סוכנים מרובי-דיירים על Bedrock AgentCore

AWS Summit Tel Aviv 2026 ,סיכום סשן — ARC401

אליעד מימון ואורן ויס, Senior Solutions Architects, AWS | דוד מלמד, Head of Emerging Technologies, Torq

10 בספטמבר 2026 | משך: 41 דקות | הופק מהקלטת הסשן

סיכום מאת קובי אלמוג

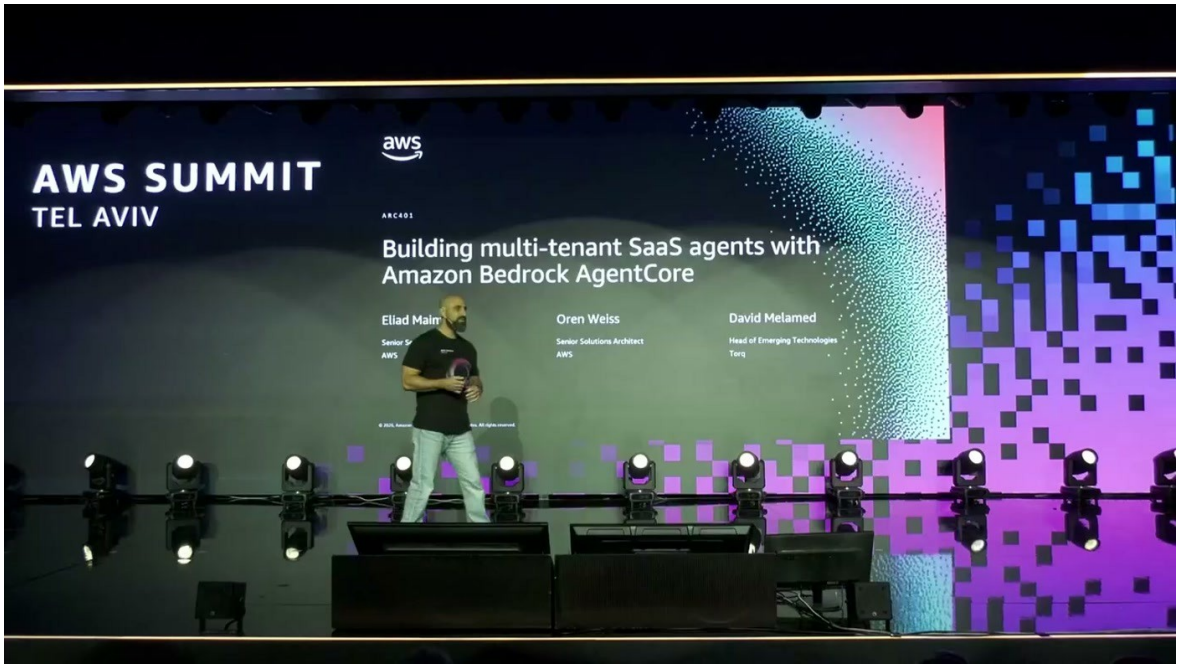
על המסמך הזה

המסמך מסכם את הסשן ARC401 מתוך AWS Summit Tel Aviv 2026, בהתבסס על ההקלטה המלאה שפורסמה באתר הסאמיט. זהו סשן ברמת 400 — הדוברים הצהירו על כך בפתיחה — והוא צולל לשאלה אחת: איך בונים אפליקציית SaaS מרובת-דיירים שבה הרכיב שנוגע בנתונים הוא סוכן אוטונומי, ולא קוד דטרמיניסטי. המסמך נבנה מתמלול אוטומטי של ההקלטה ומהסליידים שחולצו מהווידאו עצמו. חותמות הזמן בגוף המסמך מפנות לנקודה המדויקת בהקלטה.

איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. מה נאמר, ומה נגזר מזה למי שבונה SaaS.
- **פרקים 2–4 — היסודות:** חמשת האתגרים, רכיבי AgentCore, ושלושת מודלי הפריסה — Silo, Pool ו-Bridge.
- **פרקים 5–7 — הליבה הטכנית:** Identity ו-JWT, בידוד זיכרון עם RBAC מול ABAC, והפרדת נתונים ב-Knowledge Base וב-DynamoDB.
- **פרק 8 — כסף:** Observability, SaaS context, ושיוך עלויות פר דייר.
- **פרקים 9–10 — מהשטח ומה לוקחים:** שתי מערכות פרודקשן של Torq ו-Jit, ומסקנות.

> **הערת מקור** המסמך הופק מתמלול אוטומטי של ההקלטה. ההרצאה בעברית ועמוסה במונחים לועזיים, ולכן שמות שירותים ומונחים מקצועיים הופיעו בשיבוש בתמלול ונורמלו כאן למונח האנגלי המקורי מול הסליידים. הציטוטים אומתו מול ההקלטה בחותמת הזמן המצוינת ומובאים כלשונם, כולל מקרים שבהם התמלול משמר תעתיק עברי של מונח אנגלי. שמות הדוברים נלקחו מסלייד הפתיחה.



סלייד הפתיחה: שלושת הדוברים — שני ארכיטקטים מ-AWS ואיש פרודקשן מ-Torq

1. תקציר מנהלים

הסשן נפתח בשאלה בהרמת יד — מי בונה מוצרי SaaS מרובי-דיירים — והמשיך בטענה שהאתגרים המוכרים של SaaS לא נעלמו, הם רק הפכו קשים יותר. אליעד מימון: "צריך לדאוג ל-identity, isolation, הפרדת data, cost attribution, ואלה אתגרים שמלווים אותנו כבר שנים. ועכשיו תוסיפו agents למשוואה" [0:00]. הסשן מיפה חמישה אתגרים, הראה איך כל רכיב של Amazon Bedrock AgentCore נוגע באחד מהם, וסיים בשתי מערכות פרודקשן אמיתיות.

- **השינוי המהותי הוא שהגבול נקבע בזמן ריצה.** "השאלה היא לא רק איך מפרידים בין טננטים שונים, אלא איך מפרידים בין טננטים כשהאג'נט מחליט בזמן ריצה מה הוא רוצה לעשות" [0:00].
- **זליגה בין דיירים לא דורשת תוקף.** "וזה לא חייב להיות פריצה, זה יכול להיות בג בטול שלא מעביר את הטננט הידי, או טוקן לא מכיל את הטננט קונטקסט, פרומט אינג'קשן של יוזר, או אפילו אלוזינשן של המודל" [7:43].
- **האכיפה חייבת לשבת מחוץ לסוכן.** "והכל צריך לקרות מחוץ לקוד של האג'נט, באזור שהאג'נט לא יכול להשפיע עליו או לשנות אותו" [9:14]. זו ההצדקה לכל שכבת ה-Identity וה-Gateway.
- **במודל Pool, naming convention לבדו הוא לא מנגנון אבטחה.** "ההפרדה בין כל הטנטים שלנו תלויה בדבר אחד שהוא הנאמינג קונבנשן ונאמינג קונבנשן יכול להישבר אם האג'ינט יעשה טעות" [17:09]. התשובה היא ABAC עם session tags.
- **מה שלא נמדד מהיום הראשון לא ניתן לשחזור.** "אם אנחנו לא נעשה את האינסטרמנטציה הזו ולא נוציא את הסס context הזה בכל אחד מהלוגים והאקטיביטיז כבר מההתחלה לעולם לא נוכל לשחזר אותו" [21:45].
- **AgentCore הוא מודולרי, ושתי החברות אימצו רק חלק ממנו.** "אחד מהיתרונות שאנחנו ראינו באג'נט קור זה שאתה לא צריך לאמץ את הכל ביחד, אתה יכול לעשות את זה פרוגרסיבי" [31:02].

