BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

PLATFORM FOR CRYPTOCURRENCY ADDRESS COLLECTION

PLATFORMA PRO SBĚR KRYPTOMĚNOVÝCH ADRES

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. VLADISLAV BAMBUCH**
AUTOR PRÁCE

SUPERVISOR **Ing. VLADIMÍR VESELÝ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2020

Brno University of Technology
Faculty of Information Technology

Department of Information Systems (DIFS) Academic year 2019/2020

Master's Thesis Specification

Student: **Bambuch Vladislav, Bc.**
Programme: Information Technology Field of study: Information Technology Security
Title: **Platform for Cryptocurrency Address Collection**
Category: Web
Assignment:

1. Study issues connected with cryptocurrencies. Focus on the various identifiers present in the blockchain (i.e., addresses and transactions) and how they are created.
2. Learn how to parse website content and review some of the existing tools (e.g., Scrapy, Lemmit). Based on supervisor's recommendation, identify the interesting pages with cryptocurrencies metadata on the public Internet.
3. Design a platform that would automatically collect and manage metadata (e.g., cryptoaddresses, URLs, usernames) from the selected page set.
4. Implement the proposed platform. In particular, focus on: a) modularity allowing easy integration of new parsers; b) cloud-based parsing scalability; c) monitoring of modules that make up your solution.
5. Validate your implementation, conduct performance measurements, discuss platform's scalability.

Recommended literature:

- Mahto, D. K., & Singh, L. (2016, March). A Dive into Web Scraper World. In *Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on* (pp. 689-693). IEEE.
- Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). *Bitcoin and Cryptocurrency Technologies: a comprehensive introduction*. Princeton University Press.

Requirements for the semestral defence:

- Tasks 1, 2, and 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Veselý Vladimír, Ing., Ph.D.**
Head of Department: Kolář Dušan, doc. Dr. Ing.
Beginning of work: November 1, 2019
Submission deadline: June 3, 2020
Approval date: October 22, 2019

Abstract

The goal of this work is to build a platform for collecting and displaying metadata about cryptocurrency addresses from public and also dark web. To achieve this goal, I use web parsing technologies written in PHP. Challenges accompanying a website parsing are solved by scaling capabilities of Apache Kafka streaming platform. The modularity of the platform is accomplished by microservice architecture and Docker containerization.

The work creates a unique way how to search for potential crypto criminal activities, that appeared outside of the blockchain world, by building a web page application on top of this platform (that serves for managing the platform and exploring the extracted data). The platform architecture allows adding loosely coupled modules smoothly where the Apache Kafka mediates communication of the modules.

The result of this article is meant to be used for cybercrime detection and prevention. Its users can be law enforcement authorities or other agencies and users interested in reputations and credibility of cryptocurrency addresses.

Abstrakt

Cílem této práce je vytvořit platformu pro sběr a zobrazování metadat o kryptoměnových adresách z veřejného i temného webu. K dosažení tohoto cíle jsem použil technologie zpracování webu napsané v PHP. Komplikace doprovázející automatické zpracování webových stránek byly vyřešeny techonologí Apache Kafka a jejími schopnosti škálování procesů. Modularita platformy byla dosažena pomocí architektury "microservices" a "Docker containerization".

Práce umožňuje jedinečný způsob, jak hledat potenciální kriminální aktivity, které se odehrály mimo rámec blockchain, pomocí webové aplikace pro správu platformy a vyhledávání v extrahovaných datech. Vytvořená platforma zjednodušuje přidávání nových, na sobě nezávislých modulů, kde Apache Kafka zprostředkovává komunikaci mezi nimi.

Výsledek této práce může být použit pro detekci a prevenci kybernetické kriminality. Uživatelé tohoto systému mohou být orgány činné v trestním řízení nebo ostatní činitelé a uživatelé, zajímající se o reputaci a kreditibilitu kryptoměnových adres.

Keywords

web scraping, cryptocurrencies, crypto crime detection, Apache Kafka, microservices, Laravel

Klíčová slova

web scraping, kryptoměny, kybernetická bezpečnost, Apache Kafka, microservices, Laravel

Reference

BAMBUCH, Vladislav. *Platform for Cryptocurrency Address Collection.* Brno, 2020. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Vladimír Veselý, Ph.D.

Rozšířený abstrakt

Technologie kryptoměň nabízí alternativní možnosti při placení za služby různého druhu. Hlavními výhodami této alternativy jsou pseudoanonymita transakcí, rychlost a nižší poplatky za zpracování transakcí. Kryptoměny těmito vlastnosti disponují díky technologii blockchain, kterou používají. Blochchain pak přidává do celkových vlastností decentralizaci, neměnnost, transparentnost a použití asynchronní kryptografie.

Právě díky těmto výhodám, zejména pseudoanonymitě transakcí, patří kryptoměny mezi oblíbené způsoby placení za ilegální služby a produkty. Z pohledu odhalování kybernetických zločinů je tedy záhodno, aby bezpečností služby dokázaly zjistit pravou identitu uživatele podporujícího zločiné aktivity.

Při placení pomocí krytoměn, uživatel generuje množství adres, které disponují určitým finančním obnosem, a tyto adresy používá v rámci jednotlivých transakcí. Do jiné transakce může vstoupit vícero adres a jejich obnosy se tímto způsobem sčítají. Pro použití vícero adres na vstupu jedné transakce, musí mít uživatel přístup ke všem použitým adresám. To jinými slovy znamená, že je vlastní. Pokud bychom tedy dokázali určit vlastníka jedné adresy, můžeme touto závislotí rozklíčovat i celou jeho historii.

K re-identifikaci vlastníků kryptoměnových adres můžeme použít informace, veřejně dostupné na různých webových stránkách a fórech. Pro účely detekce kybernetických kriminálníků by tedy bylo výhodné, když by velké množství uživatelů prohlášovalo, že konkrétní adresa je jejich a lidé jim na ni mohou posílat peníze. Jednou z možností, jak takové informace z webových stránek dolovat, je "web-scraping".

Web-scraping funguje na pricipu automatického procházení webu pomocí robotů, kteří extrahují zajímavá data a ukládají je do persistentního úložistě. Má však ale i své stinné stránky a majitelé webových služeb se proti nim snaží bojovat. Problémy totiž nastávají, pokud z jednoho webu extrahuje data velké množství robotů. Ti pak mohou způsobit degradaci poskytovaných služeb nebo i úplné zahlcení požadavky na odpověď a následný pád serveru. Existují proto jisté mechanismy, které detekují roboty s příliš velkou četností dotazů nebo jiným nedovoleným chováním. Při návrhu dolovacích robotů je tedy třeba o zmíněných problémech vědět a snažit se o to, aby nebyli detekováni a blokováni.

Webových stránek, obsahující různé informace o krytoměnových adresách a jejich potenciálních vlastnících je velké množství. Většina těchto stránek má rozdílnou strukturu a i konkrétní informace mohou být různého typu. Tyto aspekty musíme brát v potaz a vytvořit vhodnou databázovou sturukturu, vyhovující všem požadavkům. Pro snadné použití vydolovaných dat musí být totiž struktura dat jednotná a v ní informace s maximální datovou kvalitou.

Všechny výše uvedené problémy jsou obsaženy v této diplomové práci, je navrhnuto řešení, popsána jeho implementace, testování a zhodnoceny výsledky. Výsledkem práce je platforma, poskytující snadné přidání nových robotů pro web-scraping a jednotné aplikační rozhraní pro komunikaci mezi roboty a úložištěm. Platforma umožňuje paralelní běh robotů se škálováním jednotlivých instancí technologii *Apache Kafka*. Správu robotů a hledání ve vydolovaných datech zajišťuje webová aplikace, vytvořená speciálně pro tyto účely.

Platforma je vyvinuta v jazyce PHP 7.4 s použitím *Laravel* frameworku. Web-scraping je implementován pomocí knihovny *Goutte*. Pro zajištění modularity platformy je použit návrhový vzor "micro-services" a to v kombinaci s *Docker* kontejnery. Celé řešení je umístěno na dedikovaném serveru, umožňující permanentní běh platformy, monitoring dílčích částí platformy zajišťuje *Graylog* a *Lenses.io*.

Platform for Cryptocurrency Address Collection

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Mr. Ing. Vladimír Veselý, Ph.D. The supplementary information was provided by Mr. Petr Kuznik and Mr. Vladimír Alfery. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Vladislav Bambuch
June 3, 2020

Acknowledgements

Many thanks to my colleagues Mr. Ing. Tomáš Kocman for helping with the integration of his master's thesis and Mr. Ing. Jan Pluskal for enabling the possibilities to use multiple IP addresses for the scraping purposes. Many thanks also to my supervisor Mr. Ing. Vladimír Veselý, Ph.D for a great support and professional guidance. Heartfelt thanks go to my girlfriend and my family for endless support and love.

My personal favourite food recipe is very simple, not very time-consuming, and it can produce a mass amount of prep-meal, I like to do. You will need fresh, dried and canned tomatoes, potatoes, onions, garlic, beans (whichever type you like, canned or overnight-soaked), tempeh, olive oil/butter, oregano and your favourite cheese.

You chop onions and put them onto a preheated pan with olive oil or butter. When the onions are golden-fried, you add chopped dried tomatoes and mix them together with chopped temped. While the pieces are linking together, you prepare slices of fresh tomatoes and potatoes. When the mix is almost finished, you add chopped garlic and oregano.

The ingredients are ready, so you start putting them into a baking dish. You start with the sliced potatoes followed by the sliced tomatoes and the onion-tempeh-garlic mixture. You are repeating this process of making such food levels when adding prepared beans and systematically pouring the canned tomatoes into the baking dish. When you run out of ingredients or baking dishes, you stop.

You put the baking dish(es) into your 200℃-preheated oven for about 45 minutes. Then, you switch off the oven and set your delicious cheese on top of the yummy dish. Let the cheese melt down into the underlying meal structures, and after 15 minutes, you are ready to go. Enjoy!

Contents

Chapter 1

Introduction

This thesis is about the design and implementation of a computer system that aims to be used for crypto criminal activities detection. Its outputs may be used as legal evidence of unlawful practices. The system collects important information from interesting web pages and saves it into storage for further processing. This capability might be used to match anonymous data with known entities. The re-identification of anonymous data is very useful for investigation of cryptocurrency transactions. The system provides features for data collection maintenance and also searching in the data.

Using cryptocurrency transactions, a criminal can exchange money for illegal services very easily due to its anonymization features. We can only obtain the information about who is an owner of a cryptocurrency address when the person makes it publicly available. Due to the natural behaviour of Internet websites and its dynamic content changes, we need to look up for this type of information regularly. This topic is worthy of exploration as we do not have sufficient tools freely available that would address the mentioned difficulties. It was the mentioned difficulties and interest in information security that motivated me to choose this topic.

The internet contains a lot of websites focusing on cryptocurrency address displaying, and some of them also contain owners for particular addresses subset. Other websites are dedicated to collecting addresses seen in fraudulent emails and other forms of extortion. However, we lack a web application that would unify and link all the data from mentioned websites and make possible to search in the data. The result of this thesis aims to fill this blind spot.

This work is meant to be a part of the project *Integrated platform for analysis of digital data from security incidents*[1], developed at Brno University of Technology and it is planned to integrate it with its other parts.

In Chapter 2 and Chapter 3 it is possible to read about what was necessary to investigate to understand cryptocurrency technologies and web parsing challenges. Chapter 4 contains a writing about designing back-end part of the system. Chapter 5 includes implementation details. In Chapter 6, one can read about the validation of the results and performance measurement. Chapter 7 gathers findings that did not fit into the scope of this theses. In the last, Chapter 8, there is a summary of the whole thesis and more details about the plans for the future.

[1] https://www.fit.vut.cz/research/project/1063/.en

Chapter 2

Challenges of Cryptocurrencies

This chapter contains a basic overview of cryptocurrencies and its use-cases. There are also mentioned possible threats when criminals use this technology and how can we deal against them. It focuses more on explaining how crypto addresses and transactions are created and validated. There are listed five most popular cryptocurrencies, at the of time writing, and described how they work from creation and validation point of view.

2.1 Introduction into Cryptocurrencies

Cryptocurrency enables an alternative way of how people can send their assets between each other. It is based on strong cryptography principles and leverages from decentralized network architecture. People can create a crypto wallet that uses crypto addresses in the same fashion as in the conventional payment process, where a bank handles everything. As a ledger, it uses a technology called blockchain, which stores immutable information about all transactions and protects it against malicious changes. Each transaction is validated during a process called mining that's performed by computer programs. More general information about this topic can be found in the thesis [44] and [38].

2.1.1 Crypto Wallet

Crypto wallet generates and stores public/private key pairs which are used for making crypto transactions. From a single private key, the wallet generates a particular public key. From the public key is then generated a crypto address, that is used in a transaction. Private key allows to sign the transaction where the address is present. A receiver can use the public key for verification of the transaction.

One wallet can contain multiple cryptocurrencies. A wallet keeps track of all public/private keys and all addresses generated from them.

2.1.2 Crypto Addresses

Crypto address is a hash string that is generated from a public key and participates in a crypto transaction. An address holds some assets – based on the specific cryptocurrency – which can be used in a transaction to send them into a different address. More addresses can participate in a single transaction. If there are some assets left as a result of a transaction, a wallet returns these assets to a new address generated in the sender's wallet.

2.1.3 Crypto Transactions

When a transaction is made, a wallet will try to find sufficient funds across all addresses in the wallet and sign each input from an address to the transaction by a private key paired with the address. Every transaction needs to be verified and stored in a ledger. There are two options of how the verification is done.

Proof of Work This verification works on a principle of solving a difficult algorithm. When it's done, a new transaction block is created and stored into a blockchain. This process is called mining and is performed by crypto network nodes. The node is called miner and is rewarded by a small amount of cryptocurrency for every new block verified and placed into a blockchain [36].

Proof of Stake This method works on a principle of voting if a new block is valid or not. A vote is based on an amount of money an author uses to guarantee the block is valid. Every node that creates invalid block lose the amount of coins it voted. Proof of Stakes aims to solve known issues of Proof of Work [31].

2.1.4 Blockchain

Blockchain is a sequence of blocks connected in a linear fashion. Every block contains a header with a meta-data about the block, a data and a hash of the meta-data. The header also includes a pointer to a hash of a previous block as can be seen on the figure 2.1 The data inside of every block are crypto transactions stored in a data structure called *Merkle tree*[25]. This chaining leads to the benefit of immutability of the sequence. Every change in the previous blocks produces changes in all successors. So if a hacker tried to modify the ledger, it would need to recalculate all blocks of the blockchain from the point of change and convince the verifying nodes, those new blocks are valid.

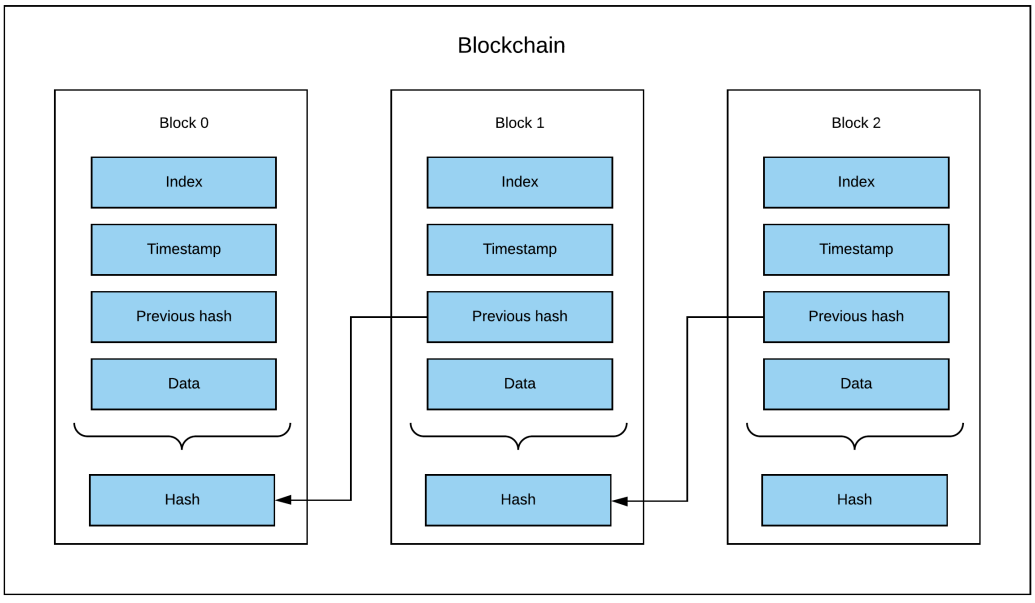

Figure 2.1: Blockchain

2.2 Cryptocrime and Cryptopolice

Using mentioned technologies disposing anonymity, decentralization and asynchronous cryptography features, it's fairly easy to commit a crime and be paid for it using cryptocurrencies. Even so, every transaction is public, no one knows who is an owner of a wallet, assigned to an address unless it exposes by its own.

Criminals use cryptocurrencies for blackmailing, frauds, sextortion activities and for business with illegal drugs, weapons, human organs and child pornography [22]. Kidnappers also uses benefits of cryptocurrencies in these days [11].

In the Czech Republic, there are multiple Police units that could benefit from the results of this thesis and possibilities it enables. For example National headquarters against organized crime[1], Institute of Criminology[2] or National Drug Headquarters[3].

2.3 Cryptocurrency Identifier Comparison

This section consists of several most favourite cryptocurrencies, according to *coinmarketcap.com*, and comparison of their address and transaction creation.

2.3.1 Bitcoin

Address Creation

Bitcoin address is almost always [2] a user's public key [7], hashed by SHA-256[4] algorithm and then encoded using base58[5]. Bitcoin also uses SHA-256 to create a checksum [39]. Before the address is encoded by base58, it has following format:

```
[one-byte version][20-byte hash][4-byte checksum]
```

`20-byte hash` is the hash of the script that will be used to spend the amount of coins. More details about bitcoin address creation is in the article [6].

