eth.events Documentation

Anyblock Analytics GmbH

Feb 06, 2019

Documentation Contents:

eth.events is a hosted Ethereum API providing sophisticated support to search, filter and aggregate events from the Ethereum blockchain, and it is based on Elasticsearch.

Constructing a query and understanding what an eth.events server returns is the most important skill you need in order to get started with eth.events.

If you are new to **Elasticsearch** and eth.events, you first should:

- Inform yourself about the *elastic-authorization* methods
- Read our *elastic developer tutorial*
- Familiarise yourself with the Elasticsearch Query DSL.
- Understand the elastic-schema and how events are *mapped*
- Leverage our *example queries* and build your own queries.

You might also take a look at our **SQL** interface:

- Check for the *sql-access* to our database
- Maybe start with looking at the *entity-relation model*
- To get a quickstart on how to use the interface you should see the *sql tutorial*

Please also visit our main Website or in case you have any further questions.

CHAPTER 1

Elastic

1.1 Authorization

To access eth.events, no matter if you just want to use the free service or have a paid service agreement, you'll need to get an API Key.

You can register for a **free** account and get an API key right now at https://account.eth.events/ in just a few seconds. Go ahead, I'll wait.

1.1.1 Sending your API key

Authorization Bearer Header

The fastest and most secure way to send your API key is via the *Authorization* header

```
curl -X GET https://api.eth.events/status/ -H 'Authorization: Bearer $mytoken'
```

Basic Auth

For compatibility reasons it's also possible to send the API key as your password via Basic Auth

```
curl -X GET https://api.eth.events/status/ -u '$myemail:$mytoken'
```

Please keep in mind that you're supposed to send your **API key**, not your password here.

Query Parameter

This is by far the least secure option, because your API key may end up in all sorts of logfiles and will even be visible to network sniffers. Only use the query parameter with short lived API keys for demonstration purposes.

```
curl -X GET 'https://api.eth.events/status/?access_token=$mytoken'
```

1.1.2 API key security

At the time of this writing there are two mechanisms in place to secure API access to your account.

Lifetime

When creating a new API key you can select a future date at which the key will no longer work. For most applications it's obviously not reasonable to switch the API key every few days, so you can create a long lived key and keep it a secret, while you may use a key with only a few days of validity for demos and other public use cases.

Domain

In case you want to use your API key in an environment where it's necessary to expose it to your intended audience (say, a website), you can additionally secure the API key with an allowed domain name. The key will then only work, if the referrer of the request matches the configured domain.

1.1.3 Why bother?

In the end this is a free service and anyone can get an API key anyways, so why all the hassle?

Well, for one we're going to use the API keys to restrict access to certain semi-private data, but mostly it's to control abuse to some extent.

We're monitoring the amount of requests and used data per account and in case your usage is **way** above *normal usage* we'll warn you and eventually disable your account.

In this case it would be a shame if some stranger on the internet just hijacked your API key.

1.2 Index endpoints

When interacting with the eth.events API, the base URL structure is always as follows:

/$technology/$blockchain/$network/$interface/...

The first triple of *technology*, *blockchain* and *network* describes the network you want to interact with. For most users this would be **/ethereum/ethereum/mainnet/**, which is also the default and can be omitted. Other possible values would be for example */ethereum/classic/morden/*.

The fourth part is the search interface you want to use. Currently only the ElasticSearch interface **/es/** is supported, but we're already working on SQL and GraphQL support.

So, for most users the base URL will be

```
https://api.eth.events/ethereum/ethereum/mainnet/es/
```

1.2.1 ElasticSearch /es/

Following the search interface, you can select the resource you want to query. Possible values are

- *block*
- *tx*
- *log*
- *event*
- *call*

which already reflects the type parameter of the ElasticSearch query syntax.

For obvious reasons we're limiting the full scope of the ElasticSearch Query DSL, but the following APIs will work as expected:

- *search*
- *_search*
- *count*
- *_count*

It's also possible to explicitly provide the desired index in the URL which follows the format **$technology-$blockchain-$network-$resource**. This is completely optional but required for ElasticSearch compatibility and is normally derrived from the selected network and resource, but must match the network and resource if present.

A simple query for the latest block would look like this:

```
https://api.eth.events/es/block/search
```

ElasticSearch Clients

In order to use a default ElasticSearch client, you can provide the following parameters:

- **server**: *https://api.eth.events/es/*
- **index**: *ethereum-ethereum-mainnet-block*
- **type**: *block*
- **username**: *your email address*
- **password**: *your API key*

ElasticSearch Data Structure

The following chapters will document the available entities and explain it's property structure.

Block

Object schema

The block object inherits it's properties from the web3 API:

- *number*: *Number* - the block number.

- *hash*: *String* - hash of the block.
- *parentHash*: *String* - hash of the parent block.
- *nonce*: *String* - hash of the generated proof-of-work.
- *sha3Uncles*: *String* - SHA3 of the uncles data in the block.
- *logsBloom*: *String* - the bloom filter for the logs of the block.
- *transactionsRoot*: *String*- the root of the transaction trie of the block
- *stateRoot*: *String* - the root of the final state trie of the block.
- *miner*: *String* - the address of the beneficiary to whom the mining rewards were given.
- *difficulty*: *BigNumber* - integer of the difficulty for this block.
- *totalDifficulty*: *BigNumber* - integer of the total difficulty of the chain until this block.
- *extraData*: *String* - the "extra data" field of this block.
- *size*: *Number* - integer the size of this block in bytes.
- *gasLimit*: *Number* - the maximum gas allowed in this block.
- *gasUsed*: *Number* - the total used gas by all transactions in this block.
- *timestamp*: *Number* - the unix timestamp for when the block was collated.
- *transactions*: *Array* - Array of transaction hashes
- *uncles*: *Array* - Array of uncle hashes.

Mapping

For some fields, there are multiple encodings available, which are nested as properties on the field. More information on those data types can be found *here*.

The following is the output of the Elasticsearch mapping for the *Block* type:

```
{
        "mappings": {
                "block": {
                        "dynamic": "false",
                        "properties": {
                                "difficulty": {
                                        "properties": {
                                                "padded": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                },
                                                "raw": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                }
                                        }
                                },
                                "extraData": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                },
                                "gasLimit": {
```

(continues on next page)

(continued from previous page)

```
                "properties": {
                    "num": {
                        "type": "long"
                    },
                    "raw": {
                        "type": "keyword",
                        "ignore_above": 256
                    }
                }
            },
            "gasUsed": {
                "properties": {
                    "num": {
                        "type": "long"
                    },
                    "raw": {
                        "type": "keyword",
                        "ignore_above": 256
                    }
                }
            },
            "hash": {
                "type": "keyword",
                "ignore_above": 256
            },
            "logsBloom": {
                "type": "keyword",
                "ignore_above": 256
            },
            "miner": {
                "type": "text",
                "fields": {
                    "raw": {
                        "type": "keyword",
                        "ignore_above": 256
                    }
                }
            },
            "nonce": {
                "type": "keyword",
                "ignore_above": 256
            },
            "mixHash": {
                "type": "keyword",
                "ignore_above": 256
            },
            "number": {
                "properties": {
                    "num": {
                        "type": "long"
                    },
                    "raw": {
                        "type": "keyword",
                        "ignore_above": 256
                    }
                }
            },
```

(continues on next page)

(continued from previous page)

```
                "parentHash": {
                        "type": "keyword",
                        "ignore_above": 256
                },
                "receiptsRoot": {
                        "type": "keyword",
                        "ignore_above": 256
                },
                "sha3Uncles": {
                        "type": "keyword",
                        "ignore_above": 256
                },
                "size": {
                        "properties": {
                                "num": {
                                        "type": "long"
                                },
                                "raw": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                }
                        }
                },
                "stateRoot": {
                        "type": "keyword",
                        "ignore_above": 256
                },
                "timestamp": {
                        "type": "date",
                        "format": "epoch_second"
                },
                "totalDifficulty": {
                        "properties": {
                                "padded": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                },
                                "raw": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                }
                        }
                },
                "transactionsRoot": {
                        "type": "keyword",
                        "ignore_above": 256
                },
                "uncles": {
                        "type": "text",
                        "fields": {
                                "raw": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                }
                        }
                }
        }
```

(continues on next page)

(continued from previous page)

```
                }
        }
}
```

Transaction

Object schema

The *tx* object inherits it's properties from the transaction object, as specified in the web3 API:

- *from*: *String* - The address for the sending account. Uses the web3.eth.defaultAccount property, if not specified.
- *to*: *String* - (optional) The destination address of the message, left undefined for a contract-creation transaction.
- *value*: *Number|String|BigNumber* - (optional) The value transferred for the transaction in Wei, also the endowment if it's a contract-creation transaction.
- *gas*: *Number|String|BigNumber* - (optional) The amount of gas to use for the transaction (unused gas is refunded).
- *gasPrice*: *Number|String|BigNumber* - (optional) The price of gas for this transaction in wei, defaults to the mean network gas price.
- *data*: *String* - (optional) Either a byte string containing the associated data of the message, or in the case of a contract-creation transaction, the initialisation code.
- *nonce*: *Number* - (optional) Integer of a nonce. This allows to overwrite your own pending transactions that use the same nonce.

