אחסון וקטורים בסקייל: Amazon S3 Vectors ו-Amazon OpenSearch

AWS Summit Tel Aviv 2026 ,סיכום סשן — TLV-ANT304

Arik Porat, Senior Solutions Architect, AWS | Liat Iusim, Solutions Architect, AWS
Meitar Ronen, AI Research Lead, Clover Security-ו Shay Ben Haim, Engineering Manager

10 בספטמבר 2026 | משך: כ-40 דקות | הופק מהקלטת הסשן

על המסמך הזה

המסמך מסכם את הסשן TLV-ANT304 מתוך AWS Summit Tel Aviv 2026, במסלול AWS For Data and Analytics. הוא נבנה מתוך ההקלטה המלאה של הסשן ומהסליידים שחולצו מהווידאו עצמו, כך שאפשר לחזור לתוכן בלי לצפות שוב בארבעים הדקות. חותמות הזמן בגוף המסמך מפנות לנקודה המדויקת בהקלטה.

הסשן בנוי משלושה חלקים: הצגה של Amazon S3 Vectors כשירות חדש לאחסון ושליפה של וקטורים, סיפור לקוח מפורט של חברת Clover Security שבנתה מערכת RAG בפרודקשן מעל השירות, וסקירה של חידושי Amazon OpenSearch בעולם החיפוש הווקטורי.

איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. מה הוצג, ומה המסקנה המעשית.
- **פרקים 2–6 — S3 Vectors:** רקע על וקטורים, הבעיה שהשירות בא לפתור, ה-API, האינטגרציות, ומתי לבחור בו.
- **פרקים 7–9 — סיפור הלקוח:** Clover Security — למה הם עברו ל-S3 Vectors, ואיך הם הפכו RAG לאיכותי בפרודקשן.
- **פרקים 10–11 — OpenSearch:** מתי הוא המנוע הנכון, ושלושה חידושים מהשנה האחרונה.
- **פרק 12 — מה לוקחים מכאן:** מסקנות ונקודות להמשך, ואחריו מונחון.

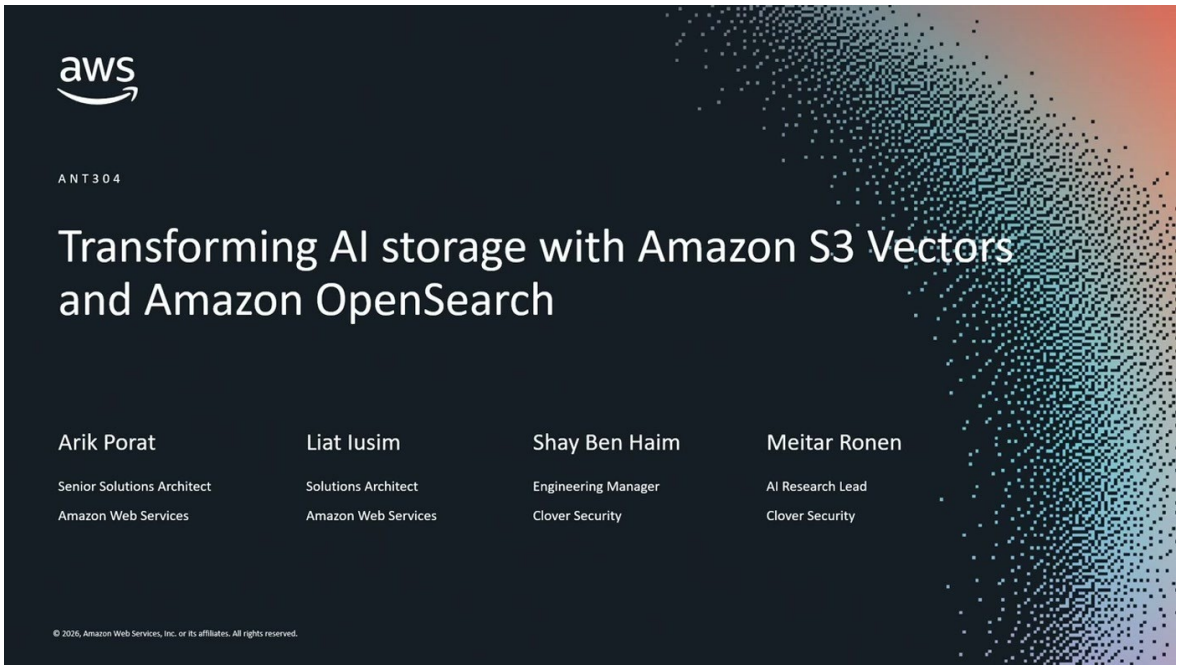
> **הערת מקור** המסמך הופק מתמלול אוטומטי של ההקלטה. התמלול מדויק בקטעים המקצועיים, אך מונחים לועזיים ושמות מוצרים מופיעים בו לעיתים בשיבוש פונטי (למשל "פריימרי צ'ארג'" במקום primary shard) — בגוף המסמך הם נורמלו למונח האנגלי. הציטוטים מובאים כלשונם מן התמלול ואומתו מול חותמת הזמן המצוינת. מספרים שהוצגו על הסליידים צוינו ככאלה.

1. תקציר מנהלים

הסשן הציג שתי תבניות שונות לאותה בעיה — אחסון ושליפה של וקטורים — ולא ניסה להכריע ביניהן. כפי שנאמר בפתיחה, השירותים נבחרו להצגה "כי הם מייצגים שני פאטרנים שונים לדברים דומים, אחד אופטימלי לעלויות וסקל והשני ללטנסי עם עושר ביכולות חיפוש" [0:00].

- **וקטורים הם היום רכיב תשתית סטנדרטי, לא נישה.** הפתיחה הגדירה אותם כ"מרכיב סטנדרטי בפלטפורמת הדאטה שלנו לצד הטבלאות" [0:00], ובהתאם AWS מציעה משפחה שלמה של פתרונות — כולל תמיכה בוקטורים ב-DynamoDB שהוכרזה לאחרונה [4:41].
- **S3 Vectors מוכר חיסכון של כ-90% תמורת ויתור על לטנסי.** "cost זה בעדיפות עליונה עם חיסכון של כ-90% לעומת פתרונות אחרים" [15:32], אבל השירות מיועד ל-workloads של sub-second latency, ומי שצריך יחידות או עשרות מילישניות הופנה במפורש לפתרון אחר.
- **המודל התפעולי הוא ההבדל המהותי, לא רק המחיר.** אין קלאסטר, אין מכונות באוויר 24/7, אין patching ואין maintenance window — רק Vector Bucket, Vector Index וקריאות API [4:41, 7:45].
- **סיפור הלקוח סיפק את ההוכחה בסקייל.** Clover Security מאנדקסים כ-100 מיליון וקטורים, ראו ירידה של יותר מ-70% ב-CPU של בסיס הנתונים שלהם, ומצאו שהפרש הלטנסי מול המתחרים היה "זניח" ב-use case שלהם [20:25].
- **RAG הוא pipeline, לא שליפה סמנטית אחת.** המסר המרכזי של Clover: ניקוי דאטה העלה את דיוק ה-retrieval "בעשרות אחוזים" [22:03], ודמיון סמנטי אינו זהה לרלוונטיות [23:41]. הפייפליין שלהם: ניקוי, רשת רחבה ל-recall, צמצום ל-precision, ו-re-rank באמצעות LLM.
- **OpenSearch ממשיך לצמצם את עלות התפעול.** שלושה חידושים הוצגו: Vector Auto-optimize שמחליף שבועות של ניסויי קונפיגורציה ב-job בעלות של 12 דולר [33:01], GPU Acceleration לאינדוקס עם שיפור פי 3.8 [36:06], ודור חדש של Serverless עם סקיילינג בשניות ו-scale to zero [37:41].