מה זה אומר בפועל

הסשן לא הציג פיצ'ר חדש אלא ארכיטקטורת אכיפה. השורה התחתונה שלו היא שהגבול בין דיירים חייב להיות נאכף על ידי התשתית — IAM, STS, metadata filtering — ולא על ידי הנחה שהסוכן יתנהג יפה. מי שבונה מערכת אג'נטית מעל נתונים רגישים יקבל כאן רשימת בדיקה קונקרטית לכל שכבה.

2. חמשת האתגרים ורכיבי AgentCore

אליעד פתח במיפוי חמישה אתגרים מרכזיים שמלווים כל SaaS agent מרובה-דיירים [1:38].

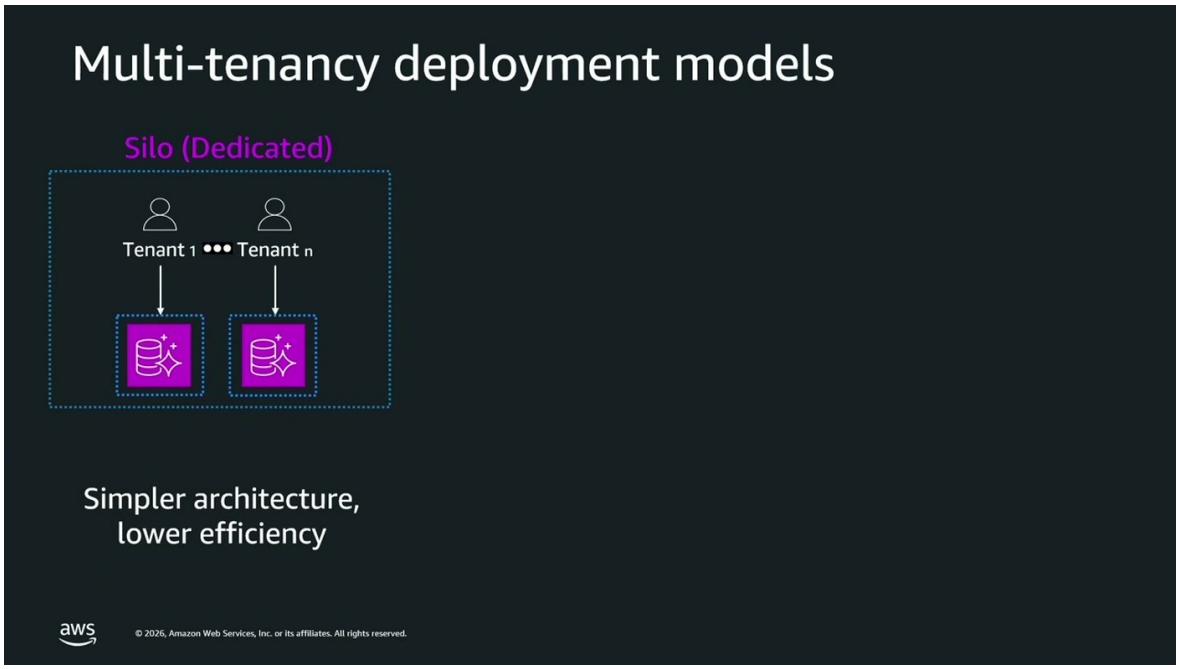
אתגר	מה השאלה בפועל
Tenant onboarding	כמה מהר דייר חדש מתחיל לקבל ערך, ומה צריך להקצות לו
Identity	איך מאמתים ומעבירים tenant ID והקשר לאורך כל שלבי הפתרון
Tenant isolation	איך מונעים cross-tenant access, לא רק מפרידים
Data partitioning	הפרדת הנתונים עצמה — לוגית ופיזית
Observability	ניטור בריאות וביצועים פר דייר, ושיוך שימוש ועלות

לפני שנגע במולטי-טננסי, אליעד פירק סוכן למרכיביו: compute (סביבת הריצה), המודל שמניע את הלוגיקה, tools — שיכולים להיות שירותי AWS או משאבים חיצוניים דרך MCP ו-identity — API ("מי האג'נט ובשם מי הוא פועל"), memory לטווח קצר וארוך, ו-observability [1:38, 3:12]. את המרכיבים האלה הוא מיפה אחד לאחד לרכיבי AgentCore.

רכיב AgentCore	מה הוא נותן, כפי שתואר
Runtime	micro-VM compute, כל סשן מבודד; תומך ב-Strands, LangGraph, CrewAI ואחרים
Gateway	גישה לכלים ב-MCP, עם אכיפת policy, guardrails ואימות
Identity	ביצוע authorization ו-validation; תמיכה ב-OAuth וב-API keys
Memory	זיכרון לטווח קצר (raw storage) ולטווח ארוך (vector store)
Observability	ריכוז לוגים, מטריקות וטרייסים ב-CloudWatch
כלים מובנים	Browser, Code Interpreter, Web Search ואחרים

עיקרון מנחה שתי אמירות שחוזרות לאורך הסשן ומסבירות למה שתי החברות בפרק 9 אימצו רק חלק מהרכיבים: "היתרון המרכזי שאתם מתמקדים ב-Agent Logic לא בונים אינפרסטרקצ'ר", ומיד אחר כך — "Agent Core בנוי מהיום הראשון לתמוך במולטי טננסי" [4:42].

3. שלושת מודלי הפריסה



המודל הראשון על הסלייד: משאב ייעודי לכל דייר, ומתחתיו הפשרה — ארכיטקטורה פשוטה, יעילות נמוכה

אליעד הציג שלושה מודלים [4:42]. ב-Silo כל דייר מקבל משאבים ייעודיים — runtime, gateway, knowledge base — וכל דייר הוא למעשה stack בפני עצמו: בידוד מקסימלי, יעילות נמוכה בניהול משאבים. ב-Pool כל הדיירים חולקים את המשאבים: יעיל בעלויות ובניהול, אך מורכב ארכיטקטונית "גם כי צריך לאכוף את ה-isolation בתוכנה". Bridge הוא שילוב — חלק מהמשאבים משותפים וחלק ייעודיים.

