NoSQL על AWS: איך בוחרים את הדאטאבייס הנכון

TLV-DAT202 — סיכום סשן, AWS Summit Tel Aviv 2026

פזי גרשון, Senior Technical Account Manager, AWS | לאוניד קורן, Principal NoSQL Solutions Architect, AWS | אבגני צודיקוב, Staff Software Engineer, AppsFlyer

10 בספטמבר 2026 | משך: כ-37 דקות | הופק מהקלטת הסשן

סיכום מאת קובי אלמוג

על המסמך הזה

המסמך מסכם את הסשן TLV-DAT202 מתוך AWS Summit Tel Aviv 2026, בהתבסס על ההקלטה המלאה שפורסמה באתר הסאמיט. הוא נבנה מתמלול אוטומטי של ההקלטה ומהסליידים שחולצו מהווידאו עצמו, כך שאפשר לחזור לתוכן בלי לצפות שוב בהרצאה. חותמות הזמן בגוף המסמך מפנות לנקודה המדויקת בהקלטה.

הסשן מחולק לשני חלקים שונים באופיים: שני דוברי AWS שעוברים על שלושה שירותי NoSQL ומשווים ביניהם פרמטר-פרמטר, ולאחר מכן מהנדס מ-AppsFlyer שמספר סיפור מיגרציה אמיתי בסקייל גדול. המסמך שומר על הסדר הזה.

איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. מה הוצג ומה שווה לזכור.
- **פרקים 2–3 — המפה:** הגישה של Purpose-Built Databases והשוואת מבט-על בין שלושת השירותים.
- **פרקים 4–6 — צלילה לשירותים:** ElastiCache ,DynamoDB ו-DocumentDB, כל אחד בנפרד.
- **פרק 7 — טבלת ההשוואה:** הטבלה המסכמת שהוצגה על הבמה, פרמטר אחרי פרמטר.
- **פרק 8 — המקרה של AppsFlyer:** למה עברו ל-DynamoDB, מה נשבר ב-POC, ואיך הרידיזיין תיקן בעיה בת שנים.
- **פרקים 9–10 — סגירה:** AI Skills לדאטאבייסים, ומה לוקחים מכאן.

הערת מקור המסמך הופק מתמלול אוטומטי של ההקלטה. התמלול מדויק בקטעים המקצועיים, אך מונחים ושמות מוצרים לועזיים הופיעו בו בשיבוש פונטי ותוקנו כאן לצורתם האנגלית (למשל "אלסטיקש" → ElastiCache, "דיינמו דיבי" → DynamoDB, "פרטישן" → partition). הציטוטים אומתו מול ההקלטה בחותמת הזמן המצוינת; מספרים שהוצגו על הסליידים סומנו ככאלה.

1. תקציר מנהלים

הסשן עונה על שאלה אחת מעשית: יש ל-AWS יותר מדי בסיסי נתונים, איזה מהם מתאים לי. שלושת הדוברים לוקחים שלושה שירותי NoSQL — Amazon ElastiCache, Amazon DynamoDB ו-Amazon DocumentDB — ומשווים ביניהם על אותם צירים: דורביליות, latency, עושר התשאול, רפליקציה גלובלית, טרנזקציות וחיפוש וקטורי. הפרק האחרון הוא עדות לקוח שממחישה מה קורה כשבוחרים נכון בסקייל של מאות מיליארדי רשומות.

- **ההבדל המרכזי הוא לא ביצועים אלא עושר התשאול.** ElastiCache עובד במיקרו-שניות אבל הוא במהותו cache; DynamoDB נותן מילי-שניות בודדות קבועות בכל סקייל אך עם יכולות תשאול פשוטות יחסית; DocumentDB הוא בעל "יכולות האינדוקס ותשאול הנתונים העשירות ביותר" מבין השלושה [23:23].
- **DynamoDB הוא היחיד שכותב וקורא במקביל בכמה אזורים.** שני מצבי רפליקציה גלובלית — MR-EC אסינכרוני (RPO של שניות בודדות, מספר אזורים ללא הגבלה) ו-MR-SC סינכרוני (RPO אפס, עד שלושה אזורים) [20:15, 21:51].
- **שלושת השירותים כבר תומכים בחיפוש וקטורי.** ב-DynamoDB זו יכולת חדשה מאוד — "בדיוק לפני חודש, DynamoDB התחיל לתמוך בחיפוש וקטורי" [17:04]. ההמלצה של AWS: לבצע חיפוש וקטורי מול השירות שבו הנתונים כבר יושבים, כדי לשמור על ארכיטקטורה פשוטה [26:28].
- **ElastiCache קיבל דורביליות, אבל זה לא בחינם.** במוד הסינכרוני אין איבוד מידע אך הכתיבות עוברות ממיקרו-שניות למילי-שניות בודדות; במוד האסינכרוני הכתיבות מהירות, "אך יש חלון של עד עשר שניות שבו ייתכן איבוד מידע" [12:21].
- **בסקייל של AppsFlyer, DynamoDB היה גם הנכון וגם הזול.** מעל 350 מיליארד רשומות, 90TB דאטה, מעל 1.5 מיליון קריאות לשנייה וכ-200 אלף כתיבות לשנייה [27:59]. על הסלייד: "Storage scales with the data, at a fraction of in-memory cost".
- **השגיאות מ-DynamoDB היו האבחנה, לא התקלה.** ה-throttling שהתקבל ביום הראשון של ה-POC חשף race condition ארכיטקטוני שהמערכת חיה איתו שנים — ועל הסלייד המסכם זה נוסח כ-"Quotas = signal".

