ארכיטקטורת תאים: בידוד, סקייל וסיפור המעבר של NICE

AWS Summit Tel Aviv 2026 ,סיכום סשן — TLV-ARC302

מריה, Solutions Architect, AWS | ברק קפטורי, Manager Solutions Architecture, AWS | אורן רבינוביץ', VP Engineering, NICE

10 בספטמבר 2026 | משך: כ-45 דקות | הופק מהקלטת הסשן

מסלול: Architecting on AWS | רמה: 300

על המסמך הזה

המסמך מסכם את הסשן TLV-ARC302 מתוך AWS Summit Tel Aviv 2026, בהתבסס על ההקלטה המלאה שפורסמה באתר הסאמיט. הוא נבנה מתמלול אוטומטי של ההקלטה ומהסליידים שחולצו מהווידאו עצמו, כך שאפשר לחזור לתוכן בלי לצפות שוב בארבעים וחמש הדקות. חותמות הזמן בגוף המסמך מפנות לנקודה המדויקת בהקלטה.

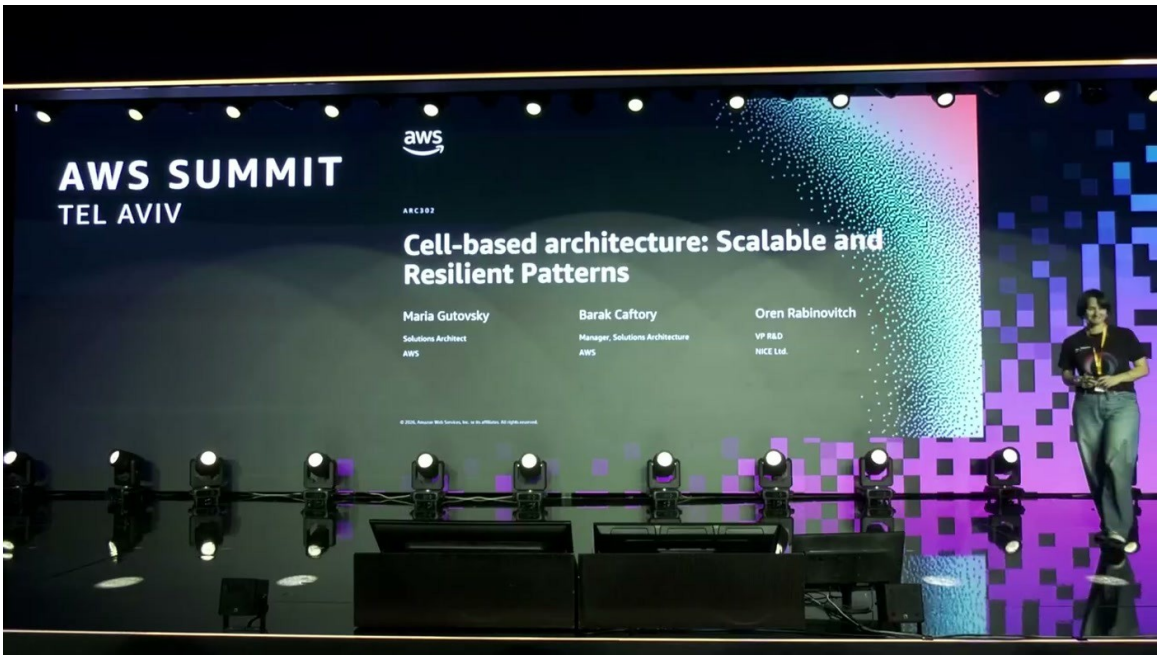
הסשן נחלק לשני חצאים ברורים: חצי ראשון תיאורטי, שבו שני ארכיטקטים מ-AWS מציגים את עקרונות ה-cell based architecture ואת ההחלטות שצריך לקבל בדרך; וחצי שני שהוא מקרה בוחן אמיתי — המסע בן שנה וחצי של NICE, שהעבירה פלטפורמת הקלטות מרובת-דיירים בסקייל של אלפי לקוחות לארכיטקטורת תאים.

איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. מה נאמר, ומה רלוונטי למי שמפעיל מערכת מרובת-דיירים.
- **פרקים 2–3 — התיאוריה:** מה הבעיה שתאים פותרים, איך נראה תא, וכמה עולה הבידוד הזה.
- **פרקים 4–6 — ההחלטות:** גודל ומספר תאים, deployment, ניטור, ושכבת הניתוב.
- **פרקים 7–10 — המקרה של NICE:** למה המודל הישן נשבר, איך שכנעו את הארגון, מה נכנס לתא, ומה יצא מזה במספרים.
- **פרק 11 — מה לוקחים מכאן:** מסקנות מעשיות ומילון מונחים.

הערת מקור המסמך הופק מתמלול אוטומטי של ההקלטה. התמלול סביר בקטעים המקצועיים, אך מונחים לועזיים ושמות מופיעים בו בשיבוש פונטי ותוקנו כאן מול הסליידים. הציטוטים אומתו מול התמלול בחותמת הזמן המצוינת ומובאים כלשונם, כולל שיבושי התמלול. מספרים שמופיעים על הסליידים צוינו ככאלה.

1. תקציר מנהלים



שלושת הדוברים: שני ארכיטקטים מ-AWS ו-VP Engineering מ-NICE

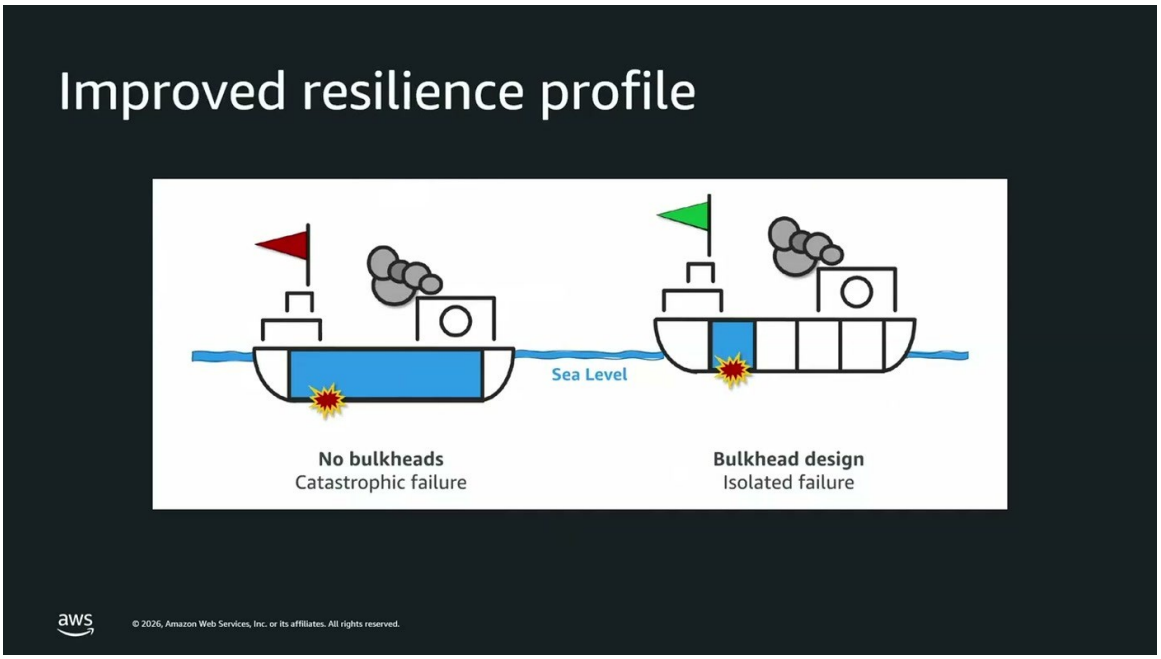
הסשן פותח בשאלה לקהל: מי מכיר cell-based architecture, ומי עובד עם מערכת כזו היום. על השאלה הראשונה עלו ידיים; על השנייה, לפי הדוברת, כמעט אף אחד. הפער הזה הוא בדיוק הנושא של השעה — לא מהם תאים, אלא מה באמת עולה ליישם אותם על מערכת שכבר רצה בפרודקשן.

