Karpenter ו-EKS Auto Mode: מי מנהל לכם את הנודים

AWS Summit Tel Aviv 2026 ,סיכום סשן — TLV-SVS301

Dima Breydo, Principal SA, AWS · Eliran Mesika, Senior SA, AWS · Fadi Tuma, Senior DevOps Engineer, Bluevine

10 בספטמבר 2026 | משך: כ-37 דקות | הופק מהקלטת הסשן

מסלול: AWS for Containers and Platform Engineering

על המסמך הזה

המסמך מסכם את הסשן TLV-SVS301 מתוך AWS Summit Tel Aviv 2026, בהתבסס על ההקלטה המלאה. זה הסשן הפותח של מסלול הקונטיינרים, והוא בנוי משלושה חלקים רצופים: צלילה טכנית ל-Karpenter, הצגה של EKS Auto Mode עם דמו מוקלט, וסיפור לקוח מחברת Bluevine. המסמך נבנה מתמלול אוטומטי של ההקלטה ומהסליידים שחולצו מהווידאו, כך שאפשר לחזור לתוכן בלי לצפות שוב. חותמות הזמן בגוף המסמך מפנות לנקודה המדויקת בהקלטה.

איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. מה הוצג ומה שווה לבדוק אצלנו.
- **פרקים 2–7 — Karpenter לעומק:** consolidation, שני ה-CRDs, מימדי הגמישות, אסטרטגיות NodePool ופיצ'רים חדשים.
- **פרקים 8–9 — EKS Auto Mode:** מה עובר לאחריות AWS, ודמו מלא של הקמת קלאסטר והרצת LLM inference.
- **פרק 10 — Bluevine:** סיפור לקוח מפינטק ישראלי, כולל המספרים שהם מדדו.
- **פרקים 12–13 — סיכום:** מה לוקחים מכאן, ומונחון מונחים.

> **הערת מקור** המסמך הופק מתמלול אוטומטי של ההקלטה. התמלול מדויק בקטעים המקצועיים, אך שמות ומונחים לועזיים מופיעים בו בשיבוש פונטי והם נורמלו ידנית מול הסליידים. הציטוטים אומתו מול ההקלטה בחותמת הזמן המצוינת. מספרים שמופיעים על הסליידים צוינו ככאלה, ולא הוסקו מהדיבור.

שקופית הפתיחה עם שלושת הדוברים ותפקידיהם

1. תקציר מנהלים

הסשן עוסק בשאלה אחת: מי מחליט אילו מכונות ירוצו מתחת לקלאסטר Kubernetes שלכם, מתי, ובאיזה מחיר. התשובה נבנית בשתי שכבות — Karpenter כמנוע החלטה על קפסיטי, ו-EKS Auto Mode כעטיפה מנוהלת שמריצה את Karpenter ואת שאר ה-add-ons בשבילכם.