2. רקע: מה הם וקטורים ואיך מחפשים בהם



שקופית הפתיחה: ארבעה דוברים — שניים מ-AWS ושניים מ-Clover Security

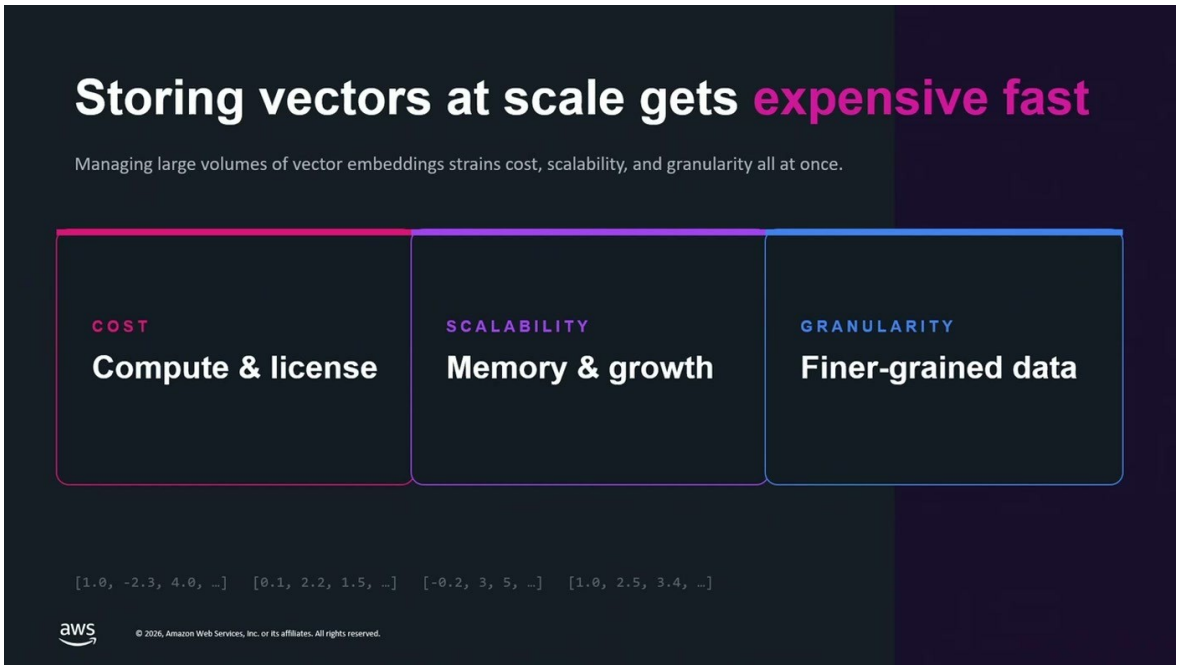
Arik Porat פתח ביישור קו קצר. וקטורים נוצרים על ידי מודל Embeddings שמקבל מידע גולמי — טקסט, תמונה או וידאו — ומחזיר וקטור של floating points. מידע בעל משמעות דומה ממוקם קרוב במרחב, ופונקציית מרחק מודדת את הקרבה [1:36].

> **Arik Porat, [0:00] —** וקטורים הם השפה שבה אייי מבין את העולם. הם מאפשרים לנו לחפש לפי דמיון והקשר, ולא לפי התאמה מדויקת של מילים.

הזרימה המעשית: דאטה לא מובנה שיושב ברובו ב-S3, אסטרטגיית chunking שמחלקת אותו לחתיכות, קריאה למודל embedding, ושמירת התוצאה ב-Vector Store שיודע לאנדקס ולחפש ביעילות [1:36].

חיפוש נאיבי של K-nearest neighbor מודד מרחק מהוקטור המבוקש לכל הוקטורים הקיימים — ובסקייל של מאות מיליוני וקטורים זה איטי. לכן בפועל משתמשים ב-Approximate Nearest Neighbor: מצמצמים את החיפוש לקלאסטר של וקטורים ומוותרים על חלק מה-recall כדי להרוויח latency. כפי שהוסבר, recall הוא "מדד מספרי בין 0-1 לתיוק תוצאות החיפוש" [3:07]. המתח הזה — recall מול latency מול עלות — הוא הציר שעליו נע כל הסשן.

3. S3 Vectors: איזו בעיה השירות בא לפתור



שלושת הצירים שהופכים אחסון וקטורים ליקר: עלות (compute ורישוי), סקלביליות (זיכרון וגדילה) וגרנולריות

הנימוק לבניית השירות הוצג כרשימת כאבים של הפתרונות הקיימים. ראשית, קלאסטרים של מכונות באוויר 24/7 — גם כשהם מנוהלים, עדיין צריך להתקין, לתקן (patch), לשדרג, לתחזק ולבצע סקייל, כלומר גם עלות compute וגם עלות תפעול. שנית, רוב הפתרונות מבוססי מסד נתונים או NoSQL, ובשביל לתת ביצועים בחיפוש הם נדרשים לטעון את הוקטורים ל-cache. כשמגיעים למאות מיליוני וקטורים, הבאת כולם ל-cache של המכונות הופכת למאתגרת, ואז מתחילים לשקול שיטות קוונטיזציה — "בקיצור מסובך" [6:13].

> **Arik Porat, [4:41]** — בשורה התחתונה רצינו לתת שירות שהוא נוח לעבודה פשוט, סקלאבילי, רזיליינט וזול

הפתרון שנבחר הוא להישען על התכונות של S3 עצמו: cloud native, serverless, ללא ניהול, סקלביליות ועמידות פרוסה על פני כל ה-region [6:13].

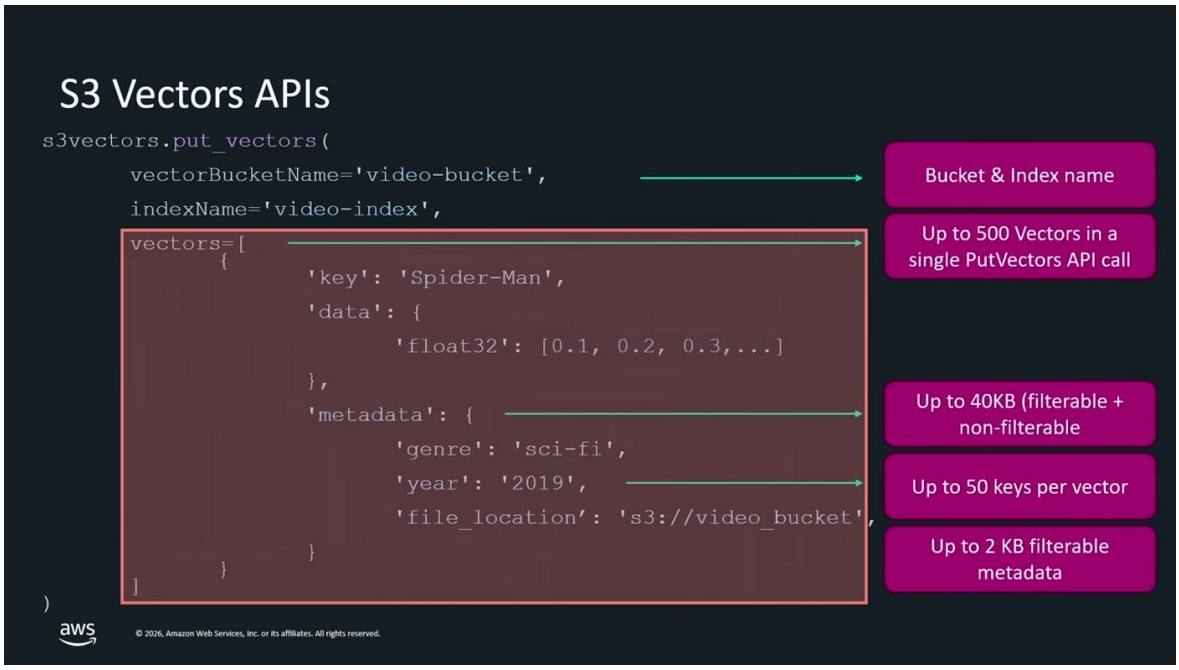
המספרים שהוצגו: השירות מיועד ל-workloads של sub-second latency, עם כ-100 מילישניות לשליפות חמות ב-cache. ניתן לייצר עד 10,000 Vector Buckets, וכל bucket יכול להכיל עד עשרות אלפי Vector Indexes — מה שמביא, כלשון הדובר, ל"מספר עצום של 20 טריליון וקטורים" [6:13]. מבחינת ingestion: עד 2,500 וקטורים בשנייה בעבודה ב-batch, ועד 1,000 וקטורים בשנייה בהכנסה של וקטור בודד [7:45]. הפרדה לאינדקסים שונים משמשת להפרדת tenants, לקוחות, או חלוקת דאטה לפי שנים.

