מעבר ל-Vibe Coding: חמישה חוקים לעבודה עם Coding Agents

AWS Summit Tel Aviv 2026 ,סיכום סשן — TLV-DVT301

Omer Tamary, Prototype Architects, AWS | Roy Miara, Member of Technical Staff, Tenzai-ו Oz Altagar

10 בספטמבר 2026 | משך: 34 דקות | הופק מהקלטת הסשן

סיכום מאת קובי אלמוג

על המסמך הזה

המסמך מסכם את הסשן TLV-DVT301 מתוך AWS Summit Tel Aviv 2026, במסלול "AWS for Developers in the AI Era". הוא נבנה מתמלול אוטומטי של ההקלטה המלאה ומהסליידים שחולצו מהווידאו עצמו, כך שאפשר לחזור לתוכן בלי לצפות שוב בשלושים וארבע הדקות. חותמות הזמן בגוף המסמך מפנות לנקודה המדויקת בהקלטה.

הסשן הועבר בעברית על ידי שני Prototype Architects מצוות הפרוטוטייפינג של AWS, ובחלקו האחרון הצטרף אליהם לקוח — Roy Miara מחברת Tenzai — שהציג מקרה בוחן על יישום אותם חוקים במערכת ייצור.

איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. מה נטען בסשן, ומה ישים כבר מחר.
- **פרק 2 — המסגרת:** למה vibe coding נשבר בפרודקשן, ומה בדיוק נמצא בתוך חלון הקונטקסט.
- **פרקים 3–7 — חמשת החוקים:** Rules/Skills/MCP, הפרד ומשול, LSP, Hooks, ו-Memory. כל פרק נפתח בסלייד שלו.
- **פרק 8 — מקרה בוחן Tenzai:** ארבע איטרציות על אותה מערכת, מסקיל אחד ועד צינור דטרמיניסטי.
- **פרק 9 — מה לוקחים מכאן:** מסקנות, ומילון מונחים בסוף.

הערת מקור המסמך הופק מתמלול אוטומטי של ההקלטה. התמלול מדויק בקטעים המקצועיים, אך מונחים לועזיים חוזרים בו בשיבוש פונטי ("קירו" = Kiro, "רולס" = Rules, "הוקים" = Hooks) — הם נורמלו לכתיב האנגלי מול הסליידים. הציטוטים אומתו מול ההקלטה בחותמת הזמן המצוינת ומובאים כלשונם, כולל שיבושי הקלטה קלים. מספרים שמופיעים על הסליידים צוינו ככאלה.

סלייד הפתיחה: קוד הסשן, הכותרת ושלושת הדוברים — שני Prototype Architects מ-AWS ונציג Tenzai.

1. תקציר מנהלים

הסשן לא ניסה לשכנע שכלי coding agents עובדים. הוא יצא מנקודת הנחה שהם עובדים מצוין ל-MVP ונשברים בפרודקשן, והציג חמישה חוקים תפעוליים שצוות הפרוטוטייפינג של AWS גיבש אחרי אלף שעות ומעלה מול CLI ו-IDE. החוט המקשר בין כל החמישה הוא משאב אחד: הקונטקסט.

- **הפער בין POC לפרודקשן הוא פער קונטקסט, לא פער מודל.** הדובר מנה פערי ידע, סטנדרטים ארגוניים, התעלמות ממקרי קצה וולידציה חלשה, ותמצת: "המילה היא קונטקסט. בעצם איזה אינפורמציה אנחנו חושפים למודל שלנו" [1:37].
- **לכל שכבה יש תפקיד — ואסור להעמיס אותה.** Rules = מה חייב לקרות, Skills = איך עושים את זה, MCP = ביצוע בפועל. האזהרה החוזרת: "[6:32] "don't overload one layer, עם דוגמה לאג'נט שמחובר ל-20 MCPs שלא בשימוש.
- **עבודה סדרתית היא בזבוז כפול.** במקום להריץ משימה אחרי משימה, בונים גרף תלויות, מזהים גלים של משימות בלתי-תלויות ומפצלים אותן לסאב-אג'נטים: "הקודינג אייג'נט שלנו נהיה מנהל עבודה" [9:37]. הודגם כפיצ'ר מובנה ב-Kiro.
- **grep הוא כלי חיפוש, לא כלי הבנת קוד.** שאלת "מה מושפע משינוי שם משתנה" נענית ב-grep בפולס-פוזיטיבס ובטוקנים מבוזבזים; LSP — אותו פרוטוקול שמריץ את ה-IDE כבר עשור — עונה עליה דרך גרף קשרים: "והכלי הזה נקרא [12:46] "LSP.
- **מה שדטרמיניסטי צריך להישאר דטרמיניסטי.** לינטינג, פורמטינג, טסטים וסריקות אבטחה לא צריכים לרוץ דרך המודל. ההבדל מהעבר הוא מי מחזיק בהגה: "היום אתם כבר לא בכיסא הנהג" [17:22] — ולכן מחברים אותם כ-Hooks לאירועים באג'נט.
- **ידע שלא נשמר נשרף מחדש בכל סשן.** החוק החמישי הוא agentic memory: מה שומרים (מחקר, באגים שנפתרו, דפוסים חוזרים) ואיפה — מקבצים ותיקיות, דרך Skills, ועד Vector DB ו-Graph DB.

רלוונטיות לארגון

שלושה מתוך חמשת החוקים אינם דורשים שום רכש או שינוי ארכיטקטורה: קובץ Rules ארגוני אחיד, חיבור LSP לכלי הפיתוח, ו-Hooks שמריצים לינט וטסטים על אירועים. שניים הנותרים — פיצול לסאב-אג'נטים ו-memory משותף — דורשים החלטה על מודל עבודה. הדוברים הדגישו במפורש שכל החוקים אינם קשורים לכלי ספציפי: "הוא לא ספציפי לקירו, הוא לא ספציפי לקלוד קוד, הוא לא ספציפי לקודקס" [31:20]. הרכיב היחיד שהוצג כשירות AWS הוא Amazon Bedrock, כמקור מודלים בתוך חשבון הלקוח.

מפת הסשן

זמן	נושא	דובר
0:00	Vibe Coding Limits — איפה נשבר הפער לפרודקשן	Omer
1:37	קונטקסט: הגדרה, מה נמצא בחלון, Context Rot ו-Compaction	Omer
4:59	חוק 1 — Rules, Skills, MCP	Omer
8:05	חוק 2 — הפרד ומשול: גרף תלויות וסאב-אג'נטים	Omer
11:13	חוק 3 — LSP במקום grep	Oz
15:50	איכות, כלים דטרמיניסטיים, וחוק 4 — Hooks	Oz
18:58	חוק 5 — Agentic Memory: מה, איך ואיפה	Oz
22:03	מקרה בוחן Tenzai — Guardrails ו-Observability	Roy
29:47	סיכום, Amazon Bedrock וצעדים הבאים	Oz