To validate Bitcoin address, at first, we need to compare checksum. Checksum is created from first 4 bytes of double hashing `version` and `hash` using SHA-256. Then we check the version byte. For the main Bitcoin blockchain, following values are valid: `['00', '05']`, for Bitcoin testnet are valid `['6f', 'c4', '3c', '26']` values.[6]

Transaction Creation

Coinbase/Generation transaction is the first transaction of a block, generated by a miner of the block. The transaction contains *coinbase*, which is a special field that is used to claim a reward of mining of the block.

Regular transaction has an input, an output parts, hash and a fee for a miner. This transaction is also prefixed by *transaction version number* which tells miners how the transaction should be validated. The input contains three fields:

[1] https://www.policie.cz/clanek/narodni-centrala-proti-organizovanemu-zlocinu-skpv.aspx
[2] https://www.policie.cz/kriminalisticky-ustav.aspx
[3] https://www.policie.cz/narodni-protidrogova-centrala-skpv.aspx
[4] https://en.wikipedia.org/wiki/SHA-2
[5] https://en.bitcoin.it/wiki/Base58Check_encoding
[6] https://github.com/christsim/multicoin-address-validator

- an outpoint – a bitcoin address, references to a previous output of a transaction.
- a signature script – is used to define specific conditions how the input should be spend. It provides data into *pubkey script*.
- a sequence number – allows to create unconfirmed transactions, that can be locked and only executed in the future. Setting sequence number to value `0xffffffff` will disable the lock [7].

The output contains:

- amount of transferred bitcoins
- output Bitcoin address
- a pubkey script – anyone who satisfy conditions in the pubkey script can use amount of transferred bitcoins in future transactions.

2.3.2 Ethereum

Address Creation

Ethereum addresses are also created from public keys as Bitcoin does [3]. The address is created by applying Keccak-256[7] on the public key, taking first 20 bytes from the result and adding `0x` at the start of the result. In compare to Bitcoin, Ethereum uses optional checksum, because the checksum was introduced through one of *Ethereum Improvement Proposal*[8] [8]. It is up to a user to publish their own address without checksum, but it can lead to typos in hand-written addresses and to destroying the coins.

To make Ethereum address with a checksum, we need to convert the address without `0x` string into hexadecimal representation. Every character with hexadecimal value higher than or equal eighth is converted into upper-case typeface. The rest of the characters is converted into lower-case typeface[9].

To validate an address by the checksum, we convert the address (without `0x`) into lower-case representation, we hash it with Keccak-256 and compare lower/upper-case of all characters [33].

Transaction Creation

Ethereum has three types of transactions [35]:

- Fund Transfer Between EOA – is used for assets transfer between two Externally-Owned Accounts
- Deploy a Contract on Ethereum Network – is used for deploying a compiled contract on the network
- Execute a Function on a Deployed Contract – when a contract is deployed, it can be executed by this transaction

Every transaction contains these attributes [27], [9]:

[7] https://keccak.team/keccak_specs_summary.html
[8] https://eips.ethereum.org
[9] https://en.wikipedia.org/wiki/Typeface

- nonce – it's a counter of executed transactions for particular account.
- gasPrice – serves as a fee for a miner. When an user creates Ethereum transaction, it defined how many coins it's willing to pay for every executed operation.
- gasLimit – is used to set a maximal limit of coins, the transaction creator is willing to spend.
- to – contains Ethereum address of a recipient.
- value – how many coins should be transferred.
- data – can be a custom message or a code to change/create a smart contract[10].

In compare to the Bitcoin transaction, there is no `from` attributes. This information is derived from a private key during the signing process.

When a user creates a transaction, it needs to be signed by the user's private key. Then the signed transaction is submitted into the user's *local Ethereum node*, which validates if the transaction is signed by the correct key.

2.3.3 Ripple - XRP

Address Creation

There are two types of Ripple addresses, *a classic address* and *X-address*. X-address[11] is currently in the process of proposal and details won't be covered in here.. The classic address is created from an account's public key and contains a checksum. The address is case-sensitive, has between 25 and 35 in length and consists of alphanumeric characters with the exclusion of the number zero `"0"`, the capital letter `"O"`, the capital letter `"I"` and lowercase letter `"l"`. The classic XRP address has the following format:

```
[20-byte account ID][4-byte checksum]
```

The account ID is produced by concatenating a value `0x00` with a hash of an account's public key. The checksum is then created from the first 4 bytes of double hashing an account ID using SHA-256. Concatenating the account ID with the checksum and sending it into XRL base58 encoding function, with a special dictionary[12], will produce the address [43].

Validation of Ripple address consists of checking its characters against the mentioned dictionary. Verification of the checksum is performed after that.

Transaction Creation

Ripple introduces several types of transactions: *Payment, OfferCreate, OfferCancel, TrustSet, AccountSet, SetRegularKey, SignerListSet, EscrowCreate, EscrowFinish, EscrowCancel, PaymentChannelCreate, PaymentChannelFund, PaymentChannelClaim and DepositPreauth.* Some of those transactions have custom attributes and lot of attributes are optional. Common attributes for all transactions are described on official website[13]. Transactions also contain additional meta-data[14].

[10] https://www.coindesk.com/learn/ethereum-101/ethereum-smart-contracts-work
[11] https://xrpaddress.info
[12] rpshnaf39wBUDNEGHJKLM4PQRST7VWXYZ2bcdeCg65jkm8oFqi1tuvAxyz
[13] https://xrpl.org/transaction-common-fields.html
[14] https://xrpl.org/transaction-metadata.html

For the purpose of transaction comparison between multiple cryptocurrencies, we will cover only the most frequent transaction `Payment`[15] and its required/strongly recommended attributes:

- Account – the unique XRP address of the sender.
- TransactionType – the type of the transaction, mentioned above.
- Fee – the cost for processing this transaction in Ripple network
- Amount – the amount of coins that should be send.
- Destination – the unique XRP address of the recipient.
- Sequence – sequence number of the sender account.
- LastLedgerSequence – defines how long can be the transaction wait for validation. (strongly recommended)
- SigningPubKey – Hexadecimal representation of the sender public key.
- TxnSignature – prove, that the transaction is originating from the Account.

2.3.4 Bitcoin Cash

Address Creation

Bitcoin Cash began as a *hard fork*[16] of Bitcoin blockchain. Therefore it needed to create new coin name and format. *Legacy* format is the standart Bitcoin 2.3.1 address and *CashAddr* is the new format, described in this section. CashAddr has following format:

- prefix – defining a blockchain for which is an address valid. `bitcoincash` for Bitcoin Cash mainnet, `bchtest` for testnet `bchreg` for regtest.
- a separator – fixed value `:` (a colon).
- payload – base32 encoded address with a checksum.

The payload of the CashAddrs is formated like this:

```
[version byte][hash 20-64 bytes][40 bits checksum]
```

The version byte describes the length of the following hash. Value `0` means the hash with 20 bytes in length, and `7` means 64 bytes of hash. The complete version byte table is available in this specification [4]. The hash is produced by hashing an account's public key in case of P2PKH[17] or a reedemScript in case of P2SH[18]. The checksum computation involves the prefix to secure an address is valid in only one blockchain and therefore, it's not possible to destroy coins by sending them into testnet. The computation result is a BCH cyclic error-correction code[19] defined over `GF2^5`[20].

[15] https://xrpl.org/payment.html
[16] https://www.investopedia.com/terms/h/hard-fork.asp
[17] https://bitcoin.org/en/glossary/p2pkh-address
[18] https://bitcoin.org/en/glossary/p2sh-address
[19] https://en.wikipedia.org/wiki/BCH_code
[20] https://en.wikipedia.org/wiki/Finite_field

Transaction Creation

Bitcoin Cash transactions are created in the same way how Bitcoin blockchain does it. There are differences in block size, transaction fees and speed of validation, but the execution process is the same [41]. Additional details about transactions are described in following specification [5].

2.3.5 Litecoin

Litecoin began as a fork of Bitcoin project[21]. It shares most of the parameters [24] except hash algorithm used for the mining [23].

Address Creation

The only difference between Litecoin and Bitcoin address are version bytes. For mainnet, Litecoin picked `['30', '05', '32']` values and for testnet `['6f', 'c4', '3a']` values. Some values `'05', '6f' and 'c4'` are shared with Bitcoin 2.3.1. It means, Litecoin addresses with these version bytes can be used in both Bitcoin/Litecoin blockchains as they were created before the fork happened.

Transaction Creation

Both Litecoin and Bitcoin uses Proof of Work type of verification. By the fork, authors of Litecoin wanted to target the bottleneck of Bitcoin transaction approval, which is the speed. So the only difference in transaction creation is that Litecoin uses *Scrypt* algorithm[22], instead of SHA-256, which make the mining 4-times faster [23].

2.3.6 Cryptocurrency Differential Table

There are summarized differences between addresses of from above-mentioned cryptocurrencies.

Name	Bitcoin	Ethereum	Ripple	Bitcoin Cash	Litecoin
Shortcut	BTC	ETH	XRP	BCH	LTC
Hash length	20B	20B	20B	20-64B	20B
PK hash alg.	SHA-256	Keccak-256	SHA-256	SHA-256	SHA-256
Checksum len.	4B	0B	4B	40b	4B
Checksum algorithm	Double SHA-256	Mixed-case encoding	Double SHA-256	BCH over $GF(2^5)$	Double SHA-256
Main/Testnet	version bytes	network ID	rippled.cfg	version bytes	version bytes
Final encoding	base58	none	base58	base32	base58

Table 2.1: Crypto currencies differences

[21] https://github.com/litecoin-project/litecoin
[22] https://litecoin.info/index.php/Scrypt

2.4 Chapter Summary

In this chapter, we4 introduced a basic overview of individual parts of cryptocurrencies and another thesis, where we can go deep in detail of this topic. The main focus in this part of the thesis was in describing the differences in address and transaction creation details for several most favourite cryptocurrencies. The attributes and algorithms for particular cryptocurrencies were summarized using the differential table in the last part of the chapter.

Chapter 3

Crypto Websites Parsing

This chapter contains a general introduction into web parsing, the motivation of doing that and also alternatives we should consider before starting parsing some pages. There will also be mentioned interesting websites containing crypto address metadata and useful technologies, making web parsing easier. In this chapter, we use expressions parsing and scraping regularly.

3.1 Motivation

We should first mention a motivation of why do we need to parse websites automatically. The main problem that web parsing solves is actually natural dynamic behaviour of the World Wide Web. Its content is changing very often, but users can see only current, the last, version of it. The history is often hidden and not even accessible. But what if a service needs to work with all the data some website has ever exposed? If the website doesn't expose API – will be discussed later – we don't have any other options but to scrape the content regularly.

List of possible use-cases:[1][2]

- A market analysis.
- Gaining a competitive advantage.
- Monitoring trends.
- News aggregations.
- SEO[3].
- Building custom datasets.

3.2 Web Parsing Workflow

Website owners usually do not want to be scraped. Therefore, they implement anti-scraping techniques. Examples of these techniques will be covered in this section. To do web parsing

[1] https://scrapinghub.com/use-cases
[2] https://captaindata.co/blog/11-reasons-why-use-web-scraping/
[3] Search engine result page monitoring

successfully, we need to have all those techniques in our minds when designing web parsing workflow.

3.2.1 Web Parsing Challenges

As mentioned above, web parsing is full of challenges we need to count with. A list of them is available below. In articles [42], [37] and [21] can be found even more challenges and possible solutions.

Structural Changes

First of all, we need to detect all structural changes in a scraping website and trigger a notification when it happened. The change can happen just by releasing new web UI, but also it can happen more frequently as prevention from being scraped. Interesting idea is to use *Declarative DOM extraction expression evaluator*[4] and be notified whenever remote site DOM structure changes occur[5].

Traffic Throttling

This technique prevents more often from DDoS attacks but can also lead to hinder the web scraping implementation. The principle is quite simple if one client produces too many requests, responses for the client are going to be slowed down or whole connection is closed.

Browser Fingerprinting

This is related to previously mentioned *Traffic throttling*. A web server can start traffic throttling by detecting too much requests from one IP address. Browser fingerprint is additional information that increases chances to detect suspicious client. Web browser sends additional data with every request to a web server.

Examples of data shared with websites by browsers:

- the User agent header – identification of operation system, the application and browser version.
- the Accept header – defines which response types are understandable for the client.
- the Connection header – controls if the network connection supposed to be open after the current session ends.
- the Accept Encoding header – which encoding the client expects in the response.
- the Accept Language header – describes which language and locale is expected.
- the list of plugins
- the platform
- the cookies preferences (allowed or not)

[4] https://github.com/gajus/surgeon
[5] https://medium.com/@gajus/it-is-a-really-silly-idea-to-use-puppeteer-to-scrape-the-web-da62a9f3de7e

- the timezone
- the screen resolution and its color depth
- the list of fonts

These data can be used to create a fingerprint of a client's browser[6].

3.2.2 Crypto Websites Parsing

Specific use case of web parsing are websites containing cryptocurrency data and metadata. Those websites are usually very dynamic from a content point of view. If they contain crypto wallets, the list of assigned crypto addresses might not be full as some specific wallets grow a lot.

It's also worth to mention that some websites show crypto address metadata based on what users posted there. To keep data quality of the scraped information at the highest possible level, we need to make sure every crypto address is validated and matched with some specific cryptocurrency address structure. Some cryptocurrencies have case sensitive addresses identifiers so it's quite error-prone when a user doesn't paste it from a clipboard.

In general, data from every website containing user inputs need to be validated. Otherwise, we can end up with tons of parsed data with no value because we can never trust the source made proper validation for us.

3.2.3 Parsing Web Forums

Web forums are very different in comparison to simple informative websites. They have their own structure of how topics are aggregated into groups and how users interact with these groups. The structure is meant to help users in their daily use cases. So not all information are easy to list out. For users, it's usually handy to list all topics assigned to one group. On the other hand, it's not necessary to list all user accounts.

These limitations make web forum parsing harder, and a developer needs to understand them when designing the scraping flow.

3.3 Useful Web Parsing Technologies

In this section, we will go through some technologies web parsing community consider the most useful.

First of all, it's necessary to mention, we focused mainly on PHP technologies as this thesis is meant to be written in this language, but some Python frameworks are even more popular and enhanced.

A short comparison of available web scraping PHP libraries can be found in this article [18].

3.3.1 Goutte

Goutte[7] is a library for web crawling and scraping that is written in PHP. In the PHP community, Goutte is kind of standard solution for scraping. It's very easy to get a web

[6] https://restoreprivacy.com/browser-fingerprinting/
[7] https://github.com/FriendsOfPHP/Goutte

page, filter some HTML elements, iterate over them and extract all necessary data. On the other hand, it doesn't handle JavaScript-generated SPAs[8]. It also doesn't offer any advanced features like *auto-throttling* or *User-Agent spoofing* natively as Scrapy 3.3.4 does.

This library is dependent on following Symfony components[9]:

- BrowserKit - the main component of Goutte, is used to simulate real browser behaviour. It can create requests, to make click on HTML components and also submit forms.
- CssSelector - this component converts CSS selectors into their XPath equivalents. It does that because XPath expressions are far more powerful and therefore preferable technique for web parsing.
- DomCrawler - a component that aims to make DOM navigation through HTML and XML documents easier.
- HttpClient - is a low-level HTTP client capable of doing simple synchronous and as well as concurrent asynchronous streamed operations.

3.3.2 Symfony Panther

Panther[10] is a browser testing and web scraping library and coexist as a part of the PHP Symfony framework. It uses almost the same Symfony Components as Goutte 3.3.1 does, but unlike Goutte, it does handle SPA websites by using real browsers to execute JavaScript. Panther is built on top of PHP Webdriver[11] library, used to control web browsers from PHP.

Full list of Panther library features:

- Executes the JavaScript code contained in web pages.
- Supports everything that Chrome (or Firefox) implements.
- Allows screenshots taking.
- Can wait for asynchronously loaded elements to show up.
- Lets you run your own JS code or XPath queries in the context of the loaded page
- Supports custom Selenium server installations.
- Supports remote browser testing services including SauceLabs[12] and BrowserStack[13].

The library can be configured to use Goutte as well, if no SPA capabilities are needed, because of the same API both libraries use [14]. Panther was introduced in 2018 and not many articles with examples can be found.

The disadvantage of executing real browser for scraping are computational costs which results in delayed PHP crawler requests and lower performance overall [18].

[8] Single Page Applications
[9] https://symfony.com/doc/current/components
[10] https://github.com/symfony/panther
[11] https://github.com/facebook/php-webdriver
[12] https://saucelabs.com
[13] https://www.browserstack.com

3.3.3 Laravel Dusk

Laravel Dusk[14] is very similar to Symfony Panther 3.3.2 and can be configured to use Webdriver as well as ChromeDriver[15] in headless mode[16]. The main use case is to create automated application tests in real browsers, but it can also be used for scraping purposes [16].

Dusk is also fairly new. It was introduced in Laravel 5.4, released in 2017[17]. Even though scraping is not its main purpose, it might be useful for Laravel projects, because of its simple integration with other Laravel features[18] like: task scheduler, queues, storage manager and notificationshttps://laravel.com/docs/master.

3.3.4 Scrapy

Scrapy[19] is a web crawling and scraping framework build in Python. It's an open-sourced project with a strong collaborative community. As a framework, it encapsulates implementation details into a high-level architecture. That means a developer only uses a simple object-oriented API to build it's own web crawling/scraping project. Scrapy can also be used to data mining, monitoring, automated testing, information processing or historical archival.

Every website request, created using this framework, is scheduled and processed asynchronously and disposes default error handling functionality. This enables us to perform very fast, parallel and fault-tolerant HTTP calls.

