

## סיפור CISCO – Operation Black Ledger

בסוף שנת 2025 התקבלו ביחידת המודיעין הטכנולוגי של צה"ל התרעות ממספר מקורות מודיעיניים בלתי-תלוים, המעידים על ניסיון מתמשך של ארגון חמאס לבצע רכישות נשק וצדד לחימה בהיקפים חריגיים, באמצעות רשותות אזרחיות, חברות קש וגורמים פיננסיים עקייפים.

על פי המידע, הרכישות אינן מתבצעות ישירות דרך גורמים צבאיים מזוהים, אלא מוסווות כעסקאות מסחריות לגיטימיות.

במסגרת פעילות איסוף מודיעינית, הועבר ליחידה מאגר נתונים גלמי של עסקאות, שמקורו בגורם אזרחי חיצוני (לוגיסטי / פיננסי), אשר כולל מידע על **לקוחות והזמןות שבוצעו על ידם**.

לכוארה, מדובר ברישום רכישות סטנדרטית לחלוין — צו שכל ארגון מסחרי גדול עשוי להחזיק. עם זאת, הערכה מודיעינית מצביעה על כך שבתוך הנתונים **מסתתרים דפוסים חריגיים**, שעשוים להעיד על:

- היררכות מבצעית
- צבירת אמצעי לחימה
- העברות כספים לא פרופורציונליות
- ניסיונות הסואנה מכוננים

## מטרת המערכת

המשימה שלכם היא לבנות **מערכת חוקית-אנגליתית בצד השרת**, שמטרתה לנתח את מאגר הרכישות ולהפיק ממנו תובנות ראשונות.

המערכת אינה נועדה להפעיל שירות גורם מסוים או לקבוע מסקנות חד-משמעות. מטרתה היא **לאთר חריגות ודפוסים חשודים, כלומר לסמן עסקאות או ל��וחות שדורשים המשך חוקיה**.

כל שאלתה שתתמוש בשירות האנגליתיקה של מערכת מייצגת **פעולה מודיעינית ממוקדת**.

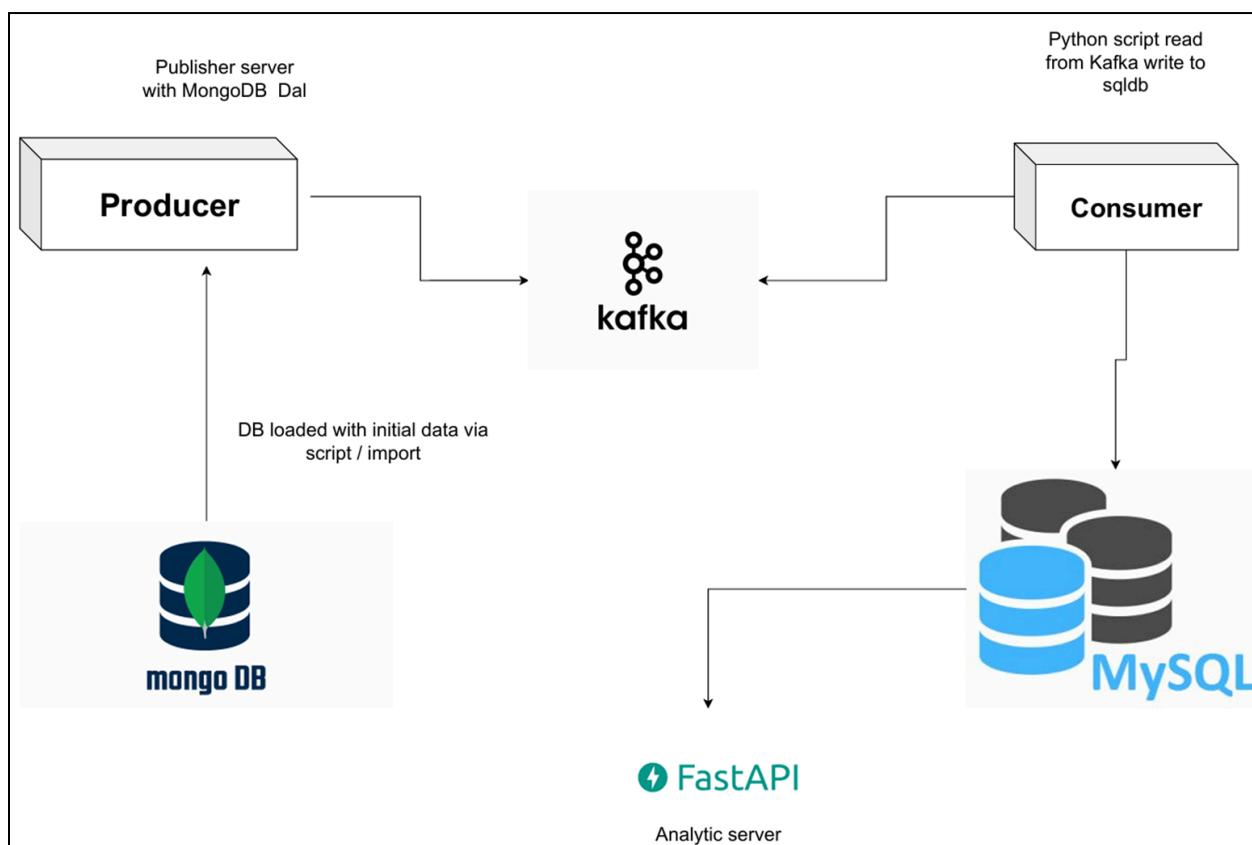
כל פלט המוחזר מהמערכת מהווה **קזה חוט מודיעיני**, עשוי להתחבר למידע נוסף בשלבים מתקדמים יותר של החוקיה.