- **הרעיון מוכר, היישום לא.** "אבל ליישם אותה במערכת אמיתית זה ממש ממש לא פשוט" [1:30]. הקהל בסשן הדגים את זה: מעט מאוד ידיים עלו לשאלה מי מפעיל מערכת cell-based בפועל.
- **השאלה אינה האם תאים, אלא איזה כאב גדול יותר.** "השאלה היא לא האם להשתמש בסלים השאלה היא האם הכאב של לא להשתמש בסלס גדול יותר מהכאב של לטפל אותם" [6:12]. תאים מוסיפים מורכבות תפעולית — הם לא שדרוג חינם.
- **גודל התא נקבע ממטריקה עסקית, לא ממספר לקוחות.** בוחרים מטריקה שמייצגת השפעה עסקית, מגדירים לה סף, וכשמגיעים לסף פותחים תא נוסף במקום להמשיך למלא את הקיים [7:47].
- **שכבת הניתוב היא ה-single point of failure החדש.** היא חייבת להיות דקה ככל האפשר: "מינימום מינימום לוגיקה והכי פשוט שאנחנו יכולים לעשות אותו" [21:45]. ההמלצה הראשונה היא DNS — "אם אפשר להשתמש ב-DNS, תשתמשו ב-DNS" [23:17].
- **הניטור הוא שלב מינוס-אחד.** לפני דשבורדים ואלרטים צריך להכניס מימד cell ID לכל לוג, מטריקה וטרייס [15:35], אחרת אין דרך לחתוך את התמונה לפי תא.
- **ב-NICE המספרים השתנו סדר גודל.** רדיוס הפגיעה לאירוע ירד ממאות דיירים ל-"פחות או יותר 20" [40:18], ימי ההקפאה ירדו ממעל חמישים בשנה לבודדים, והקרדיטים ללקוחות ירדו ב-90% [40:18]. המסע ארך שנה וחצי [32:34].

2. מה שובר מונוליט — שתי קבוצות בעיות

החלק הראשון של ההרצאה ממפה את הכאב לשתי קטגוריות. הראשונה היא blast radius — "כמה גדול הנזק יהיה עם משהו משתבש" [1:30]. כשהכול רץ בסביבה משותפת אחת, לקוח שצורך יותר מדי משאבים (noisy neighbor) מורגש מיד אצל כל השאר, כל deployment הופך למסוכן כי כל שינוי נוגע בכל המערכת, ובדיקה מלאה של המערכת כמעט בלתי אפשרית כי היא פשוט גדולה מדי. התוצאה היא שכל הלקוחות חולקים גורל אחד.

הקטגוריה השנייה היא סקלביליטי: תמיד מגיעים לצוואר בקבוק שבו רכיב אחד מגביל את כל הפלטפורמה, והוספת משאבים כבר לא משפרת ביצועים באופן ליניארי, כי התיאום בין החלקים נעשה מורכב יותר [3:03].

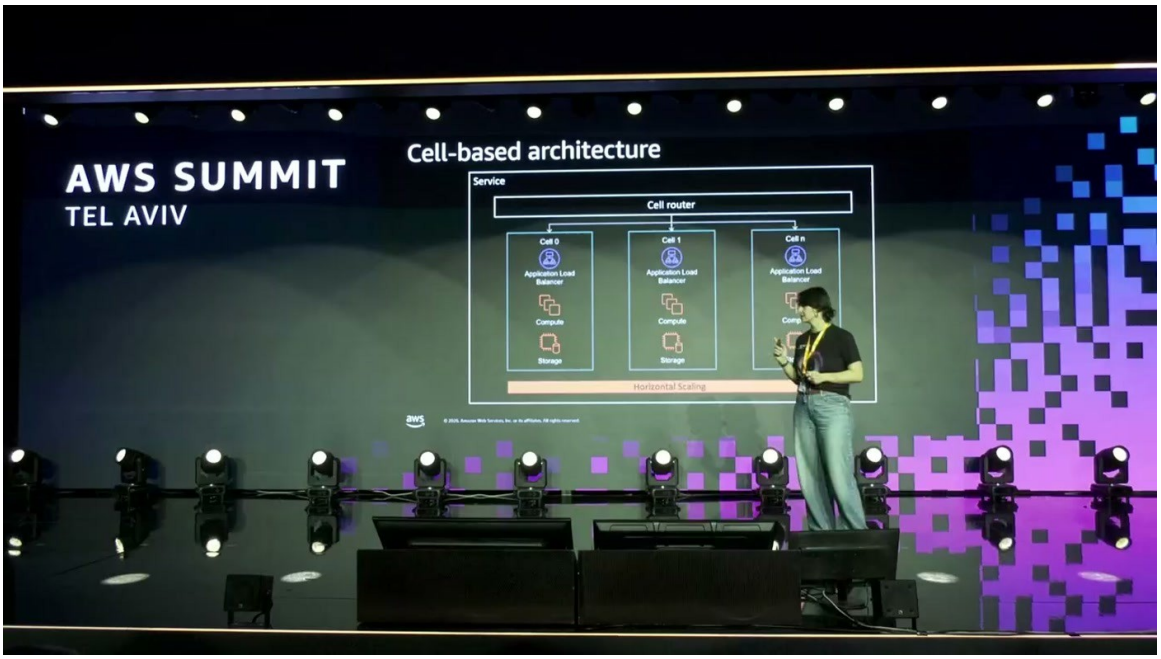


האנלוגיה מעולם הספנות: ספינה בלי מחיצות מול ספינה עם מחיצות — כשל קטסטרופלי מול כשל מבודד

האנלוגיה שנבחרה היא מחיצות בספינה: "תחשבו על הספינה בלי מחיצות פרצה אחת ומים מציפים הכל" [3:03]. הדוגמה הנגדית שהוצגה היא הטיטניק — היו לה 15 מחיצות, אבל הן לא היו גבוהות מספיק, וכשהמים עלו הם פשוט עברו מתא לתא [3:03]. המסקנה שהוצגה: בידוד חלקי אינו בידוד.

— [4:35] אותו עיקרון הוא גם קיים בסל-based architecture בידוד מלא בין הסלים בלי dependencies ביניהם

3. איך נראה תא, ומה המחיר



התא האידיאלי: load balancer, שכבת compute ושכבת storage משלו, מתחת לשכבת ניתוב משותפת אחת

בארכיטקטורה האידיאלית שהוצגה, לכל תא יש כל מה שהוא צריך כדי לרוץ בעצמו: application load balancer, שכבת compute, ואולי שכבת storage — ואין שיתוף של שום דבר בין תאים [4:35]. מעליהם יושבת שכבה דקה אחת, שכבת הניתוב, שמחליטה איזה תא יטפל בכל בקשה. היא חייבת להיות יציבה במיוחד, כי כל התאים תלויים בה.

המבנה הזה הוא שמאפשר horizontal scaling אמיתי: אם כל התאים הקיימים מלאים, מוסיפים תא נוסף במקום להיתקל בלימיטים [4:35]. אלא שבמציאות, כשכבר יש מערכת שרצה שנים בלי תאים, המעבר הזה הוא לא טריוויאלי — וזו בדיוק הסיבה שהחצי השני של הסשן מוקדש לסיפור של NICE.