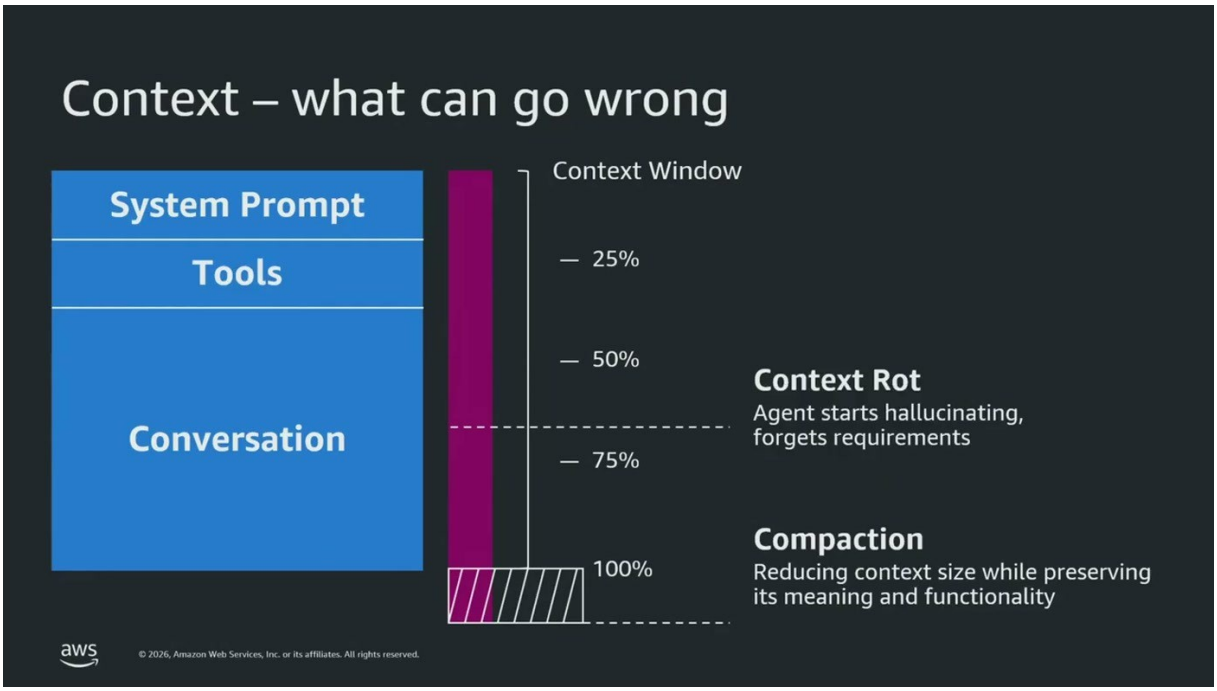
2. המסגרת: הפער הוא קונטקסט

הפתיחה הגדירה את גבולות ה-vibe coding בצורה ישירה. למוקאפים של דשבורדים, לדפי נחיתה ולכלי עזר לוקאליים — הכלים עובדים מצוין. הבעיה מתחילה במקום אחר: "ברגע שאני מדבר על עולמות הפרודקשן, לדוגמה איך אני עושה סקיוריטי, איך אני עושה פרפורמנס טוב, איך אני מדלוור בצורה יעילה, תהליכי [0:00] "CI, CD — שם, לדבריו, מתגלה הפער בין POC לבין production-grade solution.

הסיבות שמנה: הקודינג אג'נט לא יודע באילו חבילות הצוות משתמש, לא מכיר את הסטנדרט הארגוני, מתעלם ממקרי קצה, והוולידציה שלו בינונית. כל אלה, לטענתו, הם ביטויים של גורם אחד.

> **Omer Tamary, [1:37] —** אם אני רוצה לקחת את כל הדברים האלה... ואנחנו רוצים לתמצת את זה למילה אחת... המילה היא קונטקסט. בעצם איזה אינפורמציה אנחנו חושפים למודל שלנו.

מכאן עבר הדובר להגדרה מדויקת. קונטקסט הוא ה-user input וה-LLM output יחד, והם ממלאים את ה-context window — כמות הטוקנים המקסימלית שהמודל יכול להחזיק ברגע נתון. הוא ציין שיש מודלים עם 128,000 טוקנים ומודלים עם חלון של מיליון טוקן [1:37]. בתוך החלון הזה יושבים System Prompt (ההנחיות לאג'נט), Tools (אילו פעולות הוא יכול לבצע — קריאת קובץ, הרצת פקודה), ה-User Input, ולצידם ה-Tool Calls, ה-Tool Results והפלט של האג'נט [3:09].



התפלגות חלון הקונטקסט: System Prompt, Tools ושיחה — עם שני קווי השבר המסומנים בצד ימין.

הסלייד הזה נשא את הטענה המרכזית של החלק הראשון. בסביבות שלושת רבעי החלון מתחיל מה שהוא כינה בשמו המקובל — התופעה שבה איכות הפלט יורדת, האג'נט מתחיל להזות ולהתעלם מהנחיות:

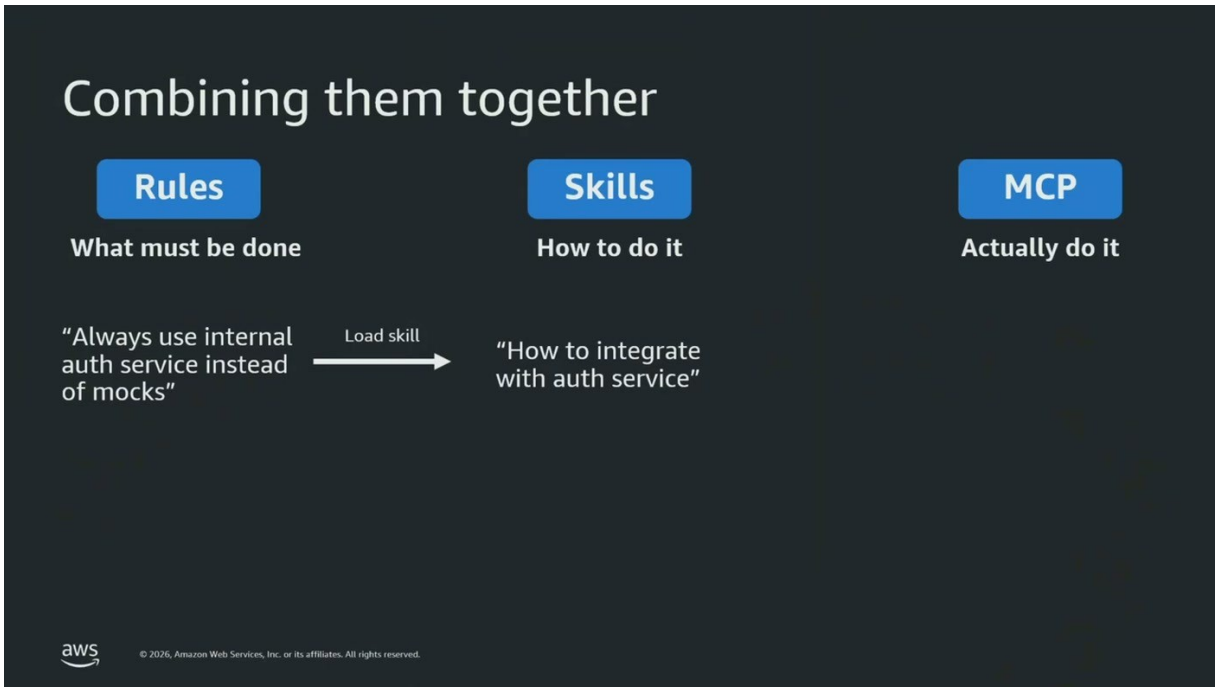
> **Omer Tamary, [3:09] —** האג'נט מתחיל, שיהיה לו hallucinations, הוא מתחיל להתעלם מההנחיות, זו תופעה ידועה שנקראת Context Rot, שאנחנו רואים שהאיכות של התוצרים, של הקודינג אג'נט שלנו, פשוט נהיית פחות טובה.

השלב הבא הוא מיצוי מלא של החלון, ואז נכנס מנגנון ה-Compaction — ההארנס מסכם את מה שנעשה עד כה כדי שיוכל להמשיך לעבוד. ההערכה שניתנה לו מהבמה הייתה מפוכחת: "אבל בינינו, זה לא תמיד עובד כזה טוב" [3:09]. מכאן הגיע המעבר המושגי שממסגר את כל השאר — מ-prompt engineering של 2022–2023 ל-context engineering: איך חושפים לאג'נט ב-long running session את המידע הרלוונטי, tools, skills ו-memories [3:09].

3. חוק ראשון — Rules, Skills ו-MCP

שלוש אבני הבניין שהופכות coding agent גנרי לאג'נט שמתאים לצרכים של הצוות. הטרמינולוגיה שהוצגה:

- **Rules —** "a read me for agents, a do's and don'ts" [4:59]. הדברים שרוצים שיופיעו בכל איטרציה ובכל משן: code styling, סטנדרטים. נטענים בתחילת הסשן ונמצאים שם תמיד.
- **Skills —** חבילות של יכולות שאורזים מראש, שיכולות להכיל קבצי Markdown, סקריפטים ו-workflows. נטענים במנגנון progressive disclosure: לקונטקסט נכנס רק ה-skill description, ורק אם האג'נט בוחר להשתמש בו הוא טוען את כל הקונטקסט הרלוונטי [4:59–6:32].
- **MCP —** הסטנדרט שמאפשר לחבר כלים חיצוניים לאג'נט. נטענים כמו כל שאר הכלים של האג'נט [6:32].



שילוב שלוש השכבות על דוגמה אחת: מהחוק הארגוני, דרך ה-skill שנטען, ועד הקריאה בפועל.

