מדריך מעשי לדאטה עבור Agentic AI

AWS Summit Tel Aviv 2026 ,סיכום סשן — TLV-DAT307

Reut Almog Talmi, Sr. Database Specialist SA, AWS · Yossi Lagstein, Sr. Solutions Architect, AWS · Rotem Gani, Software Engineer, Harmony

10 בספטמבר 2026 | משך: כ-35 דקות | הופק מהקלטת הסשן

מסלול: Databases on AWS · רמה: 300

על המסמך הזה

המסמך מסכם את הסשן TLV-DAT307 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 בשלושים וחמש הדקות. חותמות הזמן בגוף המסמך מפנות לנקודה בהקלטה שבה נאמרו הדברים.

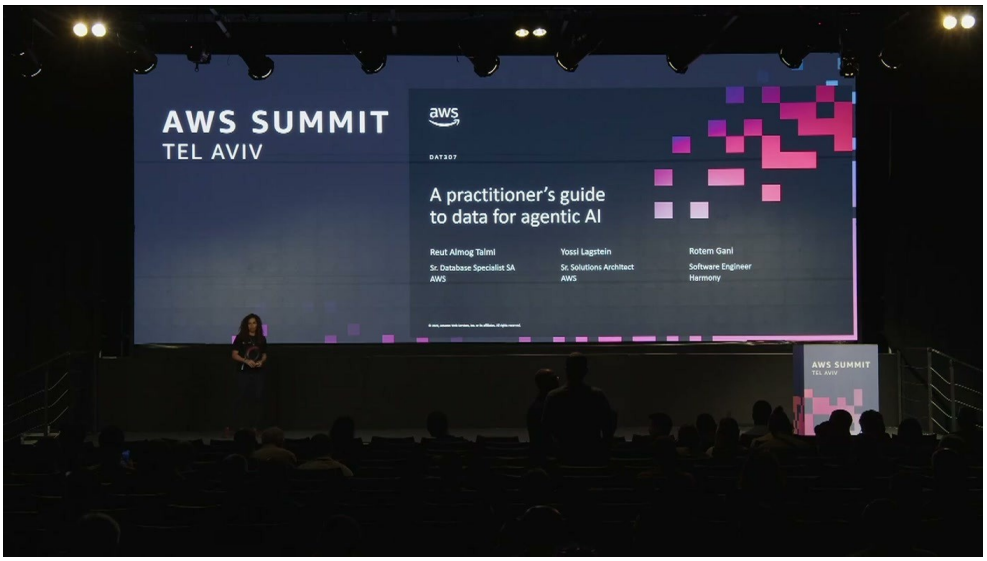
הסשן בנוי משלושה חלקים ברורים: תיאוריה (עקרונות היסוד של agentic AI וחיבורו לדאטה), מימוש על AWS (איך זה נראה עבור משתמש קצה, צרכן דאטה ומפתח), וסיפור לקוח (חברת Harmony, פלטפורמת ITSM מבוססת סוכנים).

איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. מה נאמר, ומה שווה לקחת הלאה.
- **פרקים 2–7 — התיאוריה:** ציר הזמן האג'נטי, לולאת ReAct, שכבות הזיכרון, MCP ובחירת כלים. זה החלק שאפשר להשתמש בו כחומר הדרכה פנימי.
- **פרקים 8–10 — המימוש על AWS:** שלוש חוויות משתמש — לקוח קצה, אנליסט דאטה ומהנדס דאטה — והשירותים שמאחוריהן.
- **פרקים 11–14 — סיפור הלקוח:** שלוש ארכיטקטורות אמיתיות מ-Harmony, כולל ההחלטה לעבור מ-tools מקודדים ל-MCP over APIs.
- **פרק 15 + מילון:** מה לוקחים מכאן, ומונחים שהופיעו בסשן.

הערת מקור המסמך הופק מתמלול אוטומטי של ההקלטה. התמלול מדויק בקטעים המקצועיים, אך מונחים לועזיים ושמות מוצרים הופיעו בו בשיבוש פונטי (למשל "אג'נט קור" = AgentCore, "לג פורמיישן" = Lake Formation) ותוקנו ידנית מול הסליידים. הציטוטים אומתו מול התמלול בחותמת הזמן המצוינת ומובאים בעברית כפי שנאמרו. מספרים ושמות שירותים נלקחו מהסליידים או מדברי הדוברים בלבד.

1. תקציר מנהלים

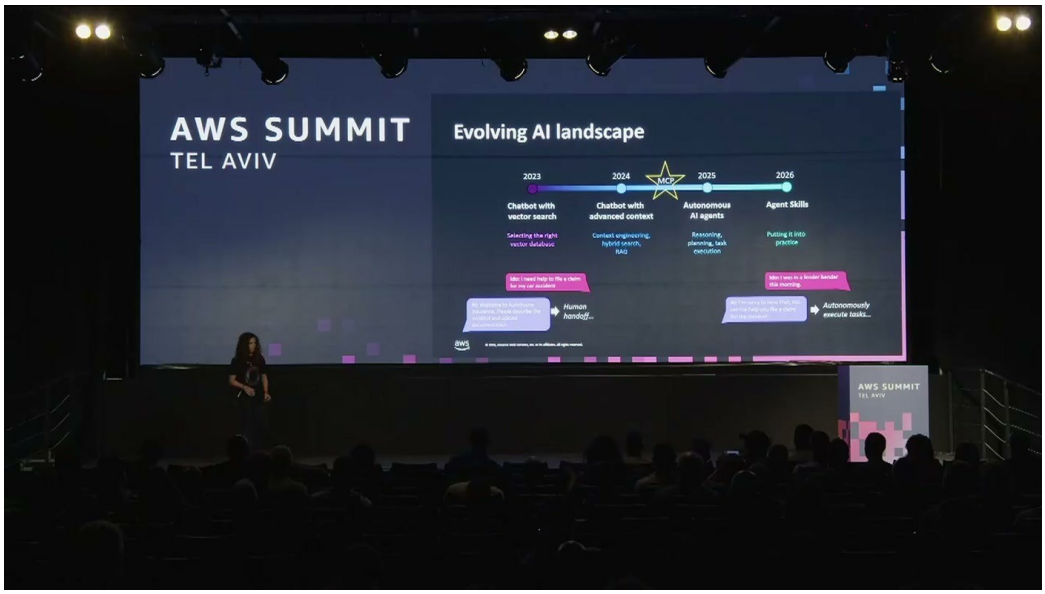


שלושת הדוברים: שתי הצגות מ-AWS ומקרה לקוח מ-Harmony, תחת מסלול הדאטאבייסים.

הסשן פותח באמירה שהיא גם התזה שלו: כולם מדברים על סוכנים, אבל סוכן בלי דאטה הוא צ'טבוט. המטרה המוצהרת הייתה לתת לאנשי דאטה — לא לאנשי ML — את הכלים לחבר את הדאטה הארגוני לסוכנים.

- **הדאטה הוא הצוואר, לא המודל.** המשפט הראשון בסשן: "מדברים היום הרבה בעולם על אייג'נטס אבל אין אייג'נטס בלי דאטה" [0:00]. כל שאר הסשן הוא פירוט של האמירה הזו לרמת שירותים וארכיטקטורות.
- **העיקרון היחיד שצריך להבין הוא לולאת ReAct.** Reason → Act → Remember, בלולאה. הדוגמה שרצה לאורך כל החלק הראשון — הצעת מחיר לביטוח רכב — הראתה שש איטרציות שבהן הקונטקסט מצטבר מתוצאות של tools שונים [6:13].
- **MCP הוא מה שהופך את זה לכלכלי.** "היופי עם MCP הוא שאנחנו לא צריכים לכתוב קוד בנפרד עבור כל סוג של מקור מידע" [9:19]. אותו עיקרון עובד מול key-value, vector ו-relational.
- **המימוש על AWS לא מחליף את הסטאק הקיים — הוא עוטף אותו.** אותם APIs, אותם data stores, אותן תשתיות; MCP רק מנגיש אותם לסוכן [14:01]. זו נקודה חשובה לארגון עם מערכות ליבה קיימות.
- **הממשל מגיע משכבות קיימות, לא מהסוכן.** הרשאות נאכפות דרך Lake Formation ו-SageMaker Unified Studio, והסוכן רץ תחת אותו role של המשתמש שפנה אליו [15:35].
- **מספר אחד ממקרה הלקוח: 60%.** Harmony דיווחה על "כ-60% ירידה בפתיחת הטיקטים" אצל לקוחות בפרודקשן [23:26] — זה המספר היחיד שהוצג בסשן, והוא מדד לירידה בפניות ולא לאחוז פתרון אוטומטי.

