provider, you will have a different url to interact with.

▼ Using zksync-cli

When executing any command with zksync-cli, you can specify RPC urls for both L1 and L2 if you choose “localnet” as your network. An example deposit command would look like:

```
npx zksync-cli@latest deposit --l1-rpc-url=http://localhost:8545 --l2-rpc-url=http://localhost:3050 --zeek
```

▼ Using Portal

You can run the Portal module locally, and point it to your Hyperchain configuration. It comes with scripts that helps pulling the Hyperchain configuration from your zksync-era repo and adapting to portal needs. Know more here. An example command would look like:

```
npm run hyperchain:migrate ../zksync-era
npm run dev:node:hyperchain
```

Stuff