הדוגמה שעל הסלייד עברה את שלוש השכבות בשורה אחת. הכלל: "תמיד תשתמש באותנטיקציה סרוויסס ואל תשתמש במוקים". ה-skill: איך עושים אינטגרציה לאותו שירות אותנטיקציה — קובץ Markdown, סקריפטים וכדומה. ה-MCP: הביצוע עצמו, "איך אני עושה retrieve לטוקנים מה-internal API" [6:32].

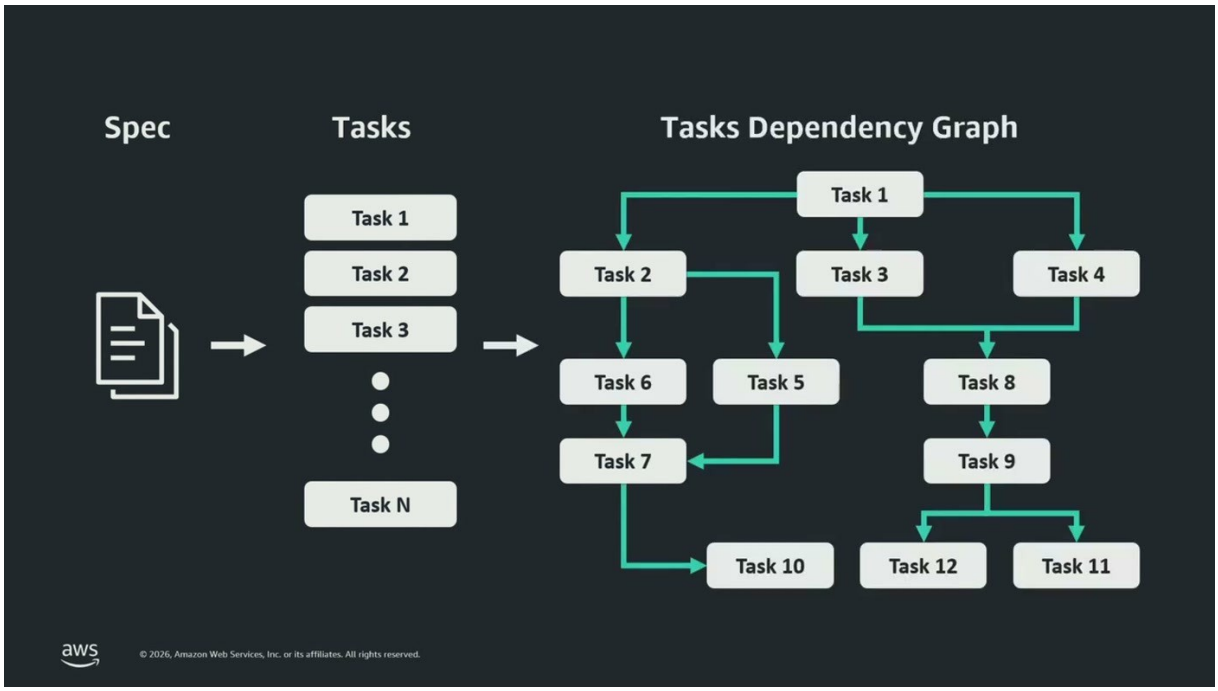
— rules **Omer Tamary, [6:32]** זה what must be done. מה חייב לעשות?... אחרי שיש לנו את הרול הזה, אנחנו בעצם עוברים להסקיל, זה בעצם איך לעשות את זה... MCP זה ממש ביצוע פעולה.

מלכודת האזהרה שנמסרה כנקודה "סופר קריטית": "don't overload one layer" [6:32]. שתי דוגמאות הכשל שהובאו — אג'נט אחד שמחובר ל-20 MCPs שונים שאיש לא משתמש בהם ורק מעמיסים על הקונטקסט, ו-Rules שסותרים זה את זה עד שהאג'נט לא יודע מה לעשות. לפני שמוסיפים משהו לאג'נט, צריך להחליט באיזו שכבה הוא עושה סדר במקום בלאגן.

4. חוק שני — הפרד ומשול

נקודת המוצא לחוק השני היא אג'נט שכבר מכיר את הסטנדרטים, מחזיק סט יכולות ומחובר ב-MCP לכלים הפנימיים. עכשיו בונים ממנו תוכנית עבודה. הדובר הגדיר spec כסט מסמכים שמגדיר מה האג'נט צריך לעשות — טכנית, פרודקטית, user journey — ומתוך ה-specs גוזרים משימות: "אפשר לחשוב על זה כמו טיקטים בג'ירה" [8:05].

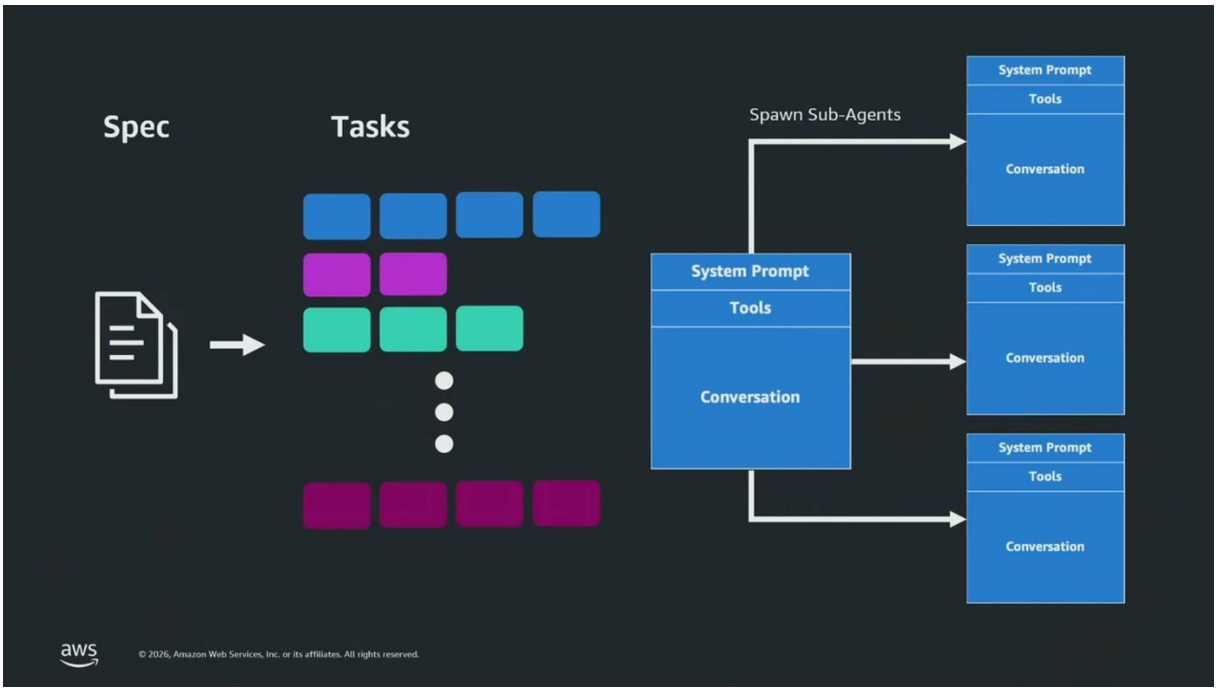
וכאן הוא הציג את שתי התקלות שצצות כשמריצים את המשימות כלשונן, אחת אחרי השנייה. הראשונה היא זמן: המשימה הראשונה לקחה שמונה דקות, השנייה שש, וכן הלאה — "רוב הזמן אנחנו פשוט בוהים במסך" [8:05]. השנייה חמורה יותר: שלוש המשימות הראשונות נגעו ל-API endpoint, הרביעית הייתה בכלל בפרונטאנד — והקונטקסט של האג'נט עדיין מלא בכל מה שקדם. "זה סתם יכול לבלבל אותו, זה סתם עולה לנו טוקנים, זמן וכסף" [8:05].



במקום רשימה שטוחה: גרף תלויות בין המשימות שנגזרו מה-spec, כולל קשרי הקדימות.

הפתרון הוא לא לצאת לביצוע מיד. קודם בונים גרף תלויות — בהינתן משימה, באילו משימות היא תלויה ומי תלוי בה. מהגרף הזה גוזרים שלבים, או "גלים", של משימות בלתי-תלויות. זה השינוי התפקידי שמאפשר את הכול:

> **Omer Tamary, [9:37] —** במקום לעבוד משימה משימה, הקודינג אייג'נט שלנו נהיה מנהל עבודה. בכל שלב הוא לוקח סט של משימות שבלתי תלויות ומטריגר סאב אייג'נטים, כל אחד עם המשימה הרלוונטית שלו, ועם הקונטקסט הרלוונטי שלו.

