ElastiCache for Valkey — יותר מסתם קאש

AWS Summit Tel Aviv 2026 ,סיכום סשן — TLV-DAT308

Yehonatan Yulazari, Sr. Software Dev Engineer, AWS | Shoval Cohen, Algo Engineering Team Lead, Via | Eran Balan, Sr. Specialist Solutions Architect, AWS

10 בספטמבר 2026 | משך: 42 דקות | הופק מהקלטת הסשן

מסלול: Databases on AWS | רמה: 300

על המסמך הזה

המסמך מסכם את הסשן TLV-DAT308 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 בארבעים ושתיים הדקות. חותמות הזמן בגוף המסמך מפנות לנקודה המדויקת בהקלטה.

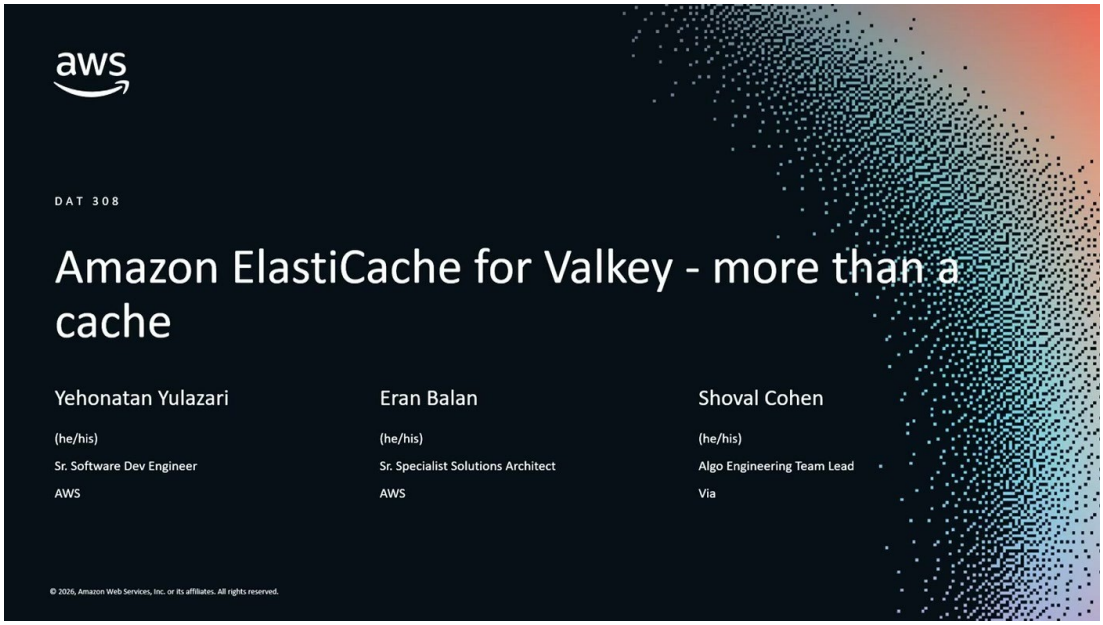
איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. מה הוצג ומה שווה לבדוק אצלנו.
- **פרקים 2–4 — המנוע:** מה זה Valkey, המעבר לריבוי תהליכונים, ומה עשו לצריכת הזיכרון.
- **פרקים 5–7 — המקרה של Via:** בעיית ה־hot slot, שינוי הארכיטקטורה, והתוצאה הכלכלית.
- **פרקים 8–11 — הפיצ'רים החדשים:** Bloom Filter, Vector Search וקאשינג סמנטי, חיפוש ואגרגציות, ו־ElastiCache Durable.
- **פרק 12 — מה לוקחים מכאן:** מסקנות מעשיות, ואחריהן מילון מונחים.

הערת מקור המסמך הופק מתמלול אוטומטי של ההקלטה. התמלול מדויק בקטעים המקצועיים, אך שמות ומונחים לועזיים מופיעים בו בשיבוש פונטי (למשל "וולקי" = Valkey, "אלסטיקש" = ElastiCache) ותוקנו כאן מול הסליידים. הציטוטים מובאים כלשונם בתמלול, כולל השיבושים, ואומתו מול חותמת הזמן המצוינת. מספרים שמופיעים על הסליידים צוינו ככאלה.

1. תקציר מנהלים

שלושה דוברים, מבנה של שלושה חלקים: מפתח מצוות ElastiCache בתל אביב הסביר מה השתנה במנוע עצמו, ראש צוות אלגוריתמים ב-Via סיפר מה קרה כשהחליפו ארכיטקטורה ומנוע בפרודקשן, ו-Solutions Architect סקר את הפיצ'רים שהפכו את Valkey ממאגר מפתח-ערך למשהו רחב יותר. החוט המקשר, כפי שנוסח בפתיחה, הוא תפעולי ולא טכנולוגי: להפסיק לקום בשתיים בלילה.



שקופית הפתיחה עם שלושת הדוברים ותפקידיהם

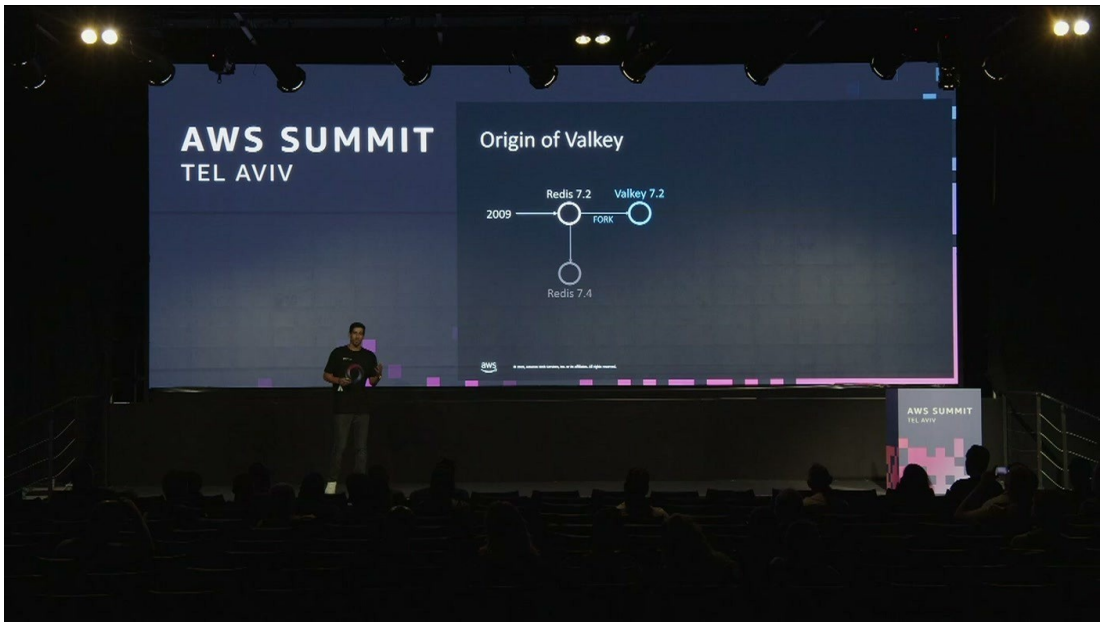
- **המעבר לריבוי תהליכונים נתן פי 3.3 בתפוקה.** הפרדת פעולות ה-IO לתהליכוני IO ייעודיים, בזמן שהפקודות עצמן ממשיכות לרוץ בתהליכון יחיד: "ראינו שיפור של 230% בביצועים" [6:12]. הסלייד מציג 360,460 בקשות לשנייה ב-Valkey 7.2 מול 1,190,000 ב-Valkey 8.0.
- **התקורה על כל מפתח ירדה בכ-50%.** שכתוב מבנה ה-hash table חיסל שלושה מצביעים לכל רשומה: "עכשיו זוכרים שהיה 52, אז יש פה חיסכון של 50% לכל מפתח ווויליו שלכם" [12:22]. אצל לקוח אמיתי זה התבטא ב-41.4% פחות זיכרון על אותה כמות דאטה.
- **Via חתכה 68% מעלות ה-ElastiCache.** "אנחנו משלמים 68% פחות מהעלות שישלמנו בעבר" [23:05]. החיסכון הגיע משלושה מקורות: מכונות קטנות ומדויקות יותר, מחיר נוד נמוך יותר ב-Valkey, ואופטימיזציית הזיכרון של המנוע.
- **האופטימיזציה שהצילה את הלייטנסי היא זו שחנקה את הסקייל.** קיבוע עיר ל-slot נתן round trip אחד לשליפה, אבל יצר shard חם שהגיע לתקרת הזיכרון בזמן שהאחרים ישבו חצי ריקים. "המצב הזה תפרס אותנו כמובן באמצע הלילה" [18:31].
- **ההחלפה עצמה מ-Redis ל-Valkey הייתה שורה ב-Terraform.** "הלכנו לטרפורם, שינינו מ-Redis ל-Walky, עשינו אפלי וזה פשוט עבר" [23:05] — אותו API, ללא שינוי קוד וללא השפעה על הלקוח.
- **Valkey כבר לא רק קאש.** Bloom Filter הסתברותי, Vector Search בזיכרון לקאשינג סמנטי, full-text search ואגרגציות על האינדקס, ו-ElastiCache Durable שנותן zero data loss במוד סינכרוני.