2. מאיפה הגענו: ארבע שנים בשקף אחד



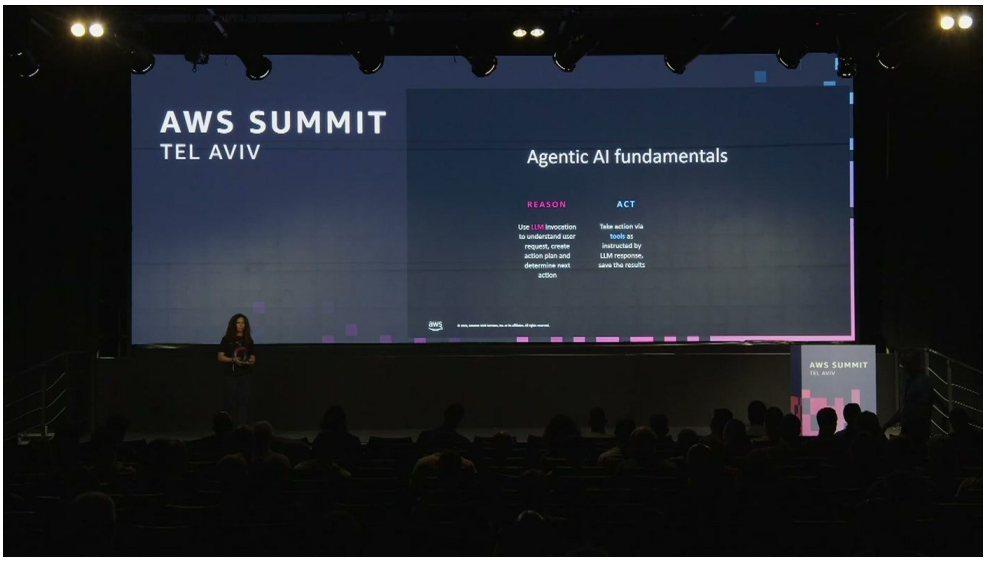
ציר הזמן שהוצג: 2023 צ'טבוט עם vector search, 2024 צ'טבוט עם קונטקסט מתקדם, 2025 סוכנים אוטונומיים, 2026 Agent Skills. הכוכב באמצע מסמן את הופעת MCP.

הדוברת פתחה בציר זמן קצר שמסביר למה השאלה "איך מחברים דאטה לסוכן" נשאלת דווקא עכשיו. ב-2023 דובר על אינטראקציות חד-פעמיות עם LLM ועל בחירת vector database. ב-2024 נוסף קונטקסט — "היכולת של אג'נט לזכור מה הוא עושה" — ואיתו hybrid search ו-RAG. לקראת סוף 2024 הופיע MCP, "שיצר סטנדרטיזציה לשימוש בכלים לצריכת דאטה ממקורות מידע שונים" [1:35].

ב-2025 הגיעו סוכנים אוטונומיים שיכולים "להסיק, ליצור, לתכנן ולבצע רצף של פעולות מקצה לקצה ללא התערבות אנושית", וב-2026, לפי הדוברת, MCP הוא כבר מיינסטרים ומה שחדש הוא Skills — "הוראות מובנות שאייג'נט יכול להפעיל שוב ושוב" [1:35].

הדוגמה הרצה שנבחרה לכל החלק התיאורטי: חברת הביטוח AutoInsure ולקוח בשם עידו שמבקש הצעת מחיר או מגיש תביעה. ב-2023 השיחה הייתה מסתיימת בהעברה לנציג אנושי; ב-2026 הסוכן מנהל את התהליך מקצה לקצה.

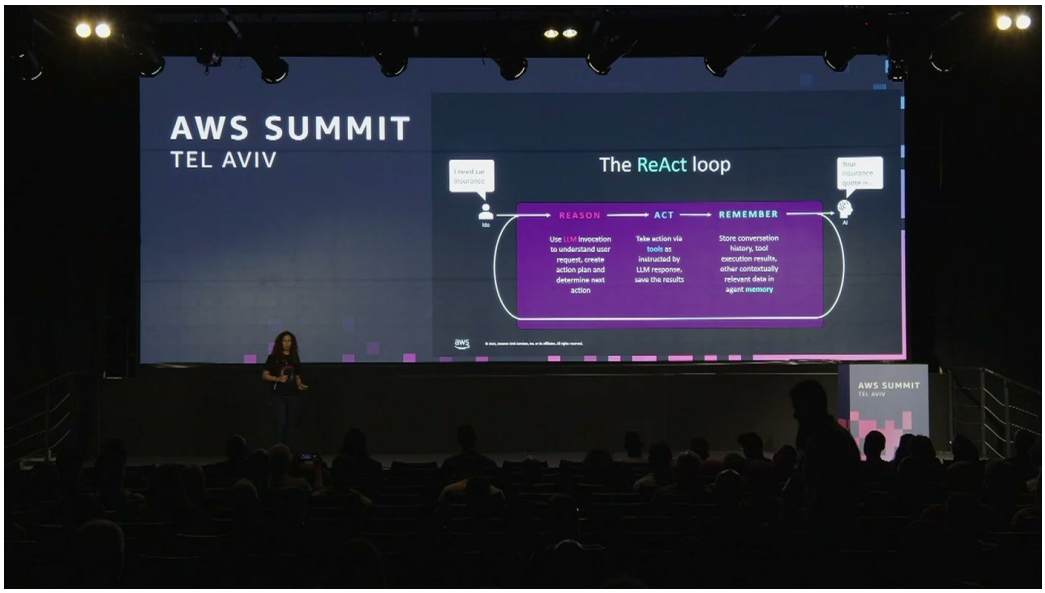
3. שלושת עמודי התווך ולולאת ReAct



שני העמודים הראשונים לפני שהשלישי נחשף: REASON — הבנת הבקשה ובניית תוכנית; ACT — הפעלת tools לפי הוראות המודל.

שלושת עמודי התווך שהוצגו הם Reasoning (ה-LLM מבין את הבקשה, בונה תוכנית עבודה ומגדיר את הצעדים הבאים), Action (הפעלת tools וקריאות API) וזיכרון. לגבי הזיכרון נאמר משפט שנתפס באולם:

> **Database Specialist SA, AWS [3:08] —** האג'נט חייב לזכור מה הוא ביצע. מה הכלים שעומדים לרשותו, מה הנחיות שלו. בלי זה תדמיינו שיחה עם דג זהב, מאוד מתסכל ולא מועיל.

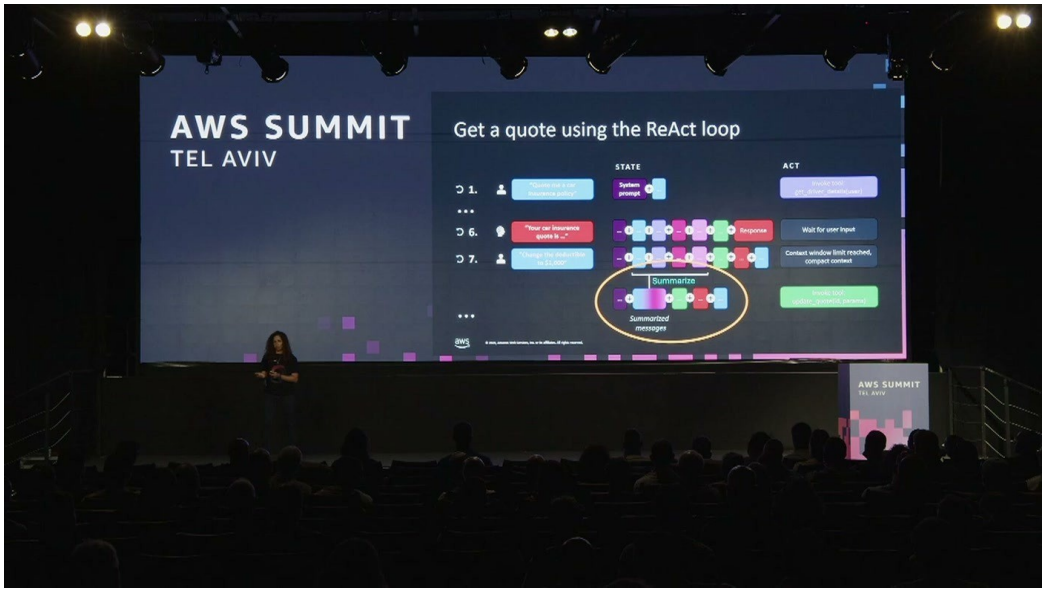


הלולאה המלאה: REASON → ACT → REMEMBER, כשהפלט של אחת מוזן בחזרה לקלט של הבאה. השרטוט מציין במפורש מה נשמר בזיכרון — היסטוריית שיחה, תוצאות הרצת כלים ומידע הקשרי.

