מ-POC לפרודקשן: בניית סוכנים על AWS עם AgentCore

tlv-aim202 — סיכום סשן, AWS Summit Tel Aviv 2026

מיכאל (Solutions Architect, AWS) · יעקב טייב (Solutions Architect, מומחה Agentic AI, AWS) · נמרוד קור (Co-Founder ו-Baz, CTO)

10 בספטמבר 2026 | משך: כ-39 דקות | הופק מהקלטת הסשן

סיכום מאת קובי אלמוג

על המסמך הזה

המסמך מסכם את הסשן tlv-aim202 — "Introduction to Building and Running Agentic Applications on AWS" — מתוך AWS Summit Tel Aviv 2026, במסלול Building AI and Agentic Applications. הוא נבנה מתמלול אוטומטי של ההקלטה המלאה ומהסליידים שחולצו מהווידאו עצמו, כך שאפשר לחזור לתוכן בלי לצפות שוב בכל הסשן. חותמות הזמן בגוף המסמך מפנות לנקודה המדויקת בהקלטה.

הסשן בנוי כשלושה חלקים ברצף אחד: מיכאל לוקח סוכן POC שרץ על הלפטופ וסוגר עליו פער אחרי פער בעזרת רכיבי Amazon Bedrock AgentCore; יעקב טייב מוסיף את שכבת הדאטה — Knowledge Bases, agentic retrieval, MCP חיצוני וזיכרון — ומרכיב את הכל ב-AgentCore Harness; ונמרוד קור מ-Baz מראה מערכת אמיתית בפרודקשן שנבנתה מעל אותם רכיבים.

איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. מה נאמר ומה שווה לקחת.
- **פרקים 2–7 — סגירת הפערים:** הסוכן של ה-POC, ואיך Gateway, Identity, Observability, Evaluation ו-Runtime סוגרים כל אחד פער אחר.
- **פרקים 8–11 — שכבת הדאטה:** Agentic Context, Knowledge Bases, agentic retrieval, MCP, זיכרון, ו-AgentCore Harness.
- **פרק 12 — המקרה של Baz:** Spec-Driven Review בפרודקשן, הבאגים שהתגלו והפתרונות הארכיטקטוניים.
- **פרק 13 — 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 ("אג'נט קור" = AgentCore, "סטרנד" = Strands) ותוקנו ידנית מול הסליידים. הציטוטים אומתו מול התמלול בחותמת הזמן המצוינת ומובאים כלשונם, כולל שיבושי תמלול. מספרים שמופיעים על הסליידים צוינו ככאלה. שם המשפחה של הדובר הראשון לא נאמר בהקלטה, ולכן מצוין תפקידו בלבד.

1. תקציר מנהלים

הסשן לא מנסה לשכנע שסוכנים עובדים. הוא מניח שכבר בניתם אחד, שהוא מרשים בהדגמה, ושהוא לא ישרוד יום אחד בפרודקשן — ואז עובר רכיב-רכיב על מה שחסר. המבנה הוא "פער, ואז הרכיב שסוגר אותו", והוא חוזר על עצמו שמונה פעמים.

- **הפער בין POC לפרודקשן הוא הנושא, לא הסוכן.** מיכאל פותח בכך ש"יש פער של ממש בין אג'נט של POC ל-אג'נט של פרודקשן", ושהדברים שמזניחים בו הם "סקל, סקיורטי, רזיליינס, אוטנטיקיישן, אוטריזיישן" [0:00].
- **סוכן עם MCP-ים מוטמעים הוא מונוליט.** כשה-MCP-ים חלק מקוד הסוכן, כל שינוי כלי דורש redeploy: "בעצם יצרתי כאן מונוליט" [3:05]. AgentCore Gateway מנתק אותם ל-endpoint אחד — והוספת כלי חדש לא נגעה בקוד הסוכן [4:36].
- **Observability של סוכן היא לא observability של אפליקציה.** הלולאה האג'נטית מייצרת iteration-ים עם input/output tokens, latency, prompt וכלים לכל אחד. כך אותר באג latency של 36 שניות שנבע מהוראת double-check בתוך ה-prompt [7:42, 9:13].
- **Evaluation הוא המענה לשאלה "מה קורה עם אלפי משתמשים".** "אני לא יכול לשבת ליד כל משתמש ולוודא שהאג'נט שלו עובד תקין" [9:13]; AgentCore Evaluation מחזיר ציון כמותי ברמת session, trace וכלי — ובנוסף הסבר מילולי לציון [10:45].
- **שכבת הדאטה היא חצי מהסיפור.** יעקב טייב מראה שהמעבר מ-RAG קלאסי ל-agentic retrieval הוא מה שמאפשר לענות על שאלות מורכבות, שבהן "בעצם הסוכן מנהל את החיפוש" [16:53].
- **AgentCore Harness מרכיב סוכן בלי קוד אורקסטרציה.** "אפס קוד באופרציה, אפס קוד באורקסטרציה" [23:02] — בחירת מודל, system prompt, memory ו-tools מתוך הקונסול, ובסוף "אני לא כותב את הקוד של הסוכן עצמו" [24:35].
- **Baz מוכיחה את הטענה על מערכת אמיתית.** מערכת Spec-Driven Review שמפצלת ל-20 סוכני ולידציה במקביל, רצה ב-AgentCore Runtime, ומשלמת רק על זמן CPU בפועל [29:10, 32:13].

2. נקודת הפתיחה: סוכן שנבנה ב-Kiro ורץ על הלפטופ

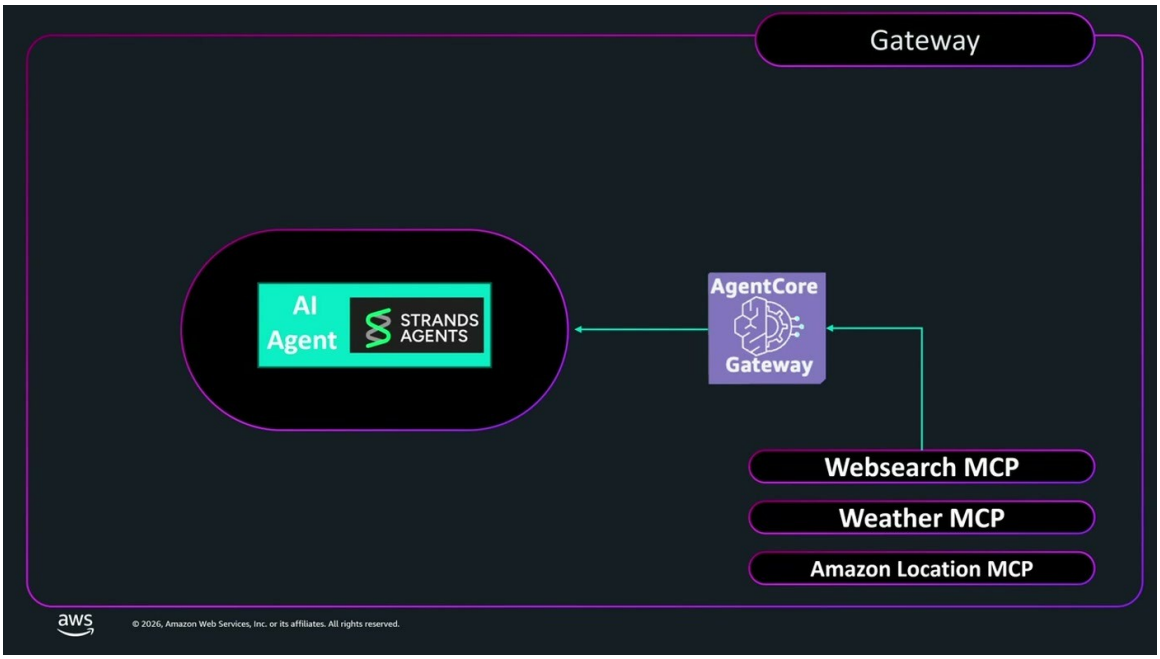
הדוגמה שמלווה את כל החלק הראשון היא travel agent — לדברי הדובר, "משהו שכבר הפך לדוגמה קלאסית בעולמות של אג'נטים" [0:00]. הוא נבנה בסביבת פיתוח אג'נטית (Amazon Kiro) מעל framework אג'נטי (Strands Agents), והדובר מדגיש שהבחירה אינה קריטית: "באותה מידה יכולתי להשתמש גם בפרמורק אחר, LangGraph, Crew AI למי שמכיר, או אפילו פלטפורמה אחרת כמו [1:32] "Codex, Claude Code — כולם מתחברים ל-Amazon Bedrock, שמנגיש את מודלי ה-frontier.