הבחירה, לדבריו, נגזרת מאסטרטגיית ה-tiering: "יכול להיות שלקוחות אנטרפרייז יקבלו יהודי סיילו" — כלומר Silo ייעודי — "והטננטים הקטנים ייכנסו לפול שלכם" [4:42], ובנוסף משיקולי time-to-value, compliance ורמת בידוד נדרשת.

היבט	Silo	Pool
משאבים	runtime, memory, gateway ו-tools ייעודיים לכל דייר	runtime, memory, gateway ו-tools משותפים
הפרדת זיכרון	פיזית — משאב נפרד	לוגית — namespace ו-actor ID
יתרון	"בידוד מלא, אם טנט אחד חווה בעיה זה לא משפיע על הטנטים האחרים" [6:13]	onboarding מהיר, עלות וניהול יעילים
חיסרון	operational overhead	הבידוד תלוי באכיפה בתוכנה
אסטרטגיית הרשאות	RBAC — execution role לכל דייר	ABAC — session tags בזמן ריצה

בשני המודלים ה-JWT הוא החוט שמחבר הכל: "לאורך כל הפתרון עובר לנו ה-JWT, JSON Web Token, שמכיל את האידנטיטי ואת הקונטקסט" [6:13]. ההבדל הוא שב-Silo הוא נוסף על הפרדה פיזית, ואילו ב-Pool — "פה ה-JWT הוא קריטי הוא בעצם מה שמבדיל בין tenant 1 לתננט 2" [6:13].

4. Tenant onboarding

ההבדל בין המודלים מתבטא מיד בהצטרפות דייר חדש [7:43]. במודל Silo ה-control plane מקצה מחדש את כל המשאבים — runtime, gateway, knowledge base — "בדרך כלל תהליך מורכב, שבדרך כלל מצליך שימוש באינפרסטרקשר הזה קוד", כלומר מחייב Infrastructure as Code. במודל Pool הכל כבר קיים, וה-control plane רק רושם את הדייר בבסיס הנתונים, מטמיע את הנתונים שלו ומוסיף אותם ל-Knowledge Base עם tenant tag. הדייר, כלשונו, "רק נכנס לתוך הפול".

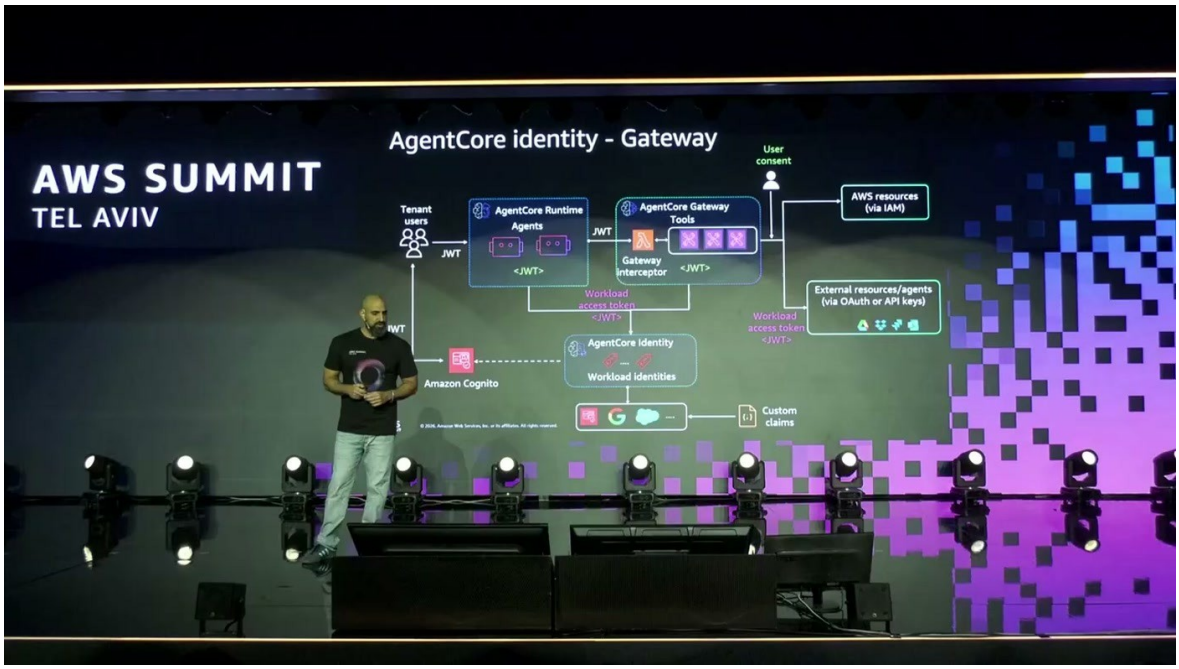
5. Identity — מה-JWT ועד ה-Gateway

זה החלק שאליעד הגדיר כאתגר הגדול ביותר, וההסבר שלו למה מתחיל בהבדל מ-API רגיל: "ב-API רגיל אתם יודעים איך נראית הקריאה איך נראה ה-input, איך נראה output", ולעומת זאת "ב-agent הוא מחליט בעצמו לאיזה כלי לקרוא, באיזה זמן ועם איזה פרמטרים וזה אומר שיש הרבה דרכים ש-agent או טננט יכול להגיע לדאטה של טננט אחר" [7:43].

מכאן נגזרו ארבע דרישות: לזהות את הדייר, לאמת, להזריק tenant context ולוודא שהוא עובר לאורך כל הפתרון, ולהגביל הרשאות כך שגם בתקלה לא תהיה גישה לנתוני דייר אחר [9:14]. והתנאי החמישי הוא המכריע — "והכל צריך לקרות מחוץ לקוד של האג'נט, באזור שהאג'נט לא יכול להשפיע עליו או לשנות אותו" [9:14].

הזרימה שהוצגה

- **Identity Provider —** בדוגמה Amazon Cognito, "אבל ניתן להשתמש בכל IDP אחר"; בדוגמה נבחר user pool לכל דייר [9:14].
- **Custom claims —** לכל משתמש נשמרים פרמטרים של הדייר: tenant ID, tier ועוד.
- **Pre-token generation Lambda trigger —** "שמאפשר לנו לקחת שדות מהCustom Claim ולהתקן אותם ב-JWT" [9:14].
- **האימות עצמו —** המשתמש שולח את ה-JWT ב-request ל-Runtime, "ה-Runtime שולח אותו ל-Identity וה-Identity מאמת אותו יחד עם ה-Identity Provider" [10:50].