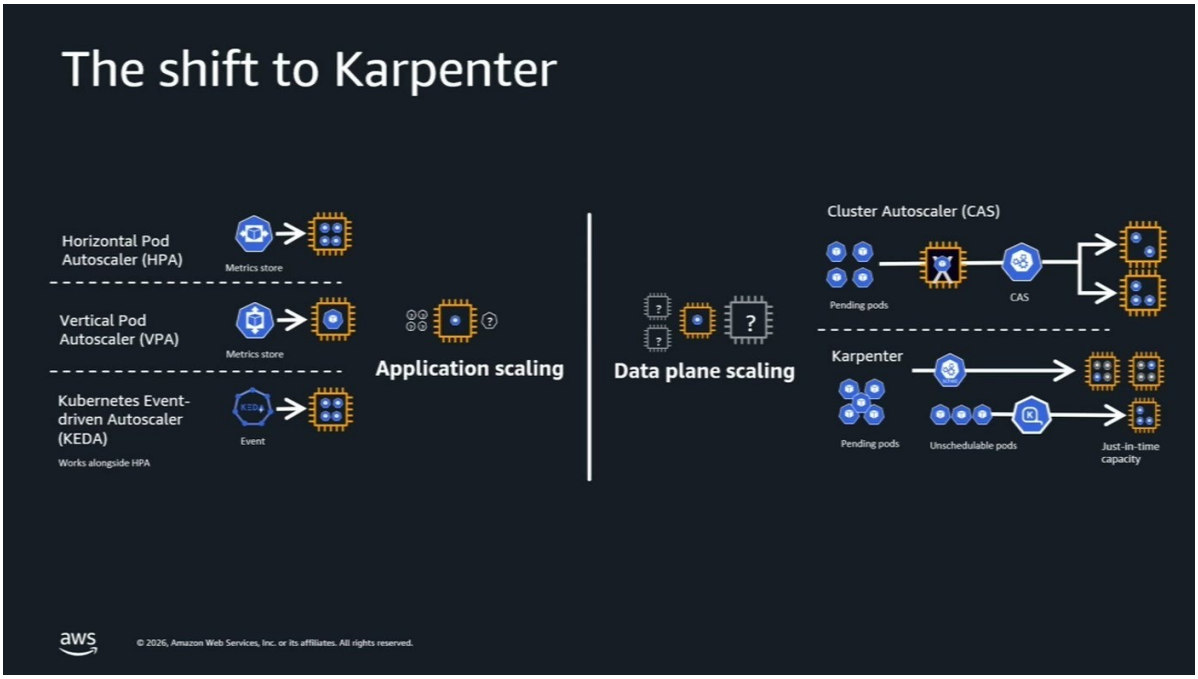
- **Karpenter הוא groupless, וזה כל ההבדל.** בניגוד ל-Cluster Autoscaler שעובד מול Auto Scaling Groups, Karpenter מסתכל ישירות על פודים ב-Pending, מבצע עליהם bin packing, ומרים את הקפסיטי המינימלי הנדרש. "אין לו אוטוסקיילינג גרופס, הוא גרופלס" [1:31].
- **האופטימיזציה רצה כל הזמן, לא רק בעלייה.** שלושה מנגנונים — הורדת נודים ריקים, איחוד פודים מנודים underutilized, והחלפת נודים בזולים יותר — מקובצים תחת consolidation, ונשלטים דרך consolidation policy אחת מתוך שלוש [1:31].
- **הגמישות שאתם נותנים ל-Karpenter היא המטבע.** ככל שה-NodePool רחב יותר במשפחות, ארכיטקטורות, AZ ו-capacity types — כך ה-bin packing יעיל יותר. "ככל שתיתנו יותר חופש לקרפנטר ככה הוא יהיה יותר יעיל בבינט פקינג שלו" [4:40].
- **EKS Auto Mode מעביר את כל הדאטה פליין לאחריות AWS.** לא רק הקומפיוט וה-autoscaling, אלא גם add-ons קריטיים כמו VPC CNI, CoreDNS ו-EBS CSI Driver, דרייברים של NVIDIA, ושדרוגי נודים ו-security patches [19:57].
- **הדמו הוכיח את הטענה מקצה לקצה.** קלאסטר Auto Mode חדש, NodePool ל-GPU על Spot, ופריסה של מודל Qwen3 בגודל 35 מיליארד פרמטרים — בלי התקנת דרייבר אחד. "לא התקנתי פה דרייברים" [23:03].
- **Bluevine מדדו את זה בפרודקשן.** כ-60% פחות operational overhead, כ-75% קיצור בזמן הקמת סביבה חדשה, ו-100% הפחתה בעיסוק בשדרוגי נודים ו-security patches [33:50], [35:20].

רלוונטיות לארגון

שני הקווים המעשיים שיוצאים מכאן הם עלות ותחזוקה. בצד העלות — Graviton, Spot ו-consolidation אגרסיבי הם שלושה מנופים שאפשר להפעיל על קלאסטר קיים בלי לגעת באפליקציה, בתנאי שה-images הם multi-architecture. בצד התחזוקה — Auto Mode מוריד שכבה שלמה של עבודת תשתית, אבל גם מוריד שליטה: מי שצריך AMI מותאם, agents משלו על הנוד, או שליטה מלאה בתזמון השדרוגים, צריך לבדוק את זה לפני שהוא זז.