הדמו עובד: הסוכן מחזיר נקודות עניין בחיפה, מציג את סטלה מריס על מפה, מביא תמונות ותחזית מזג אוויר. ואז מגיעה הפואנטה — "זה נראה מרשים זה נראה יפה אבל זה בדיוק אותו אג'ן של POC שדיברנו מקודם הוא גם רץ על הלפטופ שלי" [1:32]. מכאן ואילך הסשן הוא רשימת הפערים.

> **Solutions Architect, AWS [0:00] —** עד כמה משמעותי הפער הזה? האם אפשר לגשר על הפער הזה? בסשן שלנו אנחנו הולכים לבנות אג'נט של POC ולסגור את הפערים

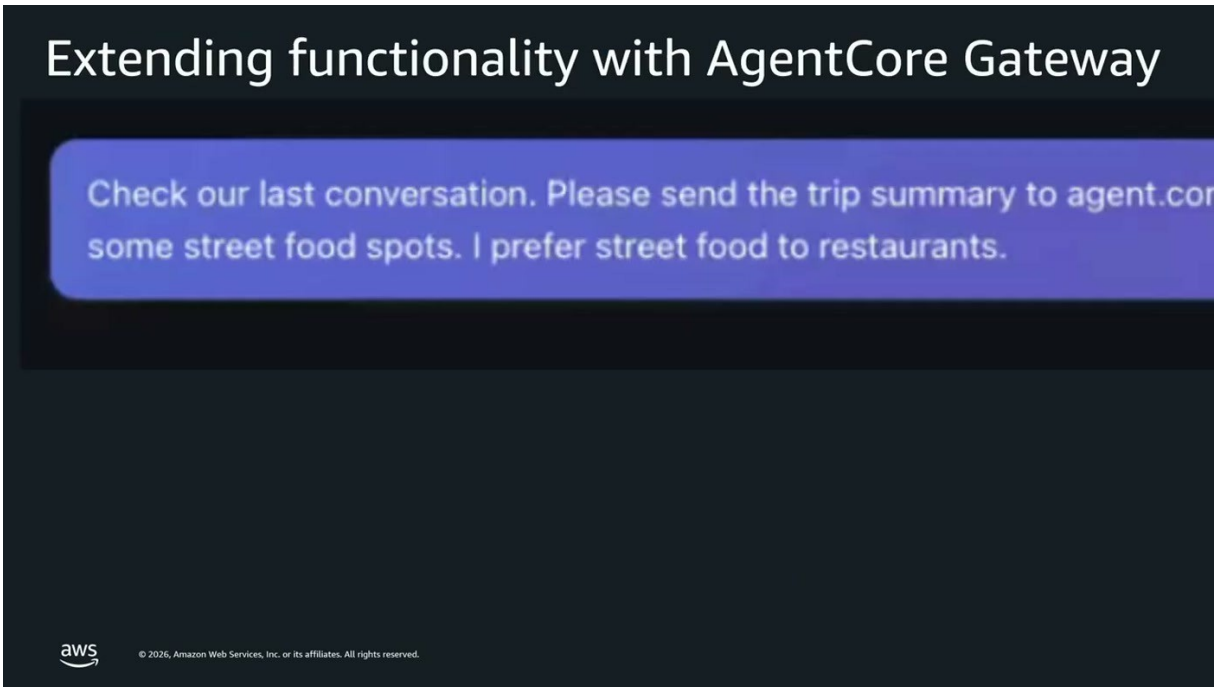
3. פער ראשון — המונוליט האג'נטי: AgentCore Gateway

הסוכן צורך כלים דרך MCP, והבעיה היא בעלות: הסוכן עצמו מנהל את ה-endpoint, את ה-authentication וה-authorization של כל MCP. "אני רוצה להוריד MCP, להוסיף MCP, צריך לשנות קוד, אני צריך לעשות redeploy לאג'נט. בעצם יצרתי כאן מונוליט" [3:05]. הדובר מדגיש שמונוליט אינו רע כשלעצמו — הבעיה היא שהוא תכנן להשתמש באותם MCP-ים כ-reusable components גם בסוכנים אחרים.



הסוכן מפסיק לנהל MCP-ים בעצמו: ה-Gateway עומד בינו לבין Websearch, Weather ו-Amazon Location.

הדגמת הערך הייתה מדויקת: בקשה לשלוח את תוכנית הטיול במייל נכשלה — לסוכן אין כלי כזה. הכלי נוסף ל-Gateway, לא לסוכן: "אני לא מוסיף אותו לאג'ן שלי, אני מוסיף אותו לגייטווי... לא צריך לעשות redeploy agent והנה התוצאה" [4:36]. יתרה מזו, שרת המייל עצמו הוא REST API — ה-Gateway יודע להתממשק ל-Lambda, ל-API Gateway, ל-REST API או ל-MCP server אחר, ולחשוף את עצמם כ-Single MCP Endpoint אחד.



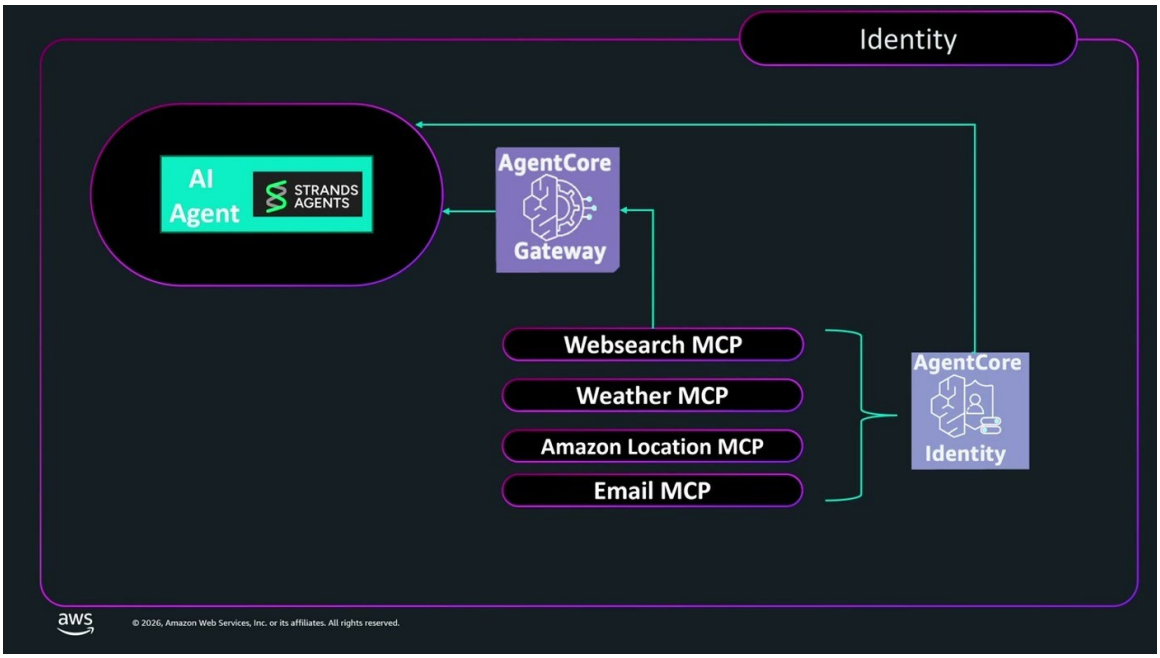
סוגי ה-targets שה-Gateway יודע לעטוף, וההגדרה שלהם במסך היצירה.

