

# סיפור כיסוי – Operation Black Ledger

בסוף שנת 2025 התקבלו ביחידת המודיעין הטכנולוגי של צה"ל התרעות ממספר מקורות מודיעיניים בלתי-תלויים, המעידים על ניסיון מתמשך של ארגון חמאס לבצע **רכישות נשק וציוד לחימה בהיקפים חריגים**, באמצעות רשתות אזרחיות, חברות קש וגורמים פיננסיים עקיפים.

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

במסגרת פעילות איסוף מודיעינית, הועבר ליחידה **מאגר נתונים גולמי של עסקאות**, שמקורו בגורם אזרחי חיצוני (לוגיستي / פיננסי), ואשר כולל מידע על **לקוחות והזמנות שבוצעו על ידם**.

לכאורה, מדובר ברשימת רכישות סטנדרטית לחלוטין — כזו שכל ארגון מסחרי גדול עשוי להחזיק. עם זאת, הערכה מודיעינית מצביעה על כך ש**בתוך הנתונים מסתתרים דפוסים חריגים**, שעשויים להעיד על:

- היערכות מבצעית
- צבירת אמצעי לחימה
- העברות כספים לא פרופורציונליות
- ניסיונות הסוואה מכוונים

---

## מטרת המערכת

המשימה שלכם היא לבנות **מערכת חקירתית-אנליטית בצד השרת**, שמטרתה לנתח את מאגר הרכישות ולהפיק ממנו תובנות ראשוניות.

המערכת אינה נועדה להפיל ישירות גורם מסוים או לקבוע מסקנות חד-משמעיות. מטרתה היא **לאתר חריגות ודפוסים חשודים, כלומר לסמן עסקאות או לקוחות שדורשים המשך חקירה**

כל שאילתה שתמומש בשירות האנליטיקה של מערכת מייצגת **פעולה מודיעינית ממוקדת**.

כל פלט המוחזר מהמערכת מהווה **קצה חוט מודיעיני**, שעשוי להתחבר למידע נוסף בשלבים מתקדמים יותר של החקירה.