רלוונטיות לארגון

שלוש נקודות שמצדיקות בדיקה פנימית. ראשית, שדרוג מנוע מ-Redis 7.2 ל-Valkey הוא drop-in replacement שמוריד עלות נוד ב-20% לפני שנגענו בכלום. שנית, אם יש אצלנו קלאסטרים עם hash tags שמקבעים לקוח או ישות ל-slot — כדאי להסתכל על מדדי ה-shard הבודד ולא על ממוצע הקלאסטר. שלישית, ElastiCache Durable פותח את הדלת לשימושים שעד היום נפסלו בגלל חשש מאובדן דאטה, במחיר של Write Latency חד-ספרתי במילישניות במקום תת-מילישנייה.

2. מאיפה Valkey הגיע

הפתיחה הייתה בשאלה לקהל — מי כבר הועף מהמיטה בגלל קאש שנפל: "כל מי שיצא לו שאירו אותו פה בשתיים בלילה או שלוש בלילה" [0:00]. מכאן עברו להגדרות. ElastiCache הוא שירות מנוהל של מסדי נתונים בזיכרון, עם לייטנסי תת-מילישנייה, פריסה multi-AZ ותמיכה בשלושה מנועים: Memcached, Redis OSS ו-Valkey.



ציר הזמן: Redis ב-2009, הפיצול והפורק ב-2024, וגרסאות Valkey שבאו אחריו

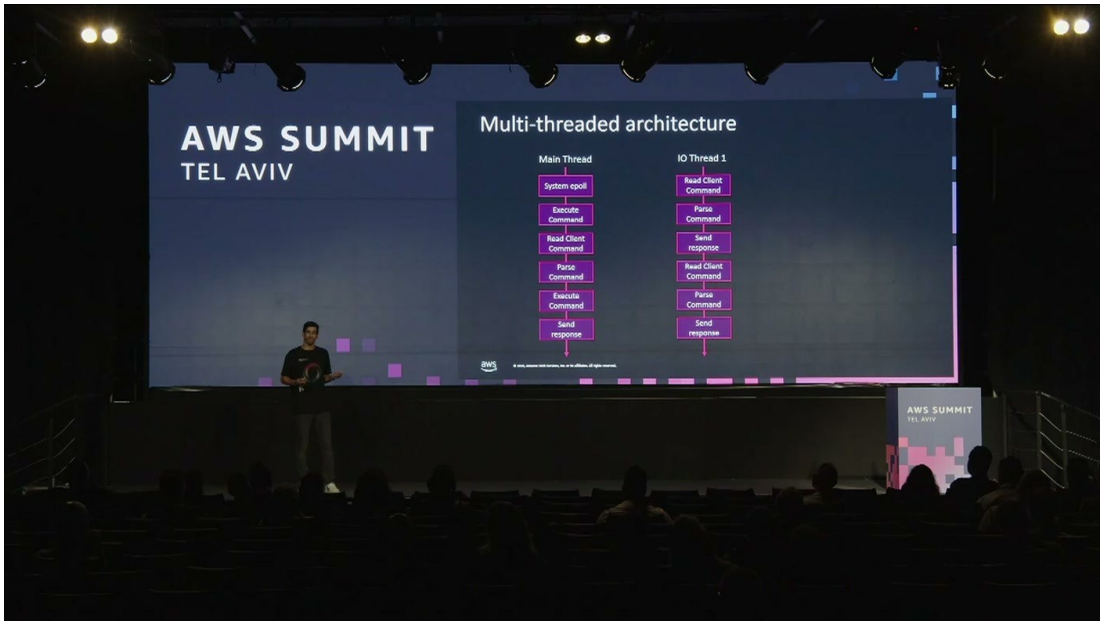
ההיסטוריה בקצרה: Redis יצא כקוד פתוח ב-2009. במרץ 2024, בגרסה 7.2, שונה הרישיון — "רדיס החליטו לשנות את הרישיון שלהם, ופה בעצם הקהילה החליטה להתפצל" [1:36]. הפורק נקרא Valkey. גרסאות 8 ו-8.1 הביאו את השינויים הארכיטקטוניים שעליהם נסוב החלק הראשון של הסשן; גרסאות 9 ו-9.1 הביאו את הפיצ'רים שנסקרו בחלק האחרון.

ארבע סיבות הוצגו לכך שהפרויקט תפס. רישוי — "הוא תחת BSD License, והוא תחת Linux Foundation, מה שאומר שהוא ישאר Open Source" [1:36]. מודל ממשל שקובע את כיוון הפיתוח. תאימות — drop-in replacement ל-Redis 7.2, בלי שינוי קוד. וקהילה. המספרים שהוצגו: "מעל 200 מיליון הורדות של קונטיינרים, מעל 1500 קומיטים" [3:10], תמיכה של 50 ארגונים ומעל 10 ספקים שמציעים אותו כשירות מנוהל — ביניהם Google Cloud, Alibaba Cloud, Oracle, Huawei, Snap ו-Ericsson.