4. המודל התפעולי וה-API

שני אובייקטים בלבד. Vector Bucket הוא סוג bucket חדש שמשמש קונטיינר לאינדקס אחד או יותר, ו-Vector Index הוא מבנה הנתונים שעליו מתבצעות כל הפעולות.

> **Arik Porat, [7:45]** — וקטור אינדקס זה המבנה נתונים הפיזי, אליו אנחנו מאנדקסים ומחפשים ועושים את כל הפעולות עליו.

ביצירת האינדקס נקבעים שני פרמטרים שאינם ניתנים לשינוי לאורך חיי האינדקס: dimension ו-distance metric. שינוי שלהם מחייב יצירת אינדקס חדש. ה-dimension נגזר ממודל ה-embedding, ונתמכים ערכים בין 1 ל-4,096; פונקציות המרחק הנתמכות הן cosine similarity ו-Euclidean distance. סוג הנתונים כרגע הוא float32 בלבד [7:45].

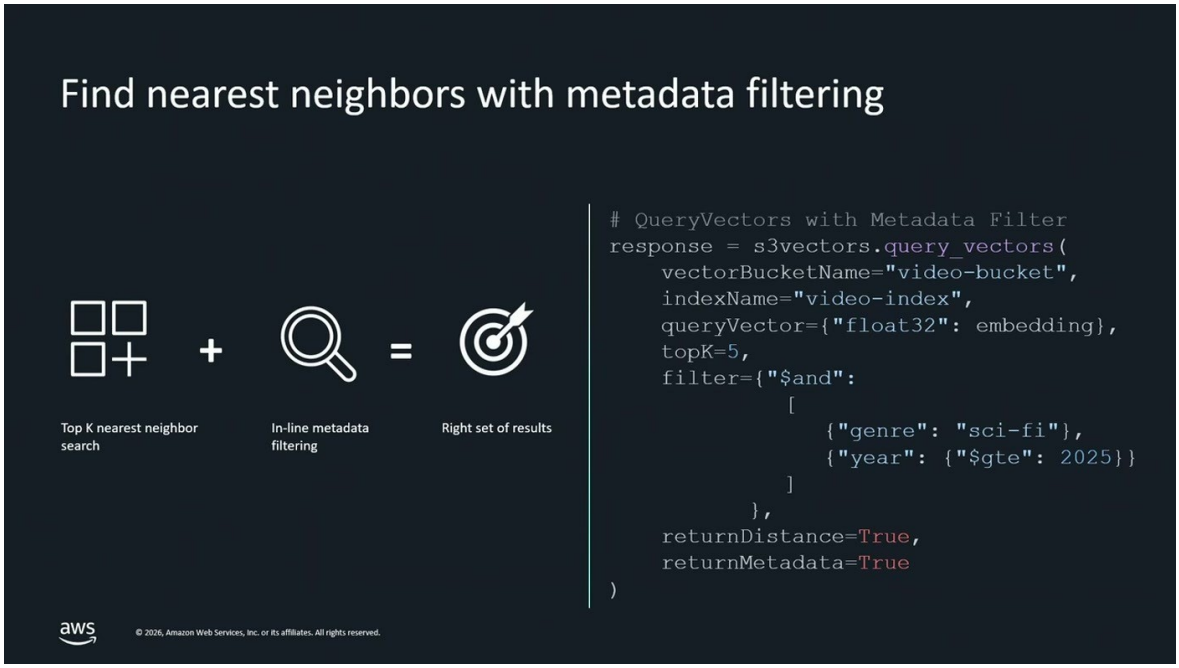


קריאת PutVectors עם המגבלות לצידה: עד 500 וקטורים בבקשה, עד 40KB מטאדאטה, עד 50 מפתחות לוקטור ועד 2KB מטאדאטה שניתנת לסינון

> **Arik Porat, [9:21] —** אנחנו יכולים להעביר עד 500 vectoring בבקשה אחת וזה גם הרבה יותר יעיל לעבוד ככה מאשר לשלוח vector vector זה משפר את הזמני ההכנסה בסדרי גודל

לכל וקטור יש key ייחודי (שמאפשר עדכון או מחיקה נקודתיים), שדה data שמחזיק את הוקטור עצמו, ורשימת metadata של זוגות key-value. המטאדאטה מתחלקת לשני סוגים: filterable ו-non-filterable. ברירת המחדל היא ש-הכול filterable, אלא אם צוינו שדות מסוימים כ-non-filterable בזמן יצירת האינדקס [9:21].

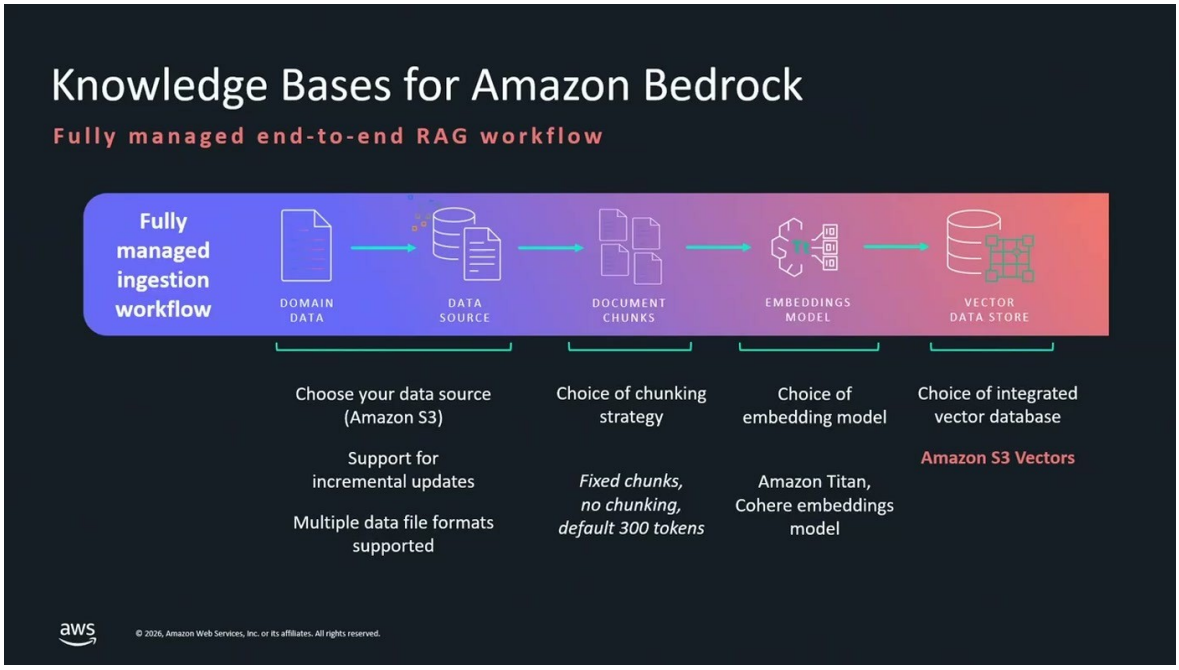
ההבחנה הזו חשובה מעשית. מטאדאטה שניתנת לסינון מאפשרת לצמצם את השליפה — "תן לי את ה-K וקטורים הכי קרובים לבקטור נתון, אבל רק אלה משנה 2019 ואילך" [9:21]. מטאדאטה שאינה ניתנת לסינון משמשת כמטען נלווה: שמירת פסקת הטקסט המקורית לצד הוקטור, כדי להחזיר תשובה ללקוח בלי קריאות API נוספות למערכות חיצוניות [10:55].



שאילתת QueryVectors: וקטור מטרה, topK=5, וסינון משולב על ז'אנר ועל שנה

בשליפה מעבירים bucket name, index name, וקטור מטרה ו-topK. נתמכים אופרטורים של exact match, greater, lower, exists, in, not in ו-, וניתן לשלב ביניהם ב-and וב-or. נקודה שהודגשה: הסינון על המטאדאטה מבוצע יחד עם החיפוש, ולא כשלב נפרד אחריו. שני דגלים נוספים שולטים בפלט — returnDistance (שימושי למיון לפי מרחק) ו-returnMetadata [10:55].

