Sprawozdanie z projektu Big Data

Szymon Rećko, Hubert Ruczyński, Mateusz Ziemła

13.01.2023

## Geneza projektu

Kryptowaluty stały się niesamowicie popularne w ciągu ostatnich kilku lat wśród wielu konsumentów i inwestorów. Zawdzięczają to głównie cenie i szybkości wykonywania transakcji z ich użyciem. Aby móc efektywnie działać na rynku kryptowalut należy posiadać narzędzia umożliwiające jego analizę. Szczególną uwagę należy poświęcić analizie podejścia użytkowników, które w znaczącym stopniu wpływa na wartość kryptowaluty.

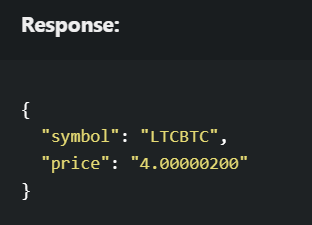
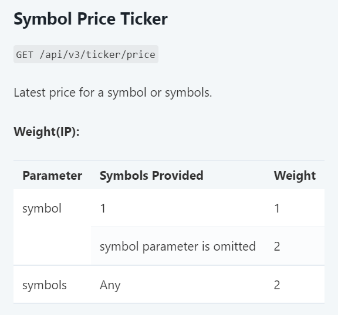
Celem naszego projektu jest zbadanie zależności między ilością tweetów publikowanych na Twitterze przez   
użytkowników, a wartościami kilku najpopularniejszych kryptowalut. Kryptowaluty, które zostaną wzięte pod uwagę to:

* Bitcoin (BTC)
* Ethereum (ETH)
* Dogecoin (DOGE)
* Cardano (ADA)
* Polygon (MATIC)
* XRP (XRP)

## Dane

### Akcje

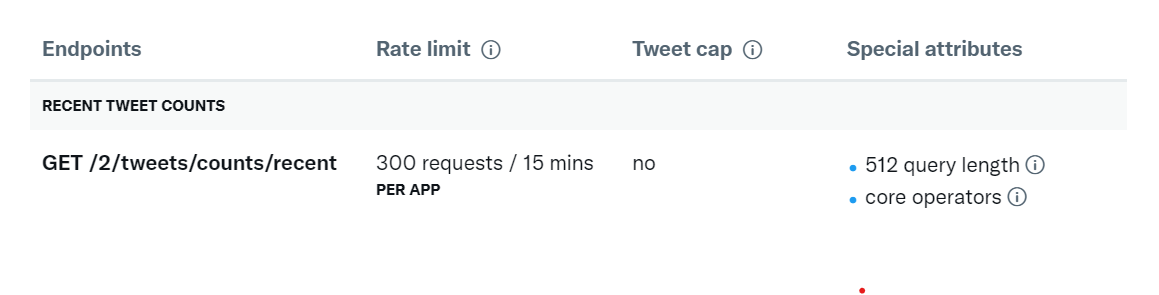
Dane o wartościach kryptowalut są pobierane za pomocą “[Binance API](https://www.binance.com/en/binance-api)”. Korzystając z ogólno dostępnej metody   
“*GET /api/v3/ticker/price”* możliwe jest pobranie jednym zapytaniem cen kryptowalut z przeliczeniem na różne rzeczywiste waluty krajowe (takie jak: USD, EUR, GBP). Dane są zwracane w ustalonym formacie JSON. Darmowa wersja API pozwala na wysłanie 1200 zapytań na minutę, przy czym różne zapytania mają różną “wagę” więc, aby nie przekroczyć limitu i nie dostać blokady na IP, wywołujemy zapytania dla wszystkich kryptowalut raz na 15 sekund.



Rysunek 1 Binance API – przykładowe zapytanie

### Tweety

Tweety powiązane treścią z kryptowalutami są pobierane za pomocą “[Twitter API v2](https://developer.twitter.com/en/docs/twitter-api)”. Do filtrowania tweetów wykorzystujemy frazę ‘crypto’, tak aby wyodrębnić interesujące nas tweety. Dzięki temu, raz na minutę pobierana jest lista tweetów zawierająca timestamp, treść tweeta, oraz jego identyfikator. Wszystkie zapytania zwracają dane w postaci JSON.