3. ריבוי תהליכונים: אופציה שלישית מול scale-up ו-scale-out

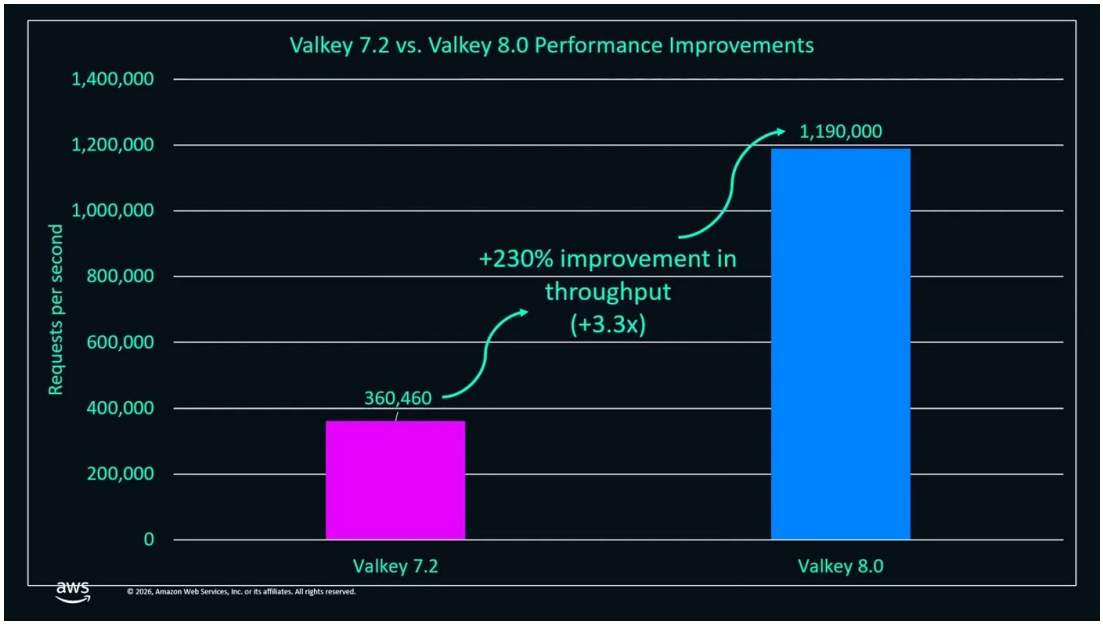
הדוגמה שנפרסה: עומס קאשינג שטוח עם פיק יומי — חברת משלוחי אוכל עם שיאים בארוחות, או חברת גיימינג עם טורנירים. בוחרים מכונה שמחזיקה את הפיק, במקרה הזה cache.M7G.2xlarge, ורואים שהפיק כמעט נוגע בתקרה. משם שתי אפשרויות מוכרות — scale-up למכונה גדולה יותר, או scale-out לפיזור הדאטה על יותר מכונות. האופציה השלישית שהוצעה: לנצל טוב יותר את החומרה שכבר יש.

הבייסליין: Valkey 7.2 היה single-threaded. תהליכון אחד מאזין לרשת, קורא את הבקשה, מבצע לה parsing, מריץ אותה וכותב את התשובה חזרה. "ראינו שקרוב ל-70% מכוח המחשוב בעצם הולך על כל הפעולות [4:41] "IO — ואת אלה אפשר להוציא החוצה.



התהליכון הראשי מול תהליכוני ה-IO, וחלוקת האחריות ביניהם

כך נולדו ה-IO threads. הקריאה מהרשת, ה-parsing והכתיבה חזרה עוברים לתהליכונים ייעודיים, בעוד הפקודות עצמן ממשיכות לרוץ במיין-תרד. זו נקודת התכנון המרכזית: אין צורך במנגנוני סינכרון מסובכים, ולכן הבטחת הלייטנסי התת-מילישנייתי נשמרת. המנגנון דינמי — עוד עבודה, עוד תהליכון.



מדידת המעבדה: 360,460 בקשות לשנייה מול 1,190,000, שיפור של 230% בתפוקה

תנאי המדידה שהוצגו: מכונת C7GN.16xlarge, 3 מיליון מפתחות, payload של 512 בתים לכל אחד. "ראינו שיפור של 230% בביצועים. שכל מי שפה מפתח יודע ששלוש ספרות זה שיפור מטורף" [6:12]. המשמעות המעשית בדוגמה: אפשר לרדת מ-2xlarge ל-xlarge ועדיין להגיע לאזור 400,000 בקשות לשנייה.

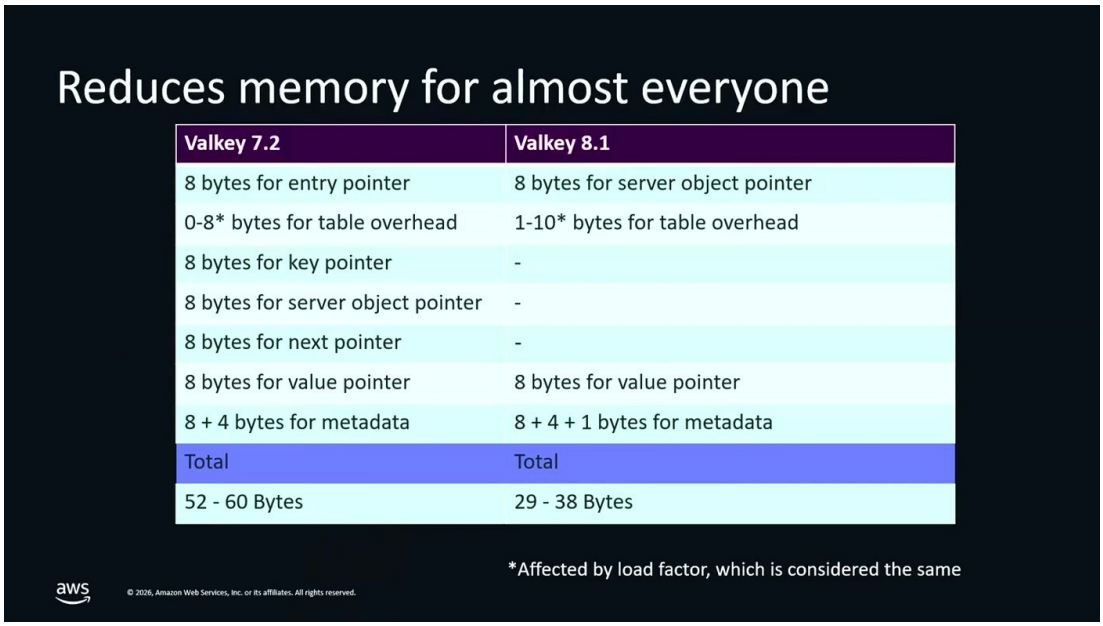
ומה שהשיפור הזה מאפשר ב-Serverless

ארכיטקטורת ElastiCache Serverless שהוצגה: ה-VPC של הלקוח מתחבר לסט NLB-ים multi-AZ ב-VPC של AWS, שמזינים צי של פרוקסים, שמנתבים לשרדים של הקלאסטר. כשמזוהה פיק, השלב הראשון הוא vertical scaling — הוספת ליבות למכונה והרמת IO threads נוספים: "מ-30,000 בקשות בשנייה, ל-240,000 בקשות בשנייה, וכל הגדילה הזאת קורה תוך חמש שניות" [7:45]. כשזה מוצה, עוברים ל-scale-out והעברת slot למכונה נוספת מה-warm pool. הסיכום המספרי: "וסך הכל בסרוויילס אפשר לגדול מ-30 אלף בקשות לשנייה למעל מיליון תוך פחות מ-5 דקות" [9:17].