5. שתי אינטגרציות: Bedrock Knowledge Bases ו-OpenSearch



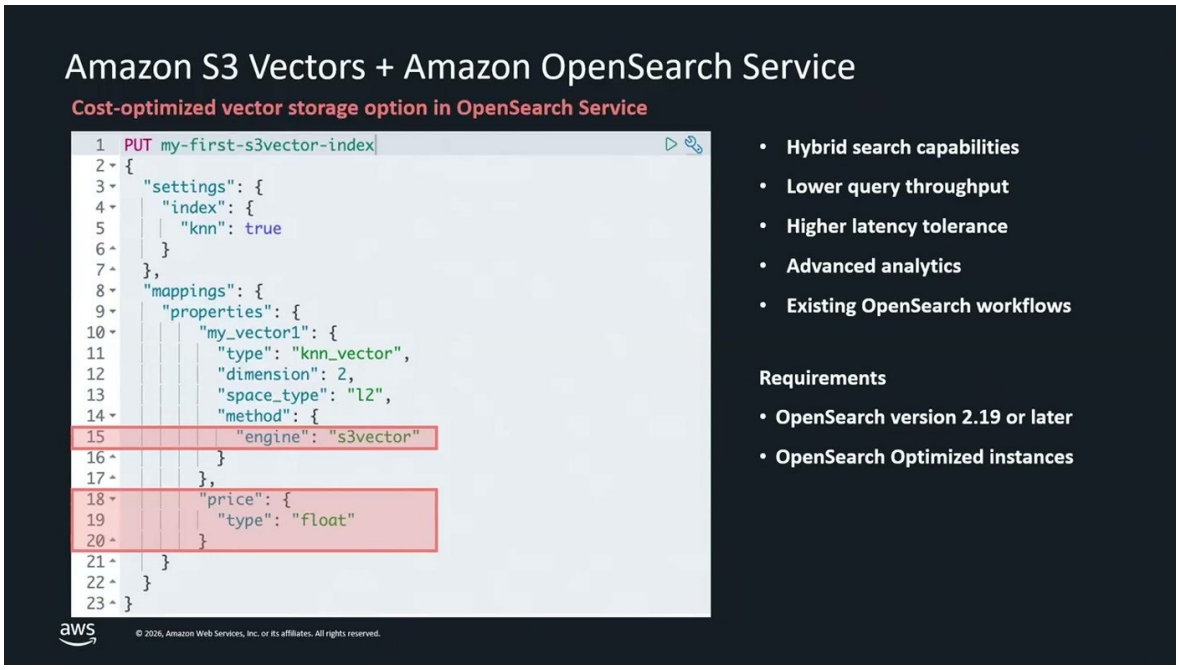
פייפליין ה-RAG המנוהל של Bedrock Knowledge Bases, עם S3 Vectors כאופציה לבסיס הוקטורים בקצה

האינטגרציה הראשונה היא עם Amazon Bedrock Knowledge Bases, שירות end-to-end ל-RAG. השירות בונה את הפייפליין כולו: לוקח את הדאטה, מפעיל אסטרטגיית chunking שבוחרים, קורא למודל embedding, ומאנדקס את התוצאה ב-Vector Store. S3 Vectors נוסף כאופציה ל-Vector Store הזה [12:25].

ההתאמה בין השניים תוארה כטבעית: "גם ככה האופי של ה-workloadים האלה הוא לא single digit ולא two digit milliseconds", כלומר RAG ממילא אינו use case של לטנסי קיצוני, ולכן החיסכון נקנה כמעט בחינם [12:25].

> **Arik Porat, [12:25] —** מרימים knowledge base בלי שום קומפיוט, בלי שום database, שום קלאסטר באוויר בכמה קליקים

ה-Knowledge Base גם נגיש כ-tool לפריימוורקים אג'נטיים קיימים בארגון — Strands Agents, LangGraph וכדומה — כך שאפשר לחבר אותו למערכות סוכנים קיימות [12:25].



הגדרת אינדקס ב-OpenSearch עם engine מסוג s3vector; בצד ימין היכולות שנשמרות והדרישות — גרסה 2.19 ומעלה ו-OpenSearch Optimized instances

האינטגרציה השנייה מיועדת ללקוחות שכבר מריצים OpenSearch ורוצים את שני העולמות: אחסון cost effective ב-S3 Vectors לצד יכולות מתקדמות של OpenSearch, כגון חיפוש היברידי (סמנטי ולקסיקוגרפי יחד) ויכולות אגרגציה [13:59].

המנגנון: יוצרים את האינדקס דרך OpenSearch עם engine חדש מסוג S3 Vectors. OpenSearch מייצר Vector Index ב-S3 Vectors, בעוד המטאדאטה נשמרת בתוך OpenSearch עצמו.

> **Arik Porat, [13:59] —** עבור כל primary charge שיש לכם ב-open search, אם לדוגמה יש לי חמישה primary charge ב-open search, ייווצרו לי חמישה S3 vector indexes

(בתמלול "primary charge" הוא שיבוש של primary shard.) בזמן שאילתה, OpenSearch מבצע offload ל-S3 Vectors, מריץ את החיפוש על כל האינדקסים במקביל, מקבל את ה-topK מכל אחד, ומחזיר לקוואורדינטור שמבצע re-ranking ואגרגציה מחדש [13:59].

המחיר: זה מתאים רק ל-use case שסובל את הלטנסי והתפוקה, ונתמך כרגע רק על OpenSearch Optimized instances [15:32].

6. מתי לבחור ב-S3 Vectors

הסיכום שניתן מן הבמה מנה ארבעה קריטריונים [15:32]:

- **עלות בעדיפות עליונה.** "חיסכון של כ-90% לעומת פתרונות אחרים, משולים רק עבור ה-usage וה-API" — אין תשלום על תשתית פנויה.
- **Zero Ops.** אין ניהול מכונות ואין תשתית לנהל; עבודה serverless לחלוטין מול קריאות API.
- **לטנסי sub-second בלבד.** כ-100 מילישניות לשליפות חמות ב-cache. "מי שמחפש Single Digit, Two Digit, אז אולי ראוי שישכל פתרונות אחרים"
- **פיצ'רים: חיפוש סמנטי עם סינון מטאדאטה.** "מבדיקות שאנחנו מגיעים ל-recall של 90% פלוס ולא ניתן כרגע לשלוט על הפרמטרים של ה-recall" — כלומר אין כוונון ידני של יחס recall/latency ברמת האינדקס, בניגוד ל-OpenSearch.

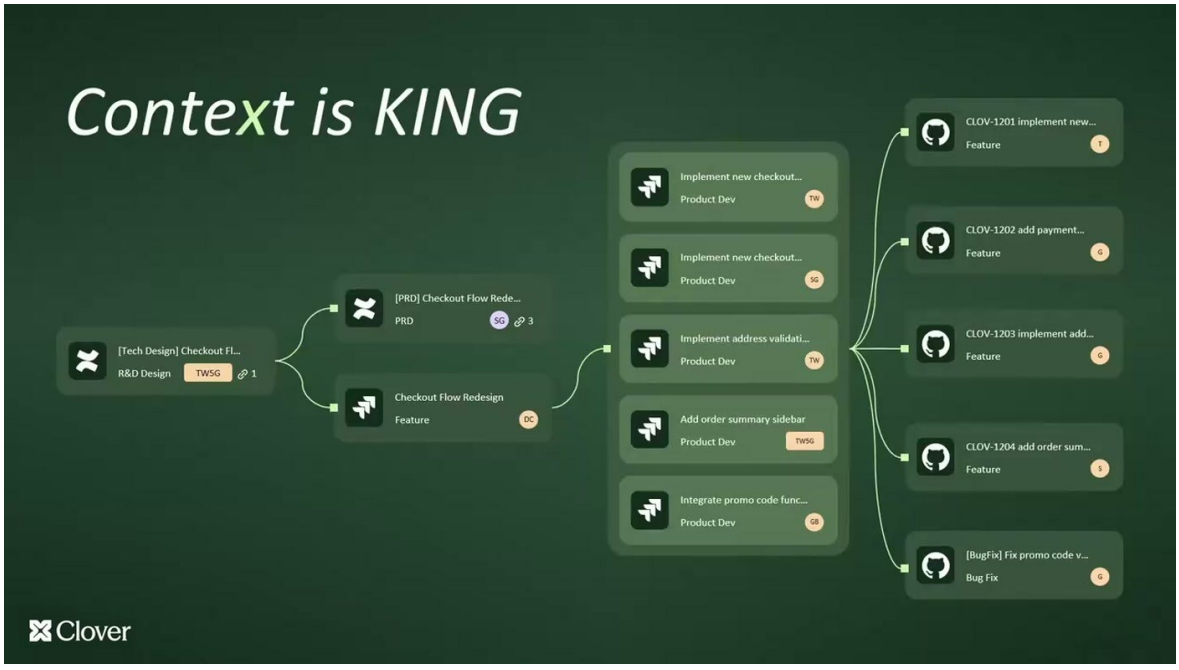
הדובר הפנה ל-QR code עם ריפוזיטורי ב-GitHub שבנה, הכולל דוגמה מלאה: לקיחת פריטי אופנה מתוך ה-Fashion Product dataset של Kaggle, ייצור embeddings, אינדוקס ב-S3 Vectors ואפליקציית חיפוש מעליהם [15:32].