Mapping

For some fields, there are multiple encodings available, which are nested as properties on the field. More information on those data types can be found here.

The following is the output of the Elasticsearch mapping for the *Transaction* type:

```
{
        "mappings": {
                "tx": {
                        "dynamic": "false",
                        "properties": {
                                "blockHash": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                },
                                "blockNumber": {
                                        "properties": {
                                                "num": {
                                                        "type": "long"
                                                },
                                                "raw": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                }
                                        }
```

(continues on next page)

(continued from previous page)

```
                },
                "contractAddress": {
                        "type": "text",
                        "fields": {
                                "raw": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                }
                        }
                },
                "cumulativeGasUsed": {
                        "properties": {
                                "num": {
                                        "type": "long"
                                },
                                "raw": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                }
                        }
                },
                "from": {
                        "type": "text",
                        "fields": {
                                "raw": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                }
                        }
                },
                "gas": {
                        "properties": {
                                "num": {
                                        "type": "long"
                                },
                                "raw": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                }
                        }
                },
                "gasPrice": {
                        "properties": {
                                "num": {
                                        "type": "long"
                                },
                                "raw": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                }
                        }
                },
                "gasUsed": {
                        "properties": {
                                "num": {
                                        "type": "long"
                                },
```

(continues on next page)

(continued from previous page)

```
                                "raw": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                }
                        }
                },
                "hash": {
                        "type": "keyword",
                        "ignore_above": 256
                },
                "input": {
                        "type": "keyword",
                        "ignore_above": 256
                },
                "logsBloom": {
                        "type": "keyword",
                        "ignore_above": 256
                },
                "nonce": {
                        "properties": {
                                "num": {
                                        "type": "long"
                                },
                                "raw": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                }
                        }
                },
                "r": {
                        "type": "keyword",
                        "ignore_above": 256
                },
                "root": {
                        "type": "keyword",
                        "ignore_above": 256
                },
                "s": {
                        "type": "keyword",
                        "ignore_above": 256
                },
                "status": {
                        "type": "boolean"
                },
                "timestamp": {
                        "type": "date",
                        "format": "epoch_second"
                },
                "to": {
                        "type": "text",
                        "fields": {
                                "raw": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                }
                        }
                },
```

(continues on next page)

(continued from previous page)

```
                "transactionIndex": {
                    "properties": {
                        "num": {
                            "type": "long"
                        },
                        "raw": {
                            "type": "keyword",
                            "ignore_above": 256
                        }
                    }
                },
                "v": {
                    "type": "keyword",
                    "ignore_above": 256
                },
                "value": {
                    "properties": {
                        "num": {
                            "type": "long"
                        },
                        "eth": {
                            "type": "double"
                        },
                        "padded": {
                            "type": "keyword",
                            "ignore_above": 256
                        },
                        "raw": {
                            "type": "keyword",
                            "ignore_above": 256
                        }
                    }
                }
            }
        }
    }
}
```

Log

Object schema

The *log* object inherits it's properties from the web3 API:

- *logIndex*: *Number* - integer of the log index position in the block.
- *transactionIndex*: *Number* - integer of the transactions index position log was created from.
- *transactionHash*: *String*- hash of the transactions this log was created from.
- *blockHash*: *String* - hash of the block where this log was in. *null* when its pending.
- *blockNumber*: *Number* - the block number where this log was in. *null* when its pending.
- *address*: *String* - address from which this log originated.
- *data*: *String* - contains one or more non-indexed arguments of the log.

- *topics*: *Array of hex strings* - Array of indexed log arguments.

Mapping

For some fields, there are multiple encodings available, which are nested as properties on the field. More information on those data types can be found here.

Note: For example the *address* is stored using the format *raw (text stored as keyword)* described on the datatypes page. In that particular case, the checksum-case formatted address can be used as a term filter query using *address.raw* and for a case-insensitive query, use *address*.

The following is the output of the Elasticsearch mapping for the *Log* type:

```
{
        "mappings": {
                "log": {
                        "dynamic": "false",
                        "properties": {
                                "address": {
                                        "type": "text",
                                        "fields": {
                                                "raw": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                }
                                        }
                                },
                                "blockHash": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                },
                                "id": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                },
                                "blockNumber": {
                                        "properties": {
                                                "num": {
                                                        "type": "long"
                                                },
                                                "raw": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                }
                                        }
                                },
                                "data": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                },
                                "logIndex": {
                                        "properties": {
                                                "num": {
                                                        "type": "long"
                                                },
                                                "raw": {
                                                        "type": "keyword",
```

(continues on next page)

(continued from previous page)

```
                                                    "ignore_above": 256
                                            }
                                    }
                            },
                            "type": {
                                    "type": "keyword",
                                    "ignore_above": 256
                            },
                            "timestamp": {
                                    "type": "date",
                                    "format": "epoch_second"
                            },
                            "topics": {
                                    "type": "keyword",
                                    "ignore_above": 256
                            },
                            "transactionHash": {
                                    "type": "keyword",
                                    "ignore_above": 256
                            },
                            "transactionIndex": {
                                    "properties": {
                                            "num": {
                                                    "type": "long"
                                            },
                                            "raw": {
                                                    "type": "keyword",
                                                    "ignore_above": 256
                                            }
                                    }
                            }
                    }
            }
    }
}
```

Event

Object schema

- *address*: *String*- address from which this event originated.
- *args*: *Array* - Array of argument objects coming from that event.
- *blockHash*: *String* - hash of the block where this event was in.
- *blockNumber*: *Number* - the block number where this event was in.
- *logIndex*: *Number* - integer of the event index position in the block.
- *event*: *String* - The event name.
- *transactionIndex*: *Number* - integer of the transactions index position event was created from.
- *transactionHash*: *String*- hash of the transactions this event was created from.
- *probability*: *Float* - the truthness of this event. 1.0 is the best.

Event arguments

The event's arguments with it's corresponding values are located in an object representation in an array of arguments. This allows different events to have different types and numbers of arguments.

The argument object's structure:

- *name* - the argument's name in human readable form
- *pos* - the index of the argument's position in the event
- *value.hex*,'value.scaled', *value.num* - the value of the events argument in it's corresponding representation
- *value.type* - the type of the argument's value, can be any type as specified for Solidity

Mapping

For some fields, there are multiple encodings available, which are nested as properties on the field. More information on those data types can be found here.

Note: For example the *address* is stored using the format *raw (text stored as keyword)* described on the datatypes page. In that particular case, the checksum-case formatted address can be used as a term filter query using *address.raw* and for a case-insensitive query, use *address*.