2. נקודת הפתיחה: סקיילינג של הדאטה פליין

Dima Breydo, Principal Solutions Architect ב-AWS, פתח בתמונה המוכרת: יש לכם קלאסטר EKS, אתם מריצים עליו workloads, וכנראה כבר עושים להם auto scaling ברמת האפליקציה — HPA או KEDA. אבל יש גם את הצד של המכונות. "וכאן היסטורית היינו משתמשים בקלאסטר אוטוסקיילר שעובד עם אוטוסקיילינג גרופס" [0:00].



משמאל — סקיילינג ברמת האפליקציה (HPA, VPA, KEDA); מימין — סקיילינג של הדאטה פליין, מול Cluster Autoscaler Karpenter

ההבדל המהותי הוא בהנחת היסוד. Cluster Autoscaler עובד מול Auto Scaling Groups, ולכן מניח שכל המכונות בקבוצה זהות. Karpenter לא: "אין לו אוטוסקיילינג גרופס, הוא גרופלס. זה אומר שאין לו את ההנחה הזאת שכל המכונות באותה קבוצה זהים" [1:31]. במקום זה הוא מסתכל על הפודים שנמצאים כרגע ב-Pending, מבצע עליהם bin packing ואגרגציה של CPU וזיכרון, ומביא את הקפסיטי המינימלי הנדרש להרצת ה-workload — בשניות.

3. Consolidation: האופטימיזציה שלא נעצרת

Karpenter לא פועל רק ברגע שצריך להעלות מכונות. הוא רץ ברקע כל הזמן. הדוגמה שהוצגה: שני פודים קטנים, כל אחד יושב על מכונה גדולה מסוג m7g.xlarge. שתי פעולות אפשריות — לאחד את שני הפודים על מכונה אחת, ולהחליף את המכונה בזולה יותר. Karpenter עושה את שתיהן [1:31].

מאחורי זה שלושה מנגנונים: הראשון מוריד נודים ריקים (עד כדי ה-DaemonSets), השני מזיז פודים מנודים שהם underutilized, והשלישי מחליף נודים בנודים זולים יותר. "שלושת המנגנונים האלו ביחד נקראים "consolidation" [1:31].

> **כלל אצבע מהבמה** "כלל אצבע תשתמשו ב-PDBs ב-POD Disruption Budgets, כי בעצם על מנת שלא כל ה-workload ירד בבת אחד וימשיך לשרת את הטרפיק, אבל מצד שני גם אל תעשו את ה-PDBs האלו יותר מדי ריסטריקטיב, כי זה עלול באופן כללי לעצור את האופטימיזיישן" [3:06].

שלוש מדיניויות consolidation

Consolidation policy	התנהגות	מתי מתאים
WhenEmpty	פועל אך ורק על נודים ריקים עד כדי DaemonSets	StatefulSets ומקומות שבהם לא רוצים workload ל-disruption
WhenEmptyOrUnderutilized	אגרסיבי — מזיז פודים ממקום למקום כל הזמן לטובת אופטימיזציה	workloads stateless שסובלים העברות

Consolidation policy	התנהגות	מתי מתאים
Balanced	כמו הקודם, אך מוסיף שיקול כדאיות: האם בכלל שווה לעשות disruption עכשיו	ברירת מחדל סבירה לרוב הסביבות

4. שני ה-CRDs: EC2NodeClass ו-NodePool

Karpenter מרחיב את Kubernetes בשני resources, וההפרדה ביניהם היא הפרדה בין "איך המכונה נראית ב-AWS" לבין "אילו מכונות מותר לבחור".

