להפסיק לכתוב קוד: מרעיון לפרודקשן עם סוכני AI

TLV-DEM306 — סיכום סשן, AWS Summit Tel Aviv 2026

קורל מגן, Technical Account Manager, AWS · דור פיברט, Solutions Architect, AWS

10 בספטמבר 2026 | משך: כ-22 דקות | הופק מהקלטת הסשן

מסלול: AWS Demos · רמה 300

על המסמך הזה

המסמך מסכם את הסשן TLV-DEM306 מתוך AWS Summit Tel Aviv 2026, בהתבסס על ההקלטה המלאה שפורסמה באתר הסאמיט. הסשן כולו הוא דמו חי: שני דוברים מ-AWS לוקחים אפליקציית מיקרו-סרוויסים לדוגמה ומריצים עליה ארבעה מודלים שונים של הפעלת סוכני קוד — מהמחשב הנייד ועד תהליך אוטונומי לחלוטין שמתנעל מאירוע ב-GitHub. המסמך נבנה מתמלול אוטומטי של ההקלטה ומהסליידים ומסכי הדמו שחולצו מהווידאו עצמו, כך שאפשר לחזור לתוכן בלי לצפות שוב. חותמות הזמן בגוף המסמך מפנות לנקודה המדויקת בהקלטה.

איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** שלוש דקות קריאה. מה הוצג ומה המסקנה המעשית.
- **פרקים 2–4 — המסגרת הרעיונית:** למה לא הולכים ישר לקוד, מה זה Agent Fleet, ומה התפקיד של Kiro בתמונה.
- **פרקים 5–9 — הדמו:** אפליקציית הבסיס, ואז ארבעת מודלי ההרצה לפי סדר הצגתם — לוקלי, ענן, fan-out, ואוטונומי.
- **פרק 10 — מה לוקחים מכאן:** המסקנות שהדוברים עצמם ניסחו, ומה שעוד אפשר להסיק.

הערת מקור המסמך הופק מתמלול אוטומטי של ההקלטה. התמלול מדויק בקטעי התוכן, אך מונחים לועזיים ושמות מוצרים מופיעים בו בשיבוש פונטי ("אג'נט קור", "קלוד קוד", "Coding Gadget") — הם נורמלו כאן למונח האנגלי הנכון מול הסליידים ומסכי הדמו. הציטוטים נבדקו מול ההקלטה בחותמת הזמן המצוינת ומובאים כלשונם בתמלול. שמות ותפקידי הדוברים נלקחו מסליד הפתיחה ולא מהתמלול.

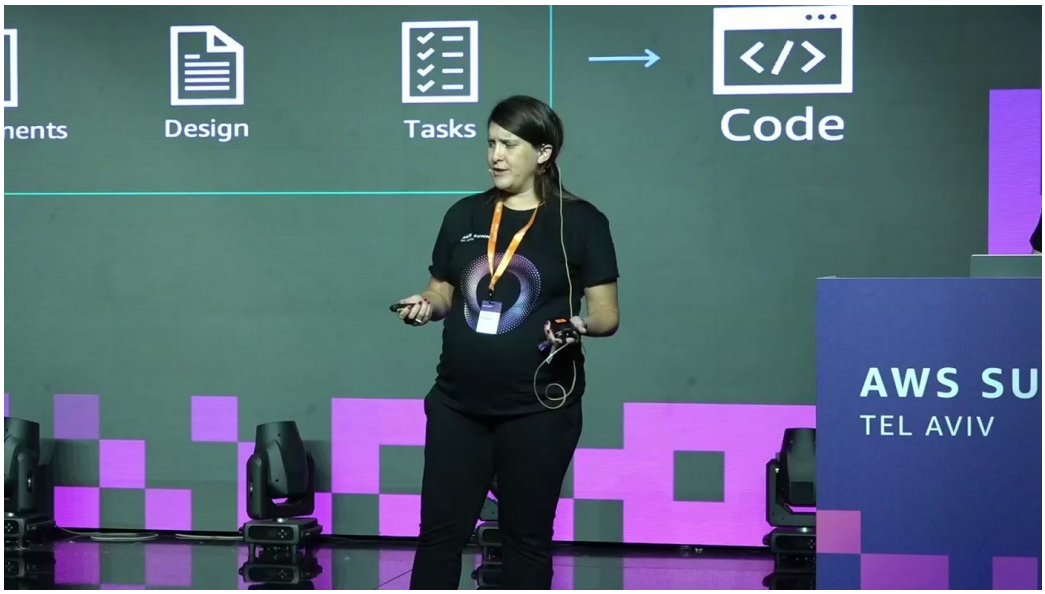
1. תקציר מנהלים

קורל מגן (Technical Account Manager) ודור פיברט (Solutions Architect), שניהם מ-AWS, לא הציגו מוצר חדש אלא שאלה תפעולית אחת: אם כבר החלטנו שהפיתוח נעשה על ידי צי סוכנים ולא על ידי סוכן בודד — איפה הצי הזה בעצם רץ? כל הדמו, כעשרים דקות, הוא תשובה אחת בארבעה מדרגים.

- **נקודת המוצא: הכלי עבר מפסיבי לשותף בתכנון.** קורל פותחת בטענה שהקפיצה האמיתית אינה בקבלת קוד מהר יותר אלא בכך שהכלי נכנס לשלבי ההגדרה: "מכלי פסיבי שנותן לנו את הפתרונות, לשותף פעיל בתכנון" [0:00]. התהליך שהיא מציירת הוא Requirements → Design → Tasks → Code, ורק בסוף כתיבת קוד.
- **המבצע הוא צי, לא סוכן יחיד.** "אנחנו כבר יודעים שכדאי ומומלץ לעבוד עם agent fleets" [1:30] — לכל סוכן מטלה, system prompt וכלים משלו, ומעליהם אורקסטרטור. הסליד מראה חלוקה מפורשת: Product Manager Agent, Architect Agent, Task Planner Agent, ולצידם Coding Agent, Security Agent ו-QA Agent.
- **השאלה האמיתית של הסשן היא שאלת ה-compute.** "אז אם כבר יש לנו תהליך מוגדר ויש לנו צי של agents, על מה עוד נשאר לנו לדבר? איפה כל זה רץ?" [1:30]. הדמו מטפס בארבע מדרגות: לוקלי, ריצה מרוחקת בענן, fan-out של סוכנים מקבילים, וסוכן אוטונומי שמונע מאירוע.
- **הנימוק להעברה לענן הוא פיזי, לא ארכיטקטוני.** דור מונה מכסה שנסגרת, סוללה, ו-Wi-Fi שנעלם ברכבת [6:13], ומוסיף מגבלת משאבים קשיחה: "המחשב שלי יודע להריץ בערך שלושה sessions של Claude Code לפני שהוא מתחיל לעלות השן ופה אני יכול להריץ עשרה, מאה, אלף" [9:16].
- **הסוכן האוטונומי הוכיח לעצמו שהקוד עובד.** בדמו האחרון, issue ב-GitHub עם label בשם AI מדליק דרך GitHub Actions ריצת CodeBuild; הסוכן מרים את האפליקציה ב-Docker Compose, עושה cURL ל-/health, מאמת את התשובה ורק אז פותח pull request [17:07].
- **המסר הסופי הוא דפוסים, לא שירותים.** "זה לא המימוש הספציפי של ה-Lambda MicroVM או ה-CodeBuild, תבחרו באיזה compute unit שאתם רוצים" [20:10].

סליד הפתיחה עם שני הדוברים: קורל מגן, Technical Account Manager, ודור פיברט, Solutions Architect, שניהם AWS.