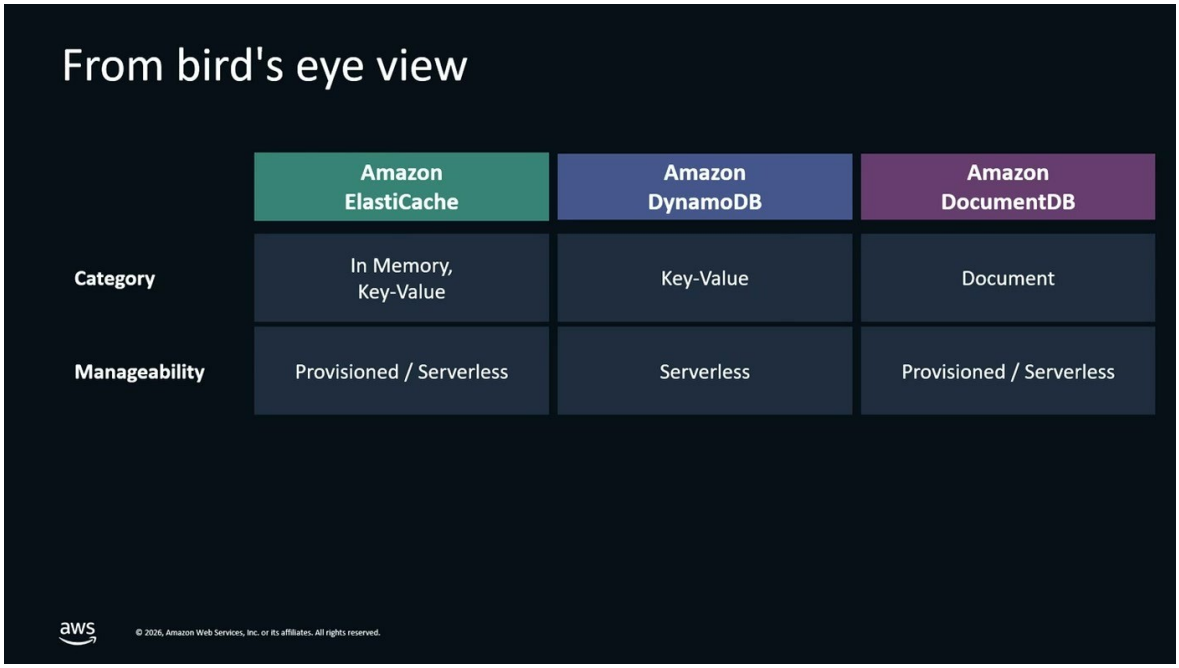
2. הרקע: Purpose-Built Databases

הפתיחה של פזי גרשון מציבה את המסגרת. במקום בסיס נתונים מונוליטי אחד שמנסה לענות על כל הצרכים, AWS בנתה פורטפוליו רחב, וההנחה היא שתמיד תיבחר ממנו ההתאמה הטובה ביותר ל-access patterns של האפליקציה [1:35]. כל השירותים בפורטפוליו הם שירותים מנוהלים.

המפה כפי שהוצגה: קטגוריית הרלציוניים (Amazon Aurora, Amazon RDS) — שלא נדונה בסשן; קטגוריית In-Memory (Amazon ElastiCache ו-Amazon MemoryDB), שבה הנתונים נקראים מהזיכרון עבור אפליקציות שדורשות קריאה בפחות ממילי-שנייה; קטגוריית Key-Value (Amazon DynamoDB) ו-Wide-Column (Amazon Keyspaces, שנועד בעיקר למיגרציות מ-Apache Cassandra); קטגוריית ה-Document (Amazon DocumentDB); ולבסוף ה-Specialized: Amazon Neptune לגרפים ו-Amazon Timestream ל-time-series [3:08, 4:39].

> **למה MemoryDB לא נכלל** הדוברת ציינה במפורש ש-MemoryDB לא הורחב בסשן, כי "מרבית הלקוחות שלנו יעדיפו את ElastiCache בזכות הביצועים המשופרים ויכולות התשאול המתקדמות" [3:08].

3. מבט-על: שלושת השירותים זה מול זה



שורות הפתיחה של ההשוואה: קטגוריית השירות ומודל הניהול. DynamoDB לבדו מסומן Serverless בלבד; ElastiCache ו-DocumentDB מציעים Provisioned או Serverless.

שלושת השירותים הם מנוהלים — highly available, מאובטחים, עם setup פשוט, וגיבויים, patches ו-failover אוטומטיים [6:10]. ההבדל הראשון הוא במודל: DynamoDB הוא serverless מלא, כלומר הוא מתאים את עצמו אוטומטית לתעבורה והלקוח תמיד על הגרסה האחרונה. גם ל-ElastiCache ול-DocumentDB יש אופציית Serverless שמאפשרת auto-scaling, "אך עדיין עם הצורך לשדרג גרסאות כדי ליהנות מיכולות חדשות" [6:10].

ההבדל השני הוא בתקרת הסקייל. DynamoDB ו-ElastiCache הם דאטאבייסים מבוזרים שתומכים ב-partitioning; DynamoDB לוקח את זה צעד קדימה — "למעשה אין לו מגבלה על כמות הבקשות בשנייה ונפח הנתונים בטבלה" [6:10]. DocumentDB תומך בקצב קריאות גבוה מאוד, אך מוגבל יותר בקצב הכתיבות.

וההבדל השלישי הוא ה-API: ElastiCache תומך ב-APIs של Valkey, Memcached ו-Redis; DynamoDB ב-HTTP APIs פרופריאטריים וגם ב-PartiQL, שפת שאילתות תואמת-SQL; DocumentDB ב-APIs של MongoDB [6:10].

4. Amazon ElastiCache

Valkey כמנוע הדיפולטיבי

החלק הראשון של הצלילה מוקדש להיכרות עם Valkey — מנוע שנוצר כ-fork של Redis 7.2 תחת ה-Linux Foundation, לאחר שRedis שינו את מודל הרישוי שלהם והקהילה יצרה חלופה פתוחה. התאימות מלאה: אותן פקודות, אותם מבני נתונים, ולכן מעבר שקוף ללא שינויי קוד [7:43].

> — Valkey **Senior Technical Account Manager, AWS [7:43]** כיום הוא המנוע המומלץ והדיפולטיבי ב-ElastiCache, עם הפחתה של כ-20% בעלות בהשוואה ל-Redis.

מאז ה-fork המנוע התקדם, והגרסה שצוינה בבמה היא Valkey 9.1. כשיוצרים cluster בוחרים אם הוא יהיה Valkey, Memcached או Redis — והפיצ'רים המתקדמים ביותר נמצאים כיום ב-Valkey [7:43].

סקייל, גלובליות ו-Data Tiering

cluster של ElastiCache תומך במיליוני פעולות בשנייה ובמאות טרה-בייט של נתונים [9:15]. אפשר להגדיר cluster גלובלי בין מספר regions, כאשר ב-primary region אפשר לקרוא ולכתוב וב-secondary אפשר רק לקרוא — עם אפשרות לקדם secondary ל-primary בעת הצורך. יכולת ה-Data Tiering מאפשרת לשמור נתונים לא רק בזיכרון אלא גם על ephemeral SSDs כדי להוזיל עלויות, כאשר ephemeral פירושו שכשהמכונה יורדת הנתונים נמחקים [9:15].

בצד החיפוש, ElastiCache תומך בשאילתה אחת שמשלבת חיפוש וקטורי, full-text search וסינון לפי טווחי מספרים. הדוגמה שניתנה על הבמה: "תמצא לי חמישה מתכונים דומים סמנטית לארוחת בוקר בריאה עם זמן הכנה של פחות מ-10 דקות שבכותרת כתובות המילים ללא גלוטן" [10:50]. היתרון: החיפוש מתבצע בזיכרון, ולכן מהיר יותר מפתרונות חיפוש וקטורי מבוססי-דיסק.

דורביליות — שני מודים

ההכרזה החדשה שהוצגה היא דורביליות: שמירת הנתונים על דיסק ולא רק בזיכרון, מה שפותח את השירות ל-use cases נוספים. היא קיימת בשני מודים. בדורביליות סינכרונית ה-primary node יושב ב-Availability Zone אחד ושתי read replicas בשתי AZs אחרות; ה-Multi-AZ transaction log משמש לשמירה דורבילית ולסנכרון הרפליקות. הכתיבה מגיעה ל-primary, נכתבת ל-transaction log, וה-primary ממתין לאישור שהנתונים נשמרו במספר AZs לפני שהוא מאשר ללקוח [10:50, 12:21].

המחיר: "זה מבטיח שהמידע נכתב לדיסק ולא יהיה data loss, אבל זה גם אומר שהכתיבות נעשות במילי שניות בודדות ולא במיקרו שניות" [12:21]. הרפליקות צורכות את העדכון מהלוג ומעדכנות את הזיכרון שלהן אסינכרונית, והקריאות ממשיכות להתבצע מהזיכרון במיקרו-שניות.