- **Semantic search על הכלים:** כשב-Gateway יש מאות כלים, אפשר לבקש ב-natural language רק את תת-הקבוצה הרלוונטית למשימה. הנימוק: "פחות כלים, אג'נט יותר מהיר ולרוב גם יותר מדויק" [4:36].
- **Governance דרך endpoint יחיד:** "אם אני מנתר את ה-endpoint הזה אני יודע בדיוק להגיד מי ניגש לכלים מסוים ומתי" [4:36].
- **Reusable components:** אותו endpoint משרת סוכנים שונים [6:07].

4. פער שני — מי מאשר את הפעולה: AgentCore Identity

היכולת לשלוח מייל נוספה, אבל היא מסוכנת כברירת מחדל. "אני לא רוצה ש-agents ישלח emailים בשמי אני רוצה שליטה מלאה לתהליך" [6:07] — ולכן נוסף כפתור authorize ב-UI שמחייב את המשתמש לבצע login בזהות הנקובה שלו.

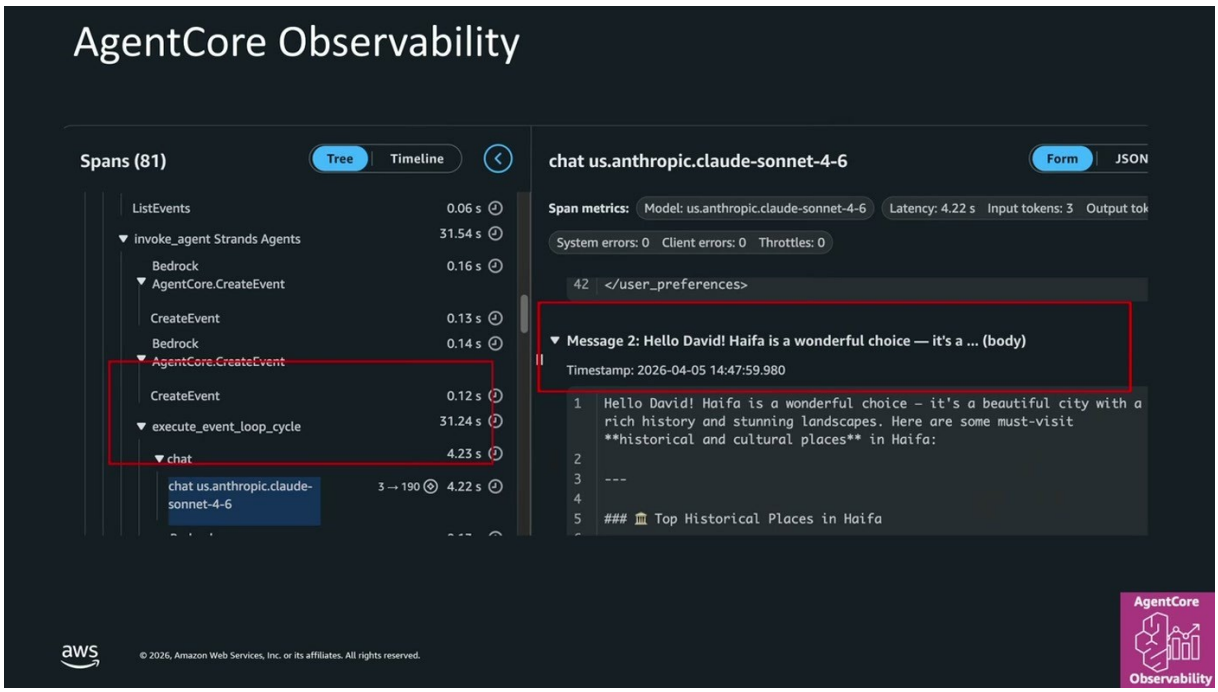
החלק הכן בסשן הוא ההודאה שזה בדיוק החלק שמפתחים נמנעים ממנו: "תמיד רציתי לערוץ קדימה לפתח פיצ'רם במקום זה תמיד מצאתי את עצמי... להתמודד עם הנושא של identity, להבין מושגים כמו machine to machine authentication, two-leg authentication, three-leg authentication" [6:07] — ועוד לפני שאלת אחסון המפתחות. AgentCore Identity מאפשר להגדיר את כל זה כמטא-דאטה — API keys או OAuth בסגנון Amazon Cognito או Okta — ולצרוך אותה מהקוד.



Identity נכנס כשכבה נוספת לצד ה-Gateway, וחולש על כל ה-MCP-ים כולל Email MCP החדש.

5. פער שלישי — למה זה איטי: AgentCore Observability

שאלה תמימה על מחיר כניסה לסטלה מריס חוזרת עם תשובה — "אבל אחרי 36 שניות" [7:42]. ההבדל מ-observability מוכר מוסבר במפורש: הסוכן חושב בלולאה, "יכול להיות שיקח לו איתרציה אחד להגיע לפתרון מאה היתרציות", ולכל iteration יש input tokens, output tokens, latency, ה-prompt שנשלח למודל והכלים שהופעלו [7:42].



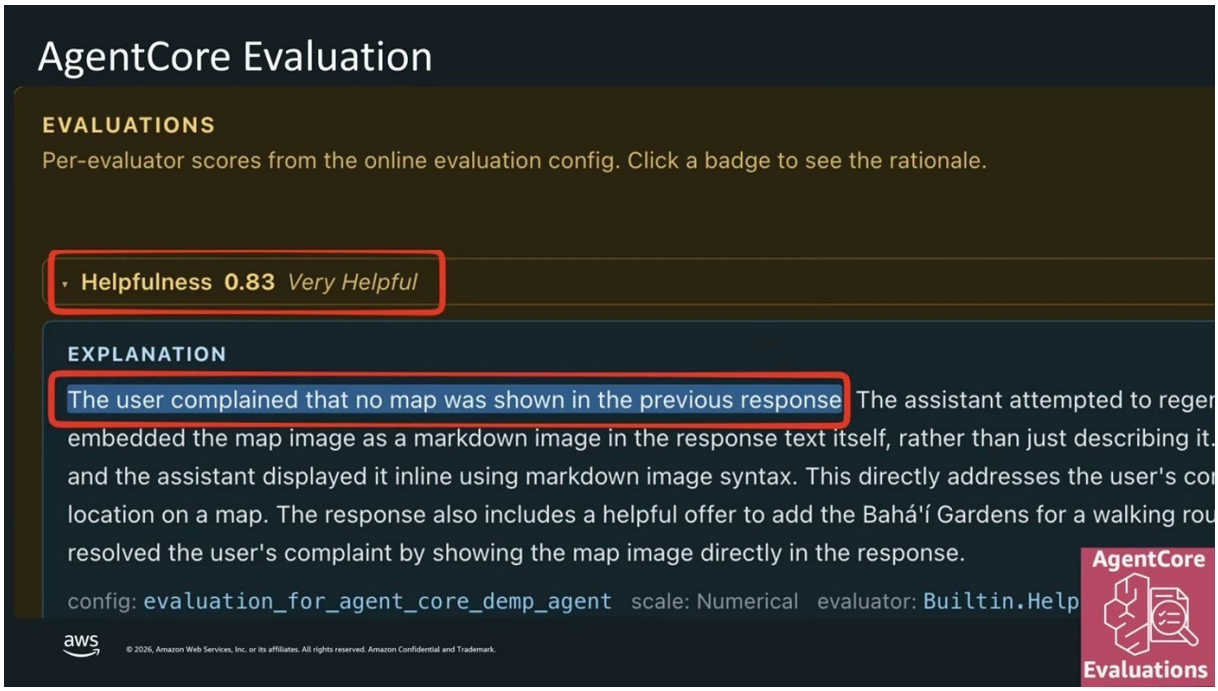
תצוגת ה-spans: ציר זמן של הקריאות, עם המודל, ה-latency וספירת הטוקנים לכל שלב.