2. הטענה הפותחת: הקוד הוא אמצעי, לא היעד



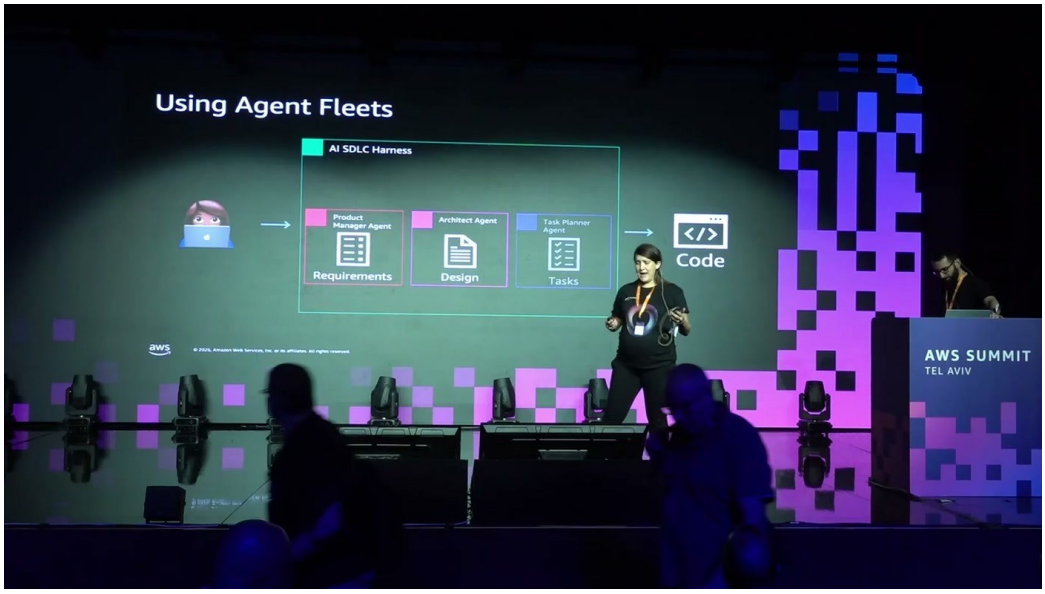
שרשרת השלבים שהוצגה: Requirements, Design, Tasks — ורק בקצה Code.

קורל פותחת בתיאור ההרגל שכולם כבר רכשו: מבקשים מהמודל משהו, מחכים כמה שניות ומקבלים פתרון. הטענה שלה היא שזה כבר לא מספיק: "כי כדי להביא מוצר משלב של רעיון למוצר פעיל בפרודקשן, יש עוד כמה דברים שאנחנו צריכים לעשות" [0:00] — להגדיר מטרות, לקבוע ארכיטקטורה, לפרק למשימות. במילים שלה, "להשקיע בתכנון".

> **— קורל מגן, [0:00]** וזו בדיוק הנקודה שבה היה השתנה בשנים האחרונות, מכלי פסיבי שנותן לנו את הפתרונות, לשותף פעיל בתכנון.

מיד אחרי ההיכרות היא מבצעת סקר קהל — מי כבר עובד עם Kiro, Claude Code או Cursor — ומעירה ש"לא מעט אנשים" הרימו יד [1:30]. זו נקודת הפתיחה שהסשן מניח: הקהל כבר בפנים, ולכן "היום אנחנו ניקח את זה צעד אחד קדימה בדמו ונראה דרכים שונות להריץ AI, אפילו ניגע בשילוב של agents אוטונומיים בתהליך הפיתוח" [1:30].

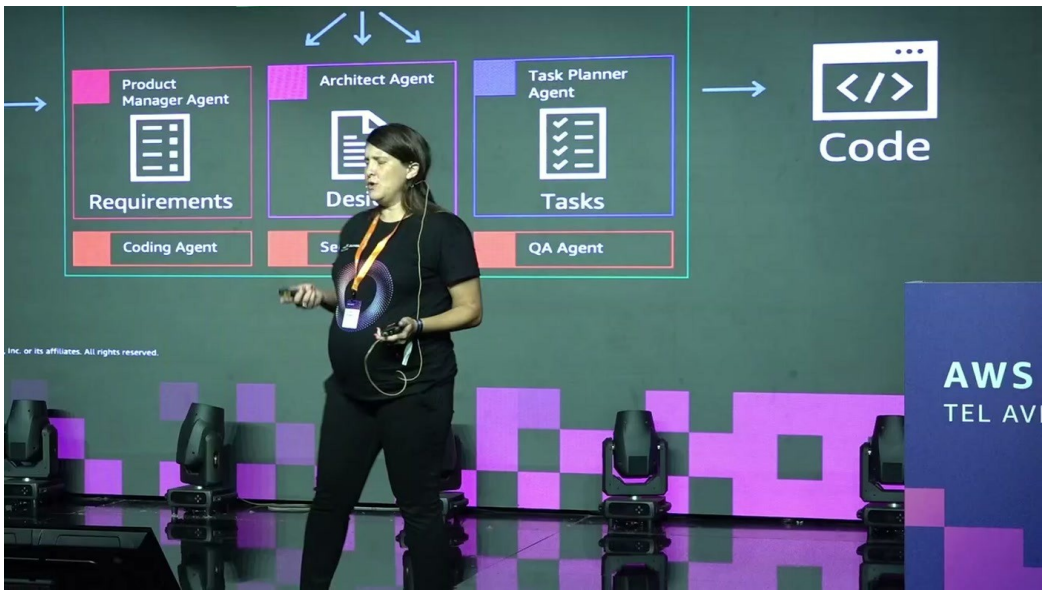
3. Agent Fleets: צי סוכנים במקום סוכן יחיד



המסגרת שהוצגה בשם AI SDLC Harness: מפתח אחד בקצה, ובתוך המסגרת שרשרת סוכנים שמייצרת קוד.

לשאלה מי מבצע את התהליך, קורל עונה בשלילה מפורשת — לא סוכן יחיד:

— קורל מגן, [1:30] אנחנו כבר יודעים שכדאי ומומלץ לעבוד עם agent fleets. כלומר, צי של agents שבו לכל agent יש מטלה ספציפית בה הוא מתמחה, system prompt ו-tools משלו, ואפילו אורקסטרטור שהחי לפנות ולחבר את כל התמונה ביחד.



החלוקה המפורטת: Product Manager Agent על Requirements, Architect Agent על Design, Task Planner Agent על Tasks — ומתחתיהם שכבת Coding Agent, Security Agent ו-QA Agent.

הסליד מפרק את השרשרת לשתי שכבות. השכבה העליונה היא שכבת ההגדרה — Product Manager Agent שאחראי על ה-Requirements, Architect Agent על ה-Design, ו-Task Planner Agent על פירוק ל-Tasks. השכבה התחתונה, שנמתחת לרוחב כל השלושה, היא שכבת הביצוע והבקרה: Coding Agent, Security Agent ו-QA Agent. רק בקצה הימני של הדיאגרמה מופיע Code.

מכאן היא מנסחת את השאלה שהדמו כולו בא לענות עליה:

— קורל מגן, [1:30] אז אם כבר יש לנו תהליך מוגדר ויש לנו צי של agents, על מה עוד נשאר לנו לדבר? איפה כל זה רץ? ועל זה אנחנו נענה בדמו שנציג לכם היום.