Scrapy requests can be configured with custom delays between each request, level of concurrency and bunch of other settings to increase its politeness, or with auto-throttling extension to handle this automatically. The main part of Scrapy are Spiders that defines how should be specific site parsed and what data should be extracted.

List of Scrapy features:

- Extended CSS selectors and XPath expressions for interactions with HTML/XML sources.
- An interactive shell console for debugging and testing Scrapy capabilities.
- Result exports in multiple file formats (JSON, CSV, XML) into multiple data storages (FTP, S3, local).
- Auto-detection and correction of data encoding.
- Auto-follow of URLs found of a web page.
- Custom scraping stats collection.
- Deploying Spiders to Scrapy Cloud[20] or other cloud solutions using Scrapyd[21].

[14] https://github.com/laravel/dusk
[15] https://chromedriver.chromium.org/
[16] https://chromium.googlesource.com/chromium/src/+/lkgr/headless/README.md
[17] https://en.wikipedia.org/wiki/Laravel
[18] https://laravel.com/docs/master
[19] https://docs.scrapy.org/en/latest/index.html
[20] https://scrapinghub.com/scrapy-cloud
[21] https://scrapyd.readthedocs.io/en/stable/overview.html

- Lot of extensions and middlewares:
  - Handling cookies and sessions
  - Handling robots.txt
  - HTTP compression, authentication and caching
  - User-Agent spoofing
  - Crawl depth limitation
  - Crawler control and execution using JSON-RPC[22]
- Telnet console for connection into running Scrapy process.
- Pre-defined reusable Spiders:
  - CrawlSpider – for crawling regular websites
  - XMLFeedSpider – for parsing XML feeds using a certain node name
  - CSVFeedSpider – similar to XMLFeedSpider but iterates over rows instead of nodes
  - SitemapSpider – for crawling a site using URLs from sitemaps

3.3.5 Lemmit

Lemmit is a part of a project *proof_platform*[23] introduced in a theses *Automated Web Analysis and Archivation*[24]. It works as a web archiving service for static and also dynamic web pages. Lemmit uses MAFF[25] data format for storing whole web page copies.

The great thing about Lemmit is the capability of archiving SPA[26], build in JavaScript. Those type of web pages doesn't show whole content when the page renders for the first time, but dynamically as a user interacts with the page. To simulate this user activity, Lemmit uses Selenium[27] framework.

This project can be very useful to connect with the platform, building in this thesis, and save a copy of every page in a time of its parsing. It can be useful for validation purposes as well as a legal burden of proof for investigators and security experts.

More information about Lemmiwinks, the ancestor of Lemmit, design and architecture can be found in work *Automatic Webpage Reconstruction*[28].

3.4 Parsing versus API Usage

The alternative way to web parsing, we should always consider, is usage of API. With API, one can get structured data easily just by sending HTTP GET request. If all desired data are available from API response and a developer always needs only the same data with a structure that doesn't change in within a time, the usage of API is the simplest and the best solution. However, there are significant limitations of API, which will be covered further [10].

[22] https://www.jsonrpc.org/specification
[23] https://gitlab.com/tomaskocman/proof$_platform$
[24] https://www.vutbr.cz/en/students/final-thesis?zp$_id$ = 121925
[25] Mozilla Archive Format
[26] Single Page Application
[27] https://selenium.dev/
[28] https://www.vutbr.cz/en/students/final-thesis/detail/114844

3.4.1 API Usage Limitations

- Availability issues – APIs don't need to provide access to all data a service maintains.
- Poor customization – APIs usually expose only a single type of data structure.
- Rate limiting – not all APIs are public and access it monitored per authenticated user.
- Legality – an usage of API can be limited by legal rules every user needs to comply.

3.4.2 Web Scraping Advantages

- Fresh data – data exposed through API don't need to be necessarily up-to-date with web content.
- Better customization – scraping allows extracting every information on a web page using whichever structure needed.
- Less strict rate limits – when used wisely, web scraping can overcome common API rate limits.
- Anonymous requests – scraping doesn't use any authentication token for access the data.

The most simple web scraping would be using sitemaps[29]. A sitemap is a public file, allocated directly on websites and contains information about all web pages available for crawlers, which the website contains. If the sitemap holds all web pages links, we don't need to crawl the website in the first place. On the other hand, this source isn't reliable as we cannot be sure all links are actually present there.

Another publicly available file allocated on websites and associated with web parsing is *robots.txt*, where restrictions for crawlers are stated. Every polite web scraping developer should respect this file and mentioned restrictions[30].

3.5 Crypto Websites Interesting for Parsing

This section describes websites, interesting for parsing and extracting cryptocurrency information. Each website contains crypto addresses and specific meta-data for each of them. Every website has a unique data structure and shows different meta-data. Therefore gathering different meta-data about one crypto address from multiple sources brings an opportunity to understand the behaviour of the crypto address owner. This possibility is essential for crypto crime detection and analysis.

3.5.1 Bitcointalk.org

Bitcointalk.org is a web forum dedicated to discussions about cryptocurrency technologies and mainly to bitcoin itself. The web was launched in 2011[31] and since that more than 2.7M user accounts have created more than 53M posts.

[29] https://www.sitemaps.org/
[30] https://www.robotstxt.org/robotstxt.html
[31] http://whois.domaintools.com/bitcointalk.org

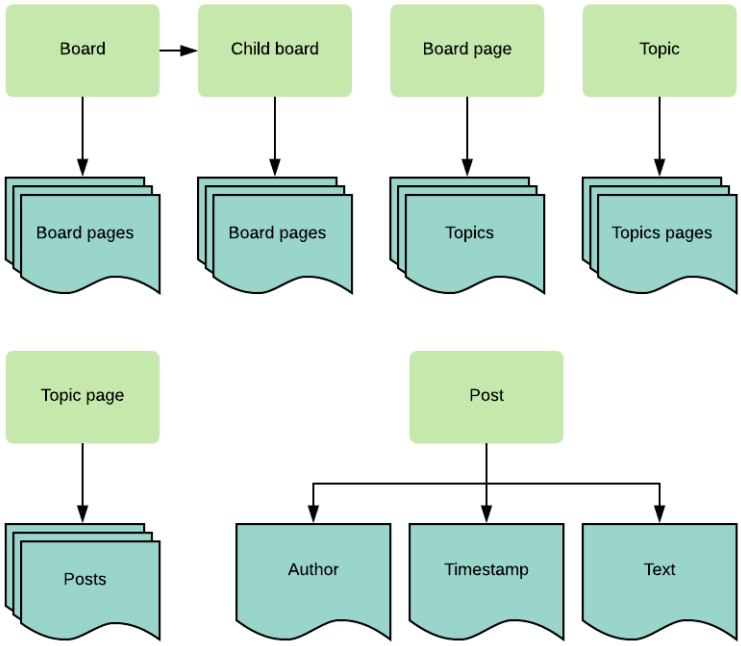

Figure 3.1: Bitcointalk structure

The structure of the website is described in figure 3.1. Every board, board page, topic, topic page and post is referenced by a number in the URL. All the numeric references can be calculated, which makes the parsing way easier as we don't need to crawl through the whole website.

Interesting information worth the extraction are crypto addresses in user posts together with author nickname, a timestamp and a link to the post itself. Crypto addresses can be found in user accounts as well, which might help with de-anonymization of the owner.

3.5.2 Bitcoinabuse.com

Bitcoinabuse.com is a web service that aims to prevent bitcoin cryptocurrency from being abused by criminals. The service maintains a public database of bitcoin addresses, reported by regular people.
Users can report one of these abuses:

- Ransomware
- Blackmail scam
- Sextortion
- Darknet market
- Bitcoin tumbler

The web service was launched in 2017[32] and since that every month, thousands of abuses are being reported. In summary, more than 90k reports have been uploaded.

[32] http://whois.domaintools.com/bitcoinabuse.com

Maintainers of this website introduce API with several endpoints. Therefore, every registered user can access the API using its own authentication token and call one of the endpoints[33]:

- Report Address
- Lookup abuse types
- Lookup distinct reports
- Check an address
- Download all reports

The most interested is the last one, which is capable of returning the whole database in CSV file. The same endpoint can also return the newest reports for the last day, which seems very useful for daily scraping.

An alternative way of parsing the website is to get recently reported addresses from *Lookup distinct reports* endpoint and then lookup for details for each of them using *Check an address* endpoint.

3.5.3 Bitinfocharts.com

Bitinfocharts.com was created in 2013[34]. It shows extensive information about a lot of cryptocurrencies. From current price, market cap, number of mined blocks, details about a transaction, up to advanced visualisations, correlations and mining calculations.

For the few most popular cryptocurrencies (Bitcoin, Bitcoin Cash, Litecoin, Bitcoin SV, Dash, Dogecoin, Bitcoin Gold, Reddcoin, Vertcoin, Namecoin, Peercoin, Blackcoin, Feathercoin, Novacoin), the website can display tables of *the richest*, *the busiest by transactions* and *the busiest by sum* addresses and their associated crypto wallets. From these tables one can extract enormous number of crypto addresses. The problem is that most of the wallets are anonymous and their name consists of some numeric identifier. Most of the identified wallets belong to cryptocurrency exchanges.
Even though only a fragment of the wallets available on this website is useful, it's still very good source worth scraping.

Data structure of this website is very simple. Every cryptocurrency has its own URL with simple pagination defined by number in it. For example Litecoin[35], where *50* is a number of pages, that can be used to simplify navigation through all the pages from 1 to 100.

3.5.4 Walletexplorer.com

The main purpose of walletexplorer.com is to connect Bitcoin addresses with specific wallets. It was created in 2014[36] and contains millions of crypto addresses. Each wallet is also matched with one of these categories: Exchange, Pools, Services, Gambling and Outdated.

Navigation through the website is fairly simple. Every wallet has numeric pagination in its URL so the page URLs can be easily generated and all the crypto addresses extracted.

[33] https://www.bitcoinabuse.com/api-docs
[34] http://whois.domaintools.com/bitinfocharts.com
[35] https://bitinfocharts.com/top-100-richest-litecoin-addresses-50.html
[36] http://whois.domaintools.com/walletexplorer.com

Data on this website can also be used for matching a wallet owner with one of the categories mentioned above.

3.6 Chapter Summary

In this chapter, we read about the problem of frequent website content changes and how is web parsing solving it. What difficulties we face when doing automatic data extractions from web pages and how parsing of websites with cryptocurrency content differs from the general websites.

We also addressed some useful technologies that help us with the parsing, namely Goutte, Scrapy and Lemmit for web archiving. There was also mentioned the difference between parsing and API usage, their limitations and advantages.

Finally, we read about particular cryptocurrency websites that focus on discussions about bitcoin technologies, reporting abuses using bitcoin, displaying exhausting information about lots of cryptocurrencies and lastly about a website aiming to detect bitcoin wallets and matching addresses. For each of the website, there was explained how important the extracted data are and how they can be used in the platform.

Chapter 4

Designing Data Collection Platform

In this chapter, we go through designing process of backend part of the platform. We cover web scraping challenges solutions, deployment in DigitalOcean cloud, concept of microservice architecture with interface contracts and how it supports modularity of a system. The chapter also contains diagrams with database scheme and architecture overview, that helps with understanding of the platform modularity.

4.1 Solving Web Parsing Challenges

In Chapter 3, section *Web parsing challenges* 3.2.1, we talked about several difficulties we have to reckon with when doing web scraping. In this section, we introduce solutions for each of them.

4.1.1 Unique Browser Fingerprints

The easiest way how to deal with browser fingerprinting is to change IP address and *User-Agent*[1] header regularly. Changing IP address will be covered in next section. For purpose of changing User-Agent header dynamically, we could use project *UserAgentGenerator*[2] or *uagent*[3]. Both of them should allow to generate the header with lots of variable properties very smoothly. We will test both of them and pick the best one. There is also an interesting project written in JavaScript, that is also generating User-Agent, but perform additional heuristics and picks only the most favourite values according to real-world usage[4].

4.1.2 Cloud Capabilities Overview

Using cloud technologies instead of running a program on local server has following benefits: *Efficiency/cost reduction*, *Data security*, *Scalability*, *Mobility*, *Disaster recovery*, *Control*, *Competitive edge*[5].

[1] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent
[2] https://github.com/phpfail/UserAgentGenerator
[3] https://github.com/miglen/uagent
[4] https://github.com/intoli/user-agents
[5] https://www.globaldots.com/blog/cloud-computing-benefits

For this thesis, the most important benefit is Scalability. It allows us to increase performance of an environment, where our service runs, easily without changes of the hardware. There are two ways how cloud technologies provide this ability. By *horizontal scaling*, that adds more processing nodes to the environment, or *vertical scaling*, that resizes existing configuration[6].

We can gain a massive advantage from horizontal scaling when solving a blocking of our crawlers by browser fingerprint detection. Imagine, we will run multiple nodes with the same code, but the only difference will be URL of the scraping web page, User-Agent header and IP address. This solution should give us outstanding performance results without hitting website rate-limits.

Another solution is to use proxy servers and send the scraping requests thought them. There is a lot of free proxy servers[7,8], but availability and reliability of those servers are questionable. To use this free-proxy approach, we would need to implement additional proxy monitoring and switch them regularly when they stop responding. An alternative is to use tools like *Tinyproxy*[9] or *Luminati*[10] which can get very pricey though.

4.1.3 DigitalOcean Advantages

DigitalOcean is very developer-friendly cloud provider and allows to setup instances with ease. It is also affordable due to pricing plans starting at $5 per month or $0.007 per hour and produces decent performance. DigitalOcean does not have such a big portfolio of products as Amazon Web Service of Google Cloud Platform provide but it is not even necessary for this project. One of the most valuable advantages is developer experience supported by numerous tutorials it presents.

The most attractive tutorials[11] for this project published by DigitalOcean are in the following list:

- How to Create a Cluster of Docker Containers with Docker Swarm on Ubuntu
- How To Install Apache Kafka on Ubuntu
- How To Install and Use PostgreSQL on Ubuntu
- Monitoring for Distributed and Microservices Deployments
- How To Deploy a Laravel Application with Nginx on Ubuntu

4.2 The Platform Modularity

This section is about principles, we decided to use, to support modularity and simplify maintenance of the platform. There are listed several technologies and patterns, that help us to achieve clear architecture design and enhance its understanding. In this section, there is also present the architecture chart 4.2.

[6] https://www.stratoscale.com/blog/cloud/scalability-cloud-organizations-win-cloud
[7] https://free-proxy-list.net
[8] http://free-proxy.cz
[9] tinyproxy.github.io
[10] luminati.io
[11] https://www.digitalocean.com/community/tutorials

4.2.1 Microservice Architecture

Microservice architecture [17] is a design pattern for building *highly maintainable and testable* software, that can be *developed by a small team* and includes *independently deployable* and *loosely coupled* modules. The pattern is suitable for complex applications [29].

The core characteristic of this pattern is the usage of several small modules, each doing the only thing and doesn't know about other small pieces of the puzzle. This approach also introduced great *fault isolation* because failure in one service do not necessarily mean crashing the whole application. This pattern also reduces the monitoring complexity of such an application [29].

Comparison with Monolith

Monolith architecture is, on the other hand, the traditional design pattern and it is still the right choice for many applications. This pattern is simple to develop because the code is built "under one roof", and developers do not need to implement communication interfaces between distant parts of the system. It is also simple to deploy, as no complicated deployment configuration that is describing inter-service communication and dependency is needed. However, it has significant limitations.

As they grow, monolithic applications are difficult to understand, especially for new developers in the team. Such applications are slower to start as a lot of code and dependencies is necessary to handle. Testing of these systems can be difficult and slow as well. Every small change in the system requires new deploy as a whole, so it also takes a lot of time [30].

Docker Containerization

Docker is a containerization[12] platform for developing, deploying and running applications. Using containers, it delivers infrastructure and hardware-independent environment. In compare to *Virtual machines*, Docker containers virtualize the operating system instead of hardware. Docker provides a lot of existing images through Docker Hub[13], which enables developers to run containers from those images and reuse pre-build code in their application [13].

Because of the loosely coupled containers, Docker is the perfect solution for building microservice applications. Every container can be written in different programming language and use unique technology stack. The only necessary thing to secure is a universal application interface that enables communication between the containers. Using Docker technology, the platform, building in this thesis, can reuse results from other web scrapping thesis if their API is compliant.

4.2.2 Kubernetes vs. Docker Swarm

Managing and deploying applications with a lot of containers, described previously 4.2.1, can be challenging. To solve these issues and to orchestrate containers, projects like *Docker Swarm* or *Kubernetes* were created.

Kubernetes allows a developer to configure the environment with higher demands and complexity. It is more suitable for extensive production applications supported by larger

[12] https://www.docker.com/resources/what-container
[13] https://hub.docker.com

team of developers. Installation of Kubernetes requires serious planning and it may differ from OS to OS and also between cloud providers. Kubernetes comes with their CLI[14] called *kubectl*, whose knowledge is necessary for a successful installation. Even though this platform can be used to deploy Docker image[15], it is incompatible with Docker CLI.

On the other hand, Docker Swarm is Docker native orchestration tool. It is simple to install, deploy and integrate with Docker Compose, tool for running multi-container applications[16], and Docker API. Due to its lightweight installation, it's more comfortable to use and faster for deployment. For purposes of this thesis, Docker Swarm is sufficient and preferable option [28].

4.2.3 Service Communication Contract

There are tons of websites that contain interesting information about cryptocurrency addresses and their metadata. It is not the main goal of this thesis to cover all of them, rather to allow smooth attachment of new parsers written even in different languages, but using unified API. This section will cover the designing of the interface.

For communication between scraping containers and other microservices in the architecture, it is necessary to design robust communication interface. Having the API implemented separately in each microservice is very error-prone as the API changes and its consequences are not easy to recognise. That means simple HTTP calls are not sufficient, and a more advanced approach is expedient [20].