The following is the output of the Elasticsearch mapping for the *Event* type:

```
{
        "mappings": {
                "event": {
                        "dynamic": "false",
                        "properties": {
                                "address": {
                                        "type": "text",
                                        "fields": {
                                                "raw": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                }
                                        }
                                },
                                "args": {
                                        "type": "nested",
                                        "properties": {
                                                "name": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                },
                                                "pos": {
                                                        "type": "long"
                                                },
                                                "value": {
                                                        "properties": {
                                                                "hex": {
                                                                        "type":
↪"keyword",
                                                                        "ignore_above
↪": 256
                                                                },
```

(continues on next page)

(continued from previous page)

```
                                                                "num": {
                                                                        "type": "long"
                                                                },
                                                                "scaled": {
                                                                        "type":
↪"double"
                                                                },
                                                                "type": {
                                                                        "type":
↪"keyword",
                                                                        "ignore_above
↪": 256
                                                                }
                                                        }
                                                }
                                        }
                                },
                                "probability": {
                                        "type": "double"
                                },
                                "blockHash": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                },
                                "blockNumber": {
                                        "properties": {
                                                "num": {
                                                        "type": "long"
                                                },
                                                "raw": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                }
                                        }
                                },
                                "logIndex": {
                                        "properties": {
                                                "num": {
                                                        "type": "long"
                                                },
                                                "raw": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                }
                                        }
                                },
                                "event": {
                                        "type": "text",
                                        "fields": {
                                                "raw": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                }
                                        }
                                },
                                "timestamp": {
                                        "type": "date",
```

(continues on next page)

(continued from previous page)

```
                                        "format": "epoch_second"
                                },
                                "transactionHash": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                },
                                "transactionIndex": {
                                        "properties": {
                                                "num": {
                                                        "type": "long"
                                                },
                                                "raw": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                }
                                        }
                                }
                        }
                }
        }
}
```

Call

Object schema

- *from*: *String* - The address for the sending account. Uses the web3.eth.defaultAccount property, if not specified.
- *to*: *String* - (optional) The destination address of the message, left undefined for a contract-creation transaction.
- *blockNumber*: *Number* - the block number.
- *hash*: *String* - hash of the transaction.
- *blockHash*: *String* - hash of the block.
- *method*: *String* - name of the method that was called during the transaction.
- *probability*: *Float* - the truthness of this call. 1.0 is the best.

Call arguments

The transaction call's arguments with it's corresponding input values are located in an object representation in an array of arguments. This allows different calls to have different types and numbers of arguments.

The argument object's structure:

- *name* - the argument's name in human readable form
- *pos* - the index of the argument's position in the call
- *value.hex*,'value.scaled', *value.num* - the value of the calls argument in it's corresponding representation
- *value.type* - the type of the argument's value, can be any type as specified for Solidity

Mapping

For some fields, there are multiple encodings available, which are nested as properties on the field. More information on those data types can be found here.

The following is the output of the Elasticsearch mapping for the *Call* type:

```
{
        "mappings": {
                "call": {
                        "dynamic": "false",
                        "properties": {
                                "args": {
                                        "type": "nested",
                                        "properties": {
                                                "name": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                },
                                                "pos": {
                                                        "type": "long"
                                                },
                                                "value": {
                                                        "properties": {
                                                                "hex": {
                                                                        "type":
↪"keyword",
                                                                        "ignore_above
↪": 256
                                                                },
                                                                "num": {
                                                                        "type": "long"
                                                                },
                                                                "scaled": {
                                                                        "type":
↪"double"
                                                                },
                                                                "type": {
                                                                        "type":
↪"keyword",
                                                                        "ignore_above
↪": 256
                                                                }
                                                        }
                                                }
                                        }
                                },
                                "blockHash": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                },
                                "blockNumber": {
                                        "properties": {
                                                "num": {
                                                        "type": "long"
                                                },
                                                "raw": {
                                                        "type": "keyword",
```

(continues on next page)

(continued from previous page)

```
                                                "ignore_above": 256
                                        }
                                }
                        },
                        "from": {
                                "type": "text",
                                "fields": {
                                        "raw": {
                                                "type": "keyword",
                                                "ignore_above": 256
                                        }
                                }
                        },
                        "to": {
                                "type": "text",
                                "fields": {
                                        "raw": {
                                                "type": "keyword",
                                                "ignore_above": 256
                                        }
                                }
                        },
                        "transactionIndex": {
                                "properties": {
                                        "num": {
                                                "type": "long"
                                        },
                                        "raw": {
                                                "type": "keyword",
                                                "ignore_above": 256
                                        }
                                }
                        },
                        "probability": {
                                "type": "double"
                        },
                        "timestamp": {
                                "type": "date",
                                "format": "epoch_second"
                        },
                        "hash": {
                                "type": "keyword",
                                "ignore_above": 256
                        },
                        "method": {
                                "type": "text",
                                "fields": {
                                        "raw": {
                                                "type": "keyword",
                                                "ignore_above": 256
                                        }
                                }
                        }
                }
        }
}
```

Contract

Object schema

- *address*: *String*- address of the deployed contract.
- *abi*: *String* - the application binary interface of the deployed contract, formatted in JSON.
- *probability*: *Float* - the truthness of this contract information. 1.0 is the best.

Mapping

For some fields, there are multiple encodings available, which are nested as properties on the field. More information on those data types can be found here.

The following is the output of the Elasticsearch mapping for the *Contract* type:

```
{
        "mappings": {
                "contract": {
                        "dynamic": "false",
                        "properties": {
                                "address": {
                                        "type": "text",
                                        "fields": {
                                                "raw": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                }
                                        }
                                },
                                "abi": {
                                        "type": "binary"
                                },
                                "name": {
                                        "type": "text",
                                        "fields": {
                                                "raw": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                }
                                        }
                                },
                                "runs": {
                                        "properties": {
                                                "num": {
                                                        "type": "long"
                                                },
                                                "raw": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                }
                                        }
                                },
                                "bytecode": {
                                        "type": "binary"
```

(continues on next page)

(continued from previous page)

```
                                },
                                "source": {
                                        "type": "binary"
                                },
                                "compiler": {
                                        "type": "keyword"
                                },
                                "library": {
                                        "type": "keyword"
                                },
                                "createdAt": {
                                        "type": "date",
                                        "format": "epoch_second"
                                },
                                "optimizations": {
                                        "type": "boolean"
                                },
                                "probability": {
                                        "type": "double"
                                },
                                "links": {
                                        "type": "nested",
                                        "properties": {
                                                "description": {
                                                        "type": "text",
                                                        "fields": {
                                                                "raw": {
                                                                        "type":
↪"keyword",
                                                                        "ignore_above
↪": 256
                                                                }
                                                        }
                                                },
                                                "link": {
                                                        "type": "keyword"
                                                }
                                        }
                                },
                                "constructorArgs": {
                                        "type": "nested",
                                        "properties": {
                                                "name": {
                                                        "type": "keyword",
                                                        "ignore_above": 256
                                                },
                                                "pos": {
                                                        "type": "long"
                                                },
                                                "value": {
                                                        "properties": {
                                                                "hex": {
                                                                        "type":
↪"keyword",
                                                                        "ignore_above
↪": 256
                                                                },
```

(continues on next page)

(continued from previous page)

```
                                                                    "num": {
                                                                            "type": "long"
                                                                    },
                                                                    "scaled": {
                                                                            "type":
↪"double"
                                                                    },
                                                                    "type": {
                                                                            "type":
↪"keyword",
                                                                            "ignore_above
↪": 256
                                                                    }
                                                            }
                                                    }
                                            }
                                    }
                            }
                    }
            }
}
```

Token

Object schema

- *address*: *String*- address of the deployed contract.
- *name*: *String* - a human readable name for the contract.
- *symbol*: *String* - the currency symbol of the contract.
- *totalSupply*: *Number* - the total supply of shared value for the currency.
- *links*: *Array* - an array of weblinks with description and link.
- *probability*: *Float* - the truthness of this token information. 1.0 is the best.

Mapping

For some fields, there are multiple encodings available, which are nested as properties on the field. More information on those data types can be found here.