ההדגמה סוגרת מעגל: הניתוח הראה איזה iteration צרך את רוב הזמן, ומה היה ה-prompt שנשלח. הסיבה ל-36 השניות הייתה הוראה בתוך ה-prompt — "שאם יש איזשהו נושא שקשור בנושא כספי תעשה דאבל צ'ק ודאבל צ'ק לוקח זמן" [9:13]. כלומר, לא באג בתשתית אלא החלטת תכן ב-prompt, שרק telemetry ברמת ה-iteration יכולה לחשוף.

6. פער רביעי — איכות בקנה מידה: AgentCore Evaluation

הבאג השני היה שקט יותר: בקשה להציג מיקום על מפה החזירה טקסט בלי תמונה, ורק בקשה חוזרת הניבה את המפה. למפתח שעושה QA לעצמו זה ברור — אבל "מה קורה אם יש לי אלפי משתמשים? אני לא יכול לשבת ליד כל משתמש ולוודא שהאג'נט שלו עובד תקין" [9:13]. הצורך מוגדר כמנגנון שרץ מאחורי הקלעים ומחזיר נתונים כמותיים על התנהגות הסוכן.

- **ברמת session:** האם הסוכן עמד במשימה כולה — "ביקשתי מאג'נט, תבנה לי תוכנית של הטיול בחיפה, במקום זה קיבלתי רשימה של תחנות דלק" [9:13].
- **ברמת trace:** שאלה ותשובה בודדות; המטריקה בדוגמת המפה היא helpfulness [10:45].
- **ברמת כלים:** האם נבחר הכלי הנכון למשימה — "ביקשתי מי אג'נט שלחת את התוכנית של טיול באימילים במקום זה קיבלתי תמונות של המקום" [10:45].



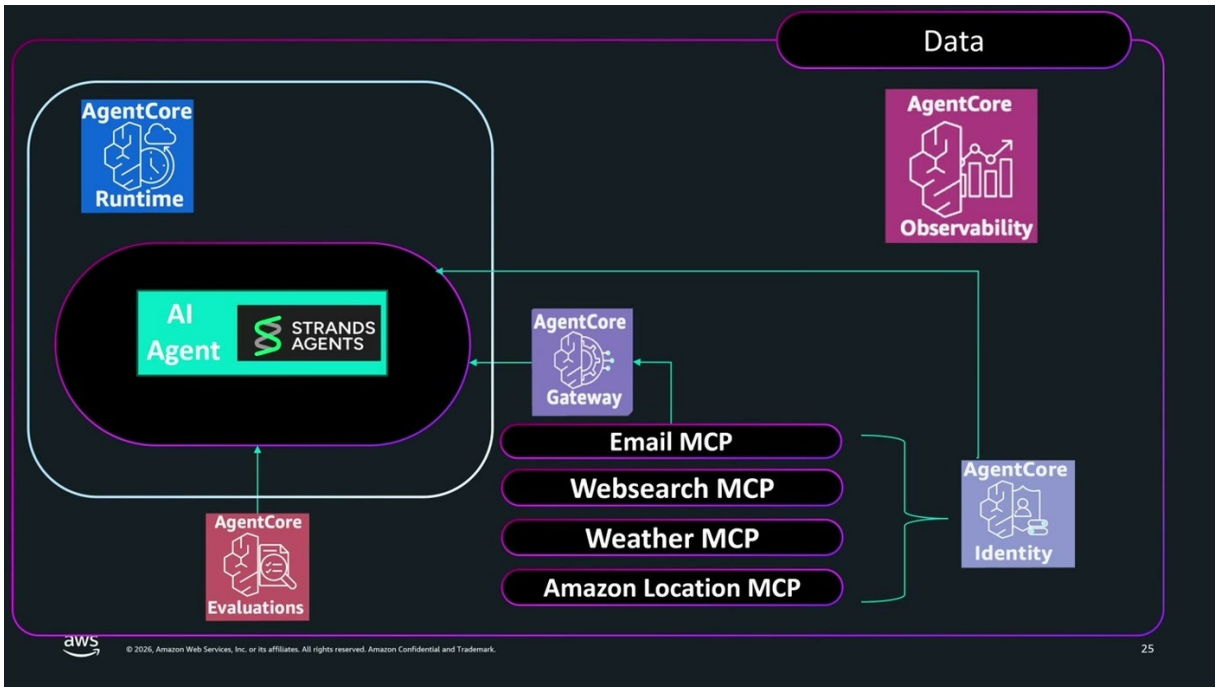
ציון helpfulness של 0.83 ("Very Helpful") לצד ההסבר המילולי שמנמק אותו — כפי שהוצג על המסך.

המדד בסולם 0–1 הוא מה שמאפשר את שכבת התפעול: thresholds, אלרטים ודשבורדים. הנקודה שהודגשה היא לא המספר אלא הנימוק — "אני לא מקבל סתם מספר אני מקבל גם הסבר מילולי למה קיבלתי את המספר הזה, אני יכול לקרוא את ההסבר הזה ובשאיפה להבין איפה אני צריך להשתפר" [10:45].

7. פער חמישי — איפה זה רץ: AgentCore Runtime

אחרי כל הסגירות, הסוכן עדיין רץ על localhost — "הוא לא סקיור, הוא לא סקיילבל, הוא אפילו לא נגיש לי אף אחד אחר" [10:45]. AgentCore Runtime הוא רכיב ה-compute המנוהל, ושלוש הסיבות שניתנו לו הן scalability (אלפי sessions, סקייל אוטומטי, serverless), אבטחה בתכן, ואינטגרציה מובנית עם כל שאר רכיבי AgentCore out of the box [12:16].

> **Solutions Architect, AWS [12:16]** — כל סשן שאני מייצר, מייצר למכונת מייקרווים חדשה. יש לי session isolation מוחלט. אין לי בעיה של noisy neighbors, אין לי בעיה של data או storage sharing.



התמונה המלאה בסוף החלק הראשון: הסוכן בתוך Runtime, מוקף Gateway, Identity, Observability ו-Evaluations.