מסלול האסימון: מהמשתמשים דרך Runtime ו-Gateway אל AWS resources ואל שירותים חיצוניים, עם workload identities במרכז

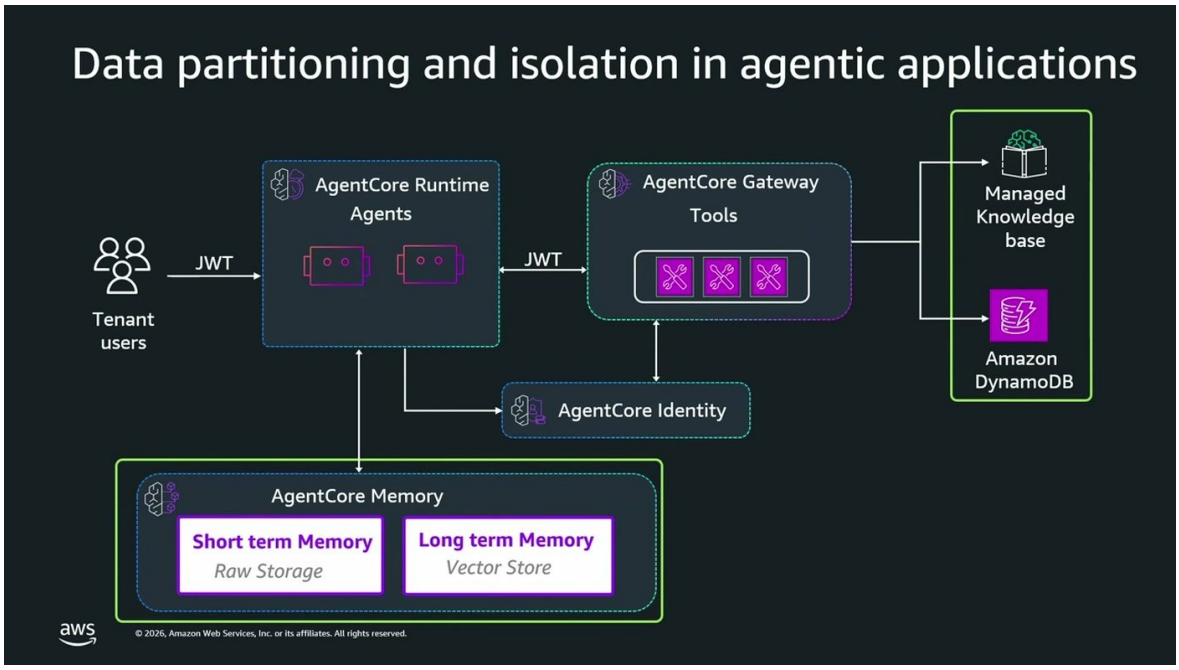
יציאה החוצה: IAM מול OAuth

לגישה למשאבי AWS משייכים execution role ל-AgentCore Runtime. לגישה למשאבים חיצוניים עם OAuth או API key מקנפגים את AgentCore Identity מול ה-IDP, ואז "האג'נט קור אידנטיטי יוצר workload identities עבור כל אייג'נט" ומנפיק workload access token עם scoped permissions שעובר לאורך הפתרון. אפשר גם להוסיף human in the loop על בקשות. אליעד הדגיש שהכל מגיע out of the box: "כל מה שצריך הוא להוסיף annotation בקוד מעל הפונקציה או בטול שהאג'נט אמור לבצע את הפעולה" [10:50].

כשמוסיפים את ה-Gateway, הכלים עוברים לשבת בתוכו והרכיבים מחליפים JWT ביניהם: "האגנט קור רנטיימפ מעביר את הJWT לגייטווי שניהם גם הרנטיימפ וגם הגייטווי יכולים לתקשר עם האגנט קור אידנטיטי לבצע אוטוריזיישן ואותנטיקיישן" [12:20]. מעל זה יושב ה-Gateway interceptor — פונקציית Lambda שיכולה לבצע transformation, validation או אימות מותאם, והיא "משתמש ב-JWT, בהדר, בריקווסט, כדי להוציא את הקונטקסט הרלוונטי" [12:20]. הפרט הזה חוזר בפרק 9, כי היעדרו בזמנו הכתיב החלטה ארכיטקטונית אצל אחת החברות.

6. בידוד זיכרון: RBAC מול ABAC

כאן אורן ויס עלה לבמה והגדיר את ההבחנה שמארגנת את כל הפרק: "דייטה פרטישנינג זו ההחלטה של איך להפריד בין הנתונים של הטנטים שלנו. אבל ההפרדה עצמה לא מספיקה לנו כדי למנוע מטנט A לגשת לנתונים של טנט B בשביל זה אנחנו צריכים להוסיף שכבת אכיפה וזה הטנט ה-isolation שלנו" [12:20].

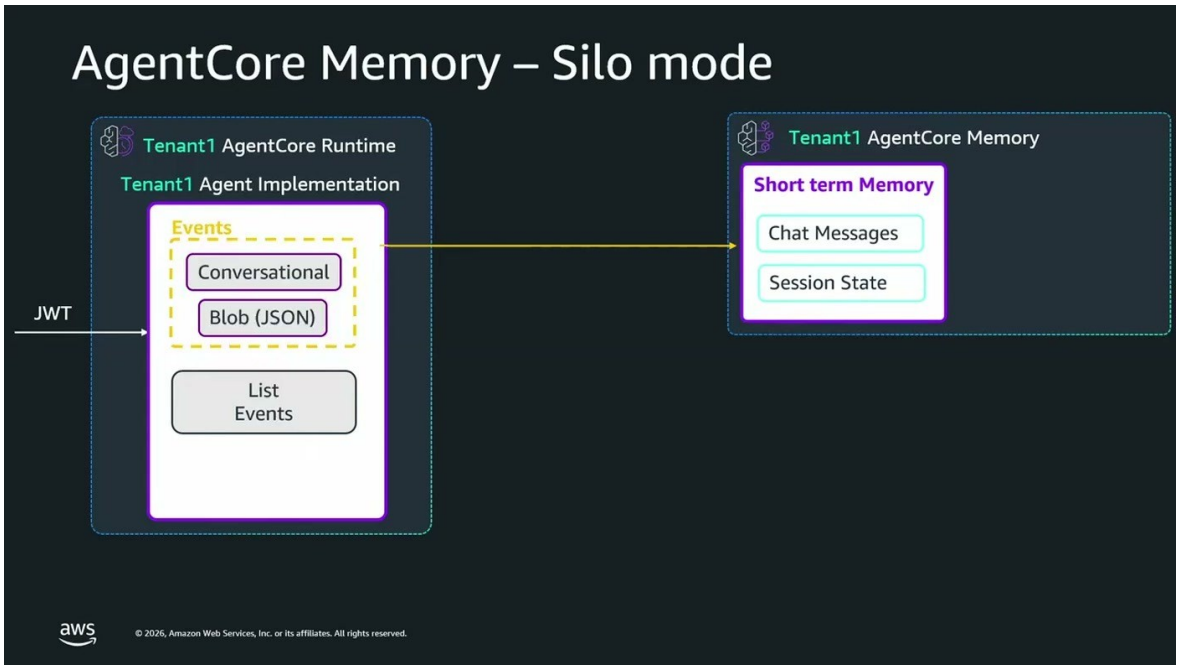


המפה המלאה: ה-JWT נכנס ל-Runtime, ממשיך ל-Gateway, ומשם אל Knowledge Base ו-DynamoDB — כשהזיכרון תלוי בנפרד מתחת

ההצדקה לרף גבוה יותר מזו של אפליקציה רגילה: כשהסוכן "רץ ונוגע בצורה אוטונומית במשאבים כמו agent core memory או ה-managed knowledge base", גבול חלש מדי עלול להוביל לגישה לנתוני הדייר הלא נכון "ואנחנו אפילו לא נדע על כך" [13:56]. לכן, לדבריו, הבחירה על הציר שבין Silo ל-Pool כבר איננה רק איזון בין עלות לבידוד: "אנחנו גם צריכים לשאול את עצמנו כמה ההפרדה שבנינו תהיה טובה וחזקה כשמצד השני נמצא אג'נט שעובד בצורה אוטונומית ולא קוד שעובד בצורה דטרמיניסטית" [13:56].