> **המסגור המרכזי** "אין החלטה ארכיזיקטונית שמגיעה בלי טרייד-אופים. והתפקיד שלנו, לזהות את האופציה הכי פחות כואבת על השולחן" [6:12]. תאים הם כלי חזק, אבל הם מוסיפים מורכבות — ולכן ההחלטה היא השוואת כאבים, לא שדרוג מובן מאליו.

4. ארבע השאלות שצריך לענות עליהן

מרגע שהמבנה ברור, ההרצאה עוברת לארבע שאלות מעשיות: כמה תאים צריכים להיות, איך עושים deployment, איך זה משפיע על מוניטורינג, ואיך עובדת שכבת הניתוב [6:12]. הוצהר במפורש שאלה אינן כל השאלות, אלא הבסיסיות שבהן.

4.1 גודל התא ומספר התאים

התשובה למספר התאים מתחילה מגודל התא. תאים קטנים נותנים blast radius קטן יותר, קל יותר לבדוק אותם ולעשות להם deploy — אבל מאות תאים קשים מאוד לניהול. תאים גדולים חסכוניים יותר ויכולים להכיל לקוחות גדולים, במחיר blast radius גדול יותר [7:47].

הפתרון שהוצע הוא לא לבחור מספר, אלא להגדיר מטריקה: "חשוב להגדיר את המטריקה או שילוב של מטריקות שמייצג את ההשפעה העסקית שלנו משהו שאפשר לבדוד, למדוד ולנטר" [7:47]. מגדירים לה סף, עוקבים אחריה, וכשמגיעים לסף — זה הסימן לפתוח תא נוסף.

> **— [7:47]** מאוד חשוב לא להמשיך לדחוף לסל שכבר מלא עוד וורקלואוד ועוד טננטים

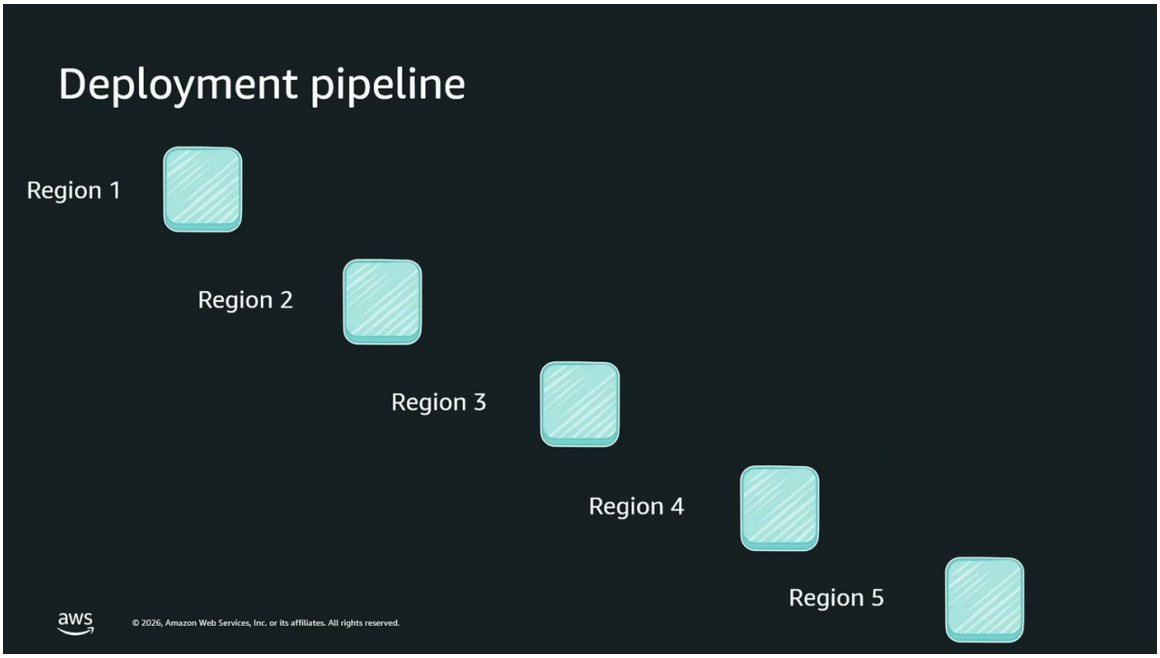
כבר בשלב הזה ניתן טיזר: המטריקה שבחרו ב-NICE אינה מספר הדיירים [7:47]. נחזור לזה בפרק 9.

שיקול נוסף הוא שלא כל הלקוחות דומים. לקוחות גדולים מאוד צורכים הרבה משאבים והופכים ל-noisy neighbors בתא משותף, ולקוחות אחרים מגיעים עם דרישות אבטחה שמשפיעות על בידוד נתונים [7:47]. תאים עונים על זה באופן טבעי — לקוח מסוים יכול לקבל תא ייעודי ואחר תא משותף — אבל כאן מגיעה אזהרה: ברגע שיש כמה סוגי תאים, צריך יותר ניתוב ויותר טיפול ב-storage, ואפשר למצוא את עצמך מנהל כמה מוצרים שונים. ההמלצה היא לשמור על אותו codebase, אותו pipeline של CI/CD, ולנהל את ההבדלים באמצעות feature flags [9:17].

5. Deployment: התא כיחידת הפצה

הנתון שפתח את הפרק: הרבה תקלות קורות מיד אחרי העלאת גרסה חדשה — "קוד חדש הוא יותר רגיש לטעויות מקוד שכבר רץ במערכת" [9:17]. מכאן ההמלצה.

> **— [9:17]** כשיש לנו Cell Based Architecture ההמלצה היא להשתמש בסל כיחידת deployment



צינור ההפצה כפי שהוא נראה כשיש תא אחד בכל אזור — חמישה שלבים סדרתיים

התרחיש שהודגם: אפליקציה שרצה בחמישה אזורים. לוקחים אזור אחד, עושים deployment, ואז ממתינים — bake time — בודקים מטריקות, ורק אז משחררים לשאר. זה עובד יפה עד שמוסיפים תאים: אם מחליטים שה-blast radius באזור גדול מדי, או שלקוח public sector דורש תא ייעודי, מגיעים לשלושה תאים בחמישה אזורים — חמישה עשר שלבים סדרתיים [10:48].

הבעיה, כפי שהוצגה, היא שקצב השינויים רק עולה: "ובזמן שמי שכותב את הקוד זה AI אייג'נט והקצף של שינויים הוא יותר מהר" [10:48] — ואז פשוט אי אפשר להרשות לעצמך צינור כזה.

fractional deployment

הפתרון שהוצג הוא הפצה במקבילים הולכים וגדלים. עדיין מתחילים בקטן — תא ראשון באזור ראשון, ממתינים ובודקים — ואז מעלים שני תאים באזורים 2 ו-3 במקביל [12:22]. הכלל שהודגש: לבדוק לפחות תא אחד בכל אזור, כי גרסה שעבדה ב-US ובסידני עדיין יכולה ליפול בפרנקפורט בגלל תווים מיוחדים בגרמנית [12:22]. אחרי שיש ביטחון שהגרסה עובדת באזור אחד לפחות בכל אזור, אפשר להגדיל את הצעדים.

> **— [12:22]** מאוד חשוב לבדוק לפחות סל אחד בכל ריג'ן

6. ניטור: שני כוחות שמושכים לכיוונים הפוכים

בשלב הזה עלה לבמה ראש צוות הארכיטקטים מ-AWS. הטענה הפותחת: ניטור הוא קריטי בכל מערכת, אבל במעבר לתאים הוא נעשה קריטי אף יותר — "שטח הפנים והמורכבות של המערכת גדלים משמעותית והיכולת שלנו להבין אותה בלי לבנות את המערכות הנכונות היא קטנה" [14:00].