הנקודה המרכזית: לא שלושת החלקים הופכים מערכת ל-agentic, אלא הכנסתם ללולאה. "מה שהופך את כל זה לאג'נטיק AI, הוא שאנחנו שמים את שלושת החלקים האלו בלולאה, מה שמאפשר לנו לחזור על התהליך" [3:08].

בתוך איטרציה בודדת: הבקשה של המשתמש, ה-system prompt ומידע מהזיכרון מרכיבים יחד את הקונטקסט; הקונטקסט עובר דרך ה-LLM; מה שיוצא זו תוכנית פעולה. בשלב ה-Act המודל מחליט אם לסיים ולהשיב למשתמש או להפעיל tools, והמידע שחוזר מהם מוזן ל-context manager שעושה caching, שומר את היסטוריית השיחה ומעריך את התוכנית להמשך [4:40].

4. שש איטרציות עד הצעת מחיר — ומה קורה כשהקונטקסט נגמר



כל שורה היא איטרציה. בעמודת STATE רואים איך המידע מצטבר בצבעים שונים לפי הכלי שהחזיר אותו; בעיגול המסומן — דחיסת ההיסטוריה ל-summarized messages כשחלון הקונטקסט מתמלא.

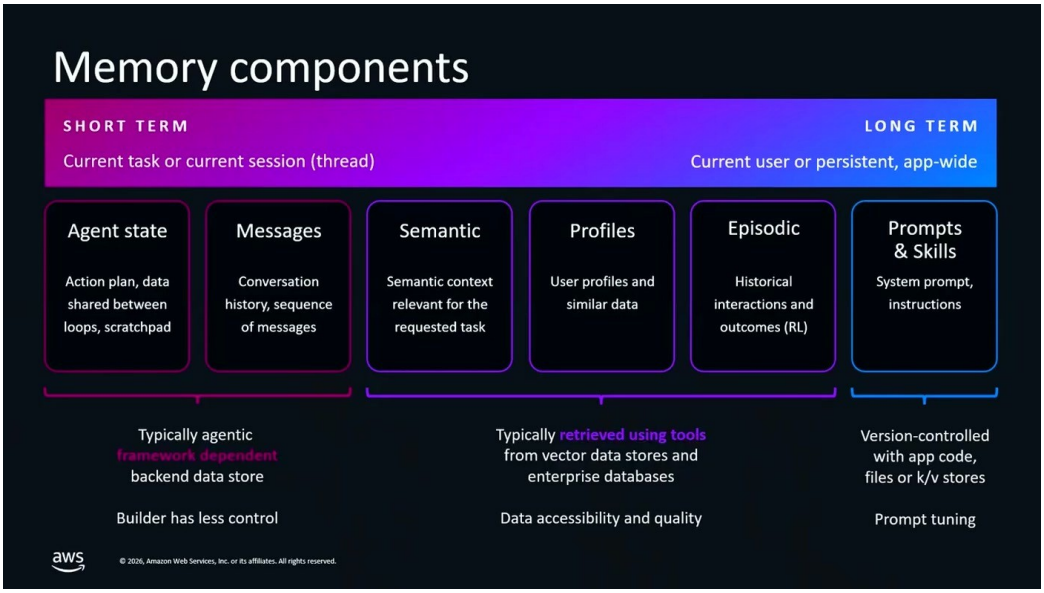
ההדגמה פירקה את הבקשה "אני צריך ביטוח רכב" לרצף קריאות: get_driver_details מחזיר מידע על הנהג; כלי נוסף מחזיר מידע על הרכב; כלי שלישי מחשב את מידת הסיכון; ורק אז נוצרת הצעת מחיר. בכל איטרציה המידע שחזר מצטרף ל-state ההתחלתי [4:40]–[6:13].

המסקנה שהוצגה: "אנחנו רואים שהאג'נט עבר דרך הלופ שש פעמים, ויש לו מספיק מידע כדי לספק לעידו תשובה עם הצעת מחיר" [6:13].

החלק המעניין יותר הוא מה שקורה אחר כך. כשהמשתמש מבקש שינוי בהצעה — בסלייד: "Change the deductible to $1,000" — הלולאה ממשיכה, אבל חלון הזיכרון מוגבל:

> **Database Specialist SA, AWS — [6:13]** הקונטקסט הוא לא אינסופי, חלון הזיכרון מוגבל, אז אנחנו מסכמים ודוחסים את המידע שהצטבר עד לשלב הזה. יש לזה טרייד אוף, אבל בגדול עם הטכניקות המתאימות הלופ יכול להמשיך ולתפקד היטב.

5. ארבע שכבות זיכרון — ומי מחזיק כל אחת



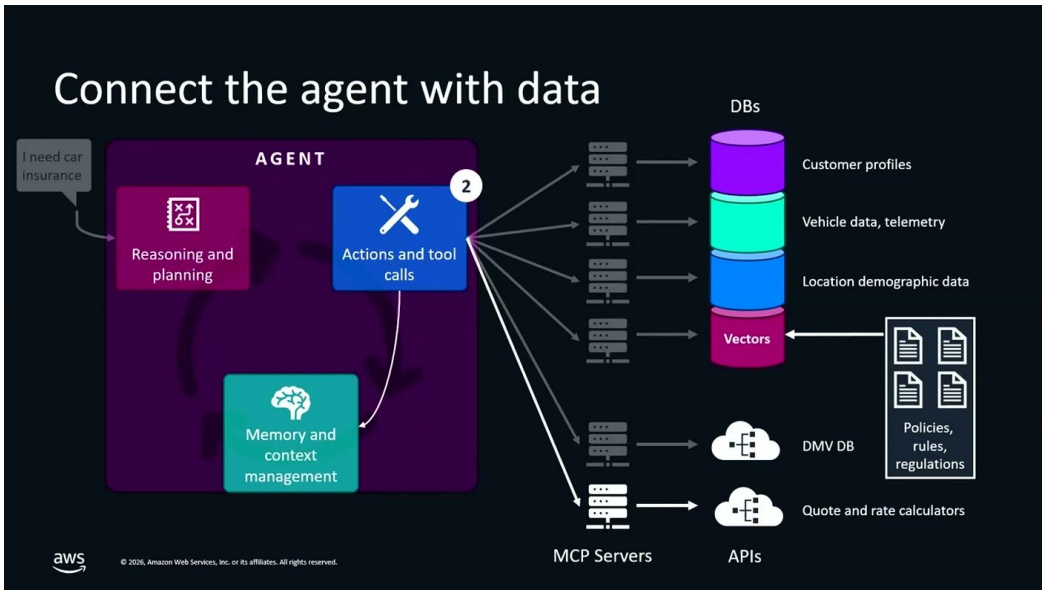
ציר קצר-טווח מול ארוך-טווח: Agent state ו-Messages בצד אחד, Semantic / Profiles / Episodic באמצע, ו-Prompts & Skills בקצה — כולל הערה מי "הבעלים" של כל שכבה ומהי מידת השליטה של המפתח.

הסלייד הזה הוא כנראה החומר הכי שמיש בסשן לצוות שבונה סוכן ראשון. הוא ממפה את הזיכרון לשלוש רמות:

- **טווח קצר — agent state:** זיכרון העבודה של הסוכן, שבו נשמר מה שחזר מהפעלת tools בין הלולאות (בדוגמה: מספר הרכב של עידו). ההערה המעשית: "לא צריך לבנות את זה לבד. agents framework כמו Strands או AgentCore Memory נותנים לכם את זה out of the box." [7:47]
- **טווח בינוני — זיכרון סמנטי:** ה-knowledge base של הסוכן — מסמכים, פוליסות ביטוח, נהלים, היסטוריית לקוחות. יושב ב-vector database: "זה יכול להיות OpenSearch או Aurora Postgres עם pgvector", והסוכן ניגש אליו דרך tools לקריאה ולכתיבה [7:47].
- **טווח ארוך — prompts ו-skills:** system prompts שהמפתחים מזינים, ו-skills — "הוראות מובנות שהאג'נט יכול לשלוף ולהפעיל שוב ושוב בלי שנצטרך להסביר לו בכל פעם מחדש". הדימוי שהוצע: "מתכונים לשימוש חוזר" [7:47].