## סקירה כללית של המערכת (High-Level Overview)

המערכת בנויה כצינור נתונים חקירתי המורכב ממספר שירותים מופרדים:

1. נתונים ריכשות נטענים תחילה למסד נתונים מבוסס MongoDB ומשמשים כמאגר גולמי.
2. רכיב פרסום (Publisher) קורא את הנתונים ומפרסם אותם כאירועים ל-Kafka.
3. רכיב צרכן (Consumer) קורא את האירועים מ-Kafka ומזריק אותם למסד נתונים רלציוני (SQL).
4. מסד נתונים רלציוני מסווג MySQL המיועד לניטוח אנליטי.
5. שרת אנליטי מבוסס FastAPI החשוף למשתמש ומאפשר ביצוע שאלות SQL חקירתיות על הנתונים המעובדים.

להלן תרשيم זרימה המתאר את מבנה המערכת וזרימת הנתונים בין רכיביה:



## עזרים מותרים

במהלך המבחן **חל איסור מוחלט** על:

- שימוש ב-**Google** או כל מנוע חיפוש שאין כלול באחד מן המקורות המותרים
  - שימוש ב-**כליל AI מכל סוג** (ככל אך לא מוגבל ל-**ChatGPT, Copilot, Claude, Gemini** וכדומה)
  - קבלת עזרה מאדם אחר
- 

### ✓ עזרים מותרים

מותר להשתמש **אר וرك** במקורות הבאים:  
**Python**

- תיעוד רשמי:  
[Python Docs](#)

[PyMongo - Documentation](#)

**MongoDB**

- תיעוד רשמי:  
[MongoDB Documentation - Homepage](#)

**MySQL**

- [MySQL Documentation](#)

**Kafka**

- [confluent kafka API — confluent-kafka 2.13.0 documentation](#)  
(Kafka Crash Course (Nana

[https://gitlab.com/twn-youtube/kafka-crash-course/-/blob/main/README.md?ref\\_type=heads](https://gitlab.com/twn-youtube/kafka-crash-course/-/blob/main/README.md?ref_type=heads)

