

Kafka Streams- Event Sourcing ו-AI היברידי מונע אירועים עם

1. מטרת הפרויקט

פרויקט גמר זה יעמיק את הבנה והיישום של ארכיטקטורות מבוזרות ובסיסות אירועים (Event-Driven Architectures) עבור סוכני AI מתקדמים. נהפוך את הבוט למכונית מצב אסינכרונית ומונעת אירועים לחלוטין, שבה Apache Kafka אינו רק "צינור" להעברת הודעות, אלא הופך להיות:

מקור האמת היחיד (Single Source of Truth): כל שינוי במצב הסוכן (קבלת שאילתה, יצירת תוכנית, ביצוע כל, קבלת תוצאה, ניסוח תשובה) נרשם כאירוע בלתי משתנה (Immutable Event) ב-Kafka.

פלטפורמה לעיבוד זרמים עם מצב (Stateful Stream Processing): רכיבים קרייטיים במערכת (כמו Aggregator ו-Orchestrator) יבנו ויעדכו את המצב הפנימי שלהם (למשל, סטטוס Plan עבור conversation ספציפי) ישירות מתוך זרמי האירועים של Kafka. זה מבטיח עמידות לכשלים, סקלabilitות ויכולת שחזור.

מנגנון תזמון מבוזר (Distributed Orchestration Mechanism): במקום לוגיקה טוරית או ריכזית, תזמון הכלים מתבצע על ידי רצף אירועים, כאשר כל אירוע מפעיל את הצעד הבא ב-Pipeline.

בפרויקט זה נתרgal :

- Event Sourcing: מידול מצב המערכת כזרם אירועים.
- Kafka Streams (אוימוש עקרונותיה): בניית רכיבים בעלי מצב (Stateful) המעבדים זרמי אירועים.
- CQRS (Command Query Responsibility Segregation): הפרדה בין פקודות (Requests) לאירועים (Events) ב-Kafka.
- Fault Tolerance & Resilience: המערכת תהיה עמידה לכשלים, תוך יכולת שחזור מצב אוטומטי.
- Microservices Architecture: פיזול המערכת ל-Microservices עצמאיים המבצעים פונקציה אחת בלבד ומתקשרים באמצעות Kafka.
- Event-Sourced AI State Machine - 2. ארכיטקטורת המערכת הכללית:

הזרימה הכללית:

- קלט מהמשתמש: `UserQueryReceived` (פקודה: ConversationEventsTopic → Router Agent).
- מפיק אירוע `PlanGenerated` (PlanAgent → ConversationEventsTopic).
- בונה מצב של ה-Plan-n `Plan-n` (Orchestrator Agent → ConversationEventsTopic).
- פולט פקודות `ToolCommandsTopic` (ToolWorkers → ToolInvocationRequested).
- מפעיל פעולה, מבצעים פעולה, ופולטים אירוע `ToolInvocationResulted` (ToolWorkers → ConversationEventsTopic).
- מעדכן את מצב ה-Plan, מפעיל עדדים נוספים, או מפיק אירוע `PlanCompleted` (Orchestrator Agent → ConversationEventsTopic).
- מזמין `PlanCompleted` (Aggregator Agent → ConversationEventsTopic).
- פקודה `SynthesizeFinalAnswerRequested` (UserInterface → UserCommandTopic).
- מזמין `Synthesis Worker` (UserCommandTopic → Synthesis Worker).
- מונסח תשובה ופולט אירוע `FinalAnswerSynthesized` (Synthesis Worker → ConversationEventsTopic).
- מציג את התשובה הסופית למשתמש (User Interface).

3. דרישות פונקציונליות וארქיטקטוניות

A. Kafka Core Infrastructure & Schema Registry

הקמת Kafka: הרצת Kafka-ו Zookeeper-ב Docker Compose (ומולץ).

:Topics

- קולט מהמשתמש ופקודות פנימיות למערכת. user-commands
- conversation-events: Event Log: ה-Event הראשי. כל אירוע שינו מצב הקשרים לאינטראקציה של סוכן (PlanGenerated, ToolInvocationResulted, PlanStepCompleted, FinalAnswerSynthesized).
- tool-invocation-requests: בקשות (פקודות) ספציפיות לביצוע כל'.
- dead-letter-queue: לטיפול בהודעות שלא עובדו בהצלחה.
- Schema Registry (קונספוטואלי/מימוש פשוט): האגדיר סכימות Avro/JSON לכל סוג אירוע ב-user-commands-conversation-events.
- conversation-events-ChiBIM: קודמו ChiBIM לודא שהם מציינים לסכימות אלו. (לדוגמה, שמרו קבצי .sozn המגדירים את הסכמה עבור כל אירוע).