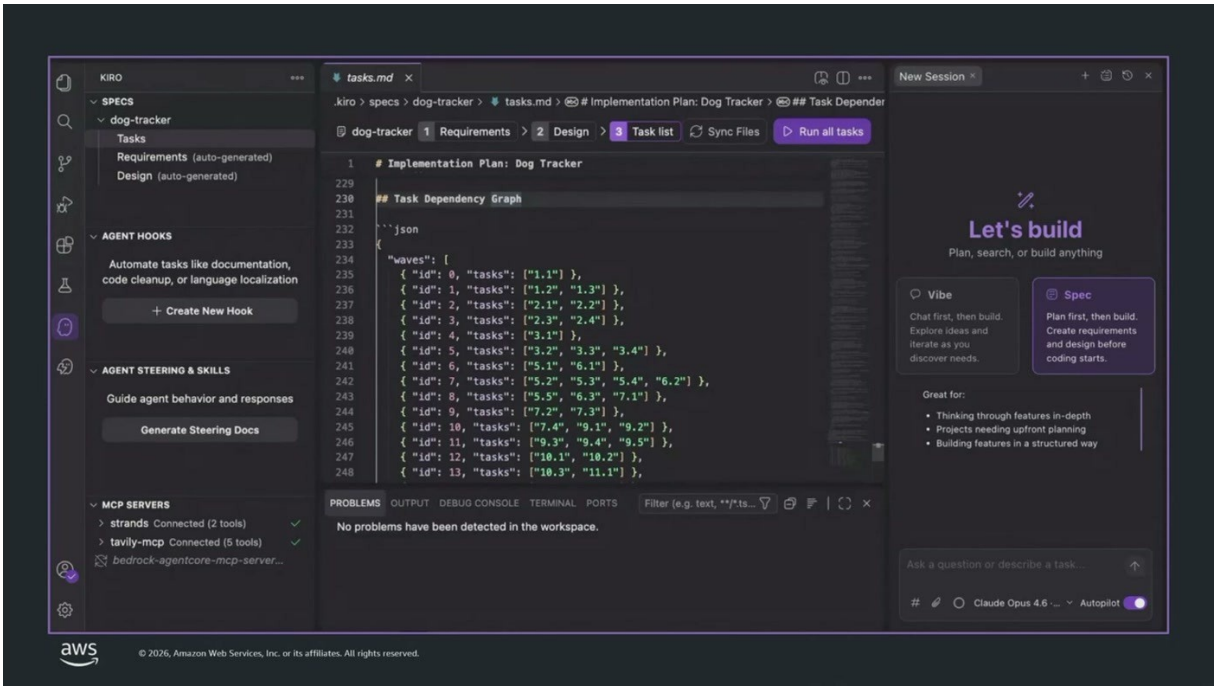


גלי המשימות הצבועים לפי שלב, והאג'נט הראשי שמפצל כל גל לסאב-אג'נטים עם System Prompt, Tools ושיחה נפרדים.

שני הרווחים מצטברים: הקונטקסט של כל סאב-אג'נט ממוקד במשימה אחת בלבד, והזמן הכולל להרצת אותו סט משימות קטן משמעותית מהריצה הסדרתית [9:37].

איך זה נראה בפועל — Kiro

הדגמת המסך נעשתה על Kiro, שתואר כ-"הקודינג אייג'נט של אמזון שתומך ב-spec driven development" [9:37]. בצד המסך מופיעים ה-Requirements, ה-Design וה-Tasks; בתוך הטסקים כל משימה מוגדרת במדויק, ובתחתית העמוד יושב גרף התלויות — "אנחנו בעצם מקבלים כבילט-אין נייטיבלי בקירו, יצירה של גרף תלויות שמאפשר לנו, במקרה הזה, לרוץ לפי גלים" [9:37].



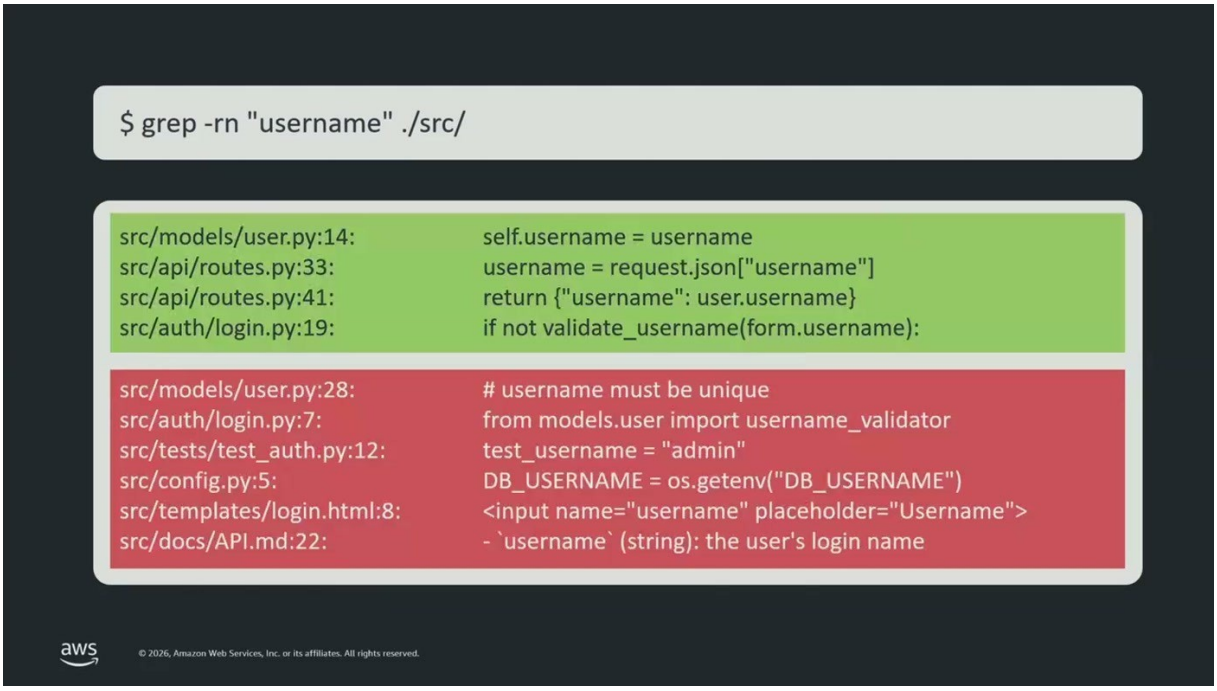
מסך Kiro בדמו: פאנל ה-Specs מימין, ולצידו קובץ התלויות שנוצר אוטומטית לכל משימה.

מבחינת הפעולה של המשתמש, הטענה הייתה שזה כפתור אחד: "פשוט ללחוץ על כפתור, run all tasks... בכל שלב הוא מטריגר את סט המשימות הבלתי-תלויות לסאב-אג'נטים ומנהל אותם" [11:13].

5. חוק שלישי — LSP במקום grep

החלק השני של הסשן נפתח בשאלה לקהל על טוקנים שנגמרים, ועבר מיד לנושא שהוגדר ככואב ביותר: code understanding. הדוגמה שנבחרה היא יומיומית עד כדי בנאליות — שינוי שם משתנה מ-username ל-user_name. השאלה שנשאלה: "מה צריך להשתנות כתוצאה מזה? מהן בעצם ההשפעות של השינוי הזה?" [12:46–11:13].