ההבחנה התפעולית שנלוותה: שתי השכבות האחרונות מתנהגות כמו קוד — נשמרות ב-repository או ב-key-value store, לא ניגשים אליהן בתדירות גבוהה, ואינן מחזיקות מידע על משתמש ספציפי [7:47]. זו הבחנה שמשליכה ישירות על שאלות של גרסאות, ביקורת ורגולציה.

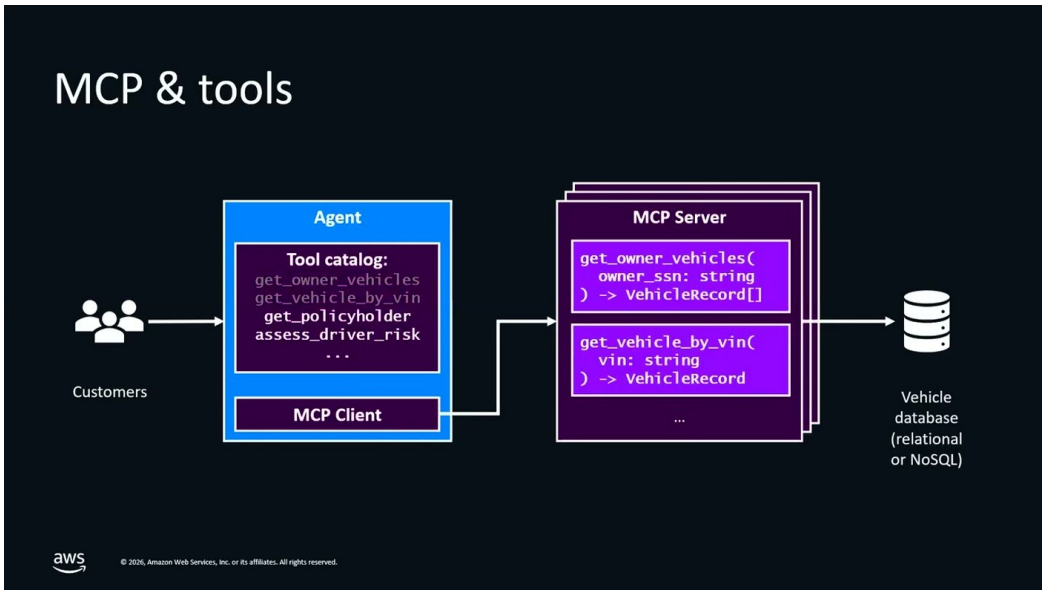
6. איך מחברים סוכן לדאטה: MCP מקרוב



מפת החיבורים: רכיב ה-Actions and tool calls פונה דרך MCP servers ו-APIs למגוון מקורות — פרופילי לקוחות, טלמטריית רכב, נתוני דמוגרפיה, vectors, מאגר רישוי, ומחשבוני תעריפים.

הזרימה שהוצגה: בקשת המשתמש מגיעה לסוכן, רכיב ה-actions פונה ל-MCP server, זה פונה לדאטאבייס ומחזיר — למשל — את פרטי הנהג; התוצאה נשמרת בזיכרון כקונטקסט ללולאה הבאה; בלולאה הבאה פונים ל-MCP server אחר ולדאטאבייס אחר [9:19]. סוגי המקורות שהוזכרו במפורש: key-value, vector, ורלציוני.

Database Specialist SA, AWS [9:19] — היופי עם MCP הוא שאנחנו לא צריכים לכתוב קוד בנפרד עבור כל סוג של מקור מידע. העיקרון זהה עבור כל דאטאבייס.



זום פנימה: MCP Client בתוך הסוכן מחזיק tool catalog; ה-MCP Server מצהיר על tools עם חתימות מלאות (get_owner_vehicles, get_vehicle_by_vin, assess_driver_risk) ומדבר עם מסד הנתונים דרך driver שיושב בתוכו.

ההסבר על המנגנון: MCP — Model Context Protocol — יוצר סטנדרט אחיד לתקשורת בין הסוכן לכלים, ומאפשר לו שני דברים: לגלות אילו כלים זמינים, ולהפעיל אותם. בתוך הסוכן יושב MCP client עם tool catalog; הוא מבצע פעולת list tools, והשרת מציג את מה שזמין. לכל tool יש פרמטרים ותיאור שמכתיבים איזה API להפעיל ולאיזה דאטאבייס לפנות, והקריאה בפועל מתבצעת דרך database driver שיושב בתוך ה-MCP server [9:19]–[10:53].

Database Specialist SA, AWS [10:53] — אנחנו לא אומרים לאג'נט תפעיל Tool X — הוא מגלה בעצמו מה זמין לו ומחליט מה להפעיל.

7. איזה כלי בכלל לבנות

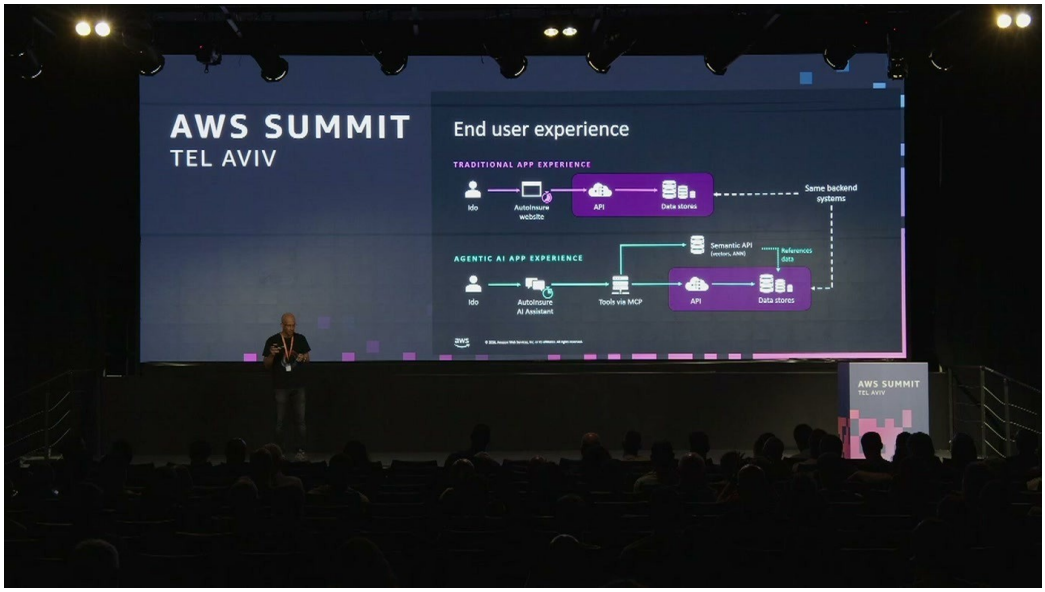
החלק הקצר שסגר את התיאוריה עסק בשאלה שרוב הצוותים נתקלים בה מיד אחרי ה-POC: לא איך לבנות tool, אלא אילו tools להגדיר. הוצעו שני צירים [10:53]:

- **מה הכלי עושה —** קורא דאטה או משנה דאטה (data retrieval מול data mutation).
- **מי משתמש בו —** משתמש פנימי או משתמש חיצוני.

הדוגמאות שניתנו: data analysts פנימיים שקוראים דאטה, developers פנימיים שמשנים דאטה, ולקוחות חיצוניים שגם הם עושים את אחד משני אלה [10:53]. הסלייד עצמו חילק את המטריצה גם לציר general purpose מול specialized. החלוקה הזו היא בפועל גם חלוקת סיכון: כלי mutation שנחשף למשתמש חיצוני הוא תא אחר לגמרי מבחינת ממשל מאשר כלי retrieval פנימי — נקודה שתחזור בחלק של Harmony תחת השם Guarded Operations.

8. מהתיאוריה למימוש: אותו backend, ממשק אחר

בנקודה הזו עלה ה-Solutions Architect [12:25] ולקח את אותה חברת ביטוח לשלוש פרסונות: עידו הלקוח, תמר שעושה אנליזות בתוך החברה, ומיכל שבונה את התשתית שהשתיים משתמשות בה.