B. User Interface & Command Producer (Node.js)

:user-interface.ts

- קולט קולט מהמשתמש (CLI).
- יוצר conversationId ייחודי (UUID), ו-timestamp.
- בונה אובייקט UserQueryReceived (פקודה).
- Kafka Topic: Producer user-commands-Producer: שולח את הפקודה ל-
- Consumer: משתמש Amazon ל-conversation-events-consumer: FinalAnswerSynthesized עם conversationId התואם, ומציג את התשובה למשתמש.

C. Agent Router (Node.js - Consumer/Producer)

:router.ts

.(UserQueryReceived consumer: מזין ל-user-commands (ספקית לפקודות Conversation Group Router_SYSTEM_PROMPT Router: מפעיל את ה-LLM (Ollama/OpenAI Fallback) המעודכן) כדי לzechot כוונות וליצור JSON plan מורכב.

Kafka Topic: Producer plan JSON-ו conversationId PlanGenerated (כל conversation-events plan -).

D. Stateful Plan Orchestrator (Node.js - Kafka Streams-like Logic)

זהו רכיב קריטי המנהל את המצב של כל plan באופן עמיד וקל-אביל.

:orchestrator.ts

:Topics consumer: מזין לשני Consumer Group

- conversation-events: PlanGenerated-ו ToolInvocationResulted (לאירועים conversation-events).
- tool-invocation-requests: (כדי לודא שהוא לא שולח בקשה כפולה).

.conversationId State Store (Persistent Key-Value Store) בנה/שחרר את מצב ה-Plan עבור כל conversationId. ניתן להשתמש ב-Node.js level-rocksdb (LevelDB, RocksDB), או לנהיל Map בזיכרון אשר מוגבה על ידי יכולת לשחרר את המצב מה-map בעת הפעלה מחדש.

.Key: conversationId

Value: אובייקט המכיל את ה-plan המלא, stepIndex נכון, תוצאות ביןימ, וסטטוס (PENDING, RUNNING, COMPLETED, FAILED).

לוגיקת עיבוד אירועים:

- כאשר מגיע State Store: אתחל את מצב ה-Plan ב-PlanGenerated.
- כאשר מגיע State Store: עדכן את מצב ה-Plan ב-ToolInvocationResulted עם התוצאה.
- :State Store Producer (פקודות): על בסיס מצב ה-Plan המעודכן ב-Producer קבל החלטה על הצעד הבא (כולל החלפת Placeholders).
- שלח פקודה tool-invocation-requests ל-ToolInvocationRequested.
- אם Plan הושלם: שלח אירוע conversation-events-ל-PlanCompleted.
- אם Plan נכשל: שלח אירוע conversation-events-ל-PlanFailed.

h. Stateless Tool Workers (Python & Node.js)

כל כי יהיה Consumer Group עצמאי, המבצע פונקציה אחת ופולט אירוע工具 invocation resulted. הם צריכים להיות Idempotent.

:rag-retriever-worker.py (Python Consumer/Producer)

- .tool: getProductInformation工具 invocation requests עבור Consumer.
- Kafka Topic: conversation-events-ל-ToolInvocationResulted Producer.
- :ilm-inference-worker.ts (Node.js Consumer/Producer)
- tool: generalChat, ragGeneration, AMAZON Lambda consumer.
- .orchestrationSynthesis
- Kafka Topic: conversation-events-ל-ToolInvocationResulted Producer.
- .math-worker.ts, weather-worker.ts, exchange-rate-worker.ts (כג"ל).
- כל worker שולח את התוצאה שלו כ אירוע conversation-events-ל-ToolInvocationResulted Topic.
- Aggregator & Final Synthesis (Node.js - Kafka Streams-like Logic).
- :aggregator.ts (Node.js Consumer/Producer)

- Kafka Topic: conversation-events-ל-Consumer Group (ספציפית לאירוע).
- .(PlanCompleted)
- :State Store (Optional) בנסיבות זה, הוא יכול לבנות את תוצאות ה-Plan על בסיס אירועי זה.
- .PlanCompleted conversationId ToolInvocationResulted
- :Producer שלח פקודה SynthesizeFinalAnswerRequested (עם כל תוצאות הבינימים) ל-Kafka Topic: user-commands.
- :synthesis-worker.ts (Node.js Consumer/Producer)
- Kafka Topic: user-commands-ל-Consumer Group (ספציפית לפקודות).
- .(SynthesizeFinalAnswerRequested)
- מפעיל את ה-ORCHESTRATION_SYNTHESIS_PROMPT עם AI.