4. הזיכרון: איך חיסלו 24 בתים לכל מפתח

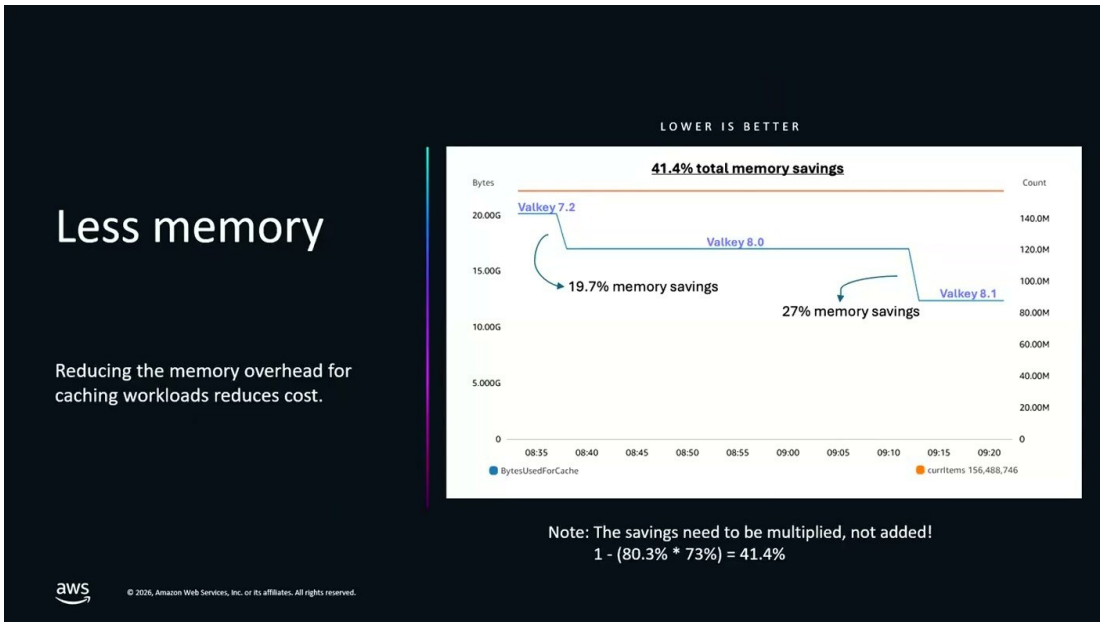
אחרי ה-CPU הגיע התור של הזיכרון, "כי לשם באמת הולך רוב הכסף שלכם" [9:17]. התקורה ב-Valkey 7.2: "בוולקי 72 ראינו שאנחנו משלמים 52 בתים עבור כל מפתח וואליו שאתם רוצים לשמור" [9:17] — ועוד 32 בתים אם משתמשים ב-TTL, ועוד 16 בתים בתצורת קלאסטר. הבדיקה על פני כלל לקוחות ElastiCache הראתה שה-P50 של מפתח וערך הוא באזור 100 בתים. כלומר תקורה בסדר גודל של חצי מהדאטה עצמו.

המבנה הישן: hash table של באקטים, ובכל באקט אובייקט entry שמחזיק שלושה מצביעים — למפתח, לאובייקט הערך, ולרשומה הבאה באותו באקט. כל חץ כזה הוא חיפוש כתובת זיכרון שעולה בערך 100 ננו-שניות, וכשפקודה שלמה נמדדת במיקרו-שניות, זה משמעותי גם בלייטנסי וגם ב-CPU. שלושת הצעדים שנעשו: הכנסת המפתח עצמו לתוך ה-entry, מיזוג המטא-דאטה של אובייקט הערך לתוך ה-entry והצבעה ישירה על הערך, וביטול מצביע ה-next על ידי ניצול העובדה שהמעבד ממילא מביא 64 בתים — שמונה כתובות — בכל שליפה, כך שהבאקט נקרא כשמונה מצביעים סמוכים.



פירוט התקורה שורה-שורה: 52–60 בתים ב-Valkey 7.2 מול 29–38 ב-Valkey 8.1

"העלמנו את המצביע של ה קי, את המצביע של הוולי ואת המצביע של הנקסט, כלומר את העלמנו 24 בתים והיינו צריכים להוסיף עוד איזשהו בית אחד רק בשביל המטאדאטה" [12:22]. התוצאה: "עכשיו זוכרים שהיה 52, אז יש פה חיסכון של 50% לכל מפתח וואליו שלכם" [12:22].

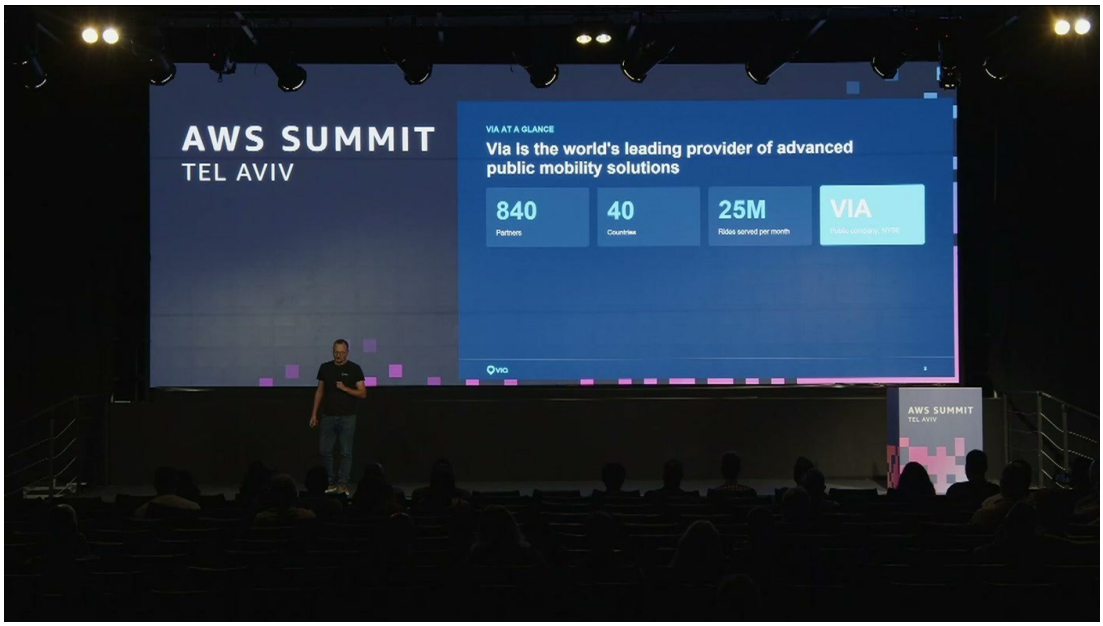


שחזור על נתוני לקוח: 19.7% במעבר ל-8.0 ועוד 27% ב-8.1 — מוכפלים ל-41.4%

הסלייד מדגיש נקודה שקל לפספס: החיסכונים מוכפלים ולא מחוברים — 1 פחות 80.3% כפול 73% שווה 41.4%. ובונוס: ביטול חיפושי הזיכרון חסך גם עבודת מעבד. במיקרו-בנצ'מרק על ה-hash table החדש, בתרחיש miss מלא, "ראינו ירידה של 70% מצריכת ה-CPU" [13:53]. לשני השיפורים — ריבוי התהליכונים והזיכרון — הוצג ברקוד לבלוגים הטכניים של Valkey עם המימוש והבנצ'מרקים.