7. Clover Security: RAG בפרודקשן

פתיחת חלק הלקוח — Shay Ben Haim, Engineering Manager, ו-Meitar Ronen, AI Research Lead

Clover Security עוזרת לארגונים גדולים להכניס אבטחה כבר בשלב ה-design, בעולם שבו coding agents מייצרים כמויות אדירות של פיצ'רים. התיאור שניתן: מסמך אפיון מאיש מוצר, design עם Claude Code, אישור של ראש צוות או ארכיטקט — ואז צוואר הבקבוק, אישור נוסף של צוות הסקיוריטי "שלא מצליח לעמוד באומס" [18:45].

המערכת מתחברת למערכות הארגוניות של הלקוח — Confluence, Jira, Slack, GitLab, Claude Code — ומזהה אוטומטית אילו פיצ'רים בעבודה. היא מקשרת מסמכי אפיון לטיקטים הרלוונטיים ב-Jira ולאחר מכן לאימפלמנטציה בקוד, כדי לייצר security review שקורה במקביל לפיתוח, "בתהליך שלוקח דקות בדדות במקום שבועות ארוכים" [18:45].



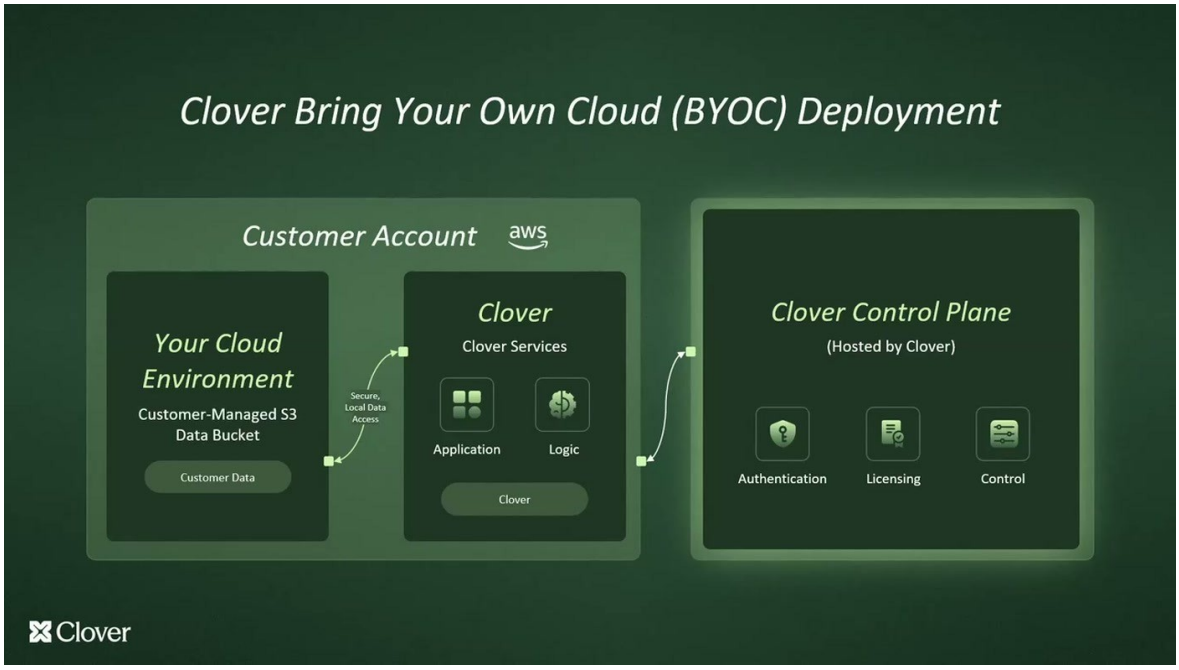
שלושת השלבים של המסע: POC של RAG, בעיות סקייל, וארכיטקטורה מחדש מעל S3

הם יצאו לדרך בתחילת 2024 עם RAG מעל בסיס הנתונים הרלציוני הקיים שלהם. זה עבד — עד שזה לא.

> **Shay Ben Haim, [18:45] —** מהר מאוד פגענו בקיר של פרודקשן, ברגע שהגענו לסקייל אמיתי של לקוחות, יצירת ושמירת הווקטורים חנקו לנו את הדאטאבייס עד לרמה ש-50% מהבקשות שלנו יהיו נחשלות על טיימאוט

המסקנה הייתה שאי אפשר להמשיך לפתור את זה בכסף ובהרחבה אופקית לנצח [20:25].

8. למה דווקא S3 Vectors — שיקולי ההנדסה



פריסת Bring Your Own Cloud: ה-bucket והאפליקציה יושבים בחשבון הלקוח, ה-control plane מתארח אצל Clover

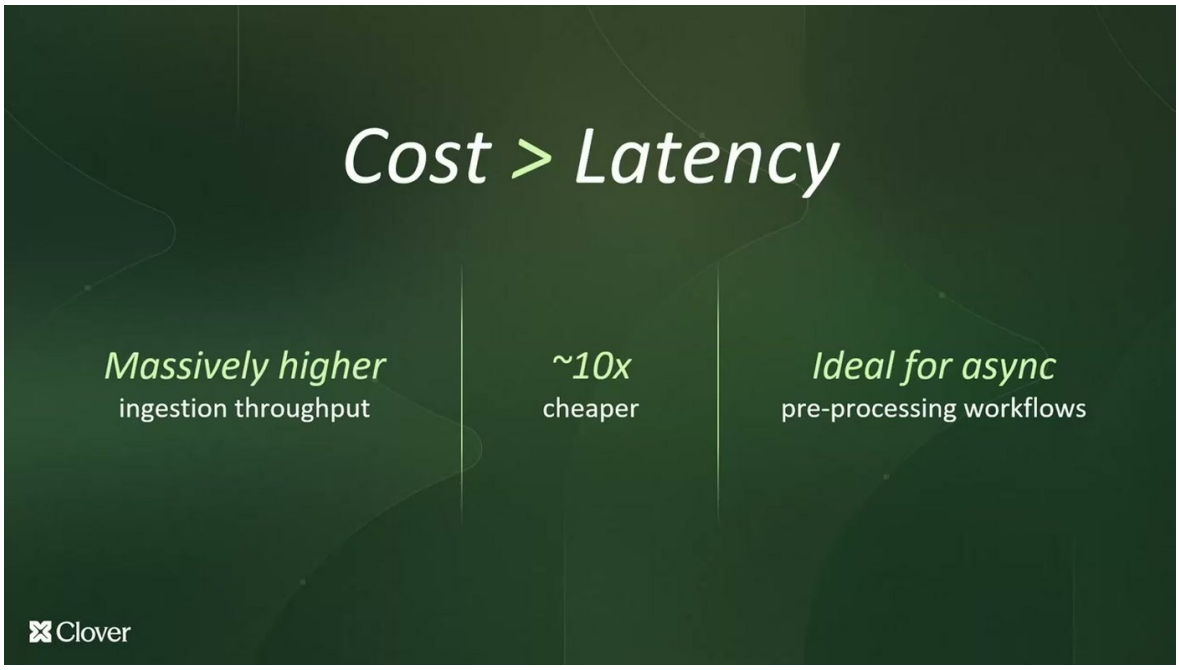
אילוץ שהגיע מכיוון לא צפוי הכריע את הבחירה.