The following is the output of the Elasticsearch mapping for the *Token* type:

```
{
        "mappings": {
                "token": {
                        "dynamic": "false",
                        "properties": {
                                "address": {
                                        "type": "text",
                                        "fields": {
                                                "raw": {
                                                        "type": "keyword",
```

(continues on next page)

(continued from previous page)

```
                                        "ignore_above": 256
                                }
                        }
                },
                "symbol": {
                        "type": "keyword",
                        "ignore_above": 256
                },
                "name": {
                        "type": "text",
                        "fields": {
                                "raw": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                }
                        }
                },
                "type": {
                        "type": "keyword",
                        "ignore_above": 256
                },
                "decimals": {
                        "type": "long"
                },
                "totalSupply": {
                        "properties": {
                                "num": {
                                        "type": "long"
                                },
                                "raw": {
                                        "type": "keyword",
                                        "ignore_above": 256
                                }
                        }
                },
                "links": {
                        "type": "nested",
                        "properties": {
                                "description": {
                                        "type": "text",
                                        "fields": {
                                                "raw": {
                                                        "type":
↪"keyword",
                                                        "ignore_above
↪": 256
                                                }
                                        }
                                },
                                "link": {
                                        "type": "keyword"
                                }
                        }
                },
                "probability": {
                        "type": "double"
                }
```

(continues on next page)

(continued from previous page)

```
                                    }
                                }
                            }
}
```

1.3 Data types

1.3.1 Elasticsearch types

All the types from the blockchain as well as their encodings have to be represented with a Elasticsearch type for indexing and searching. Most blockchain types that are no number types are represented as a string in Elasticsearch (*keyword* or *text* type).

To learn more about the types in Elasticsearch, visit their documentation.

1.3.2 Eth.events encoded types

This are the types that represent a value on the blockchain. For some values, there are alternative encodings available.

raw

The raw value as read by the ethereum node used for indexing. This corresponds to the normal type as defined in the *web3 API*. Implemented as a *keyword* type in Elasticsearch.

scaled

A double floating point number that directly represents the ether value. Due to rounding this is not as accurate as using the *raw* or *padded* value directly. Implemented as a *double* type in Elasticsearch.

padded

A hexadecimal data type, where the hex string is padded to the biggest possible value. This allows e.g. for string sorting of big integer fields Implemented as a *keyword* type in Elasticsearch.

num

An integer representation of a value. *num* values are only accessible for integers that fit within the *long* type of Elasticsearch (64bit). BigInteger blockchain types (e.g. *uint256*) are only accessible from the *hex* representation Implemented as a *long* type in Elasticsearch.

hex

A hex string, representing the hex encoding of a value. This also includes hex encodings of BigInteger blockchain types (e.g. *uint256*), since they are not accessible from a *num* type. Implemented as a *keyword* type in Elasticsearch.

raw (text stored as keyword)

An explicit property that makes the *keyword* Elasticsearch type of a string accessible, when the default value is of type *text*. This is useful for allowing literal searches instead of pattern matching text searches. Implemented as a *keyword* type in Elasticsearch.

1.4 Example queries

The following queries are meant to be a building block for your own eth.events queries.

You also will need a registered eth.events API-Account in order to run the REST calls. Keep in mind to replace the `$mytoken` variable in the `cURL` commands with your personal API token.

1.4.1 Block

Get block by blockhash

The results will contain only the block with the given block *number*. This requires no body.

```
GET /ethereum/ethereum/mainnet/es/block/
↪0xf44f60a66257d1c6c8afd2a64aaeb306d9c471d5d38b6dc277811455192ecee1/
```

Select by block number

The results will contain only number selected with the given term.

HTTP-Method/Endpoint:

```
POST /ethereum/ethereum/mainnet/es/block/search/
```

JSON body:

```
{
  "query": {
    "bool": {
      "filter": {
        "term": {
          "number.num": 6600000
        }
      }
    }
  },
  "_source": ["number.num"]
}
```

Execute the request with cURL:

```
curl -X POST \
https://api.eth.events/ethereum/ethereum/mainnet/es/block/search/ \
-H 'Authorization: Bearer $mytoken' \
-H 'Content-Type: application/json' \
-d '{
  "query": {
    "bool": {
      "filter": {
        "term": {
          "number.num": 6600000
        }
      }
    }
  },
  "_source": ["number.num"]
}'
```

Filter empty blocks

The results will contain only blocks that are empty (include no transactions).

HTTP-Method/Endpoint:

```
POST /ethereum/ethereum/mainnet/es/block/search/
```

JSON body:

```
{
  "query": {
    "bool": {
      "must_not": {
        "exists": {
```

(continues on next page)

(continued from previous page)

```
          "field": "transactions"
        }
      }
    }
  }
}
```

Execute the request with cURL:

```
curl -X POST \
https://api.eth.events/ethereum/ethereum/mainnet/es/block/search/ \
-H 'Authorization: Bearer $mytoken' \
-H 'Content-Type: application/json' \
-d '{
  "query": {
    "bool": {
      "must_not": {
        "exists": {
          "field": "transactions"
        }
      }
    }
  }
}'
```

Filter last 5 known blocks (sorted)

The results will only contain the 5 most recent blocks (highest block number) on the index. Sorted in descending order (highest block number first).

HTTP-Method/Endpoint:

```
POST /ethereum/ethereum/mainnet/es/block/search/
```

JSON body:

```
{
  "sort": {
    "number.num": "desc"
  },
  "size": 5
}
```

Execute the request with cURL:

```
curl -X POST \
https://api.eth.events/ethereum/ethereum/mainnet/es/block/search/ \
-H 'Authorization: Bearer $mytoken' \
-H 'Content-Type: application/json' \
-d '{
  "sort": {
    "number.num": "desc"
  },
  "size": 5
}'
```

1.4.2 Transaction

Filter by the transaction's block's hash

The results will contain all transactions that are included in the specified block, identified with it's *blockHash*.

HTTP-Method/Endpoint:

```
POST /ethereum/ethereum/mainnet/es/tx/search/
```

JSON body:

```
{
  "query": {
    "bool": {
      "filter": [
        {
          "term": {
            "blockHash":
↪"0x4e3a3754410177e6937ef1f84bba68ea139e8d1a2258c5f85db9f1cd715a1bdd"
          }
        }
      ]
    }
  },
  "size": 200
}
```

Execute the request with `cURL`:

```
curl -X POST \
https://api.eth.events/ethereum/ethereum/mainnet/es/tx/search/ \
-H 'Authorization: Bearer $mytoken' \
-H 'Content-Type: application/json' \
-d '{
  "query": {
    "bool": {
      "filter": [
        {
          "term": {
            "blockHash":
↪"0x4e3a3754410177e6937ef1f84bba68ea139e8d1a2258c5f85db9f1cd715a1bdd"
          }
        }
      ]
    }
  },
  "size": 200
}'
```

Filter by a range of block numbers

The results will contain all transactions, that are included in a block, that is within the specified boundaries of the block number range. The block number has to be greater than or equal to 6400000 (*gte*) and less than or equal to 6500000 (*lte*). The results will show a maximum of 200 blocks, in no particular order.

HTTP-Method/Endpoint:

```
POST /ethereum/ethereum/mainnet/es/tx/search/
```

JSON body:

```
{
  "query": {
    "bool": {
      "filter": [
        {
          "range": {
            "blockNumber.num": {
              "gte": 6400000,
              "lte": 6500000
            }
          }
        }
      ]
    }
  },
  "size": 200
}
```

Execute the request with cURL:

```
curl -X POST \
https://api.eth.events/ethereum/ethereum/mainnet/es/tx/search/ \
-H 'Authorization: Bearer $mytoken' \
-H 'Content-Type: application/json' \
-d '{
  "query": {
    "bool": {
      "filter": [
        {
          "range": {
            "blockNumber.num": {
              "gte": 6400000,
              "lte": 6500000
            }
          }
        }
      ]
    }
  },
  "size": 200
}'
```

Filter by receiving or originating address

The results will contain all transactions, whose sender (*from*) or receiver (*to*) is has the specified address.