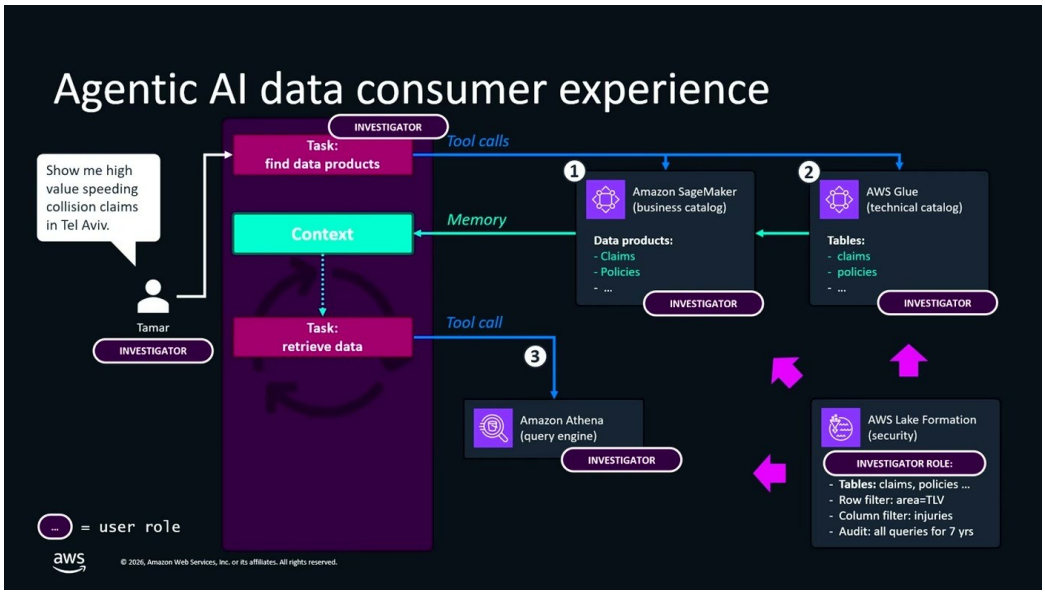
שתי שכבות באותו שקף: למעלה המסלול המסורתי — אתר, API, data stores; למטה אותו מסלול בדיוק, אלא שהקלט הוא טקסט חופשי וה-tools-נחשפים דרך MCP, עם semantic API נפרד לווקטורים.

הטענה המרכזית של החלק הזה היא שהמעבר לעולם האג'נטי אינו פרויקט החלפה. המשתמש מקבל מסך שבו הוא כותב טקסט חופשי; הטקסט מתורגם לקריאת API באמצעות tool ו-MCP; ומאחור — "זה אותו דאטאבייס, אותם API שגם היה קודם לכן, רק עשינו את זה התאמה לעולם האג'נטי באמצעות [12:25] "MCP. הכוכבית היחידה: מוסיפים vector database שמחזיק את הקונטקסט של הדומיין — במקרה הזה דומיין הביטוח [14:01].

גם בצד הפנימי התמונה זהה. במקום לכתוב "סיקוולים מורכבים", המשתמש כותב בשפה חופשית, וזו מתורגמת לקריאות SQL שמונגשות דרך MCP:

Solutions Architect, AWS [14:01] — שוב, אותם דאטה סטורס, אותם API, אותם תשתיות, רק הדרך השימוש בהם יותר קלה ודרך הבנייה יותר מהירה.

9. צרכן ויצרן דאטה — ואיפה נאכפות ההרשאות

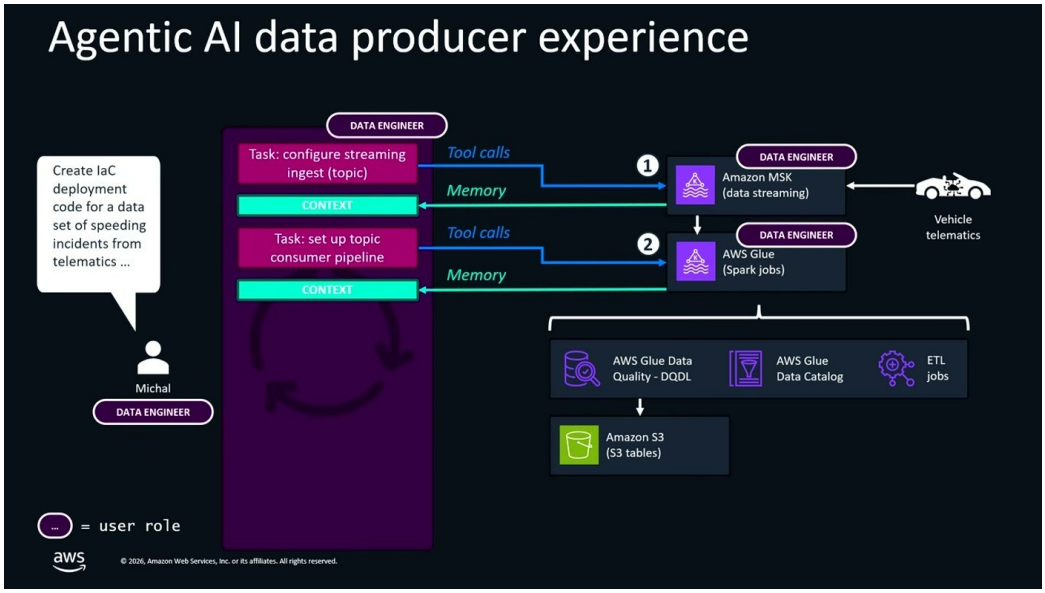


מסלול הצרכן: תמר בתפקיד Investigator שואלת על תביעות התנגשות במהירות גבוהה בתל אביב; הסוכן מגלה מוצרי דאטה ב-SageMaker Unified Studio, מקבל את מגבלות Lake Formation (סינון שורות לתל אביב, מיסוך עמודות, ביקורת) ומריץ את השאילתה ב-Amazon Athena.

הדוגמה של תמר היא הדוגמה החשובה ביותר בסשן מבחינת ממשל. היא חוקרת, יש לה role של Investigator, ונקבע במפורש: "האג'נט שמוקם מולה פועל תחת אותו רול של אותם הרשאות, כמו לתמר עצמה" [14:01].

הזרימה: הסוכן צריך קודם להבין מהו ה-data domain שלו, ולשם כך פונה ל-SageMaker Unified Studio — "הסביבה שמתכללת את השירותים האנליטיים של AWS" — שם יושב Business Catalog שבו מזוהים מקורות המידע הרלוונטיים (תביעות ופוליסות). מתחתיו Technical Catalog, שברוב המקרים הוא AWS Glue. שכבת ההגבלות עצמה מוגדרת ב-Lake Formation — בדוגמה, לתמר מותר לראות רק אירועים בתל אביב. הקונטקסט המלא חוזר לסוכן, שמתרגם טקסט חופשי ל-SQL ומריץ מול Amazon Athena, שפונה ל-S3 Tables בפורמט Iceberg [15:35].

למה זה חשוב הנקודה המעשית: אף אחת מההגבלות אינה מוגדרת בתוך הסוכן. הן חיות בשכבות שכבר קיימות בארגון — קטלוג, Lake Formation, ה-role של המשתמש — והסוכן פשוט יורש אותן. זו התשובה לשאלה "מי מבטיח שהסוכן לא יראה יותר ממה שמותר".



מסלול היצרן: מיכל, מהנדסת דאטה, מבקשת מהסוכן קוד IaC ו-topic לצריכת אירועי מהירות מטלמטריה; המסלול עובר דרך Amazon MSK, AWS Glue (data quality, catalog, ETL jobs) ו-Amazon S3.

בצד היצרן, הסוכן של מיכל מגדיר את ה-cluster ואת ה-topic ב-Amazon MSK, בונה streaming job, מוסיף את ה-Glue Data Catalog ואת ה-ETL job. ההערה שהוסבר סביבה מה שקורה כשמידע מגיע שבור:

> **Solutions Architect, AWS [17:12] —** יש לנו פה עוד איזושהי יכולת שיכולה לשלוט בקוולִיטי של המידע ולסנן מידע לא נכון... מגיע פתאום איזה מהירות של 500 קמ"ש, אנחנו עדיין לא בפורמולה אחת, זה כנראה לא סביר. או קורדינטות לא נכונות — אפשר להגדיר quality.