> **Shay Ben Haim, [20:25] —** לקוחות הגיעו עם דרישה שהמוצר שלנו צריך לשבת בזויבת הענן שלהם מה שנקרא bring your own cloud deployment

S3 Vectors התיישב על הדרישה הזו "בול": קל בהרבה לעבוד איתו מאשר עם מתחרים שדרשו hosting בסביבת הלקוח [20:25].

מעבר לכך הוצגו שלושה ממצאים מספריים מן הגרפים שעל הבמה:

- **סקייל ingestion:** "היום אנחנו עומדים על סדר גודל הכנסה של 100 מיליון וקטורים" [20:25], עם דגש חוזר על עבודה ב-batch — "ממש ממש חשוב לעשות את ההכנסות בבטשים ולא אחד אחד חלילה".
- **העומס על בסיס הנתונים:** "ראינו ירידה דרסטית ב-CPU שלו ברגע שהוצאנו את האינג'סטשן של הבטשים לתוך הדאטאבייס, כמו שרואים בגרף, ירידה של יותר מ-70%" [20:25].
- **הלטנסי לא היה חסם:** "ראינו שהפר latency בינו לבין המועמדים אחרים היה זניח ולא פקטור ביוז קייס שלנו" [20:25], משום שרוב העיבוד אצלם מתרחש אסינכרונית.



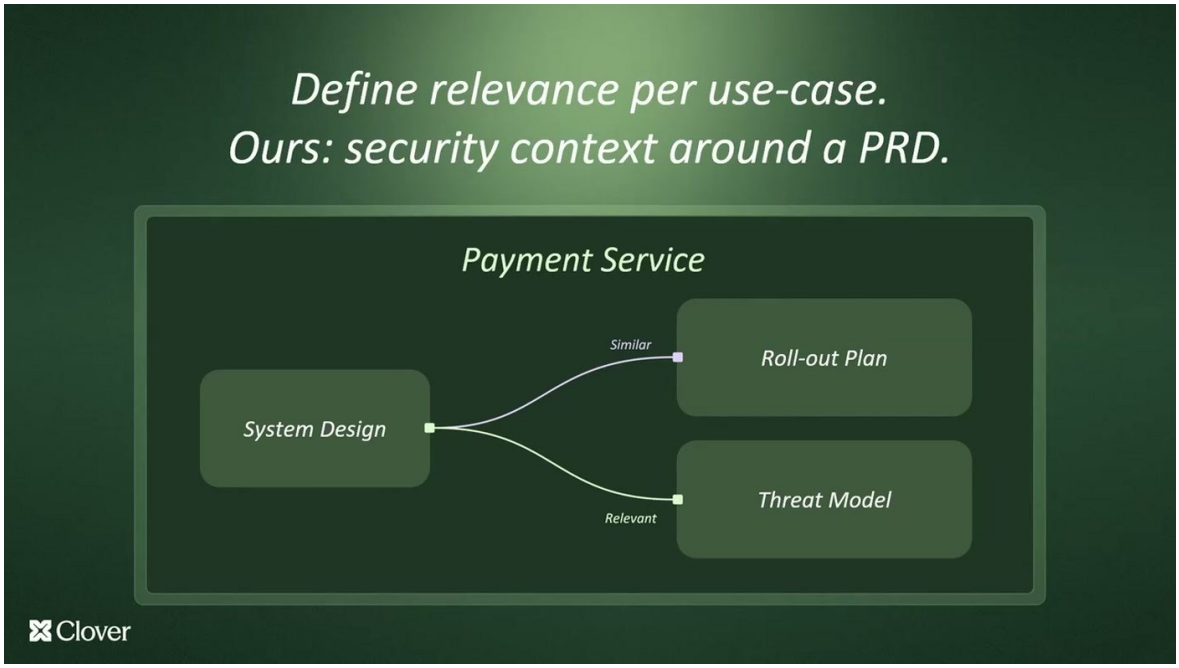
סיכום ההכרעה של Clover: תפוקת ingestion גבוהה משמעותית, זול בערך פי 10, ומתאים לזרימות עיבוד מקדים אסינכרוניות

9. איכות: איך הופכים RAG למשהו שעובד בפרודקשן

החלק השני של הרצאת הלקוח, מפי Meitar Ronen, עסק במה שהתשתית לא פותרת.

> **Meitar Ronen, [22:03] —** לקחת את כל הדאטה, לזרוק אותו לאמבדינגס ולעשות מעלב סמנטיק סרטש עובד מעולה כשבונים דמו MVP. ברגע שעוברים לפרודקשן, בסביבת אנטרפרייז, ויש מיליוני מסמכים, טיקטים, פול ריקורס, זה נשבר ממש ממש מהר.

האתגר הראשון הוא הפשוט ביותר, ולכן זה שמדלגים עליו: הדאטה מלוכלך. צריך לנקות HTML, markup, טבלאות ותמונות — אחרת מבצעים embedding לרעש — garbage in, garbage out. התוצאה שדווחה: "אחרי שאנחנו באמת ניקינו את הדאטה, הדבר הקטן הזה העלה אצלנו את הדיוק של הרטריבל בעשרות אחוזים" [22:03].



למה דמיון סמנטי מטעה: מסמך Roll-out Plan דומה למסמך האפיון אך אינו רלוונטי ל-review, בעוד Threat Model רלוונטי אך רחוק סמנטית

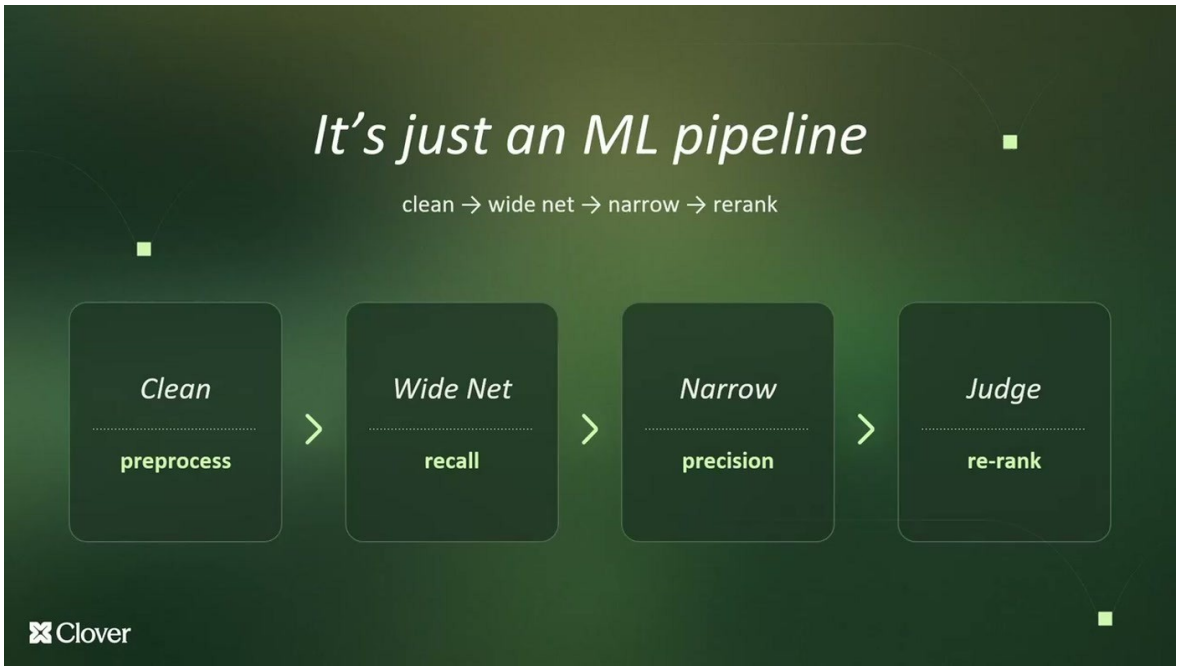
האתגר השני עמוק יותר. בהקשר של security review על מסמך אפיון, דמיון סמנטי אינו זהה לרלוונטיות.