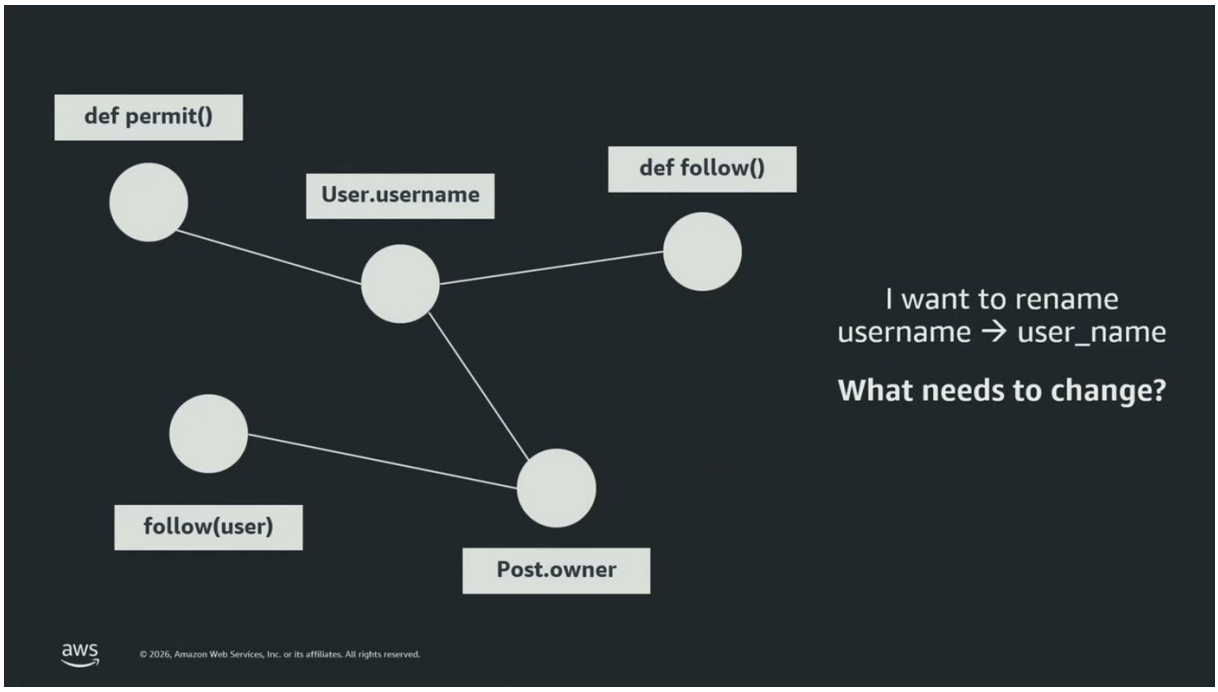
מי שיסתכל מה הכלי עושה מאחורי הקלעים יראה, לדברי הדובר, קריאה ל-grep" — grep לצורך העניין עושה קונטרול F, מחפש את הטקסט בקבצים ומחזיר לו תשובות" [12:46]. התשובות האלה מכילות גם מידע לא רלוונטי: קבצי דיזיין, תיעוד, טסטים — לא מעט false positives.



פלט grep על "username" מסומן בשני צבעים: ההתאמות שבאמת רלוונטיות מול הרעש שנכנס לקונטקסט.

> **Oz Altagar, [12:46] —** במילה אחת... מדובר על wasted context, אותו משאב יקר שאנחנו כל כך רוצים לדאוג לו, פשוט הולך לפח בלי שאנחנו יודעים.

הטענה הבאה הייתה הטענה המעניינת בפרק: זו בעיה פתורה, ופתרנו אותה מזמן. עשר שנים לוחצים right click ב-IDE ומקבלים view references או קפיצה ישירה למימוש. התחושה שהובעה מהבמה היא שבמעבר לכלים החדשים "איבדנו פיצ'רים וכלים כל כך משמעותיים ויעילים שעברנו איתם עכשיו 10 שנה, 15 שנה" [12:46]. הכלי שמריץ את אותם IDE-ים מאחורי הקלעים הוא LSP — והוא, לדבריו, לא מאופשר by default בהרבה מהכלים האגנטיים, כך שצריך לחבר אותו ידנית.



אותה שאלת rename, מוצגת כגרף קשרים בין הסימבולים במקום כרשימת התאמות טקסט.

מה ש-LSP עושה, בתיאור שניתן, הוא לפרק את הקוד ולארגן אותו כגרף קשרים — מי משתמש במי ומי משפיע על מי. שאלות כמו שינוי שם משתנה נענות באופן מיידי, בלי לפתוח קבצים ובלי לחפש בתוכם [14:17]. ארבעת הפרמטרים שנמנו כרווח:

- **מהירות —** התשובה מגיעה ישירות, במקום חיפוש תורי קובץ אחרי קובץ.
- **קונטקסט —** לא מכניסים לחלון מידע לא רלוונטי.
- **עלות —** "קונטקסט שווה שווה טוקנים. אני משלם פחות על טוקנים, אני גם מבזבז פחות טוקנים" [14:17].
- **Correctness —** "אני לא מבלבל את המודל עם מידע שהוא לא צריך" [14:17].

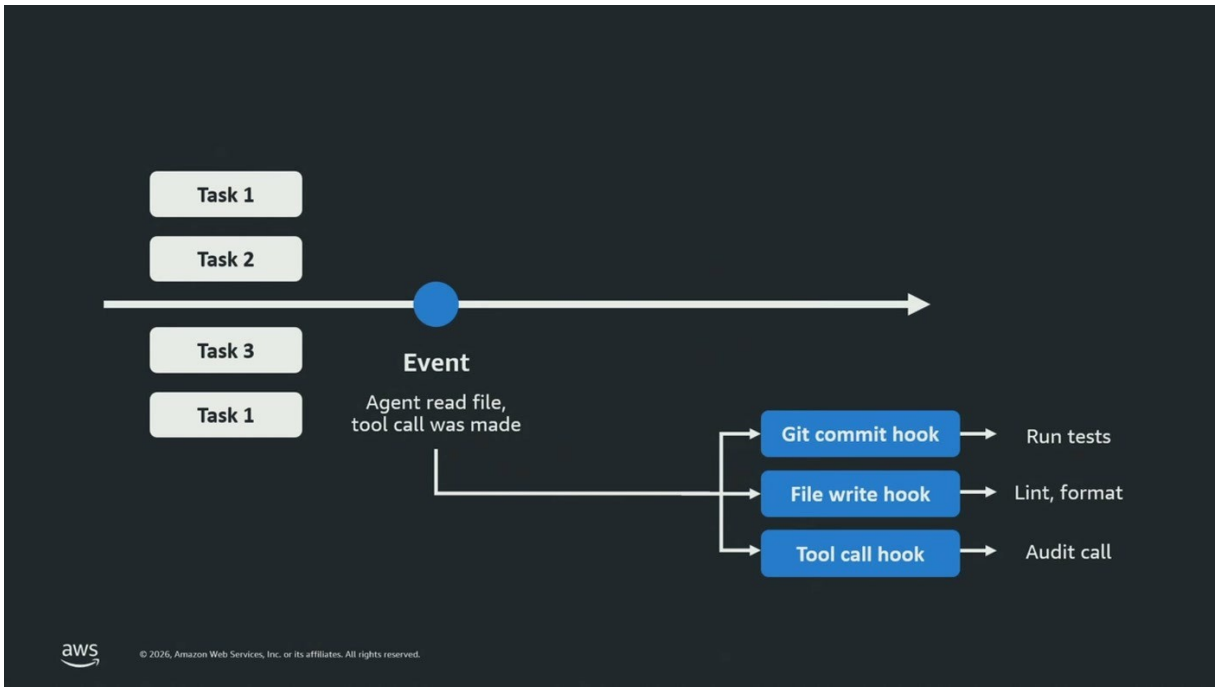
6. חוק רביעי — Hooks ודטרמיניזם

המעבר לחוק הרביעי נעשה דרך שאלה: ארבעת הפרמטרים לעיל מאפשרים לרוץ מהר, "לראות 10x, 100x מהפיתוח הנוכחי שלכם, אבל האם זה גם איכותי? איך אנחנו בכלל מגדירים איכות?" [14:17]. הסלייד פרש מימדים — readability, correctness, security ועוד — וחילק את הכלים שמודדים אותם לשתי משפחות: כלים שדורשים בן אנוש בתהליך (code review של חבר צוות), וכלים דטרמיניסטיים. הדגש הושם על הצד הימני: linting, formatting, type checking, security scanning לאיתור סודות בקוד או ספריות פגיעות, ובעיקר טסטים — TDD ו-Red-Green-Refactor, שלטענתו "משמעותיים מתמיד" [15:50–17:22].

וכאן הגיעה ההתנגדות הצפויה מהקהל — את זה אנחנו עושים כבר שנים. התשובה הגיעה בשתי מכות. הראשונה, אנלוגיה:

> **Oz Altagar, [17:22] —** הייתם נותנים לבן אדם לחשב מה זה סינוס 42? או 3000 בריבוע? ברור שלא, נכון. יש לנו מחשבון, יש לנו כלי דטרמיניסטי שעושה את זה.