כאן מוצג המתח המרכזי של הפרק: מצד אחד צריך מבט הוליסטי ואגרגציות ברמת הצי כולו; מצד שני אסור ליצור shared fate חדש בין מערכת הניטור לבין התאים [14:00]. מערכת הניטור לא אמורה להיות התלות המשותפת שמבטלת את הבידוד שבשבילו נבנו התאים.

6.1 שלב מינוס אחד: מימד ה-cell

> **— [15:35]** וזה ה-cell ID. צריכים להוסיף את זה לכל לוג, לכל מטריקה, לכל הטרייסים שלנו

מימד ה-cell צריך להפוך ל-first class citizen בכל הטלמטריה, ורק אחריו אפשר לעשות חיתוכים, פילוחים, אלרטים ודשבורדים על בסיסו [15:35]. הדובר מכנה את זה "סטפ מינוס אחד" — מה שקורה לפני שמתחילים לחשוב על ניטור בכלל.

6.2 ריבוי נקודות מבט, ודשבורדים בכמה רמות

המערכת עצמה לא צריכה להיות הסמכות הבלעדית לשאלה אם היא עובדת. הדרכים שהוזכרו: client-side monitors, ותנועה סינתטית שמגיעה ממקומות שונים — ובארכיטקטורת תאים חשוב לעשות את זה עבור כל התאים וכל ה-endpoints [15:35].

דשבורדים צריכים להיות בגרנולריות משתנה, מרמת הצי כולו ועד תא בודד, ולא רק לדשבורדים אפליקטיביים אלא גם לתשתיתיים ולעסקיים: אם יש מטריקות עסקיות, עוקבים אחריהן ברמת התא וגם מייצרים מהן אגרגציה גלובלית [17:07].

6.3 סופת האלרטים

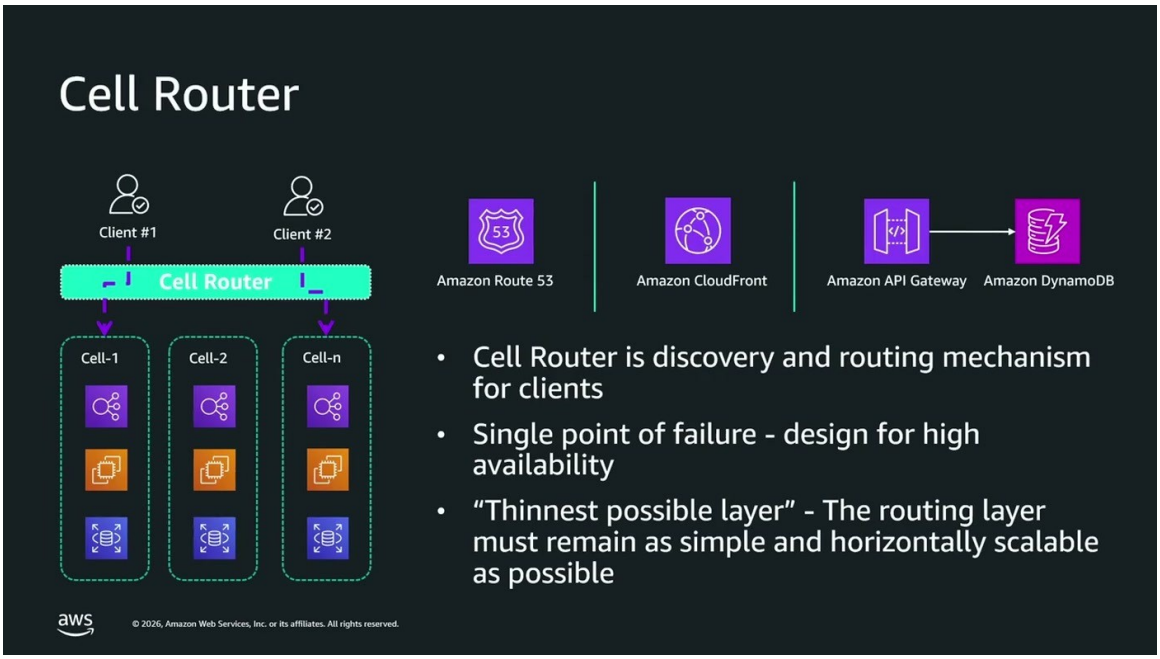
התופעה המוכרת — תקלה אחת שמייצרת מבול אלרטים שמקשה על root cause analysis — מתעצמת ככל ששטח הפנים גדל. הכלי שהומלץ הוא composite alarms: לוגיקה שמחברת בין מטריקות כדי לייצר אלרטים בגרנולריות משתנה, נתמך ב-CloudWatch ובספקי observability אחרים [17:07]. המטרה המעשית: אלרט שמזהה שהתקלה היא cross-cutting על פני כמה תאים, במקום חמישה אלרטים נפרדים שכל אחד מצביע על תא אחר [18:39].

6.4 השאלות החדשות שאפשר לשאול

רמה	השאלה שאפשר לשאול עכשיו	למה זה משנה
צי	כמה לקוחות מושפעים ברגע נתון ואיזה אחוז מכלל הלקוחות [18:39]	קובע את ה-severity ומי קופץ באמצע הלילה
צי	מה מצב ה-cell router עצמו [18:39]	בלי זה, תקלה בהרבה תאים נראית כמו תעלומה
צי	האם יש תא חם שמקבל הרבה יותר תעבורה מהשאר [20:11]	מצביע על placement שגוי של דיירים, או על ניסיון DDoS
תא	מה הניצולת של התא, והאם מתקרבים לסף [20:11]	מתי צריך להתחיל בתהליך הקמת תא חדש

> **המשמעת הארגונית** "להקים סל לרוב זה פעולה שהיא מורכבת ולוקחת זמן ויש לה עלויות, וזה לא משהו מרוב המקומות שאני ראיתי, שזה פשוט קליק, אלא זה כן תהליך ארגוני לא טריוויאלי" [20:11]. מכאן המלצה לקבוע סף נמוך מספיק — ולהתעקש עליו, כי ברגע האמת מפתה להגיד "בוא נגדיל את הסל הזה עוד קצת" [20:11].

7. שכבת הניתוב: שלושה מנגנונים



ה-cell router ושלושת השירותים שממשים אותו: Route 53, CloudFront, API Gateway ו-DynamoDB כ-backing store

ה-cell router הוא הרכיב שמתווסף בדיוק כשעוברים לתאים, ותפקידו discovery וניתוב לקוחות. מכיוון שהוא single point of failure, עיקרון ההנחיה הוא לשמור אותו דק ככל האפשר.