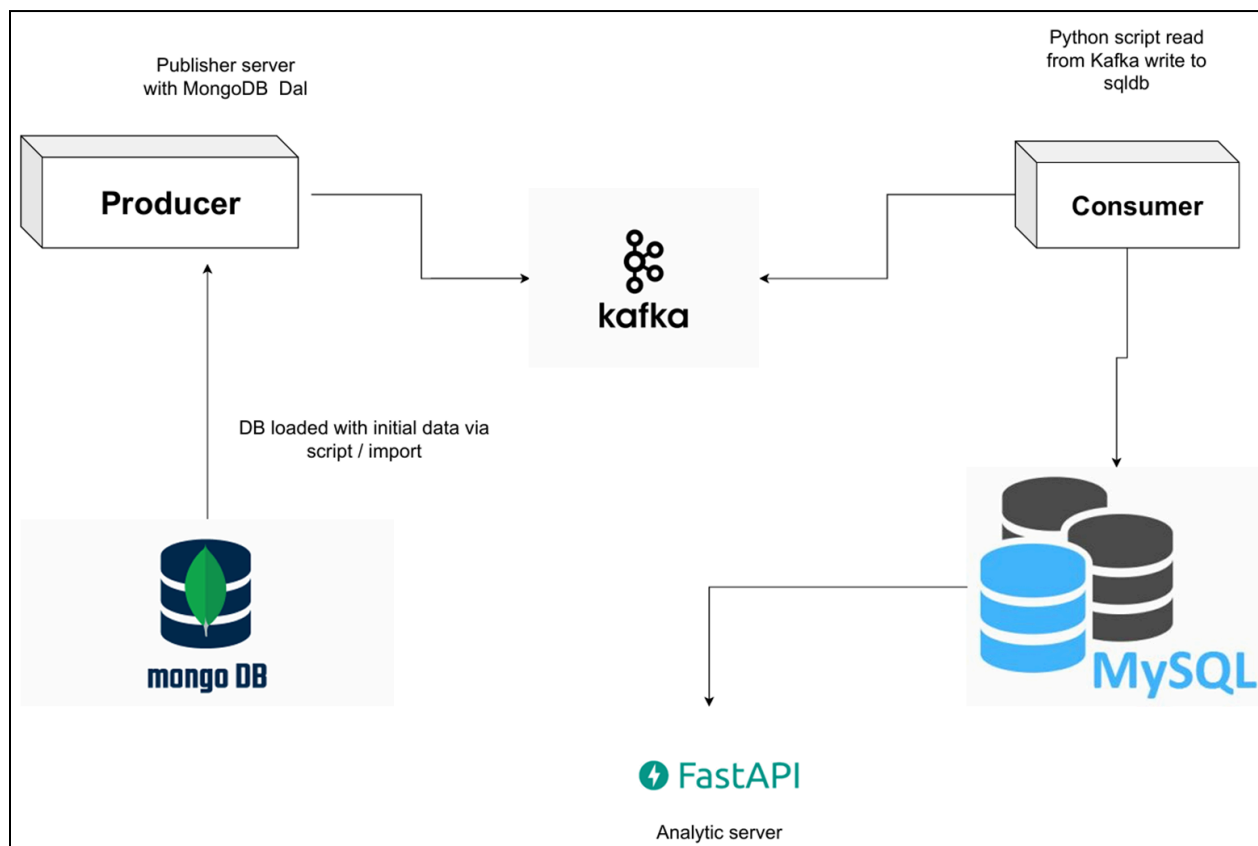
---

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

המערכת בנויה כצינור נתונים חקירתי המורכב ממספר שירותים מופרדים:

1. נתוני רכישות 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 מסוג MySQL המיועד לניתוח אנליטי.
5. שרת אנליטי מבוסס FastAPI חשוף למשתמש ומאפשר ביצוע שאילתות SQL חקירתיות על הנתונים המעובדים.

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



## עזרים מותרים

במהלך המבחן חל איסור מוחלט על:

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

---

## עזרים מותרים

מותר להשתמש אך ורק 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 Lint (בדיקת 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 לממש מערכת חקירתית מבוססת אירועים, הכוללת מספר שירותים נפרדים, בהתאם לדרישות המפורטות להלן. המימוש חייב לעמוד בדרישות כפי שהן מוגדרות בסעיף זה, ללא שינויי ארכיטקטורה או זרימת נתונים.

---

## מסד נתונים גולמי – MongoDB

בשלב זה תקבלו קובץ JSON מוכן בשם

`suspicious_customers_orders.json`

הקובץ כולל דאטה מעורב (mixed data) – גם מסמכי `Customers` וגם מסמכי `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 בדיוק כפי שהם מופיעים בקובץ ה-JSON המקורי.

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

---

## טעינת הנתונים למסד

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

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

העיקר:

הנתונים חייבים להיות נטענים במלואם למסד, ומתוכם תבצעו את כל השאילתות והעיבודים בהמשך.

## 2. Publisher – פרסום נתונים ל-Kafka

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

**דרישות פונקציונליות:**

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

- יש לעבור על המסמכים אחד-אחד.
  - כל מסמך יישלח ל-Kafka כ-Event נפרד.
  - בין שליחת מסמך אחד למשנהו יש להמתין חצי שנייה (0.5 שניות).
  - התהליך יימשך עד לסיום שליחת כל המסמכים בקולקשן.
  - כל האירועים יישלחו ל-Topic יחיד ב-Kafka.
  - הנתונים יישלחו ל-Kafka במבנה המקורי כפי שהם נשמרים ב-MongoDB, ללא שינוי שמות שדות וללא שינוי מבני.
- רכיב זה יכול להיות ממומש כשרת אפליקטיבי או כסקריפט Python עצמאי.
- 