**Docker**

- תיעוד רשמי:  
[Docker Docs](#)

Docker Hub (Images, Tags, Usage):  
[Docker Hub](#)

**FastAPI**

- תיעוד רשמי:  
[FastAPI](#)

**Git**

- תיעוד רשמי:  
[Git - Reference](#)

**YAML**

YAML (בדיקות תקינות קבצי YAML):  
[The YAML Validator](#)

#### אתרי עזר כלליים

W3Schools: •

[W3 Schools](#)

GeeksforGeeks: •

[GeeksforGeeks](#)

Stack Overflow: •

[/https://stackoverflow.com](https://stackoverflow.com)

---

## דרישות מימוש (Implementation Requirements)

עליכם למשם מערכת חוקית מבוססת אירעים, הכוללת מספר שירותים נפרדים, בהתאם לדרישות המפורטות להלן.

המימוש חייב לעמוד בדרישות כפי שהן מוגדרות בסעיף זה, ללא שינוי ארכיטקטורה או זרימת נתונים.

---

## מסד נתונים גולמי – MongoDB

בשלב זה תקבלו קובץ JSON מוקן בשם  
`suspicious_customers_orders.json`

הקובץ כולל **דאטה מעורב (mixed data)** – גם מסמכים `Customers` וגם מסמכים `Orders` – כולם באותו קובץ.

#### מבנה מסד הנתונים

עליכם ליצור ב-MongoDB:

- **Database אחד בלבד**
- **Collection אחד בלבד**

כל המסמכים (גם `Customers` וגם `Orders`) יישמרו **באותו Collection**.

חשוב:

מבנה המסמכים ושמות השדות חייבים להישמר בדיק כפי שהם מופיעים בקובץ ה-JSON המקורי.

אין לשנות שמות שדות, אין להוסיף שדות ואין להסיר שדות.

---

## טעינת הנתונים למסד

את קובץ ה-JSON עליים לטעון אל MongoDB לפני תחילת העבודה. מרגע שהנתונים נטען למסד – משם תתבצע כל העבודה בהמשך הפרויקט. ניתן לטעון את הנתונים בכל דרך שתבחרו. הבחירה היא שלכם. לדוגמה:

- **MongoDB Compass (דרך ה-UI)** – ייצור Database ו-Collection ולאחר מכן ביצוע Import לקובץ ה-JSON.
- **קוד Python** – כתיבת סקריפט שמתחבר ל-MongoDB ומכוון את הנתונים.
- **Mongo Shell / mongosh**
- **Docker / Docker Compose** – כולל שימוש ב-`mongoimport` או סקריפט `init`.

העיקר: הנתונים חייבים להיות נתונים במלואם למסד, ומתחם תבצעו את כל השאלות והউবودিম בהמשך.

## Kafka – פרטום נתונים ל-Publisher .2

עליכם למשריך **Publisher**, האחראי על קריית הנתונים ממסד MongoDB ושליחתם ל-Kafka בצורה המדמה זרימת נתונים (Streaming).

דרישות פונקציונליות:

- מסד MongoDB כולל **Collection אחד בלבד**, המכיל מסמכים מסוגים שונים (Orders ו-Customers) יחד.
- ה-**Publisher** חייב לקרוא את הנתונים מתוך Collection באמצעות שליפה מדורגת (Batch Processing / Pagination).
- הקריאה תתבצע במסנות של **30–50 מסמכים בכל פעם** (יש לבחור מספר קבוע בטוווח זה, ויש להימנע אליו באופן עקבי).
- לאחר שליפת כל Batch: •

- יש לעبور על המסמכים אחד-אחד.
  - כל מסמך ישלח ל-**Event Kafka** נפרד.
  - בין שליחת מסמך אחד לשני יש להמתין חצי שנייה (0.5 שניות).
  - התהילה ימשך עד לסיום שליחת כל המסמכים בקובץ.
  - כל האירועים ישלחו ל-**Topic Kafka** יחיד ב-**Topic Kafka**.
  - הנתונים יישלחו ל-Kafka במבנה המקורי כפי שהם נשמרים ב-**MongoDB**, ללא שינוי שמות שדות ולא שינוי מבנה.
- רכיב זה יכול להיות ממומש כשרת אפליקטיבי או כסקריפט Python עצמאי.
- 

### 3. Kafka – תיוך אירועים

Kafka משמשת כשכבה תיון בין שירות ה-**Producer** ושירות ה-**Consumer**.