EC2NodeClass — ההגדרות ה-AWSיות

כאן מגדירים subnets, security groups, user data, tags, דיסק ו-AMI [3:06]. לגבי ה-AMI הוצגה המלצה חדה: אפשר לנעול גרסה מסוימת (pinning) — "וזה בעצם מה שמומלץ לפרודקשן" — או להגדיר @latest, ואז פיצ'ר בשם drift מזהה שיצאה גרסה חדשה ומגלגל את המכונות בהתאם [3:06].

NodePool — מרחב האפשרויות

ה-NodePool מגדיר את כל הגמישות: אילו משפחות מכונות, אילו דורות, מה להחריג ומה לכלול. הכללה מפורשת של instance types ספציפיים פחות מומלצת, "כי בעצם זה אומר שתצטרכו כל הזמן לנהל רשימה" [4:40].

> **Dima Breydo — [4:40]** למעשה יש פה איזשהו טרייד אוף בין חופש לשליטה. ככל שתיתנו יותר חופש לקרפנטר ככה הוא יהיה יותר יעיל בבינט פקינג שלו, במיוחד אם אתם עובדים עם ספוטים — תיתנו כמה שיותר אפשרויות.

הצד השני של אותו trade-off: לפעמים באמת צריך סוג מסוים של GPU — instance, או דרישת licensing שמחייבת מכונות ספציפיות. אז מגבילים, מקבלים פחות יעילות ויותר שליטה [4:40].

5. מימדי הגמישות

Availability Zones

Karpenter תומך בהגדרת AZ. ה-use case הבסיסי הוא EBS: "Elastic Block Storage הוא זונל, וקרפנטר בעצם ידע להרים את המכונה באותו מקום, באותו Availability Zone שבו נמצא הדיסק" [4:40]. ה-use case השני הוא ההפוך — פיזור מכוון על פני AZ לצורכי זמינות גבוהה [6:11].

ארכיטקטורות: x86 מול Graviton

לצד x86 ו-AMD קיימים מכונות Graviton בארכיטקטורת ARM, "שהם ככה עד 40% טובים יותר מבחינת price performance" [6:11] — כלומר זולים יותר, ולכן אם תכללו אותם ב-NodePool הם ייבחרו כשהם זמינים. התנאי הטכני: multi-architecture builds, כלומר images שכוללים גם ARM וגם x86.

> **מלכודת** "שימו לב, כל ה-workload צריך לדעת לרוץ ב-ARM" [6:11] — מספיק DaemonSet אחד שלא נתמך, וכל הקלאסטר נגרר חזרה ל-x86.

Capacity types

- **Spot בלבד או On-Demand בלבד.** ההגדרה הפשוטה.
- **Fallback to On-Demand.** מגדירים את שניהם: ה-workload ירוץ על Spot, ואם אין Spot או שהוא נלקח — יעבור ל-On-Demand. בחישוב הבא של Karpenter, אם Spot שוב זמין, ה-workload חוזר אליו [6:11].
- **Spot-to-Spot consolidation.** קיים אבל כבוי by default; צריך להפעיל. כשמופעל, Karpenter יחליף Spot ב-Spot זול יותר [7:43].
- **Reserved capacity.** משמש לצד Spot ו/או On-Demand כשיש קפסיטי מוקצה מראש — ODCR עם Reservation ID, או Capacity Blocks for ML למכונות P לחלון של עד שבועיים לתהליכי אימון [7:43].

לגבי הפסקת Spot: "אתם מקבלים התראה של שתי דקות", ובזמן הזה נעשה cordoning ו-spot interruption handler מטפל בפינוי — "אבל בזמן הזה קרפנטר כבר יביא לנו מכונה חדשה, ברוב המקרים זה ייקח פחות משתי דקות" [7:43].

6. Supply side מול demand side

המודל המנטלי שהוצג להחלטה "איפה ה-workload ירוץ" מורכב משני צדדים שחייבים להסכים. ה-supply side הוא ה-NodePool — היקום של כל האפשרויות שהגדרתם: משפחות, AZ, ארכיטקטורות, capacity types. ה-demand side הוא הפוד עצמו, שמשתמש בקונסטרקטים סטנדרטיים של Kubernetes: nodeSelector, affinity, taints ו-tolerations, topology spread constraints ועוד [9:18].