5. Via: מה האלגוריתם באמת עושה מול הקאש

"אני שובל כהן, ראש צוות אלגו אינג'נירינג בוויה" [13:53]. Via מספקת פתרונות תחבורה ציבורית מתקדמים: "אנחנו עובדים יותר משמונה מאות לקוחות, בכ-40 מדינות ברחבי העולם, ומשרתים כ-25 מיליון נסיעות בחודש באמצעות המערכת שלנו" [13:53]. החברה נסחרת בבורסה בניו יורק.



המספרים שהוצגו: 840 שותפים, 40 מדינות, 25 מיליון נסיעות בחודש

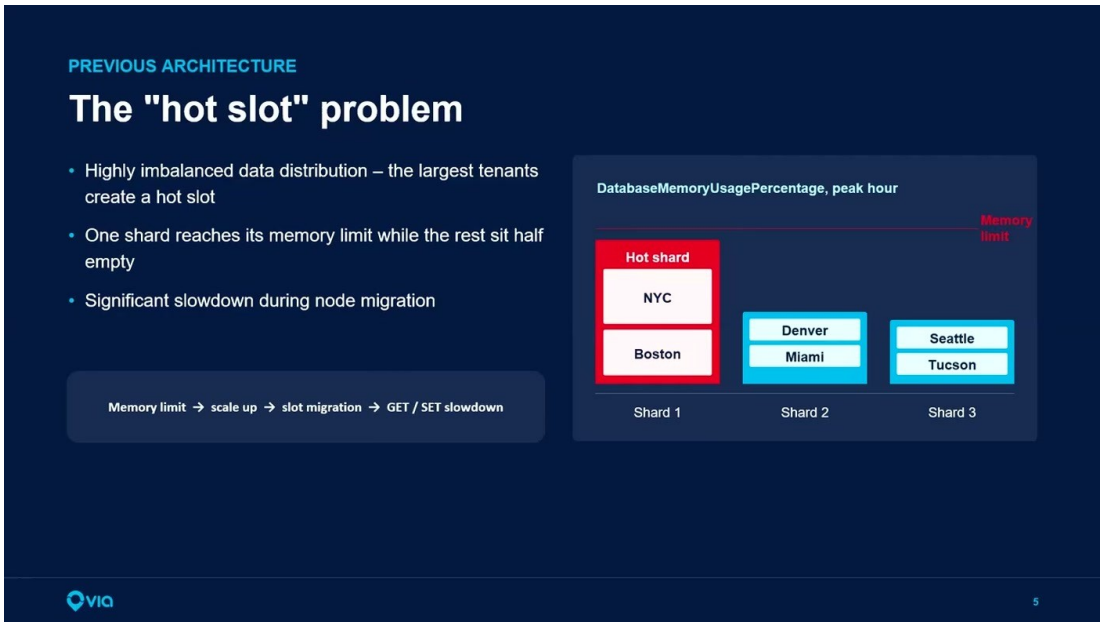
המוצר הוא תחבורה ציבורית לפי דרישה. נוסע מבקש נקודת איסוף, יעד וזמן, והמערכת משבצת אותו לתוך ואן שכבר בדרך עם מסלול, נוסעים והתחייבויות קיימות. כדי לדרג כל אפשרות שיבוץ צריך לדעת כמה זמן ייקח להגיע לאיסוף, להורדה, ואיך זה משנה את משך הנסיעה של כל מי שכבר ברכב. "אפשר להגיע ל-100,000 חישובים של משחי זמן נסיעה בבקשה אחת" [15:24] — וזה בלתי אפשרי בזמן שהנוסע ממתין.

הפתרון: חישוב מראש של מאות אלפי זמני נסיעה בעיר, באמצעות אלגוריתמי shortest path כמו Dijkstra על גרף הרחובות, ואחסון התוצאות ב-ElastiCache. "אנחנו מסתמכים בצורה מאוד משמעותית על אלסטיקש" [13:53]. בזמן אמת האלגוריתם שולף את מה שהוא צריך ומדרג.

6. בעיית ה-hot slot

הארכיטקטורה הקודמת: קלאסטר אחד לכל אזור גיאוגרפי, שמחזיק את כל הערים באותו אזור. אין ערבוב בין ערים — שליפה תמיד נוגעת בעיר אחת ויחידה. מכאן נולדה האופטימיזציה: אם כל המפתחות של עיר יושבים על shard אחד, אפשר לשלוף 100 אלף מפתחות ב-round trip רשת אחד. המימוש נעשה עם hash tags: "לכל קי יש פרפיקס של שם העיר שלו ודהיינו שכל קי עם אותו פרפיקס יגיע לאותו השלוט" [16:56].

והבעיה: "נוצר לנו מצב שהקלאסטר שלנו היה מאוד לא מאוזן" [18:31]. קיבוע עיר ל-slot מבטיח איפה הנתונים יישבו, אבל לא אומר דבר על גודלם. ניו יורק ובוסטון נחתו על אותו shard שהתקרב לתקרת הזיכרון שלו, בזמן שערים קטנות ישבו על shards חצי ריקים.



פיזור לא מאוזן בשעת שיא — shard אחד בגבול הזיכרון, השניים האחרים חצי ריקים

"המצב הזה תפרס אותנו כמובן באמצע הלילה שהגילינו שהשרד הזה מגיע למגבלת היכולת שלו ואנחנו חייבים להגדיל אותו" [18:31]. וכאן נסגר המעגל: הגדלת shard ב-ElastiCache מרימה מכונה חדשה ומעבירה את הדאטה slot אחרי slot. באותה תקופה Via רצה על Redis single-threaded, וכל slot הכיל ג'יגות. "חווינו סלו דאונד מאוד מאוד משמעותי במערכת בעקבות השינוי הזה. זו הייתה נקודת המפנה" [18:31]. השרשרת כפי שהופיעה על הסלייד: GET/SET slowdown ← slot migration ← scale up ← memory limit.

7. הארכיטקטורה החדשה והתוצאה

המטרה הוגדרה במדויק: להסיר את קיבוע העיר ל-slot, בלי לאבד את הביצועים שהאלגוריתם צריך. הצעד הראשון היה הורדת ה-hash tag, כך שמפתחות ניו יורק התפלגו באופן אחיד על כל השרדים. הצעד השני החליף את השליפה: "השתמשנו במה שנקרא non-atomic em-get ו-em-set בפייפליינים" [20:02]. הקליינט מחשב את פונקציית ה-hash על כל מפתח, מבין לאיזה shard הוא שייך, ומוציא בקשות multi-get לכל השרדים במקביל בפייפליין.