והשנייה, ההסבר למה זה עולה שוב דווקא עכשיו: "היום אתם כבר לא בכיסא הנהג. מי שמוביל היום את התהליכים הוא קלוד, הוא צ'אט ג'י-פי-טי, הוא קירו, הוא הקודינג אייג'נט שאיתו אתם עובדים" [17:22]. כשהאג'נט הוא זה שמחליט מה להריץ ומתי, הדטרמיניזם צריך להיכנס חזרה לתמונה באכיפה — ולא בהסתמכות על שיקול דעתו.



אירועים על ציר הריצה מחוברים לפקודות: git commit מפעיל טסטים, כתיבת קובץ מפעילה lint, קריאת כלי נרשמת ל-audit.

Hooks הם המנגנון שמתחבר לאירועים שונים בעבודת האג'נט — קריאת קובץ, הרצת כלי, commit — ומריץ עליהם פקודות דטרמיניסטיות. הדוגמאות שניתנו: "כשאני קורא לגיט קומיט, תריץ את הטסטים. או בכל טול שאני קורא לו, זרוק את זה באודיט, כדי שאחרי זה, שייפול לי הפרודקשן, אני יכול להבין בדיוק מה קרה" [17:22]. נקודה נוספת שנאמרה: את תוצאת ה-hook אפשר להזרים חזרה למודל, "ולתת לו בעצמו להבין איפה הוא יכול להשתפר ומה הצעדים הבאים שהוא יכול לקחת" [18:58]. בסיכום בסוף הסשן זה נוסח כ"אוטונומיה מפוקחת" [29:47].

7. חוק חמישי — Agentic Memory

החוק החמישי נפתח בסיפור מוכר. מפתח נתקל בשגיאה שקשורה ל-PostgreSQL, יושב עליה כארבעים דקות, מדבג ופותר. למחרת מפתח אחר מהצוות נתקל באותה בעיה בדיוק, יושב עליה חצי שעה — ולא מצליח. "הוא שרף על זה מלא טוקנים, מלא זמן, ופשוט משהו שהיה יכול להיפתר ברגע, אם הייתי משתף איתו את המידע הזה" [18:58].

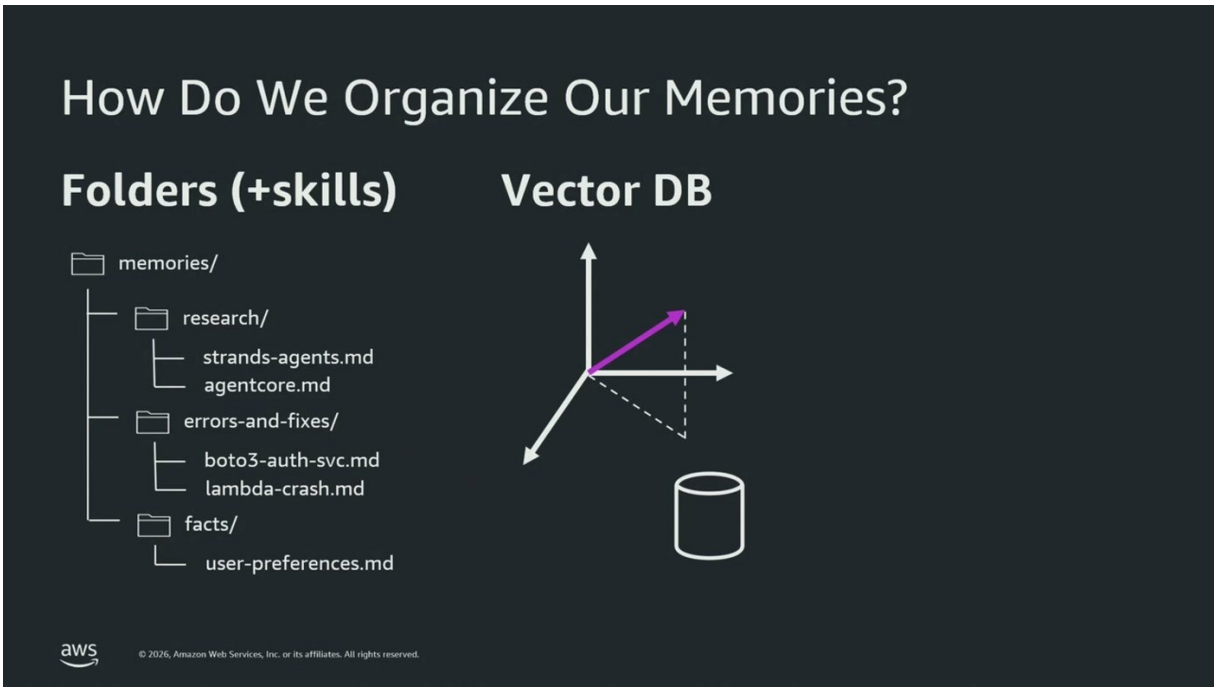
שני סשנים על אותה שגיאת asyncpg: מספר הניסיונות, הזמן והטוקנים שנשרפו בכל אחד — והתוצאה ההפוכה.

הסלייד הצמיד מספרים לסיפור. לפי המוצג על המסך: Session #1 — ארבעה ניסיונות, 42 דקות, 42.3k טוקנים, סומן Fixed; Session #2 — שלושה ניסיונות, 36 דקות, 56.7k טוקנים, סומן Not Fixed. אותה שגיאה, שני סשנים, ורק אחד הגיע לפתרון.

הדובר הקדים תשובה להתנגדות ש"הכלי כבר עושה את זה בשבילי": מי שטרח להסתכל מה באמת נשמר בזיכרון האוטומטי, לדבריו, הבין שזה לא מספיק ולא פותר את הבעיה שבאה לפתור [20:30]. משם הוא פירק את הנושא לשתי שאלות.

מה שומרים

- **מחקר יקר.** "אם אני עושה מחקר על ספרייה חדשה שהאג'נט לא הכיר, ושרפתי על זה 100 אלף טוקנים ושעתיים מהחיים שלי, אין שום סיבה שאני אעשה את זה שוב" [20:30].
- **באגים שכבר נפתרו.** המקרה מהדוגמה — פתרון שנמצא פעם אחת לא צריך להיחקר מחדש.
- **דפוסים חוזרים.** תבניות שכבר קרו עשר פעמים בקוד: "אני לא ארצה להמציא את הגלגל בפעם האחת עשרה" [20:30].



ספקטרום האחסון: מבנה תיקיות מקומי בצד אחד, מרחב וקטורי בצד השני.

השאלה השנייה — איפה שומרים — הוצגה כספקטרום שנע בין מורכבות, מחיר ויכולת גישה. בקצה השמאלי, וגם השימושי והקל ביותר, וגם מה שרוב הכלים עושים by default: קבצים ותיקיות לוקאלית על המחשב, למשל תיקיות research, errors-and-fixes ו-facts. צעד קדימה — Skills. צעד נוסף — Vector DB לחיפושים סמנטיים. ובקצה הרחוק — Graph DB [20:30–22:03].