> — **Senior Technical Account Manager, AWS [12:21]** במוד האסינכרוני הכתיבות מהירות יותר, אך יש חלון של עד עשר שניות שבו ייתכן איבוד מידע.

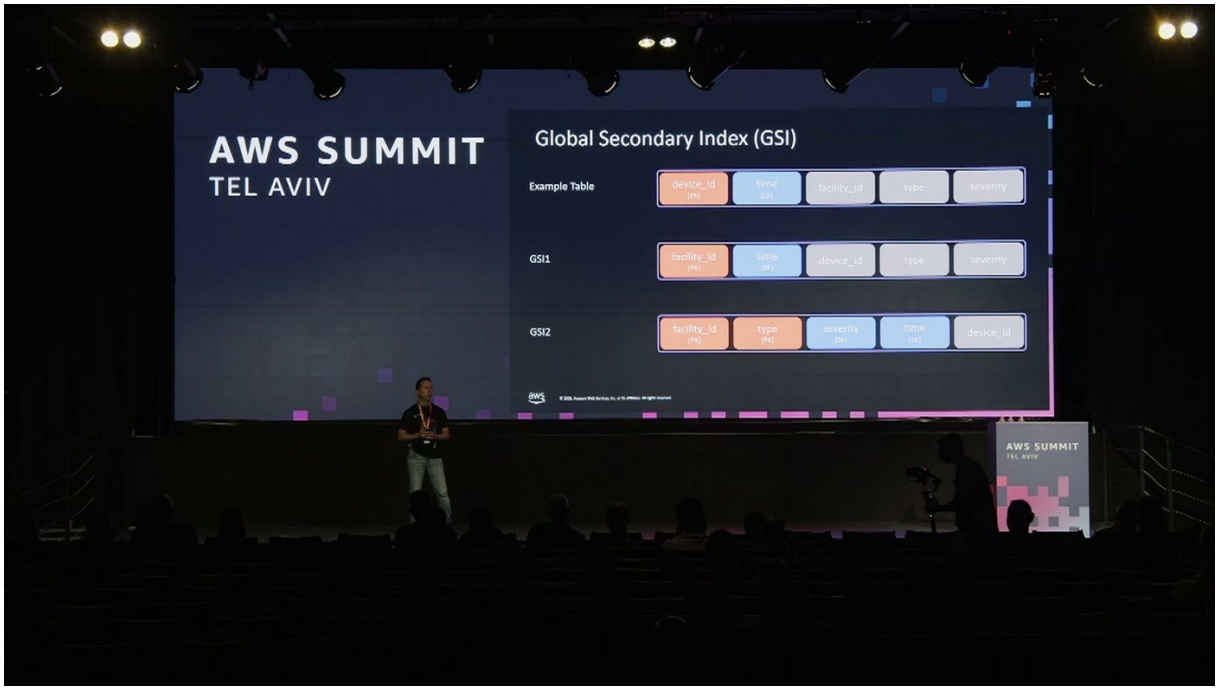
5. Amazon DynamoDB

לאוניד קורן פותח בהגדרה: DynamoDB הוא key-value database שמספק זמני תגובה קבועים של מילי-שניות בודדות בכל סקייל. השירות serverless — אין מכונות ואין cluster להגדיר, "אתם פשוט יוצרים טבלה ואחרי כמה שניות אתם מתחילים להשתמש בטבלה", והקצבים גדלים אוטומטית עם התעבורה [13:53].

מה שהוא כן ומה שהוא לא: הוא תומך בשליפה לפי מפתח וגם ב-range queries על רשומות בעלות אותו ערך partition key; הוא מאפשר לשמור ולתשאל מידע מקונן, אך לאנדקס ולתשאל ביעילות אפשר רק שדות ברמה העליונה. במפורש — הוא לא תומך באגרגציה של נתונים, בחיפוש טקסט חופשי או בשאילתות גאו-ספיישל [13:53]. הוא כן תומך ב-streaming של שינויי נתונים ובטרנזקציות ACID-compliant [15:26].

מודל הנתונים: Partition Key ו-Sort Key

טבלה מכילה items, וכל item מכיל attributes. הסכמה גמישה ולכן רשומות שונות יכולות להכיל שדות שונים. השדה היחיד שחובה להגדיר ביצירת הטבלה הוא ה-partition key, שקובע איך הרשומות מתפזרות על פני ה-partitions. ה-sort key הוא אופציונלי — אך אם הוגדר, הוא חייב להופיע בכל אחת מהרשומות. הוא תומך בקשרים של one-to-many ומאפשר range queries כמו גדול-מ, קטן-מ, between ו-begins with. יחד השניים יוצרים את המפתח הראשי הייחודי [15:26].



דוגמת ה-IoT מהבמה: טבלה עם device_id כ-PK ו-time כ-SK, ומעליה שני GSIs שמסובבים את המפתחות — facility_id ב-GSI1, ושילוב facility_id+type כ-PK מורכב ב-GSI2.

הדוגמה שנבחרה היא use case של IoT: מכשירים ששולחים דיווחים, כאשר ה-ID של המכשיר הוא ה-partition key וזמן הדיווח הוא ה-sort key, לצד שדות נוספים — ID של המתקן, סוג הדיווח וחומרתו [15:26]. מהטבלה הזו אפשר לשלוף ביעילות דיווחים של מכשיר מסוים, ולשלב עם טווח זמנים. כדי לשלוף לפי שדות אחרים מגדירים GSI — Global Secondary Index — עם partition key ו-sort key שונים משל הטבלה [17:04].

> **פרט שקל לפספס** ב-GSI אפשר להרכיב את ה-partition key ואת ה-sort key מיותר משדה אחד. "למעשה כל אחד מהם יכול להכיל עד ארבע שדות ב-GSI" [17:04] — מה שמאפשר לענות על שאלות כמו "כל הדיווחים מסוג X ממתקן Y, ממוינים לפי חומרה וזמן" בלי לשכפל טבלאות.

חיפוש וקטורי — יכולת בת חודש

"בדיוק לפני חודש, DynamoDB התחיל לתמוך בחיפוש וקטורי" [17:04]. היכולת נשענת על אותם עקרונות של השירות — זמני תגובה קבועים בכל סקייל, serverless, תשלום לפי שימוש — ואפשר להוסיף אינדקס וקטורי גם על טבלה חדשה וגם על טבלה קיימת [18:39]. ה-use cases שנמנו: חיפוש סמנטי, RAG, המלצות לפי התנהגויות קודמות, זיכרון אג'נטי, וזיהוי אנומליות והונאות.

Principal NoSQL Solutions Architect, AWS [18:39] — כבר אין צורך לשלב את DynamoDB עם דאטאבייס וקטורי נפרד בשביל היכולות האלה, אלא פשוט אפשר להריץ אותם ישירות על הנתונים שלכם ב-DynamoDB.