HTTP-Method/Endpoint:

```
POST /ethereum/ethereum/mainnet/es/tx/search/
```

JSON body:

```
{
  "query": {
    "bool": {
      "should": [
        {
          "term": {
            "from": "0xa1e4380a3b1f749673e270229993ee55f35663b4"
          }
        },
        {
          "term": {
            "to": "0xa1e4380a3b1f749673e270229993ee55f35663b4"
          }
        }
      ]
    }
  }
}
```

Execute the request with `cURL`:

```
curl -X POST \
  https://api.eth.events/ethereum/ethereum/mainnet/es/tx/search/ \
  -H 'Authorization: Bearer $mytoken' \
  -H 'Content-Type: application/json' \
  -d '{
    "query": {
      "bool": {
        "should": [
          {
            "term": {
              "from": "0xa1e4380a3b1f749673e270229993ee55f35663b4"
            }
          },
          {
            "term": {
              "to": "0xa1e4380a3b1f749673e270229993ee55f35663b4"
            }
          }
        ]
      }
    }
  }'
```

Select by transaction hash

The results will contain only the transaction with the given transaction *hash*.

HTTP-Method/Endpoint:

```
POST /ethereum/ethereum/mainnet/es/tx/search/
```

JSON body:

```
{
  "query": {
```

(continues on next page)

(continued from previous page)

```
    "bool": {
      "filter": [
        {
          "term": {
            "_id": "0x5c504ed432cb51138bcf09aa5e8a410dd4a1e204ef84bfed1be16dfba1b22060
↪"
          }
        }
      ]
    }
  }
}
```

Execute the request with `cURL`:

```
curl -X POST \
  https://api.eth.events/ethereum/ethereum/mainnet/es/tx/search/ \
  -H 'Authorization: Bearer $mytoken' \
  -H 'Content-Type: application/json' \
  -d '  {
    "query": {
      "bool": {
        "filter": [
          {
            "term": {
              "_id":
↪"0x5c504ed432cb51138bcf09aa5e8a410dd4a1e204ef84bfed1be16dfba1b22060"
            }
          }
        ]
      }
    }
  }'
```

1.4.3 Log

Filter by causing transaction's sender

The results will contain all logs, where the sender of the transaction that caused the log to be emitted has the specified address.

HTTP-Method/Endpoint:

```
POST /ethereum/ethereum/mainnet/es/log/search/
```

JSON body:

```
{
  "query": {
    "bool": {
      "filter": [
        {
          "term": {
            "transactionHash":
↪"0xca9b47a8bfd1c8c0e184992e0a2714558603182fc4a7f2ac16cf16f6be4f0a2a"
```

(continues on next page)

(continued from previous page)

```
          }
        }
      ]
    }
  }
}
```

Execute the request with cURL:

```
curl -X POST \
  https://api.eth.events/ethereum/ethereum/mainnet/es/log/search/ \
  -H 'Authorization: Bearer $mytoken' \
  -H 'Content-Type: application/json' \
  -d '{
      "query": {
        "bool": {
          "filter": [
            {
              "term": {
                "transactionHash":
↪"0xca9b47a8bfd1c8c0e184992e0a2714558603182fc4a7f2ac16cf16f6be4f0a2a"
              }
            }
          ]
        }
      }
    }'
```

Filter by emitting contract

The results will contain all logs that were emitted from the specified contract.

HTTP-Method/Endpoint:

```
POST /ethereum/ethereum/mainnet/es/log/search/
```

JSON body:

```
{
  "query": {
    "bool": {
      "filter": [
        {
          "term": {
            "address": "0x12459c951127e0c374ff9105dda097662a027093"
          }
        }
      ]
    }
  },
  "size": 100
}
```

Execute the request with cURL:

```
curl -X POST \
  https://api.eth.events/ethereum/ethereum/mainnet/es/log/search/ \
  -H 'Authorization: Bearer $mytoken' \
  -H 'Content-Type: application/json' \
  -d '{
    "query": {
      "bool": {
        "filter": [
          {
            "term": {
              "address": "0x12459c951127e0c374ff9105dda097662a027093"
            }
          }
        ]
      }
    },
    "size": 100
  }'
```

1.4.4 Event

Filter by event name

The results will contain all events with the specified event name.

HTTP-Method/Endpoint:

```
POST /ethereum/ethereum/mainnet/es/event/search/
```

JSON body:

```
{
  "query": {
    "bool": {
      "filter": [
        {
          "term": {
            "event": "Transfer"
          }
        }
      ]
    }
  }
}
```

Execute the request with `cURL`:

```
curl -X POST \
  https://api.eth.events/ethereum/ethereum/mainnet/es/event/search/ \
  -H 'Authorization: Bearer $mytoken' \
  -H 'Content-Type: application/json' \
  -d '{
    "query": {
      "bool": {
        "filter": [
          {
```

(continues on next page)

(continued from previous page)

```
            "term": {
              "event": "Transfer"
            }
          }
        ]
      }
    }
  }'
```

Filter by emitting contract

The results will contain all events that where emitted by the specified contract.

HTTP-Method/Endpoint:

```
POST /ethereum/ethereum/mainnet/es/event/search/
```

JSON body:

```
{
  "query": {
    "bool": {
      "filter": [
        {
          "term": {
            "address": "0xcfb98637bcae43C13323EAa1731cED2B716962fD"
          }
        }
      ]
    }
  }
}
```

Execute the request with cURL:

```
curl -X POST \
  https://api.eth.events/ethereum/ethereum/mainnet/es/event/search/ \
  -H 'Authorization: Bearer $mytoken' \
  -H 'Content-Type: application/json' \
  -d '{
    "query": {
      "bool": {
        "filter": [
          {
            "term": {
              "address": "0xcfb98637bcae43C13323EAa1731cED2B716962fD"
            }
          }
        ]
      }
    }
  }'
```

Filter by ERC20 contract's address and *from* address

The results will contain all events that where emitted by the specified contract, and where the *from* argument of the event matches the specifid address. Although this query is tailored for ERC20 contracts, there is no parameter that specifically filters for the ERC20 interface.

HTTP-Method/Endpoint:

```
POST /ethereum/ethereum/mainnet/es/event/search/
```

JSON body:

```
{
  "query": {
    "bool": {
      "filter": [
        {
          "term": {
            "address": "0xcfb98637bcae43C13323EAa1731cED2B716962fD"
          }
        },
        {
          "nested": {
            "path": "args",
            "query": {
              "bool": {
                "filter": [
                  {
                    "term": {
                      "args.name": "_from"
                    }
                  },
                  {
                    "term": {
                      "args.value.hex": "0x8d7a4f88e494de0ca71c4b1b469613ec9d12686c"
                    }
                  }
                ]
              }
            }
          }
        }
      ]
    }
  }
}
```

Execute the request with cURL:

```
curl -X POST \
  https://api.eth.events/ethereum/ethereum/mainnet/es/event/search/ \
  -H 'Authorization: Bearer $mytoken' \
  -H 'Content-Type: application/json' \
  -d '{
    "query": {
      "bool": {
        "filter": [
          {
```

(continues on next page)

(continued from previous page)

```
            "term": {
              "address": "0xcfb98637bcae43C13323EAa1731cED2B716962fD"
            }
          },
          {
            "nested": {
              "path": "args",
              "query": {
                "bool": {
                  "filter": [
                    {
                      "term": {
                        "args.name": "_from"
                      }
                    },
                    {
                      "term": {
                        "args.value.hex": "0x8d7a4f88e494de0ca71c4b1b469613ec9d12686c"
                      }
                    }
                  ]
                }
              }
            }
          }
        ]
      }
    }
  }'
```

1.4.5 Specialised queries

Find entity by hash

The results will contain either a block with the specified block hash or all transactions, whose sender (*from*) or receiver (*to*) has the specified address. Queries like this are useful if the type of the entity that a hash represents is not known in advance.