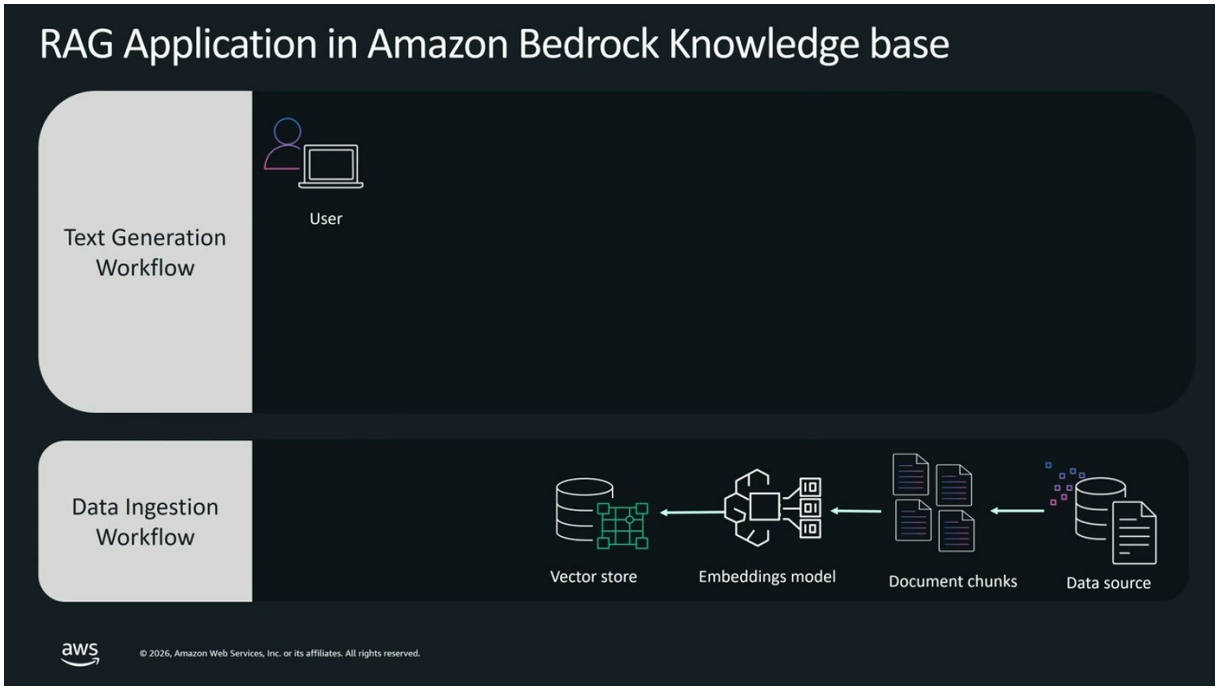
8. השכבה החסרה: Agentic Context

יעקב טייב, Solutions Architect מומחה Agentic AI, פותח בשאלה אחת: "האם לסוכן שלכם יש את הידע ואת הכלים שהוא צריך כדי לבצע את המשימה שנתתי לו" [13:48]. מה שהסוכן רוצה: מידע מהרבה מקורות, עדכני אך גם היסטורי, איכותי ונושא הקשר עסקי. מה שיש בפועל הפוך — "הדאטה בדרך כלל הומוגני, מכיל מלא מלא מלא דופליקציות, חלקו לא רלוונטי, בטח שהוא לא כולל את הקונטקסט העסקי" [13:48].

המעבר מ-raw data ל-knowledge מוצג כשתי תחנות. תחנת ה-processing היא ניקוי רעש (הדוגמה: PII) ואז טרנספורמציה למבנה אחיד. תחנת ה-metadata היא enrichment, ומתוכה הודגשו שניים: קטגוריזציה — "סוכן צריך לדעת שמסמך מסוים הוא מסלול טיול, ומסמך אחר הוא רוויושי מסעדה"; והקשר שהסוכן זקוק לו, למשל שמסלול הטיול על הגרפיטי בניו יורק עודכן לפני ארבע שנים ואולי אין טעם להשתמש בו [15:20].

9. מ-RAG ל-Agentic Retrieval

דרך המלך להנגשת מסמכים פנימיים היא אפליקציית RAG — ב-AWS, Knowledge Base. הצד הראשון הוא הייצור: data sources (למשל S3), chunking לחלקים הגיוניים, מעבר במנוע embedding שממיר טקסט, תמונה, וידאו או פודקאסט ל-"וקטור מתמטי, אובייקט מתמטי" [15:20], ושמירה ב-vector store. הצד השני הוא הצריכה: שאילתת המשתמש עוברת לאותו מרחב וקטורי, נשלף הקונטקסט הרלוונטי, והסוכן מחזיר תשובה.



שני חצאי ה-RAG ב-Bedrock Knowledge Base — ingestion משמאל, שליפה וייצור טקסט מימין.

המגבלה הוצגה ביושר: זה עובד מצוין כששאלה ממופה לקטע קונטקסט מסוים — "באיזה רחוב נמצא מגדל אייפל". זה נשבר בשאלות שאנשים באמת שואלים, כמו השאלה שהוצגה על המסך: איפה אפשר לשחות עם הילדים, לשתות קפה ולקרוא ספר מספרייה מקומית. כאן נדרש agentic retrieval: "בעצם הסוכן מנהל את החיפוש" [16:53] — הוא מפרק את השאלה, מתכנן, מושך מקורות, ולפעמים מחליט שזה לא מספיק ומושך עוד [18:26].

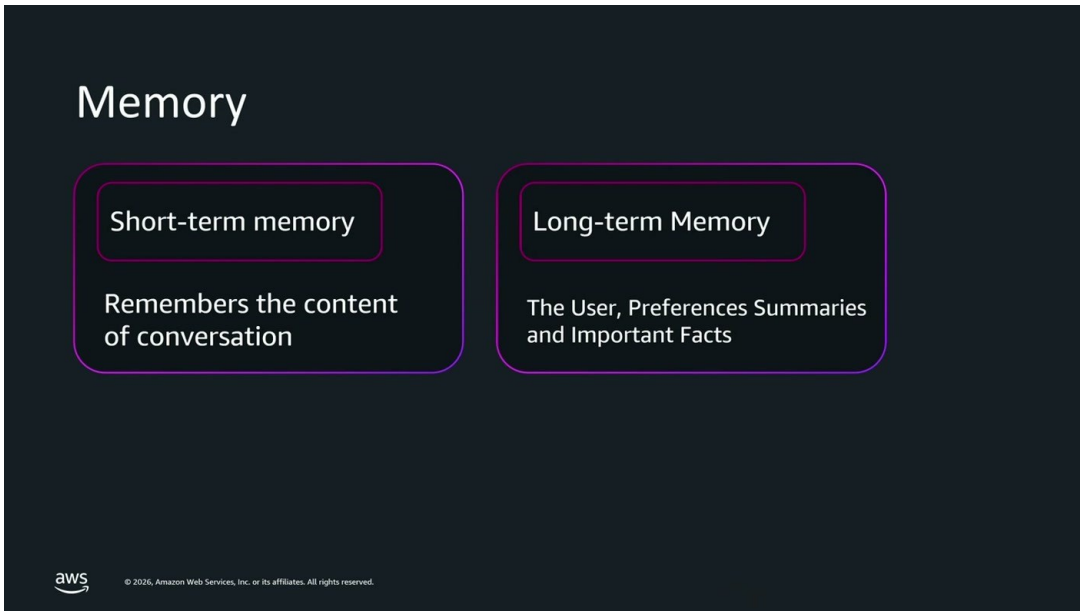
ששת שלבי ה-agentic retrieval, מ-User asks ועד Return, עם לולאת Re-plan and iterate.

החלק המעשי היה הדגמת קונסול: יצירת Knowledge Base ב-AgentCore — שם, בחירת מקור מידע (S3), Create — ומיד אחריה חיבורו ל-Gateway, מנימוק מפורש של governance: "זה יאפשר לי הרבה מאוד שליטה למי מותר לגשת לאותו knowledge base" [19:58].

10. כלים חיצוניים, ממשל וזיכרון

מסמכים פנימיים זה חצי; הזמנת טיסות, קונצרטים ומסעדות זה שירותים שאינם שלי. התשובה היא שרתי MCP — סטנדרט פתוח שבו הסוכן מגלה את הכלים דינמית: "הוא לא צריך לדעת איזה כלים נמצאים שם, כל מה שהוא צריך לדעת זה לגשת לאותו שרת והוא יגלה איזה כלים נמצאים שם בצורה דינמית לגמרי" [19:58]. ה-Gateway כמעט תמיד ישולב מעליהם — גם כנקודת חיבור אחת וגם כשכבת ממשל: מי מורשה לקנות טיסות, לאן, מתי, באיזה מחיר, ובאיזה סוכן — "אחד מהם משמש ל-VIP customers שיש לי, שממש יכולים לקנות את הטיסות, ואחד אחר משמש לשאר המשתמשים" [21:31].