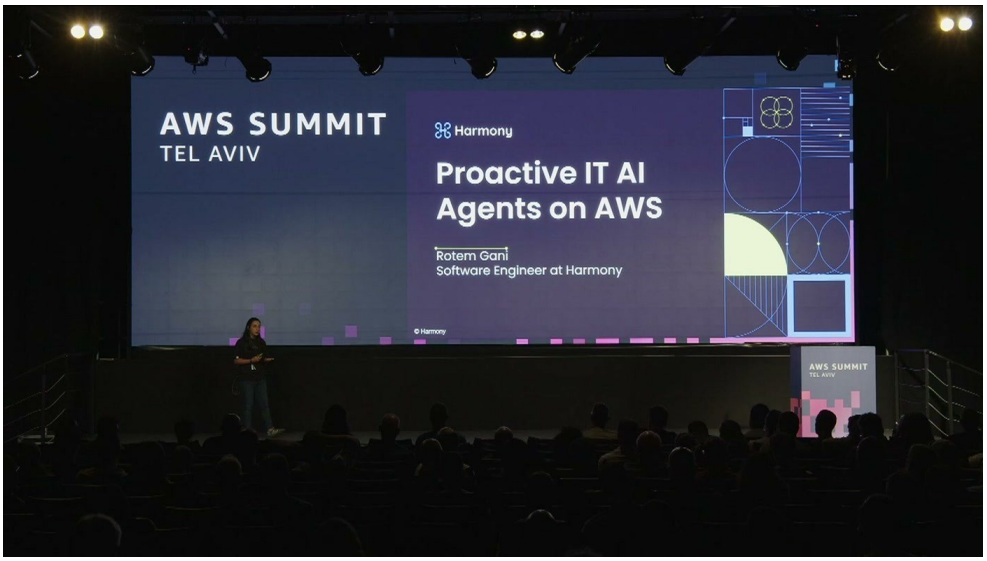
הפלט נכתב ל-S3 Tables, ההרשאות מוגדרות ב-Lake Formation, והכול מפורסם תחת ה-catalog של SageMaker Unified Studio — כלומר אותו קטלוג שממנו הסוכן של תמר גילה את מוצרי הדאטה. זה סוגר את הלולאה בין היצרן לצרכן [17:12].

10. שכבת השירותים: איפה רץ הסוכן עצמו

החלק שסגר את המימוש מיפה את השירותים [18:44]. השוואה שהוצעה: כמו שבעולם הדאטאבייסים עברנו לשירות מנוהל, כך גם להרצת סוכנים:

- **Amazon Bedrock AgentCore —** סביבת הרצה לסוכנים. תואר כ-runtime "מאוד דומה ללמבדה", עם Memory שבנוי לעולם האג'נטי ו-Gateway שמאפשר גישה ל-tools ול-MCP. גם סוכן המשתמש החיצוני וגם סוכן המשתמש הפנימי רצים שם.
- **Amazon Bedrock —** הנגשת המודלים. הנימוק שניתן: "תחת Bedrock יש לנו עושר עצום של LLM, ככה שאפשר לבחור את ה-LLM הרלוונטי ל-use case".
- **Bedrock Knowledge Bases —** תואר כ"שירות RAG מנוהל", כשמאחוריו מקורות לבחירה: OpenSearch, Postgres, S3 Tables.
- **מקורות הדאטה —** טרנזקציוניים (DynamoDB, Aurora) ואנליטיים, שאליהם ניגשים גם כן דרך MCP.
- **ה-ETL עצמו —** "כל ה-ETL'ים האלה שהיינו בונים בעצם בקידוד מסורתי, אנחנו יכולים היום לטפל באמצעות כלי פיתוח אג'נטיים".

11. מקרה לקוח: Harmony — סוכני IT פרואקטיביים



פתיחת החלק השלישי: Software Engineer מ-Harmony מציגה את הפלטפורמה ואת שלוש הארכיטקטורות שמאחוריה.

החלק האחרון היה סיפור לקוח, והוא גם החלק הקונקרטי ביותר. Harmony פועלת בעולם ה-IT Service Management — ITSM. הפתיחה קבעה את הרף:

> **Software Engineer, Harmony [20:20] —** בעולם האנטרפרייז, הערך האמיתי של מודל לא נמצא ביכולת שלו לדבר איתך, לענות לך על שאלות או לראות איתך הזדהות, אלא ביכולת שלו לפתור עבורך בעיה ממשית.

ההסבר מה זה ITSM ניתן לקהל שברובו אנשי דאטה: כל מה שמאפשר לארגון לנהל את העובדים, המכשירים, ההרשאות והתוכנות — למשל, מי מוודא שלעובד חדש שמתחיל ביום שני יש לפטופ והרשאות [20:20]. או בניסוח שנאמר על הבמה: "ITSM הוא העורף הפנימי שמחזיק את הארגון עובד בלי שמרגישים בו, עד שהוא מתחיל לפקשש" [21:52].

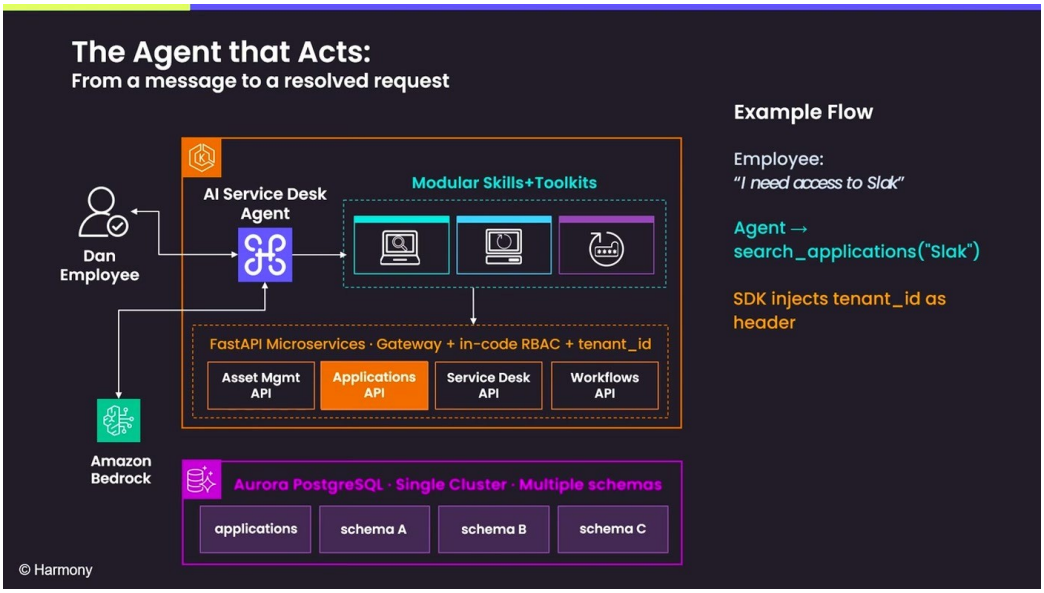
ארבע הבעיות שנמנו: עובדים מחכים ("טיקט שנפתח ביום ראשון, עדיין פתוח ביום רביעי"); אנשי IT שצריכים לג'נגל בין מערכות כדי להרכיב את תמונת הבעיה; ידע קבור אצל אנשים ספציפיים ("במקרה הגרוע, בראשו של מנהל ה-IT, שמן הסתם אולי נמצא עכשיו בחופשה"); ומה שקורה כשהחברה גדלה — הבעיות מתרבות בחזקה [21:52].

העובדים פונים ל-Harmony דרך Slack, Teams או פורטל ייעודי, והסוכן עונה, מדווח ומבצע. היעד שהוגדר: "להפוך את מנהלי IT ממנהלים של טיקטים למנהלים של מערכת שעובדת עבורם" [21:52]. שלושת העקרונות שעליהם זה נשען [23:26]:

- **Multi-tenant isolation —** כל tenant תחום לסנדבוקס דאטה משלו, בלי זליגה בין tenants.
- **AI agents with context —** בזמן שהסוכן רץ יש לו כבר מספיק קונטקסט לפתור את רוב בעיות המשתמשים.
- **Zero Trust Data Access —** כל נגישות למידע שניתנת לסוכן מבוצעת ברמת אבטחה גבוהה, מאומתת בכל בקשה.