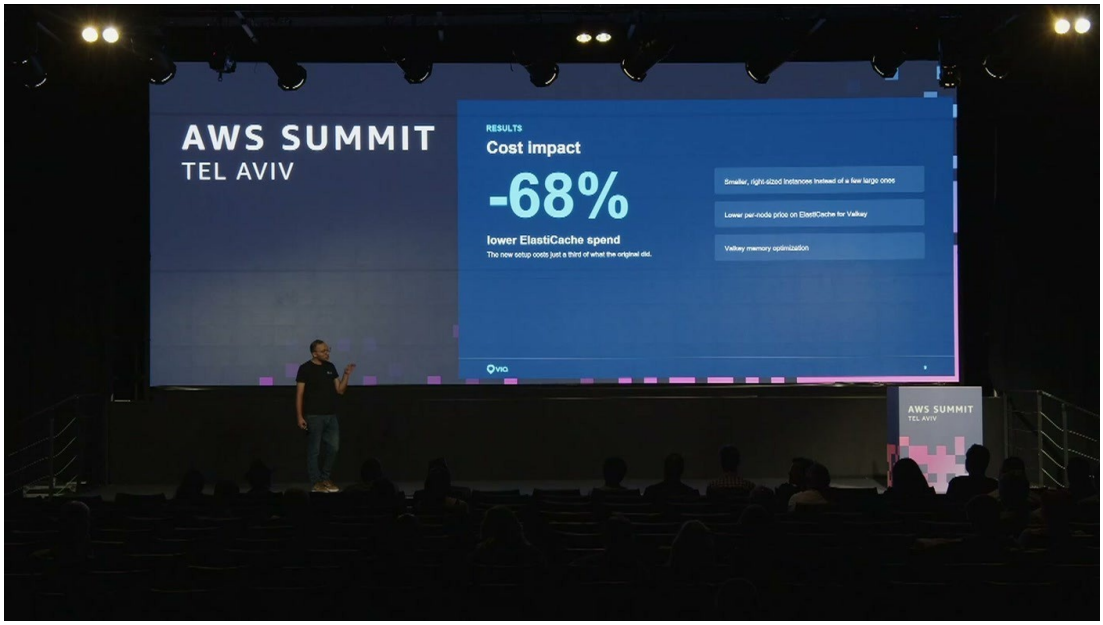
הפשרה המודעת: כל shard עשוי לענות בזמן אחר, ולכן אין אטומיות בזמן — ולעומס הזה זה התאים. מבחינת האלגוריתם דבר לא השתנה: "ביקשנו 100,000 keys וקיבלנו את אותם 100,000 [20:02] "values, והקליינט מיזג את התשובות מכל הפייפליינים.

- **פיזור אחיד:** אין יותר shard בודד שמגיע לתקרה לפני האחרים — כולם מתמלאים יחד.
- **שרדים קטנים יותר:** אין צורך להחזיק את כל ניו יורק על מכונה אחת, אלא רק חלק מהמידע.
- **מיגרציה בטוחה:** ה-slots חזרו להיות קטנים, ולכן העברתם ממשיכה לענות לבקשות GET/SET במקביל, "בצורה סימנס לחלוטין" [21:33].
- **אוטו-סקיילינג בפרודקשן:** "היום בפרודקציה אנחנו מעלים ומורידים שרדים בצורה אוטומטית" [21:33].

המחיר: בקשות כתיבה גדולות במיוחד נעשו איטיות יותר מול כמה שרדים. הפתרון היה להוציא את הכתיבה לתהליכון אסינכרוני ברקע — בזמן שהנוסע מקבל את התשובה, הכתיבה עדיין רצה. "אמנם, קיבלנו פה איטיות מסוימת בגלל המעבר לאכיטקטורה הזו, אבל זה לא משפיע על הפלו הקריטי של הזמנת נסיעה" [21:33].

ולמה בכלל לעבור ל-Valkey

כחלק מהחשיבה מחדש הוחלט גם להחליף מנוע. הסיבות: שיפורי ה-CPU והזיכרון שהוצגו בחלק הראשון, מחיר נוד נמוך ב-20% מ-Redis, וקלות ההחלפה — אותו API בדיוק. "הלכנו לטרפורם, שינינו מ-Redis ל-Walky, עשינו אפלי וזה פשוט עבר, זה לא השפעה על הלקוח" [23:05].



התוצאה הכלכלית כפי שהוצגה, עם שלושת מקורות החיסכון בצד

"אנחנו משלמים 68% פחות מהעלות שישלמנו בעבר בעקבות שינוי ארכיטקטורה הזאת המערכת שלנו היום עולה שליש" [23:05]. שלושת מקורות החיסכון: מעבר למכונות קטנות ומדויקות יותר שמתאימות לצריכה בפועל, מחיר נוד נמוך יותר ב-Valkey, ואופטימיזציית הזיכרון שמאפשרת לאחסן את אותה כמות מפתחות בפחות זיכרון.