התשובה ניתנת מראש כמפת דרכים בת שלוש מדרגות [3:04]: "אנחנו נתחיל מהרמה הבסיסית ביותר, מהרצה לוקלית על המחשב. נתקדם להרצה מרחוק על הענן ונראה פיתוח באמצעות agent אוטונומי באופן מלא." בפועל הדמו הציג ארבעה מצבים, כשההרצה בענן התפצלה לריצה בודדת ארוכה ול-fan-out מקבילי.

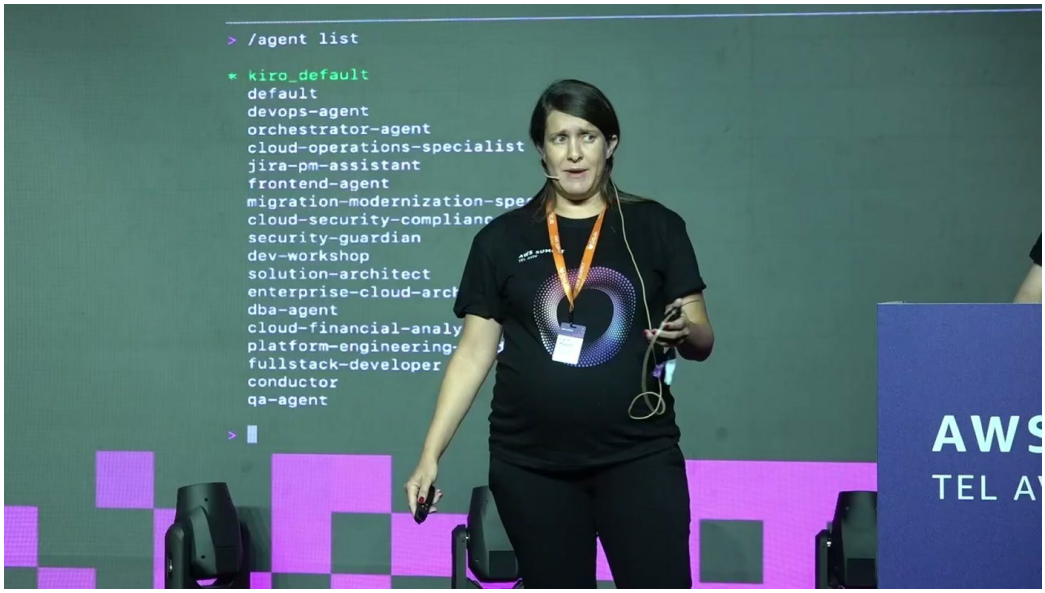
4. Kiro: ה-harness שמריץ את הצי

מיצוב Kiro על הסליד — "The AI SDLC Harness for prototype to production" — לצד מסך ה-CLI שמציג את רשימת הסוכנים המוגדרים.

הרוח הלבנה שעל הסליד היא הלוגו של Kiro. קורל מגדירה אותו כך: "Kiro הוא סביבת פיתוח מבוססת agentic AI של AWS" שמלווה את כל מחזור הפיתוח מהרעיון ועד קוד, בדיקות ושינויים [3:04]. הניסוח על הסליד עצמו מדויק יותר ומסביר את מקומו בסיפור — לא IDE, אלא harness: התשתית שמריצה ומחברת את הסוכנים.

נקודה מעשית שהיא מדגישה: זו לא חבילה אחת.

> **— קורל מגן, [3:04]** חשוב להכיר שקירו מגיע בפלייברים שונים. פלייברים של CLI, IDE, iOS App ו-Web App. אנחנו גם נראה בדמו שילוב ושימוש בקלוד קוד על בדרוק.



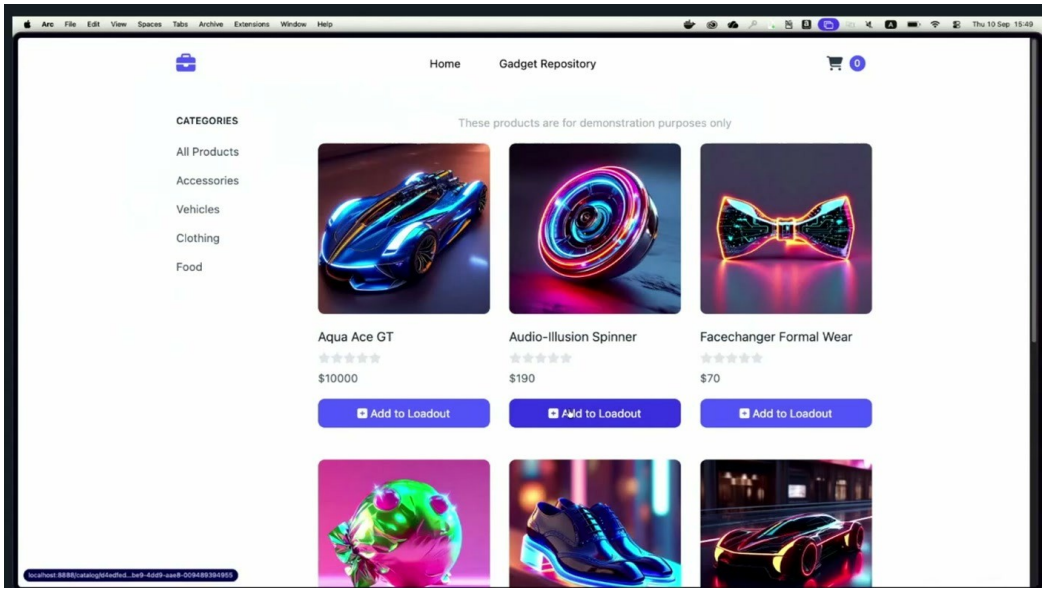
פלט הפקודה /agent list ב-Kiro CLI: הסוכנים המוגדרים מראש, בהם devops-agent, orchestrator-agent, security-guardian, dba-agent, fullstack-developer ו-qa-agent.

רשימת הסוכנים שמופיעה על המסך היא ההמחשה הקונקרטית ל-fleet מהפרק הקודם. היא כוללת, בין היתר: kiro_default, default, devops-agent, orchestrator-agent, cloud-operations-specialist, jira-pm-assistant, frontend-agent, migration-modernization-specialist, cloud-security-compliance-officer, security-guardian, dev-workshop, solution-architect, enterprise-cloud-architect, dba-agent, cloud-

conductor, fullstack-developer, platform-engineering-lead, financial-analyst ו-qa-agent. כלומר:
התפקידים מהדיאגרמה אינם מטאפורה, אלא ישויות מוגדרות שאפשר לבחור ביניהן מה-CLI.

5. נקודת המוצא של הדמו: להכיר פרויקט שלא כתבת

על הדמו הדוברים הדגישו במפורש שזה live demo — "זאת אומרת שהוא לא נעשה פה בחדר הזה" [3:04]. כלומר ההקלטות שעל המסך הן ריצות אמיתיות שבוצעו מראש, ולא סרטוני מוצר.