> **על המספר** המספר שהוצג: "התשובה הזו עובדת כבר היום בפרודקשן עם לקוחות ממשיים שמדווחים על כ-60% ירידה בפתיחת הטיקטים מצד העובדים שלהן" [23:26]. שימו לב לניסוח — הירידה היא בפתיחת טיקטים, לא בהכרח באחוז הטיקטים שנפתרים אוטומטית. תיאור הסשן הרשמי מנסח זאת כ"פתרון של 60%+ מהטיקטים"; הניסוח שנאמר על הבמה הוא המדויק יותר.

12. The Agent that Acts — מהודעה לבקשה שנפתרה



הזרימה המלאה: העובד דן כותב "I need access to Slak" (שגיאת כתיב מכוונת); הסוכן בוחר בכלי search_applications; ה-SDK מזריק tenant_id כ-header; והקריאה מגיעה ל-Applications API ומשם ל-cluster — Aurora PostgreSQL אחד, מספר schemas.

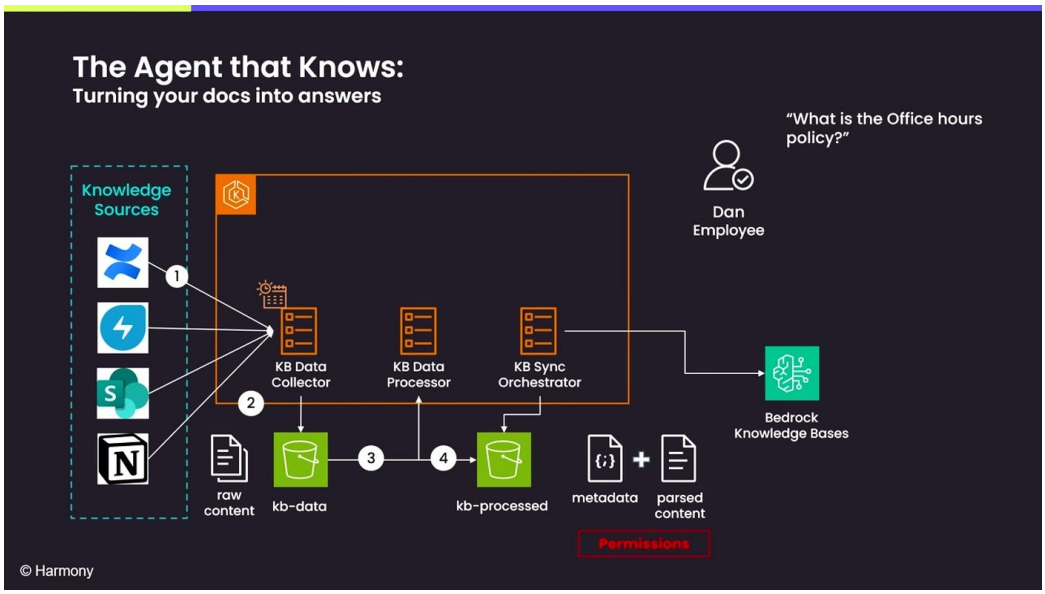
הדוגמה הראשונה בנויה סביב פרט קטן שמסגיר את כל הארכיטקטורה: המילה Slack נכתבה בכוונה בשגיאה. "חסרה מלא האוצי במילה סלק, זה בכוונה שגיאתי מצד דן, תכף נראה למה" [23:26].

מה שקורה מאחורי הקלעים [24:59]: הסוכן מצויד במודלים דרך Amazon Bedrock ובסט של skills ו-tools. הוא בוחר ב-tool בשם search_applications, ומאחור מתבצעת קריאת API למיקרו-שירות Applications — "בהרמוני אנחנו משתמשים בשיטת המיקרו-שירותים, לכל שירות התפקיד האחד והיחיד שלו". בתווך מועברים ה-tenant ID וה-employee ID של דן, "כדי לוודא שכל גישה של מידע עבור דן תבוצע וייחשף בפניו אך ורק המידע שהוא מורשה לראות".

משם מתבצעת שאילתת SQL ל-cluster של Aurora PostgreSQL, ל-schema של applications, עם אינדקס מיוחד שפותר את שגיאת הכתיב של דן. אם נמצאות כמה אפליקציות, מתבצע re-ranking לפי מידת הפופולריות של האפליקציה בארגון הספציפי. התשובה חוזרת לסוכן, שמבצע reasoning ושואל את דן אם התכוון ל-Slack — וברגע שדן מאשר, הסוכן נותן לו את הגישה [24:59].

שלוש החלטות תכנון ששווה להעתיק מכאן: cluster אחד של Aurora עם schema נפרד לכל מיקרו-שירות; הזרקת זהות ב-SDK ולא בקוד כל שירות; ו-ranking שמשתמש בסיגנל ארגוני (פופולריות) ולא רק בדמיון טקסטואלי.

13. The Agent that Knows — מסמכים לתשובות



צינור הידע: מקורות חיצוניים (בהם Confluence ו-Notion) → KB Data Collector → KB Data Processor → KB Sync Orchestrator, כשההרשאות נשמרות צמוד לתוכן לאורך כל הדרך, ועד Bedrock Knowledge Bases.

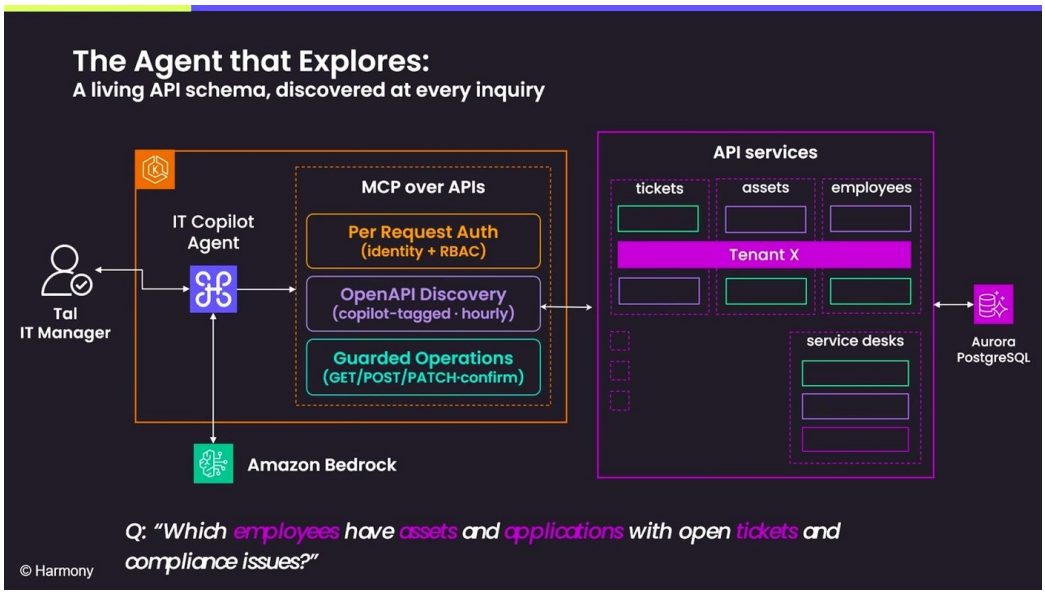
הדוגמה השנייה שינתה פרספקטיבה משאילתת פעולה לשאלת ידע: דן שואל "מה הן שעות הפעילות של המשרד שלי?" כדי לענות על שאלות מהסוג הזה נדרשת עבודת הכנה מקדימה על הדאטה [26:30]:

- **איסוף מתוזמן:** גישה למקורות המידע של הארגון, כולל מקורות חיצוניים כמו Confluence ו-Notion, האיסוף מוגדר כמתוזמן, כדי לוודא שהמידע שנמצא אצל Harmony הוא תמיד העדכני ביותר. המידע מובא ל-S3 bucket ייעודי.
- **עיבוד ונרמול:** שלב processing שמנרמל את הדאטה.
- **הרשאות צמודות לדאטה:** "אנחנו שומרים בצימוד ממש לדאטה עצמו, את סט ההרשאות, set permissions. שוב, כדי לוודא שכל גישה שתיעשה לדאטה הזה אחר כך, תהיה לפי מידת ההרשאות של הלקוח שביצע את הפנייה".
- **אורקסטרציה מנוהלת:** Bedrock Knowledge Bases, עם OpenSearch Serverless לשלב האינדוקס הסמנטי.