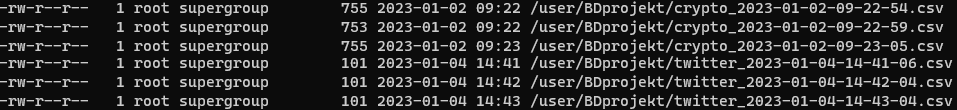
Rysunek Twitter API – przykładowe zapytanie

## Projekt architektury lambdaDiagram Description automatically generated

Rysunek Architektura Lambda

## Składowanie danych

**Apache Hadoop** - Jako warstwę składowania master data set ‘u wybraliśmy Apache Hadoop. Jedną z kluczowych zalet Hadoop Distributed File System (HDFS) jest jego zdolność do zapewnienia wysokiej dostępności i odporności na uszkodzenia. Dane przechowywane są w wielu węzłach w klastrze w więcej niż jednej kopii, co oznaczą, że mimo awarii jednego z serwerów nadal można uzyskać dostęp do danych z innych węzłów. Dodatkową zaletą jest skalowalność. Wraz ze wzrostem ilości danych w głównym zbiorze, do klastra można dodać więcej węzłów, co pozwala systemowi obsługiwać większe ilości napływających danych. Ponadto użycie Hadoop’a pozwala nam obsługiwać zarówno dane strukturalne i niestrukturalne, a także SQLowe i NO-SQLowe, zatem nie musimy dodawać kolejnych narzędzi tak jak byłoby to w przypadku Hive. Dane zawarte w głównym zbiorze są odświeżane co 15 sekund. Pliki znajdują się w formacie CSV i dla łatwiejszego dalszego przetwarzania zawierają będą timestampy zarówno w nazwie jak i wewnątrz zawartości. Na poniższym zrzucie ekranu widać przykładową listę plików, które są składowane w HDFS. Z łatwością wskazać można sposób rozróżniania danych dotyczących Twittera oraz kryptowalut.



Rysunek Przykład danych w HDFS

**Apache Hive** – W Hive składujemy będziemy widoki pochodzące z głównego zbioru danych. Zdecydowaliśmy się na Hive, gdyż łatwo sięga się do niego po dane z perspektywy aplikacji klienckiej. Każde uruchomienie skryptu w Sparku składuje w Hive obecny stan wszystkich znanych danych. W Hive przechowujemy dane w postaci rzędowej, dla łatwego dostępu do wszystkich danych, z partycjonowaniem po czasie, z którego pochodzą dane, w przypadku obu tabel.

A black screen with white text

Description automatically generated with low confidence A picture containing text

Description automatically generated

Rysunek Przykład danych w Hive

**Apache HBase** - Jako platformę NO-SQL wybraliśmy HBase, do którego ładujemy dane pozyskiwane z API dotyczącego kryptowalut. Zdecydowaliśmy się na niego, ponieważ informacje o nim poznaliśmy na zajęciach, a jest także jednym z narzędzi sprawnie działających razem z Hadoop’em oraz ze Sparkiem. Dla łatwiejszego dostępu stworzyliśmy dodatkowe kolumny, symbol\_1 oraz symbol\_2, wskazujące osobno na symbole użyte w parach handlowych. Pogrupowaliśmy kolumny według tego czy opisują dane o cenie w danym momencie (price, time) oraz symboli walut których dotyczą dane.