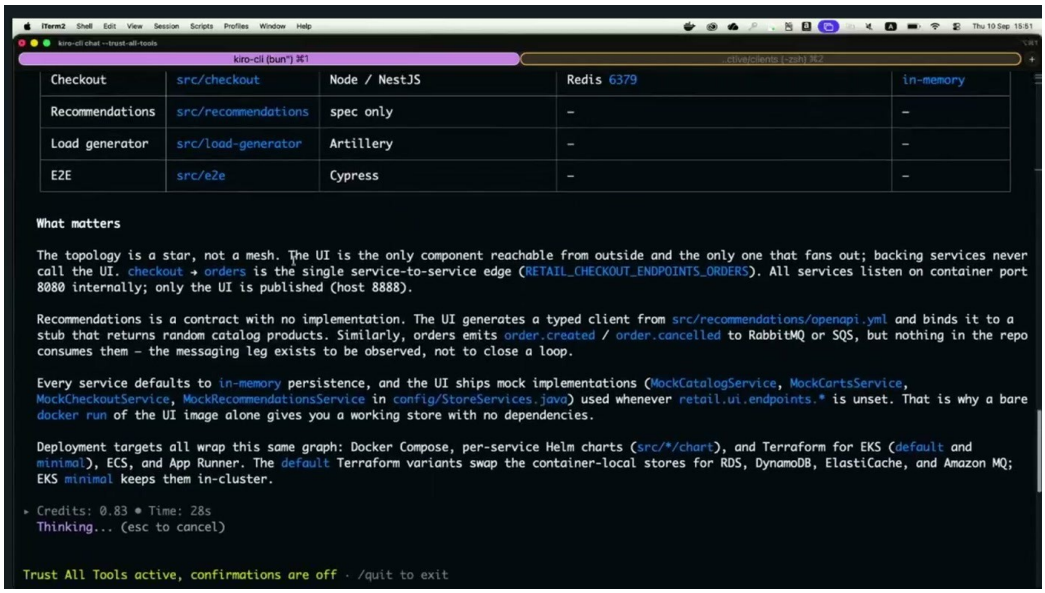


אפליקציית הבסיס לדמו: חנות מקוונת לדוגמה עם קטלוג מוצרים, מחירים ועגלת קניות, שרצה מקומית על localhost:8888.

האפליקציה היא retail store לדוגמה — קטלוג, מחירים, רכישה ודף מוצר. התרחיש שמוצג הוא הצטרפות לפרויקט קיים: "בואו נגיד שהצטרפנו עכשיו לפרויקט מסוים ואנחנו רוצים לעבוד על האפליקציה הזאת" [3:04]. עוד לפני שמדברים על משימה, יש את המכשול הרגיל — דף GitHub עם source files, readme, קבצי YAML, "הרבה קבצים, הרבה טקסט" [4:40].

המהלך הראשון הוא להעביר את עבודת הלימוד עצמה לסוכן:

— קורל מגן, מתארת את הפרומפט של דור, [4:40] קירו תעזור לי ללמוד את הפרויקט, להכיר את הקבצים, ואפילו תייצר לי דיאגרמה של ה-dependencies בין המיקרו סרוויסים השונים.



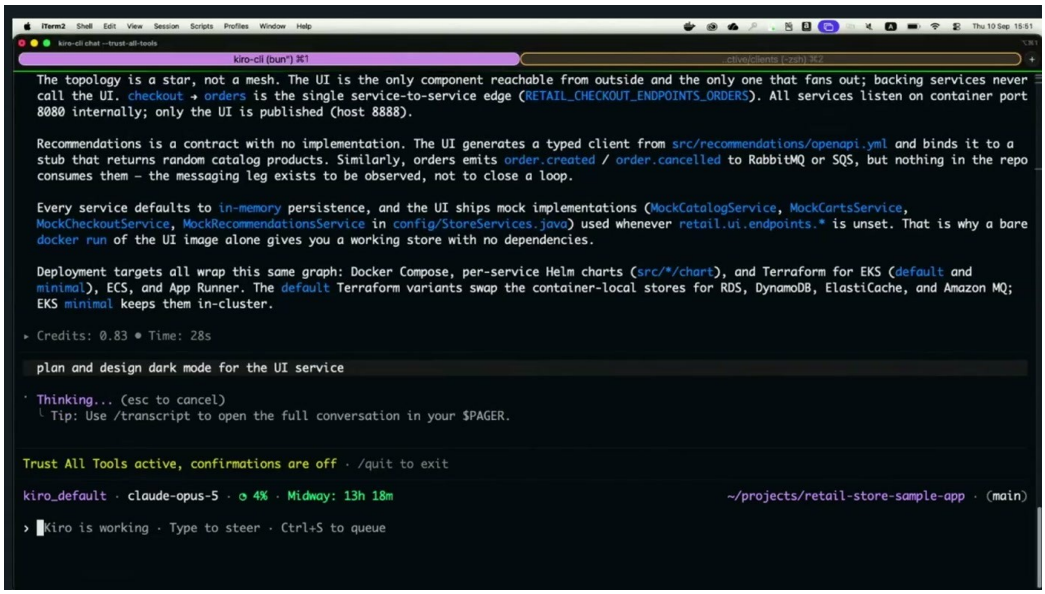
הפלט של Kiro CLI: טבלת שירותים (נתיב, שפה, מנגנון persistence) ומתחתיה סעיף "What matters" שמסכם את הטופולוגיה במילים.

הפלט על המסך הוא שני חלקים. הראשון הוא טבלה ששורותיה הן השירותים — Checkout (src/checkout, Node/NestJS, Redis 6379), Recommendations (spec only), Load generator (Artillery), E2E

(Cypress) ועוד — עם העמודות נתיב, שפת תכנות, דאטאבייס ואחסון. קורל מציינת שהטבלה ניתנת לשליטה: "אפשר לשלוט בזה, להוסיף, לשנות כמה שרק נרצה" [4:40].

החלק השני, "What matters", הוא מה שמבדיל את הפלט מדוקומנטציה אוטומטית רגילה — הסוכן כותב מסקנות ולא רק עובדות. למשל שהטופולוגיה היא כוכב ולא רשת ("The topology is a star, not a mesh"), שה-UI הוא הרכיב היחיד שנגיש מבחוץ והיחיד שמפזר קריאות, ושכל השירותים מאזינים על פורט 8080 פנימי בעוד שרק ה-UI מפורסם על 8888. הוא גם מציין ש-Recommendations הוא חוזה ללא מימוש, ושהודעות order.created / order.cancelled נשלחות ל-RabbitMQ או SQS אך אף רכיב ברפוזיטורי לא צורך אותן.

6. מצב א' — ריצה לוקלית: המשימה האטומית



הפרומפט הלוקלי בתחתית מסך Kiro CLI: "plan and design dark mode for the UI service", עם שורת הסטטוס שמציגה את הסוכן kiro_default ואת המודל claude-opus-5.

המשימה הראשונה היא הוספת dark mode. הנימוק לבחירתה הוא הקריטריון שעל פיו מחליטים היכן להריץ:

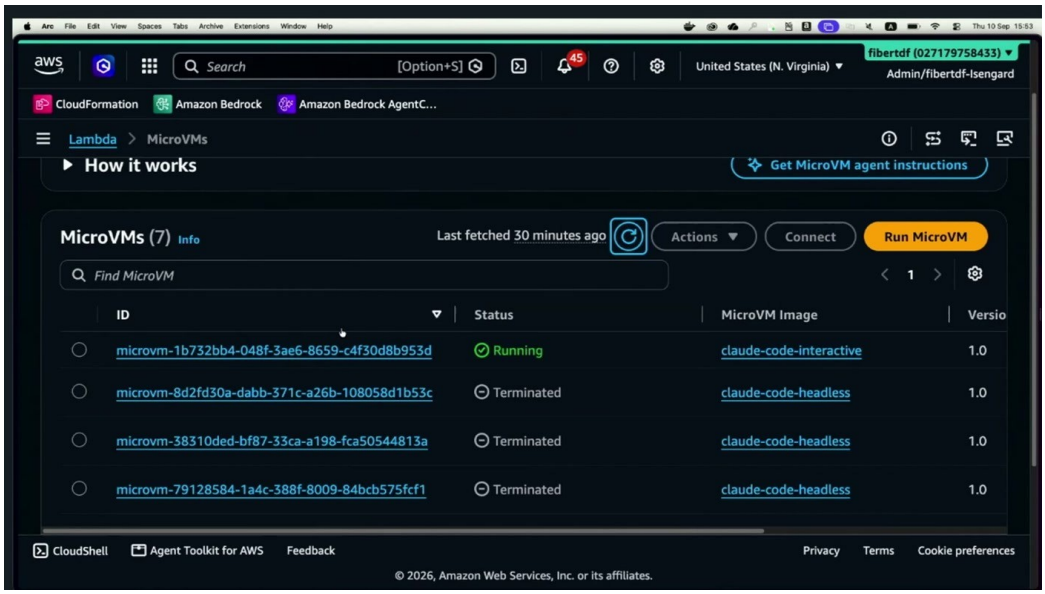
> **— קורל מגן, [6:13]** בסך הכל מדובר על פיצ'ר שהוא נחשב די אטומי, לא היינו מצפים שלקירו ייקח מעבר לחצי שעה להשלים אותו, ואנחנו גם נעבוד עליו לוקולית ממש נקליט במחשב.