ה-flow זהה בשלושת השירותים: יוצרים אינדקס וקטורי על הטבלה; בזמן הכתיבה משתמשים בשירות כמו Amazon Bedrock כדי לייצר את הייצוג הווקטורי, וכותבים אותו כשדה נוסף ברשומה; בזמן החיפוש מייצרים אמבדינג לטקסט החיפוש ומריצים את החיפוש מול הטבלה כדי למצוא את הרשומות הקרובות ביותר סמנטית [18:39].

רפליקציה גלובלית: MR-EC מול MR-SC

כאן, לדברי הדובר, נמצאות יכולות שאין בשני השירותים האחרים. DynamoDB תומך בשני מצבים של טבלה גלובלית: Multi-Region Eventual Consistency (MR-EC) ו-Multi-Region Strong Consistency (MR-SC) [20:15].

פרמטר	MR-EC (דיפולט)	MR-SC
סוג הרפליקציה	אסינכרונית	סינכרונית
זמני כתיבה וקריאה strongly-consistent	נמוכים יותר	גבוהים יותר
מה רואה קריאה strongly-consistent	עלולה להחמיץ כתיבה מקבילה באזור אחר	תמיד מידע עדכני
RPO	בדרך כלל עד שניות בודדות	אפס
מספר אזורים	כל מספר של אזורים	עד שלושה אזורים

הסיבה לפער ב-latency ב-MR-EC היא שהפעולות מתייחסות אך ורק למצב ב-region המקומי, ואינן בודקות אם התרחשה במקביל פעולה רלוונטית באזור אחר [20:15]. מנגד, ב-MR-SC "ה-RPO כאן הוא אפס, מה שאומר שגם אם יהיה כאן disaster recovery גלובלי — עדיין מובטח שלא יהיה פה שום איבוד מידע" [21:51].

6. Amazon DocumentDB

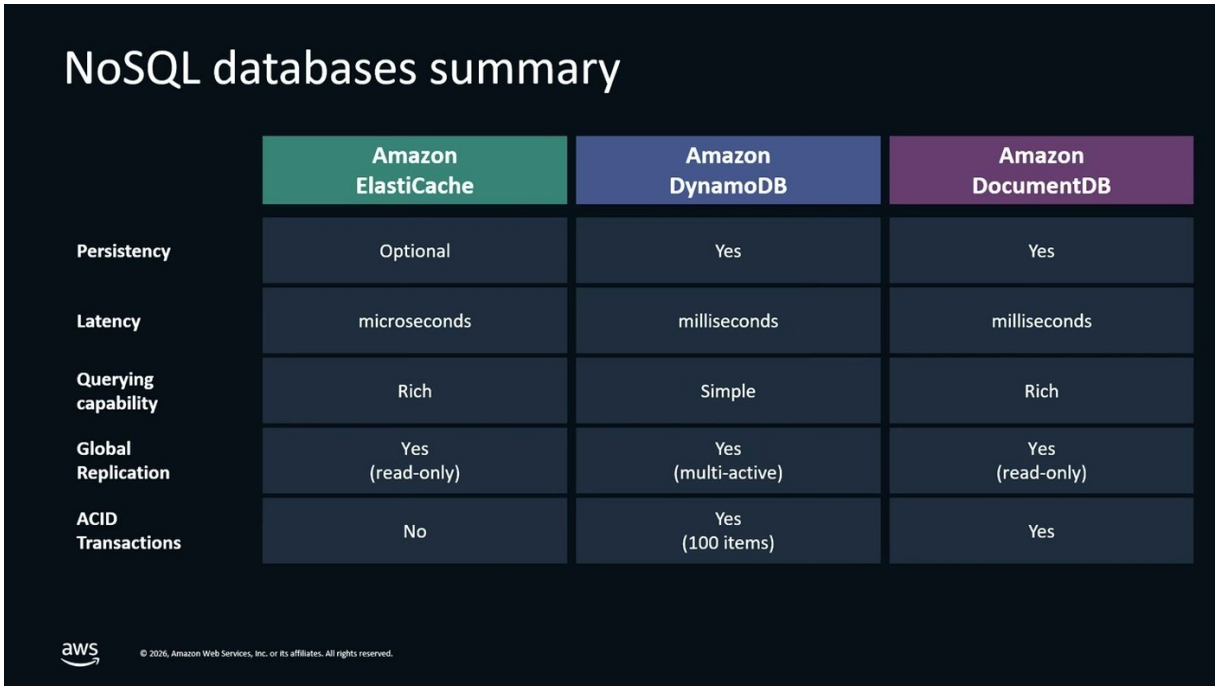
DocumentDB הוא document database, וככזה הוא מספק את יכולות האינדוקס והתשאול העשירות ביותר מבין השלושה. הוא תומך ב-API של MongoDB, כך שאפליקציה שעובדת מול MongoDB תוכל לעבוד מולו ללא שינוי קוד או עם מעט מאוד שינוי [21:51]. הוא אופטימלי לשמירת מבני נתונים מקוננים כמו JSON, ומאפשר לאנדקס ולתשאל ביעילות שדות ברמה העליונה, שדות מקוננים וגם מערכים — ותומך באגרגציה, ב-joins בין טבלאות, בשליפות גאו-ספיישל, בחיפוש טקסט חופשי ובחיפוש וקטורי [23:23].

הארכיטקטורה: הפרדת עיבוד מאחסון

השירות תוכנן עבור הענן, ומפריד בין שכבת עיבוד הנתונים לשכבת אחסון הנתונים. כל שכבה יכולה לגדול ולקטון עצמאית. בגלל שהנתונים אינם נשמרים בשכבת העיבוד, אפשר להוסיף ולהחליף מכונות ב-cluster "תוך פחות מ-10 דקות ללא תלות בכמות הנתונים שיש לנו" [23:23].

מספרי התקרה של cluster עד 256TB נתונים, לפחות עשרות אלפי כתיבות בשנייה, ומיליוני קריאות בשנייה [23:23]. גם כאן יש אופציית Serverless, ואפשר להגדיר cluster גלובלי עם primary region אחד לקריאה וכתיבה, ואחד או יותר secondary regions לקריאה בלבד.

7. טבלת ההשוואה המסכמת



הטבלה המלאה כפי שהוצגה בסוף החלק של AWS: Persistency, Latency, Querying capability, Global Replication ו-ACID Transactions על פני שלושת השירותים.