> **Meitar Ronen, [23:41]** — דמיון סמנטי זה לא בדיוק אחד לאחד כמו רלוונטיות

הדוגמה שניתנה: עבור שירות תשלומים, מסמך Roll-out Plan יקבל ציון דמיון גבוה מאוד — כי הוא עוסק באותם דברים ומשתמש באותם מושגים — אך רלוונטיותו ל-security review נמוכה. לעומת זאת מסמך ארכיטקטורה או threat model, שהם רלוונטיים מאוד, יקבלו ציון similarity נמוך [23:41].

המענה הוא פייפליין בן ארבעה שלבים. תחילה ניקוי. אחר כך פריסת רשת רחבה — סף נמוך לדמיון סמנטי או top-K גבוה, במטרה להביא כמה שיותר מועמדים, כשהמטרה המוצהרת בשלב הזה היא לא לפספס דבר [25:13]. לאחר מכן צמצום באמצעות filterable metadata שנשמרת בזמן ה-pre-processing — סוג המסמך, הטכנולוגיה המעורבת, הפיצ'רים שהוא נוגע בהם — כדי להפריד בין מה שדומה סמנטית למה שדומה וגם רלוונטי.

> **Meitar Ronen, [25:13]** — השלב האחרון בפייפליין שלנו, והוא חייב להיות עכשיו כי הוא יקר, אנחנו לוקחים את השורטליסט שהרטריבה לחזיר לנו, ונותנים לאל אליהם להיות השופט.



ארבעת השלבים כפי שהוצגו: Clean, Wide Net (recall), Narrow (precision), Judge (re-rank)

ה-LLM מקבל את ההגדרה הארגונית של רלוונטיות ואת המסמכים ששרדו, ומבצע re-ranking. מכיוון שמגיעה אליו רשימה קטנה בלבד, העלות החישובית נשמרת נמוכה, ובמקביל השלב מגן מפני false positives [25:13].

המסקנה שהם ביקשו שייקחו מהחלק שלהם: RAG הוא ברוב המקרים רק חלק מהפייפליין, לא הפייפליין כולו. קודם מגדירים בדיוק מה זו רלוונטיות בארגון, ואז מתקדמים לפי הסדר re-rank ← precision ← recall [26:50].

10. OpenSearch: מתי הוא המנוע הנכון

Liat Iusim פתחה את החלק האחרון בשאלה שממסגרת את ההבדל: מה אם צריך לתמוך באלפי queries בשנייה, או בלטנסי של לא יותר מעשרות מילישניות לשאילתה, או ביכולות חיפוש מתקדמות יותר [26:50]?

OpenSearch הוא פרויקט קוד פתוח שמאגד שלוש יכולות במנוע אחד — אנליטיקס, חיפוש וקטורי וחיפוש לקסיקלי — ולפי המספרים שהוצגו יש לו "מעל שני מיליארד הורדות" ומעל ארבע מאות ארגונים תורמים [26:50]. Amazon OpenSearch Service הוא הגרסה המנוהלת, שנועדה לצמצם את העומס התפעולי של עבודה עם קלאסטר — הקמה, הוספת מכונות, הגדלה — ומוסיפה אינטגרציות עם S3 Vectors, Bedrock, Firehose ועוד. נקודת עלות שצוינה: בשירות המנוהל אין תשלום על תעבורה בין Availability Zones [28:25].

שלושה מקרים שבהם OpenSearch הוא הבחירה [28:25]:

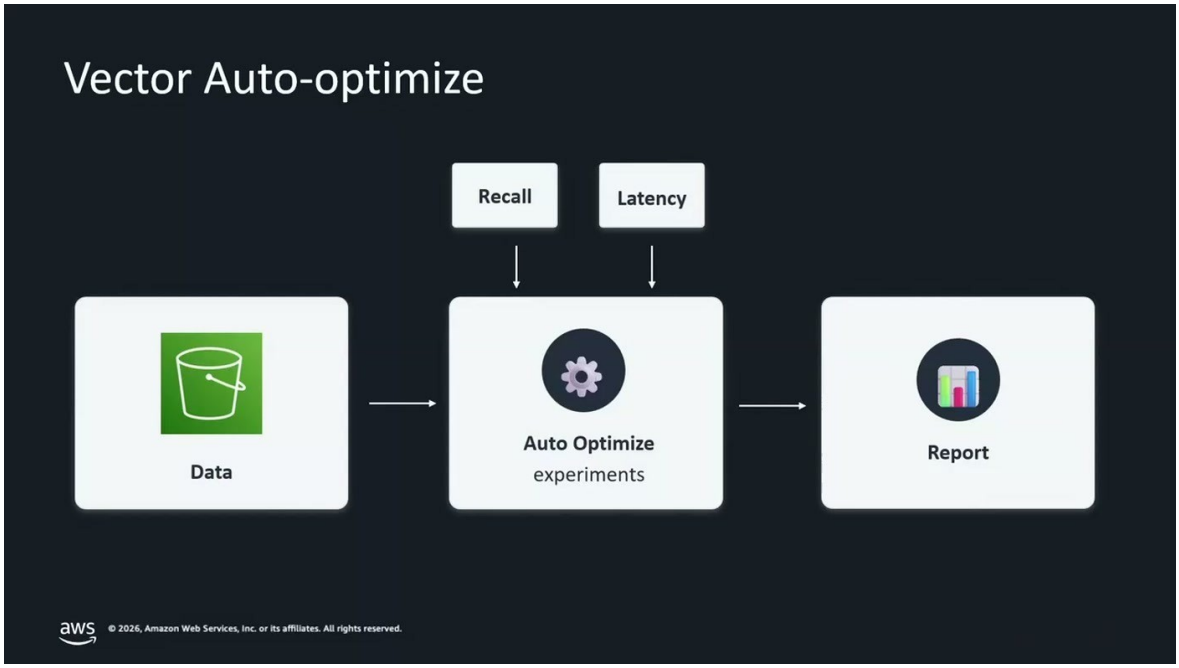
- **סקייל וביצועים גבוהים במיוחד.** מיליארדי וקטורים או אלפי שאילתות בשנייה, תוך שמירה על לטנסי נמוך, על גבי קלאסטר שגדל הוריזונטלית.
- **חיפוש היברידי.** דירוג תוצאות לפי רלוונטיות המחושבת משילוב של חיפוש וקטורי וחיפוש לקסיקלי — משפר איכות ודיוק בהרבה use cases.
- **גמישות.** סט קונפיגורציות רחב ששולט באיזון בין latency, recall ועלות. "כל יוזקייס מצריך איזון אחר" [29:56].

11. שלושה חידושים מהשנה האחרונה

11.1 Vector Auto-optimize

הגמישות של OpenSearch היא גם עלותו: כדי להגיע לקונפיגורציה אופטימלית צריך להקים קלאסטר, להגדיר קונפיגורציה התחלתית, לאנדקס כמויות דאטה, למדוד — ולחזור על כך שוב ושוב.

> **Liat Iusim, [31:28] —** אנחנו רואים לקוחות שעובדים על זה גם שבועות שלמים



זרימת Vector Auto-optimize: דאטה ב-S3, שני אילוצים (recall ו-latency), ניסויים אוטומטיים, ודוח המלצות

מזינים דאטה משלכם ב-bucket ב-S3, מגדירים שני פרמטרים — recall ו-latency הנדרשים — והשירות מריץ את הניסויים. הודגש שהבדיקות אינן רצות על הקלאסטר שלכם ובוודאי לא על פרודקשן; הן רצות serverless על משאבים שהשירות מקים מאחורי הקלעים [31:28].

בסיום מתקבל דוח עם המלצות קונפיגורציה, וניתן בלחיצת כפתור לייצר אינדקס בפרודקשן עם הפרמטרים שנבחרו — כשהדאטה שהיה ב-bucket מאונדקס אוטומטית לקלאסטר. בדוגמה שהוצגה, עבור דרישה של recall גדול מ-0.9

ולטנסי של 200–300 מילישניות לשאילתה, התקבלו שלוש המלצות, כל אחת עם ה-recall הצפוי וההשפעה על הזיכרון [33:01].