HTTP-Method/Endpoint:

```
POST /ethereum/ethereum/mainnet/es/block,tx/search/
```

JSON body:

```
{
  "query": {
    "bool": {
      "should": [
        {
          "ids": {
            "values": [
              "0x4e3a3754410177e6937ef1f84bba68ea139e8d1a2258c5f85db9f1cd715a1bdd"
            ]
          }
        },
```

(continues on next page)

(continued from previous page)

```
        {
          "term": {
            "from":
↪"0x4e3a3754410177e6937ef1f84bba68ea139e8d1a2258c5f85db9f1cd715a1bdd"
          }
        },
        {
          "term": {
            "to": "0x4e3a3754410177e6937ef1f84bba68ea139e8d1a2258c5f85db9f1cd715a1bdd"
          }
        }
      ]
    }
  }
}
```

Execute the request with `cURL`:

```
curl -X POST \
  https://api.eth.events/ethereum/ethereum/mainnet/es/block,tx/search/ \
  -H 'Authorization: Bearer d2560f14-1935-44e7-ad3e-a1718dc03bd2' \
  -H 'Content-Type: application/json' \
  -d '{
    "query": {
      "bool": {
        "should": [
          {
            "ids": {
              "values": [
                "0x4e3a3754410177e6937ef1f84bba68ea139e8d1a2258c5f85db9f1cd715a1bdd"
              ]
            }
          },
          {
            "term": {
              "from":
↪"0x4e3a3754410177e6937ef1f84bba68ea139e8d1a2258c5f85db9f1cd715a1bdd"
            }
          },
          {
            "term": {
              "to":
↪"0x4e3a3754410177e6937ef1f84bba68ea139e8d1a2258c5f85db9f1cd715a1bdd"
            }
          }
        ]
      }
    }
  }'
```

1.5 Tutorials

To get started with eth.events queries, you should have a look at our tutorials. They are a good starting point for getting used to the Elasticsearch syntax and the schemes and structuring of the query-results.

1.5.1 Simple contract event query

Welcome!

Within this tutorial you will learn how to retrieve and analyze data from the Ethereum Blockchain with the help of the eth.events API.

We will show you how to retrieve data from eth.events using the ElasticSearch Query DSL.

You will learn

- How to access eth.events
- How to use different methods to query the eth.events API
- How to write a basic query returning some events
- How a return object is structured and which data it returns
- How to filter events for a specific contract or a specific event type
- How to sort the events by blocknumber

What you must know already

This tutorial is written for programmers, who have some experience with JSON, Rest-APIs and the basic structure of HTTP-requests.

What you need

If you want to play around with the HTTP-requests, you should install an HTTP client. The good ol' terminal users might use `cURL`. For advanced usage and a graphical UI we recommend using Postman. We provide copy-pasteable commands for `cURL` throughout the tutorial, so if you want to follow along, it is advisable to install the software first.

You also will need a registered eth.events API-Account in order to run the REST calls. The token of your account will be used in the following tutorial as `$token`. Please replace the variables with your user and password.

Create an eth.events query step-by-step

Retrieve all events indexed by eth.events

On the eth.events endpoint `/ethereum/ethereum/mainnet/es/event/search`, you are able to query all events from the Ethereum mainnet.

A simple GET request to `/ethereum/ethereum/mainnet/es/event/search` shows us 10 events in no particular order.

Execute the request with `cURL`:

```
curl -X POST \
  https://api.eth.events/ethereum/ethereum/mainnet/es/event/search/ \
  -H 'Authorization: Bearer $mytoken' \
  -H 'Content-Type: application/json'
```

The returned JSON starts with meta-information about the processing of the query (not shown here).

The query results are shown under the `"hits"` keyword in the retrieved JSON data-structure:

```
"hits":{
  "total":69502921,
  "max_score":1,
  "hits":[
    ...
  ]
}
```

You get the `"total"` number of hits. It represents the total number of events in the eth.events index.

> The `"max_score"` isn't very interesting to us in general, because we mostly filter for boolean conditions, that can only be `0` (not returned by the query at all) or `1`.

The actual events are listed under the `hits.hits` keyword. If we look at one of the events, we can observe the general structure of an event.

```
{
  "_index":"ethereum_2",
  "_type":"event",
  "_id":"0x92c1b864051b9e6758ab217bc70e0d8641d5f830e16b0a7d15ba78ef2356ba9c_e_52",
  "_score":1,
  "_routing":"0x251d33d4ab03fb675bb2d09304a4aca28b943373c0bd8dbc85402d9e23f4f061",
  "_parent":"0x92c1b864051b9e6758ab217bc70e0d8641d5f830e16b0a7d15ba78ef2356ba9c",
  "_source":{
    "args":[
      {
        "name":"hash",
        "value.hex":"b
↪'eb8dd23ef00be18cb4a263b4271e2f9c28bb47a239f179001691f6e887a6ed47'",
        "value.num":null,
        "value.scaled":null,
        "value.type":"bytes32",
        "pos":0
      },
      {
        "name":"registrationDate",
        "value.hex":"0x59948642",
        "value.num":1502905922,
        "value.type":"uint256",
        "pos":1,
        "value.scaled":null
      }
    ],
    "event":"AuctionStarted",
    "logIndex":{
      "num":52,
      "raw":"0x34"
    },
    "transactionIndex":{
      "num":92,
      "raw":"0x5c"
    },
    "transactionHash":
↪"0x92c1b864051b9e6758ab217bc70e0d8641d5f830e16b0a7d15ba78ef2356ba9c",
    "address":"0x6090a6e47849629b7245dfa1ca21d94cd15878ef",
```

(continues on next page)

(continued from previous page)


```
    "blockHash":"0x251d33d4ab03fb675bb2d09304a4aca28b943373c0bd8dbc85402d9e23f4f061",
    "blockNumber":{
      "num":4145267,
      "raw":"0x3f4073"
    },
    "error":null,
    "str":"AuctionStarted(b\"\\xeb\\x8d\\xd2>\\xf0\\x0b\\xe1\\x8c\\xb4\\xa2c\\xb4
↪'\\x1e/\\x9c(\\xbbG\\xa29\\xf1y\\x00\\x16\\x91\\xf6\\xe8\\x87\\xa6\\xedG\",␣
↪1502905922)",
    "timestamp":"2017-08-11T17:52:02"
  }
}
```


Again, we see meta information that is related to Elasticsearch internals (not shown here).

We want to focus on the event fields, under the `"_source"` keyword:

- `"event"` - event name
- `"blockNumber"` - the block, where it was omitted
- `"timestamp"` - approximate timestamp, when it was included in the blockchain

Each argument of an event is an element in a list `"args"`.

Filter events from a specific contract

You are probably interested in filtering for events that belong to a specific smart contract.

To demonstrate that, we will examine one of the DAI's `DSToken` contracts.

The contract for the DAI Stablecoin on the mainnet resides under the address `0x89d24A6b4CcB1B6fAA2625fE562bDD9a23260359`

The `"address"` field is where the originating contract address is given. You will have to restrict the results with Elasticsearchs filtering methods.

We don't want to use the very limited GET query. We will send a POST request to eth.events, where we provide additional parameters in the body of the HTTP-request:

```
{
  "query":{
    "bool":{
      "filter":{
        "term":{
          "address":"0x89d24a6b4ccb1b6faa2625fe562bdd9a23260359"
        }
      }
    }
  }
}
```

Execute the request with `cURL`:

```
curl -X POST \
  https://api.eth.events/ethereum/ethereum/mainnet/es/event/search/ \
  -H 'Authorization: Bearer d2560f14-1935-44e7-ad3e-a1718dc03bd2' \
  -H 'Content-Type: application/json'
```

(continues on next page)

(continued from previous page)

```
  -d '{
    "query": {
      "bool": {
        "filter":
          {
            "term": {
              "address": "0x89d24a6b4ccb1b6faa2625fe562bdd9a23260359"
            }
          }
      }
    }
  }'
```

The query has to be specified in the `"query"` parameter. We use a filter context `"bool": {"filter": ...}` because we are only interested in filtering elements.

In the `"term"` parameter of the filter context, we require the results to exactly match the specified value in the `"address"` argument of the event, namely the address of the DAI contract.

Filter for a specific type of event

Now every event under the `hits.hits` keyword originates from the contract of interest. but there are still different types of events present in the queries result.

The `"event"` field contains the name of the event, and if you look through the results from the last query, you will most likely see 2 different types of events, `Approval` and `Transfer`.