- כל הנתונים בין שני השירותים הללו חייבים לעبور דרך Kafka
- כל עיבוד נתונים לאחר שלב הפרסום יבוצע אך ורק דרך **Consumer**

---

### 4. Consumer – עיבוד נתונים וכתיבה ל-SQL

עליכם למשריך **Consumer**, האחראי על קראית האירועים מ-Kafka והזרקתם למסד נתונים רלוונטי.

דרישות פונקציונליות:

- ה-**Consumer** חייב לקרוא אירועים מ-Kafka עבור לקוחות (טבלת **customers**) והזמנות (טבלת **orders**)
- ה-**Consumer** אחראי על:

- ייצור מסד הנתונים הרלציוני
  - ייצור הטבלאות הנדרשות
  - הכנסת הנתונים למסד הנתונים תוך מילוי לטבלאות הרלוונטיות לפי סוג המידע
- יש ליצור טבלה נפרדת ללקחות וטבלה נפרדת להזמנות.
  - יש להגדיר:
    - Primary Key בטבלת הלקחות
    - Customer ID בטבלת ההזמנות, המקשר לטבלת הלקחות דרך השדה ID השדה (mbhigint השדות).
- ה-Consumer ממומש כסקריפט Python עצמאי והוא חסוף למשתמש.
- 

## 5. מסד נתונים רלציוני – SQL

- מסד הנתונים הרלציוני משמש לשכבה האנליטיקת של המערכת.
- הנתונים במסד זה מיועדים לניטוח בלבד על ידי השרת האנליטי עליו מפורט מטה.
- 

## 6. שרת אנליטי – Fast API

- עליכם למשר **שרת אנליטי**, אשר מתקשר עם מסד הנתונים הרלציוני (MySQL) ומהווה את שכבת הניטוח של המערכת.
- שרת זה חסוף למשתמש הקצה, ומשמש כմמשק דרכו מתבצעות שאלות אנליטיות על הנתונים המעובדים.
- 

## דרישות פונקציונליות

- השירות חייב להתחבר למסד הנתונים MySQL באמצעות קוד.
- יש למשר חיבור מסודר ומוגדר למסד הנתונים (אין להשתמש בחיבורים אד-הוק בתוך כל ראות).

- כל Endpoint במערכת מייצג שאלה אנאליטית / עסקית נפרדת.
- כל ראיות חייב:
  - לבצע שאילתת מול מסד הנתונים
  - להחזיר תוצאה בפורמט JSON תקני

השרת מיועד **לקראיה וניתוח בלבד**.

פירוט האנדפונטים והשאלות האנאליטיות הנדרשות יפורט בהמשך המסמך.

---

### מבנה קוד נדרש בשרת האנאליטי

עליכם לחלק את קוד השירות בצורה מודולרית וברורה, בהתאם לעקרונות הפרדת אחריות (Separation of Concerns).

מבנה מינימלי נדרש:

- `main.py` אחראי על אתחול האפליקציה והרצת השירות.
  - `connection.py` אחראי על יצירת החיבור למסד הנתונים MySQL.
  - `(dal.py)` (Data Access Layer) אחראי על מימוש הלוגיקה של השאילות מול מסד הנתונים.
  - אין לכתוב לוגיקה SQL ישירות בתוך הראותים.
  - `(אופציוני) routes.py` קובץ נפרד להגדלת הראותים וה קישור שלהם לפונקציות מתוך ה-DAL.
- .`include_router` ניתן להשתמש ב-Router API ולהזכיר main באמצעות

אם בחרתם שלא ליצור קובץ `routes.py`, ניתן למש את הראותים ישירות בתוך `main.py`, אך עדין יש לשמור על הפרדה ברורה בין שכבת הראותים לשכבה ה-DAL.