הפער האחרון בשכבת הדאטה הוא זיכרון, והוא הוצג דרך תקלה יומיומית: הסוכן המליץ על מסעדת פסטה, נאמר לו שהמשתמש רגיש לגלוטן — "אבל אז פתחתי סשן חדש ועוד פעם הוא מציע לי על מסעדת פסטה" [21:31].

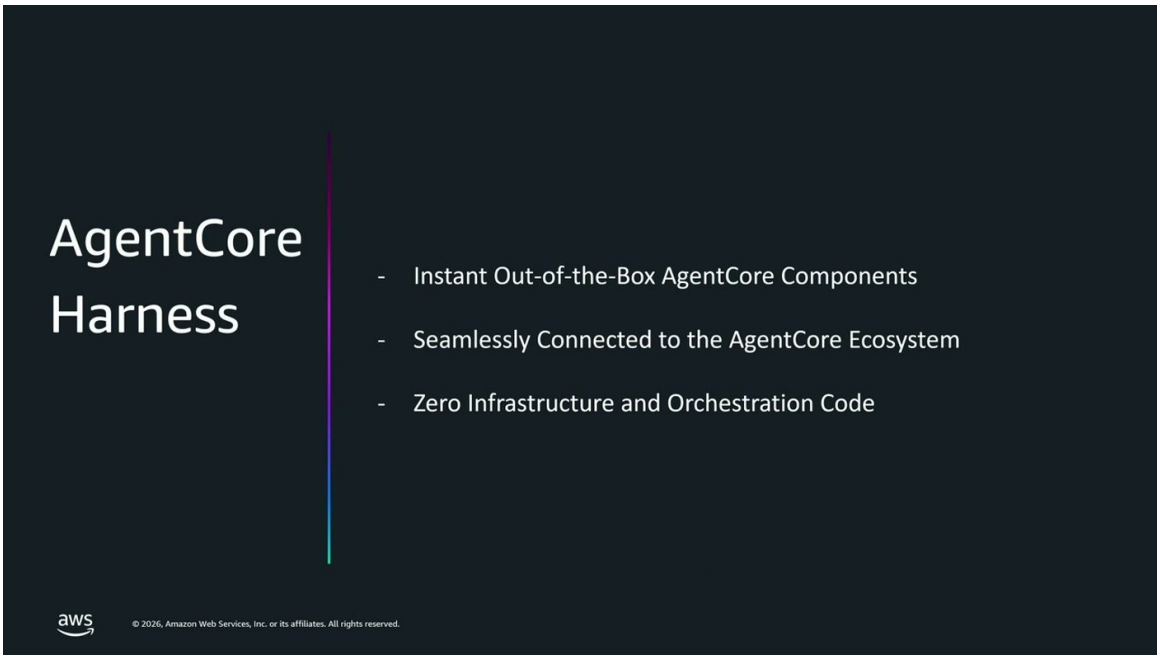


שני סוגי הזיכרון: תוכן השיחה הנוכחית מול סיכומים, העדפות ועובדות קבועות על המשתמש.

- **זיכרון מנוהל out of the box:** AgentCore Memory מספק את שתי הרמות בלי מימוש עצמי [23:02].
- **בידוד לפי משתמש:** "נקרא actor ID, כל יוזר שהשתמש היא isolated, לכל אחד יש את הזיכרון שלו" [23:02].
- **זה החלק שקשה לבנות לבד:** "מי ממכם שהתחיל לפתח memory לפני שנה יודע שזה פשוט קשה" — מתי שומרים, איפה, איך עושים consolidation, מתי מוחקים [23:02].

11. AgentCore Harness — הרכבה בלי קוד אורקסטרציה

אחרי שהדיאגרמה התמלאה קוביות, הדובר מקדים את ההתנגדות הצפויה: "תגיד זה מסובך הסיפור הזה? זה מורכב?... כל הסיפור של חיבור הכלים הזה, חבר את הממורי, חבר את הגייטוי, חבר ה-identity, קצת אוברוולמינג בשבילי" [23:02]. התשובה היא AgentCore Harness.



שלוש ההבטחות של ה-Harness כפי שהופיעו על השקופית: רכיבים מוכנים, חיבור לאקוסיסטם, ואפס קוד תשתית ואורקסטרציה.

— Solutions Architect, AWS [23:02–24:35] הוא חלק מהאקו-סיסטם, והוא דורש, שימו לב, אפס קוד באופרציה, אפס קוד באורקסטרציה. בעצם אני לא כותב את הקוד של הסוכן עצמו

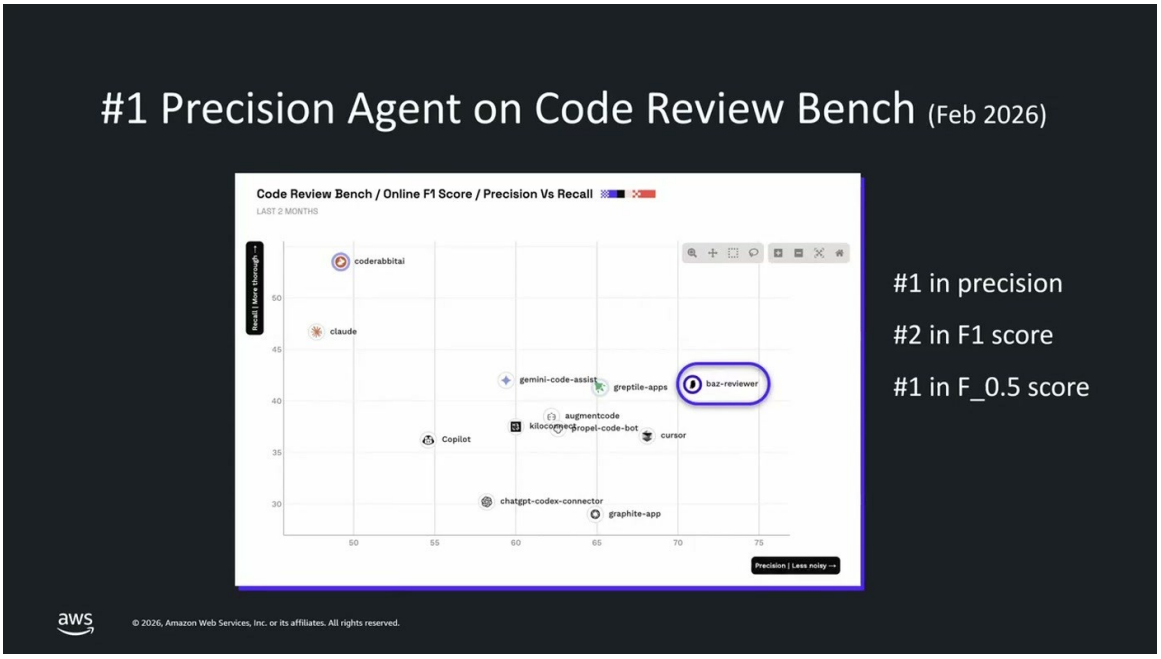
הדגמת הבנייה בקונסול היא רצף שלבים: שם; בחירת מודל מתוך מאות המודלים ב-Bedrock (כולל ספקים חיצוניים); הדבקת system prompt שמגדיר מי הסוכן ואילו כלים יש לו; חיבור memory; ואז tools — חיבור ה-Gateway כ-single point לכל הכלים, בתוספת Browser Tool. לחיצה על Create, והסוכן מוכן לבדיקה ב-Test Harness [24:35].

שתי שיחות הבדיקה תוכננו כדי לאמת רכיבים, לא רק תשובות. הראשונה ("אני עכשיו ברומא... לשחות, לשתות קפה, ולקרוא ספר בספרייה המקומית") נועדה להראות חיבור לכלים, שליפה מה-Knowledge Base והגעת chunks. השנייה נפתחה כ-session חדש וביקשה טיול דומה בלונדון, כדי לוודא שה-memory באמת נשמר: "אבל זה בעצם מה שרצינו לבדוק שהוא גם זוכר" [26:06].

