**Data Engineer**

**Final project**

Naya college

23/10/2022

**Boston Transportation Real-Time Data**

Nitay Yacobovitch

Shoham Gilady

Dor Izmaylov

**תוכן עניינים**

[**רקע מטרות הפרויקט ויישום**](#_krn7fuva6qn1) **3**

[**טכנולוגיות נבחרות**](#_w9m5b5dqvd9w) **5**

[**ארכיטקטורה**](#_evlz3f7931f8) **7**

[**אפיון התהליכים**](#_x920aaslrctu) **8**

[**תיעוד התהליכים**](#_4a5ilxvdtqsv) **12**

[**1. דרישות קדם**](#_jv3qi0vgiz8c) **12**

[**2. Configuration File**](#_ua0pr335i7c7) **12**

[**3. Telegram Bot**](#_14q6x7t1qg3n) **12**

[**4. Alert Dashboard**](#_ely3axrs81p7) **12**

[**5. Archive and Monthly/Yearly Report**](#_3yjvi2100fr2) **22**

# 

# **רקע מטרות הפרויקט ויישום**

**רקע**

רשות התחבורה של מסצ'וסטס ביי, הידועה יותר בשם T, היא אחת ממערכות התחבורה הציבורית הוותיקות ביותר בארצות הברית. זוהי גם מערכת התחבורה הגדולה ביותר במסצ'וסטס.

כחטיבה של מחלקת התחבורה של מסצ'וסטס (MassDOT), ה-MBTA מספק רכבת תחתית, אוטובוס, רכבת, מעבורת ושירותי מעבר למזרח מסצ'וסטס ולחלקים מרוד איילנד.

MBTA מאפשרת לפנות אליה בצורה חינמית ולקבל מידע נרחב על כלי התחבורה שלה, המיקום שלהם בלייב לוח הזמנים המתוכנן ולוח הזמנים בפועל, בנוסף ניתן להוציא מטה דאטה כמו התראות, מסלולים, תחבורה, תחנות, וכו'.

בחרנו להתחבר לדאטה שלהם כיוון שהוא עשיר וניתן ליישם עליו מגוון רחב של תהליכים ופתרונות.

**מטרות**

מטרות עסקיות:

1. לאפשר לנוסע לקבל דיווחים בזמן אמת על נסיעה שהוא מתכנן, לקבל דיווח בנוגע לתזמון הנסיעה, איפה התחנה הכי קרובה אליו והאם לוח הזמנים מתעכב או לא.
2. לעזור לנוסע לקבל מידע על ההתראות הקיימים ב MBTA, ולדעת בזמן אמת האם יש התראה רלוונטית למסלול הנסיעה הנסיעה שלו.
3. שליחת סיכום חודשי דרך הסלאק למנהלים ובעלי עניין שונים על פעילות התחבורה הציבורית.

מטרות טכנולוגיות/טכניות:

1. לעשות שימוש בכלים ובטכנולוגיות עליהם למדנו בקורס
2. לפתח כמה תהליכים עם מטרות ויכולות שונות וזאת כדי שנוכל להתנסות במגוון רחב יותר של טכנולוגיות והתנסות בהטמעה של pipelines מסוגים שונים ולמטרות שונות.
   1. עיבוד של דאטה בריל טיים והתממשקות מול לקוח על ידי BOT
   2. עיבוד של דאטה בריל טיים עם היסטוריה מצומצמת והצגה שלו בדשבורד אינפורמטיבי
   3. בניית archive לשמירת כל ההיסטוריה הקיימת

**יישום**

כדי לענות על המטרות שהוצגו קודם החלטנו לבנות 3 תהליכים עיקריים:

1. בוט טלגרם:  
   תהליך זה יציג תהליך בזמן אמת מול משתמש קצה. התקשורת עם המשתמש תתבצע בטלגרם כאשר האלגוריתם יודע לקבל מהמשתמש את המסלול הרצוי שלו. מאחורי הקלעים נעשית בדיקה מול ה API של MBTA כדי לחפש את כלי התחבורה (אוטובוס או כל כלי אחר) אשר נמצאים בקרבת מקום ועונים על המסלול אותו היוזר מבקש.  
   היוזר יקבל מהבוט רשימה של המסלולים הטובים ביותר, ויצטרך לבחור את המסלול המועדף עליו.  
   הבוט בהתאם, יציג ליוזר את המסלול על גבי מפה וישלח לו את זמני ההגעה הצפויים לתחנה.  
   בסוף, הבוט פועל באופן עצמאי כדי לעדכן את המשתמש במידה ויש שינוי משמעותי בזמני ההגעה לתחנה (משמע אם האוטובוס מאחר או מקדים).
2. דשבורד להצגת Alerts:  
   תהליך זה יציג למשתמש אנליזה בסיסית של סוגי האלרטים הפעילים הקיימים במערכת. בנוסף, היוזר יוכל לראות את כל כלי התחבורה על המפה של מסצ'וסטס עם מידע על: קיבולת (כמות נוסעים), האם קיים אלרט פעיל על כלי תחבורה זה ואם כן מהו.  
   בנוסף היוזר יכול לראות רשימה אינפורמטיבית של האלרטים הקיימים במערכת לפי דרגת חומרה.  
   היוזר יכול לסנן את המידע בהתאם למה שהוא רוצה לראות (דוגמא, לסנן ספציפית על אוטובוס שהוא מתעניין לגביו)
3. דוח תקופתי (יומי/חודשי/שנתי):  
   באמצעות הדו"ח התקופתי ניתן יהיה לראות מגוון רחב של נתונים אגרגרטיביים אודות השימוש והפעולה של התחבורה הציברות במסצ'וסטס. (לדוגמא: קווים שנוטים להיות בתפוסה גבוהה, אוטובוסים שנוטים לאחר ללוחות הזמנים הקבועים מראש, נסיעה במהירות גבוהה, תדירות, תחנות "פופולאריות" ועוד…) בעזרת נתונים אלו ניתן יהיה להסיק מסקנות ולקבל החלטות עסקיות בהתאם.   
   בנוסף, משום שהדו"ח מבוסס על מגוון רחב של נתונים, ניתן לעקוב באמצעות הדו"ח אחר אנומליות והשפעה של אירועים מיוחדים בתחבורה הציבורית.

# **טכנולוגיות נבחרות**

1. מקורות מידע:
   1. MBTA Api - במהלך סריקה ראשונית שביצענו למקורות API שונים (דרך אתר rapid ועוד…) מצאנו API המספק מידע על תחבורה ציבורית בארה"ב. כשהתחלנו לתחקר את ה API ראינו שהמידע שם מאוד דל, במקביל המשכנו לחפש עוד באינטרנט והגענו לחברת MBTA, אשר מספקת מידע על כלי התחבורה השייכות לה (רכבת תחתית, אוטובוס, רכבת, מעבורת) במסצ'וסטס. המידע שהגיע מה- API הזה היה מפורט ונרחב יותר ולכן בחרנו לעשות בו שימוש.
   2. Google API - שימש להמרה בין כתובות לקואורדינטות מדויקות של מיקום. בנוסף, הדרך הפשוטה ביותר להצגת מפה עם מסלול של האוטובוס הנבחר בבוא הייתה להתחבר לגוגל ולהביא משם את המפה הרלוונטית.

PIPELINES: בהתחלה חשבנו לעשות שימוש בכלים של אמזון (S3, athena וכו'...) אבל אחרי מחקר מעמיק ראינו שהכלים הללו דורשים מאיתנו השקעה כספית, ואם הולכים לכיוון של טריאל לכל כלי יש את המגבלות שלו. הזמן להגשת הפרויקט התארך ולא רצינו להגיע למצב שתקופת הטריאל לא תספיק לנו ונצטרך להקים את הכל מחדש. לכן שינינו כיוון ועשינו שימוש בכלים חינמיים.

1. זרימת דאטה:   
   בחרנו להשתמש ב kafka ו- nifi כדי להזרים את הנתונים ל TARGET השונים בפרויקט מכמה סיבות:
   1. בוט ודשבורד - תהליכים אלו הם תהליכי real-time ,לכן בחרנו ב kafka. זוהי טכנולוגיה מתאימה ל real time, ואמינה. במידה והתהליך נופל באמצע, הדאטה לא הולך לאיבוד אלא נשמר ב epics השונים.
   2. Archive - אחד הדברים שאנחנו רצינו לאחסן הוא פעילות רצופה של קווי האוטובוסים והתחבורה הציבורית בכלל. משום שמדובר בנתונים שמשתנים כל רגע, החלטנו שאנחנו נאסוף סטטוס עבור כל כלי התחבורה הציבוריים כל דקה. על מנת להעביר את הנתונים האלה במהירות ל HDFS בצורה אמינה ורציפה השתמשנו ב KAFKA ובשביל להאזין בזמן אמת, לעבד את הנתונים ולהביא אותם לצורת האחסון המתאימה השתמשנו ב NIFI.
2. עיבוד מידע:  
   שימוש ב spark ופייתון לעיבוד של הדאטה וטרנספורמציה שלו בהתאם לצרכי התהליך.  
   זוהי בחירה טבעית שכן spark ו- kafka מתממשקים ועובדים טוב יחד למימוש תהליכים real time.  
   כמו כן, משום שהמידע הגולמי התקבל בפורמט JSON, השתמשנו בפייתון כדי לשלוף אותו ולשטח אותו בצורה קלה לפני השליחה שלו ל KAFKA או ל HDFS.
3. ארכיון:  
   בחרנו להשתמש בפלטפורמה של קלאודרה (HDFS), מהסיבה הפשוטה שהיא חינמית ומספקת את התשתית הנדרשת ל - big data. שמירה ועבודה עם נפחים גדולים של דאטה.
4. Databases:
   1. MongoDB: נבחר לתמוך בתהליך הבוט. במונגו נשמרות הבחירות של משתמש הקצה, והדאטה לתמיכה בתהליך הכולל של הבוט. מונגו הינו בדאטה בייס שיודע לתמוך בתהליכים אופרטיביים, משמע שליפה, עדכון והכנסה של מידע במהירות.
   2. MySQL: דאטה בייס זה נבחר לתמוך בדשבורד האינפורמטיבי וזאת מהסיבה שהדשבורד מציג מידע בנפחים לא גדולים ולכן הביצועים יהיו טובים.