> **Liat Iusim, [33:01] —** חשוב להגיד, עלות כזו של ג'וב שמריץ את כל הבדיקות, 12 דולר לג'וב אחד

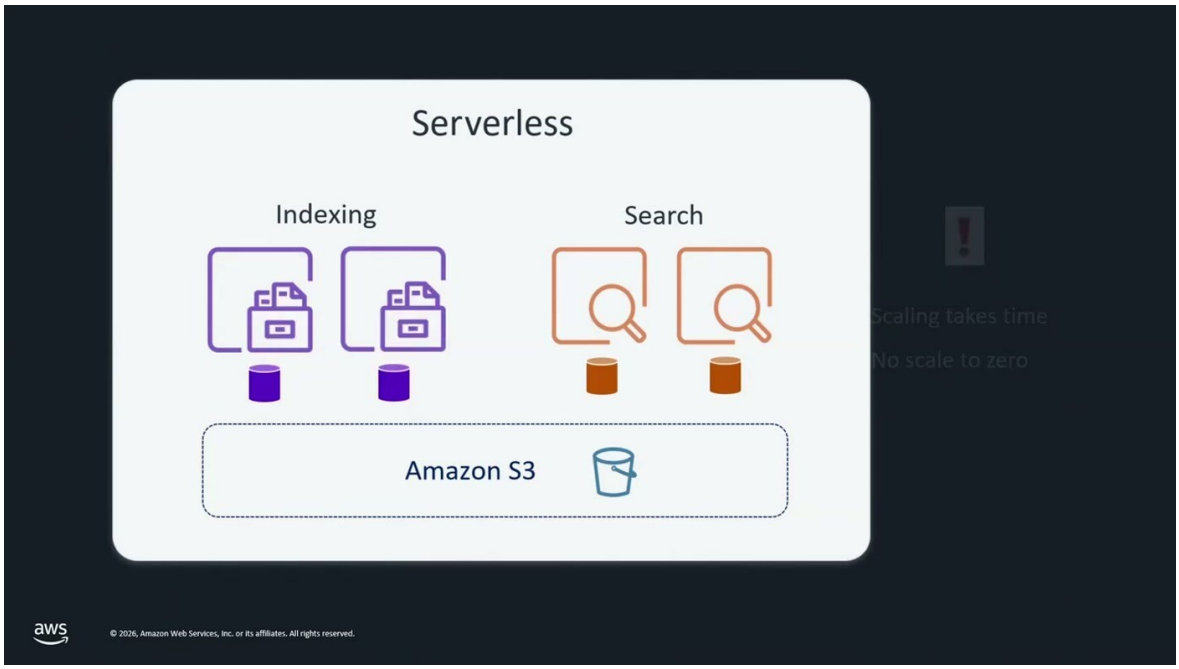
11.2 GPU Acceleration for Vector Indexing

אינדוקס של כמויות דאטה גדולות הוא פעולה כבדה שיכולה לקחת ימים, משום ש-OpenSearch בונה ברקע מבני נתונים פנימיים — בוקטורים, ברוב המקרים גרפים. ההנחה שמדובר בפעולה חד-פעמית שגויה: כל החלפת מודל embedding, כל הוספת קוונטיזציה וכל שינוי סכמה מחייבים אינדוקס מחדש של כל הדאטה [34:31].

הפתרון: מעבר לקלאסטר שלכם, AWS מתחזקת warm pool של GPU instances. כשהשירות מזהה עומס אינדוקס שחוצה סף מסוים, מתבצע offload של העבודה למכונות הללו, והתשלום הוא רק על המשאבים שנוצלו בפועל. כשהעומס יורד מתחת לסף, הקלאסטר חוזר לעבוד כרגיל [36:06].

ב-benchmark שהוצג (עם הפניה ל-blog post ב-QR code), אינדוקס של 113 מיליון וקטורים הניב "שיפור בביצועים של פי 3.8" [36:06].

11.3 הדור הבא של OpenSearch Serverless



מבנה ה-Serverless: וורקרים נפרדים לאינדוקס ולחיפוש, עם שכבת S3 שמסנכרנת ביניהם

בדור הקודם (Classic Generation) לכל worker היה דיסק צמוד ששימש hot storage, ושכבת S3 סינכרנה בין שכבת האינדוקס לשכבת החיפוש. הוספת worker חייבה גם הוספת דיסק, bootstrap והעתקת דאטה אליו — פעולה כבדה שהאטה את הסקיילינג, ומאותה סיבה גם מנעה scale to zero אמיתי: גם כשהמערכת הייתה idle, נשארו משאבי compute למעלה [37:41].

הדור הבא, OpenSearch Serverless Next Generation, משנה את הארכיטקטורה ומוסיף שכבת storage משותפת לכל הוורקרים. worker חדש עולה ישירות מעל ה-storage המשותף, בלי bootstrap ובלי העתקת דאטה.

> **Liat Iusim, [37:41] —** הסקיילינג הופך להיות הרבה יותר מהיר, הופך להיות עניין של שניות, מתקום משהו שיכול לקחת גם כמה דקות בדור הישן

> **Liat Iusim, [39:19] —** אם הסרצ' או האינדקסים, כל אחד מהם היה איידל במשך עשר דקות, המשאבי קומפיוט ירדו לאפס ולא יהיה עליהם תשלום

12. מה לוקחים מכאן

הסשן נחתם במשפט שמסכם אותו טוב יותר מכל טבלת השוואה: "יש המון פתרונות בחיפוש וקטורי, תנסו לחשוב מה מתאים ליוסקי שלכם" ו"לפני שאתם הולכים לממש" [39:19]. להלן הנקודות המעשיות.

נושא	מה לעשות איתו	זמן
בחירת Vector Store	התחילו מדרישת הלטנסי. sub-second → S3 Vectors; עשרות מילישניות ומטה → OpenSearch	15:32
עלות מול ביצועים	כ-90% חיסכון הוא המספר שהוצג, אך הוא נקנה בוויתור על כוונון recall/latency	15:32
פרמטרים בלתי הפיכים	dimension ו-distance metric נקבעים ביצירת האינדקס ואינם ניתנים לשינוי	7:45
עבודה ב-batch	עד 500 וקטורים בבקשה. שני דוברים נפרדים הדגישו שזה משנה בסדרי גודל	9:21, 20:25
מטאדאטה	החליטו מראש מה filterable — זה מה שיאפשר לצמצם מועמדים בשלב ה-precision	9:21, 25:13
איכות RAG	נקו את הדאטה לפני הכול. זהו השיפור הזול ביותר שדווח בסשן	22:03
הגדרת רלוונטיות	אל תניחו שדמיון סמנטי = רלוונטיות. הגדירו רלוונטיות במפורש, והוסיפו re-rank	23:41
כוונון OpenSearch	לפני שמשקיעים שבועות בניסויים ידניים — Vector Auto-optimize, 12 דולר ל-job	33:01
פריסה בסביבת לקוח	BYOC היה השיקול המכריע אצל Clover, לא המחיר ולא הביצועים	20:25

13. מונחון

מונח	משמעות
Vector Bucket	סוג bucket חדש ב-S3, קונטיינר לאינדקס וקטורים אחד או יותר
Vector Index	מבנה הנתונים שאליו מאנדקסים ושעליו מחפשים; dimension ומטריקת מרחק נקבעים בו לצמיתות
ANN	Approximate Nearest Neighbor — חיפוש מקורב שמוותר על recall בתמורה ל-latency
Recall	מדד בין 0 ל-1 לדיוק תוצאות החיפוש — כמה מהשכנים הנכונים באמת הוחזרו
Filterable metadata	שדות מטאדאטה שניתן לסנן עליהם בזמן השליפה; ברירת המחדל ב-S3 Vectors
Knowledge Base	שירות RAG מנוהל ב-Amazon Bedrock, מ-chunking ועד אינדוקס בבסיס וקטורים
חיפוש היברידי	שילוב של חיפוש וקטורי וחיפוש לקסיקלי לדירוג אחיד של תוצאות
Re-ranking	דירוג מחדש של שורטליסט תוצאות, אצל Clover באמצעות LLM כשופט רלוונטיות
BYOC	Bring Your Own Cloud — פריסת המוצר בתוך חשבון הענן של הלקוח
Vector Auto-optimize	הרצת ניסויי קונפיגורציה serverless ב-OpenSearch והפקת דוח המלצות
Scale to zero	ירידה מלאה לאפס משאבי compute בזמן idle, בדור החדש של OpenSearch Serverless