Development tools

Here we will cover tools that are used by developers to interact with the Vector network. These tools are used to build applications that interact with the Vector network. It includes servers, libraries, and SDKs that allow programmers to have a more efficient development process.

Blockfrost Backend RYO

Blockfrost is a backend service that provides a RESTful API to interact with the Vector network. It is a tool that allows developers to build applications that interact with the Cardano blockchain. The Blockfrost API provides access to a wide range of data, including blocks, transactions, addresses, and more.

You can either use Demeter's instance of Blockfrost or you can run it locally. We will cover how to run it locally.

It boots up a server that listens for API requests that will query the Vector DB Sync database and submit transactions to the Vector network.

It depends on the Vector DB Sync tool to get its data and Vector Ogmios to evaluate and submit transactions to the node.

Read vector db sync docs to know how to connect to the database and vector ogmios docs.

Once you have the database and ogmios running you can either run blockfrost from source or with docker. Here's the docker command:

```
docker run \
  -p 3000:3000 \
  -e BLOCKFROST_CONFIG_SERVER_LISTEN_ADDRESS=0.0.0.0 \
  -e BLOCKFROST_CONFIG_SERVER_PORT=3000 \
  -e BLOCKFROST_CONFIG_SERVER_DEBUG=true \
    -e BLOCKFROST_CONFIG_DBSYNC_HOST=demeter.run/... \
    -e BLOCKFROST_CONFIG_DBSYNC_USER=dmtr... \
    -e BLOCKFROST_CONFIG_DB_SYNC_PORT=5432 \
    -e BLOCKFROST_CONFIG_DB_SYNC_DATABASE=database \
    -e OGMIOS_HOST=demeter.run/...
    -e OGMIOS_HOST=1337 \
  -e NODE_ENV=development \
  -e NODE_APP_INSTANCE=dev \
    ghcr.io/apex-fusion/blockfrost-backend-ryo:v2.1.3
```

If you use the source code you can set the environment variables in a `.env` file and run:

```
yarn install
yarn dev
```

Here is an example of cool queries you can make to it, we use `httpie` to make request. You can use curl or any other tool you like.

- Addresses involved in transactions in a block

```
http localhost:3000/blocks/1549265/addresses
```

- Transaction details

```
http localhost:3000/txs/264d92abf63e84d5e606878f9daba415956054e1cdcec
```

- Transaction utxos

```
http localhost:3000/txs/a8afe24f5c244c0b8eb185cfc46c591652ee6b6928f25
```

Blockfrost JS SDK

Now that you have the Blockfrost backend running you can use the Blockfrost JS SDK to interact with it. This SDK is a JavaScript library that provides a simple and easy-to-use interface to interact with the Blockfrost API. It is open source and can be found here

You can install it in a javascript package with:

```
npm i @apexfusionfoundation/blockfrost-js
```

Here's an example of the same queries done with the SDK:

```
import { BlockFrostAPI } from "@apexfusionfoundation/blockfrost-js";

const cumstomBackend = "http://localhost:3000"
const bl = new BlockFrostAPI({ customBackend });

const addrs = await bl.blocksAddressesAll(1549265);
const tx = await bl.txs("264d92abf63e84d5e606878f9daba415956054e1cdce
const utxos = await bl.txsUtxos("a8afe24f5c244c0b8eb185cfc46c591652ee
```

```
console.dir({ addrs, tx, utxos }, { depth:null });
```

Notice that each of this functions returns a promise that resolves to the typed data you requested.

Vector JS SDK

Vector JS SDK is a Typescript library that provides a simple and easy-to-use interface to interact with the Vector network. It is open source and can be found here.

We only provide the core package of the SDK, you can install it in a javascript package with:

```
npm i @apexfusionfoundation/vector-sdk-core
```

It contains the core Typescript types and data structures that represent the Vector network. It is used by the Blockfrost JS SDK to interact with the Vector network.

Vector Serialization Library

Vector Serialization Library is a Typescript library that provides a simple and easy-to-use interface to serialize and deserialize data structures used by the Vector network. It is open source and can be found here.

We are currently only providing the node package of the library, you can install it in a javascript package with:

```
npm i @apexfusionfoundation/vector-serialization-lib-nodejs
```

It is automatically generated from rust code and provides a WASM module that can be used in the browser or in a Node.js environment.

Vector Blaze

Vector Blaze is an easy to use set of Typescript packages that rely on Vector Blockfrost Backend RYO in order for Blaze to interact with Vector.

Let's learn by example how to use Blaze to interact with the Vector network.

First you need to install some packages on an empty typescript project:

```
npm i \
    @apexfusionfoundation/vector-blaze-core \
    @apexfusionfoundation/vector-blaze-query \
    @apexfusionfoundation/vector-blaze-sdk
```

Let's start with the simplest example, pay 5 AP3X to your own address. For it you need to create a simple utility function that will initialize a blaze instance:

```
const provider = new Blockfrost({
  // don't worry about this, we will override it
  network: "cardano-mainnet",
  projectId,
});

// set the provider url to the blockfrost backend you are using
provider.url = "http://localhost:3000";

/**
  * This function initializes a blaze instance with a wallet
  */
const blazeWithSeed = async () => {
  const mnemonic =
    "i am a seed phrase that is used to generate a wallet";

  if (!mnemonic) {
    throw new Error("Missing seed phrase");
```

```
  }
  const entropy = mnemonicToEntropy(mnemonic, wordlist);
  const masterkey = Bip32PrivateKey.fromBip39Entropy(Buffer.from(entr
  const wallet = await HotWallet.fromMasterkey(
    masterkey.hex(),
    provider,
    2,
    AddressType.EnterpriseKey
  );

  return Blaze.from(provider, wallet);
};
```

Now, we are in a position to create a transaction that pays 5 AP3X to your own address:

```
const pay5ApexToSelf = async () => {
  console.log("PAYING 5 APEX TO MYSELF");
  const blaze = await blazeWithSeed();

  const tx = await blaze
    .newTransaction()
    .payLovelace(blaze.wallet.address, 5_000_000n)
    .complete();

  const signedTx = await blaze.signTransaction(tx);

  const txId = await blaze.submitTransaction(signedTx);
  console.log(txId);
};
```

You can check the transaction on the explorer by copying the transaction id and pasting it on the search bar.

You have successfully created a transaction that pays 5 AP3X to your own address.

We will showcase more usecases in the Basic examples section.

Privacy policy Terms of service