בחרנו בשתי סוגים שונים של דאטה בייסים בשביל המגוון וההתנסות בפרויקט.

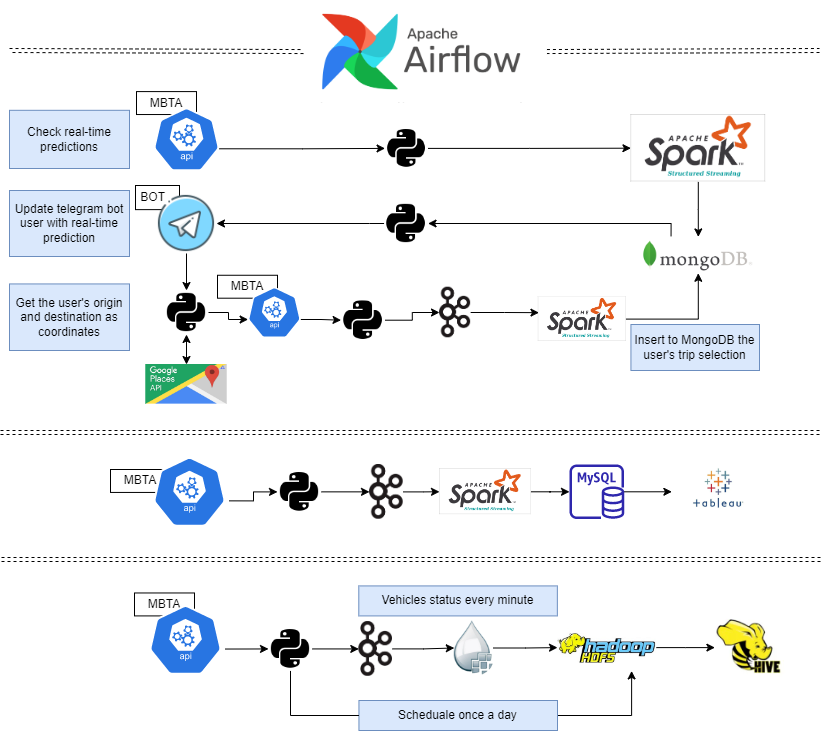
1. תחקור נתונים:  
   בדו"ח התקופתי, עיבוד ותשאול המידע נעשו על ידי HIVE. HIVE היא שפת תשאול נתונים עם ממשק SQL המאפשרת שירותי סיכום נתונים, שאילתות, וניתוחים. בפנינו עמדו שתי אופציות עיקריות, IMPALA ו- HIVE. אנחנו בחרנו להשתמש ב HIVE משום ש HIVE מותאם לשאילתות כבדות, המכילות JOINS, אגרגציות, טרנספורמציות ורצות הרבה זמן. וזאת, עקב כך שהשאילתות עתידות לרוץ על DATA גדול מאוד.
2. כלי תצוגה ו interface מול משתמש קצה:
   1. Telegram Bot:  
      פלטפורמה נוחה לתשאול אינטראקטיבי, חינמית וקלה למימוש באמצעות python.
   2. Tableau:  
      כלי זה נבחר בגלל ידע קודם שהקבוצה רכשה. בנוסף היה לנו רישיון קיים לתוכנה, לכן זו הייתה הבחירה הטבעית לפרויקט.
   3. Slack:

היה ניסיון ראשוני למימוש באמצעות GMAIL, אבל גילינו שיש הגבלות אבטחה חזקות מדי ולכן עברנו למימוש ב slack. פלטפורמה נוחה לשליחת דוחות, חינמית. היוזר מקבל התראה עם הדוח החדש שנוצר ישירות לטלפון/מייל/אפליקציה במחשב.

1. תזמון:  
   airflow

# **ארכיטקטורה**

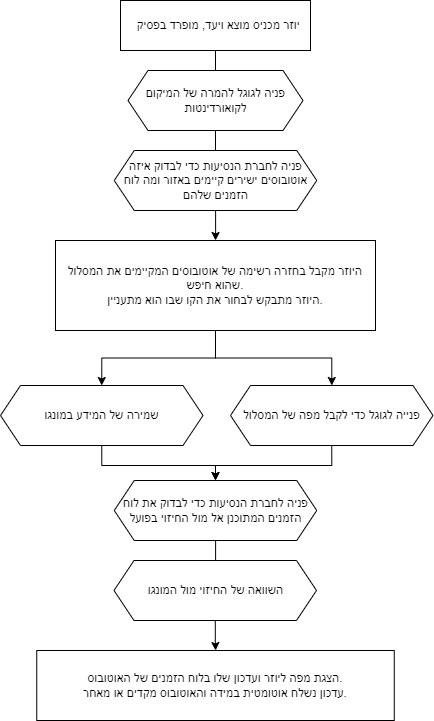
<https://app.diagrams.net/#G1jYdT-pu49mfsnA1K6VqkkHYG_bSNDa1k>



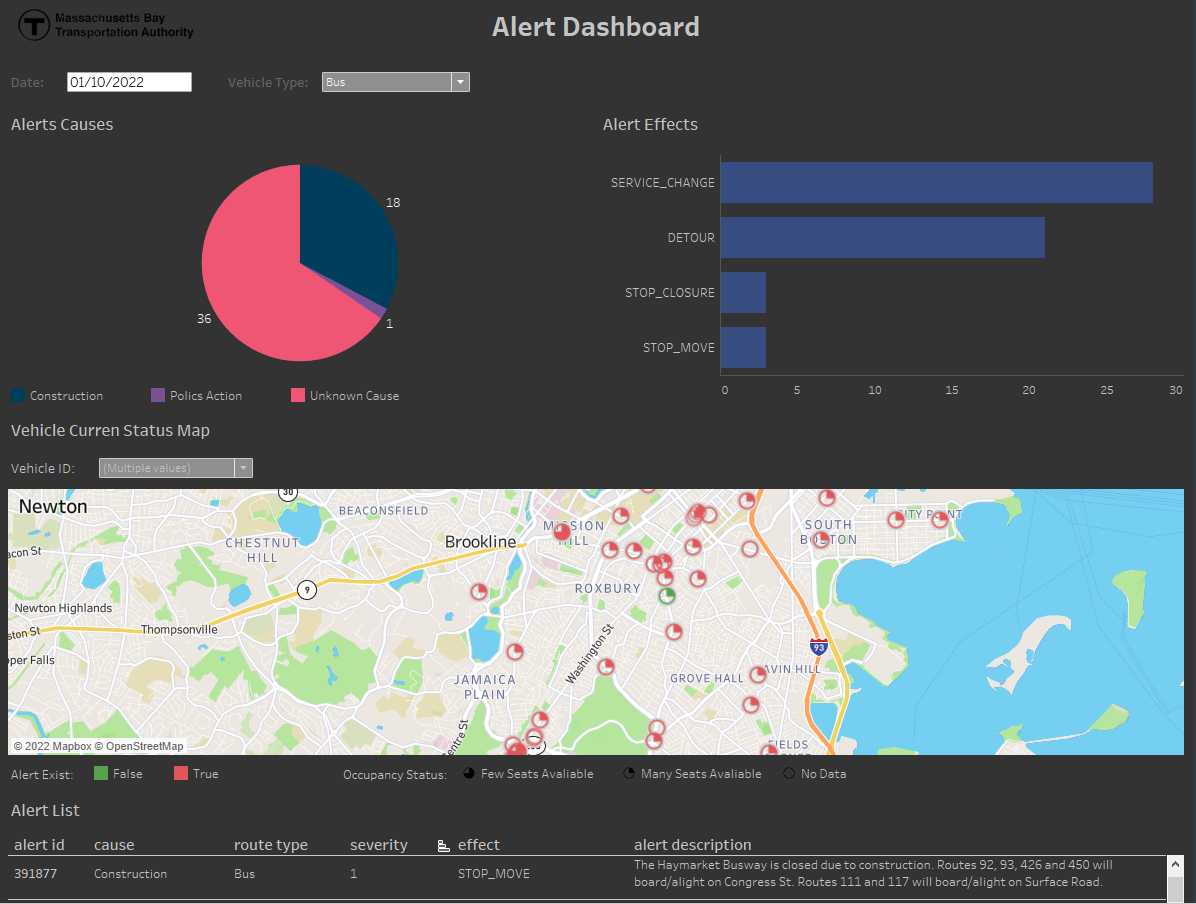
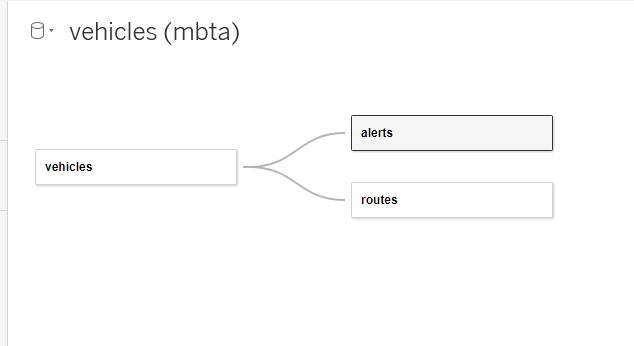
# 

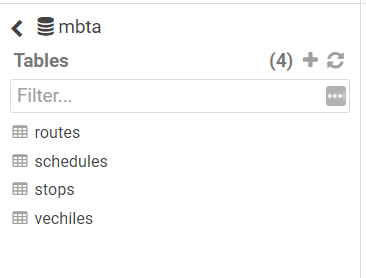
# **אפיון התהליכים**

1. **Telegram Bot:**
   1. לוגיקה:

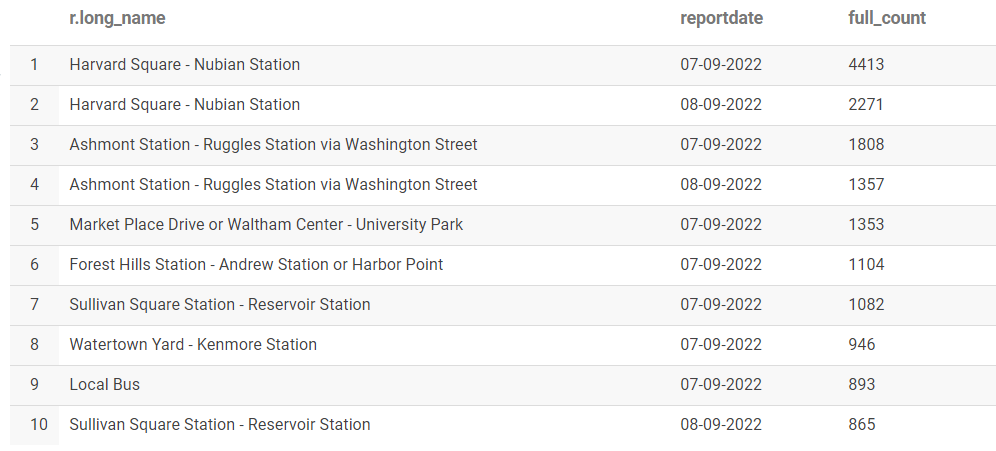


* 1. הנחות בתהליך:
     1. הרדיוס סביב המיקום שעבורו מחפשים תחנות הוא 0.005
     2. יש רק מסלולים ישירים (ללא החלפות)
     3. המסלול שמוצג POLYLINE הוא בין שתי הקואורדינטות שהמשתמש הכניס (ולא בהכרח התחנות עצמן שעבורן מוצג המסלול)
     4. כדי להבין איזה קווים עוברים בין שתי התחנות, מחפשים SCHEDULES בין הזמן הנוכחי +20 דקות עבור תחנת המוצא, ובין הזמן הנוכחי +100 דקות עבור תחנת היעד
     5. אם הצגנו למשתמש מספר מסלולים לבחירה (שיש להם SCHEDULE) אבל אין PREDICTION בריל טיים, מבקשים ממנו להציג שוב
     6. הסקריפט שיבדוק את הזמן העכשווי עבור המסלול ירוץ כל X זמן ואם הזמן מתקצר או מתעכב הוא יציג הודעה למשתמש
     7. לכל PREDICTION ו SCHEDULE יש ARRIVAL TIME ו DEPARTURE TIME. אם אחד מהם הוא NULL אז נשים את אותו ערך בשניהם.

1. **Alert Inform Dashboard**
   1. תצוגה:  
      <https://public.tableau.com/app/profile/shoham2947/viz/AlertsDashboard-MBTA/AlertDashboard>
   2. יכולות:
      1. ניתן לקבל מידע על יום ספציפי שנבחר (יכול לתמוך בכמה ימים אחורה)
      2. ניתן לפלטר את הדשבורד לפי סוג של כלי תחבורה
      3. בלחיצה על סיבה ל alert, כל הדשבורד יושפע בהתאם
      4. מפה:
         1. תצוגה של כל כלי התחבורה שיש עליהם כרגע מידע, ניתן לפלטר ולקבל מידע רק על קווים נבחרים.
         2. מהסתכלות על האייקון של הקו ניתן לראות האם קיים עבורו ALERT ומה רמת התפוסה של האוטובוס (כמה אנשים יש עליו).
         3. Tooltip: מספק מידע נוסף על הקו.
      5. רשימה של האלרטים ממויינים לפי רמת החשיבות, כולל הצגה מלאה של תיאור האלרט.
   3. Tables:
      1. באמצעות קאפקא וספארק יצרנו 3 טבלאות מרכזיות ב - Mysql
      2. יצירת דאטה סורס בטאבלו המקשר בין שלושת הטבלאות בהתאם לצרכי הדשבורד:  
         

* **Archive ודוח חודשי/שנתי:**פירוט טבלאות בסיס הנתונים בHIVE:  
  

תרומה ויכולות:

* מערכת לשמירת היסטוריה של נתונים גולמיים בצורה פשוטה וזמינה לניתוח כולל הצלבת מידע ממערכות שונות.
* באמצעות הנתונים הנאספים ניתן לבצע ניתוחים על המידע על מנת להסיק תובנות עסקיות.
* ניתן להוסיף סוגים של נתונים נוספים כמו: מידע של חיישנים, אירועים ועוד… ולשלב איתם את הנתונים הקיימים על מנת ליצור דוחות תקופתיים.
* מחסן הנתונים מכיל המון מידע הניתן לתחקור ומבט לאחור.
* ישנן כמה שאילתות שהכנו על מנת להמחיש את התובנות שאנחנו יכולים לקבל מהמידע.
  + בדיקת מה הם הקווים שמאחרים הכי הרבה ללוח הזמנים ובכמה:   
      
    באמצעות השליפה הזאת ניתן לבסס החלטות של שינוי לוחות זמנים על מנת להיות מדוייקים יותר.
  + בדיקה מה הם הקווים העמוסים ביותר על סמך כמות התראות על העומס המתקבלות מהקווים.   
      
    באמצעות השליפה הזאת ניתן לבסס החלטות של תגבור קווים מסויימים על מנת שיהיה פחות צפוף ונוח יותר לקהל הנוסעים.

# 

# **תיעוד התהליכים:**

## **דרישות קדם:**

* 1. הקמת סביבה ב GCP - יצירת account ייעודי לפרויקט, הקמת דוקר ופריסת ה image של הקורס על גבי המכונה. פריסת image של nifi.
  2. התקנה של Pycharm והקמת אינטרפרטר משותף על גבי הדוקר.