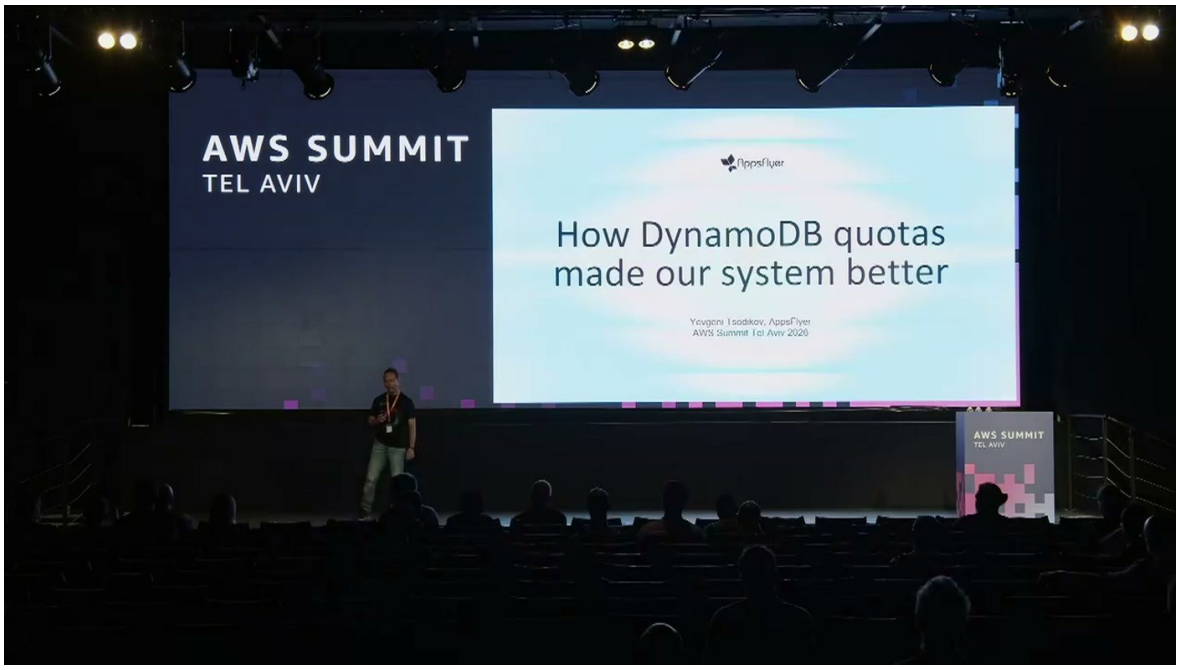
הסלייד הזה מרוכז כל מה שנאמר בפרקים הקודמים. ב-ElastiCache הנתונים כברירת מחדל נשמרים רק בזיכרון, כי הייעוד העיקרי הוא caching; הדורביליות קיימת אך "יש לזה השלכה על זמני הכתיבות ועל העלות" [24:55]. ב-DynamoDB וב-DocumentDB הנתונים תמיד נשמרים בצורה דורבילית.

- **Latency:** ב-ElastiCache מיקרו-שניות לקריאה ולכתיבה, אלא אם שומרים דורבילית בצורה סינכרונית. ב-DynamoDB וב-DocumentDB מילי-שניות — כאשר ב-DynamoDB הזמנים אינם מושפעים מהסקייל ונשארים קבועים, וב-DocumentDB הסקייל ובעיקר אופי השליפות כן יכולים להשפיע [24:55].
- **עושר התשאול:** DynamoDB — פשוט יחסית. ElastiCache ו-DocumentDB — עשיר יותר, כאשר DocumentDB הוא העשיר מבין השלושה [26:28].
- **רפליקציה גלובלית:** שלושתם תומכים, אבל DynamoDB הוא היחיד שמאפשר גם לכתוב וגם לקרוא במקביל מכמה אזורים, וגם לבחור בין רפליקציה סינכרונית לאסינכרונית [26:28].

- **טרנזקציות:** DynamoDB ו-DocumentDB תומכים בטרנזקציות ACID-compliant על רשומות מרובות. ElastiCache תומך בטרנזקציות "אבל חלק ממאפייני ה-ACID חסרים שם" [26:28].
- **חיפוש וקטורי:** בשלושתם. ההמלצה — להריץ אותו מול השירות שבו הנתונים כבר נמצאים, לשמירת ארכיטקטורה פשוטה.

> **Principal NoSQL Solutions Architect, AWS [26:28] —** אני כן אציין ש-ElastiCache יבצע את החיפוש הווקטורי הכי מהיר, כי הכל מתבצע בזיכרון, ו-DynamoDB יתמוך בחיפוש וקטורי בסקייל הכי גבוה.

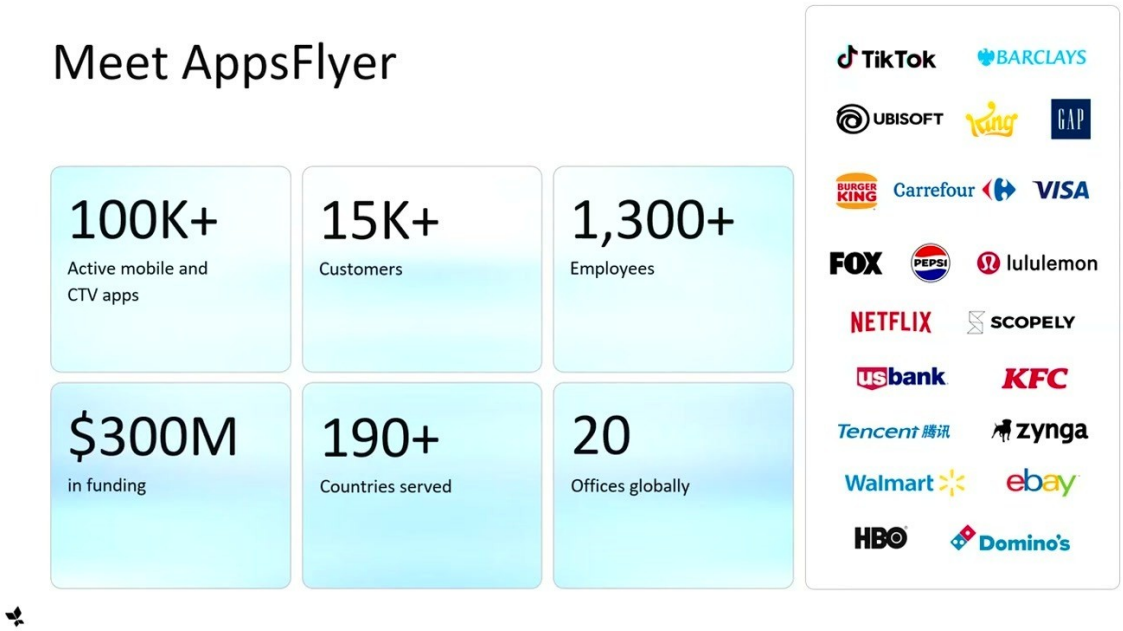
8. המקרה של AppsFlyer: איך קווטות שיפרו את המערכת



כותרת החלק השלישי. הטענה המרכזית של הדובר היא בכיוון הפוך מהאינטואיציה: המגבלות של השירות היו מה שתיקן את הארכיטקטורה.

אבגני צודיקוב, Staff Software Engineer ב-AppsFlyer עם מעל 16 שנים בתעשייה ושש שנים בחברה, הוביל את המיגרציה מ-self-managed clusters ל-DynamoDB [26:28].