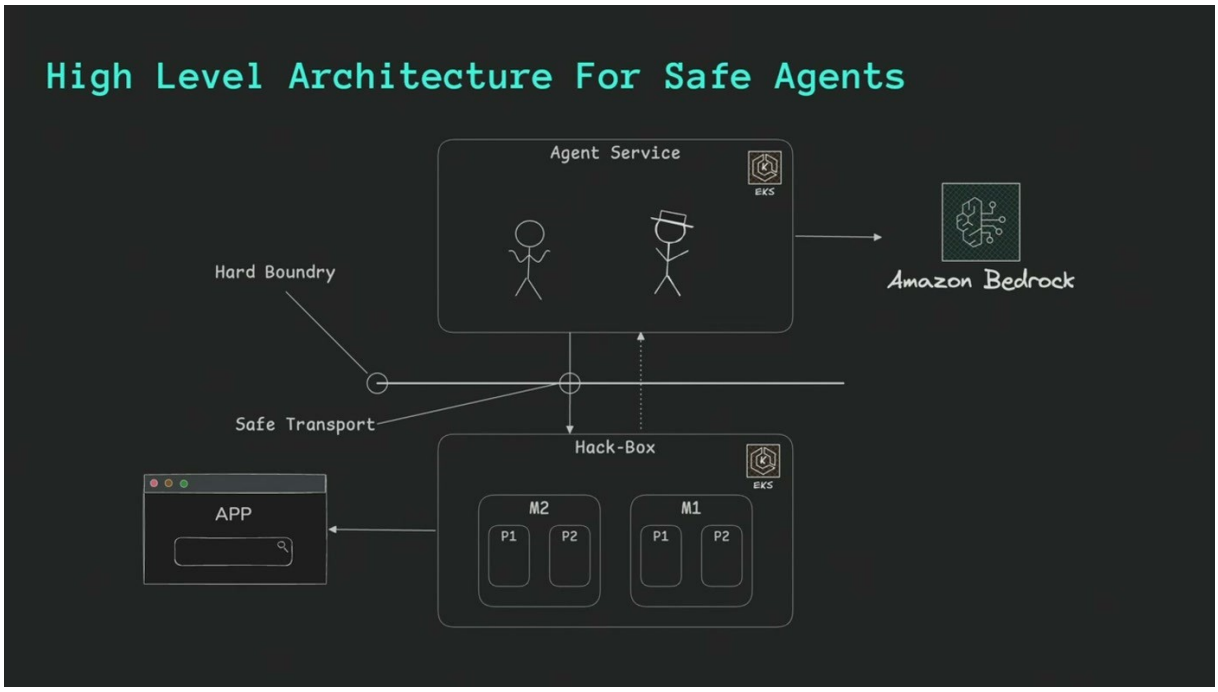
8. מקרה בוחן — Tenzai

החלק האחרון הועבר על ידי Roy Miara מ-Tenzai, שהציג איך חמשת החוקים נראים במערכת ייצור אמיתית. ההגדרה שנתן לחברה: "אנחנו טנזאי, אנחנו חברת AI, אנחנו חברת סייבר, אנחנו מתעסקים בתחום הזה של AI hackers, AI red teaming, penetration testing" [22:03]. הם בונים סוכנים שפורצים למערכות, לאפליקציות ולרשתות — מתוך מטרה להגן.

> **Roy Miara, [23:36]** — מצד אחד, אנחנו צריכים להיות מאוד מאוד מסוכנים... ולהיות מסוגלים להיכנס לתוך המערכות האלה. מצד שני, אנחנו צריכים לא לעשות אף פעם נזק.

מתוך המתח הזה נבחר ה-case study: guardrails. הוא הזכיר בהקשר זה את הסיפור הציבורי על אג'נטים שברחו מהסנדבוקס כדוגמה לכמה הנושא חשוב [23:36]. הטענה שהוא ביקש לשכנע בה את הקהל הייתה ממוקדת:

> **Roy Miara, [23:36]** — AI לקוד זה מגניב, אבל AI לעולמות האלה של טלמטריה, דיבאגינג, ואובזרוובליטי ואבלואציות, זה אפילו פאוור הרבה יותר חזק.



הארכיטקטורה: קלאסטר האג'נטים מול Amazon Bedrock, וה-hack-box מעבר לגבול המופרד, עם התעבורה לאפליקציה דרך נתיב יחיד.

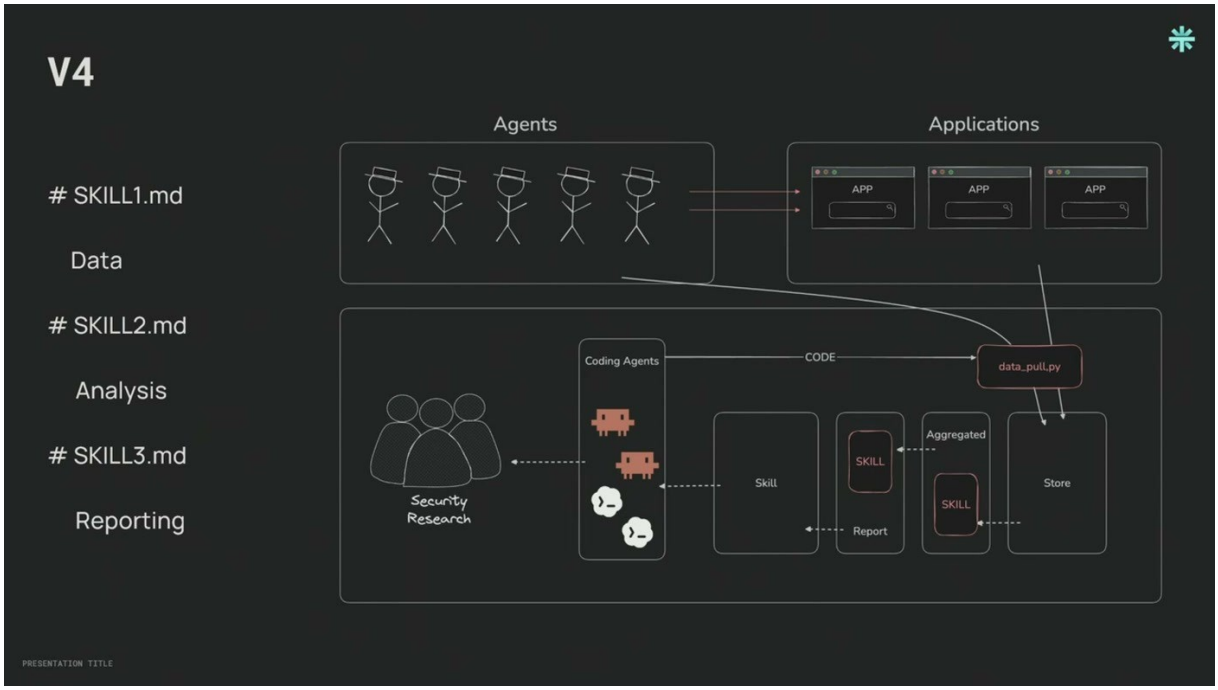
המערכת שתוארה: האג'נטים וההארנס יושבים בקלאסטר משלהם, רצים על EKS וקוראים למודלים ב-Bedrock. מולם יושב סנדבוקס נפרד שהם מכנים hack-box, בסביבה אחרת, עם boundary חזק ושליטה הדוקה על התעבורה — "ודרך שם ורק שם אנחנו ניגשים לאפליקציה" [25:08]. ה-Guardrail Agent יושב בין האג'נט התוקף לבין האפליקציה, מסתכל על כל מה שהאג'נט המרכזי עושה, ושואל שאלה אחת: האם אנחנו עושים עכשיו משהו שיכול לסכן את היוזר שלנו [25:08].

ארבע איטרציות על אותו כלי

הליבה של הקייס היא פרוגרסיה, לא ארכיטקטורה. כדי לפתח את ה-guardrail הם חייבים לאסוף טלמטריה משני האג'נטים, להצליב ביניהם, ולהבין מה קרה ואיך מתקנים.

- **skill — V1 אחד.** סקיל שלוקח את המידע, מעבד אותו ומחזיר תשובות. "זה נהדר, כי קודם כל קורה משהו, אני מקבל תשובות" — אבל לאורך זמן, אותה שאלה מחזירה תשובות שונות [25:08].
- **V2 — סקריפט דטרמיניסטי.** כלי CLI לשליפת המידע. הסקיל עדיין מסביר לאג'נט איך להפעיל, אבל השליפה עצמה כבר דטרמיניסטית: "אני נותן לו מחשבון, אז עכשיו הוא לא צריך לחשב דברים בראש" [26:40].
- **V3 — פירוק לגרף.** שבירת התהליך לחתיכות קטנות: סקיל נוסף לאגרגציה, בנפרד מהשליפה ומההחזרה. האגרגציה נעשית זהה — "אבל אנחנו עדיין מגיעים למסקנות שונות" [26:40].

- **V4 — הפרדת הדיווח.** גם הריפורטינג הפך לתהליך בפני עצמו. שלושה סקילים עם תלות ביניהם, MCP שקורא לדאטה, ו-data pull שמושך את המידע בצורה זהה [26:40].