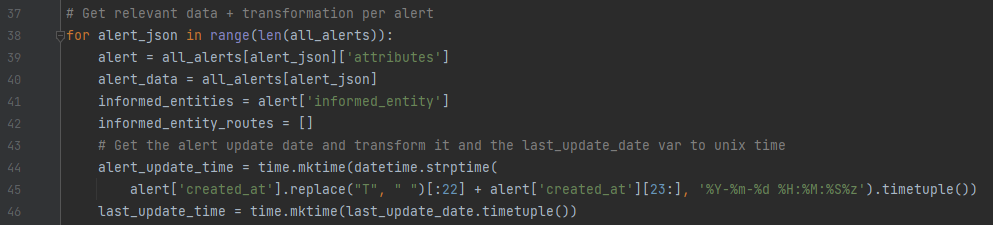
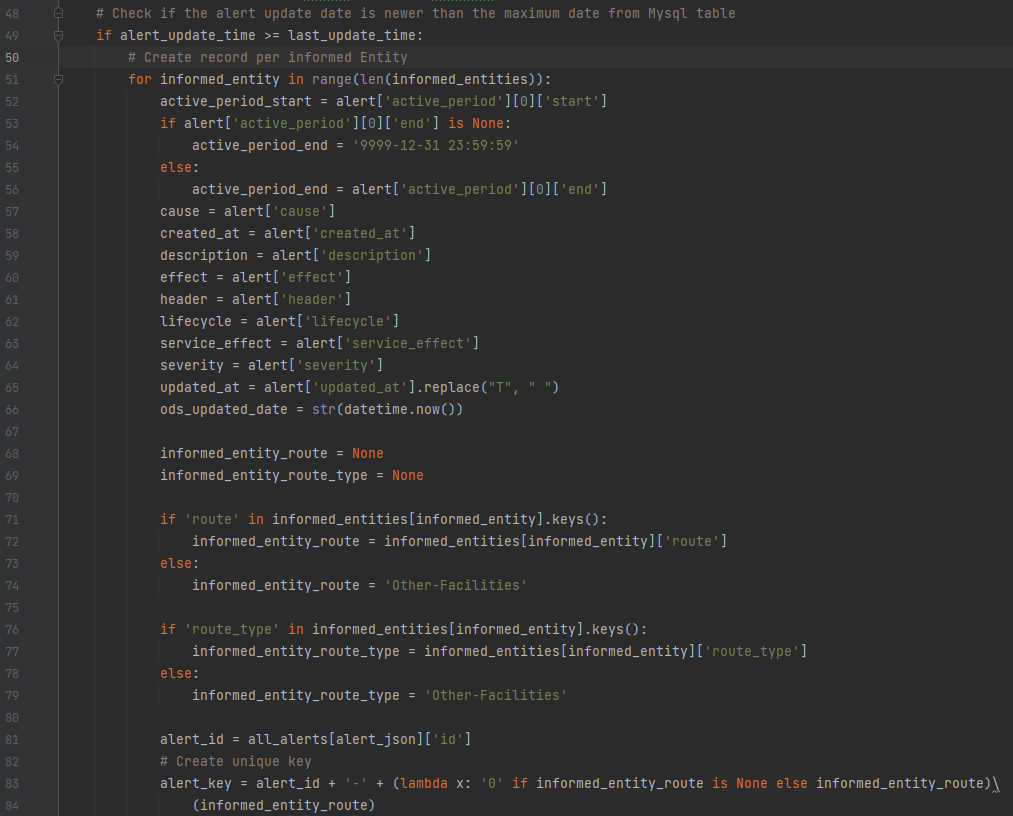
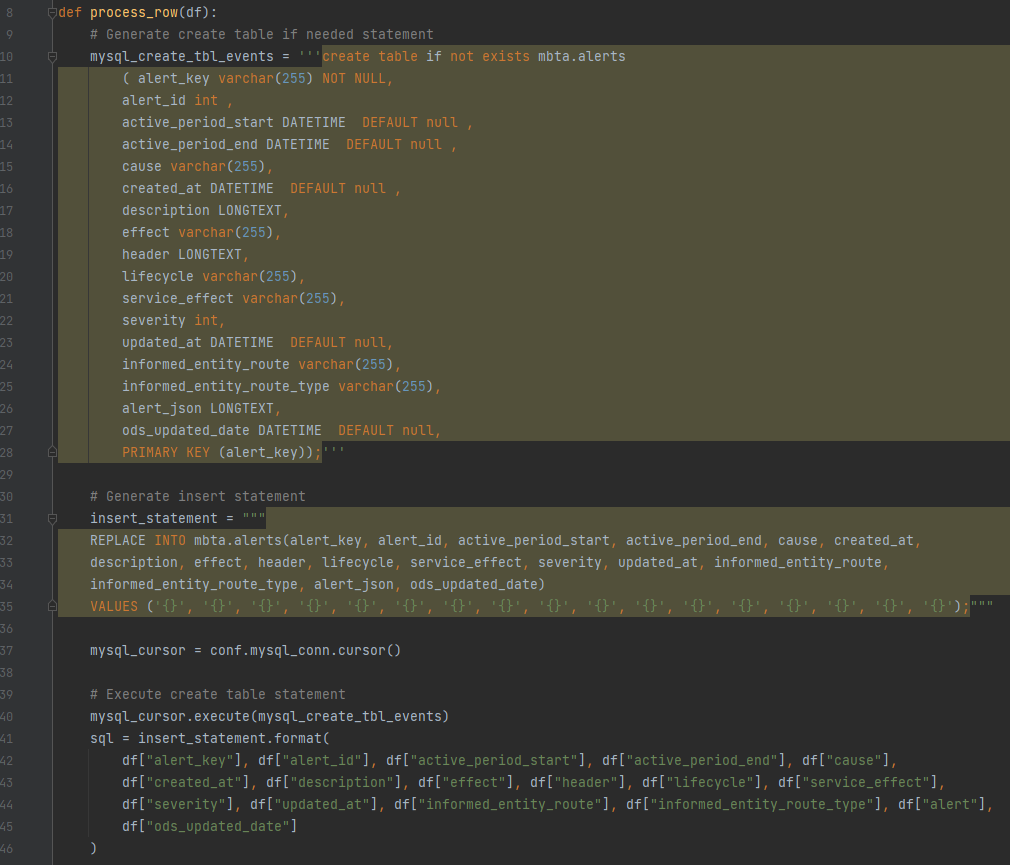
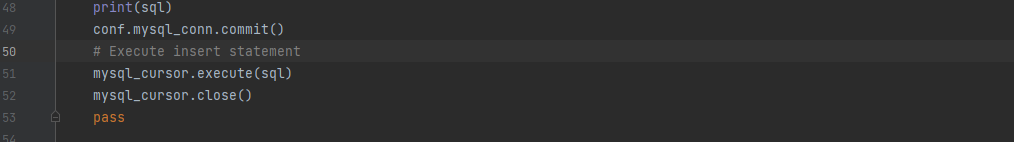
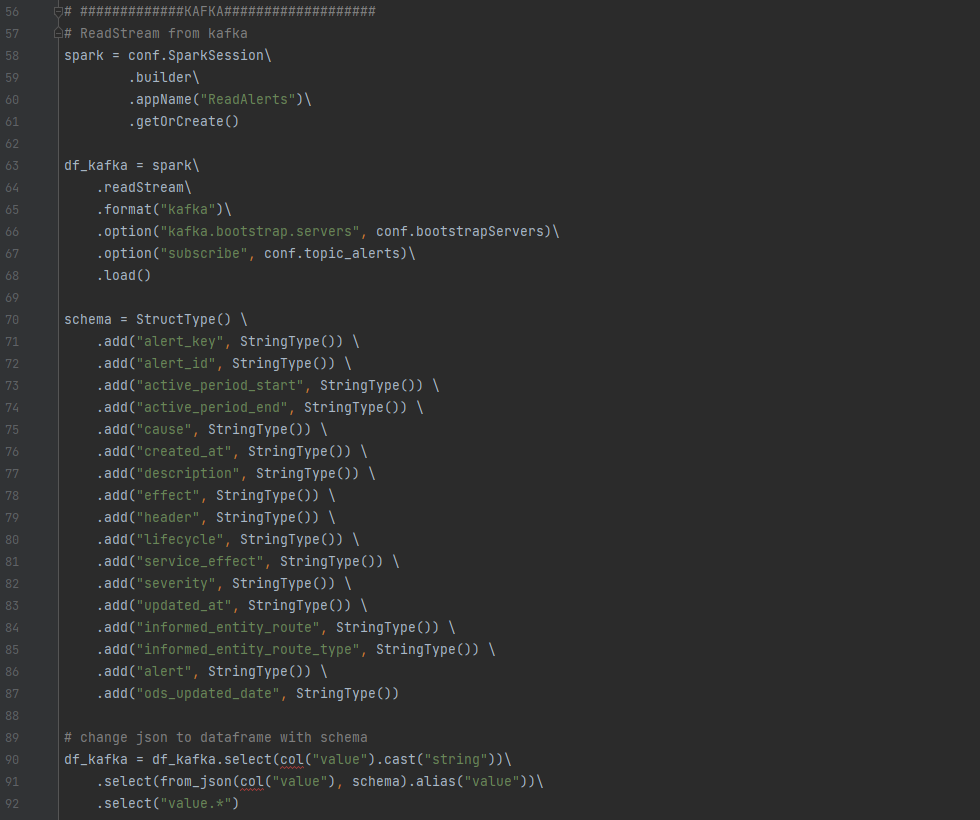
## **Configuration File:**

לטובת הפרויקט, יצרנו קובץ קונפיגורציה משותף, קובץ זה מכיל את כל החיבורים לפלטפורמות הנדרשות, שם גם יימצא כל המידע הרגיש (פרטי התחברות, כתובות IP ופורטים)

## **Telegram Bot:**

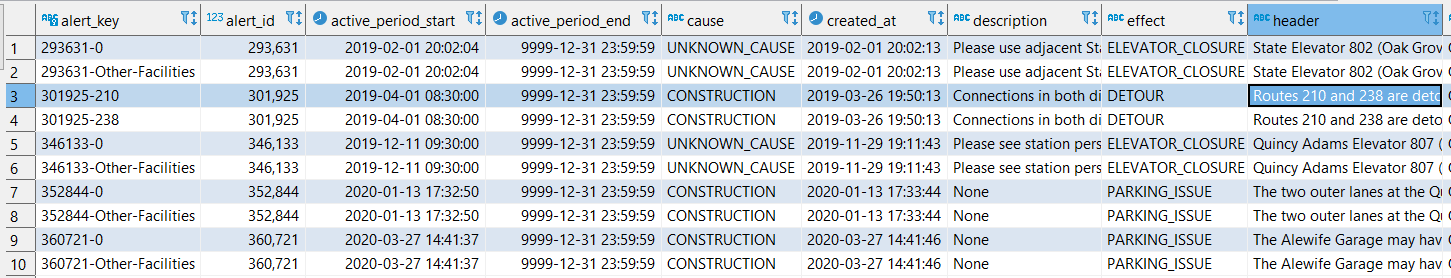
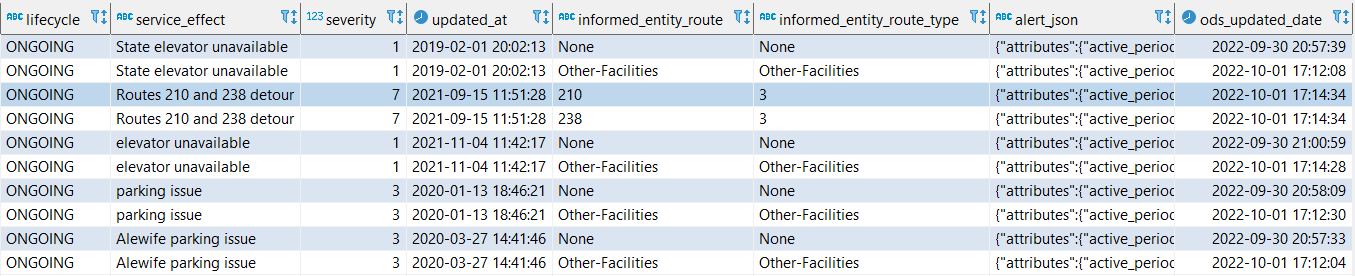
דור