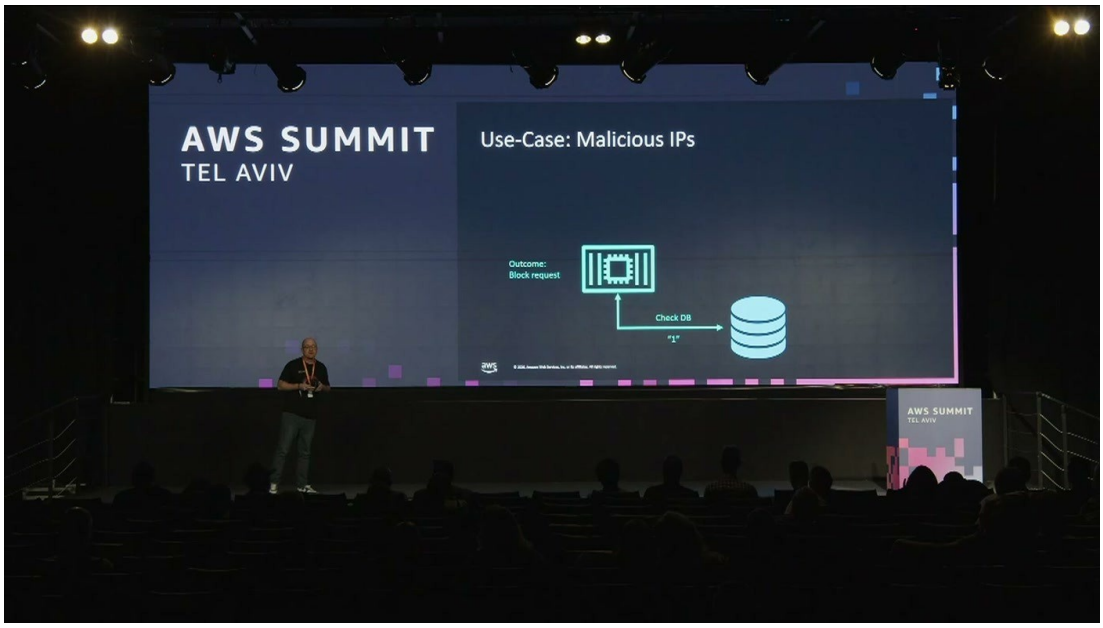
שלוש המסקנות שסיכמו את חלק הלקוח

- **להסתכל מעבר לממוצעי הקלאסטר:** "כשאנחנו ניתרנו את הממוצעים של הקלאסטר, הוא היה נראה מצוין" [23:05] — בזמן ש-shard בודד כבר נגע בתקרה.
- **להגן על הנתיב הקריטי:** היה כאן קנס ביצועים, אבל הוא הוזז אל מחוץ למסלול של הזמנת נסיעה.
- **כל אופטימיזציה היא trade-off:** "כל אופטימיזציה היא סוג של טרייד אוף של המערכת כולה" [24:37]. המסקנה כפי שנוסחה: "הסרנו אופטימיזציה שנתנה לנו מהירות מקומית ונינו מערכת שנותנת לנו גם מהירות וגם יכולת לגדול" [24:37].

8. Bloom Filter: לענות על שאלת קיום בלי לשמור את הדאטה

החלק השלישי פתח בבעיית unique user existence. הפתרון הרגיל הוא set שמחזיק את כל המשתמשים, ובודקים חברות בו. "זה פתרון מאוד נהדר, אבל הבעיה בו ככל שבאמת כמות היוזרים גדולה ובמאות מיליונים" [24:37] — הזיכרון והעלות גדלים בהתאם.

Bloom Filter, שיצא ב-Valkey לפני כשנה, הוא מבנה נתונים הסתברותי שלא שומר את האיברים עצמם. בכל בדיקה מופעלת סדרת פונקציות hash על המפתח. ההתנהגות אסימטרית: "בשימוש בבלום פילטר יכול להיות פוסט פוזיטיב" עם סיכוי נמוך באזור האחוז, אבל "לעומת זאת, בעיית false negative לא יכולה לקרות" [26:13]. תשובה שלילית היא ודאית. השימושים שהוזכרו: האם המשתמש כבר ראה מודעה, סינון ספאם, ובדיקת זמינות של שם משתמש בהרשמה.



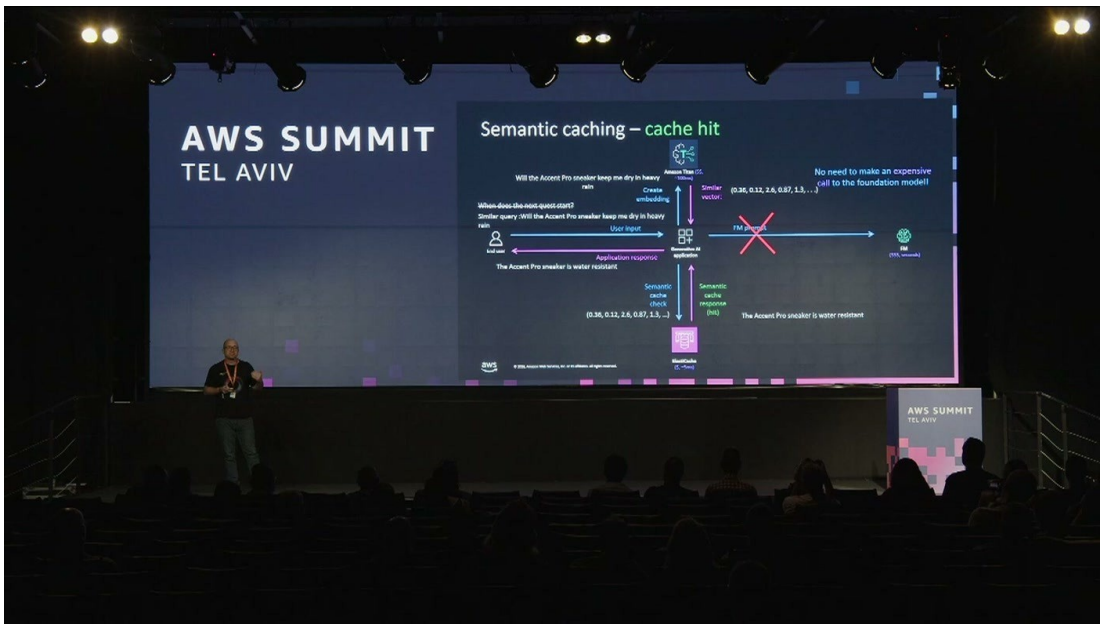
תרחיש ה-IP הזדוני: אימות מול המסד הרלציוני רק כשהמסנן מחזיר תשובה חיובית

הדוגמה הייתה לקוח סייבר שמנהל רשימת malicious IPs. במקור הרשימה ישבה במסד רלציוני; כשהטראפיק עלה נוספה שכבת קאש ב-Valkey שהחזיקה את כל הכתובות ב-set — וככל שהרשימה גדלה, גדלה איתה העלות. במעבר ל-Bloom Filter, תשובה שלילית מספיקה כדי להעביר את הבקשה הלאה; תשובה חיובית נשלחת לאימות מול המסד הרלציוני. מכיוון שמדובר באחוז זעום מהבקשות, הרוב המוחלט עדיין נענה מהקאש. "בסך הכל הוא הגיע ל-98% חיסכון בכמות הדאטה שהוא שמר בקש" [29:18].

9. Vector Search וקאשינג סמנטי

ל-AWS יש מגוון פתרונות vector search; מה שמייחד את ElastiCache הוא ש"ה-Vector Embeddings נשמרים in-memory" [29:18] — ומכאן הלייטנסי הנמוך והתפוקה הגבוהה, "וכל זה גם ב-Recall Rate מאוד גבוה של מעל 95%" [29:18]. ארבעה use cases הוצגו: קאשינג סמנטי, agentic memory, RAG, ומערכות המלצה שבהן הלייטנסי קריטי. עבור agentic memory הוזכרו אינטגרציות עם פריימוורקים — "כמו LandGraph, MAM Zero, Strand Agents" [30:49], כלומר LangGraph, Mem0 ו-Strands Agents.

הדוגמה לקאשינג סמנטי: אתר מסחר לציוד ספורט עם צ'טבוט. לקוחות שונים שואלים את אותה שאלה על נעל בניסוחים שונים — למשל האם דגם Accent Pro עמיד למים. "קש רגיל לא יעזור פה, נכון? כי ה-K'י עצמו שונה, כי הסטרינג הוא שונה" [30:49]. בחיפוש וקטורי ממירים את המחרוזת ל-embedding נומרי שמייצג את המשמעות, ומחרוזות בעלות משמעות דומה יקבלו וקטורים קרובים במרחב.