[21:45] מינימום מינימום לוגיקה והכי פשוט שאנחנו יכולים לעשות אותו כמובן לא יותר פשוט ממה שהוא חייב להיות —

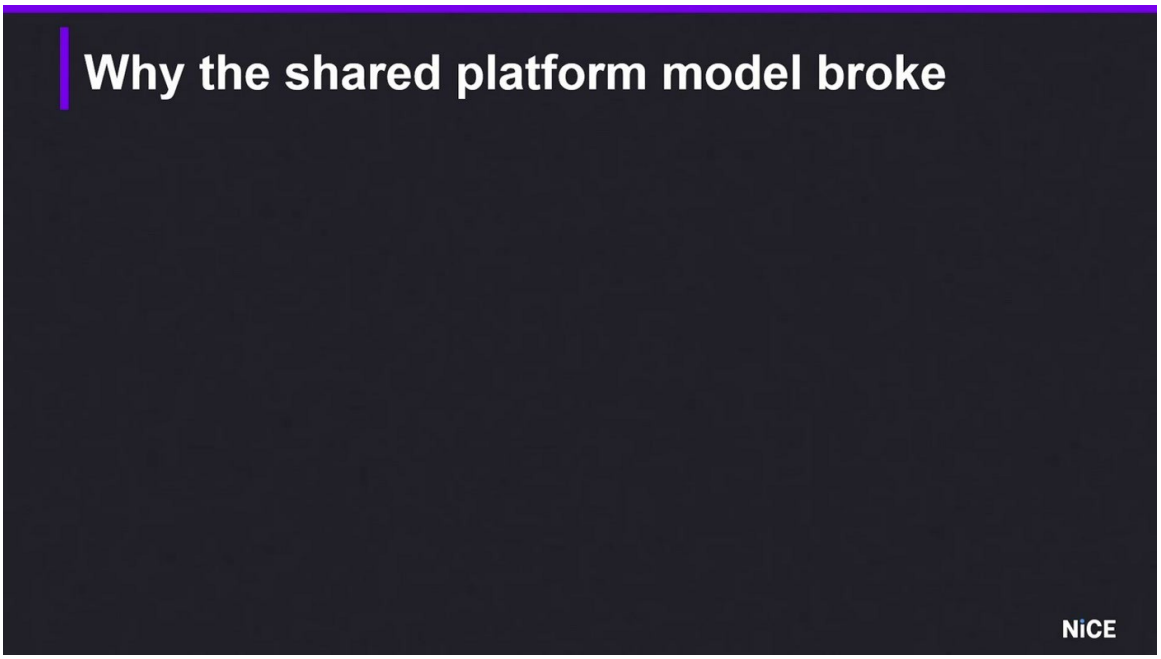
הפיתוי שהוזכר במפורש: אם כבר יש שכבה שכולם עוברים דרכה, אולי נשים בה גם אותנטיקציה, ואם כבר אותנטיקציה אז גם אוטוריזציה. האזהרה: "אם הוא לא עובד, אז בעצם פספסנו את כל הרעיון של צמצום של Blast Radius" [21:45].

מנגנון	ב-AWS	יתרון	מגבלה
DNS	Amazon Route 53	latency נמוך מאוד, very well established, 100% SLA [23:17]	מורכבות ניתוב מוגבלת; מחייב sub-domains
Edge	CloudFront + CloudFront Functions	לוגיקה מורכבת יותר על דומיין יחיד [23:17]	latency נוסף — נמוך אבל לא אפס, אין caching של resolving [24:48]
Gateway	API Gateway + DynamoDB כ-backing store	גישה ל-headers ול-body, throttling ו-rate limiting [24:48]	הכי הרבה לוגיקה בשכבה שאמורה להיות דקה

ההמלצה חד-משמעית: "אם אפשר להשתמש ב-DNS, תשתמשו ב-[23:17] "DNS, ולצידה הטענה ש-"ראוט 53 נותן SLA 100%, אי אפשר לנצח את זה" [23:17]. הדוגמה שניתנה היא חברה שנותנת לכל לקוח — sub-domain enterprise A, enterprise B — וממפה ממנו לתא הרלוונטי. צוין ש-AWS עצמה משתמשת בדפוס הזה: כמעט לכל URL של שירות יש prefix שמכוון למשאב, ולעיתים קרובות מאחורי הקלעים גם לתא ספציפי [23:17].

לבסוף, שלוש הגישות אינן בלעדיות ולקוחות משלבים ביניהן — אבל אסור לשכפל לוגיקת ניתוב בין שכבות, כך שגם ה-DNS וגם ה-gateway מנתבים ומשמשים fallback זה לזה. "כל שכבה שתעשה את הדבר האחד שהיא מאוד טובה בו" [26:25].

8. NICE: למה המודל המשותף נשבר



סדר הגודל של הפלטפורמה כפי שהוצג על השקף: 27K לקוחות ב-150 מדינות, מעל 85% מ-Fortune 100, 20B אינטראקציות בשנה

החצי השני של הסשן הוא מקרה בוחן. NICE מספקת פתרונות לעולם מרכזי השירות: הקלטה של כל התעבורה, ניתוח שלה, ופתרונות AI ואנליטיקה מעליה — הכול ב-real time [26:25]. המשמעות התפעולית: "כל ירידה, האותקה להכי קטנה, פשוט מונעת מהם לטפל ולנהל את העבורה" [27:57], ובמקרה של לקוחות מפוקחים, אי-הקלטה היא גם עניין רגולטורי.

המסע לענן התחיל ב-2018 בהחלטה לכתוב את כל המערכת מחדש, cloud native, על AWS. הצמיחה הייתה מהירה: ב-2025 הפלטפורמה הגיעה ל-3,000 ומעלה לקוחות ו-16 אזורים [27:57]. ואז — "הסקל העצום הזה שהיה יתרון מאוד חזק שלנו באיזשהו נקודה הפך גם לריסק מאוד גדול" [27:57].

ההנחיה שנוסחה פנימית כמדיניות ארכיטקטורה

ארבעת מקורות הכאב

- **Hard limits של חשבון בודד.** המערכת נולדה מעל AWS account אחד, עם המגבלות שלו — למשל IAM roles. "מתישהו פגשנו את ה-Limits האלה? המון פעמים" [27:57]. הגדלות עזרו, אבל המגבלה נשארה constraint.
- **היקף הפגיעה בכשל.** חמש דקות של תקלה מתורגמות ל-"160 אלף שיחות לא מוקלטות" [29:33]. לקוחות מחויבים ברגולציה, מודדים אחוזי הקלטה, וסופגים קנסות.
- **דרישות של לקוחות אנטרפרייז.** בידוד מדיירים אחרים, וימים שבהם אסור לגעת במערכת — לקוחות מסגמנט הקמעונאות לא רוצים שיגעו במערכת לקראת Cyber Monday או Christmas [29:33].
- **Noisy neighbors.** במערכת בסדר גודל כזה תמיד יש דיירים שמייצרים לחץ על משאב ופוגעים באחרים [29:33].

המסקנה שנוסחה: אי אפשר להשקיע רק בווקטור אחד — resilience של המערכת הקיימת. "אנחנו חייבים לייצר וקטור מקביל שבי דיזיין" שיקטין את ההשפעה של תקלה שתקרה במערכת [29:33].

9. איך שכנעו את הארגון

זה אולי החלק המקורי ביותר בהרצאה. הטענה: ה-R&D והארכיטקטים הבינו את הצורך, אבל "ללכת לשכנע ארגון שישקיע בכזאת ארכיטקטורה וכזאת אימפלמנטציה זה לא טריוויאלי" [31:04], ודיאגרמות ארכיטקטוניות לא עושות את העבודה מול מקבלי ההחלטות. לכן אספו data points מפרודקשן והפכו אותם ל-business case.