> *Note*: the feature of filtering by arguments and cleartext names of events is unique to eth.events and it's most outstanding feature. When using the usual `web3` interface, an event and it's values are encoded in a 64 byte hexstring. To decode the event to a human readable and easy to filter representation, the hexstring has to be decoded with the help of the ABI of the events contract.
>
> In eth.events, the events are already decoded and indexed for you!

The DAI contract is following the ERC20 token standard.

From the DAI-Stablecoins ERC20 contracts code, we can see what events are defined:

```
contract ERC20Events {
    event Approval(address indexed src, address indexed guy, uint wad);
    event Transfer(address indexed src, address indexed dst, uint wad);
}
```

If we are interested in one type of event (`"Transfer"`), we have to introduce another `"term"` filter, that gets appended to the `"filter"` list:

```
{
  "query":{
    "bool":{
      "filter":[
        {
          "term":{
            "event.raw":"Transfer"
          }
        },
        {
```

(continues on next page)

(continued from previous page)

```
          "term":{
            "address":"0x89d24a6b4ccb1b6faa2625fe562bdd9a23260359"
          }
        }
      ]
    }
  }
}
```

The `"event"` field defaults to a `text` type for full-text searching. We want to match the event name exactly (case sensitive), so we filter for the `event.raw` field, which is of type `keyword`. To learn more about the differences between `text` and `keyword` types in Elasticsearch, look here.

Execute the request with `cURL`:

```
curl -X POST \
  https://api.eth.events/ethereum/ethereum/mainnet/es/event/search/ \
  -H 'Authorization: Bearer d2560f14-1935-44e7-ad3e-a1718dc03bd2' \
  -H 'Content-Type: application/json'
  -d '{
    "query": {
      "bool": {
        "filter": [
          {
            "term": {
              "event.raw": "Transfer"
            }
          },
          {
            "term": {
              "address": "0x89d24a6b4ccb1b6faa2625fe562bdd9a23260359"
            }
          }
        ]
      }
    }
  }'
```

Retrieving sorted results

You may notice that the `"timestamp"` of the events is outdated and that they are not sorted by their `"blockNumber"`.

In order to change that, the query has to be modified again:

```
{
  "query":{
    "bool":{
      "filter":[
        {
          "term":{
            "event.raw":"Transfer"
          }
        },
        {
```

(continues on next page)

(continued from previous page)

```
          "term":{
            "address":"0x89d24a6b4ccb1b6faa2625fe562bdd9a23260359"
          }
        }
      ]
    }
  },
  "sort":{
    "blockNumber.num":{
      "order":"desc"
    }
  },
  "size":5
}
```

The `"sort"` parameter outside of the `"query"` nesting tells eth.events which field should be used for sorting.

We specify the `.num` attribute of the `blockNumber`, because we want the integer representation and not a hex encoding.

With `"order":"desc"`, the events will be sorted in descending order of the block, where they were included in the blockchain.

Execute the request with `cURL`:

```
curl -X POST \
  https://api.eth.events/ethereum/ethereum/mainnet/es/event/search/ \
  -H 'Authorization: Bearer d2560f14-1935-44e7-ad3e-a1718dc03bd2' \
  -H 'Content-Type: application/json'
  -d '{
    "query":{
        "bool":{
          "filter":[
              {
                "term":{
                    "event.raw":"Transfer"
                }
              },
              {
                "term":{
                    "address":"0x89d24a6b4ccb1b6faa2625fe562bdd9a23260359"
                }
              }
          ]
        }
    },
    "sort":{
        "blockNumber.num":{
          "order":"desc"
        }
    },
    "size":5
  }'
```

Restricting result size

In the last query we specified the `"size"` parameter with a value of `5`. This will limit the number of retrieved events to 5. For testing queries, it is advisable to set this to a small number.

With `"size":-1`, all filtered results are retrieved from the server. You will need to use this in conjunction with a carefully selected range filter, for example a range of block-numbers.

Where to go from here

The best starting point is the Elasticsearch documentation. There you'll learn how to construct more complex filter queries or how to combine filters with a boolean logic.

If you are not interested in single events, but rather on cumulated properties and statistics, you should have a look at the various possibilities of aggregations. .. The example queries at http://eth.events make .. extensive use of aggregations and show how eth.events can be used to .. plot various metrics of different smart contracts.

CHAPTER 2

SQL

2.1 Authorization

To access the SQL interface you need to retrieve your access credentials.

You can register for a **free** account and get the SQL access credentials right now at https://account.eth.events/ in just a few seconds. Go ahead, I'll wait.

2.1.1 Decide on a SQL-Client

Get yourself a proper SQL-Client to access our database. We currently favour PGAdmin4, but it's up to you:

- PGAdmin4
- Valentino Studio

2.1.2 Connect to the database

Setup your connection as mentioned in the account backend. If you need help or any errors occur, please don't hesitate to ask for help from us.

2.1.3 Note

Please keep in mind that the SQL interface is still in a beta status, hence the service might not be constantly available and the database schema could change from one day through another. Now you might proceed using the SQL database or go on to the *tutorial*.

2.1.4 Contact

We might also be able to help you to pin down and visualize the data relevant for your specific use case. Just email us at contact@eth.events - we are excited to see our data used in many new ways!

2.2 Entity Relation Model

2.2.1 Overview

Please also refer to the leading schema defined in elastic *elastic*

2.2.2 SQL tables and data sources

Blocks (*Block*), transactions (*TX*) and logs (*Log*) are coming from the Ethereum blockchain. *Trace* and *InternalTrace* are client specific - here we only save the data from our Parity node. The *Call* and *Event* table contain some of our enriched data, e.g. the decoded name of the contract method (in method and event columns respectively). To enrich the data, we use the tables *Contract* and *Token* to match the hashes etc. As you can see, we are only able to make the data human-readable if we have the contract ABI. If you are interested in monitoring your own (non-standard) contract - contact us and we can integrate your ABI so that you can interpret the data for your business case more easily. Most data in *Contract* and *Token* is pulled from Etherscan at the moment. Particularly regarding contract ABIs we have been in talks about establishing an independent registry. If you want to help or can provide funding to support this mission to help the whole Ethereum ecosystem - please contact us. The arguments of either the *Contract*, the *Call* or the *Event* are encoded as a JSON array in their enclosing tables. In case of referencing a contract that would mean the constructor arguments at the time of creation.

2.2.3 ID fields (primary keys)

The ID (primary key) of the table *Trace* is composed of Keccak256 of: `block-hash + _ + tx-hash + _ + transaction_index`

The ID (primary key) of the table *Log* is composed of Keccak256 of: `block-hash + _ + tx-hash + _ + log_index`

The ID of *Event* is inherited from *Log* table. All other ID fields should be self-explanatory.