מה המערכת עושה ובאיזה סקייל

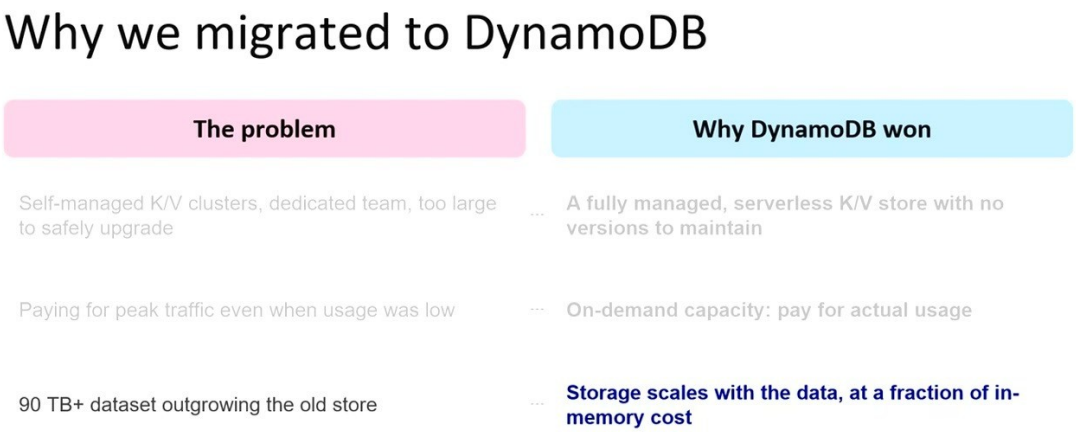


מספרי החברה כפי שהוצגו על הסלייד: 100K+ אפליקציות מובייל ו-CTV פעילות, 15K+ לקוחות, 1,300+ עובדים, $300M גיוס, 190+ מדינות ו-20 משרדים.

AppsFlyer היא פלטפורמה למדידת קמפיינים של פרסום — לחצתם על פרסומת בטלפון, או התקנתם אפליקציה אחרי שראיתם מודעה בטלוויזיה, והם מדדו את זה. המערכת שעליה נסוב הסיפור היא Identity Resolution: בכל התקנת אפליקציה מתקבלים מספר מזהי מכשיר, והמשימה היא לקבוע שכולם שייכים לאותו אדם — "וזה מה שאנחנו עושים מיליוני פעמים בשנייה" [27:59].

> **Staff Software Engineer, AppsFlyer [27:59] —** יש לנו מעל 350 מיליארד רשומות, 90 טרה של דאטה, ואנחנו מבצעים מעל מיליון וחצי קריאות בשנייה ובערך 200 אלף כתיבות בשנייה.

למה עזבו את ה-self-managed



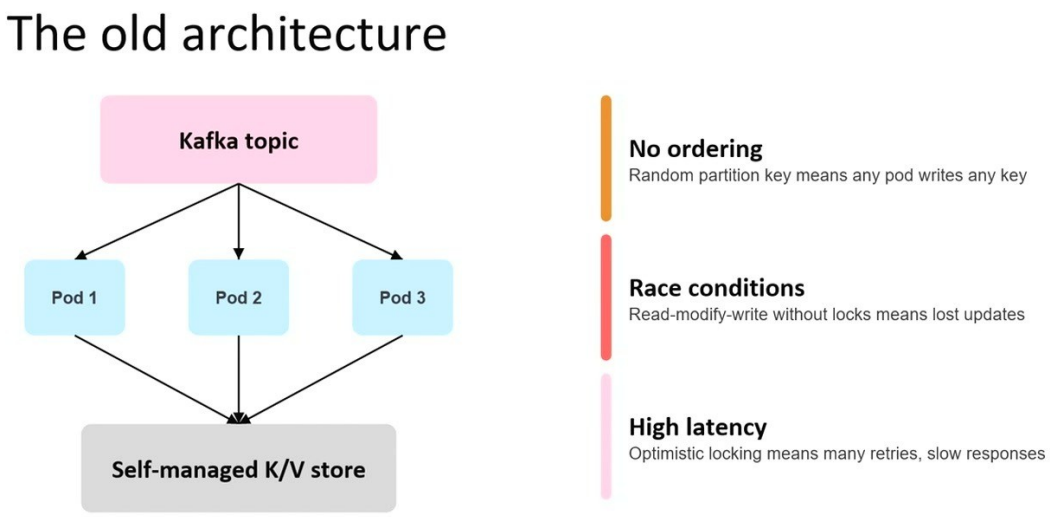
שלוש שורות של בעיה מול פתרון. השורה המודגשת — היקף הדאטה של 90TB+ שהמערכת הישנה לא עמדה בו, מול אחסון שגדל עם הנתונים בשבריר מעלות הפתרונות In-Memory.

המצב לפני המיגרציה: שני clusters self-managed בגודל 100–200 nodes כל אחד. "הם self-managed, כלומר, אנחנו ה-self, אנחנו צריכים לנהל אותם" — צוות ייעודי שאחראי עליהם, ושדרוג גרסה של הדאטאבייס, של ה-nodes או של ה-EKS מאחורי הקלעים הוא אירוע שמתכוננים אליו מספר ספרינטים מראש [27:59].

> **Staff Software Engineer, AppsFlyer [29:30] —** ב-DynamoDB, המוצר הוא מנוהל לחלוטין. אני עושה get, אני עושה put, והנושא של שדרוגי גרסה זאת בעיה של AWS, שקוף מבחינתי.

- **בזבוז על peak:** עם nodes אי אפשר לעשות auto-scaling גמיש, ולכן ה-clusters אוישו לפי ה-peak traffic ועוד margin — "זה אומר שיש הרבה בזבוז ואנחנו משלמים הרבה כסף לשווא". מול זה, מודל ה-on-demand capacity ב-DynamoDB גובה לפי שימוש בפועל [29:30].
- **תקרת נפח:** 90TB דאטה הגיעו למגבלות הלוגיות והפיזיות של ה-clusters. בDynamoDB הטבלה גדלה יחד עם כמות הדאטה, ובהשוואה לפתרונות in-memory אחרים ב-AWS — "כל זה בשבריר של העלות" [29:30].
- **גיבוי קונסיסטנטי:** איך מגבים אלפי partitions על פני nodes רבים כך שהגיבוי יהיה קונסיסטנטי בינו לבין עצמו? Point-in-Time Recovery מאפשר חזרה לכל נקודה ב-35 הימים האחרונים ברזולוציה של שנייה, ובגלל שהוא מבוסס change log הוא מבטיח קונסיסטנטיות [29:30, 31:03].

הארכיטקטורה הישנה — וה-race condition שחיו איתו

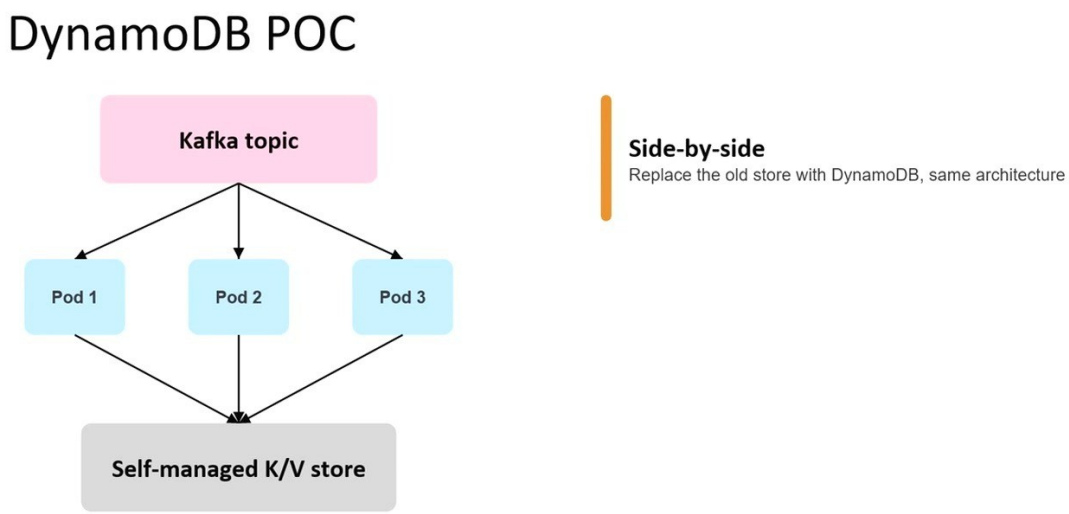


הזרימה הישנה: Kafka topic ללא סדר, שלושה pods שכל אחד יכול לטפל בכל אירוע, ומתחתם ה-self-managed K/V store. משמאל שלוש התוצאות — אין סדר, race conditions ו-latency גבוה.