Text

Description automatically generated

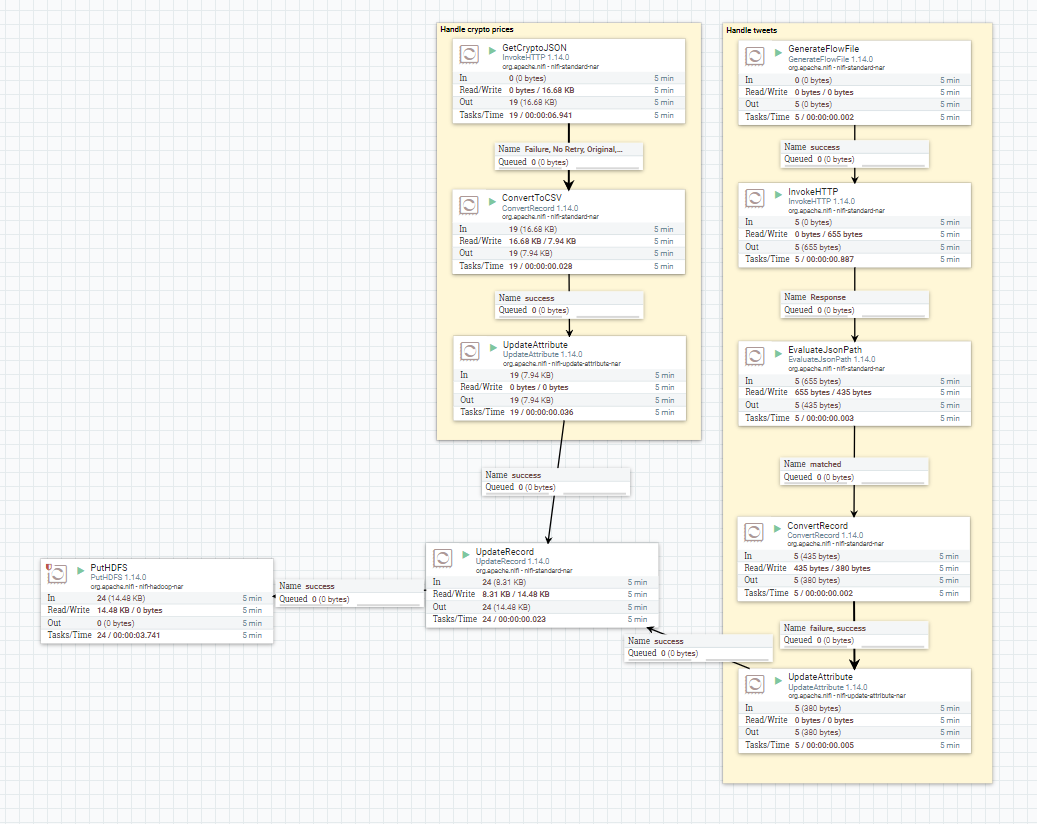
A black screen with white text

Description automatically generated with low confidence

Rysunek Przykład danych w HBase

## Przepływ danych

**Apache NiFi** – NiFi służyć nam będzie w procesach pozyskiwania danych z API oraz ich wstępnego preprocessing ’u tak aby do HDFS’a trafiały już wstępnie przygotowane dane. NiFi pobierać będzie zatem tweety i komunikować się z Binance oraz Twitter API. Głównym powodem użycia Apache NIFI w tym projekcie była jego zdolność do obsługi szerokiej gamy źródeł i formatów danych. Jest to możliwe dzięki wbudowanym procesorom, które zapewniają również możliwość manipulacji danymi np. filtrowanie i agregację. Ponadto można go łatwo integrować z innymi użytymi przez nas systemami. Apache NIFI dostarcza dodatkowo interfejs użytkownika ułatwiający monitorowanie i zarządzanie przepływem danych. Jest to przydatne do inspekcji zbieranych danych, a także rozwiązywania wszelkich problemów, które mogą pojawić się na tym etapie. Nasze flow w NIFI dla kryptowalut składa się kolejno z: pobrania danych w formacie JSON, konwersji do formatu CSV i zmiany rozszerzenia pliku na “.csv”. Flow dla tweetów jest analogiczne z tą różnicą, że na początku generowany jest flowfile z danymi potrzebnymi do wykonania zapytania do API. Niezależnie od ścieżki, do danych zostaje dodany timestamp, a następnie dane umieszczane są w HDFS.



Rysunek Architektura NiFI

**Apache Spark** – Spark wykorzystywany będzie w pozostałych przypadkach, gdzie potrzebne będą bardziej skomplikowane operacje. Pierwszym jego zadaniem będzie przetworzenie danych z HDFS tak aby do Hive’a trafiały już w formie sensownych Batch Views. Ponadto za pomocą Sparka właśnie przekazywać będziemy dane z HBase i Hive do warstwy prezentacji. Jako że numpy średnio radzi sobie z przetwarzaniem dużych danych te przetwarzane będą wewnątrz Sparka z poziomu Python’owego Notebook’a i biblioteki PySpark.