---

## דרישות Endpoints אונלייטיים

עליכם למש בשירות ה-`FastAPI` את Endpoints העונים על השאלות העסקיות המופיעות בחלק זה.

שיםו לב כי כתובות הניטובים הן **דרישת חובה** ויש להקפיד על נושא זה שעליו גם יינתן ציון.

(זכרו כי שירותי נוספים ביחידה תלויים בגישה מוגדרת וברורה לנוטרים, וחירגה מהגדירה זו עשויה להוביל לפגיעה בעשייה המודיעינית, ובציוון 😊)

### Route 1

`GET /analytics/top-customers`

#### שאלה עסקית

מי הם עשרת הלוקוחות בעלי כמות ההזמנות הגבוהה ביותר במערכת?

המטרה: לזהות את הלוקוחות הפעילים ביותר מבחינת מספר הזמנות.

---

### Route 2

`GET /analytics/customers-without-orders`

#### שאלה עסקית

אילו לקוחות קיימים במערכת אף מעולם לא ביצעו הזמנה?

המטרה: לזהות לקוחות לא פעילים.

---

### Route 3

`GET /analytics/zero-credit-active-customers`

#### שאלה עסקית

אילו לקוחות בעלי מסגרת אשראי 0 ביצעו בפועל הזמנות במערכת?

המטרה: לזהות חוסר התאמה בין מסגרת האשראי לבין פעילות בפועל.

---

## מבנה פרויקט נדרש

הפרויקט חייב להיות מאורגן בצורה מודולרית וברורה, כך שכל שירות מופרד לוגית ופיזית, אך מנווה תחת Docker Compose אחד מרכזי.

```
project-root/
  └── docker-compose.yml
  └── .env

  └── producer/
      ├── app/
      │   ├── main.py
      │   ├── mongo_connection.py
      │   └── kafka_publisher.py
      ├── Dockerfile
      └── requirements.txt

  └── consumer/
      ├── app/
      │   ├── main.py
      │   ├── kafka_consumer.py
      │   ├── mysql_connection.py
      │   └── schema.sql (אופציונלי)
      ├── Dockerfile
      └── requirements.txt

  └── analytics-api/
      ├── app/
      │   ├── main.py
      │   ├── connection.py
      │   ├── dal.py
      │   └── routes.py (אופציונלי)
      ├── Dockerfile
      └── requirements.txt

  └── shared/ (אופציונלי)
      └── models/

  └── sql-init/
      └── init.sql (אופציונלי)
```

**הסבר קבצים:**

## **(חובה) docker-compose.yml**

קובץ זה חיב:

- להרים את כל השירותים:
  - MongoDB
  - Kafka
  - MySQL
  - Producer
  - Consumer
  - Analytics API
- להגדיר רשת משותפת
- להגדיר volumes לפי הצורך
- להגדיר משתני סביבה

המערכת כוללת צריכה באמצעות הפקודה:

```
docker compose up --build
```

---

## **/producer**

מכיל את קוד ה-Publisher:

- אחראי לקרוא ממונגו
- להזרים לפקה
- אינם מכיל לוגיקה SQL
- אינם מתקשר ל-MYSQL

קוד חייב להיות מופרד מה-Consumer.

---

## **/consumer**

מכיל את קוד ה-Consumer:

- קורא מפקה
- ממין לפי type
- יוצר טבלאות במידת הצורך
- מכניס נתונים ל-MYSQL
- אחראי על סכימת הנתונים

אין בו קוד API FastAPI.