כאן קורל מפנה לדור את השאלה שמניעה את שאר הסשן: "מה נעשה אם נרצה לפתח פיצ'ר לא אטומי אלא פיצ'ר רובסטי שצפוי לקחת אפילו מספר שעות?" [6:13]. דור מקבל את ההבחנה ומרחיב אותה לרשימת משימות: לפתח קומפוננטה חדשה, לבצע ריפקטור משמעותי, לחקור כמה כיוונים ולהחליט ביניהם — "שיכולים לקחת שעה, שעתיים, 4 שעות, 8 שעות" [6:13].

הכשל שהוא מתאר אינו כשל תוכנה אלא כשל סביבה:

> **— דור פיברט, [6:13]** ואני פוחד שהמכסה של הלפטופ שלי יסגר והמחשב ילך לישון או שאני אצא מהבית למשרד ובדרך אין לי וייפיי או שתיגמר לי הסוללה בלפטופ, כל מיני אירועים כאלה שיכולים לקרות ולהפסיק את הריצה של האג'נט שלי.

7. מצב ב' — להזיז את הסוכן לענן



קונסולת Lambda MicroVMs עם שבע מכונות: אחת במצב Running עם image בשם claude-code-interactive, והשאר Terminated עם image בשם claude-code-headless.

הפתרון שדור מציע פשוט: אותו סוכן, אותו Kiro CLI או Claude Code — רק לא על הלפטופ. בדמו הוא מתחבר למכונה שהוכנה מראש ומכילה Claude Code, עושה לה connect ומקבל טרמינל בתוך הקונסול [7:45]. הוא כבר ביצע clone לפרויקט, נכנס לתיקייה ומפעיל את הסוכן בתוכה.

חשוב לשים לב שדור מסייג את בחירת השירות כבחירת נוחות ולא כהמלצה ארכיטקטונית:

> **— דור פיברט, [7:45]** בחרתי בשירות הזה כי יש לו UI מאוד נחמד שאנחנו יכולים לראות בקונסול את הטרמינל, אבל אפשר להריץ את הקלוד קוד שלנו, את כל האגנטים שלנו על כל מיני אפשרויות ב-AgentCore Runtime וב-EC2 וב-ECS ובמלא מקומות.

הוא גם מציין שהבחירה בין הכלים אינה חלק מהטיעון: "במקרה הזה קירו יעבוד גם פה, פשוט צריך להכין מכונה עם קירו" [7:45]. היתרון הראשון של המהלך הוא עמידות:

> **— דור פיברט, [9:16]** היא לא תיקבה, לא יהיה לה בעיות של וי-פיי, אין לה סוללה, ואני יכול לקום, להיכנס לרכבת, וכשאני אגיע הביתה, המשימה עדיין תמשיך, או שכבר תסתיים.

היתרון השני הוא מגבלת המשאבים, והוא זה שמייצר את המדרגה הבאה בדמו:

> **— דור פיברט, [9:16]** המחשב שלי יודע להריץ בערך שלושה סשנים של קלוד קוד לפני שהוא מתחיל לעלות השן ופה אני יכול להריץ עשרה, מאה, אלף. אף אחד לא עוצר אותי חוץ ממי שהחייה לבדג'ט שלי.

8. מצב ג' — fan-out: חמישה סוכנים במקביל

כאן דור חוזר ל-Kiro CLI הלוקלי שלו ומפעיל skill שהוא כתב בעצמו:

> **— דור פיברט, [9:16]** אני אקפוץ חזרה לקירו CLI הלוקלי שלי ואני יכול להגיד לו remote fanout. זה סקיל שאני כתבתי שאומר לו תדליק אגנטים על הענן, תעשה להם אופלוד למשימות.

הפרומפט שהוא מקליד בפועל, בעברית ואנגלית מעורבות: "spawn 5 agents to perform a code review for this project. Focus on UI UX, security, usability, coding standards" [9:16]. כלומר, כל סוכן מקבל אספקט אחר של אותו code review.

> **— דור פיברט, [10:55]** ועכשיו קירו CLI הלוקלי שלי מבין שיש לו משימה שמבקשת ממנו להדליק חמש מכונות בענן. בכל מכונה כזאת יהיה קלוד קוד שיעשה קלון לפרויקט, הוא ייתן לו את המשימה, כל אחד מהם יקבל אספקט אחר של הקוד ריוויו.

לאחר שהמכונות מסיימות, ה-Kiro הלוקלי אוסף את התוצאות ומציג אותן על המחשב. דור מסביר שזו בחירה מכוונת להדגמה — "עכשיו כמובן שגם את הקירו הלוקלי אפשר להזיז לענן כמו שעשינו קודם, אבל פה רציתי להראות פעולה היברידית כזאת" [10:55]. בקונסול רואים חמש מכונות חדשות, וכאן נכנסת הבחנה טכנית שמסבירה את עמודת ה-image בצילום הקודם:

> **— דור פיברט, [10:55]** ההבדל בין Headless לאינטראקטיב זה שאין לי טרמינל שאני נכנס אליו. Headless זה משהו שהוא פועל עם API, כי הקירו CLI שלי מדבר עם המכונות האלה ב-API.

> **הסתייגות של הדובר** דור עצמו מדגיש שה-fan-out בדמו הוא דוגמה מינימלית: "כמובן שזה דוגמת דמה ובמציאות אתם תעשו פעולות הרבה יותר מגניבות, כמו תתכנן ואז תריץ ואז תבדוק, ואז תשרשרו אותם" [10:55]. כלומר, הצעד הבא הוא לא רק מקביליות אלא שרשור.