12. המקרה של Baz: Spec-Driven Review בפרודקשן

שקופית הפתיחה של החלק השלישי, עם הדובר ותפקידו כפי שהוצגו.

נמרוד קור, Co-Founder ו-CTO ב-Baz, הציג פלטפורמה ל-code review ואת התהליך הארכיטקטוני שעברה: "אנחנו בנינו אג'נטיק גרף מעל AWS ורציתי לספר לכם את התהליך שעשינו ולמה בחרנו במה שבחרנו" [27:38]. נקודת הפתיחה הייתה איתור באגים שפוגעים בפרודקשן, ומשם התרחבה למפרט מול runtime, security review ותכנון.



מיקום המוצר בבנצ'מרק חיצוני ל-code review, לפי הנתונים שהוצגו על השקופית (Feb 2026).

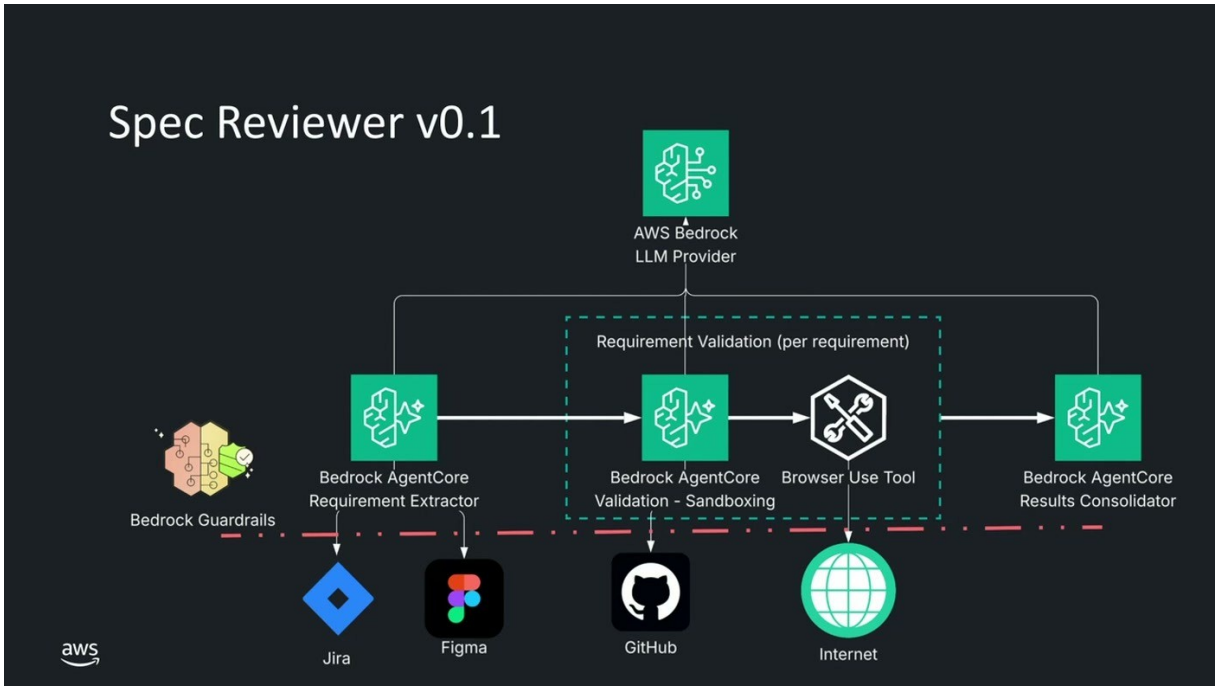
הבעיה שהובילה לשינוי הגישה מוכרת לכל מי שמייצר קוד עם LLM: הפיצ'ר עובר CI, עובר גם את ה-review, ובכל זאת לא עושה את מה שהתבקש — "כי קלוד ראה את הצד הזה, ראה את הצד הזה, בנה, אבל זה לא עושה את מה שרציתי שזה יעשה" [29:10].

> **Co-Founder ו-CTO, Baz [29:10]** — קוד נכון זה לא קוד שעושה את מה שאני רוצה שהוא יעשה... וכמו שיש לנו Spec Driven Development בעצם מה שאנחנו בנינו זה Spec Driven Review

הגישה: לחבר את הסוכן ל-Jira כדי להבין מה היה הטיקט, ל-Figma כדי להבין את העיצוב, ל-GitHub כדי לראות את הקוד, ול-Playwright כדי לגשת לדפדפן ולשחק בפועל ב-UI ולבדוק אם הוא עושה את מה שהוא אמור [29:10].
הפריסה הראשונית הייתה פשוטה — Docker image שרץ ב-AgentCore Runtime — ונימוק העלות היה מפורש: "זה שברנטיים בעצם לא משלמים על הזמן שהCPU לא רץ" — בהמתנה לתשובת LLM או לטעינת UI [29:10].

שלושת הכשלים שהתגלו בשימוש עצמי

- **הסוכן גזר דרישות מהקוד במקום מהטיקט:** "הקוד זה לא דוגמה למה היה צריך לעשות" — התוצאה הייתה דרישה טכנית מדי שהיא למעשה החלטת מימוש, והסוכן אישר את עצמו [30:42].
- **ולידציה שהתייאשה באמצע:** עם רשימה של שלוש דרישות הוא בדק את כולן; עם עשרים — "בחמישית-שישית הוא אומר טוב אחי הפיצ'ר אחלה בוא נעבור למשימה הבאה" [30:42].
- **הרצת קוד זר על המכונה:** הסוכן נכנס ללינקים ולוחץ כפתורים, ומשמעות הדבר היא ש"הוא בעצם מריץ קוד על המכונה שהוא יושב עליה" [30:42].

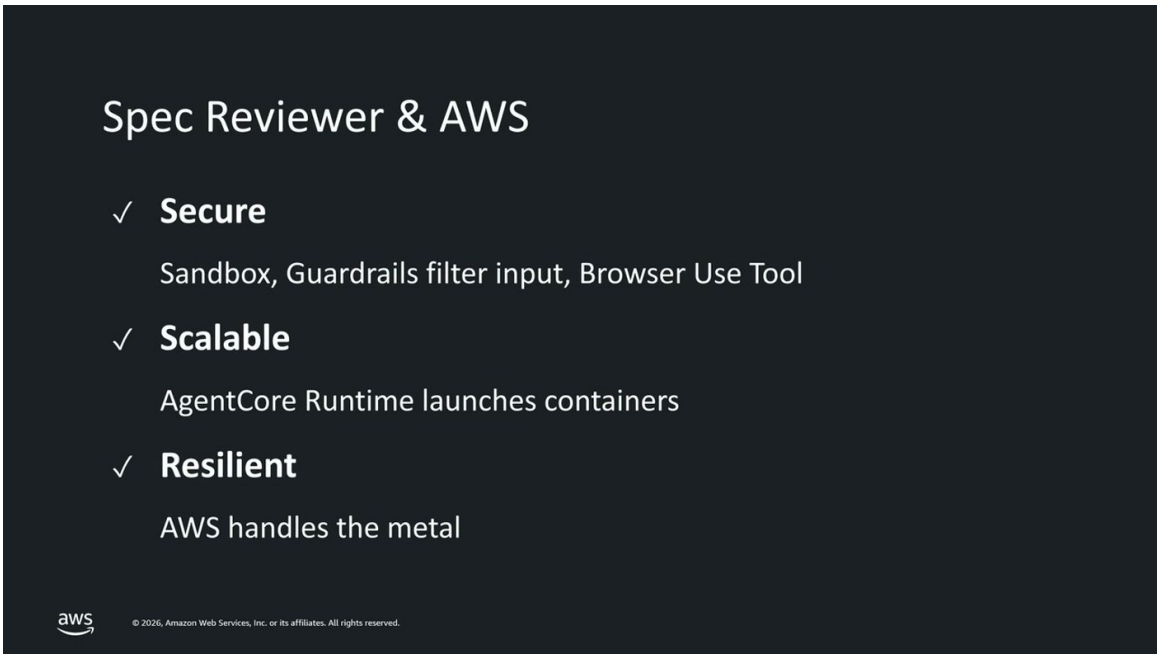


הארכיטקטורה שנבנתה בתגובה: חילוץ דרישות, ולידציה לכל דרישה בנפרד, Browser Use Tool וקונסולידציה.