---

## /analytics-api

מכיל את שרת ה-`FastAPI` האנליטי:

- מתחבר ל-MySQL בלבד
- מכיל DAL
- מכיל Endpoints אנליטיים בלבד
- אינם מבצע שינוי נתוני

## דרישות עבודה עם Git ו-`Branches` (חובה)

עבודה נכונה עם Git היא חלק בלתי נפרד מהציון.  
אי עמידה בדרישות ניהול גרסאות (`Branches / Commits / Merges`) תגרור הפקחת ניקוד, גם אם המערכת  
עובדת.

---

### 1 חובה `Branches`

בריפזיטורי חיבים להופיע לפחות הפחות `Branches` הבאים:

- `main`
- `dev`
- `feature/streaming-pipeline`
- `feature/analytics-api`

שימוש לב: **הבדיקה תתבצע על התוכן ב-`main` בלבד, אך קיומ שאר הבראנץ'ים יבדק וכן שם מכילים קוד רלוונטי.**

---

## 2) הגדרות ותפקידים

### main

- מכיל אך ורק קוד סופי, יציב ועובד.
- אסור לבצע פיתוח ישיר על `main`.
- אסור לבצע `commit` ישיר ל-`main`.
- רק קוד שכבר בדיקה ומיזוג מ-`dev` יופיע בו.

### dev

- העבודה המרכזית.
- משמש לאינטגרציה בין רכיבי המערכת לפני מיזוג ל-`main`.
- אין למיזוג ל-`main` קוד שלא עבר דרך `dev`.

### feature/streaming-pipeline

- Branch לפיתוח רכיבי הזרימה (Publisher + Consumer) ותשתיות ה-pipeline:
  - קרייה מ-MySQL → פרסום ל-Kafka → צריכה וכתיבה ל-MySQL
- כל הפיתוח של רכיבים אלו יבוצע בתחום Branch זה, ולא ישירות על `dev`.

### feature/analytics-api

- Branch לפיתוח שרת ה-UI FastAPI האנליטי (Analytics API) וה-DAL.
  - כל העבודה על הראותים והשאלותות תבוצע בתחום Branch זה.
-

### 3) כללי עבודה (Workflow)

- ✓ העבודה מתחילה מ-`dev` וונפתחים ממנו Branchים מסוג `Branch` מסוג `dev` •
  - ✓ כל פיצ'ר מפותח ב-`Branch` המתאים:
    - `feature/streaming-pipeline`
    - `feature/analytics-api`
  - ✓ לאחר סיום פיצ'ר ובדיקה:
    - מבצעים `merge` חוזרת ל-`dev`
  - ✓ לאחר שכל המערכת עבדה מקופה לפחות:
    - מבצעים `merge` מ-`dev` ל-`main`
  - ✓ בסוף המבחן, `main` חייב להכיל מערכת עובדת בהתאם לדרישות.

---

### 4) קומיטים (Commits) – דרישות חובה

עליכם להציג היסטוריה של עבודה מדורגת וברורה:

- קומיטים קטנים ולוגיים (לא "הכל בבנת אחת") •
  - הודיעו `commit` ברורות ותיאוריות
  - לא: `fix`, `update`, `changes`
- `add kafka producer`, `create mysql tables`, `add analytics route 1`:
  - אין להעלות את כל הפרויקט ב-`commit` אחד גדול
  - עבודה ללא היסטוריה קומיטים תקינה תיחס כעבודה לא מקצועית ותגרור הפחתת ניקוד

## 5) מה נבדק?

- שהתוכן הסופי נמצא ב-`main` ועומד בדרישות
- קיום `dev`
- קיום שני Branchים נוספים: `feature/streaming-pipeline` ○ `feature/analytics-api` ○
  - תהליך עבודה מסודר: פיתוח ב-`feature` → `merge` ל-`dev` → `merge` ל-`main`
  - איקות היסטורית הקומיטים (כמויות, היגיון, ניסוח)
  - **איקות הקוד, שמירה על YOURSELF - DON'T REPEAT YOURSELF, קוד נקי**