### 3. Kafka – תיווך אירועים

- Kafka משמשת כשכבת תיווך בין שירות ה-Producer ושירות ה-Consumer.
- כל הנתונים בין שני השירותים הללו חייבים לעבור דרך Kafka
  - כל עיבוד נתונים לאחר שלב הפרסום יתבצע אך ורק דרך Consumer
- 

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

עליכם לממש רכיב **Consumer**, האחראי על קריאת האירועים מ-Kafka והזרקתם למסד נתונים רלציוני.

דרישות פונקציונליות:

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

- יצירת מסד הנתונים הרלציוני
- יצירת הטבלאות הנדרשות
- הכנסת הנתונים למסד הנתונים תוך מיון לטבלאות הרלוונטיות לפי סוג המידע

- יש ליצור **טבלה נפרדת** ללקוחות וטבלה נפרדת להזמנות.

- יש להגדיר:

- Primary Key בטבלת הלקוחות

- Foreign Key בטבלת ההזמנות, המקושר לטבלת הלקוחות דרך השדה Customer ID

- מבנה הטבלאות חייב להיות בהתאמה למבנה הנתונים המקוריים (מבחינת השדות).

ה-Consumer ממומש כסקריפט Python עצמאי ואינו חשוף למשתמש.

## 5. מסד נתונים רלציוני – SQL

מסד הנתונים הרלציוני משמש לשכבת האנליטיקה של המערכת.

- הנתונים במסד זה מיועדים לניתוח בלבד על ידי השרת האנליטי עליו מפורט מטה.

## 6. שרת אנליטי – Fast API

עליכם לממש שרת Fast API אנליטי, אשר מתקשר עם מסד הנתונים הרלציוני (MySQL) ומהווה את שכבת הניתוח של המערכת.

שרת זה חשוף למשתמש הקצה, ומשמש כממשק דרכו מתבצעות שאילתות אנליטיות על הנתונים המעובדים.

## דרישות פונקציונליות

- השרת חייב להתחבר למסד הנתונים MySQL באמצעות קוד.
- יש לממש חיבור מסודר ומוגדר למסד הנתונים (אין להשתמש בחיבורים אד-הוק בתוך כל ראוט).

- כל Endpoint במערכת מייצג **שאלה אנליטית** / **עסקית נפרדת**.

- כל ראוט חייב:

- לבצע שאילתה מול מסד הנתונים

- להחזיר תוצאה בפורמט JSON תקני

השרת מיועד **לקריאה וניתוח בלבד**.

פירוט האנדפוינטים והשאלות האנליטיות הנדרשות יפורט בהמשך המסמך.

---

## מבנה קוד נדרש בשרת האנליטי

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

מבנה מינימלי נדרש:

- `main.py`  
אחראי על אתחול האפליקציה והרצת השרת.
- `connection.py`  
אחראי על יצירת החיבור למסד הנתונים MySQL.
- `dal.py` (Data Access Layer)  
אחראי על מימוש הלוגיקה של השאילתות מול מסד הנתונים.  
אין לכתוב לוגיקה SQL ישירות בתוך הראוטים.
- `routes.py` (אופציונלי אך מומלץ)  
קובץ נפרד להגדרת הראוטים והקישור שלהם לפונקציות מתוך ה-DAL.  
ניתן להשתמש ב-API Router ולחבר אותו ל-main באמצעות `include_router`.

אם בחרתם שלא ליצור קובץ `routes.py`, ניתן לממש את הראוטים ישירות בתוך `main.py`, אך עדיין יש לשמור על הפרדה ברורה בין שכבת הראוטים לשכבת ה-DAL.