## Wsadowa analiza danych

Dane pobierane są z utworzonych wcześniej Hive oraz HBase poprzez Apache Spark, następnie przerzucane są do ramki pandas w Python, aby użyć biblioteki seaborn do stworzenia wizualizacji odpowiadających na pytania:

1. Czy względne zmiany wartości kryptowalut zależą od waluty państwowej na którą tą wartość rzutujemy?
2. Czy częstsze tweetowanie o kryptowalutach (zawierające słowo ‘crypto’) wpływa na kursy kryptowalut?
3. Czy kurs kryptowalut wpływa na częstsze tweetowanie o kryptowalutach?
4. Czy kursy kryptowalut w czasie różnią się istotnie?
5. Czy da się zabezpieczać przy inwestycji w kryptowaluty poprzez kupowanie odwrotnie skorelowanych walut?
6. Jaka waluta się najmniej zmienia i jest najbezpieczniejsza?

Na przykładzie bitcoina zaprezentowanego na pierwszym wykresie Rysunek 8 widać, że kursy w pewnym stopniu różnią się od siebie w zależności od waluty w której kupić chcemy daną kryptowalutę. Różnice widoczne są szczególnie na drugim peaku. Na podstawie wykresu Rysunek 9 jesteśmy w stanie stwierdzić, że to liczba tweetów wpływa na wartości kryptowalut. Co więcej zauważyć można, że kolejne kursy w czasie wyglądają bardzo podobnie, co potwierdza następna wizualizacja Rysunek 10 prezentująca korelacje między kursami kryptowalut. Warto wyróżnić tutaj walutę XRP, która jako jedyne zachowuje się niejako odwrotnie od reszty, przez co może zostać użyta jako zabezpieczanie wymieniane w punkcie 5. Potwierdzenie tej relacji widoczne jest na ostatnim wykresie Rysunek 11. Dodatkowo stworzyliśmy analizę, która określa w jakiej walucie odnotowujemy najmniejsze zmiany – jest najbardziej stabilna, co widoczne jest w tabelce Tabela 1.

Chart

Description automatically generated

Rysunek 8 Porównanie ceny kryptowaluty w różnych walutach krajowych

A picture containing text, document

Description automatically generated

Rysunek 9 Porównanie wartości kryptowalut z ilością tweetów w czasie

Chart, treemap chart

Description automatically generated

Rysunek 10 Macierz korelacji kryptowalut

Chart, line chart

Description automatically generated

Rysunek 11 Porównanie kursów XRP oraz DOGE Coin

Table

Description automatically generated

Tabela 1 Analiza stabilności kryptowalut

## Repozytorium

Do pracy nad projektem zdecydowaliśmy się wykorzystać środowisko, w którym każdy z nas czuje się swobodnie, a mianowicie GitHuba. Repozytorium jest publiczne, projekt tworzony jest jako Open Source, a poniżej zamieszczony jest link do repozytorium: <https://github.com/HubertR21/BigDataProject>.

## Testy

### Obsługa testów

Aby rozpocząć pipeline testowania należy podążać za niniejszą instrukcją. Pierwszym krokiem jest stworzenie folderu do testowania wewnątrz hdfs za pomocą komendy: *hadoop fs -mkdir /user/testowanie.* Następnie sprawdzamy, że folder jest pusty za pomocą: *hadoop fs -ls /user/testowanie,* co wygląda tak:

Text

Description automatically generated

Następnie należy dostać się do NiFI za pomocą: <http://localhost:9443/nifi/> , zlokalizować grupę procesów o nazwie testowanie, przejść do niej i włączyć wszystkie znajdujące się w niej procesory. Dane zaciągać w ten sposób przez co najmniej 3,5 minuty (4 krotne stworzenie pliku potrzebnego do zaciągnięcia danych przez procesor GenerateFlowFile dla tweetów) i wyłączyć po tym czasie. Aby sprawdzić wyniki działania NiFI wywołujemy następującą komendę: *hadoop fs -ls /user/testowanie.* W efekcie uzyskujemy wynik podobny do tego:

Text

Description automatically generated

Rysunek 12 Test\_NiFI

Kolejnym krokiem jest przetestowanie przetwarzania danych za pomocą PySpark do Hive oraz HBase wykonywane przez plik *projekt/sparkito\_test.py.* Domyślnie odpowiednie tabele (twitter\_data\_test, crypto\_data\_test, oraz projekt\_test) nie znajdują się w Hive oraz HBase, co sprawdzić możemy poprzez komendy:

*beeline -e 'show tables;' -u jdbc:hive2://localhost:10000 -n vagrant p vagrant*

Text

Description automatically generated

*/usr/local/hbase/bin/hbase shell*

*list*

Text

Description automatically generated

Aby rozpocząć proces przetwarzania należy wykonać następującą komendę: *python* *projekt/sparkito\_test.py.* W efekcie skrypt stworzy tabele w Hive: twitter\_data\_test oraz crypto\_data\_test, natomiast w HBase stworzona zostanie tabela projekt\_test. Obserwacje pochodzące z dwóch pierwszych wypisywane są także podczas działania skryptu. Wyniki sprawdzić można za pomocą następujących komend, w efekcie których powinniśmy otrzymać wyniki podobne do tych:

*beeline -e 'show tables;' -u jdbc:hive2://localhost:10000 -n vagrant p vagrant*

*beeline -e 'SELECT \* FROM crypto\_data\_test LIMIT 10;' -u jdbc:hive2://localhost:10000 -n vagrant p vagrant*

*beeline -e 'SELECT \* FROM twitter\_data\_test LIMIT 10;' -u jdbc:hive2://localhost:10000 -n vagrant p vagrant*

Text

Description automatically generated Text

Description automatically generated

Text

Description automatically generated

Rysunek 13 Tests Hive

*/usr/local/hbase/bin/hbase shell*

*list*

*scan ‘projekt\_test’*

*describe ‘projekt\_test’*

Text

Description automatically generatedText

Description automatically generated

Text

Description automatically generated

Rysunek 14 Tests HBase

Ostatnim etapem jest tworzenie wizualizacji, które testowane jest za pomocą przejścia przez kolejne komórki w pliku Plots\_test.ipynb. Opisane tam działania sprawdzają czy jesteśmy w stanie odczytywać dane z Hive i HBase oraz czy wykresy generują się poprawnie. Aby uruchomić notatnik należy wpisać komendę *jupyter notebook*, następnie wejść do folderu projekt i otworzyć plik Plots\_test.ipynb. Wizualizacje powinny wyglądać podobnie do zamieszczonych poniżej:

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated Diagram

Description automatically generated with medium confidence

Chart, treemap chart

Description automatically generated Chart, line chart

Description automatically generated

Rysunek 15 Tests Python

### Wyniki

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Testowane narzędzie | Cel testu | Jak działa test? | Spodziewany rezultat | Przykładowy rezultat |
| NiFI | Sprawdzenie pobierania za pomocą API i transformacji danych w NiFI. | Użytkownik podąża za kolejnymi poleceniami [x] tworząc folder hdfs, włączając procesory testowe oraz sprawdza czy dane się pobrały. | Wewnątrz HDFS pojawiają się dane pobrane przez NiFI z odpowiednimi nazwami (crypto/twitter\_date). | Rysunek 12 Test\_NiFI |
| Spark  Hive  HBase | Sprawdzenie czy skrypt napisany w PySpark poprawnie obsługuje dane. | Użytkownik podąża za kolejnymi poleceniami [x] sprawdzając stan przed wykonaniem skryptu, wywołując skrypt, oraz sprawdzając stan po jego wywołaniu. | Wewnątrz Hive pojawiają się dwie tabele twitter\_data\_test oraz crypto\_data\_test, a wewnątrz HBase jedna projekt\_test. | Rysunek 13 Tests Hive  Rysunek 14 Tests HBase |
| Python  (wykresy) | Sprawdzenie czy wykresy są generowane. | Użytkownik włącza jupyter notebook, otwiera odpowiedni plik ipynb i wywołuje przygotowane komórki. | Wykresy łączą się z danymi i poprawnie generują (bez błędów). | Rysunek 15 Tests Python |

## Podział pracy

Szymon Rećko:

* Rozplanowanie projektu.
* NiFi: komunikacja z API, wstępny preprocessing danych, zapisanie danych do HDFS.
* Konfiguracja HDFS’a.

Hubert Ruczyński:

* Rozplanowanie projektu.
* Zbieranie danych za pomocą Sparka do warstwy prezentacji.
* Stworzenie notebook’a jako warstwy prezentacji (wizualizacja).
* Nadzorowanie i debugowanie każdego etapu.

Mateusz Ziemła:

* Konfiguracja HBase’a.
* Konfiguracja Hive’a.
* Konfiguracja Sparka między Hadoop’em, a Hive’em.