שלושת המזהים של AgentCore Memory

כתיבה לזיכרון קצר-טווח דורשת שלושה מאפיינים: memory ID (המזהה שמתקבל ביצירת הרכיב), session ID (מזהה הסשן בתוך ה-Runtime), ו-actor ID. השלישי הוא הקריטי, "והוא חשוב כי הוא זה שקובע את החלוקה של הנתונים בתוך הממורי". ההסבר שנתן אורן: "אתם יכולים לחשוב עליה ממש כמו על פריימריקי בתוך דאטה ביס" [13:56]. אותו actor ID משמש גם בתוך ה-namespaces שמפרידים את הזיכרונות ארוכי-הטווח.

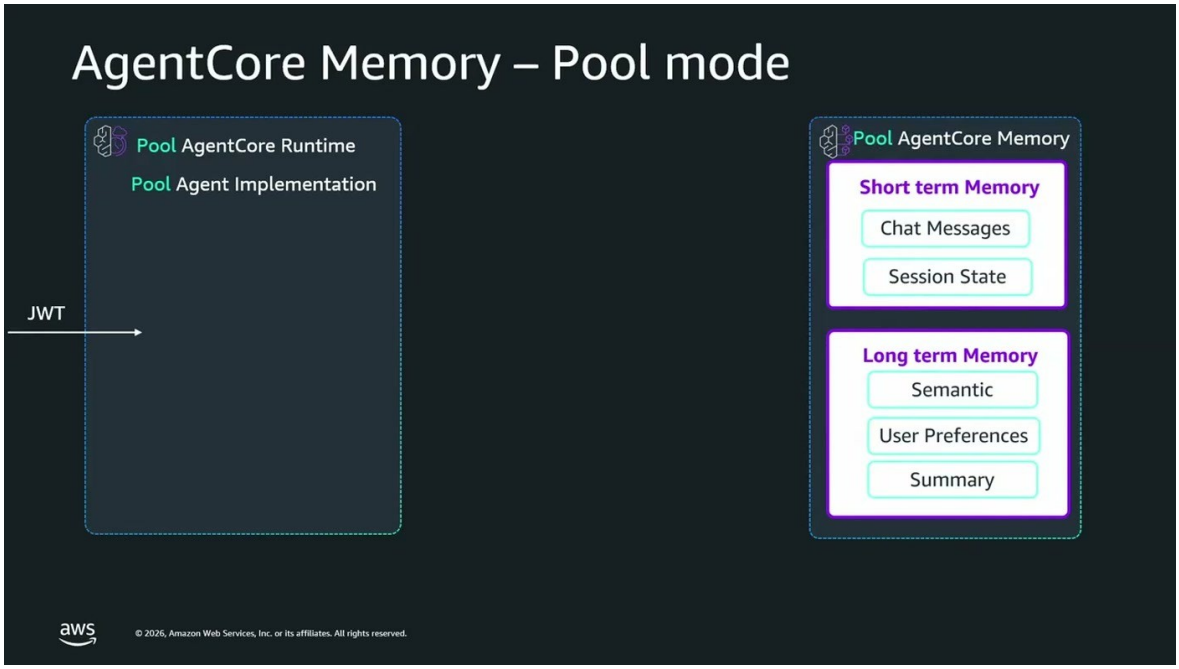


מודל Silo: Runtime וזיכרון נפרדים לכל דייר; ה-events נכתבים כ-conversational או כ-blob JSON

RBAC — Silo, והמגבלה שלו

ב-Silo יש הפרדה פיזית בין המשאבים, ולכן מספיק RBAC: מצמידים ל-Runtime execution role ובתוכו הרשאה ספציפית ל-ARN של רכיב הזיכרון של הדייר. ההמלצה לגבי actor ID היא להשתמש ב-subject מתוך ה-JWT — "המשמעות היא ה-identity של ה-user של הטנט" [15:32].

> **מגבלה** אורן הצביע על מגבלה שקל לפספס: גם ב-Silo, כל המשתמשים של אותו דייר חולקים את אותו משאב זיכרון ואת אותו ARN — "ולכן אנחנו לא מסוגלים לעשות בטכניקה הזו איסוליישן או הפרדה ממש בין היוזרים של הטנטים" [15:32]. בידוד ברמת משתמש דורש את הטכניקה של מודל Pool.



מודל Pool: Runtime אחד וזיכרון אחד לכל הדיירים — אין כבר הפרדה פיזית בין הזיכרונות

Pool — קונבנציה, ולמה היא לא מספיקה

ב-Pool ההמלצה היא למשוך מה-JWT נתון נוסף — tenant ID — ולשרשר אותו ל-subject, וכך לקבל actor ID שמפריד גם בין דיירים וגם בין המשתמשים שלהם, כולל בתוך ה-namespace של הזיכרון ארוך-הטווח [17:09]. ומיד

אחר כך הגיעה האזהרה: "ההפרדה בין כל הטנטים שלנו תלויה בדבר אחד שהוא הנאמינג קונבנשן ונאמינג קונבנשן יכול להישבר אם האג'ינט יעשה טעות אם יש לנו איזשהו בק בקוד" [17:09].

מאחר שיש ARN אחד בלבד להגן עליו, RBAC כבר לא רלוונטי, והפתרון הוא ABAC — Attribute Based Access Control. בפועל מוסיפים ל-policy תנאי שמתיר גישה רק ל-actor ID שזהותו בזמן ריצה זהה לזו ששימשה ביצירת רכיב הזיכרון, ותנאי מקביל על ה-namespaces [17:09].

איך זה נתפר יחד

הזרימה המלאה [18:44]: המשתמש מתאמת, ה-JWT מגיע ל-Runtime ומשם לסוכן. הסוכן בונה ממנו actor ID שהוא שרשור של tenant ID ו-subject, ומעביר אותו כ-session tag ל-STS ברגע שהוא מבצע assume role ל-STS. execution role מחזיר scoped-down permissions שנותנות גישה ל-actor הספציפי בלבד.