מה מדדו	מה מצאו (לפני)	מקור
Blast radius לאירוע	אינסידנטים שמאות לקוחות הושפעו מהם	[31:04]
Throttling	בממוצע כ-20 אירועים בחודש	[31:04]
ימי מורטוריום (הקפאת הפצות)	מעל חמישים ימים בשנה — אזור שלם לא מקבל עדכונים	[31:04]
קרדיטים ללקוחות בעקבות תקלות	סכום לא מבוטל	[32:34]
סקרי שביעות רצון	15% מהתגובות נגעו ביציבות המערכת	[32:34]

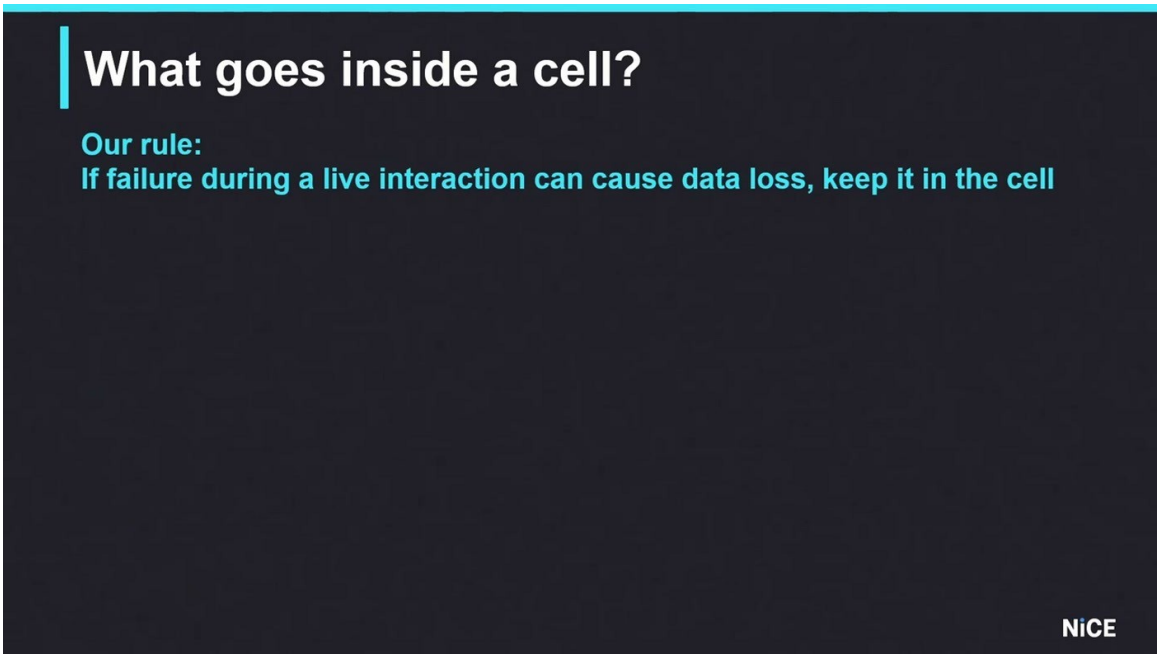
המסגור שעבד "כל הדברים האלה הצליחו לעזור לנו לשכנע את הארגון שמדובר פה בהשקעה, בהפחתה של ריסק עסקי. זה לא השקעה שהיא רק השקעת השתיתית" [32:34]. אחרי אישור הארגון התחיל מסע של שנה וחצי [32:34].

10. מה נכנס לתא, ואיך הועברו 3,000 לקוחות

10.1 בחירת ה-workload

ההחלטה הראשונה הייתה לא להתחיל מהקצוות. NICE בחרה דווקא את ה-workload הקריטי ביותר ללקוחות — כל מה שקשור בהקלטת התעבורה מלקוח הקצה למרכז השירות — "בחרנו ב-workload הזה כי רצינו לייצר אימפקט, לא רצינו לקחת משהו צטידי שבסוף הלקוחות והארגון לא היה חווים איזשהו ולום משמעותי" [32:34].

מה שרצו לקבל מהתא: בידוד כשלים כך שתקלה לא תזלוג לתאים אחרים; שליטה מלאה פר תא ב-deployment וב-observability; scale אופקי — להתרחב על ידי הוספת תאים ולא על ידי הגדלת תא; והפחתת shared state bottlenecks [34:07].



הכלל הבינארי שהכריע לכל שירות אם מקומו בתא או בשכבה הגלובלית

> **— [34:07]** החוק אמר שאם השירות, אותו מייקרוסרוויס, אותו סרוויסס, למדה, היא יכולה לסכן הקלטה, היא תהיה בתוך הסל

לפי הכלל הזה נכנסו לתא כל הטיפול ב-audio records, טיפול במדיה, וה-real time state של ההקלטה. מחוץ לתא נשארו שירותים גלובליים — למשל אותנטיקציה, או עיבוד אודיו שקורה אחרי ההקלטה [34:07]. הכלל הזה הוא גם מה שממשיך להכריע בעיצוב של שירותים חדשים היום.

10.2 ניתוב: tenant ID כ-sub-domain

מפת הניתוב: שירות קונפיגורציה שמזין Routing DB, ולמבדה שמעדכנת את המיפוי מ-tenant ל-URL פנימי

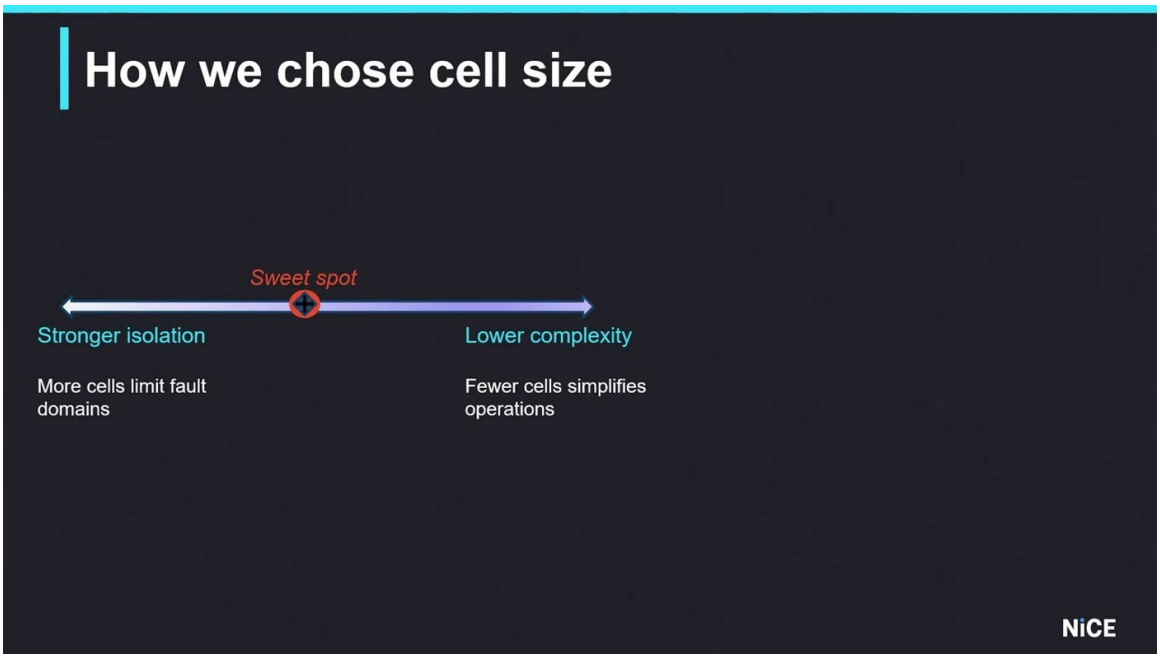
נעשו שינויי קוד שאיפשרו להכניס את ה-tenant ID כ-sub-domain לכל URL שמגיע למערכת, ומכאן אפשר היה להשתמש ב-Route 53 כראוטר [35:39]. מאחורי הקלעים יושבת למבדה על מערכת הקונפיגורציה: כשמתווסף תא או דייר חדש, היא מעדכנת את ה-routing map בכניסה המתאימה.

ההחלטה החשובה למעבר הדרגתי הייתה ה-default cell.