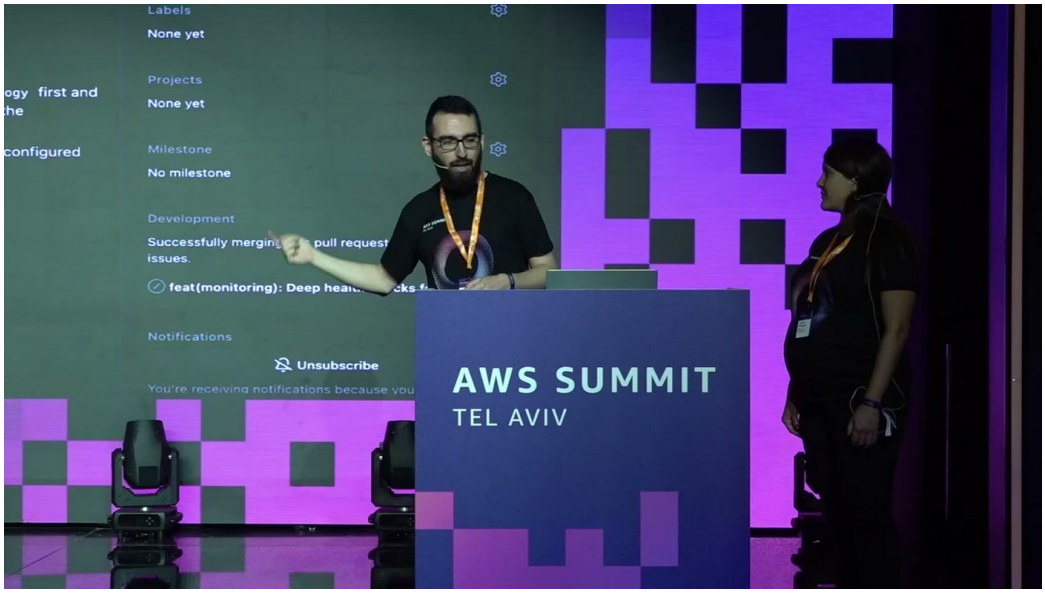
9. מצב ד' — סוכן אוטונומי שמונע מאירוע

קורל מסמנת את המגבלה המשותפת לשלושת המצבים הקודמים ושואלת את דור האם כל התהליכים שהוצגו עד כה, עם Claude Code ועם Kiro, מצריכים הפעלה ידנית [10:55]. דור מאשר — בכל אחד מהם הוא היה צריך להדליק את הסוכן ולכתוב פרומפט.

> **— דור פיברט, [12:33]** לפעמים יש אירועים שקורים ואנחנו היינו רוצים להדליק קודינג אייג'נט בלי התערבות אדם. למשל אם פתחו לנו ספורט טיקט, אז אנחנו יכולים אולי שידלק אייג'נט ויקרא את הלוגים ויקרא מה היוזר עשה ויתן לנו איזשהו סיכום של מה קרה באמת.

הדוגמה השנייה שהוא נותן היא זו שרצה בדמו: issue שנפתח על ידי ה-PM, וסוכן שקם, מייצר POC ומחזיר פיסת קוד עובדת שאפשר להמשיך ממנה [12:33].

9.1 הטריגר



ה-issue ב-GitHub לאחר שהתהליך הסתיים: בסעיף Development מופיע ה-pull request המקושר, feat(monitoring): Deep health checks.

דור פותח issue לפיצ'ר שאינו קיים במערכת: deep health checks לקומפוננטת ה-UI תחת /health. בתיאור הוא מבקש שכאשר מגיעה בקשת HTTP ל-/health, הקומפוננטה תפנה לכל שאר הקומפוננטות בבק-אנד ותבדוק אותן. הוא מוסיף הנחיה מפורשת לאימות: "יש לי Docker Compose בתוך הרפוזיטורי, תשתמש בו כדי להרים גרסה חיה של האפליקציה ותבדוק שבאמת ה-health check הזה עובד" [14:03].

פתיחת ה-issue לבדה אינה עושה דבר. מה שמפעיל את התהליך הוא label:

> **— דור פיברט, [14:03]** אבל אם אני אשים label של AI — כי אני כזה מאוד יצירתי, שמתי label של AI — וזה באמת הטריג את ה-GitHub Actions שהטריגו איזשהו פייפליין שרץ, והדבר הזה מדליק לי agent בענן.

9.2 ה-compute: CodeBuild כמכונה אפמרלית

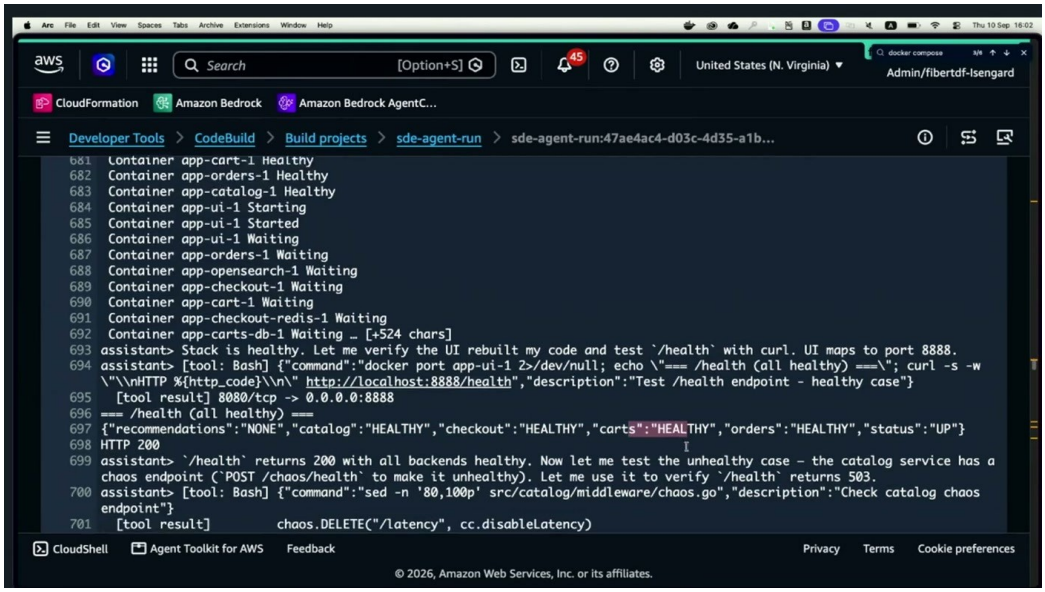
כאן דור מחליף שירות ביחס לדמו הקודם, ומסביר למה:

> **— דור פיברט, [14:03]** במקרה הזה אני משתמש ב-CodeBuild. CodeBuild זה שירות שבמקור מיועד ל-CI, הוא מעלה לנו מכונות Just-in-Time, מכונות אפמרליות. זה מאוד מתאים ליוזקייס הזה שאנחנו צריכים: קרא איזשהו אבנט, צריך קומפיוט שיעלה, ניתן לו משימה, וכשהוא יסיים לעבוד אז המכונה הזאת תלך לישון.

בלוגים רואים את רצף הפעולות: התקנת dependencies, clone, ואז הפעלת Claude Code ב-headless mode. דור מסביר שכל שורת assistant בלוג היא תור של הסוכן, "הוא רץ ב-headless mode כמו שדיברנו מקודם, כי אין

פה אינטראקציה ביני לבין ה-AI. הסוכן פותח בחקירה — LS, git log, קריאת ה-README — ורק אז [15:35] מתחיל לפתח.