> **— אורן ויס, [18:44]** היתרון בזה זה שאנחנו עכשיו לא צריכים ליצור role per tenant, אלא יש לנו role אחד לכל הטננטים שלנו וההרשאות מצומצמות בזמן ריצה באמצעות session tag שמחושב בצורה דינמית

7. Knowledge Base ו-DynamoDB

אותה לוגיקה הוחלה על שאר המשאבים [18:44]. במודל Silo יש Managed Knowledge Base וטבלת DynamoDB לכל דייר, ההפרדה פיזית, והאכיפה היא RBAC — הרשאה ל-ARN הספציפי בתוך ה-execution role.

במודל Pool יש טבלה אחת משותפת, ולכן שמים את ה-tenant ID כחלק ממפתח החלוקה. DynamoDB תומך ב-ABAC, ולכן אפשר להוסיף condition מסוג leading keys שמתיר גישה רק כשה-tenant ID הוא חלק ממפתח החלוקה [18:44].

> **פער בכיסוי** ה-Managed Knowledge Base אינו תומך ב-ABAC — ולכן נדרש מנגנון אחר. זו אחת הנקודות המעשיות ביותר בסשן.

הפתרון הוא metadata attributes: בזמן ה-ingestion של כל קובץ ל-Knowledge Base "אנחנו נדביק ליד כל קובץ, קובץ JSON קטן, שיחיל attributes, אחד מהם יהיה ה-tenant ID". בזמן שליפה הסוכן מעביר את ה-tenant ID וה-Knowledge Base מבצע metadata filtering ומחזיר רק את המסמכים הרלוונטיים — "ממש כמו להוסיף where condition ל-SQL query שלנו" [20:14]. נקודה שכדאי לשים לב אליה: זהו סינון ברמת האפליקציה, לא גבול IAM, וזו בדיוק הסיבה שאחת החברות בפרק 9 סירבה להסתמך עליו במקום אחד.

8. Observability ו-unit economics

במערכת רגילה ל-observability יש תפקיד אחד — לוודא שהמערכת תקינה: כמה שגיאות, מה ה-latency. ב-SaaS נוסף תפקיד שני, שיוך שימוש ועלויות לכל דייר. בלעדיו, כדברי אורן, "לא נוכל להגיד כמה עולה לנו כל טננט וסך הכל לא נוכל גם לזהות אם איזשהו טנט שובר לנו את מודל הרווחיות שלנו" [20:14].

בעולם האג'נטי זה מחריף, כי "כל טננט שלנו מריץ אג'נטים אחרים וכל אג'נט יכול לצרוך טוקנים בצורה אחרת לקרוא לכלים אחרים", ולכן צריכת שני דיירים יכולה להיות שונה לחלוטין — "זה משהו שמערכת אובזרוביליטי לא הייתה צריכה להתמודד איתו קודם" [20:14, 21:45].

SaaS context

הפרקטיקה עצמה, לדבריו, לא השתנתה: עדיין יש דיירים שניגשים למשאבים, רק שהמשאבים הם הסוכנים. מה שנדרש הוא אינסטרומנטציה — שכל סוכן יפלוט בלוגים ובאקטיביטיז אובייקט JSON קטן שנקרא SaaS context, ובו tenant ID, agent ID, region, ו-service tier (standard או priority). ואלה, הדגיש, "לא נתונים שרירותיים אלא נתונים שנגזרים מתוך הצרכים של אנשי הביזנס שלנו" [21:45].

> **נקודת אל-חזור** האזהרה החזקה ביותר בפרק: "אם אנחנו לא נעשה את האינסטרמנטציה הזו ולא נוציא את הSS context הזה בכל אחד מהלוגים והאקטיביטיז כבר מההתחלה לעולם לא נוכל לשחזר אותו" — לא בעזרת backfill ולא בעזרת דשבורד [21:45]. זו החלטה ארכיטקטונית שחייבת להתקבל ביום הראשון.

AgentCore Observability מרכז טלמטריה מה-Gateway, ה-Runtime וה-Identity אל CloudWatch, כולל תמיכה ב-OpenTelemetry ודשבורדים מובנים. אבל, ציין אורן, tenant ID מעולם לא היה מושג מסדר ראשון בעולם ה-observability — לא ב-CloudWatch ולא בפתרונות אחרים — מהסיבה הפשוטה שלא כל אפליקציה היא SaaS. לכן מי שבונה SaaS עובר לעולם של custom metrics ו-custom dashboards, בעזרת AWS Distro — ADOT for OpenTelemetry [23:15].

מהטוקנים אל העלות

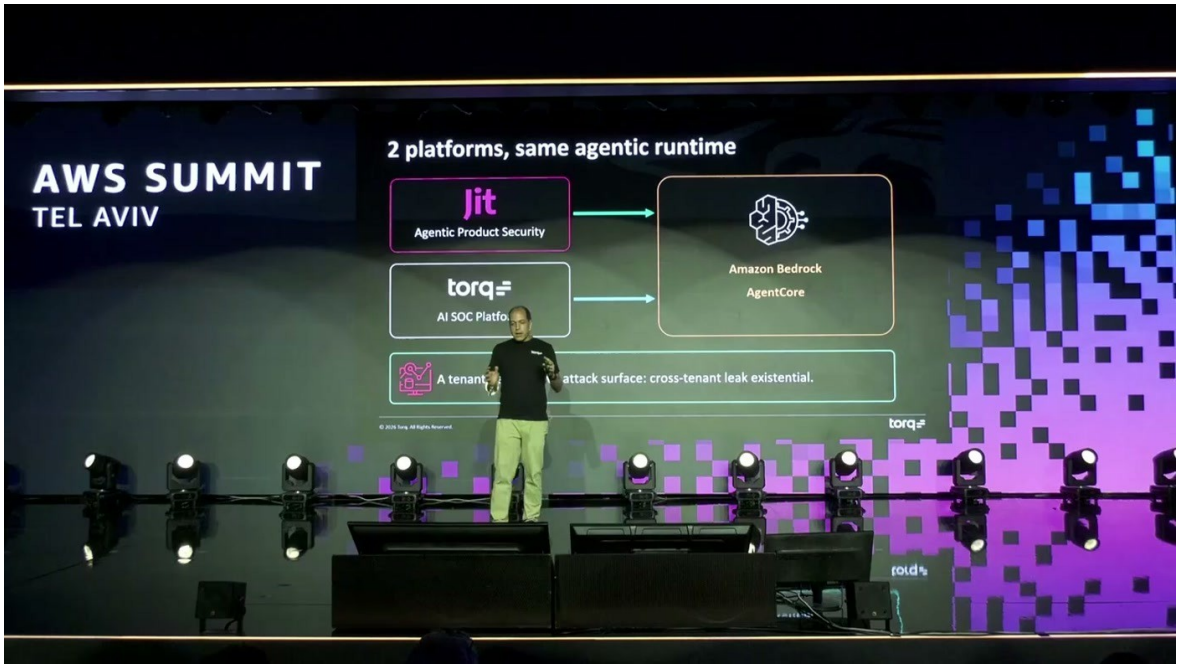
מאחר שבעולם ה-generative AI צריכת הטוקנים היא מה שמניע את העלות, ההמלצה היא ספרייה קטנה שמוציאה מטריקות של tenant ID, המודל שבשימוש, ומספר טוקני הקלט והפלט. משם מריצים queries ואגרגציות ב-CloudWatch Log Insights: "זה עדיין לא פותר לנו את שאלת העלות אבל זה פרוקטי מצוין כי אנחנו יכולים לקחת את מספר התוקנים ופשוט להכפיל אותו בעלות לפר מיליון תוקנים" [24:46].