2.2.4 Probability fields

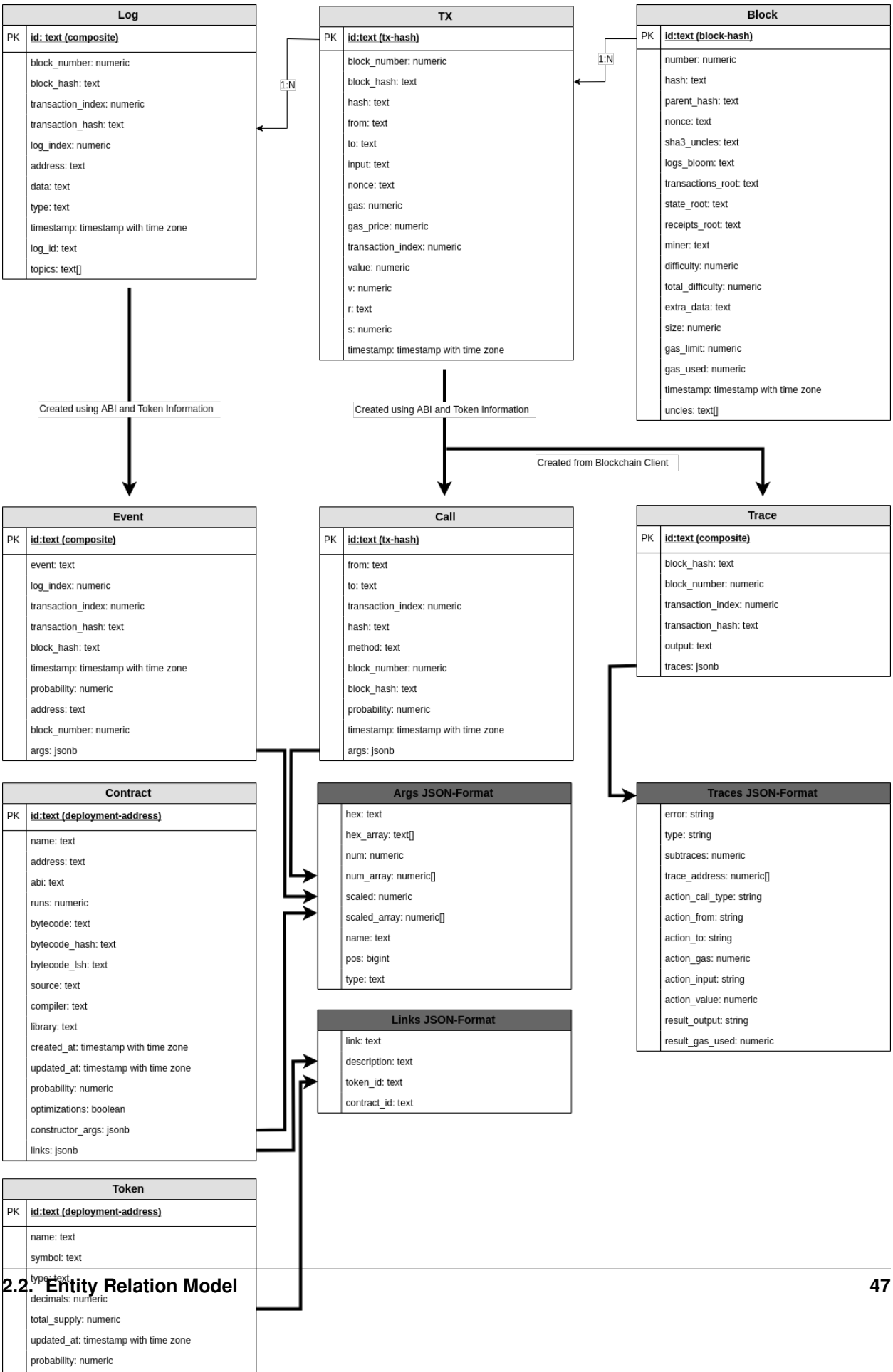
The *Contract* and *Token* tables contain fields for numeric probability. Currently it is set to 1 for all data that we can retrieve from the blockchain or verify via Etherscan. In the near future we plan to compare new contracts on the blockchain against existing ones in order to identify the type (e.g. ERC-20 token, Augur prediction markets, etc.) and enable automatic decoding of events as much as possible. These probabilities are also reflected in the *Call* and *Event* table.

2.2.5 Sub-Entities and their JSON representation

The sub entities - like arguments, links and traces - are encoded in their enclosing tables using the official JSON SQL-Type. Here are some resources to get familiar with that data type:

- JSON Datatype
- JSON Functions

All subentities are stored using the Postgresql-Type **JSONB**.


Log
PK
id: text (composite)
block_number: numeric
block_hash: text
transaction_index: numeric
transaction_hash: text
log_index: numeric
address: text
data: text
type: text
timestamp: timestamp with time zone
log_id: text
topics: text[]
1:N
TX
PK
id:text (tx-hash)
block_number: numeric
block_hash: text
hash: text
from: text
to: text
input: text
nonce: text
gas: numeric
gas_price: numeric
transaction_index: numeric
value: numeric
v: numeric
r: text
s: numeric
timestamp: timestamp with time zone
1:N
Block
PK
id:text (block-hash)
number: numeric
hash: text
parent_hash: text
nonce: text
sha3_uncles: text
logs_bloom: text
transactions_root: text
state_root: text
receipts_root: text
miner: text
difficulty: numeric
total_difficulty: numeric
extra_data: text
size: numeric
gas_limit: numeric
gas_used: numeric
timestamp: timestamp with time zone
uncles: text[]
Created using ABI and Token Information
Created using ABI and Token Information
Created from Blockchain Client
Event
PK
id:text (composite)
event: text
log_index: numeric
transaction_index: numeric
transaction_hash: text
block_hash: text
timestamp: timestamp with time zone
probability: numeric
address: text
block_number: numeric
args: jsonb
Call
PK
id:text (tx-hash)
from: text
to: text
transaction_index: numeric
hash: text
method: text
block_number: numeric
block_hash: text
probability: numeric
timestamp: timestamp with time zone
args: jsonb
Trace
PK
id:text (composite)
block_hash: text
block_number: numeric
transaction_index: numeric
transaction_hash: text
output: text
traces: jsonb
Contract
PK
id:text (deployment-address)
name: text
address: text
abi: text
runs: numeric
bytecode: text
bytecode_hash: text
bytecode_lsh: text
source: text
compiler: text
library: text
created_at: timestamp with time zone
updated_at: timestamp with time zone
probability: numeric
optimizations: boolean
constructor_args: jsonb
links: jsonb
Args JSON-Format
hex: text
hex_array: text[]
num: numeric
num_array: numeric[]
scaled: numeric
scaled_array: numeric[]
name: text
pos: bigint
type: text
Traces JSON-Format
error: string
type: string
subtraces: numeric
trace_address: numeric[]
action_call_type: string
action_from: string
action_to: string
action_gas: numeric
action_input: string
action_value: numeric
result_output: string
result_gas_used: numeric
Links JSON-Format
link: text
description: text
token_id: text
contract_id: text
Token
PK
id:text (deployment-address)
name: text
symbol: text
type: text
decimals: numeric
total_supply: numeric
updated_at: timestamp with time zone
probability: numeric

2.2.6 Full Data model for the Ethereum SQL index

2.3 Tutorials

To get started with eth.events SQL interface, you should have a look at our tutorials. Please also make sure you have had a look at the *database schema*.

2.3.1 Basic SQL Usage Examples

Welcome!

Within this tutorial you will learn how to retrieve and analyze data from the Ethereum Blockchain with the help of the eth.events SQL interface.

We will show you how to retrieve data from eth.events using common SQL language.

What you must know already

This tutorial is written for programmers, who have some experience with SQL. You should also have visited the *authorization* page and setup our connection to the database. Remember that the subentities are stored using the Postgresql-Type **JSONB**. For more information on that, please take a look at the *ER model*.

Basic eth.events SQL queries

Choose one of the databases available. All of them are encoded as the triple of:

```
<technology>_<chain>_<network>
```

The main chain is named *ethereum_ethereum_mainnet*.

After choosing a blockchain, you might continue with the example queries.

Find the latest block

```
SELECT * FROM block ORDER BY number DESC LIMIT 1
```

This will show the whole block. But you can use a shorter form:

```
SELECT max(number) FROM block
```

Find events for a given block

```
SELECT * FROM event
WHERE event.block_number = 7075271
```

Find calls for a transaction hash

```
SELECT * FROM call
WHERE call.hash = '0xadd837afa5b68987eb9f0167ad65cbb8131f57da84db56a19acf4a5a98bd35da'
```

Find transactions for a given contract

For our example we use the address of the TenXPay token contract:

```
SELECT * FROM tx
WHERE tx.to = '0xB97048628DB6B661D4C2aA833e95Dbe1A905B280'
LIMIT 100
```

Find specific events for a given contract

For our example we use the address of the TenXPay token contract again, however we would like to know the values of transfers greater than *1ETH*:

```
SELECT arg->'scaled', arg ->'num'
FROM "event",jsonb_array_elements(args) arg
WHERE event = 'Transfer' AND address = '0xB97048628DB6B661D4C2aA833e95Dbe1A905B280'
AND (arg->'num')::numeric > 1000000000000000000
LIMIT 100
```

Where to go from here

You may continue with taking a look at the *Elasticsearch tutorial*. Please let us know if you have any further questions or need some help with your application.