Protocol Buffers[17] and especially *gRPC*[18], combining this technology with *RPC*[19], is the solution for mentioned concerns. It empowers a developer to create specific data type definition of every message that will be sent in the architecture network. To start the communication over the network, every part of the system has to accept this contract. This approach strengthens the visibility of API changes, because a service with an outdated contract, and therefore invalid communication protocol, cannot be build and deployed [15].

gRPC is language-independent technology and enables to generate the interface methods from central contract storage automatically when changes appear. For inter-microservice communication is gRPC considered as probably the most viable solution [19]. An example of how such implementation can look like is on picture 4.1.

For this thesis, the technology could makes integration with other web scraping and archival projects, built under the hood of the *Integrated platform for analysis of digital data from security incidents*[20], much more accessible. PHP wrapper for gRPC[21] seems to be the perfect choice for this work as it has to be done in the language.

[14] Command Line Interface
[15] https://codeburst.io/getting-started-with-kubernetes-deploy-a-docker-container-with-kubernetes-in-5-minutes-eb4be0e96370
[16] https://docs.docker.com/compose
[17] https://developers.google.com/protocol-buffers
[18] https://grpc.io
[19] Remote Procedure Call
[20] https://www.fit.vut.cz/research/project/1063/.en
[21] https://www.grpc.io/docs/tutorials/basic/php

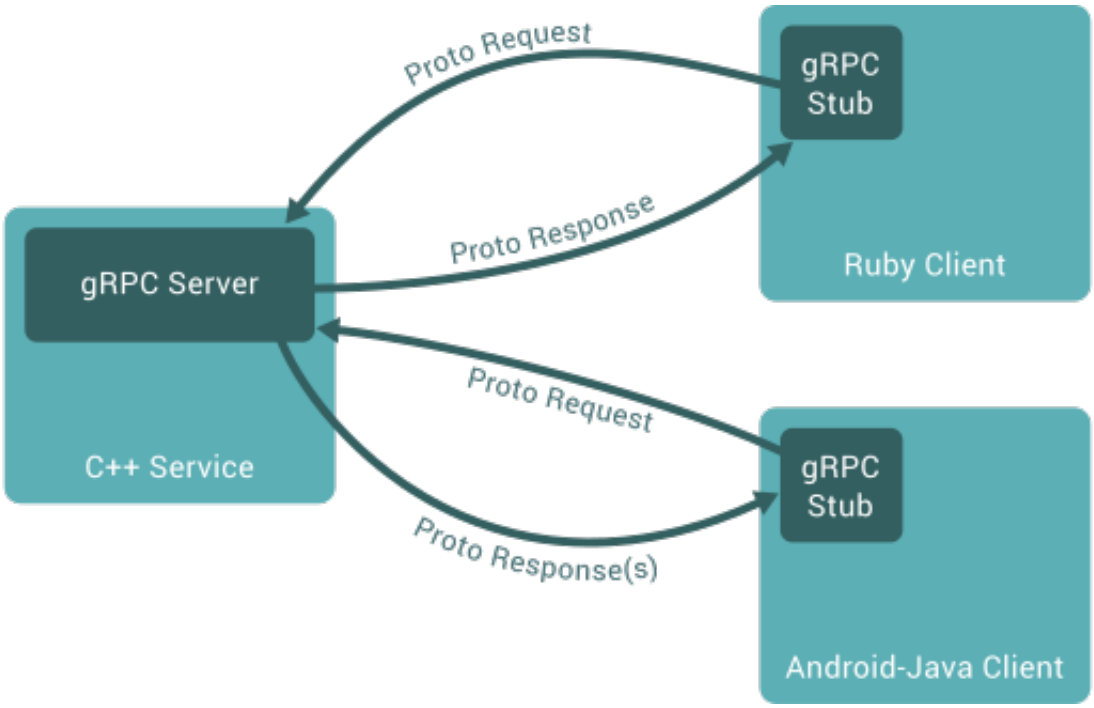

Figure 4.1: gRPC usage example [15]

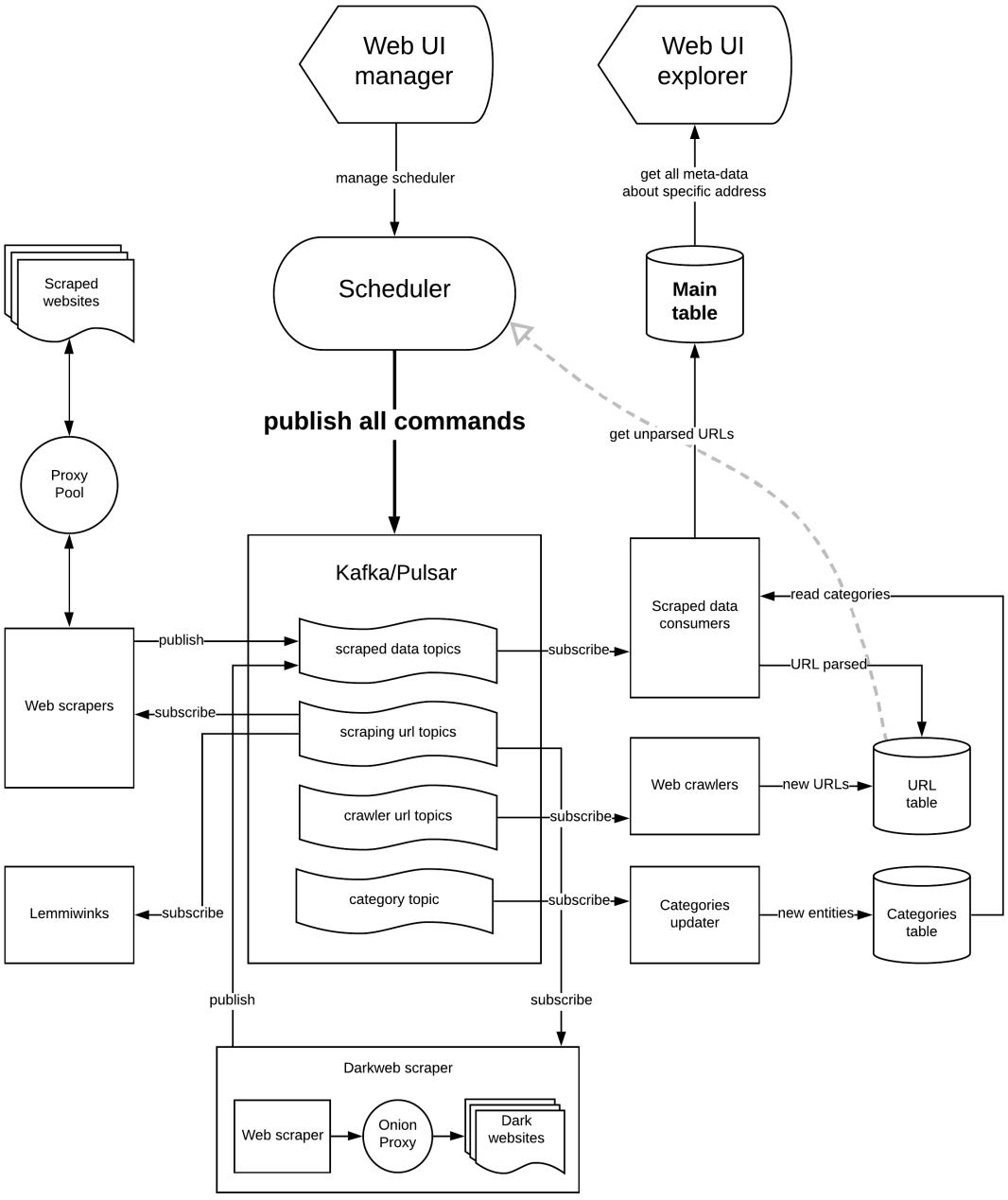

Figure 4.2: Platform architecture

4.3 Platform Data Layer Architecture

This section covers the platform architecture resume from extracted data point of view. It includes database scheme charts, data flow elucidation and clarification of used technologies.

4.3.1 Usage of PostgreSQL Database

PostgreSQL is an open-sourced, object-relation database engine with advanced SQL functions and capabilities. For use-cases, where multiple tables with possible dependency links to each other are required, relation databases are preferred. PostgreSQL is a reliable and extensible solution with proven architecture supporting robust data integrity. Behind the project, there is a wide open-source community, dedicated to delivering innovative features continuously. Therefore, PostgreSQL is the production-ready choice for many organizations[22].

In this thesis, PostgreSQL is used for following purposes:

- saving all extracted data and their meta-data in a unified structure;
- for meta-data about individual web pages;
- for additional information about cryptocurrencies and websites categorization.

Saving Web Crawling Results

Every website that suppose to be scraped is crawled at first, and all required URLs are extracted. Individual URL addresses are stored with additional meta-data. For example if a page has been parsed already or if a page is the last one from a sequence of pages. This meta-data can be used for running scrapers on appropriate pages or for another robots to know where to start a new round of crawling a website. On the picture 4.3 is an example of tables used for scraping *bitcointalk.org* forum.

[22] https://www.postgresql.org/about/

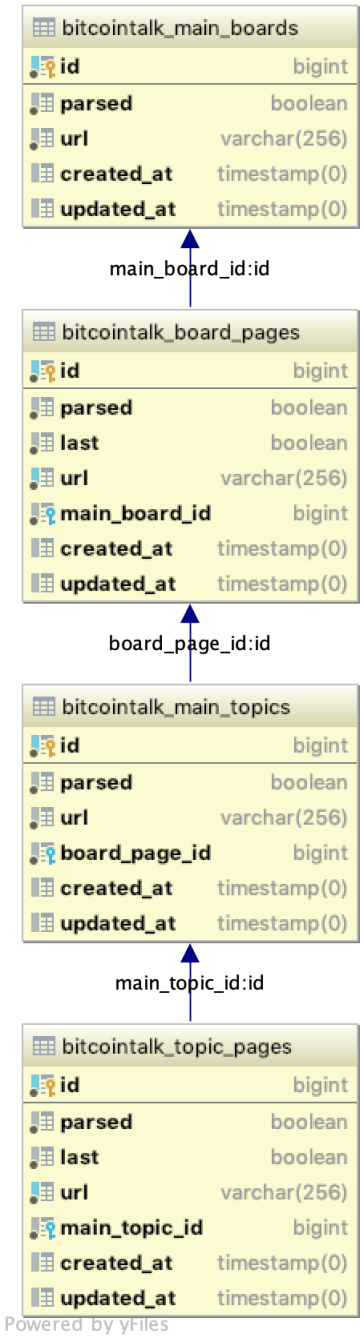

Figure 4.3: Web crawling tables dependency (generated from DataGrip)

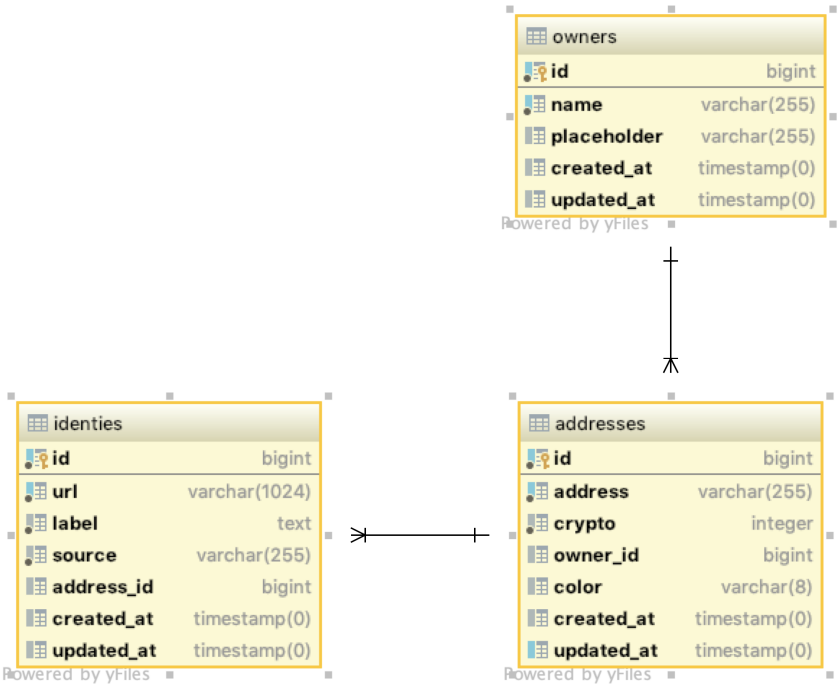

Figure 4.4: Main tables ER-diagram

Unified Database Schema for All Parsers

The tables relations can be seen on ER-diagram 4.4. Database schema for storing the scraped data consists of particular tables:

- owners - contains re-identified owners of crypto wallets;
- identities - there are stored meta-data about a page from where an owner and its address has been extracted;
- addresses - contains scraped crypto addresses and their meta-data.

4.3.2 Using Apache Kafka

Kafka is a distributed streaming platform, allowing to publish and subscribe to particular data flows. It is used for building reactive event driven applications and also for building data pipelines for reliable communication between systems. Kafka provides very fast and fault-tolerant data processing for which has been chosen in many enterprise solutions[23].

In compare to traditional message-queuing system, Kafka is capable to store messages persistently and doesn't delete them when they are read. That enables to reproduce whole sequence of events if needed and to read one message multiple times using different logic of data consumers. This technology combines two concepts, *queuing*[24] and *publish-subscribe*[25],

[23] https://kafka.apache.org/powered-by
[24] https://en.wikipedia.org/wiki/Message_queue
[25] https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern

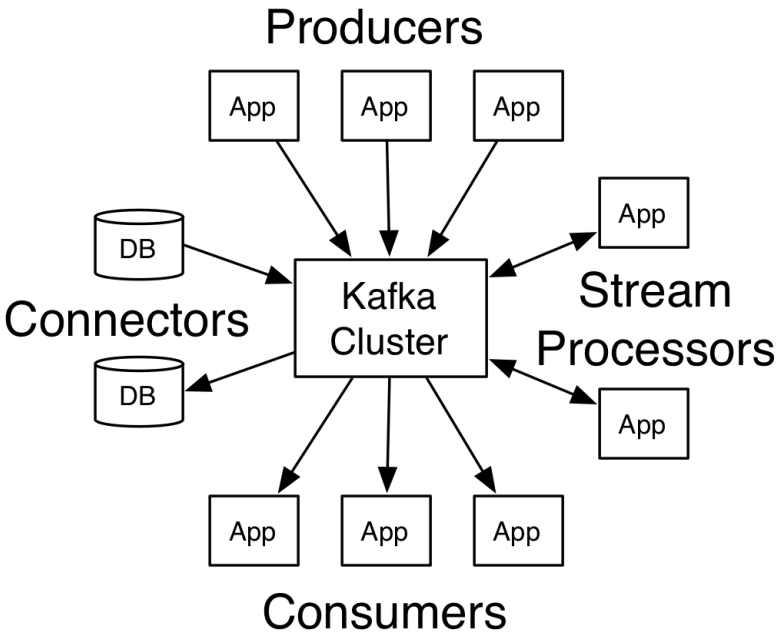

Figure 4.5: Apache Kafka APIs

and solves their individual issues using *Kafka Consumer Group* [34]. That means advantages from both models – *in-order delivery* and *parallel processing* – are merged together [1].

This data streaming platform can be operated as a cluster on multiple servers which makes it excellent choice for fault tolerance and data replication. One cluster stores streams of records aggregated into categories called *topics*. Each record has a *key*, a *value* and a *timestamp* [1].

Kafka exposes following main APIs[26]:

- Producer API – for publishing of messages into topics;
- Consumer API – for making subscriptions on multiple topics;
- Streams API – for building streaming processor that is subscribed on input topics and publishes into output topics;
- Connector API – for building reusable connectors for existing applications and services (e.g., to connect Kafka with database and stream data changes).

4.4 Scheduling of Task Execution

Laravel scheduler allows to create and manage Cron[27] tasks directly in code, without necessity of connecting to a server through SSH. Having the scheduler as a part of Laravel

[26] https://kafka.apache.org/documentation/api
[27] https://en.wikipedia.org/wiki/Cron

application provides excellent capabilities of connection with other Laravel tools[28]. For example *Artisan commands*, *Mail* or *Eloquent*. On the other hand, Laravel is capable to schedule also pure shell commands.

The API of this tool includes several human-friendly methods like: `everyMinute()`, `hourly()` or `weeklyOn()` and also method `cron()` for custom settings according to *crontab*[29]. It can run tasks in background which is useful for long-running processes. Another feature is *Task Hooks* that allows calling custom code before or after a scheduled task execution.

Where does this scheduler takes a place in the platform is shown on the figure 4.2. It executes numerous processes, for example:

- crawling websites to search for new URL addresses;
- starting scrapers by publishing URL addresses, that hasn't been parsed yet, into a Kafka topic;
- fetching data about crypto entities assigned to particular category.

4.5 Archiving Web Content

Lemmit has been already introduced in Chapter 3.3.5. It is a project for scraping whole web pages and storing raw DOM data for additional utilization. It can be easily added into the platform by subscribing to all topics containing URL addresses, waiting for scraping, using Kafka Consumer API.

This project can also be used as a middleware between Kafka topics and web scraping modules. Lemmit would store full DOM copy in advance and provides the copy to scrapers. This variant would solve issues with websites full of JavaScript. Lemmit already handles those cases, executes the JavaScript in a web browser and stores the DOM when it is everything loaded [22]. The pitfall of this variant could be performance issues as the described process is way complicated than pure web scraping. It might be solved by horizontal scaling of the instance though.

Another option is to use the combination of web scrapers and Lemmit only for websites, that uses a lot of JavaScript and traditional scraping is not efficient. This alternative seems the most beneficial.