זרימת cache hit: שאלה מנוסחת אחרת מוצאת את התשובה הקיימת והקריאה למודל נחסכת

הזרימה: השאלה עוברת דרך מודל embedding — למשל Titan ב-Bedrock, או self-hosted ב-EC2 — ואז מול הקאש. ב-miss הראשון הבקשה עוברת ל-LLM, התשובה חוזרת למשתמש ונשמרת בקאש. בשאלה השנייה, השונה בניסוח, החיפוש הווקטורי מחזיר את התשובה הקודמת, והקריאה היקרה ל-LLM נחסכת. הנקודה שהודגשה: מודל ה-embedding עולה ומשהה בסדרי גודל פחות מקריאה ל-LLM.

מבחינת API — FT.CREATE יוצר אינדקס מעל מבנה נתונים קיים (hash או JSON), ומגדיר את אלגוריתם החיפוש, את מימד הווקטור ואת פונקציית המרחק. FT.SEARCH מחזיר את ה-k nearest neighbors. הפיצ'ר יצא ב-ElastiCache for Valkey 8.2, ובגרסה 9 שיצאה בערך במאי נוספו שני פיצ'רים מרכזיים.

10. Full-text search ואגרגציות

ההרחבה הראשונה הביאה חיפוש טקסט חופשי, prefix, suffix, טיפול בשגיאות כתיב, חיפוש נומרי לפי שדות כמו מחיר או תאריך, וחיפוש מדויק לפי tag או קטגוריה. את כל אלה אפשר לשלב עם חיפוש וקטורי בשאילתה אחת וב-round trip יחיד למנוע. ההתנהגות בזמן אמת הודגשה: "ברגע שאנחנו מכניסים אייטמים להשמאפ, האינדקס מתעדכם בצורה סינכרונית, וככה אין בעיות של [35:27] "read after right consistency.

הסכמה שהוצגה כבר לא מכילה רק וקטור: title כטקסט חופשי, brand וקטגוריה כ-tag, ו-price כשדה נומרי. שתי הדוגמאות שהוצגו — חיפוש טקסטואלי של "wireless headphones" תחת קטגוריית אלקטרוניקה, ושאילתת המלצה שמחפשת מוצרים דומים לווקטור קלט ומצמצמת אותם ל-price range.

שלוש הטענות על אגרגציות: פייפליינים מורכבים, תצוגות חיות, וניתוח במקום שבו הדאטה יושב

ההרחבה השנייה היא אגרגציות. עד כה, כדי לנתח דאטה שיושב בזיכרון היה צריך למשוך אותו לשכבת האפליקציה, או לבנות תהליך CDC לעבר מסד אנליטי אחר. עכשיו — לדשבורד בזמן אמת או ל-leaderboard — החישוב נעשה בתוך Valkey. הדוגמה שהוצגה: FT.AGGREGATE על אינדקס המוצרים, GROUP BY קטגוריה, חישוב average price ו-COUNT לכל קטגוריה, ו-SORT BY לפי המחיר הממוצע.

11. ElastiCache Durable

"זה למעשה הפיצ'ר המרכזי האחרון שיצא ממש לפני כמה שבועות" [37:03]. הבעיה שהוא פותר: לקוחות אוהבים את הלייטנסי התת-מילישנייתי, אבל חוששים מאובדן דאטה — כתיבה שנקלטה ב-primary ועדיין לא הגיעה ל-read replica עלולה להיעלם בנפילה.

הארכיטקטורה: קלאסטר רגיל של primary ושתי read replicas בשלושה availability zones, ומעליו שכבה מנוהלת על ידי AWS בשם Multi-AZ Transaction Log. הקריאות נשארות כמו בקלאסטר רגיל — strong consistent מה-primary, eventual consistent מהרפליקות.

שני מודים

- **סינכרוני — zero data loss:** כל כתיבה ל-primary עוברת ל-Transaction Log, והלקוח מקבל acknowledge רק אחרי ה-acknowledge משם. "הוא כבר לא Sub Milliseconds כמו הריב, הוא Single Digit Millisecond, אבל הוא באמת נותן Zero Data Loss" [38:33].
- **אסינכרוני — RPO של 10 שניות:** ה-primary לא ממתין; הכתיבה נכנסת ל-durability buffer ומתעדכנת ב-log ברקע. "הדאטה האחרון שנכתב לא ישן מ-10 שניות... לכן מה שנגיע, נגיע ל-RPO של 10 שניות" [40:07].

12. מה לוקחים מכאן

- **שדרוג מנוע הוא הרווח הזול ביותר.** המעבר מ-Redis OSS ל-Valkey הוא drop-in — אותו API, שינוי הגדרה אחד — ומביא איתו את שיפורי ה-CPU והזיכרון בלי שינוי קוד. המספרים שסוכמו בסוף: "חיסכון של 20% בנוטבייס קלאסטר מול Redis Open Source" [40:07] "ובסרברלס קלאסטר של 33%" [41:42].
- **לבדוק את ה-shard, לא את הממוצע.** זו המסקנה המעשית הישירה ביותר מהמקרה של Via. ממוצע קלאסטר בריא יכול להסתיר shard שכבר נוגע בתקרה.
- **hash tags הם חרב פיפיות.** הם מבטיחים round trip אחד, אבל גם מבטיחים פיזור לא מאוזן כשגודל הישויות שונה. שווה למפות איפה אצלנו יש קיבוע כזה.
- **קאשינג סמנטי הוא נקודת כניסה טבעית ל-GenAI.** אם כבר יש צ'טבוט שקורא ל-LLM על שאלות חוזרות בניסוחים שונים, שכבת vector cache חוסכת גם לייטנסי וגם עלות קריאה.
- **Durable פותח שימושים שנפסלו קודם.** כל מקום שבו נפסל Valkey בגלל חשש מאובדן כתיבה — שווה בדיקה מחדש, עם הבנה שהמחיר הוא מעבר מתת-מילישנייה למילישניות חד-ספרתיות.
- **הפרויקט פתוח להשפעה.** בסיום הוזכר שה-roadmap של Valkey פומבי, שאפשר להצביע על פיצ'רים ולתרום קוד.

מילון מונחים

מונח	מה זה, כפי שהוצג בסשן
Valkey	פורק קוד פתוח של Redis 7.2 שנוצר במרץ 2024 בעקבות שינוי הרישיון. תחת BSD License ו-Linux Foundation.
IO Threads	תהליכונים ייעודיים שמטפלים בקריאה מהרשת, ב-parsing ובכתיבה חזרה, בעוד הפקודות ממשיכות לרוץ בתהליכון הראשי.
Hash tag	תחילית שקובעת לאיזה slot המפתח ימופה — כל המפתחות עם אותה תחילית יישבו על אותו shard.
non-atomic MGET/MSET	שליפה או כתיבה של מפתחות מפוזרים על פני כמה שרדים בפייפליין מקבילי, ללא הבטחת אטומיות בזמן.
Bloom Filter	מבנה הסתברותי לבדיקת קיום. תשובה שלילית ודאית; חיובית עשויה להיות false positive בהסתברות נמוכה.
Semantic caching	קאשינג לפי משמעות ולא לפי מחרוזת: השאלה מומרת ל-embedding, ותשובה קיימת עם וקטור קרוב מוחזרת במקום קריאה ל-LLM.
FT.CREATE / FT.SEARCH / FT.AGGREGATE	יצירת אינדקס, חיפוש (כולל k nearest neighbors) ואגרגציה מעל hash או JSON.