> **[35:39]** — ה-default cell זה היה איפה שכל הלקוחות שעוד לא עברו לצורת סלים עדיין ישבו

כך אפשר היה להעביר לקוחות קיימים לתאים באופן אינקרמנטלי, ובמקביל לנתב לקוחות חדשים ישירות לתאים החדשים [35:39]. חשוב לשים לב לצד השני של המטבע: כל עוד רוב הלקוחות יושבים ב-default cell, הסיכון המקורי עדיין קיים — נקודה שאורן חוזר אליה בלקחים.

10.3 גודל התא



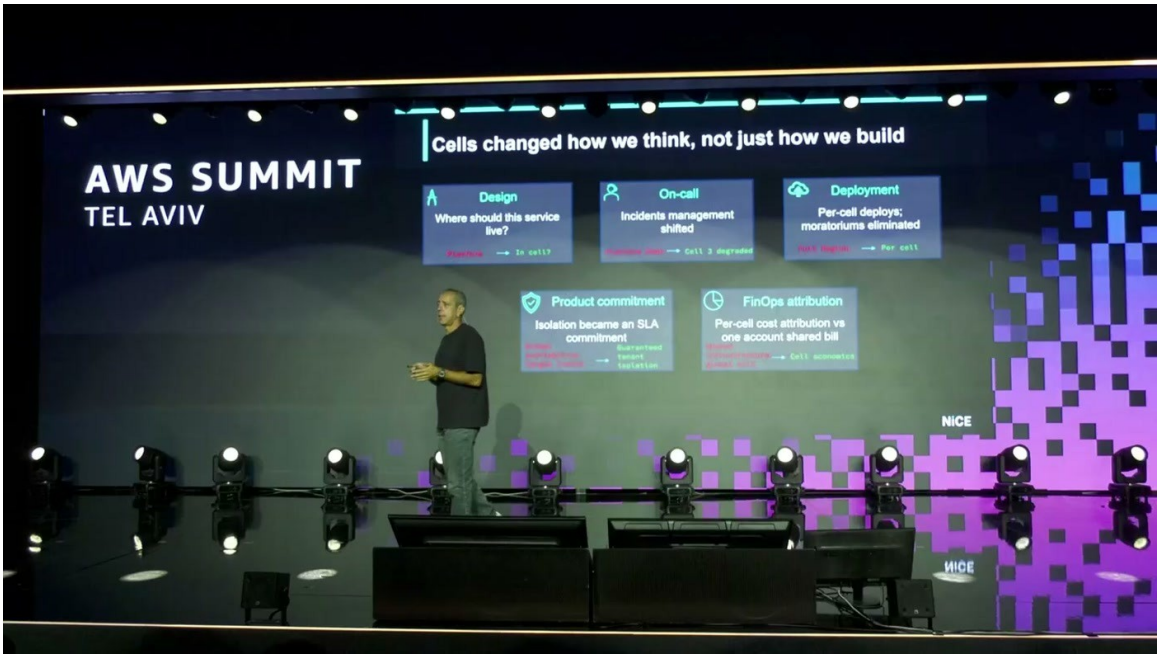
ה-sweet spot בין בידוד חזק יותר (יותר תאים) לפשטות תפעולית (פחות תאים)

כאן נסגר הטיזר מפרק 4. המטריקה שנבחרה אינה מספר דיירים אלא עומס רגעי.

> **— [37:11]** להגדיר איזושהי מטריקה, שאומרת שסל מסוים לא יטפל ביותר מעשרת אלפים הקלטות בו זמנית

שתי התוצאות שהוזכרו: ראשית, רוב התאים יצאו מאוזנים מבחינת compute בלי תלות במספר הדיירים שבהם. שנית, ערך הסף נבחר כך שלא יהיה צורך לפצל דייר בין כמה תאים — מורכבות שרצו להימנע ממנה [37:11]. לצד זה הודגש שלכל תא יש overhead בעלות, כי רכיבי availability משוכפלים בתוכו, ושה-operational surface מתרחב [37:11].

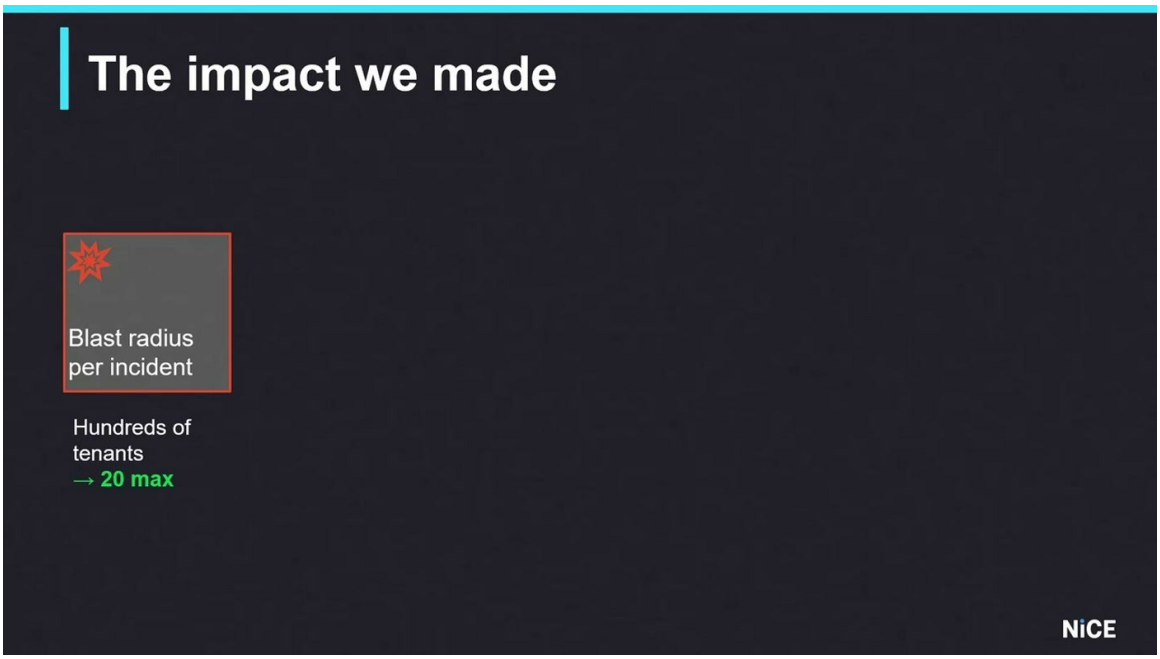
11. מה זה שינה — בארגון ובמספרים



חמישה תחומים שבהם השפה הארגונית השתנתה: עיצוב, on-call, deployment, התחייבות מוצר, ו-FinOps

הטענה: המעבר לתאים אינו פרויקט הנדסי בלבד אלא שינוי תרבותי [38:48]. חמש הזירות שהוצגו:

- **עיצוב.** כל שירות חדש או פונקציונליות חדשה מתחילים בשיח על מה יישב בתא ומה לא [38:48].
- **On-call.** שיחת פרודקשן מהתמיכה כבר לא שואלת מה קורה בפלטפורמה, אלא איזה תא נפגע — ונכנסים ל-observability של אותו תא, ולפעמים לשכבה שמעליו [38:48].
- **Deployment.** ימי הקפאה כבר לא מקפיאים אזור שלם; הם יכולים להיות ספציפיים לתא בודד או לכמה תאים [38:48].
- **התחייבות מוצר.** בעבר, כשלקוחות גדולים ביקשו התחייבות לבידוד, "זה היה יותר איזשהו "wishful thinking [40:18]. היום אפשר לשים לקוח כזה בתא משלו — כשירות פרימיום שמתומחר בהתאם.
- **FinOps.** לתא יש attributes של עלות, מה שמאפשר חיתוך עלות פר תא במקום מבט כולל על כל המערכת [40:18].