4.6 Monitoring of The Scraping Processes

Monitoring of the final product is one of the most important steps in software development. Without proper real-time behaviour analysis and alerting system, it is nearly impossible to maintain and operate complex systems [40]. There are a lot of solutions available and massive competition between them because all those providers know how important monitoring is. One of the providers with plenty of customers and year-proven reliability[30] is *Datadog*.

Datadog is a great platform for monitoring every single piece of the platform, building in this thesis. It has an immense amount of integrations with a wide range of third-party

[28] https://laravel.com/docs/master
[29] http://man7.org/linux/man-pages/man5/crontab.5.html
[30] https://www.datadoghq.com/customers

solutions[31]. It has a straight forward installation through *Datadog Docker Agent*[32] that can be configured to simply collect all data and logs from other Dockerized services in a cluster. Datadog disposes of a lot of features, for example *Application Performance Management*, *Network Monitoring*, *Log Management*, *Metrics and Dashboards* or *Outlier and Anomaly Detection*[33].

Not all of the mentioned features are needed for this thesis, but some of them can be very useful in following use-cases[34]:

- Apache Kafka monitoring – Producer, Consumer, Topic statistics, Message rate;
- System metrics – CPU, Memory, Storage and Network usage;
- PostgreSQL DB – Sizes, Long-running queries, Deadlocks, IOPS;
- Network monitoring – can be used for monitoring of scraping proxy servers and their responses;
- Health checks – for tracking of running web scraper microservices;
- Dashboards with regular data-quality check results;
- Alerting when one of the metrics, generated in use-cases from above, hits a defined limit;
- Correlation between multiple metrics during an issue investigation.

Advanced features are paid, but for basic usage and all necessary integration is sufficient free plan.[35]

[31] https://www.datadoghq.com/product/integrations/all
[32] https://hub.docker.com/r/datadog/agent
[33] https://www.datadoghq.com/product
[34] https://docs.datadoghq.com/integrations
[35] https://www.datadoghq.com/pricing

4.7 Chapter Summary

In this chapter, we talked about how can we prevent from being blocked by websites when scraping their content and how cloud and proxy solutions can help us to achieve that. We introduced the advantages of microservice architecture and why it should be used for this thesis. There was also mentioned how can be the final platform deployed to a cloud provider and differences between Kubernetes and Docker Swarm. We explained how critical it is to focus on proper communication contract when designing robust application architecture and how can Protocol Buffers help with this goal. There was also presented a high-level platform architecture diagram, where in the centre lies Apache Kafka streaming solution, providing fault-tolerant communication between each module. For completeness of the diagram description, we mentioned data layer powered by PostgreSQL, Laravel Scheduler for planing the workflow and crucially important monitoring of the whole platform by Datadog.

Chapter 5

Building The Data Collection Platform

This chapter contains implementation details, chosen standards and used technologies. The content is related to both backend and frontend parts of this platform and also to all third-party projects that have been reused.

5.1 Technical Stack Overview

The platform is built in PHP 7.4. The language couldn't be chosen because of the dependency on the project [26]. On the other hand, the version number could be picked freely. The main reasons for choosing this variant were newly introduced features that have been partially taken from the functional paradigm or other modern languages. These features make development easier, helps to produce cleaner code and support type safety. The last-mentioned feature is essential when building software that aims to be used as a platform for other development in the future.

These are the discussed features:

- array spread operator;
- null coalescing assignment operator;
- typed class properties;
- arrow functions.

Both backend and frontend parts are built with Laravel framework, and everything runs in Docker containers. The Laravel container stack consists of two separate Docker images, the first for the frontend and the second for the backend part. Every scraping worker is implemented as the Laravel Artisan command while all of them use the same backend Docker image.

The entire Docker stack is composed of various pre-buílt images publicly available in Docker Hub[1] or Elastic Docker[2]:

- bitnami/zookeeper – Apache Zookeeper for managing Kafka brokers;

[1] https://hub.docker.com
[2] https://www.docker.elastic.co

- bitnami/kafka – Apache Kafka implementation;
- lensesio/lenses – Web-based monitoring tool for Apacke Kafka;
- postgres – PostgreSQL DB image;
- graylog/graylog – Web-based log management system for monitoring the platform behaviour;
- mongo – MongoDB image for Graylog configuration files;
- elasticsearch/elasticsearch-oss – Elasticsearch DB storing logs for Graylog tool.

The communication with the Kafka provides *php-rdkafka* client[3], and all the necessary requirements[4] are ensured by the backend image. Therefore, potentially fairly difficult Kafka installation on a developer machine is simplified and platform-independent development secured by the Docker.

The last module, but a crucial one, of the container ecosystem, is the proxy service using technologies Selenium[5], Tinyproxy[6] and HAProxy. The proxy module has another separated Docker image, inspired by project *docker-rotating-proxy* [32].

Error handling is provided by Sentry[7], and some custom warning and error logs are also sent into the Graylog service running locally.

5.2 The Final Platform Architecture

The architecture introduced in the section 4.2 of the Design chapter does not change, but not all the parts of the planned architecture have been implemented during this thesis. Some parts are meant to be added in the future.

The final architecture (depicted in Figure 5.1) consists of the following modules:

- Web crawlers – collect URL addresses from websites, store them into a database and shares them with other modules through Kafka topics.
- Web scrapers – parse web pages according to crawled URL addresses and extract interesting metadata. The data are streamed back to Kafka for further processing.
- Scrape consumer – consumes resulting metadata from all scrapers and stores them into the database with a unified schema.
- Proxy service – allows making HTTP requests to a single website from multiple IP addresses at the same time. It ensures application firewalls will not block the scrapers. The module is provided by the project *Automated Web Analysis and Archivation* [22].
- Apache Kafka with Zookeeper[8] – the core module of the entire system. It is a scalable, robust and fault-tolerant streaming platform that assures all the modules can communicate in a simple and unified way.

[3] https://github.com/arnaud-lb/php-rdkafka
[4] https://arnaud.le-blanc.net/php-rdkafka-doc/phpdoc/rdkafka.setup.html
[5] https://www.selenium.dev
[6] https://tinyproxy.github.io
[7] https://sentry.io
[8] https://zookeeper.apache.org

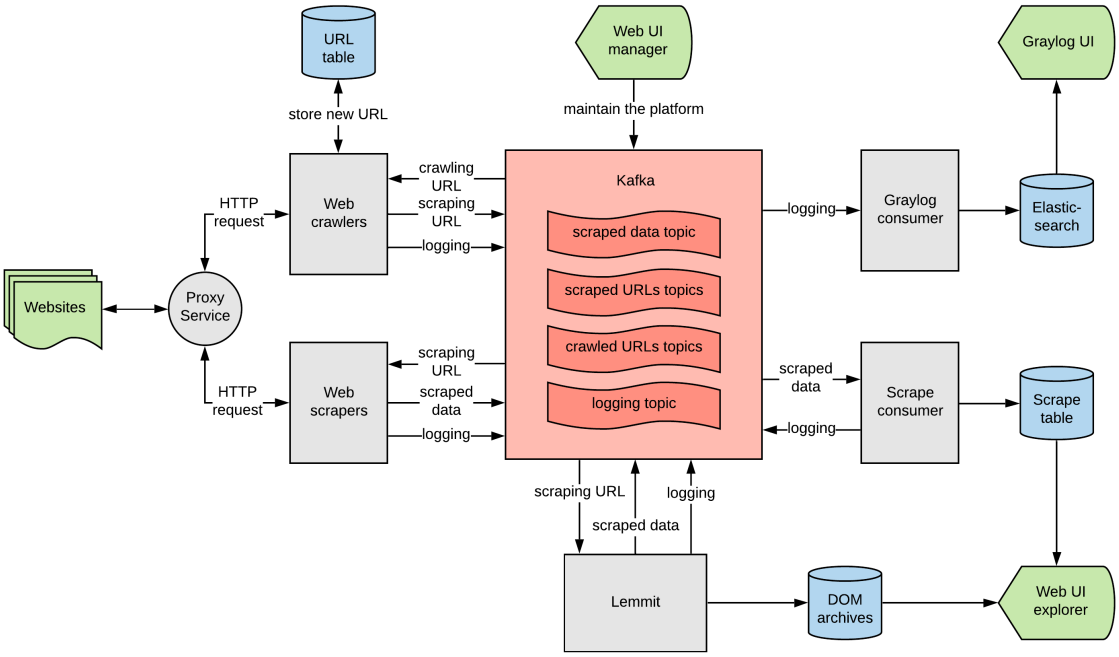

Figure 5.1: The platform architecture diagram. The blue elements are databases, the green are associated with a web browser, the grey are data-processing modules, and the red one is the core – Apache Kafka.

- PostgreSQL – stores the resulting scrapes and information about processing statuses of all websites in *URL table*, *DOM archives* and *Scrape table*.
- Graylog monitoring tool – web-based monitoring tool for most of parts of the platform.
- Lemmit – it gets URL addresses from Kafka topics so that archives whole DOM structure of a webpage. The DOM structure is used as evidence that data were present on a particular webpage at the time of scraping [22]. Integration of this module was not achieved in this thesis after all but for better understanding of the design capabilities, it is still present in the diagram.
- Web UI – allows an admin to manage the platform, to run parsing jobs manually and to schedule them. It also empowers a user to inspect the scraped data in order to see cryptocurrency activities that appeared outside of blockchain world. Every scraped information is linked to the proof mentioned above.

5.2.1 Scraping Workflow

This section explains the scraping workflow and clarifies the behaviour of the scraping parts. There are mentioned five essential modules the workflow consists of, and not all of them will be used for all websites the platform is going to parse. The *Web Crawler* and the *Web Scraper* could be sometimes single module if the use case requires it. The beauty of this architecture allows such changes without breaking dependency on other modules.

Proxy Service

The proxy consists of two service layers. The first layer is the smart Selenium proxy that returns fully rendered DOM even if the requested webpage contains JavaScript code and it also caches incoming requests from the web-crawlers/scrapers. This layer is provided by the project [22].

The second layer is connected to the Tinyproxy instances in Azure Cloud. The layer provides automatic IP address election or rotation using *HAProxy*[9]. The HAProxy can be configured to handle as many Tinyproxy instances as we have available and it implements several load balancing algorithms – we can choose from – to rotate IP addresses. The most known algorithm from the list[10] is probably *Round-robin* used in many other election-based applications.

Web Crawlers

Every web crawler of this platform takes a URL address from a Kafka topic and extracts all URL addresses from the webpage according to a specification. The crawlers differ in which type of webpage are taking from a topic and what URL addresses they are looking for there. The web crawler stores the extracted data into PostgreSQL and also shares them with other web-crawlers or scrapers using a specific Kafka topic. This last part can be simplified by using Kafka PostgreSQL Connector[11] which is capable to stream data from the database when some changes appear.

The Web Scrapers

The web scrapers accept a URL from specific Kafka topic and extract a crypto address metadata from the webpage. When the webpage scraping is done, a scraper marks the webpage as parsed and updates the database accordingly. The worker streams the metadata into an output Kafka topic when done.

Apache Kafka

The core of the entire platform is the Apache Kafka that mediates communication between all the related modules and ensure the exchanged messages arrive in all subscribed participants. There is a single topic for each message type, and each of them has several partitions. When the data producers stream messages into multiple partitions, they enable the possibility to read the topic by multiple consumers. This simple behaviour secures the parallel processing of the exchanged data.

Result Keeper

This module is subscribed to the Kafka topic that is common for all scrapers and contains the extracted crypto address metadata. The-keeper reads the data and stores it into the database. A scraper implementation can be language-independent because of this design. The common procedures can be moved into a single module, and scraping modules do not need to take care of it at all. Therefore, when a database schema changes happen,

[9] http://www.haproxy.org
[10] https://cbonte.github.io/haproxy-dconv/1.7/configuration.html4-balance
[11] https://debezium.io/documentation/reference/connectors/postgresql.html

or there will be a requirement to store the extracted data in a different database, the only necessary change will be in this result keeper module.

5.2.2 Benefits for Web Scraping

This section contains the justification of chosen design pattern and technologies and how web scraping gains advantage from this architecture. There are mentioned several benefits of the architecture and how are they related to the web scraping.

Tuning the Platform Scalability

The scraping part of the platform is separated into multiple small modules that comply with *Single-responsibility principle*[12]. The advantage over monolithic architecture is, among other things 4.2.1, fine-tuned scalability of particular modules. In the final state, where a huge amount of data sources is connected with the platform, there could be some modules unable to catch-up with the incoming data volume. The-incoming data would be stored in a Kafka topic which could grow over time. This issue is easily solved by scaling up the problematic module and therefore, increasing the processing bandwidth.

Language-independent Scrapers

There was already mentioned how Docker containers enable building language-independent modules 4.2.1, but is it has not been described why is it beneficial for this scraping-platform use case yet. Given the fact that there are many websites with interesting data from Open-source intelligence (OSINT) point of view, this platform needs to enable the smooth integration of parsers for particular websites. Programming complexity of a parser itself should not be demanding if a developer follows correct design patterns and the rest of the potential troubles are already solved by this platform. Therefore, those parsers could be implemented as results of school projects or theses, here at BUT FIT, and without the necessity to vendor-lock a programming language.

Reprocessing of Already Scraped Data

Among the advantages Apache Kafka brings to the architecture is the ability to reprocess data, stored in topics, by changing an offset from where should a consumer read a topic. The raw unchanged data are still present in the Kafka even when it is already read by a consumer. This is extremely valuable when it comes to dealing with errors during data processing.

A run-time error can always appear during the data transformation execution, but when it is fixed, we don't need to run the whole data flow from the start when we have all the middle steps saved in Kafka topics. The same approach applies for errors during the scraping part, and we can leverage from having the crawlers data backup as well.

Swapping A Module Implementation

The microservice design pattern, introduced in section 4.2.1, ensures a created module is loosely coupled and does not rely on implementation details of the others. This feature enables swapping a module implementation smoothly and without any breaking changes

[12] https://en.wikipedia.org/wiki/Single-responsibility_principle

if an API is the same. For example, we can introduce a new proxy service that will be rewritten into a different language to gain a performance boost in the future. That could be quite challenging with a monolithic architecture.

Fine-grained Monitoring

The importance of extensive monitoring has been already explained in section 4.6, and the platform architecture also takes it into account. The microservices are easier to monitor and analyze the run-time behaviour. This benefit is tightly connected with the *Tuning The Platform Scalability* 5.2.2 and only fine-grained monitoring can allow alerting when a consumer stops catching up with the data ingress and needs to scale up. Another monitoring-related necessity is also health checks, proving a module behaves according to its specification.

5.3 Implementation of The Scraping Modules

This section describes the implementation details of the platform scraping part. The content also introduces an application programming interface and class architecture of the scraping modules. The application interface allows reusing the core functionality and connecting additional data sources smoothly.

5.3.1 Namespace Dependency Diagram

Figure A.1 shows the hierarchical structure of PHP namespace dependencies the platform scraping part is build by. The Figure is generated using project *dephpend*[13]. The structure allows quickly extending the functionality and adding new data source parsers without any extra difficulties.

There was already mentioned that every scraping module is an Artisan Command 5.1. All the commands have to be placed in the namespace `App\Console\Commands` in order to be registered by the Laravel Artisan service. The commands namespace is composed of `Common` and `Bitcointalk` parts, but other namespaces are meant to be added for every additional data source. The common namespace covers all the commands that are not a data-source-dependent, for example, the one for storing scraped cryptocurrency metadata into a database.

The definition of all scraping commands inherits functionality from classes in the namespace `App\Console\Base`. This namespace contains data-source-related definitions as well as methods common for all modules, for example, Goutte-related 3.3.1 code.
The `KafkaClient` consists of core classes for communication with the Apache Kafka. There are three types of communication variants:

- Kafka Consumer – can read from a specified Kafka topic and perform some custom operations. For example, it can store data into a database.
- Kafka Producer – can read from a custom data source (a database for example) and write into a specified Kafka topic.
- Kafka ConProducer – the combination of both principles from above. In can read from an input topic and write into an output topic. The majority of the scraping modules inherits from this class as it is the most frequent use-case in this platform.

[13] https://github.com/mihaeu/dephpend

From the diagram A.1 and the previous paragraphs, it results, that the code has to deal with multiple inheritances because every scraping module is an Artisan command with a lot of extended functionality. PHP allows defining *Traits*[14] by which a developer can implement multiple inheritances, but on the other side, a trait cannot inherit from another trait.

This project contains several PHP Traits for simple decoration of a class. Every class that is meant to be used only for inheriting general methods is defined as an Abstract Class[15].

5.3.2 Bitcointalk Scraping Details

Bitcointalk.org forum is a static website, and the parsing can be easily separated into two stages, *crawling* and *scraping*. The stages can be executed simultaneously due to be designed architecture, where scraping workers process the crawled data when ready. The website parsing is performed by five crawlers and two scrapers that are depicted in the Figure 5.2.

The following list describes what each of the crawling workers does. All of them stores their results into different tables in the PostgreSQL database, but this can be simplified by using Confluent Kafka Connectors[16].

- *main_boards_crawler* – starts on the website index and searches for all URLs containing the string `?board=N.M`, where `N` is an unsigned number and `M` has to be a zero. The crawler then sends all the data into the topic `btalkMainBoards`. When the crawler finishes the process it recursively crawls the URLs it found before and searches for children boards. The children boards come into the same Kafka topic.
- *board_pages_crawler* – continuously reads the URLs produced by the previous crawler and searches for other URLs with `?board=N.M`, where both `N` and `M` are unsigned numbers. This crawler stores the results into `btalkBoardPages` Kafka topic.
- *main_topics_crawler* – continuously reads data from the `btalkBoardPages` topic and gathers URLs containing `?topic=N.M`, where `N` is an unsigned number and `M` has to be a zero. The crawler uses output topic `btalkMainTopics`.
- *topic_pages_crawler* – reads data from the `btalkBoardPages` topic and parses the content in the similar fashion as *board_pages_crawler* but the URLs have to contain `?topic=N.M`. The results are stored in `btalkTopicPages`.
- *user_profiles_crawler* – uses the `btalkTopicPages` Kafka topic as the input stream and focuses onto user profile URLs, which contains `action=profile` string, on a bitcointalk topic page. It stores the data into `btalkUserProfiles` topic.