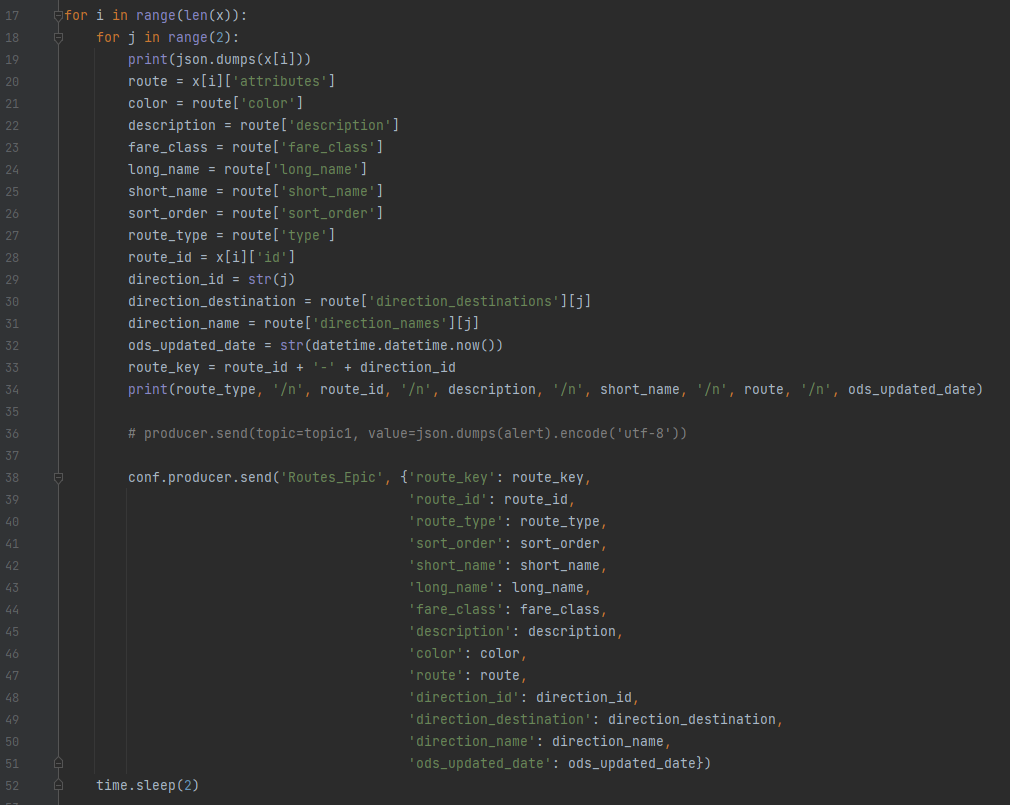
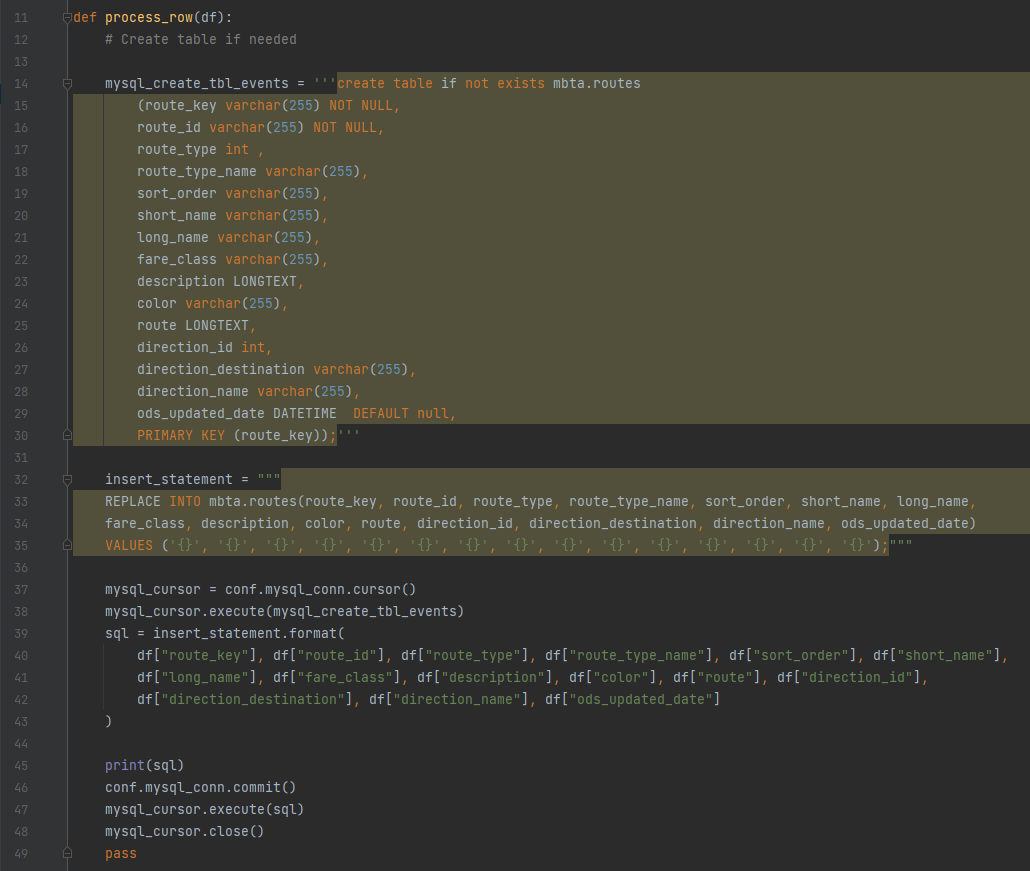
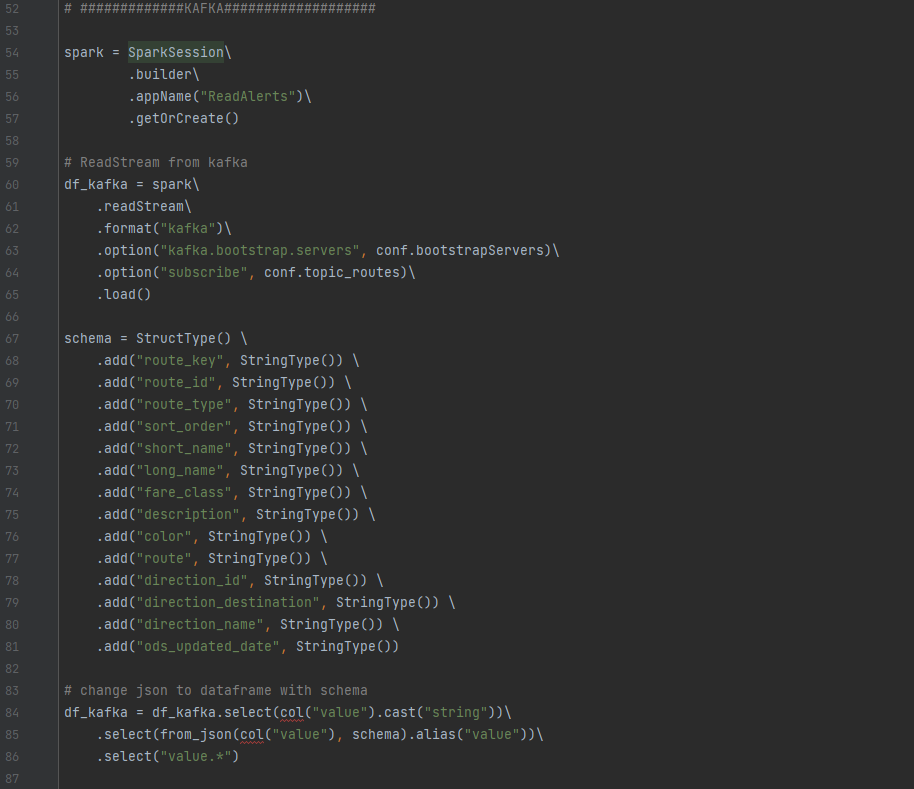
## **Alert Dashboard:**

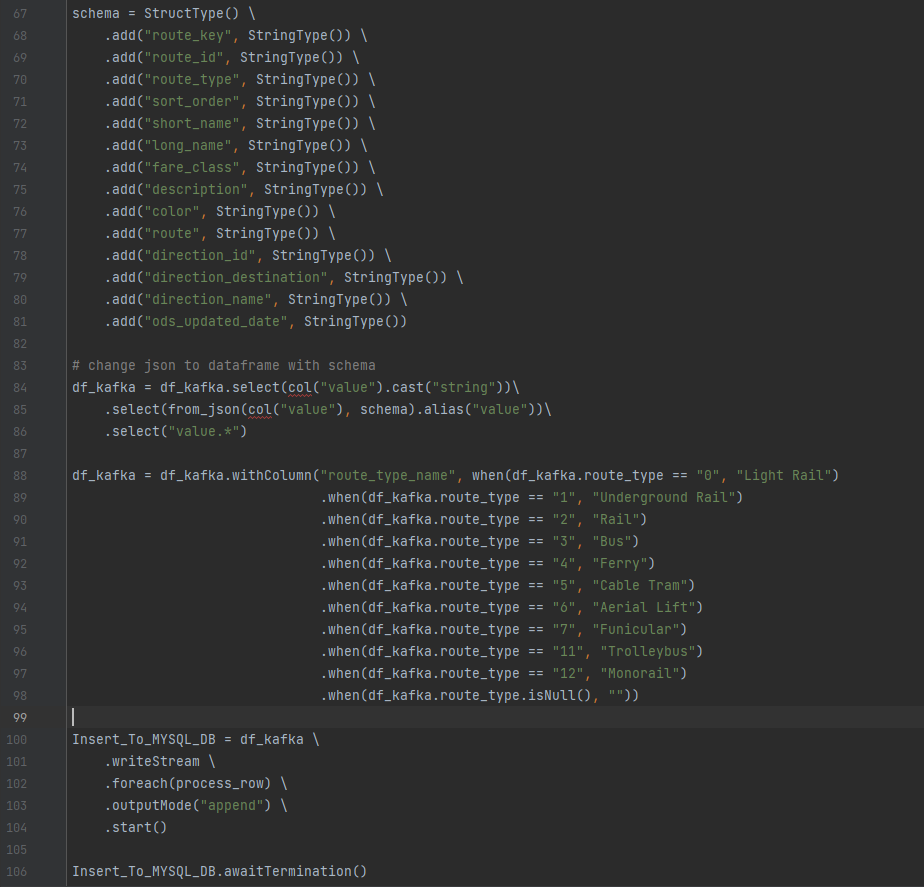
* 1. ספריות בשימוש:  
     time, requests, json, datetime, pyspark, kafka, mysql.connector,os
  2. Alert Topic:
     1. Producer: תמיכה בטעינה אינקרמנטלית:  
        בכל קריאה מה API, מקבלים את כל ההתראות הקיימות במערכת. כל התראה מגיעה עם שדה last\_update\_date.  
        כדי להעביר רק את ההתראות החדשות + התראות שהתעדכנו מהטעינה האחרונה שבוצעה ל MySql, יצרנו פונקציה שמביאה את התאריך עדכון המקסימלי מ - MYSQL.
     2. מעבר על כל התראה, ובדיקה של תאריך העדכון. לפני ביצוע ההשוואה יש להמיר את הפורמטים של התאריכים שיהיו זהים
     3. אחד השדות שמגיע כחלק מה JSON של ההתראה הוא שדה informed\_entities. שדה זה מכיל list של האובייקטים אשר מושפעים מהתראה זו. בשביל הדשבורד, הייתי צריכה לשטח את המידע וליצור רשומה עבור כל אובייקט מושפע, ולא להעביר את המידע ל MySql בצורה של List.  
        לכן יצרתי לולאה שעוברת על הליסט הזה ויוצרת Record לכל informed\_entitie.  
        informed\_entitie יכול לחזור יותר מפעם אחת ב list. לכן בקוד יש שמירה בצד של ה informed\_entities שכבר תועדו. בסוף, כדי לא ליצור רשומות כפולות לפני השליחה של ה Record ל- Topic, יש בדיקה האם ה informed\_entitie חוזר על עצמו או שהוא חדש, במידה והוא חדש מעבירים אותו הלאה.  
        בתוך הלולאה מתבצעת השמה של הדאטה הרלוונטי במשתנים וביצוע טרנספורמציה של הדאטה במידת הצורך.  
        