הודעות נכנסות ל-topic בקפקא שאין לו partition key, כלומר אין סדר בין האירועים וכל pod יכול לטפל בכל אירוע. מכאן ששני pods יכולים לכתוב במקביל עבור אותו מפתח — race condition. המערכת עובדת בתבנית read-modify-write, ואם שני pods מקבלים החלטה שונה וכותבים אותה, נוצר איבוד מידע [31:03].

הפתרון שהיה במקום: optimistic locking — קוראים את הגרסה של הרשומה, ובמעמד הכתיבה, אם הגרסה השתנתה, עושים retry לכל התהליך. "זה עובד, אבל החיסרון הוא שיש איזושהי איטיות במערכת, והלקוח לפעמים מרגיש את זה. אבל זה עובד, וזה עובד שנים, ולא רצינו לגעת בזה" [31:03].

ה-POC שנשבר ביום הראשון



הכוונה המקורית: להחליף רק את שכבת האחסון ולהשאיר את שאר הארכיטקטורה זהה — side-by-side, אותם pods, אותו זרם הודעות.

"וכשהתחלנו את ה-POC, רצינו לקחת את המערכת בדיוק כמו שהיא, ולהריץ את DynamoDB בצורה של side by side" — פשוט להחליף את ה-self-managed ב-DynamoDB. "ואיך שאנחנו התחלנו את ה-POC? ביום הראשון שלו התחלנו לקבל שגיאות של throttling מהטבלה שלנו" [31:03].

> **הקווטה שהפילה את ה-POC** DynamoDB מאפשר virtually unlimited scale, אבל לכל partition יש קווטה משלו: "כל partition מאפשר לעשות אלף כתיבות בשנייה ושלושת אלפים קריאות בשנייה" [32:36]. הכתיבות של AppsFlyer, שהתפזרו ללא סדר, נגסו בקווטה הזו — והשגיאות הצביעו על ה-hot partitions.

בנקודה הזו היה אפשר להוסיף retry על ה-exception ולהמשיך הלאה, כפי שנעשה קודם עם ה-optimistic locking. ההחלטה הייתה הפוכה: "אנחנו קיבלנו החלטה לעבוד בצורה שונה ולא לחזור על אותה החלטה, אלא לטפל בשורש הבעיה" — ולהיכנס לרידיזיין [32:36].

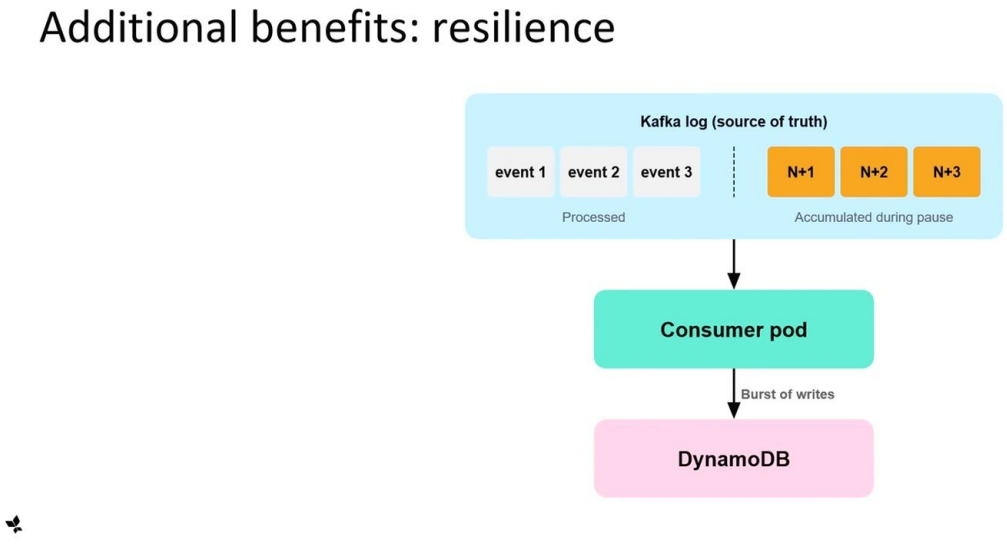
הרידיזיין: יישור partition keys בין קפקא ל-DynamoDB

התובנה: קפקא, כמו DynamoDB, היא גם מערכת partitioned — ואפשר ליישר בין ה-partition keys של שתי המערכות. וזה מה שנעשה: אותו string שימש כמפתח בהודעה בקפקא וב-item ב-DynamoDB. התוצאה — כל ההודעות עם אותו מפתח הגיעו לאותו partition, ולכן לאותו consumer ולאותו pod. ומכאן אפשר היה לטפל בכל מפתח בthread אחד [32:36].

> **Staff Software Engineer, AppsFlyer [32:36] —** בגלל שאין race condition, אנחנו לא צריכים יותר את הנעילות. ה-latency ירד והמערכת עכשיו בריאה יותר. פודים לא דורכים אחד לשני על הרגליים.

מעבר לביטול הנעילות, ה-fleet יכול כעת לגדול ולקטון יחד עם העומס בצורה ששומרת על הגבלת העומס לכל partition ולכל מפתח: "בגלל שאנחנו עושים את זה אין יותר throttling, אין יותר hot partitions" [34:12].

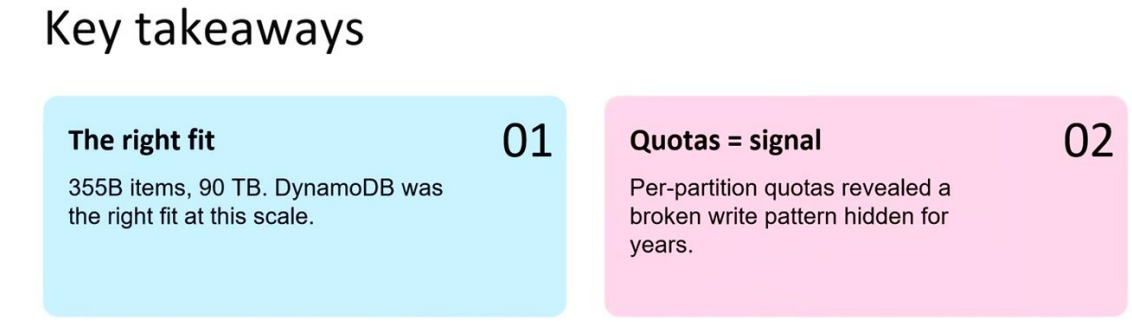
בונוס לא צפוי: resilience



לוג הקפקא כמקור אמת: אירועים שנצברו במהלך עצירה נשטפים דרך ה-consumer pod אל DynamoDB כ-burst של כתיבות, וה-on-demand capacity עולה איתו.