הפתרון לכשל הראשון היה הפרדת סוכנים: אחד עם גישה לכלי Jira ו-Figma כדי להבין מה רצו להשיג, והשני עם גישה ל-GitHub ולדפדפן — "וכך הוא יכול רק לבדוק בלי שהוא יוכל לשנות את הרקווירמנט" [32:13]. הפתרון לכשל השני היה פיצול המשימה: "האג'נט הראשון בעצם מייצר רשימת דרישות, ואז רצים 20 אג'נטים, כל אחד בודק רק את הדרישה שלו. אז גם הקונטקסט יותר קטן וגם הבלבול פוחט" [32:13] — ומכאן גם הנימוק לסקייל אלסטי: "אני משלם בסוף כמה שאני משתמש, לא צריך לתחזק אינסטנסים" [33:46].

ארבע שכבות האבטחה שהוצגו

- **כל קלט חיצוני עובר guardrails:** "כל האינפוטים שמגיעים מבחוץ, ג'ירה, גיטה, פיגמה וכו, זה הכל חשוד חברים, זה הכל וירוסים" — נגד prompt injection, כולל טקסט בפונט בלתי נראה [33:46].
- **credentials זמניים ומוגבלי רשת:** מי שמצליח לחלץ אותם מקבל גישה זמנית ל-Bedrock הזמינה רק מתוך ה-VPC [33:46].
- **Browser Use Tool במקום דפדפן מקומי:** "הוא מלוכלך אבל הוא נזרק בסוף הסשן והסוכן שלי נשאר נקי" [33:46].
- **Sandbox ברמת ה-Runtime:** סביבה מנותקת שלא מכירה ריפוזיטוריים אחרים, לקוחות אחרים או משימות אחרות [33:46].



סיכום התרומה של השירותים לשלוש התכונות שנדרשו: אבטחה, סקייל ועמידות.

התוצאה הוצגה על דוגמה אמיתית מה-GitHub של החברה: מתוך הדרישות שנגזרו, שתיים לא מומשו בפיצ'ר. הטענה לגבי אופן ההצגה הייתה שהיא מה שמייצר שינוי בפועל — צילום מסך של האפליקציה עם ריבועים אדומים על מה שפוספס: "אני לא מכיר מפתח שיודע להתווכח עם כזאת קומנט" [35:17]. סיכום הרווח התשתיתי היה חד: "אני לא צריך קוברנטיס קלסטר אני לא צריך כלום, AWS דואגים לי להכל ואני רץ כמעט [35:17] "serverless.

החלק נחתם בתזה על מחזור חיי הפיתוח: ב-2025 הוא ideation, planning, building, validating; ב-2026 ה-AI השתלט על ה-building — "אף אחד כבר לא כותב קוד" [36:47] — והכיוון הוא אוטומציה גם של ה-planning וגם של ה-validation, כדי שמפתחים, ראשי צוותים וארכיטקטים יתרכזו ב-ideation וביצירתיות.

13. מה לוקחים מכאן

הסיכום שנאמר מהבמה קצר וממצה: "ראינו שאפשר to build for scale, run time. ראינו שאנחנו עושים secure by design, identity, run time. ראינו שאנחנו תמיד רוצים גברם, למי מותר להזמין טיסות, למי אסור... ראינו לזכור שהסוכן יכיר אותנו, AgentCore Memory, וכמובן [36:47] "Observability and Evaluation.

- **המיפוי פער→רכיב הוא הצ'ק-ליסט המעשי מהסשן.** MCP מוטמע → Gateway; פעולה בשם המשתמש → Identity; latency לא מוסבר → Observability; איכות בקנה מידה → Runtime → localhost; Evaluation; שכחה בין sessions → Memory.
- **הפרדת סוכנים היא בקרה, לא רק ארכיטקטורה.** אצל Baz ההפרדה בין הסוכן שמייצר דרישות לזה שמוודא אותן היא מה שמנע מהמערכת לאשר את עצמה — שיקול רלוונטי לכל מערכת ביקורת אוטומטית.
- **פיצול לפי יחידת עבודה פותר גם דיוק וגם עלות.** ולידציה per requirement מקטינה קונטקסט, מפחיתה טעויות, ובמקביל מתיישבת עם תמחור לפי זמן CPU בפועל.
- **כל קלט ממערכת חיצונית הוא משטח תקיפה.** Jira, Figma ו-GitHub הוגדרו מפורשות כמקורות לא מהימנים שדורשים guardrails ו-credentials זמניים.
- **Evaluation שווה משהו רק כשהוא כמותי ומנומק.** ציון מאפשר thresholds ואלרטים; ההסבר המילולי הוא מה שמאפשר לתקן.
- **ה-Harness משנה את סף הכניסה, לא את הארכיטקטורה.** מי שרוצה להתחיל בלי לכתוב קוד אורקסטרציה יכול; הרכיבים והשיקולים שמאחוריהם נשארים אותם רכיבים.

14. מילון מונחים

מונח	מה זה, כפי שהוצג בסשן
AgentCore Gateway	רכיב שמנתק את ה-MCP-ים מקוד הסוכן וחושף אותם כ-Single MCP Endpoint; עוטף גם Lambda, API Gateway ו-REST API, ותומך ב-semantic search על הכלים.
AgentCore Identity	ניהול זהות והרשאות של הסוכן כמטא-דאטה — API keys או OAuth (Cognito, Okta) — במקום מימוש עצמי.
AgentCore Observability	טלמטריה ברמת ה-iteration האג'נטי: tokens, latency, ה-prompt שנשלח והכלים שהופעלו.
AgentCore Evaluation	ציון כמותי ברמות session, trace וכלי, בתוספת הסבר מילולי; בסיס ל-thresholds, אלרטים ודשבורדים.
AgentCore Runtime	compute מנוהל ו-serverless לסוכנים, עם בידוד session מלא לכל הרצה.
AgentCore Memory	זיכרון קצר טווח (השיחה) וארוך טווח (סיכומים, העדפות ועובדות), מבודד לפי actor ID.
AgentCore Harness	הרכבת סוכן מהקונסול — מודל, system prompt, memory ו-tools — ללא קוד תשתית או אורקסטרציה.
MCP	פרוטוקול פתוח שמסדיר איך הסוכן מתקשר עם הכלים שלו; מאפשר גילוי כלים דינמי מול שרת.
Knowledge Base	אפליקציית RAG מנוהלת ב-Bedrock: ingestion (chunking, embeddings, vector store) ושליפה.
Agentic Retrieval	שליפה שבה הסוכן מפרק את השאלה, מתכנן, שולף, מעריך ומתכנן מחדש עד שיש לו מספיק קונטקסט.
Spec-Driven Review	המקבילה ל-Spec-Driven Development: ביקורת קוד מול הדרישות שנגזרו מהטיקט והעיצוב, לא מול הקוד עצמו.
Browser Use Tool	דפדפן מנוהל בקונטיינר חד-פעמי, כדי שקוד מאתרים חיצוניים לא ירוץ על מכונת הסוכן.