     4. Consumer:  
        פונקציית process\_row, מקבלת את ה record הרלוונטי מקפקא ושולחת פקודת REPLACE (לתמוך בטעינה אינקרמנטלית) ל MySql.
     5. קריאת ה record מה producer וביצוע שינויים בעמודות במידת הצורך.  
        
     6. תוצאה סופית:

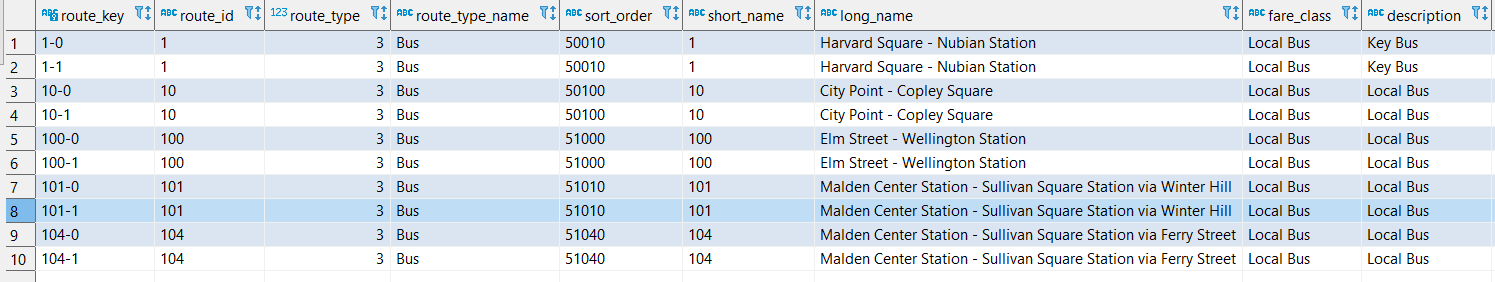
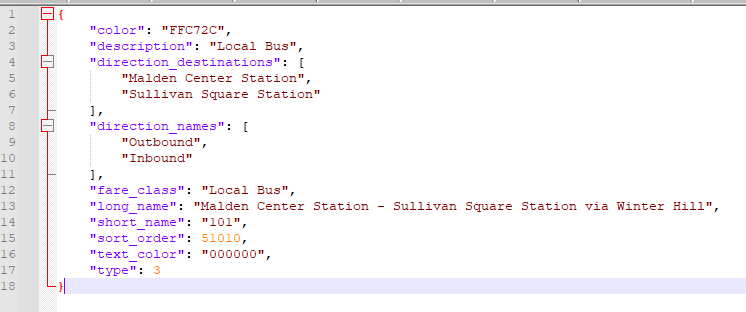
מבנה ג'ייסון התחלתי:

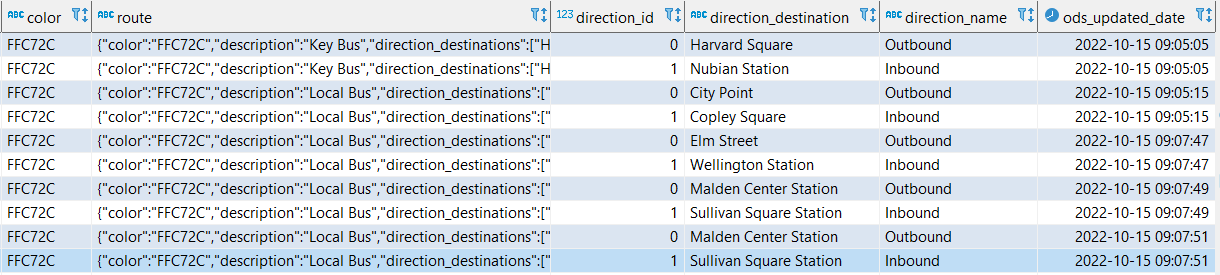
מעבר למבנה טבלאי ב MySQL:



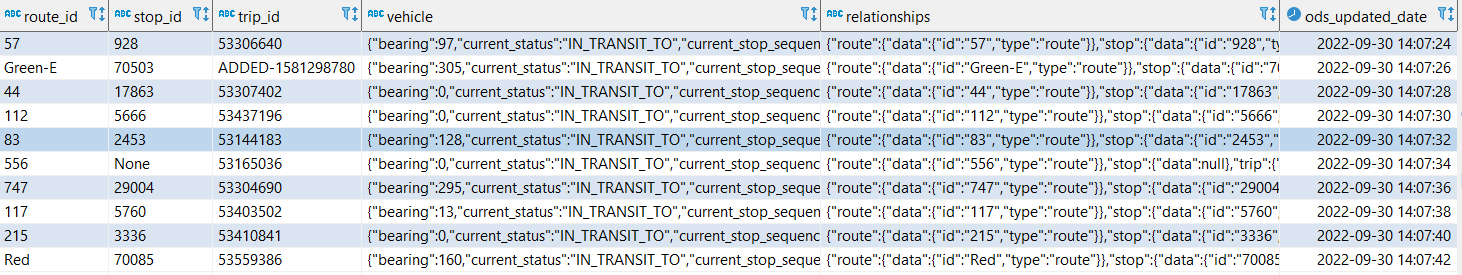
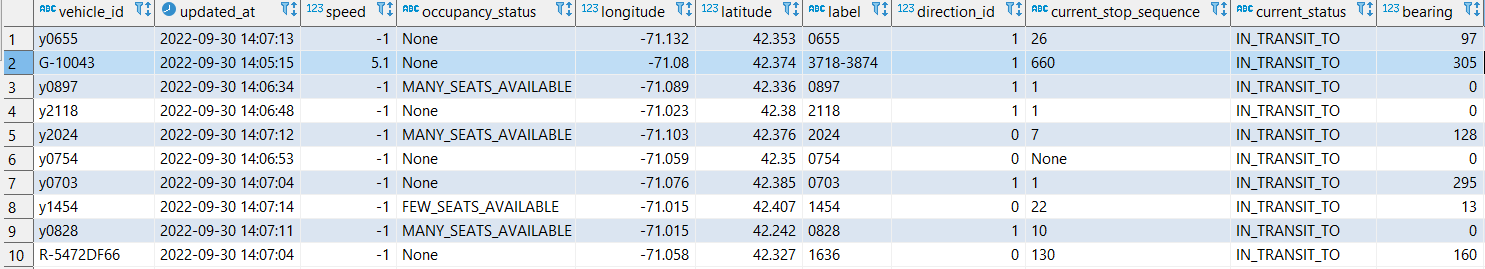
* 1. Routes Topic:
     1. Producer:   
        API זה לא מספק שדה עדכון רשומה, לכן לא הייתה אפשרות לתמוך בטעינה אינקרמנטלית בשיטה שנעשתה עבור ההתראות.  
        לכן תתקיים תמיכה רק בהכנסת הדאטה ל MySql.  
        ה Produce מייצא את הדאטה למשתנים ומבצע טרנספורמציה במידת הצורך:
     2. Consumer:  
        שיטה זהה להתראות