סוג עלות	מקור הנתונים	שיטת החישוב
צמוד טוקנים	CloudWatch Log Insights	מספר טוקנים × עלות למיליון טוקנים
שירותים שאינם צמודי טוקנים — S3, DynamoDB, AgentCore Runtime	Cost and Usage Report דרך Athena	חלוקה יחסית, למשל לפי מספר ה-invocations של כל דייר

בקצה הזרימה יושבת פונקציית Lambda שמתשאלת את שני המקורות, מייצרת אגרגציות ושומרת את נתוני השימוש והעלות ב-DynamoDB — "ואז נוכל לחבר את זה לכלי בי, כמו Amazon Quick למשל", כך שאנשי העסקים יוכלו לשאול כמה עלה כל דייר ומה הייתה העלות פר service tier [24:46].

9. מהשטח: שתי מערכות פרודקשן

החלק האחרון הועבר לדוד מלמד, co-founder ולשעבר CTO של Jit — פלטפורמה אג'נטית בעולם ה-product security — שנרכשה השנה על ידי Torq, שם הוא כיום Head of Emerging Technologies. Torq, לדבריו, היא "ה-AI SOC market leader" ובונה workflows אוטומטיים לצוותי אבטחה, דטרמיניסטיים ולא-דטרמיניסטיים [26:20, 27:56]. שתי הפלטפורמות בחרו באותו runtime.



שתי פלטפורמות, אותו agentic runtime — ומתחתן המשפט שמסביר את רמת החומרה: דליפה בין דיירים היא קיומית

דוד הסביר למה הנושא חד יותר בתחומו: "אם באופן כללי זה חשוב בכל תעשייה, בתעשיית סייבר סקורטי זה עוד יותר חשוב", כי הנתונים הם ממצאי vulnerabilities בקוד ותוצאות של כל כלי האבטחה בארגון הלקוח [29:29].

runtime — Jit ו-identity בלבד

המשתמשים מתחברים לפלטפורמה ומקבלים JWT; השיחה עם הסוכנים זורמת ב-streaming דרך SSE. ה-Runtime מאמת את האסימון דרך AgentCore Identity וה-JWT authorizer מול ה-IDP, שהיה במקרה הזה Frontegg [29:29]. את הכלים Jit אירחה בעצמה: מ-AgentCore Runtime אחד הסוכנים מדברים ב-MCP אל Runtime שני שמריץ שרת MCP שנבנה בבית. הכלים ניגשים ל-S3 לקבצים שהעלו לקוחות ול-Neo4j ששימש כ-context graph כדי לעגן את הסוכנים בנתוני הלקוח. הזיכרון מומש ב-DynamoDB לטווח קצר וב-MongoDB לטווח ארוך, וה-observability ב-Langfuse ב-self-hosting [31:02].

מדוע לא Gateway, לא AgentCore Memory ולא AgentCore Observability? "אחד מהיתרונות שאנחנו ראינו באג'נט קור זה שאתה לא צריך לאמץ את הכל ביחד, אתה יכול לעשות את זה פרוגרסיבי" [31:02]. בפירוט: ה-Gateway בזמנו "לא נתן לנו את האפשרות להעביר את האידנטיטי בתוך הכלים. עכשיו אפשר לעשות את זה דרך האינטרספטור כמו שראיתם, אבל בזמנו זה בידתי אפשרי" — ולכן נבחר Runtime, שמאפשר להעביר את ה-JWT דרך ה-header ישירות לקונטיינר. ולגבי הזיכרון: "אנחנו התחלנו את הפרויקט הזה לפני AgentCore", וכבר היה פתרון עובד [32:40].

איפה נעצרת האחריות של AgentCore

> **— דוד מלמד, [34:10]** אג'נקור בעצם נותן את כל השכבה של תנת את ה-isolation מתשתית, אבל לא מבחינת דאטה

ולכן את בידוד הנתונים Jit מימשה בעצמה, ברובה במודל Pool — "מודל שהוא בסוף הכי יעיל מבחינת קוסט, מבחינת מנג'מנט" — כל מאגר עם הטכניקה שלו: ב-DynamoDB leading keys, ב-S3 prefix ו-STS tags, וב-MongoDB tenant-scoped IDs [34:10].

ההחלטה המעניינת החריג היחיד הוא ה-context graph ב-Neo4j, שנפרס ב-Silo. הנימוק: "לא רצינו לסמוך רק על מיטה דטה פילטרינג, בעצם על טאגינג שיש בתוך הגרף, כדי לפלטר את הטננט. לא רצינו לסמוך על זה שאנשים כותבים את הקוריס בצורה מספיק בטוחה" [34:10], "בעצם אנחנו מייצרים גרף פר לקוח, וככה אנחנו בטוחים שאין זליגת דייטה, כי שם נמצא הדייטה הכי חשוב" [35:43]. זו הדגמה מעשית של מודל Bridge: Pool בכל מקום, Silo במאגר הרגיש.