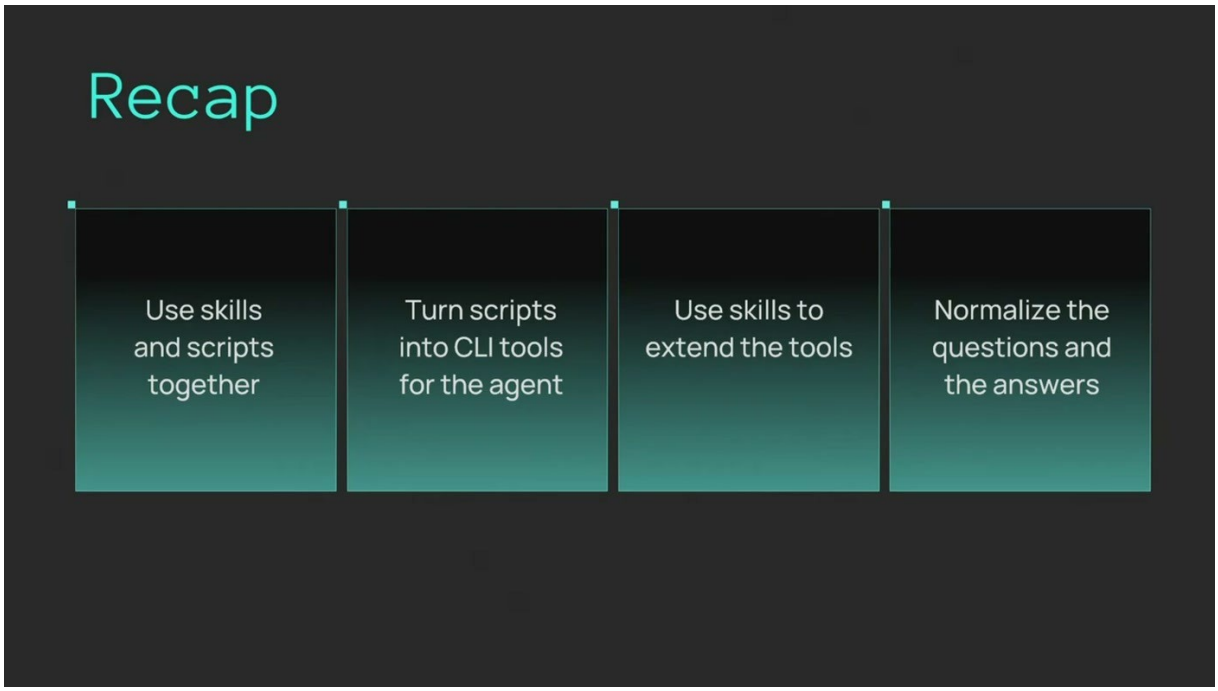


המצב הסופי: שלושה סקילים — Data, Analysis, Reporting — מעל אותו צינור שליפה.

מה שנשאר אחרי V4, לדבריו, היא בעיה אנושית: מהנדסים וחוקרים שעדיין שואלים את אותה שאלה בניסוחים שונים [28:14]. הפתרון שהוצג, מתוך צילומי מסך של הסקילים עצמם, הוא להסביר לאג'נט שני דברים נוספים — מה המשתמש מתכוון כששואל שאלה מסוימת ומה הוא אמור לעשות בתגובה, ומה מצופה ממנו להחזיר: "אנחנו מצפים שלכל טענה שלו יהיה איזה סימוך, יהיה איזשהו ביסוס, תראה לי שורת לוג שזה קרה בה", ושתמיד יבדוק שיש לו מספיק coverage [28:14].

> **Roy Miara, [29:47] —** ובסוף אנחנו רוצים לנרמל, לנרמל גם את השאלות שאנחנו שואלים וגם את התשובות שאנחנו מקבלים.

9. מה לוקחים מכאן



ארבעת הצעדים שסיכם נציג Tenzai: מסקילים, דרך הפיכת סקריפטים לכלי CLI, ועד נרמול השאלות והתשובות.

הסיכום שנמסר מהבמה חזר על החוט: קונטקסט וטרמינולוגיה (ההבדל בין Skill ל-MCP ל-Rules), סאב-אג'נטים ומקביליות "בלי לדרוך אחד לשני על הרגליים", החשיבות של LSP ו"אין שום סיבה לוותר על כל הכלים שחווינו ב-IDE לאורך השנים", Hooks ואוטונומיה מפוקחת, ולבסוף memory כדרך לשפר את האג'נט לאורך זמן [31:20–29:47].

ישים מיד, בלי תלות בכלי

- **לכתוב קובץ Rules ארגוני אחד ולוודא שאין בו סתירות.** זו השכבה הזולה ביותר, והיא נטענת בכל סשן ממילא.
- **לעשות ניקיון MCP.** כל שרת שלא בשימוש פעיל יוצא. הוא עולה טוקנים בכל בקשה, גם כשאיש לא קורא לו.
- **לחבר LSP לכלי הפיתוח האגנטי.** לא מאופשר by default ברוב הכלים; זה שינוי קונפיגורציה בודד עם השפעה ישירה על עלות ועל דיוק.
- **להגדיר שני Hooks ראשונים.** טסטים על commit, ו-audit על כל קריאת כלי. הראשון מגן על האיכות, השני נותן מסלול חקירה כשמשהו נופל בפרודקשן.
- **להתחיל memory מתיקייה, לא ממסד נתונים.** תיקיות research ו-errors-and-fixes בריפו הן הצעד הראשון; Vector DB ו-Graph DB באים אחר כך, אם בכלל.

מה לא נאמר בסשן

מספר ה-"1000+ שעות" שבכותרת לא פורק לנתונים בגוף ההרצאה, ולא הוצגה מדידה השוואתית לפני ואחרי יישום החוקים. המספרים היחידים שהוצגו כמדידה הם אלה שעל סלייד שני הסשנים (42 דקות מול 36 דקות, 42.3k מול 56.7k טוקנים). ההדגמה של גרף התלויות והרצת הגלים נעשתה ב-Kiro בלבד; לא הוצג איך מיישמים את אותו דפוס בכלים אחרים, אף שנאמר שהוא אינו תלוי כלי.

הרכיב מ-AWS

השירות היחיד שהוצג בפירוט בסוף הוא Amazon Bedrock, ולא כחלק מהחוקים אלא כתשובה לשאלה "מי המודל שמריץ את כל הדבר הזה מאחורי הקלעים, מי המוח של התהליך" [31:20]. הנימוקים שניתנו: צריכת מודלים מספקים שונים דרך אותם אג'נטים, בתוך חשבון ה-AWS של הלקוח "בלי שלאף אחד יש גישה או יכולת לשלוט במידע שלכם", קרבה ריג'נלית לצורכי קומפליינס ולטנסי, ותשלום פר-טוקן במקום נעילה למושבים [31:20].

מילון מונחים

מונח	מה זה, כפי שהוצג בסשן
Context Window	כמות הטוקנים המקסימלית שה-LLM יכול להחזיק ברגע נתון. נאמר שיש מודלים עם 128,000 ועד מיליון טוקן.
Context Rot	ירידת איכות הפלט כשהחלון מתמלא — הזיות והתעלמות מהנחיות, בערך משלושת רבעי החלון.
Compaction	סיכום אוטומטי של ההיסטוריה על ידי ההארנס כשהטוקנים נגמרים, כדי לאפשר המשך עבודה.
Context Engineering	יורשו של prompt engineering: אילו tools, skills ו-memories נחשפים לאג'נט לאורך סשן ארוך.
Rules	"README for agents" — do's and don'ts שנטענים בכל סשן. מה חייב לקרות.
Skills	חבילות יכולת (Markdown, סקריפטים, workflows) שנטענות ב-progressive disclosure. איך עושים.
MCP	סטנדרט לחיבור כלים חיצוניים לאג'נט. הביצוע בפועל.
Progressive Disclosure	טעינת ה-skill description בלבד לקונטקסט, והרחבה לתוכן המלא רק כשהאג'נט בוחר להשתמש בו.
Spec	סט מסמכים שמגדיר מה האג'נט צריך לעשות — טכנית, פרודקטית ו-user journey. ממנו נגזרות המשימות.
Tasks Dependency Graph	מיפוי התלויות בין משימות, שממנו נגזרים גלים של משימות בלתי-תלויות שאפשר להריץ במקביל.
LSP	Language Server Protocol — הפרוטוקול שמאחורי ה-IDE. מייצג את הקוד כגרף קשרים ועונה על שאלות references ו-impact ישירות.
Hooks	חיבור פקודות דטרמיניסטיות לאירועים בעבודת האג'נט (commit, כתיבת קובץ, קריאת כלי).
Agentic Memory	שימור ידע מסשן לסשן: מחקר, באגים שנפתרו ודפוסים חוזרים. מתיקיות מקומיות ועד Vector/Graph DB.
Kiro	הקודינג אג'נט של אמזון, שתומך ב-spec driven development וכולל גרף תלויות מובנה והרצת גלים.