[14] https://www.php.net/manual/en/language.oop5.traits.php
[15] https://www.php.net/manual/en/language.oop5.abstract.php
[16] https://www.confluent.io/connectors

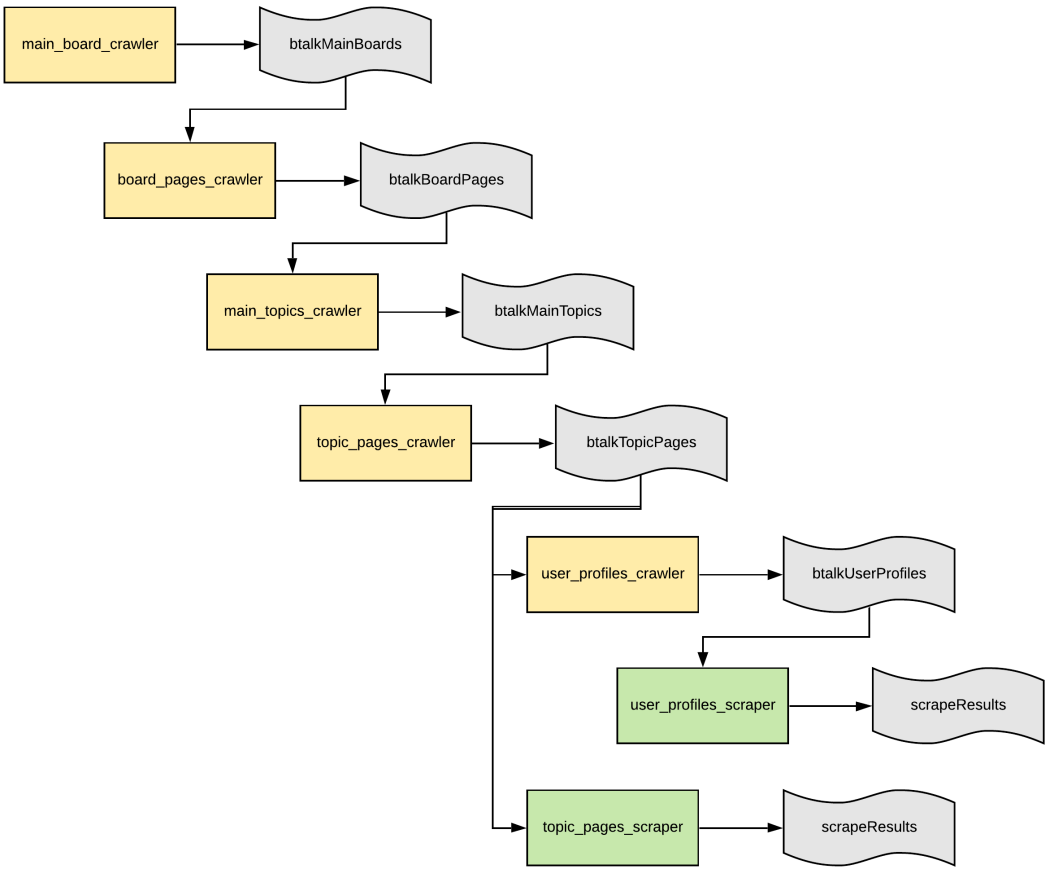

Figure 5.2: The diagram of bitcointalk.org website processing. The yellow rectangles mean web-crawlers, the green ones are web-scrapers, and the grey shapes illustrate Kafka topics. It may look like the process in synchronous, but it is needed to remember all the rectangles work asynchronously and can be scaled up separately.

The second group of workers is covered in the following list:

- *topic_pages_scraper* – reads the same topic as *user_profiles_crawler* does but using different Kafka Consumer Group 4.3.2, so the simultaneous reads do not collide. The scraper extracts crypto address metadata from the content and streams it into the topic `scrapeResults`.
- *user_profiles_scraper* – reads topic `btalkUserProfiles` and scrapes crypto addresses from the forum user profiles. The data are also streamed into the same output topic as *topic_pages_scraper* uses.

5.3.3 Bitcoinabuse Scraping Details

Getting data from *bitcoinabuse.com* is very simple as the service exposes API, which requires authentication token of a registered user. The API allows requesting the server resources once per two seconds, which is not very convenient for mining the data one by one. Luckily, *bitcoinabuse.com* provides the possibility to download whole their database by one request.

There is a list of API endpoints[17] interesting for our use-case:

- `/api/reports/distinct` – it allows crawling the reports one by one. Not much useful for us.
- `/api/download` – enables downloading whole *bitcoinabuse.com* database or its subset based on query parameter `time_period`. The parameter accepts the following values: `1d` `30d` or `forever`.

The scraping consists only from a single module that is requesting one-day-old data every day. The downloaded data are stored as a temporary CSV (Comma-separated values) file that is read afterwards. The bitcoinabuse module makes the transformation of the data into the unified structure and puts the transformed data into the topic `scrapeResults`.

5.3.4 Adding New Data Source

There are so many ways how a website can be organized that it is nearly impossible to define a single integration process for all the potential sources. Therefore, the architecture has to be flexible enough to cover all the use cases that can appear. The significant differences can be seen from the previous sections about Bitcointalk 5.3.2 and Bitcoinabuse 5.3.3 scraping but even though both websites were easily integrated.

The integration process can be separated into the following steps:

- Inheriting functionality from the platform core class `CryptoParser`. All the new source-specific classes and constants should use namespaces corresponding to the current order A.1.
- Using the Kafka traits from `KafkaClient` introduced in section 5.3.1. Some sources will need only one trait, and some will get the use of all three traits.

[17] https://www.bitcoinabuse.com/api-docs

- Creating DB models with the help of `BitcointalkModel` in `App\Models\Pg\<source>` namespace if needed. Again, no general model for all the use cases can be defined.
- Adding new docker-compose file with the definition of new workers into `docker/dev` folder. The services, described in the file, should have mounted volume into the host machine for easier development.
- Putting new entry into `TASKS` array in `Task.php` if a new worker should be scheduled from the web-based application.

There are two types of messages in the Kafka topic currently. Both of them are defined in `App\Models\Kafka` namespace. The first one, the `ParsedAddress`, is used to store a crypto address metadata into the `scrapeResults` Kafka topic. The `UrlMessage` allows storing crawled URL addresses into source-specific Kafka topics.

5.4 The Platform Monitoring

The monitoring of the platform behaviour supposed to be achieved by Datadog that was introduced in the section 4.6 of the platform designing chapter. We came across some difficulties in integration with the free version of the monitoring service. Therefore, we have chosen an open-source alternative, Graylog[18].

Graylog tool is used for overseeing the entire platform. All scraping modules stream logs into this tool. Graylog uses Elasticsearch DB to store the logs and MongoDB for managing configuration files. The tool is capable of displaying metrics generated from logs, creating alerts, dashboards, investigating log streams and has many other features that are useful for monitoring such complex systems. Figure 5.3 shows the *bitcointalk.org* monitoring dashboard with graphs from multiple scraping modules. The modules produce the following notable metrics:

- stored – number of records stored into PostgreSQL database;
- produced – number of messages streamed into Kafka output topic;
- consumer – number of messages consumed from Kafka input topic;
- warning – number of connection warnings, in this case, it is when the scraper hits rate-limits of a particular website.

[18] https://www.graylog.org/products/open-source

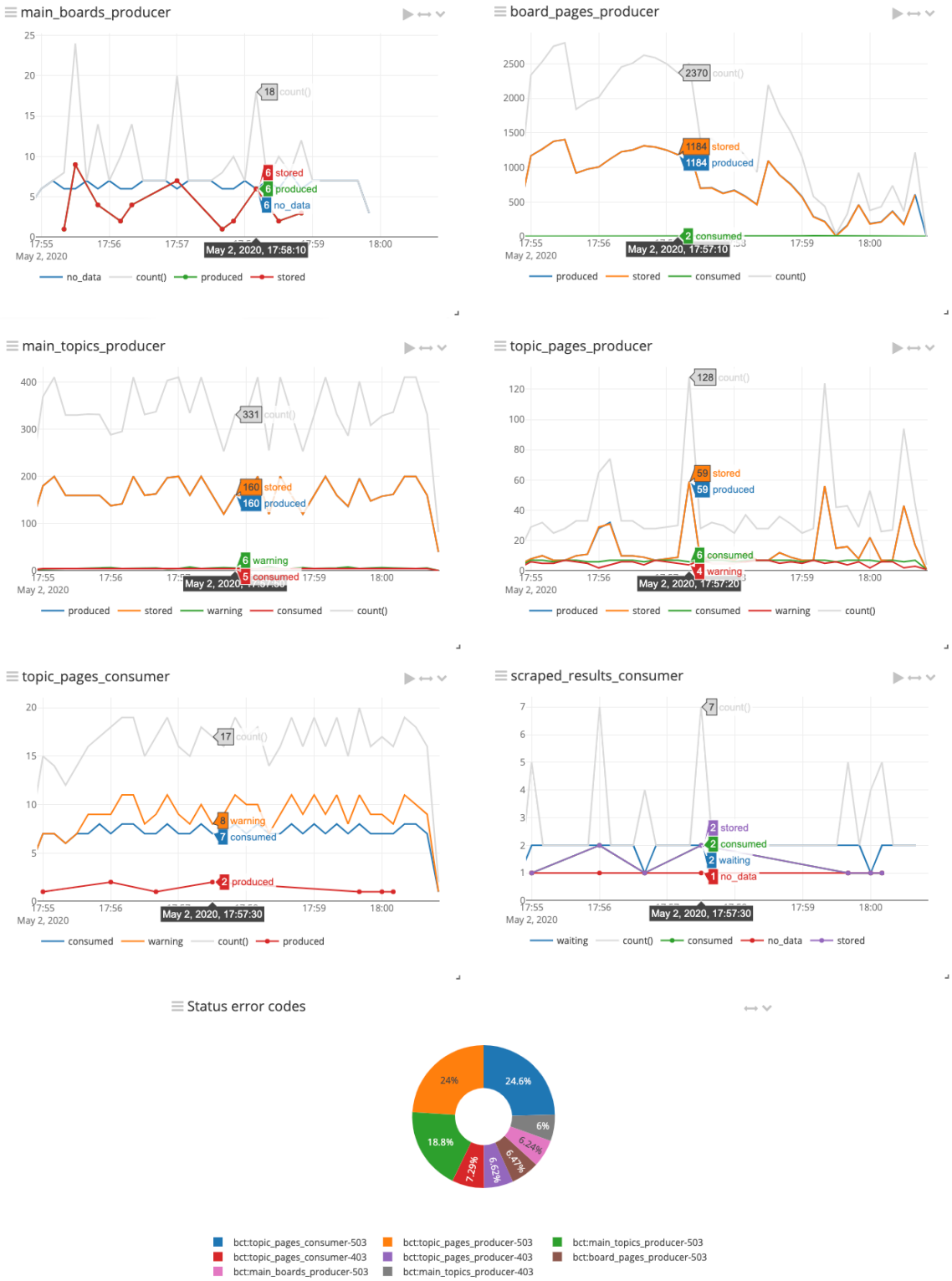

Figure 5.3: Bitcointalk.org monitoring dashboard showing behaviour of several crawling/scraping modules and pie chart with network errors that appeared during the website parsing.

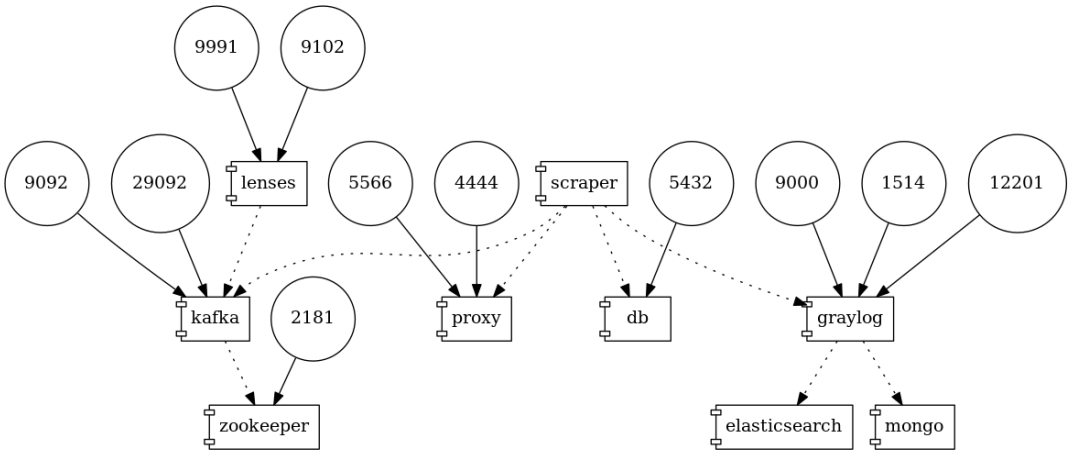

Figure 5.4: The diagram of docker-compose configuration shows dependencies between particular services and exposed ports.

5.5 Platform Modularity

In this work, the Laravel framework is employed for web scraping, communication with the PostgreSQL database and for building CLI commands that manage separate modules. Scheduling features in the platform are implemented by Laravel Scheduler.

Even though a single programming language is used almost for the whole platform, it does not mean all future extensions has to be written in it. The idea is that all core modules are language independent and communicate through a unified API. It means a specific web scraper module can be implemented by any programming language. If the module meets the API requirements, it can easily reuse common modules. This principle allows fast prototyping of new scraping modules without the need to understand the complexity of the entire system. The problematic part of website scraping is fully covered by this work, and the more straightforward parts can be added seamlessly.

Platform modularity is powered by Docker containers. The modules are divided into several categories, that can be seen in the platform diagram 5.1:

- Web crawlers/scrapers;
- Apache Kafka;
- PostgreSQL database;
- Graylog with supplementary databases;
- Web server powering the web application;
- Proxy server.

Every mentioned category runs in a separate Docker container, and in the case of Web crawlers/scrapers, it is expected to have tens of containers running simultaneously. All the modules are managed by Docker Compose that allows defining container dependencies,

internal network communication and many other properties[19]. The Docker Compose setup is illustrated in the Figure 5.4. There are also additional Docker containers for maintenance and development purposes, in the Docker Compose setup, but they are not important for the understanding of the platform modularity.

5.6 Executing The Platform Processes

This section guides how to maintain the platform infrastructure, where do the monitoring tools run when started and how can be the scraping modules executed. The Figure 5.4 can be helpful to understand the execution flow in this section.

5.6.1 The Infrastructure Maintenance

To execute the core of the platform, we need to run `docker-compose -f infra.yml up` in the `/docker/prod` folder. The command will start following Docker containers:

- Kafka + Zookeeper;
- Proxy service;
- PostgreSQL, Mongo and Elasticsearch databases;
- Lenses service;
- Graylog service.

For stopping a specific infrastructure container, it is needed to execute command

```
docker-compose -f infra.yml stop <container-name> (kafka|graylog|etc)
```

or we can skip the `container-name` to stop the entire platform core.

5.6.2 Starting Bitcointalk Scrapers

A Docker Compose file `bitcointalk-base.yml` is a place where the bitcointalk modules are configured. We can turn on the whole scraping flow by executing a single command

```
docker-compose -f infra.yml -f bitcointalk-base.yml up -d
```

or we can specify the `container-name` as well. The scaling of a specific worker is accomplished by adding `--scale <container-name>=N` where the `N` defines how many replicas should be started. So, for example:

```
docker-compose up -d --scale <container-name>=5 <container-name>
```

5.6.3 Starting Bitcoinabuse Scraper

A Docker Compose file `bitcoinabuse-base.yml` contains definitions of services that extracts one or thirty days old data or the whole database using *bitcoinabuse.com* API. Executing this command will start the scraping process, where `<t_period>` illustrates the type of the time period:

```
docker-compose -f infra.yml -f bitcoinabuse-base.yml up -d <t_period>
```

The command can be scheduled by the created web-based application using Laravel Scheduler 4.4.

[19] https://docs.docker.com/compose

5.7 Searching In The Results

This section introduces a web-based application that is capable of searching in the scraped metadata and scheduling repetitive scraping jobs.

The described platform is capable of generating a massive amount of cryptocurrency metadata. Only an intuitive search engine with a friendly user interface can maximize the data usage though. A simple web-based application has been implemented in PHP as a part of the work, to reach the maximal potential of the scraped data.

The web search engine has three major use-cases where a user can search for the following information and receive these properties:

- a cryptocurrency address:
  - category – Exchange, Mining pool, Person, Scam, etc.;
  - currency – BTC, LTC, ETH and others;
  - owner – an identified internet entity;
  - references – what web pages contain the address;
  - timestamps – when was the address scraped for the first time and when was updated at the last time;
- a source of a scraped data:
  - URL – web address of the source;
  - type – Web forum, Social network, Abuse report tool etc.;
  - addresses – which addresses were scraped from the source;
- an owner or wallet:
  - category – Exchange, Person etc.;
  - sources – which websites contain a mention of the owner;
  - addresses – list of addresses assigned to the wallet.

Figure 5.5 shows screenshots of the implemented web-based search engine. The engine displays result from search by a bitcoin address. The result depicts a user `zielar` and on which web page was its address seen. Every user activity is associated with a DOM copy from the time of scraping, so a user of this application can see the proof from where was the information taken. The DOM copies will be provided by archival capabilities of the integrated project [22].