9.3 האימות העצמי

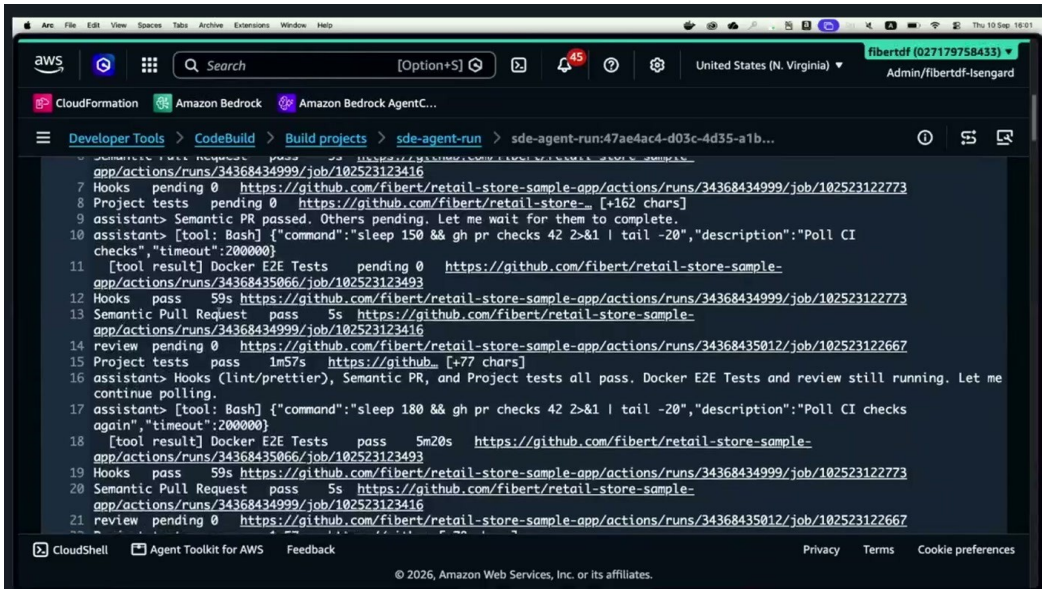


לוג הריצה ב-CodeBuild: הסוכן מריץ curl ל-localhost:8888/health ומקבל בחזרה סטטוס לכל שירות — catalog, checkout, carts, orders — עם HTTP 200.

זה החלק המעניין בדמו. הסוכן מרים את הסטאק ב-Docker Compose, ממתין שכל הקונטיינרים יהיו Healthy, ואז מאמת בפועל:

> **— דור פיברט, [17:07]** ובפקודה הבאה הוא ממש עושה cURL ל-localhost.health, ומבדע שבאמת הוא מקבל את ה-health checks של הקומפוננטות down the line. הוא באמת הוכיח לעצמו שבאמת הפיצ'ר עובד, שהוא הצליח לממש את מה שהוא בא לעשות, והוא ייצר לנו pull request.

בצילום המסך אפשר לראות שהאימות לא נעצר במקרה התקין. אחרי שהתקבל {"catalog":"HEALTHY","checkout":"HEALTHY","carts":"HEALTHY","orders":"HEALTHY","status":"UP"} עם HTTP 200, הסוכן ממשיך לבדוק את המקרה השלילי: לשירות ה-catalog יש chaos endpoint, והוא משתמש בו כדי לוודא ש-/health באמת מחזיר 503 כשמשהו שבור.



הסוכן מריץ gh pr checks בלולאה ומדווח על סטטוס כל בדיקה — Hooks, Semantic Pull Request, Project tests, Docker E2E Tests — עד שכולן עוברות.

הסוכן פותח pull request ואז ממתין ל-CI. דור מדגיש שזו בחירת מימוש פרטית שלו: "זה במימוש שלי, כשאתם תבנו משהו כזה תעשו מה שאתם רוצים, אבל אצלי הוא מחכה שה-CI יעבור, וכשה-CI עבר בהצלחה הוא הולך לישון, הוא סיים את עבודתו" [17:07]. ה-PR שנוצר כולל summary של מה שנבנה, הסבר על אופן המימוש, ופירוט הבדיקות שהסוכן החליט שצריך לבצע.

10. פינג-פונג בין סוכנים

בסיכום הדמו דור מתאר את מה שהוצג: מישהו עם עניין בפרויקט — product manager או בעל עניין אחר — פתח issue, שם label, קמה מכונה, רץ coding agent, הוא מימש את הפיצ'ר, בדק שהכל תקין ויצר pull request [18:38]. ואז הוא מוסיף את השכבה שמעבר:

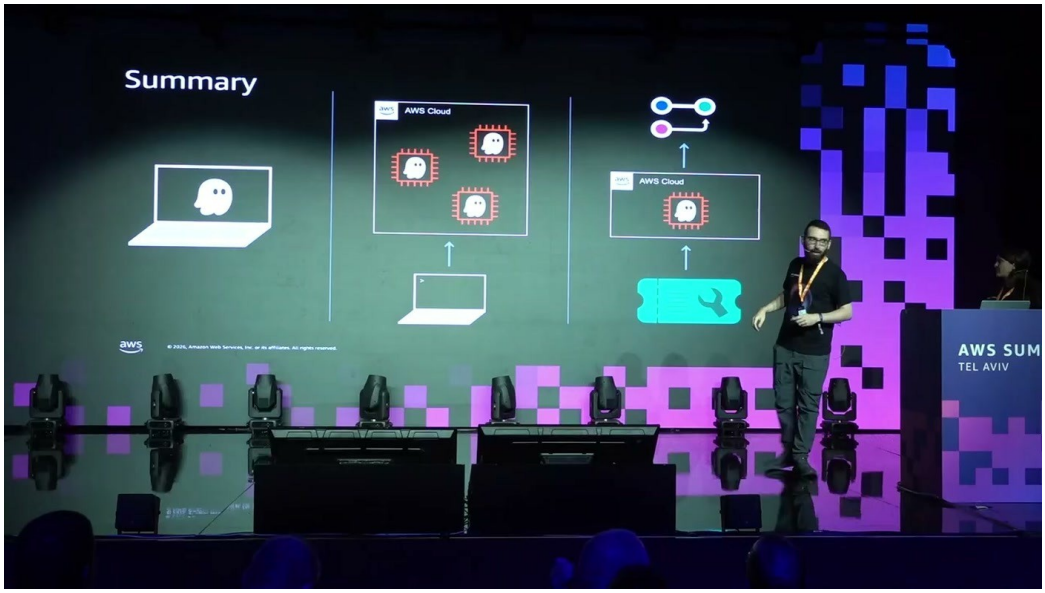
> **— דור פיברט, [18:38]** זה לא חייב לעצור כאן, אנחנו יכולים להיות יותר יצירתיים אפילו מזה. מה אם יהיה לנו agent נוסף שלוקח את ה-pull request הזה ועושה code review ונותן feedback? ובאמת מימשנו גם את זה.

החיבור בין שני הסוכנים נעשה דרך ה-CI, ולא דרך ערוץ תקשורת ייעודי. מכיוון שסוכן הפיתוח מחכה שה-CI יעבור, וה-code review של הסוכן השני הוא חלק מה-CI, נוצר לולאה:

> **— דור פיברט, [18:38]** ובגלל שאמרתי שהאג'נט מי מקודם לא הולך לישון עד שהCI לא עובר, וזה חלק מהCI, אז הם עשו ביניהם פינג פונג עד שהם מגיעו לקוד שהוא ברמה די טובה, או טובה מספיק לפחות.

> **הגבול שנשמר** גם בתרחיש האוטונומי לחלוטין, דור לא מסיר את האדם מהתמונה: "אני כבן אדם יעבור על זה ואבדוק שזה באמת מה שרציתי" [18:38]. ה-pull request הוא נקודת המסירה לאדם, לא עקיפה שלו.