הגנה על הכלים — שלוש שכבות

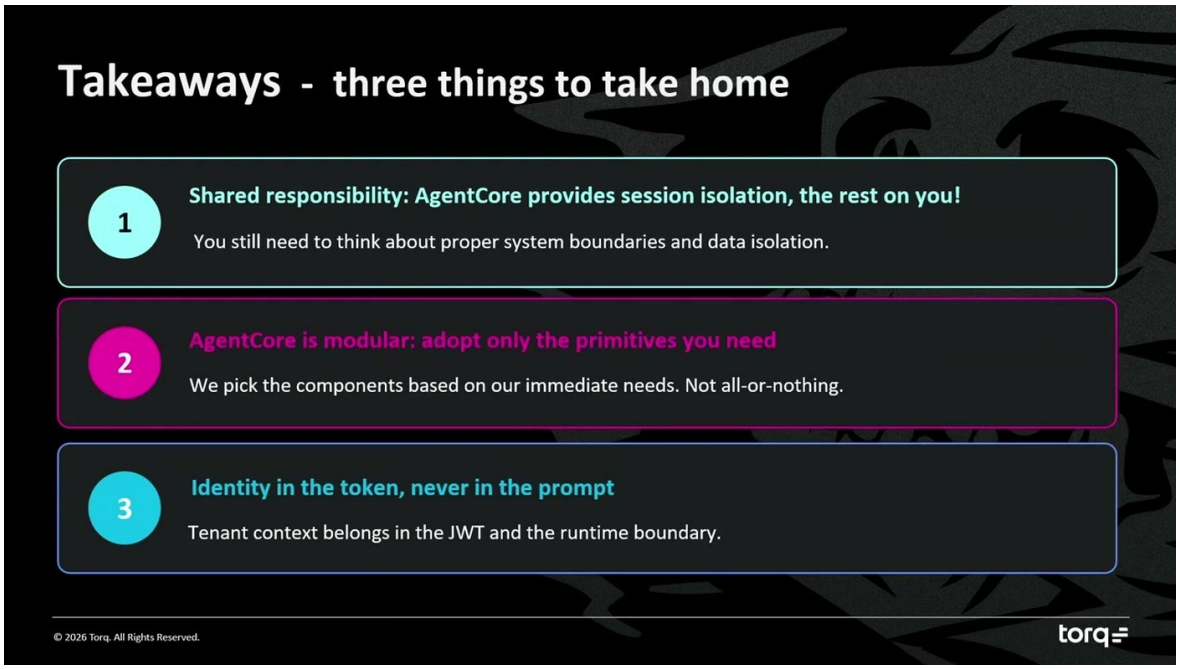
- **אינטגרציה —** אין גישה לכלי אם ללקוח אין את האינטגרציה המתאימה, כי הכלי לא יוכל לעשות דבר בסביבתו.
- **Tier עסקי —** לפי החבילה שהלקוח רכש נפתחות פיצ'רים, והפיצ'רים פותחים חלק מהכלים.
- **RBAC ברמת המשתמש —** הפרדה בין כלים read-only לכלים שמבצעים שינוי בסביבה. הכל מומש "על ידי טאגים, בעצם על ידי דקורטורים בפייטון מעל הכלים" [35:43].

Torq — צורך אחר לגמרי

ב-Torq הדרישה הייתה שונה: "לא היינו צריכים את כל מה שעשינו ב-GIT, לא היינו צריכים את ה-identity. מה שהיינו צריכים זה בעצם יכולת להריץ בסקל, בסקל מאוד גדול, את האג'נטים שלנו ברנטיים" [37:15]. נוסף על כך הייתה דרישה להביא container משלהם, כדי להריץ coding agent עם ה-framework הפנימי ועם Torq DSL — השפה שבה נבנים ה-workflows. לכן נבחר AgentCore Runtime בלבד, "ששם אנחנו הרווחנו מזה שכל session is isolated, ו-agent core מנהל את כל ה-session בשבילנו", ומעליו נבנתה שכבת persistence שמעלה את ה-artifacts של כל סשן ל-S3 ומחזירה אותם לפי session ID [37:15].

ולמה לא פשוט Lambda על micro-VM? "כאן אנחנו רצינו להנות מכל האקוסיסטם והסשן לייפסייקל, וזה גם מאפשר לנו להתקדם אחר כך לעוד קומפוננטות" [37:15].

10. מה לוקחים מכאן



שלוש הנקודות שסגרו את הסשן, כלשונן על הסלייד

- **Shared responsibility חל גם כאן.** "ב-AWS Agent Core, נותן את כל המעטפת, את כל ה-infrastructure, ואת ה-session isolation ברמת ה-VM, והלקוח צריך לקחת על עצמו את השאר" [38:57] — ובראש ה"שאר" נמצא בידוד הנתונים.
- **אימוץ מודולרי, לא all-or-nothing.** Jit לקחה Runtime ו-Torq; Identity לקחה Runtime בלבד. שתיהן הצהירו שהן שוקלות רכיבים נוספים בהמשך [38:57].
- **Identity in the token, never in the prompt.** כלשון הסלייד. הנימוק: prompt injection הוא מהווקטורים הנפוצים ביותר, ולכן העברת הזהות בתוך ה-prompt הופכת את הפתרון ללא מספיק מאובטח [34:10].
- **במודל Pool, ABAC הוא לא אופציה — הוא הגבול.** naming convention מפריד, session tag ב-STS אוכף. ההבדל בין השניים הוא ההבדל בין תיעוד לבין בקרה.
- **האינסטרומנטציה של SaaS context היא החלטת יום ראשון.** אין דרך לשחזר אותה בדיעבד, וממנה נגזרת היכולת לענות על שאלת ה-unit economics.

נקודות להמשך

נושא	למה זה חשוב	חותמת זמן
Gateway interceptor	מאפשר העברת identity לתוך הכלים — היעדרו בעבר הכתיב ארכיטקטורה אחרת ב-Jit	32:40 ,12:20
Managed Knowledge Base ללא ABAC	ההפרדה נשענת על metadata filtering ברמת האפליקציה, לא על IAM	20:14
ADOT ו-custom metrics	tenant ID אינו מושג מסדר ראשון ב-CloudWatch; בלי זה אין שיוך עלות	23:15
AgentCore Policy ו-Guardrails	הוזכרו בסיום כנושאים שלא נכנסו לסשן והופנו להעמקה	40:30

11. מונחון

מונח	משמעות
Silo	מודל פריסה שבו כל דייר מקבל stack משאבים ייעודי משלו
Pool	מודל פריסה שבו כל הדיירים חולקים את אותם משאבים, והבידוד נאכף בתוכנה
Bridge	שילוב בין השניים — חלק מהמשאבים משותפים וחלק ייעודיים
RBAC	הרשאות לפי תפקיד; מתאים כשלכל דייר יש משאב נפרד עם ARN משלו
ABAC	הרשאות לפי מאפייני הבקשה בזמן ריצה; הדרך לאכוף בידוד על משאב משותף
Session tag	מאפיין שמועבר ל-STS בזמן assume role ומצמצם את ההרשאות דינמית
Actor ID	המזהה שקובע את חלוקת הנתונים בתוך AgentCore Memory; מפתח ראשי לכל דבר
Leading keys	תנאי ב-IAM שמתיר גישה ל-DynamoDB רק כשמפתח החלוקה תואם את הדייר
Metadata filtering	סינון מסמכים ב-Knowledge Base לפי attributes שהוצמדו בזמן ה-ingestion
SaaS context	אובייקט JSON שנפלט בכל לוג ומכיל tenant ID, agent ID, service tier ו-region
ADOT	AWS Distro for OpenTelemetry — הכלי לאינסטרומנטציה של מטריקות מותאמות
Gateway interceptor	פונקציית Lambda שמיירטת בקשות ב-Gateway ומחלצת הקשר מתוך ה-JWT
Workload identity	זהות שמנפיק AgentCore Identity לסוכן עצמו, לצורך גישה למשאבים חיצוניים

הקריאה לפעולה בסיום: הפניה להעמקה בבניית סוכנים מרובי-דיירים על AgentCore