.Kafka Topic: conversation-events-> FinalAnswerSynthesized Producer •

4. ניטוח ביצועים, עמידות ובחרת מודלים (Benchmarking & Resilience)

מדדית זמן (End-to-End Latency):

מעקב אירועים: כל אירוע (UserQueryReceived, PlanGenerated, ToolInvocationRequested, ToolInvocationResulted, FinalAnswerSynthesized timestamp-conversationId) ייבחר (ToolInvocationResulted, FinalAnswerSynthesized conversationId) על ידן (של יצירת האירוע). ה-user-interface או שירות נפרד יכול לחשב את זמן ה-End-to-End Latency על ידן. איסוף כל האירועים עבור conversationId נתון.

מדדיהם: End-to-End Latency, Latency per Worker, Throughput (events/sec), Consumer Lag
תרחישי resilience חובה להדגים:

- קריית Worker: הריצו Plan מורכב. כבו את אחד המ-Workers Python Workers באמצעות Python. הציגו כי ה-Plan נתקע. הרימו את המ-Worker שוב. הציגו שה-Plan ממשך מאיפה שהפסיק.
- קריית Orchestrator.ts: כבו את המ-Orchestrator.ts באמצעות orchestrator.ts. הרימו אותו מחדש. הציגו שהוא משחזר את מצבו מה-Plan conversation-events וממשיך את Plan.
- דוגמה: הדגמו את טיפול המ-Workers באירועים כפויים (לדוגמה, על ידי שיגור ידני של EVENTS duplication או ToolInvocationRequested פעמיים).

טבלת המרכיבים ורחבת בדוק README.md (Benchmarking):

רכיב המערכת / תרחיש	מודול (ספק)	זמן עיבוד ממוצע לאירוע (ms)	קצב אירועי מרבי (Events/sec)	איךות / דיקן (1-5)	עלות משוערת
Router (PlanGenerated)	Ollama (Llama3)	?	0
Router (Fallback)	OpenAI GPT-3.5	5	\$
Orchestrator (ToolInvocationRequested)	Stateful Processor	N/A	0
Tool: RAG Retrieval	HF Embedding (Python)	5	0
Tool: LLM Infer (Ollama)	Ollama (Llama3)	?	0
Tool: LLM Infer (OpenAI)	OpenAI GPT-3.5	5	\$
Aggregator (SynthesizeFinalAnswerRequested)	Stateful Processor	N/A	0
Final Synthesis	OpenAI GPT-3.5	5	\$
End-to-End Latency (Complex Plan)	Total over multiple events	5	Total

নিতוח ומסקנות (מורחב ב-README.md):

- הבדיקה את בחירת Kafka כ-Event Store: כיצד שימוש ב-Sourcing משפר את עמידות המערכת, יכולת השחזר ו-Auditability (מעקב אחר כל פעולה)?
 - Stateful Processing Power: הסבירו כיצד aggregator-or-orchestrator ניצלו את זרמי האירועים של Kafka כדי לנהל מצב במצבית ועמידה, במקרה להסתמך על זיכרון נייף או DB חיצוני לכל שינוי.
 - יתרונות CQRS ו-Idempotency: כיצד הפרדת Command/Event והקפדה על Idempotency תורמים לאמיניות המערכת מבוזרת ואסינכרונית?
 - Trade-offs של Event Sourcing: מהם האתגרים בפתרון זה (מורכבות, debugging, Eventual Consistency?)
- шиיפורים עתידיים: הצעות לשיפורים נוספים DSL כמו Kafka Streams (בעבר js.js) Node יש ספריות כמו Confluent Schema Registry, KSQL, Observability tools (kafka-streams Grafana/Prometheus).

5. תוכרי הגשה

- קוד מקור מלא: (散布 לקבצים שונים תחת src/node src/schemas ו-node.).
- מודל אירועים (Schemas): תיקיית src/schemas עם קבצי JSON לכל סוג אירוע ופקודה.
- כל רכיב (router.ts, orchestrator.ts, Workers) יומחש כ-Microservice עצמאי עם Consumer Group משלו.
- prompts/: כל הпромפטים מעודכנים ומופרדים.
- products/: קבצי מוצר.
- docker-compose.yml: להרמת Kafka, Zookeeper, ChromaDB, Ollama, וכל המיקרים.
- README.md: מפורט כנדרש, כולל: דיאגרמת ארכיטקטורה.
- הסבר מפורט על Stateful Stream Processing ו-Event Sourcing בקונטקט של הפרויקט.
- הוראות הפעלה מלאות.
- דוחות Benchmarking, Resilience Analysis ומסקנות מפורטות.
- קובץ לוג לדוגמה (Execution Log):
- לפחות 3 תרחישי Orchestration מורכבים.
- לפחות 2 תרחישי RAG.
- חובה: הדגמה של תרחישי Resilience (קריסה ושחזור).

בצלחה רבה בפרויקט!