5.8 Chapter Summary

This chapter described implementation details and the process of building the platform designed in the previous Chapter 4. There were introduced employed technologies and how they are connected with the platform using Docker containers. The benefits of this architecture have been explained on concrete use cases related to web scraping. We also talked about implementation details of the platform scraping part and how can be added new data sources into the data flow. At the end of the chapter, there was presented the web-based application for searching in the scraped data. The application has several use cases that were analyzed, and one of them was illustrated by GUI screenshots.

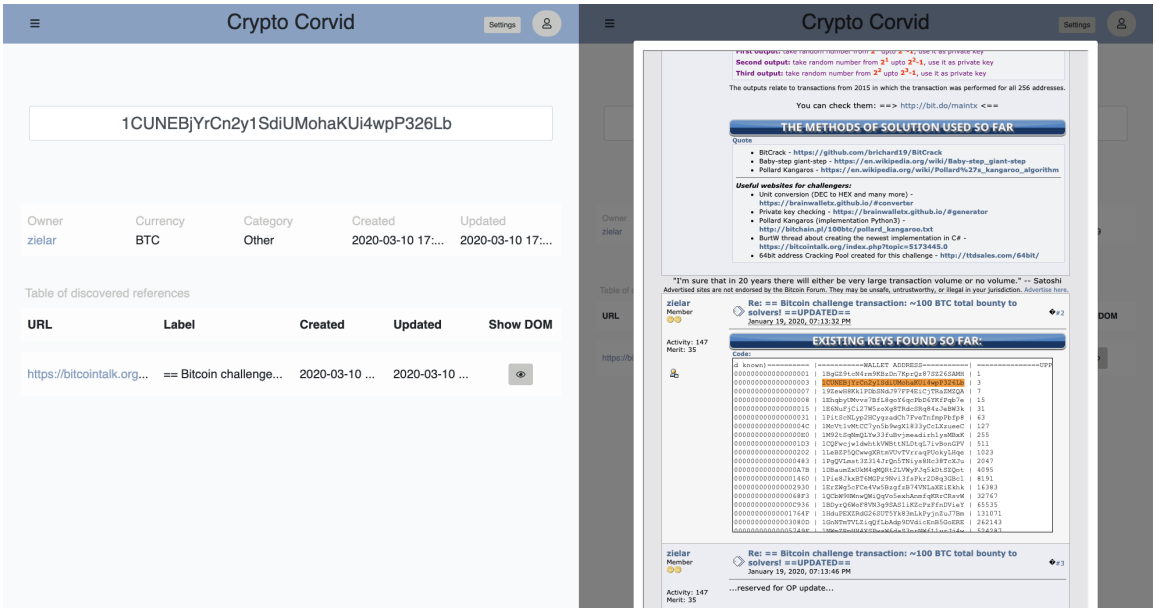

Figure 5.5: On the left screenshot, there are metadata as a result of a cryptocurrency address lookup. The right side shows a copy of a DOM associated with the searching address. The DOM copy is displayed after clicking on the „Show DOM“ button.

Chapter 6

Platform Verification and Discussion of Results

This chapter describes several ways of how the platform is evaluated and possible testing improvements that might be implemented in the future.

When the infrastructure part of the platform is executed, it takes around one minute (on an average personal computer) to get the core functionality prepared for operation. The core functionality includes the data streaming module with two monitoring tools and databases. After that, the scraping/crawling modules can be triggered. Due to the event-based architecture, the scraping modules are independent on the crawlers and are just waiting in a loop until a message appears in their input data-stream.

6.1 Deployment to The Remote Server

The platform is meant to run permanently or at least for several days per a new data source. Therefore, it is necessary to deploy it to a dedicated server.

6.1.1 Production Environment Setup

The dedicated virtual server has the following hardware configuration:

- 10x Intel Xeon Silver 4210 2.20GHz
- 64GB RAM
- 1.2TB SSD

The HAProxy service, described in the previous section 5.1, has been connected to five Azure Cloud instances running the Tinyproxy HTTP/HTTPS proxy daemon, creating up to five times performance boost.

6.1.2 Findings During The Deployment

The communication with the infrastructure and monitoring services – Kafka, database, proxy server, Web application, Lenses – from the local machine can be done by SOCKS5 SSH Tunnel after running `ssh -D <local_port> -N <virtual_server>`. The created tunnel can be then set in a web browser as a proxy.

The deployment process required several changes in the docker-compose configuration files. New folder `docker/prod` for production docker-compose files has been created. The services described in the production configuration files use pre-built Docker images and do not have mounted volume into the host machine.

After running the scraping process for multiple days, it also detected a few issues in the handling of unexpected run-time errors and high memory consumption. The problem with memory drain started with increasing of data in the database. The modules were fetching all records in one table, which required more and more hardware resources. Connecting of profiling tool *blackfire.io*[1] with the modules helped with detection of the issue, and the further fix was straightforward.

The scraping process is accompanied by regular network errors that might be related to a temporary website overloading and therefore hitting the rate limit. Another failure case is connected to the proxy behaviour. The proxies are most of the time stable but have some hiccups from time to time. On the other hand, none of the mentioned issues is unexpected and does not create any obstacles. The URL addresses that could not be parsed are still stored in the DB with a mark `parsed = false`. Therefore, once the first iteration is completed, we can restore the failed URLs into the Kafka and process them again.

The used Laravel Scheduler appeared like a not very reliable solution when working with Docker containers. When it runs inside of a container, it needs to have access to the host environment to execute other containers. This implicates contradictory thinking of Docker container – that should run without knowing about each other.

6.2 Checking The Processed Data

One of the ways how can we verify the platform works as expected is to check if some data are lost in the process. There are several touchpoints where we can measure the data discrepancies.

6.2.1 Kafka vs. Database Comparison

Every crawler stores URL metadata into a database and streams them into Kafka topic simultaneously. This is the first touchpoint where we can verify the behaviour. First of all, we know the numbers from the monitoring 5.3, but we can also perform SQL query in the database as well as in the Kafka, so all the numbers have to match for a specific time–topic–crawler combination. After performing this, we can be sure the data are not lost during the process.

We can get total number of produced messages from one scraper by this Graylog query on a specific period of time:

```
source:"bct:user_profiles_consumer" AND logType:produced
```

The following query checks total number of consumed messages:

```
source:"scraped_results_consumer" AND logType:consumed
```

In Lenses.io SQL studio[2], we can perform this type of validation query, which validates data stored in the `scrapeResults` Kafka topic:

[1] https://blackfire.io
[2] https://docs.lenses.io/sql/index.html

```
SELECT *
FROM scrapeResults
WHERE _meta.timestamp > '2020-05-20 10:00:00';
```

The validation of the stored data in the database can be done by this query, for example:

```
SELECT count(*)
FROM identies
WHERE created_at > '2020-05-20 10:00:00';
```

When some numbers do not match, we can check out the Graylog again and find out if some error appeared during the storing process or if the data are already present in the database by the following query:

```
source:"scraped_results_consumer" AND (logType:error OR logType:exists)
```

6.2.2 Web Scraper Verification

The second touchpoint is verifying whether the scraped data are actually associated with the correct webpage. Currently, this is manual work and can be surely automated. We can use the web application, built on top of the platform, to search for a specific crypto address and click on the link, associated with the search result 5.5. The address has to be present on the page.

6.2.3 Reference Dataset Comparison

Due to the fact that some websites were scraped by other projects at FIT BUT in previous years, we can also compare the older data with currently scraped results.

Sadly, there is not high-quality dataset available for the scraping of the *bitcointalk.org* website – on which were all the data checks performed. The available dataset contains only 2.5 thousands of parsed crypto addresses, which may be caused by the wrong implementation of previously used scraper solution or some database issues.

6.2.4 Total Data Prediction

During the scraping process, it is very convenient to know how long will it approximately take to parse the whole website if we do not know it beforehand. For the rough estimation, is it sufficient to calculate averages from already parsed data and estimate the final counts from it. In the case of *bitcointalk.org* scraping, we can use SQL query described in appendix B.

6.2.5 Total Scraped Crypto Addresses

The platform successfully scraped 360,000 addresses from the *bitcointalk.org* web forum and 35,000 addresses from the *bitcoinabuse.com* web service. The counting query for those results is present in appendix C.

6.3 The Platform Performance Measurements

The platform performance is tested on the parsing of whole *bitcointalk.org* web forum. The performance measurements of the Apache Kafka platform itself is not part of this thesis, but it is covered by the study *Kafka versus rabbitmq* [12], for example.

6.3.1 Tuning The Modules Performance

Every scraping or crawling module can have different computational complexity, and it is very convenient to tune their processing separately. Therefore, having the same delay between sequential HTTP calls for all the modules does not make sense. The more complex modules should have a lower delay and very simple processes can wait for a higher period of time. Such a module delay tuning can be easily performed by changing `SCRAPER_TIMEOUT` environment in the docker-compose files, and it can bring the performance closer to the optimal numbers.

6.3.2 Parallelization of The Parsing Modules

There was already stated how can we start multiple replicas of a single module by defining the `--scale` in the section 5.6.2. The number of replicas is tightly bound to the number of partitions that make up a particular Kafka topic. The Apache Zookeeper handles the responsibility of assigning a topic partition to a Kafka consumer – parsing module in our case. In the ideal situation, the numbers are the same. When we have more partitions than consuming modules, the Zookeeper solves the distribution by assigning more partitions to a single module. On the other hand, having more consumers does not bring any additional value, and the spare consumers will wait in an endless loop. With this knowledge in mind, there were created ten partitions per topic in the Apache Kafka. This amount gives us a chance to increase the quantity of the proxy instances up to ten, but the number is not definite, and the scaling capabilities have only hardware limits.

6.3.3 The Test Case Details

The *bitcointalk.org* website consists of the pages in the first column of the following table. The table also contains numbers of data per each page type and estimated time for scraping without parallelization.

	Required HTTP requests
main boards	248
board pages	30,859
main topics	1,228,246
topic pages	3,100,334
user urls	3,100,334
user profiles	1,453,171
summary	8,913,192

Table 6.1: Bitcointalk.org pre-scraping analysis.

In total, the website scraping requires approximately 9 millions of HTTP requests. The website application firewall allows one request per seconds, which is in the worst-case scenario 2,476 hours of scraping or 103 days. Therefore, for real data usage, it is necessary to scale up the process.

When we look at the ratio between the total number of requests and the amount of scraped pages per type, we see `topic pages` require about 34% and `main topics` about 14% of the processing time. This is the information we need to take into account when setting the `SCRAPER_TIMEOUT` parameter for particular scraping/crawling modules.

To make the scraping process smooth and without necessary interventions, for example, changing the timeout parameter or manually adding more module replicas, we need to increase the delay for modules with lower scraped/total ratio and decrease it for parsers with high ratio.

We need to include also hardware requirements for each parsing module, to make the final timeout calculation as precise as possible. Previously introduced *blackfire.io* is perfect solution for such findings. The blackfire.io profiling found that crawling of `board pages` demands approximately two times more resources than the `topic page` or the `user profile` scrapers.

These results emerge from the timeout calculation:

	Delay in milliseconds			
Module name	Delay divider	One proxy	Five proxies	Nine proxies
main_board_producer	0.29425	23,789	4,758	2,643
board_pages_producer	1.29425	5,409	1,082	601
main_topics_producer	0.9437	7,417	1,483	824
topic_pages_producer	1.2968	5,398	1,080	600
user_profiles_producer	2.4597	2,846	569	316
topics_pages_consumer	2.4107	2,904	581	323
user_profiles_consumer	1.3004	5,383	1,077	598

Table 6.2: Bitcointalk.org scraping measurements.

The *Delay divider* is composed of the importance and the computational difficulty of a particular module. For example, the *topic_pages_producer* is very important as it needs to process over one million of HTTP calls, but it is less difficult than the *topics_pages_consumer*. Therefore the producer needs to wait a little bit more than the consumer when though it gathers data for the second module.

This table illustrates expected versus measured scraping times with and without the use of parallelization:

	Sequentially	Five to nine proxy servers
expected times	103 days	~16 days
measured times	not measured	~19 days

Table 6.3: Bitcointalk.org scraping measurements.

There were available five proxy servers to use for the platform at the start of the scraping. Then we increased the amount to nine servers. Therefore, the measurement is calculated using seven proxies in average.

The difference between *expected times* and *measured times* while using proxy servers is caused by required maintenance and on-demand process fixes. This attempt was not automated much and required assistance very often. Sadly, it took approximately 19 days to finish the attempt and another one would not be complete on time.

It is also necessary to note that the scraping cannot be performed on the one-request-per-second boundary because we would risk hitting the rate limit, potentially. The scraping performance varies page to page because the page size and amount of metadata differ as well. Every scraper runs at 90% of the maximal speed level, and therefore, it has enough space for expanding if necessary.

6.4 Chapter Summary

This chapter described the deployment process, its challenges and what flaws have been found when the parsing run for several days. There were mentioned data comparison procedures and verification touchpoints in the platform data flow.

We also discussed performance measurements, what was the test use case, what has to be taken into account during up-scaling of the parsing modules and how Apache Zookeeper makes the process parallelization simple. Overall, the chapter proves the capabilities of Apache Kafka and what a huge potential the platform has.

Chapter 7

Future Work

This chapter describes potential enhancements which did not fit into the scope of this thesis. There are available technology changes or using different approaches as well as expanding the feature stack of already used technologies.

7.1 Lemmit Integration

The integration of the Lemmit DOM archival module was not accomplished in this work, after all. This missing piece will be the first thing that is going to be implemented in the near future.

7.2 Confluent Kafka

Probably the most viable and exciting change would be using Confluent[1] which wraps Apache Kafka with a lot of awesome features. There are two variants available: *Confluent Cloud* and *Confluent Platform*. The first one is a hosted version and useful for business-critical applications where availability and technical support is highly required. This version is also pricey and not much needed for our use case. On the other hand, the second option brings the following features in an open-source package[2]:

- Connectors – Offers community connectors developed and supported by Confluent;
- Kafka Streams – Offers a simple library that enables streaming application development within the Kafka framework;
- ksqlDB – Build event streaming applications that use stream processing with a light-weight SQL syntax;
- Kafka Connect – Delivers an advanced API for connecting external sources and destinations into Kafka;
- Schema Registry – Offers a central registry for the format of Kafka data to guarantee compatibility;
- Clients – Supports Kafka integration using C++, Python, Go, and .NET;

[1] https://www.confluent.io
[2] https://www.confluent.io/downloadconfluent-platform

- REST Proxy – Provides universal access to Kafka from any network connected device via HTTP.

By using *Connectors* and *ksqlDB* would make data processing much simpler and lightweight as we would not need to support direct database connection from crawling/scraping modules. The whole data flow would be organized through Kafka, and the modules would gain additional performance.

Schema Registry, we could define strict communication protocol on top of Kafka topics, and all the misconfigured messages would be dropped. This feature would get usability and multilingual programming support to the next level.

7.3 Proper Monitoring and Logging Solution

Graylog is a very good open-source project and consists of several handy features. It was truly easy to set up the logging. For the initial development steps and for the scope of this thesis, the tool is incredibly beneficial. When we move this project further, we will need more complex solution for monitoring the entire platform including the Kafka (broker throughput, consumer group lags and other essential metrics), the proxy solution which is currently not monitored at all and also the database queries and performance. Also, the visualization capabilities are not great in Graylog. All of this would be quite difficult to accomplish with the current technology stack.

For the visualization purposes, we could connect *Grafana*[3] to Graylog or rebuild the logging using *ELK stack*[4]. There is also already mentioned *Datadog* in the section 4.6 that could cover all the previously illustrated use cases and which was initially designed as the logging/monitoring solution, but the free version did not allow the full adoption of the tool.

7.4 Various Testing Tactics

Honestly, this project is not very well tested from the unit of end-to-end point of view. Usage of the PHP version 7.4 leverages the typed class properties, mentioned in the section 5.1, and also enforcing `strict_types` declaration in all PHP files by pre-commit hooks makes the code more trustworthy. But none of it cannot fully substitute proper unit testing.

Testing of module integration into the platform is also necessary to automate. Currently, the measurement is performed manually, but with an increasing amount of modules, this situation can be more time-consuming.

The end-to-end testing, covering all aspects of the data flow, would be super advantageous in the future. It might be needed to implement a test suit where some changes happen in a mocked website, particular modules have to react accordingly, and correct data changes appear in the database.

Automated data quality checks are not covered in this thesis but can be surely marked as a significant factor in a validation of data reliability. For now, the quality of inserted data is handled by strict SQL constraints and also the validation of the scraped data in the application layer. The automated data quality checks could be done, for example, in *Apache Airflow*[5].

[3] https://grafana.com
[4] https://www.elastic.co
[5] https://airflow.apache.org

7.5 Extending The Feature Set

One of the projects we could utilize is *Sherlock*[6] which is able to search for specific nickname across various websites[7] and return potential matches with additional metadata. We could use it as a lookup table to enhance the metadata we scrape from different web services.

Already noted Apache Airflow could be used as an alternative to Laravel Scheduler because the used command scheduler showed its unfriendliness, mentioned in the section 6.1.2, when combined with Docker.

Another feature related to the web application, which did not fit into the scope of this thesis, is proper authentication of users. On the other hand, the application is not exposed to the public network, and the only connection to the virtual server is through SSH protocol. Adding this feature should be fairly simple because of the Laravel framework that has everything necessary already prepared. The only thing is to enable the authentication capabilities and to prepare a database for the users.

[6] https://github.com/sherlock-project/sherlock
[7] https://github.com/sherlock-project/sherlock/blob/master/sites.md

Chapter 8

Conclusion

The goal of this thesis was to implement a platform for collecting cryptocurrency addresses and web application for managing this platform. The platform meant to be highly modular with monitoring of each module and suppose to run in a cloud solution to utilize could-based scalability. The core of this platform should parse interesting web pages containing cryptocurrencies metadata and store the data into storage with a unified database scheme. The data should be extracted from publicly available sources according to the definition of OSINT.