11. הסיכום שהדוברים ניסחו

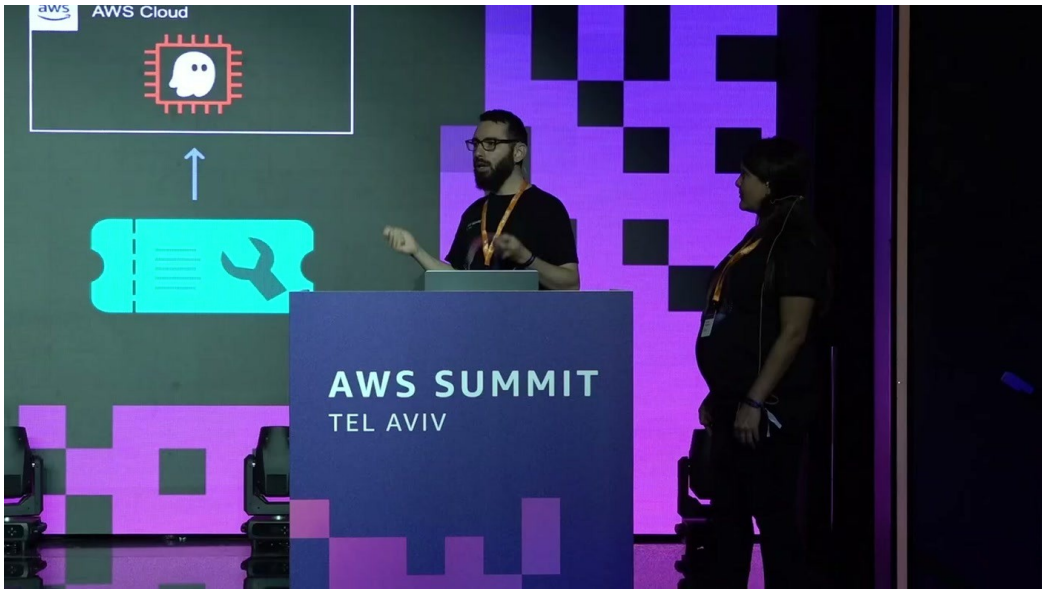


סליד הסיכום, שלושה מצבים משמאל לימין: סוכן על הלפטופ; לפטופ שמדליק כמה סוכנים ב-AWS Cloud; וטיקט שמדליק סוכן בענן שמזין חזרה ל-pipeline.

דור פותח את הסיכום בהסתייגות מהשירותים הספציפיים:

> **— דור פיברט, [20:10]** זה לא המימוש הספציפי של הלמדה מיקרו VM או הקוד בילד, תבחרו באיזה קומפיוט יוניט שאתם רוצים, אבל הרעיונות האלה של אנחנו יכולים להריץ אייג'נטים לוקליים כשזה מתאים לנו, ואם אנחנו צריכים משהו שירוץ לזמן ארוך אז נריץ אותם בענן, ואם אנחנו צריכים המון אייג'נטים במקביל אז אפשר להריץ המון אייג'נטים על הענן.

- **הסוכן הלוקלי:** Kiro CLI שרץ על המחשב האישי. מתאים למשימות אטומיות, כאלה שלא צפויות לעבור חצי שעה.
- **הלפטופ כמפעיל ענן:** הסוכן הלוקלי מדליק סוכנים ב-AWS, מפזר אליהם משימות ואוסף את התוצאות חזרה — התצורה ההיברידית שדור הציג ב-fan-out.
- **הסוכן שרץ בענן:** מכונה בענן שמריצה סוכן ואפשר להתחבר אליה. עמידה בפני ניתוק, סוללה ומכסה סגורה, ולא מוגבלת ל-CPU וזיכרון של הלפטופ.
- **התהליך האוטונומי:** אירוע מדליק מכונה אפמרלית just-in-time, שתופסת את האירוע, מטפלת בו — ולפי דור זה יכול להיות אימות שבקשת הפיצ'ר איכותית, או שחזור של באג — ובסוף מכניסה תיקון ל-codebase.



התקריב על המצב הרביעי: טיקט שמדליק סוכן אפמרלי בענן, ללא התערבות אדם בהפעלה.

12. מה לוקחים מכאן

- **הקריטריון להחלטה איפה להריץ הוא משך המשימה, לא מורכבותה.** הכלל המעשי שעלה מהדמו: משימה שצפויה להסתיים תוך כחצי שעה — לוקלית; משימה של שעות — בענן. זו החלטה תפעולית פשוטה שאפשר לאמץ מיד, בלי לשנות שום דבר בסטאק.
- **fan-out הוא הדרך לעקוף את מגבלת התחנה.** מספר ה-sessions שאפשר להריץ מקומית הוא קטן (דור נקב בשלושה). ברגע שהסוכנים רצים בענן, מספר הסוכנים המקבילים הופך לשאלה תקציבית ולא טכנית.
- **ההבחנה interactive מול headless היא ההבחנה המעשית.** ריצה שאדם מתחבר אליה דורשת טרמינל; ריצה שסוכן אחר מנהל דורשת API. שתי התצורות הופיעו זו לצד זו באותה קונסולה בדמו.
- **אימות עצמי הוא מה שהופך את התוצר לשמיש.** ההנחיה שנתנה לתהליך את ערכו לא הייתה "תכתוב את הפיצ'ר" אלא "תרים את האפליקציה ותוכיח שהוא עובד". הסוכן הריץ Docker Compose, עשה cURL, ובדק גם את מסלול הכשל דרך ה-chaos endpoint.
- **ה-CI הוא מנגנון התיאום בין סוכנים.** הפינג-פונג בין סוכן הפיתוח לסוכן ה-review לא דרש פרוטוקול חדש — הוא נבנה על תנאי קיים ("אל תסיים עד שה-CI עובר") ועל בדיקת CI חדשה. זו דרך זולה לשרשר סוכנים בארגון שכבר יש לו pipeline.
- **ה-pull request נשאר נקודת המסירה לאדם.** בשום שלב בדמו לא הוצע merge אוטומטי. כל התהליך האוטונומי מסתיים ב-PR עם summary, שממתין לבדיקת מהנדס.

13. מילון מונחים

מונח	הופיע בתמלול כ-	מה זה
Kiro	קירו	סביבת פיתוח מבוססת agentic AI של AWS. מגיע כ-CLI, IDE, iOS App ו-Web App.
AI SDLC Harness	—	המונח שעל הסליד למסגרת שמריצה את שרשרת הסוכנים לאורך מחזור הפיתוח, מ-Requirements ועד Code.
Agent Fleet	אייג'נט פליטס	צי סוכנים, כשלכל סוכן מטלה, system prompt וכלים משלו, ומעליהם אורקסטרטור.
Orchestrator	אורקסטרטור	הסוכן שמחלק את העבודה בין שאר הסוכנים ומחבר את התוצרים. מופיע גם ברשימת ה-agent list/כ-orchestrator-agent.
Headless mode	הדלס	הרצת סוכן ללא טרמינל אינטראקטיבי; הבקרה נעשית דרך API. זו התצורה שבה סוכן אחד מפעיל סוכנים אחרים.
Lambda MicroVM	למדה מיקרו VM	השירות שבו הודגמה הרצת סוכן בענן עם חיבור לטרמינל מהקונסול. נבחר בדמו בשל נוחות התצוגה.
CodeBuild	קוד בילד	שירות CI של AWS, שמשמש כאן להעלאת מכונות אפמרליות just-in-time בתגובה לאירוע.
fan-out	פאנאאוט / remote fanout	פיזור משימה אחת למספר סוכנים מקבילים בענן, ואיסוף התוצאות חזרה. בדמו מומש כ-skill שהדובר כתב ל-Kiro CLI.
Deep health check	דיפ הלת' צ'ק	הפיצ'ר שפותח בדמו: endpoint ב-/health שמפיץ בדיקה לכל שירותי הבק-אנד ומחזיר סטטוס מצרפי.

הסשן נסגר בבקשת דירוג דרך QR קוד ובתודה לצוות ההפקה [20:10].