בזמן שאילתה, Bedrock Knowledge Bases פותר גם את שלב ה-retrieval: הדאטה מגיע ל-AI Service Desk Agent, שמבצע reasoning, עונה לדן, ובנוסף מצייד אותו ברשימת citations — ה-URLs שעליהם התשובה מבוססת [28:03].

ההחלטה הארכיטקטונית הראויה לציון כאן היא הצמדת ההרשאות לדאטה כבר בשלב ה-ingestion, ולא בדיקתן בזמן השאילתה. זו הדרך היחידה שבה RAG נשאר בטוח כשמקור אחד משרת מספר tenants ורמות הרשאה.

14. The Agent that Explores — סכמת API חיה בכל פנייה



הגישה החדשה: IT Copilot Agent מול MCP over APIs, עם שלושת המנגנונים — Per Request Auth (identity + RBAC), OpenAPI Discovery (copilot-tagged, hourly) ו-Guarded Operations (אישור לפני GET/POST/PATCH משנה-מצב). למטה: השאלה החוצה סכמות שהניעה את השינוי.

הדוגמה השלישית החליפה פרספקטיבה — מהעובד למנהלת ה-IT. היא רוצה מבט רוחבי על כל ה-tenant, ולשאול שאלות כמו "לאלו עובדים יש טיקטים פתוחים עם בעיות קומפליינס" [28:03]. השאלה הזו חוצה שלוש סכמות — employees, assets, tickets — וזה מה ששבר את הגישה הקודמת:

> **Software Engineer, Harmony [29:34] —** השאילתה הזו חוצה סכמות... ובנוסף לכך אנחנו היינו רוצים לאפשר מצב שבו אנחנו כמה שפחות משנים קוד באג'נט הספציפי.

התשובה הייתה סוכן חדש שנמצא בפיתוח — IT Copilot Agent — ומעבר ארכיטקטוני מ-tools מקודדים ל-MCP over APIs, על שלושה עקרונות [29:34]:

- **Per Request Authentication —** כל בקשה לקבלת מידע מגובה ב-tenant ID וב-identity של המבקש, כמו בדוגמה הראשונה.
- **OpenAPI Discovery —** המנגנון המרכזי בדוגמה, שתואר כ"גלה, ואז קרא".
- **Guarded Operations —** לפני פעולה שמשנה מצב בדאטאבייס, הסוכן מוודא מול המשתמש שהפעולה מותרת לפני שהוא מבצע אותה.

המימוש של Discovery הוא הפרט המעשי ביותר בסשן כולו. Harmony בנויה על מיקרו-שירותים, ולכל אחד יש OpenAPI schema שהוא מכריז עליה. כדי שה-Copilot יגלה endpoint, כל מה שנדרש הוא לתייג אותו בתגית של Copilot. בכל ריצה הסוכן מבצע תהליך אינדוקס: "הוא בעצם מגלה את כל ה-OpenAPI specs שמתויגים עם התגית שלו, ובכך הוא יודע — אה, מגניב, אני יכול להשתמש ב-endpoint הזה". לכל endpoint יש description ו-request/response schema, ובכך הסוכן יכול לבצע reasoning ולהבין מתי להשתמש ב-API מסוים [31:05].

> **Software Engineer, Harmony [31:05] —** ברגע שאני רוצה לחשוף API חדש שיתגלה על ידי ה-IT Copilot, כל מה שאני צריכה זה לגשת אליו ולתייג אותו עם התגית הזו של Copilot.

ההדגמה סגרה את הלולאה: כשהשאלה הורחבה וכללה גם את סכמת ה-applications, כל מה שנדרש היה לתייג את המיקרו-שירות הזה , בלי צורך בדפלוימנט, וכבר בריצה הבאה, בשלב האינדוקס, הסוכן מכיר גם אותו ויכול לענות [32:37]. השירותים שנמנו בסיכום: Aurora PostgreSQL כבסיס הנתונים הראשי, Amazon Bedrock להנגשת מודלי LLM, OpenSearch ו-S3. סיכום הדוברת: "AWS הם לגמרי מכפיל כוח שמאפשר לנו להנגיש אחלה מוצר" [32:37].

15. מה לוקחים מכאן

- **המפה של שכבות הזיכרון היא כלי עבודה, לא שקף.** ההבחנה בין agent state (חולף, מנוהל ע"י framework), זיכרון סמנטי (vector store, מתעדכן), ו-prompts/skills (מתנהגים כמו קוד, ב-repository) קובעת איפה כל פיסת מידע חיה, מי מגרסֵן אותה ומה נכנס לביקורת.
- **אל תבנו integration layer חדש.** המסר החוזר בשני החלקים של AWS: אותם APIs, אותם data stores. MCP הוא adapter מעל מה שקיים, וזו הסיבה שהמעבר אינו פרויקט החלפה.
- **ההרשאה שייכת לשכבת הדאטה, לא לסוכן.** בצד AWS זה Lake Formation וה-role של המשתמש; בצד Harmony זה tenant ID בכל בקשה והרשאות הצמודות לדאטה כבר ב-ingestion. בשני המקרים הסוכן יורש הרשאות ולא מגדיר אותן — וזה מה שמאפשר לתת לו גישה לדאטה רגיש.
- **Discovery מבוסס תגיות הוא התובנה הכי ניתנת להעתקה בסשן.** אם כבר יש OpenAPI specs, תיוג endpoints + אינדוקס תקופתי נותן הרחבת יכולות של סוכן בלי דפלוימנט ובלי שינוי קוד בסוכן. זו תבנית שמתאימה כמעט לכל ארגון עם ארכיטקטורת מיקרו-שירותים.
- **כל פעולה שמשנה מצב צריכה שער.** Guarded Operations אצל Harmony הוא היישום של הציר שהוצג בתיאוריה (retrieval מול mutation): ככל שה-discovery אוטומטי יותר, כך נדרש אישור מפורש יותר לפני כתיבה.
- **מה שלא נאמר:** לא הוצגו נתוני עלות, לטנציה, דיוק או שיעור שגיאות של הסוכנים, ולא הוצג דמו חי — כל הארכיטקטורות הוצגו כדיאגרמות. מי שבוחן את התבניות האלה לפרודקשן יצטרך למדוד את הפרמטרים האלה בעצמו.

הסגירה מצד AWS הייתה קריאה לפעולה קצרה: "הכדור עובר אליכם. תתחילו לבנות, תעשו את ההתאמות הרלוונטיות. אתם צריכים עזרה — אנחנו כאן, לכל אחד מכם יש אקאונט מנג'ר" [34:08].

מילון מונחים

מונח	מה זה, כפי שהוצג בסשן
ReAct loop	Reason → Act → Remember בלולאה. המבנה שהופך שימוש ב-LLM למערכת אג'נטית [3:08].
MCP	Model Context Protocol. סטנדרט אחיד לתקשורת בין סוכן לכלים; מאפשר גילוי כלים זמינים והפעלתם [9:19].
Agent state	זיכרון העבודה קצר-הטווח של הסוכן; מחזיק את מה שחזר מהפעלת כלים בין איטרציות [7:47].
Skills	הוראות מובנות שהסוכן שולף ומפעיל שוב ושוב; "מתכונים לשימוש חוזר", מנוהלים כמו קוד [7:47].
Context compaction	סיכום ודחיסה של ההיסטוריה כשחלון הקונטקסט מתמלא, כדי שהלולאה תוכל להמשיך [6:13].
Bedrock AgentCore	סביבת הרצה מנוהלת לסוכנים; runtime, Memory, Gateway ל-tools ול-MCP [18:44].
Bedrock Knowledge Bases	שירות RAG מנוהל; מקורות אפשריים: OpenSearch, Postgres, S3 Tables [18:44].
SageMaker Unified Studio	הסביבה שמתכללת את השירותים האנליטיים של AWS; מחזיקה Business Catalog מעל Glue [15:35].
Lake Formation	שכבת ההרשאות על הדאטה לייק — סינון שורות, מיסוך עמודות וביקורת [15:35].
OpenAPI Discovery	תיוג endpoints בתגית ייעודית + אינדוקס תקופתי בסוכן — "גלה, ואז קרא" [29:34].
Guarded Operations	אישור מפורש מהמשתמש לפני פעולה שמשנה מצב בדאטאבייס [29:34].
ITSM	IT Service Management — ניהול העובדים, המכשירים, ההרשאות והתוכנות בארגון [20:20].