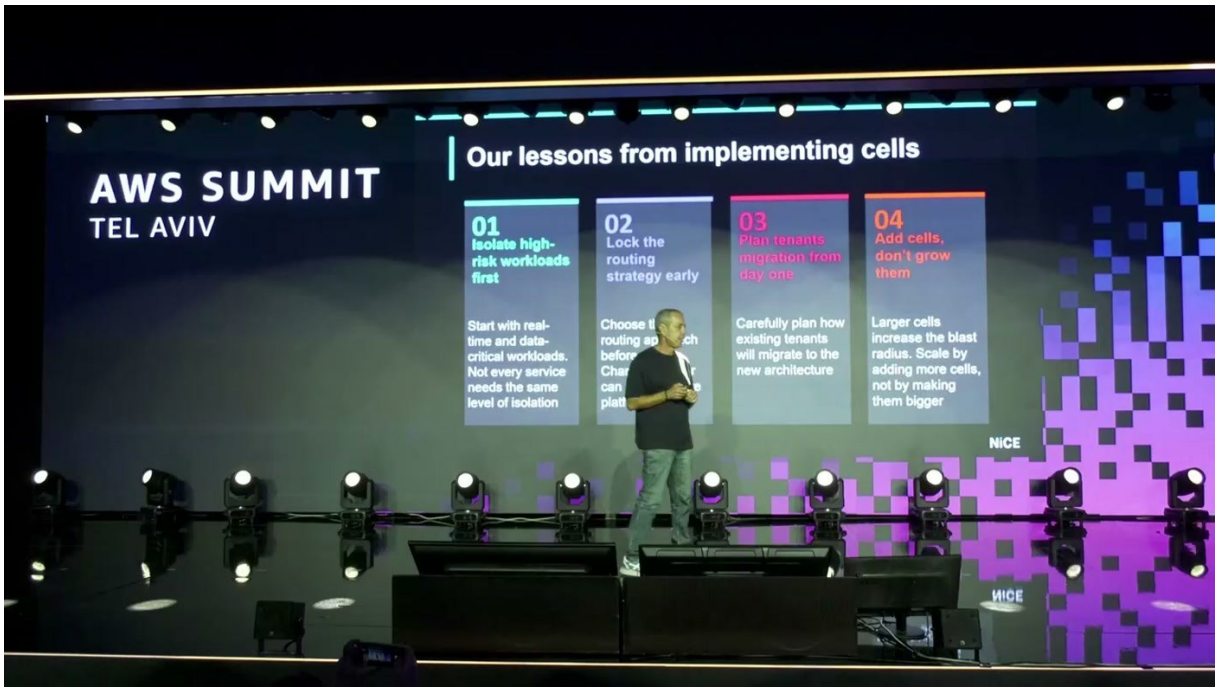


השקף שפתח את סקירת התוצאות — רדיוס הפגיעה לאירוע ירד ממאות דיירים לעשרים לכל היותר

מדד	לפני	אחרי	מקור
Blast radius לאירוע	מאות דיירים	כ-20 לכל היותר	[40:18]
אירועי throttling	כ-20 בחודש	"כמעט ואין לנו כאלו"	[40:18]
ימים ללא אפשרות deployment	מעל 50 בשנה	בודדים, ולא עוצרים אזור שלם אלא תאים	[40:18]
קרדיטים ללקוחות	—	ירידה של 90%	[40:18]
תגובות על יציבות בסקרים	15%	פחות מ-2%	[41:51]

> **— [41:51]** אז כמו שאתם מבינים, כשכזה אימפקט נוצר, בוודאות אני יושן יותר טוב בלילה

12. מה לוקחים מכאן



ארבעת הלקחים שסיכמו את חלק המקרה הבוחן

- **התחילו מה-workload בעל הסיכון הגבוה, לא מהקל.** "לקחת workload שהוא קריטי, כי רצינו לייצר איזשהו אימפקט" [41:51] — אבל להעביר אותו לפרודקשן בזהירות, על מעט דיירים ותאים בודדים, ורק אז להתרחב.
- **נעלו את אסטרטגיית הניתוב מוקדם.** "ראוטינג זה החלטה סופר חשובה, קשה לשנות אותה אחרי שמתחילים במסע הזה" [41:51]. ב-NICE נעשו כמה POCs לפני שנבחר Route 53.
- **תכננו את הגירת הדיירים הקיימים מיום ראשון.** "אפילו אם יש לך סלים אם כל שאר המערכת נמצאת בסל אחד בודד הריסק קיים" [43:25]. ה-default cell הוא גשר, לא יעד.
- **הוסיפו תאים, אל תגדילו אותם.** "יש איזשהו bad habit כזה שכשיש סל, אולי נמשיך לגדול בתוך הסל, לפני שנעביר ונייצר סל חדש" [43:25]. הכלל: כשהמטריקה נחצית — עוברים לתא הבא.
- **הכינו את ה-business case במספרים, לא בדיאגרמות.** חמשת המדדים שאספו ב-NICE (פרק 9) הם תבנית שאפשר להעתיק לכל ארגון ששוקל מהלך דומה.
- **הכניסו cell ID לטלמטריה עוד לפני שיש תאים.** זה שלב מינוס-אחד [15:35] — זול לעשות מראש ויקר להשלים בדיעבד.

שאלות פתוחות למי ששוקל את המהלך

- מהי המטריקה שמייצגת אצלנו השפעה עסקית, ואיזה סף עליה אנחנו מוכנים להתחייב אליו?
- כמה זמן לוקח לנו היום להקים תא חדש — וכמה מזה הוא תהליך ארגוני ולא טכני?
- אילו שירותים אצלנו יכולים לסכן את הפעולה הקריטית, לפי כלל ההכרעה של NICE?
- האם שכבת הניתוב שלנו יכולה להסתפק ב-DNS, או שיש דרישה שמחייבת דומיין יחיד?
- האם מערכת הניטור שלנו יוצרת shared fate חדש בין התאים?

נספח: מילון מונחים

מונח	פירוש כפי שהוצג בסשן
Cell (תא)	יחידה עצמאית ומלאה של המערכת — load balancer, compute ו-storage משלה — בלי תלויות בתאים אחרים [4:35]
Blast radius	היקף הנזק כשמשהו משתבש — כמה לקוחות מושפעים ובאיזה אחוז מכלל הלקוחות [1:30, 18:39]

מונח	פירוש כפי שהוצג בסשן
Cell router	השכבה הדקה שמבצעת discovery וניתוב לקוחות לתא הנכון; single point of failure מעצם הגדרתה [21:45]
Default cell	התא שבו יושבים כל הלקוחות שטרם הועברו לארכיטקטורת התאים — גשר להעברה אינקרמנטלית [35:39]
Noisy neighbor	דייר שצורך משאבים בצורה שפוגעת בדיירים אחרים בסביבה משותפת [1:30, 29:33]
Fractional deployment	הפצה שמתחילה בתא אחד ומתרחבת למקבילים גדלים, עם בדיקה של לפחות תא אחד בכל אזור [12:22]
Bake time	זמן ההמתנה והאימות בין שלבי הפצה [10:48]
Composite alarm	אלרט שמחבר כמה מטריקות ללוגיקה אחת, כדי לצמצם סופות אלרטים ולזהות תקלות cross-cutting [17:07]
Shared fate	מצב שבו רכיב משותף גורם לכך שכשל אחד מפיל את כולם — מה שתאים נועדו למנוע [14:00]
Moratorium days	ימים שבהם לקוחות אוסרים כל שינוי בפרודקשן, למשל לקראת Cyber Monday [29:33, 31:04]