The prerequisite for the implementation of the platform was to get acquainted with blockchain, cryptocurrencies and transactions and with how they are created. It was also necessary to review existing tools for website parsing and implement its own parsing solution using the OOP paradigm and PHP Laravel framework. The solution had to be platform-independent. The platform independency supposes to be achieved by Docker containerization and using loosely coupled modules communicating through the Apache Kafka streaming platform.

First of all, I have learned about all the mentioned topics necessary for understanding cryptocurrency challenges. I did research about website parsing and existing tools, which are Scrapy and Lemmit. I picked up several websites full of cryptocurrency addresses and designed how to parse each of them using as universal as possible principles.

I have successfully designed and implemented platform with the aim of modularity and easy addition of new webpage parsers. The whole platform composes from few Docker containers. The parsing core of this platform consists of several tasks that are scheduled from a web application, and all of them are monitored through Graylog service. The tasks are implemented by Laravel Artisan commands, and the platform runs on PostgreSQL database engine.

The processing pipeline outputs metadata about cryptocurrency addresses which have context outside of blockchain world. For the searching purposes, there was implemented web-based application. The gained information can support crypto criminal activities detection. This work solves most of the web parsing issues and enables seamless extensibility of scraping modules that can be implemented during networking courses at our faculty or as part of Bachelor's or Master's thesis.

The platform has been deployed to a dedicated server and tested on scraping of two web services. One of them, *bitcointalk.org* web forum, has been chosen as a test case for performance measurements. Sequential scraping of the website would take about 103 days, but the designed platform with seven proxy servers handled it in 19 days. The parsed data have been evaluated and also extensively monitored during the scraping process. In total,

the scraping platform generated 360,000 addresses from *bitcointalk.org* and 35,000 addresses from *bitcoinabuse.com* websites.

This work can be expanded by adding more parsing modules. It is also planned to integrate project *Platform for automated analysis and archiving of data from the web* built at Brno University of Technology and to connect all parts of the project [22] with the platform. The core functionality is published as open-source software. The results have been presented in the Excel@FIT 2020 conference in the form of an article. The article has received the price of the Expert board and also the Public award. The poster for the conference is available in appendix D. The author plans to continue with further development with this platform and already introduced possible improvements in this text.

Bibliography

[1] APACHE.ORG. *Kafka Introduction* [online]. apache.org [cit. 2020-01-01]. Available at: https://kafka.apache.org/intro.

[2] BADRETDINOV, T. *How to create a Bitcoin wallet address from a private key* [online]. freecodecamp.org [cit. 2020-01-01]. Available at: https://www.freecodecamp.org/news/how-to-create-a-bitcoin-wallet-address-from-a-private-key-eca3ddd9c05f.

[3] BADRETDINOV, T. *How to create an Ethereum wallet address from a private key* [online]. freecodecamp.org [cit. 2020-01-01]. Available at: https://www.freecodecamp.org/news/how-to-create-an-ethereum-wallet-address-from-a-private-key-ae72b0eee27b/.

[4] BITCOINCASH.ORG. *Address format for Bitcoin Cash* [online]. bitcoincash.org [cit. 2020-01-01]. Available at: https://www.bitcoincash.org/spec/cashaddr.html.

[5] BITCOINCASH.ORG. *Transaction Spec for Bitcoin Cash* [online]. bitcoincash.org [cit. 2020-01-01]. Available at: https://www.bitcoincash.org/spec/transaction.html.

[6] BITCOIN.IT. *Technical background of version 1 Bitcoin addresses* [online]. en.bitcoin.it [cit. 2020-01-01]. Available at: https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses.

[7] BITCOIN.ORG. *Transactions guide* [online]. bitcoin.org [cit. 2020-01-01]. Available at: https://bitcoin.org/en/transactions-guide.

[8] BUTERIN, V. and SANDE, A. *EIP-55* [online]. github.com [cit. 2020-01-01]. Available at: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-55.md.

[9] CODETRACT. *Inside an Ethereum transaction* [online]. medium.com [cit. 2020-01-01]. Available at: https://medium.com/@codetractio/inside-an-ethereum-transaction-fa94ffca912f.

[10] DATAHEN. *Data Harvesting War: Scraping vs using API* [online]. https://www.datahen.com/ [cit. 2020-01-01]. Available at: https://www.datahen.com/blog/data-harvesting-war-scraping-vs-using-api/.

[11] DISPARTE, D. *Crypto Crime Is Taking A Violent Turn* [online]. forbes.com [cit. 2020-01-01]. Available at: https://www.forbes.com/sites/dantedisparte/2019/01/28/crypto-crime-is-taking-a-violent-turn.

[12] DOBBELAERE, P. and ESMAILI, K. Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper. *ACM*

International Conference on Distributed and Event-based Systems. 2017, no. 11, p. 227–238.

[13] Docker. *Docker overview* [online]. docker.com [cit. 2020-01-01]. Available at: https://docs.docker.com/engine/docker-overview/.

[14] Dunglas, K. *Introducing Symfony Panther: a Browser Testing and Web Scrapping Library for PHP* [online]. symfony.com [cit. 2020-01-01]. Available at: https://symfony.com/blog/introducing-symfony-panther-a-browser-testing-and-web-scrapping-library-for-php.

[15] Google. *GRPC Guides* [online]. grpc.io [cit. 2020-01-01]. Available at: https://grpc.io/docs/guides/.

[16] Gugnani, T. *Crawling website using Laravel Dusk Spider* [online]. medium.com [cit. 2020-01-01]. Available at: https://medium.com/@tushargugnani_54389/crawling-website-using-laravel-dusk-spider-bbbbe487a21.

[17] Hasselbring, W. Microservices for Scalability: Keynote Talk Abstract. *ACM/SPEC on International Conference on Performance Engineering*. 2016, no. 7, p. 133–134.

[18] Hayes, D. B. *PHP Web Scraping: What to know before you start with Symfony Panther, Goutte, and more* [online]. thoughtfulcode.com [cit. 2020-01-01]. Available at: https://www.thoughtfulcode.com/php-web-scraping.

[19] Indrasiri, K. *Build Real-World Microservices with gRPC* [online]. thenewstack.io [cit. 2020-01-01]. Available at: https://thenewstack.io/build-real-world-microservices-with-grpc/.

[20] Irannejad, M. *GRPC in Microservices* [online]. gitconnected.com [cit. 2020-01-01]. Available at: https://levelup.gitconnected.com/grpc-in-microservices-5887caef195.

[21] Jane, M. M. *How to bypass anti-scraping techniques in web scraping* [online]. bigdata-madesimple.com [cit. 2020-01-01]. Available at: https://bigdata-madesimple.com/how-to-bypass-anti-scraping-techniques-in-web-scraping/.

[22] Kocman, T. *Automated Web Analysis and Archivation*. Brno, CZ, 2019. Master's thesis. Brno University of Technology, Faculty of Information Technology. Available at: https://www.vutbr.cz/en/students/final-thesis/detail/121925.

[23] litecoin.info. *Comparison between Litecoin and Bitcoin* [online]. litecoin.info [cit. 2020-01-01]. Available at: https://litecoin.info/index.php/Comparison_between_Litecoin_and_Bitcoin.

[24] litecoin.info. *Litecoin* [online]. litecoin.info [cit. 2020-01-01]. Available at: https://litecoin.info/index.php/Litecoin.

[25] Mappo, J. *Merkle Tree—What is it and why use it?* [online]. hackernoon.com [cit. 2020-01-01]. Available at: https://hackernoon.com/merkle-tree-what-is-it-and-why-use-it-8m2a63xjd.

[26] Matoušek, P. *Integrated platform for analysis of digital data from security incidents* [online]. fit.vut.cz [cit. 2020-01-01]. Available at: https://www.fit.vut.cz/research/project/1063/.en.

[27] Murthy, M. *Life Cycle of an Ethereum Transaction* [online]. medium.com [cit. 2020-01-01]. Available at: https://medium.com/blockchannel/life-cycle-of-an-ethereum-transaction-e5c66bae0f6e.

[28] Naser, H. *Kubernetes Vs. Docker Swarm* [online]. vexxhost.com [cit. 2020-01-01]. Available at: https://vexxhost.com/blog/kubernetes-vs-docker-swarm-containerization-platforms/.

[29] Richardson, C. *Pattern: Microservice Architecture* [online]. microservices.io [cit. 2020-01-01]. Available at: https://microservices.io/patterns/microservices.html.

[30] Richardson, C. *Pattern: Monolithic Architecture* [online]. microservices.io [cit. 2020-01-01]. Available at: https://microservices.io/patterns/monolithic.html.

[31] Rosic, A. *Proof of Work vs Proof of Stake* [online]. blockgeeks.com [cit. 2020-01-01]. Available at: https://blockgeeks.com/guides/proof-of-work-vs-proof-of-stake/.

[32] Rotich, B. *Docker-rotating-proxy* [online]. github.com [cit. 2020-01-01]. Available at: https://github.com/brians-bytes/docker-rotating-proxy.

[33] Singh, A. *How can we do Ethereum address validation?* [online]. quora.com [cit. 2020-01-01]. Available at: https://www.quora.com/How-can-we-do-Ethereum-address-validation.

[34] Swanepoel, H. *A super quick comparison between Kafka and Message Queues* [online]. hackernoon.com [cit. 2020-01-01]. Available at: https://hackernoon.com/a-super-quick-comparison-between-kafka-and-message-queues-e69742d855a8.

[35] Tam, K. C. *Transactions in Ethereum* [online]. medium.com [cit. 2020-01-01]. Available at: https://medium.com/@kctheservant/transactions-in-ethereum-e85a73068f74.

[36] Tar, A. *Proof-of-Work, Explained* [online]. cointelegraph.com [cit. 2020-01-01]. Available at: https://cointelegraph.com/explained/proof-of-work-explained.

[37] VASWANI, J. *9 Proven Ways to Bypass Anti Scraping Techniques 2020* [online]. bloggersideas.com [cit. 2020-01-01]. Available at: https://www.bloggersideas.com/anti-scraping-techniques.

[38] Vokráčko, L. *Activity Alarm for Cryptocurrency Blockchains*. Brno, CZ, 2018. Master's thesis. Brno University of Technology, Faculty of Information Technology. Available at: https://www.vutbr.cz/en/students/final-thesis/detail/114872.

[39] Walker, G. *Checksum* [online]. learnmeabitcoin.com [cit. 2020-01-01]. Available at: https://learnmeabitcoin.com/guide/checksum.

[40] Winkler, C. J. and Scheithauer, G. Challenges of business service monitoring in the internet of services. *International Conference on Information Integration and Web-based Applications Services*. 2008, no. 10, p. 613–616.

[41] Won, D. *Bitcoin vs. Bitcoin Cash: The Full Comparison* [online]. exodus.io [cit. 2020-01-01]. Available at: https://exodus.io/blog/bitcoin-vs-bitcoin-cash.

[42] Wulf, P. de. *A guide to Web Scraping without getting blocked in 2020* [online]. scrapingbee.com [cit. 2020-01-01]. Available at: https://www.scrapingbee.com/blog/web-scraping-without-getting-blocked.

[43] xrpl.org. *Address Encoding* [online]. xrpl.org [cit. 2020-01-01]. Available at: https://xrpl.org/accounts.html#address-encoding.

[44] Zrnčík, H. *Identification of cryptocurrency users*. Brno, CZ, 2018. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Available at: https://www.vutbr.cz/en/students/final-thesis/detail/121961.

Appendix A

Namespaces Diagram

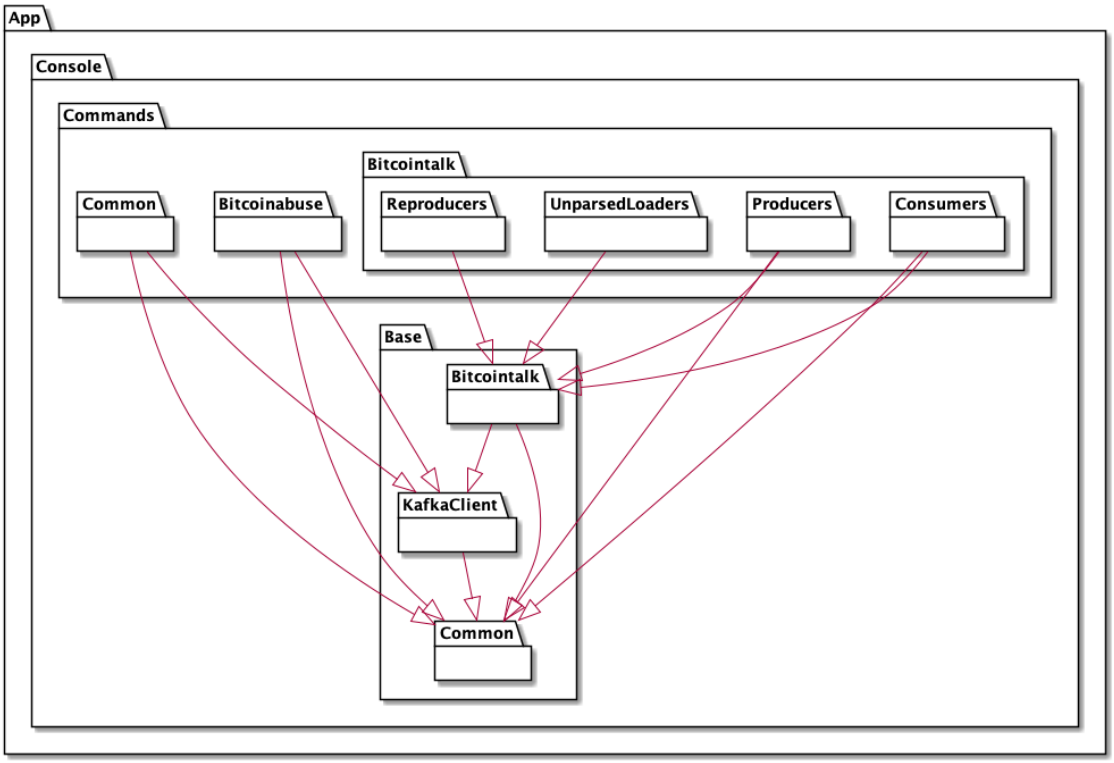

Figure A.1: The scraping core PHP namespaces dependencies diagram.

Appendix B

Total Data Prediction

```
WITH board_pages AS (
  SELECT
    round(avg(cnt), 2) as avg
  FROM (
     SELECT count(*) as cnt,
        parent_url
     FROM bitcointalk_board_pages
     GROUP BY parent_url
     ORDER BY cnt DESC
  ) as counts
), main_topics AS (
  SELECT
    round(avg(cnt), 2) as avg
  FROM (
     SELECT count(*) as cnt,
        parent_url
     FROM bitcointalk_main_topics
     GROUP BY parent_url
     ORDER BY cnt DESC
  ) as counts
), topics_pages AS (
  SELECT
    round(avg(cnt), 2) as avg
  FROM (
     SELECT count(*) as cnt,
        parent_url
     FROM bitcointalk_topic_pages
     GROUP BY parent_url
     ORDER BY cnt DESC
  ) as counts
),
```

```
actual_counts AS (
  SELECT
    (SELECT count(*) FROM bitcointalk_board_pages) as act_bpages_cnt,
    (SELECT count(*) FROM bitcointalk_main_topics) as act_mtopics_cnt,
    (SELECT count(*) FROM bitcointalk_topic_pages) as act_tpages_cnt,
    (SELECT count(*) FROM bitcointalk_user_profiles) as act_uprofile_cnt
), user_profiles AS (
  SELECT ROUND(( 1.0 * act_uprofile_cnt / act_tpages_cnt), 2) as uprofile_avg
  FROM actual_counts
), avg_stats AS (
  SELECT
    count(*) as main_boards_cnt,
    (SELECT avg FROM board_pages) as bpages_avg,
    (SELECT avg FROM main_topics) as mtopics_avg,
    (SELECT avg FROM topics_pages) as tpages_avg
  FROM bitcointalk_main_boards
), predict_counts AS (
  SELECT
    main_boards_cnt,
    round(main_boards_cnt * bpages_avg, 2) as pred_bpages_cnt,
    round(main_boards_cnt * bpages_avg * mtopics_avg, 2) as pred_mtopics_cnt,
    round(
      main_boards_cnt * bpages_avg * mtopics_avg * tpages_avg, 2
    ) as pred_tpages_cnt,
    round(
      main_boards_cnt * bpages_avg * mtopics_avg * tpages_avg * uprofile_avg, 2
    )as pred_uprofile_cnt
  FROM avg_stats, user_profiles
), pred_time AS (
  SELECT
    ROUND(
      (main_boards_cnt + pred_bpages_cnt + pred_mtopics_cnt + 2 * pred_tpages_cnt
        + pred_uprofile_cnt) / (3600 * 24), 2
    ) as pred_total_days
  FROM predict_counts
)
SELECT
  pred_total_days,
  main_boards_cnt,
  pred_bpages_cnt,
  pred_mtopics_cnt,
  pred_tpages_cnt,
  pred_uprofile_cnt,
  round(act_bpages_cnt / pred_bpages_cnt, 2) * 100 as bpages_perc,
  round(act_mtopics_cnt / pred_mtopics_cnt, 2) * 100 as mtopics_perc,
  round(act_tpages_cnt / pred_tpages_cnt, 2) * 100 as tpages_perc,
  round(act_uprofile_cnt / pred_uprofile_cnt, 2) * 100 as uprofile_perc
FROM predict_counts, actual_counts, pred_time
```

Appendix C

Total Data Counts

```
SELECT count(*)
FROM addresses
WHERE id IN (
    SELECT address_id
    FROM identies
    WHERE regexp_match(url, '.*bitcointalk|bitcoinabuse.*') IS NOT NULL
);
```

Appendix D

Excel@FIT Poster