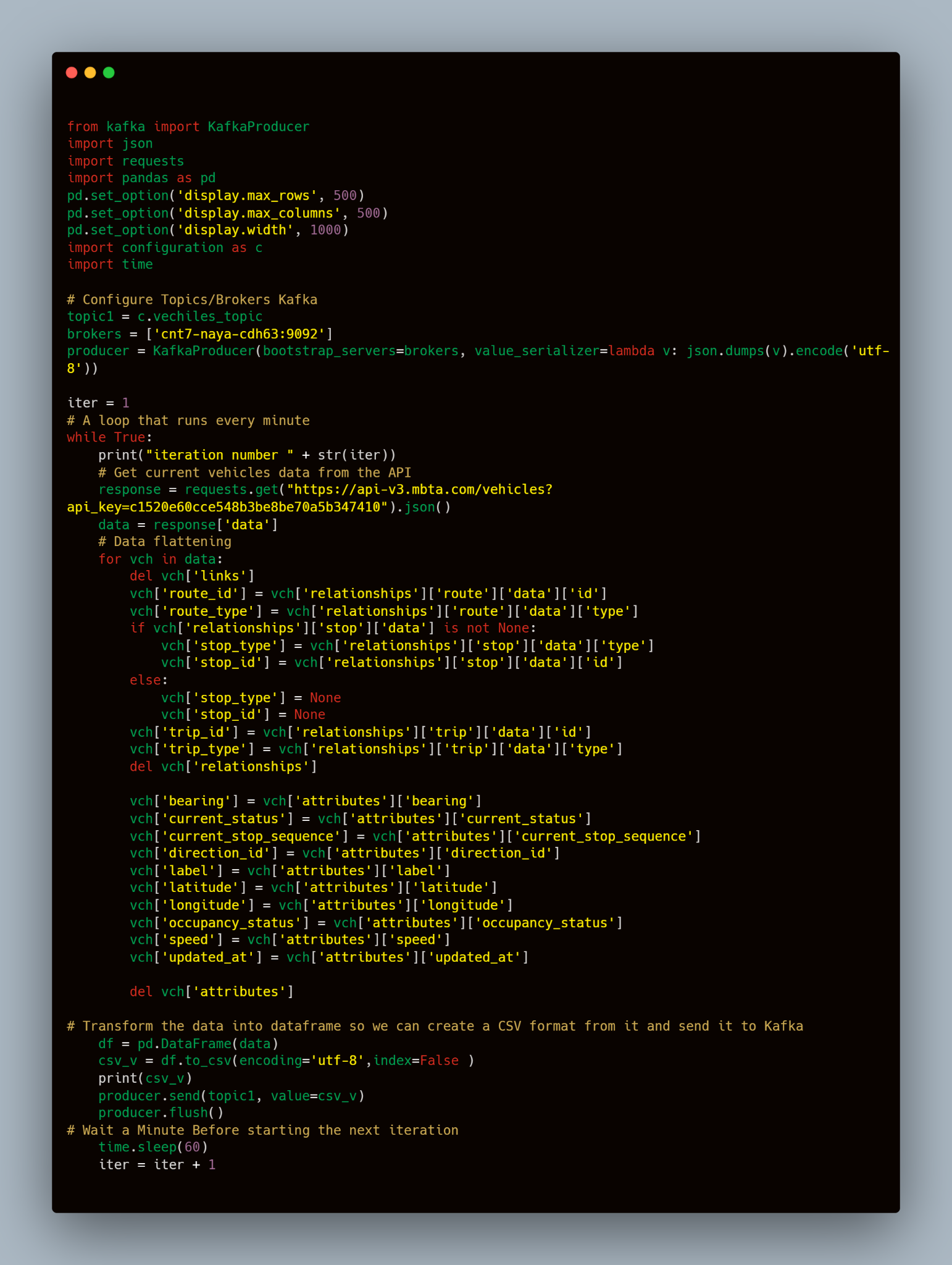
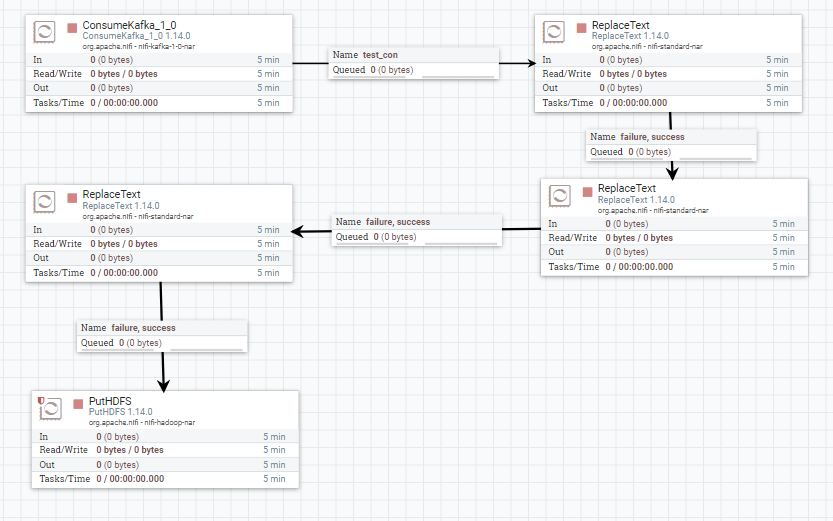
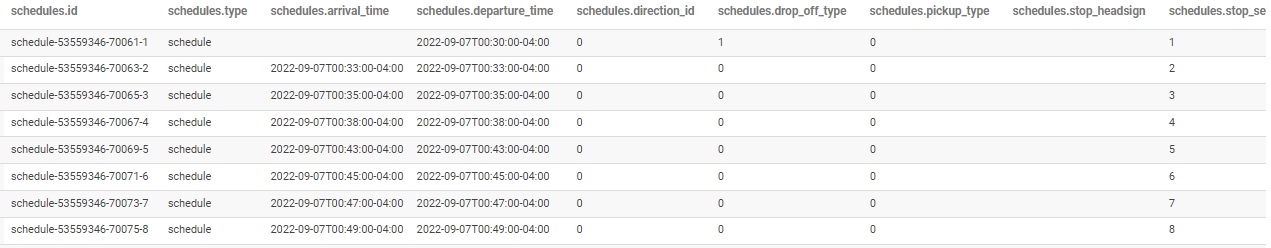
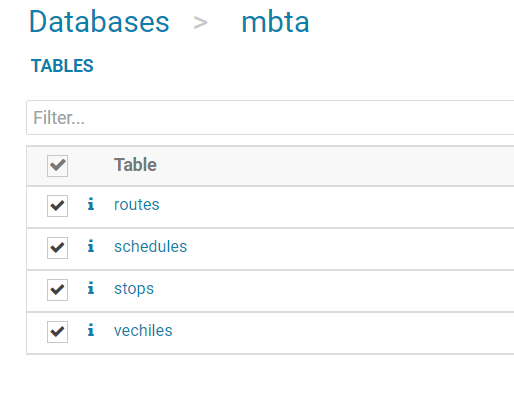
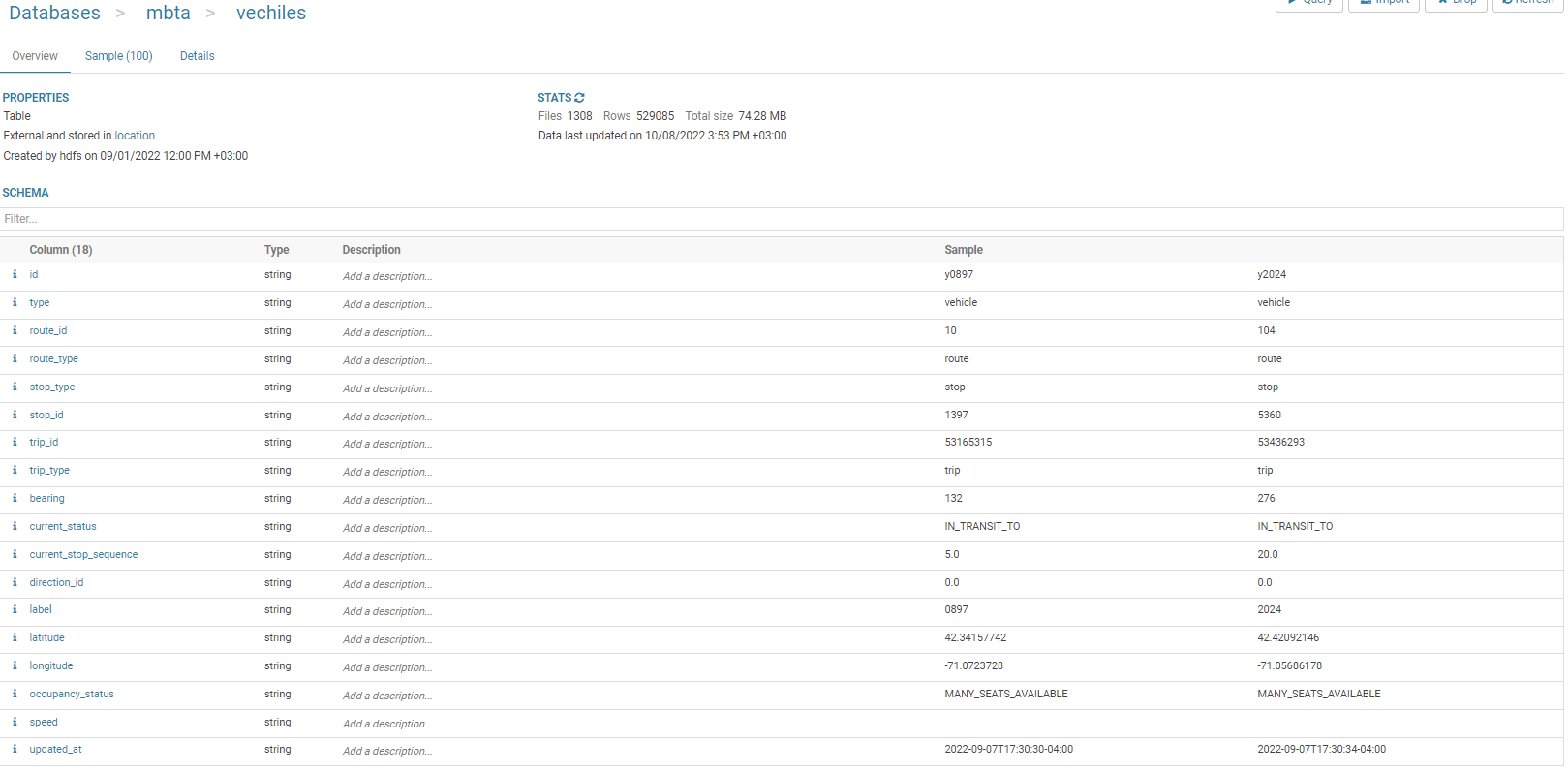
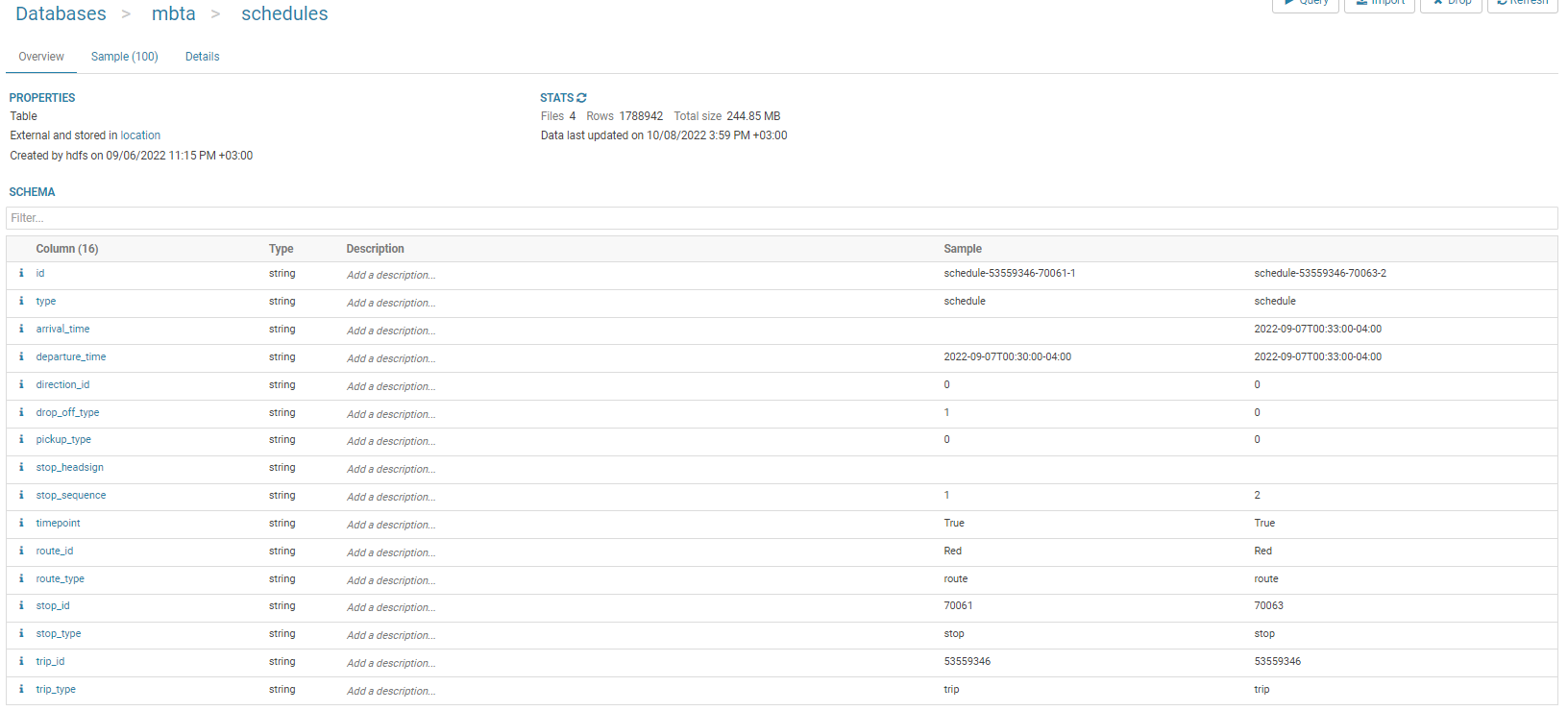
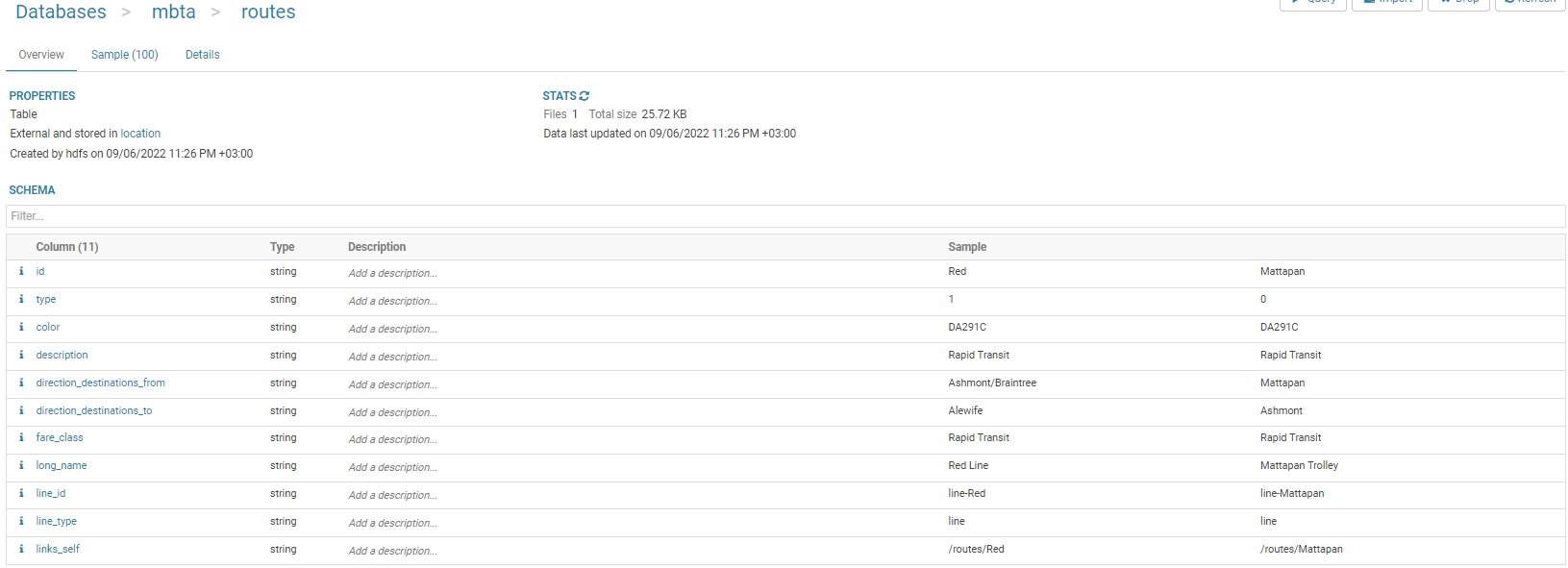
* + 1. תוצאה סופית:  
       מבנה ג'ייסון התחלתי:  
       מעבר למבנה טבלאי ב MySQL:



* 1. Vehicles Topic:  
     בוצע באותה לוגיקה של ה API's האחרים, ההבדל המשמעותי הוא בהכנסה ל MySQL.  
     כאן הטעינה מתבצעת ב Insert, כיוון שבכל פנייה ל API זה מקבלים את המיקומים של כלי התחבורה נכון לנקודת הזמן שבה פנינו ל API. זהו מידע חדש.  
     בעתיד - יש לתמוך בשמירת היסטוריה אשר רלוונטית לדוח האלרטים בלבד (יומיים, שלושה או שבוע)
     1. תוצאה סופית:  
        מבנה ג'ייסון התחלתי:  
        

מעבר למבנה טבלאי ב MySQL:  


## **Data Warehouse**

1. שליפת הנתונים מתוך הAPI
   1. נתונים סטטים על נתונים קווים (Routes) ונתוני תחנות (Stops) נאספו בצורה חד פעמית והועלו לבסיס הנתונים בHIVE.
   2. Vehicles - נתונים הרכבים נאספים ומשוטחים כל דקה באמצעות PYTHON  
      
   3. Vehicles - באמצעות NIFI מאיזינים לTOPIC המתאים בKAFKA ועושים טרנספורמציות מתאימות לנתונים כדי שHIVE יוכל לזהות אותם כCSV כשיהיה על הHDFS. אחריי הטרנספורמציה NIFI מעלה את הנתונים לHDFS   
        
      הנתונים בטבלת HIVE:   
      
   4. טבלת Schedules - מכילה את לוחות הזמנים העדכניים. פעם ביום נשלפים כל הנתונים אודות לוחות הזמנים לאותו יום עבודה. משום שמדובר בכמות גדולה של נתונים לשליפה אנחנו שיטחנו את הנתונים בPYTHON והעלנו ישירות לHDFS  
        
        
        
      הנתונים בטבלת SCHEDULES:  
      
   5. תהליכים דומים נבנו גם עבור Stops ועבור Routes. משום שמדובר במידע סטטי אנחנו לא טוענים אותו בצורה רציפה. אך במידת הצורך ניתן למחוק את הנתונים מהHDFS ולהפעיל את הבאת הנתונים החדשים מהAPI באמצעות PYTHON דרך KAFKA ודרך NIFI עד שמיר
   6. מעל הHDFS ישנו בסיס נתונים בHIVE בשם MBTA.   
      מצ"ב סקריפטים ליצירת הטבלאות בHIVE:  
        
        
      ****

# 

# **Architecture Requirements Overview**

## Prerequisites

To be able to maintain solution the following are required:

* Create a GCP account - configure and deploy the Course Docker image on a GCP
* Pycharm installation on local developer machine.

## Internal and External Data Sources

* [Massachusetts BayTransportation Authority](https://www.mbta.com/) V3 API– we used the MBTA API that contains real-time and static data on the different services,trips, lines, vehicles, alerts and few other data items. It returns the data in JSON format.
* Google API - To create and present a map
* Telegram bot - user’s selection within the telegram bot

## Technologies

**Kafka**

Apache Kafka is a distributed messaging system providing fast, highly scalable and redundant messaging through a pub-sub model (publish-subscribe). All incoming data is first placed in Kafka and all outgoing data is read from Kafka. Kafka is the most poplar platform for streaming data and it is an [open source](https://en.wikipedia.org/wiki/Open-source_software) software platform. Being open source means that it is essentially free to use and has a large network of users and developers who contribute towards updates, new features and offering support for new users.

We use it for real-time publish-subscribe data from our sources.

Other alternatives: pub-sub or Amazon Kinesis

**Spark structure streaming:**

Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine.Can use the [Dataset/DataFrame API](https://spark.apache.org/docs/2.2.0/sql-programming-guide.html) in Scala, Java, Python or R to express streaming aggregations, event-time windows, stream-to-batch joins, etc.

Good connection to Kafka, works in real time. Uses to pull data from Kafka (request and stocks prices) and manipulate it and push it back to Kafka (for another topic)

**MySQL**

**Hive**

**Tableau**

Interactive data visualization software that focused on business intelligence.

Will be used in this project as a visualization tool for creating our on-demand dashboard base on our BI models.

**Slack**

**Airflow**

Apache Airflow is an open-source tool to programmatically author, schedule, and monitor workflows. It is one of the most robust platforms used by Data Engineers for orchestrating workflows or pipelines. You can easily visualize your data pipelines' dependencies, progress, logs, code, trigger tasks, and success status.

Other alternatives: Luigi and Oozie