לפעמים צריך לעשות replay למידע היסטורי, או לעצור את הטראפיק לשעה ולחזור לטפל בו מאוחר יותר. אפליקטיבית, שני המקרים נראים כמו burst של טראפיק. הממצא: כיוון שהעבודה היא ב-on-demand capacity, "כשה-demand עולה גם ה-capacity עולה איתו ביחד והטבלה מתמודדת בצורה שקופה לחלוטין עם שני ה-bursts האלה" — בלי provisioning מראש ובלי rate limiter שנועד למנוע קריסה של הטבלה [34:12].

הסייג שהדובר עצמו מוסיף: זה עובד כי DynamoDB בנוי לזה, "אבל גם בגלל שעכשיו הארכיטקטורה שלנו בריאה יותר ותואמת לשיטת עבודה הזאת" [34:12].



שתי המסקנות הראשונות מהסלייד המסכם: ההתאמה הנכונה בסקייל של 355B items ו-90TB, והקווטות כאות אזהרה שחשף דפוס כתיבה שבור בן שנים.

שימו לב לפער קטן: על הסלייד מופיע 355B items, בעוד שבדיבור נאמר "מעל 350 מיליארד רשומות" [27:59, 34:12]. בסיכום שבעל-פה נמנו שלוש מסקנות: בסקייל הזה DynamoDB הוא גם הפתרון הנכון וגם הכלכלי ביותר מבין ה-NoSQL offerings של AWS; הקווטות הציפו בעיה ארכיטקטונית קיימת; ומודל התשלום לפי שימוש חסך כסף וגם איפשר התמודדות שקופה עם bursts [34:12, 35:44].

Staff Software Engineer, AppsFlyer [35:44] — אם יש מספר מערכות מבוזרות בחברה שווה ליישר בין ה-partition keys ביניהם, כדי שהמערכת תדבר באותה שפה ותוכל לטפל בהודעות בצורה סדרתית ולפי סדר הגיוני.

9. סגירה: AI Skills לדאטאבייסים

שני ה-QR codes שהוצגו בסיום — משמאל skill לבחירת הדאטאבייס המתאים, ומימין אוסף skills פר-שירות.

לפני הסיום הוצגה יכולת חדשה של AI Skills לדאטאבייסים של AWS, שמטרתה "לעזור ל-agent להשתמש בשירותים בצורה אופטימלית" [35:44]. הוצגו שני קודים לסריקה: האחד מפנה ל-skill שעוזר לבחור את הדאטאבייס המתאים לצרכים, והשני לאוסף skills פר-דאטאבייס — DynamoDB, ElastiCache, DocumentDB ועוד.

10. מה לוקחים מכאן

- **התחילו מה-access pattern, לא מהשם.** השאלה הראשונה אינה "איזה דאטאבייס הכי טוב" אלא "האם אני שולף לפי מפתח, או שאני צריך אגרגציה, joins וטקסט חופשי". התשובה הזו לבדה מכריעה בין DynamoDB ל-DocumentDB.
- **דורביליות ב-ElastiCache מחייבת החלטה מודעת.** אם מפעילים אותה, בחרו בין הסינכרוני (אפס איבוד, כתיבות במילי-שניות) לאסינכרוני (כתיבות מהירות, חלון סיכון של עד 10 שניות). הבחירה הזו היא החלטת מוצר, לא הגדרת תשתית.
- **אל תוסיפו דאטאבייס וקטורי רק בשביל וקטורים.** שלושת השירותים כבר תומכים בחיפוש וקטורי; ההמלצה מהבמה היא להריץ אותו היכן שהנתונים כבר יושבים. ElastiCache מהיר יותר, DynamoDB סקלאבילי יותר.
- **RPO אפס גלובלי עולה בגמישות טופולוגית.** MR-SC נותן אפס איבוד מידע גם בתרחיש DR גלובלי, אבל מגביל לשלושה אזורים ומייקר את ה-latency. MR-EC פותח כל מספר אזורים במחיר RPO של שניות.
- **בחנו קווטות לפני שמוסיפים retry.** שיעור המפתח מ-AppsFlyer: שגיאת throttling מ-DynamoDB היא לעיתים אבחון ולא תקלה. הקווטה של 1,000 כתיבות ו-3,000 קריאות ל-partition היא מה שחשף דפוס כתיבה שבור.
- **יישרו partition keys בין מערכות מבוזרות.** כשקפקא ו-DynamoDB חולקים את אותו מפתח, ה-race conditions נעלמים מעצמם — יחד עם הנעילות, ה-retries וה-latency שהן גררו.
- **on-demand capacity הוא גם כלי resilience.** מעבר לחיסכון, הוא ספג replay של מידע היסטורי ו-catch-up אחרי עצירת טראפיק בלי provisioning ובלי rate limiter.

11. מילון מונחים

מונח	מה זה, כפי שהוצג בסשן
Valkey	Fork קוד-פתוח של Redis 7.2 תחת ה-Linux Foundation, תואם מלא ל-Redis. המנוע הדיפולטיבי ב-ElastiCache, כ-20% זול יותר.

מונח	מה זה, כפי שהוצג בסשן
Data Tiering	שמירת נתונים ב-ElastiCache לא רק בזיכרון אלא גם על ephemeral SSDs, להוזלת עלות.
Partition Key / Sort Key	PK קובע את פיזור הרשומות על ה-partitions; SK (אופציונלי) קובע את סדרן בתוך אותו PK ומאפשר range queries. יחד — המפתח הראשי.
GSI	Global Secondary Index — אינדקס עם PK ו-SK משלו, שמאפשר שליפות לפי שדות אחרים. כל אחד מהם יכול להורכב מעד ארבעה שדות.
MR-EC	Multi-Region Eventual Consistency — טבלה גלובלית ברפליקציה אסינכרונית. RPO של שניות בודדות, ללא הגבלת מספר אזורים.
MR-SC	Multi-Region Strong Consistency — טבלה גלובלית ברפליקציה סינכרונית. RPO אפס, עד שלושה אזורים.
PartiQL	שפת שאילתות תואמת-SQL שנתמכת ב-DynamoDB לצד ה-HTTP APIs.
PITR	Point-in-Time Recovery — חזרה לכל נקודה ב-35 הימים האחרונים ברזולוציה של שנייה, מבוססת change log ולכן קונסיסטנטית.
On-demand capacity	מודל חיוב ב-DynamoDB לפי שימוש בפועל, שמתאים את ה-capacity לביקוש ללא provisioning מראש.
Throttling	דחיית בקשות כשחורגים מקווטת ה-1,000 partition — כתיבות ו-3,000 קריאות בשנייה.
Hot partition	partition שמקבל חלק לא פרופורציונלי מהתעבורה ולכן חוטף throttling לפני השאר.
Optimistic locking	כתיבה מותנית בגרסת הרשומה, עם retry אם הגרסה השתנתה. הפתרון הישן של AppsFlyer ל-race conditions.