---



## דרישות Endpoints אנליטיים

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

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

(זכרו כי שירותים נוספים ביחידה תלויים בגישה מוגדרת וברורה לנתונים, וחריגה מהגדרה זו עשויה להוביל לפגיעה בעשייה המודיעינית, ובציון 😊)

### Route 1

GET /analytics/top-customers

#### שאלה עסקית

מי הם עשרת הלקוחות בעלי כמות ההזמנות הגבוהה ביותר במערכת?

המטרה: לזהות את הלקוחות הפעילים ביותר מבחינת מספר הזמנות.

---

### Route 2

GET /analytics/customers-without-orders

#### שאלה עסקית

אילו לקוחות קיימים במערכת אך 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 מנוהל 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:

### docker-compose.yml (חובה)

קובץ זה חייב:

- להרים את כל השירותים:

- MongoDB

- Kafka

- MySQL

- Producer

- Consumer

- Analytics API

- להגדיר רשת משותפת

- להגדיר volumes לפי הצורך

- להגדיר משתני סביבה

המערכת כולה צריכה באמצעות הפקודה:

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

---

#### /producer

מכיל את קוד ה-Publisher:

- אחראי לקרוא ממוגו

- להזרים לקפקה

- אינו מכיל לוגיקה SQL

- אינו מתקשר ל-MYSQL

הקוד חייב להיות מופרד מה-Consumer.

---

#### /consumer

מכיל את קוד ה-Consumer:

- קורא מקפקה

- ממין לפי type

- יוצר טבלאות במידת הצורך

- מכניס נתונים ל-MYSQL

- אחראי על סכימת הנתונים

אין בו קוד FastAPI.

---

## /analytics-api

מכיל את שרת ה-FastAPI האנליטי:

- מתחבר ל-MYSQL בלבד
- מכיל DAL
- מכיל Endpoints אנליטיים בלבד
- אינו מבצע שינויי נתונים

## דרישות עבודה עם Git ו-Banches (חובה)

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

---

### 1 Branches חובה

בריפוזיטורי חייבים להופיע לכל הפחות ה-Banches הבאים:

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

שימו לב: הבדיקה תתבצע על התוכן ב-main בלבד, אך קיום שאר הבראנצ'ים ייבדק וכן שהם 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 שעבר בדיקה ומיזוג מ-dev יופיע בו.

### dev

- Branch העבודה המרכזי.
- משמש לאינטגרציה בין רכיבי המערכת לפני מיזוג ל-main.
- אין למזג ל-main קוד שלא עבר דרך dev.

### feature/streaming-pipeline

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

### feature/analytics-api

- Branch לפיתוח שרת ה-FastAPI האנליטי (Analytics API) וה-DAL.
  - כל העבודה על הראוטים והשאלות תתבצע בתוך Branch זה.
-

### (3) כללי עבודה (Workflow)

- העבודה מתחילה מ-**dev** ונפתחים ממנו Branchים מסוג **\*/feature** ✓
  - כל פיצ'ר מפותח ב-Branch המתאים: ✓
  - **feature/streaming-pipeline**
  - **feature/analytics-api**
  - לאחר סיום פיצ'ר ובדיקתו: ✓
  - מבצעים **merge** חזרה ל-**dev**
  - לאחר שכל המערכת עובדת מקצה לקצה: ✓
  - מבצעים **merge** מ-**dev** ל-**main**
  - בסוף המבחן, **main** חייב להכיל מערכת עובדת בהתאם לדרישות. ✓
- 

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

עליכם להציג היסטוריית עבודה מדורגת וברורה:

- קומיטים קטנים ולוגיים (לא "הכל בבת אחת")
  - הודעות commit ברורות ותיאוריות
  - לא: **fix, update, changes**
  - כן: **add kafka producer, create mysql tables, add analytics route 1**
  - אין להעלות את כל הפרויקט ב-**commit** אחד גדול
  - עבודה ללא היסטוריית קומיטים תקינה תיחשב כעבודה לא מקצועית ותגרור הפחתת ניקוד
-

## (5) מה נבדק?

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