> **Dima Breydo — [9:18]** ועל מנת שפוד מסוים יגיע למכונה מסוימת, שני הצדדים — ה-supply side, ה-NodePool, וגם ה-demand side — צריכים להסכים. וקרפנטר הוא סוג של דבק ביניהם.

הדבק עובד כך: Karpenter שם על הנודים well-known labels — למשל architecture או capacity-type — והפוד יכול להתייחס אליהן ב-nodeSelector שלו [9:18].

7. אסטרטגיות NodePool ופיצ'רים חדשים

אסטרטגיה 1 — NodePool יחיד ורחב

"נוט פול יחיד, זאת הדרך המומלצת להתחיל איתה, ככה בעצם רוב הלקוחות שלנו מתחילים" [9:18]. מגדירים NodePool אחד רחב שמשרת את כל הצוותים — כמה משפחות, multi-AZ, multi-architectural — "ואז אתם מקבלים איזי אופריישנס... בהרבה מקרים זה מספיק" [9:18].

אסטרטגיה 2 — NodePool נוסף עם taints

כשזה לא מספיק — GPU, AMI אחר, או דרישת tenant separation ב-multi-tenancy — מוסיפים NodePool. ההמלצה: להשתמש ב-taints ו-tolerations ספציפית עבור ה-NodePool הנוסף, כך שרק הפודים שמתכוונים להגיע אליו יגיעו, וכל השאר ימשיכו כברירת מחדל ל-NodePool הרחב [10:49].

אסטרטגיה 3 — Weighted NodePools

משקל הוא שדה שמגדירים על NodePool, וכשיותר מ-NodePool אחד מתאים לפוד — זה עם המשקל הגבוה מנצח. ה-use case שהוצג: Savings Plan או RI. יש קומיטמנט על instance types מסוימים, מגדירים NodePool עם אותם types ועם limit בגודל הקומיטמנט, ואז "קודם כל אתם תנצלו אותו, ואחרי שהוא יגיע ל-limit כל השאר בעצם ילך לנוטפולים האחרים שלכם" [10:49].

Baseline Capacity

פיצ'ר חדש יחסית, למי שרוצה כמות קבועה של נודים שרצה כל הזמן — workload בגודל סטטי שלא רוצים לחכות בו ל-cold start. בדוגמה שהוצגה: 20 מכונות, עם limit של 25. חשוב להבין את התפקיד של ה-limit: "זה לא אומר שהנוטפול יגדל דינמית, הוא לא יגדל דינמית, הוא יהיה בגודל 20" — ה-limit קיים כ-headroom ל-Karpenter כשהוא צריך לבצע החלפות [12:20]. מי שרוצה גם גדילה דינמית — שני NodePools שעובדים יחד.

Capacity Buffer (alpha)

פיצ'ר שנוסף בחצי השנה האחרונה ונמצא ב-alpha; זו ספסיפיקציה של Kubernetes ש-Karpenter מממש. ה-use case הוא cold start. במקום להחזיק כמות קבועה של נודים, מגדירים buffer שממומש על ידי פודים וירטואליים: "אלו לא פודים אמיתיים, אלו פודים שלא קיימים בשום מקום חוץ מבחישובים של קרפנטר" [13:50]. אפשר להגדיר אותו באבסולוטי או כאחוז מה-workload הראשי — למשל 10%. ברגע שה-workload גדל, יש כבר מקום פנוי, ובמקביל Karpenter מביא קפסיטי נוסף לפודים הווירטואליים הבאים [13:50].

> **אזהרה מהבמה** "שימו לב, Node Overlay — פיצ'ר מאוד מאוד מתקדם. אל תשתמשו בו אם יש לכם דרך אחרת לפתור את מה שאתם רוצים לפתור, אבל יש הרבה מצבים שבהם זה כן מאוד מאוד מועיל" [15:24].

Node Overlay (alpha)

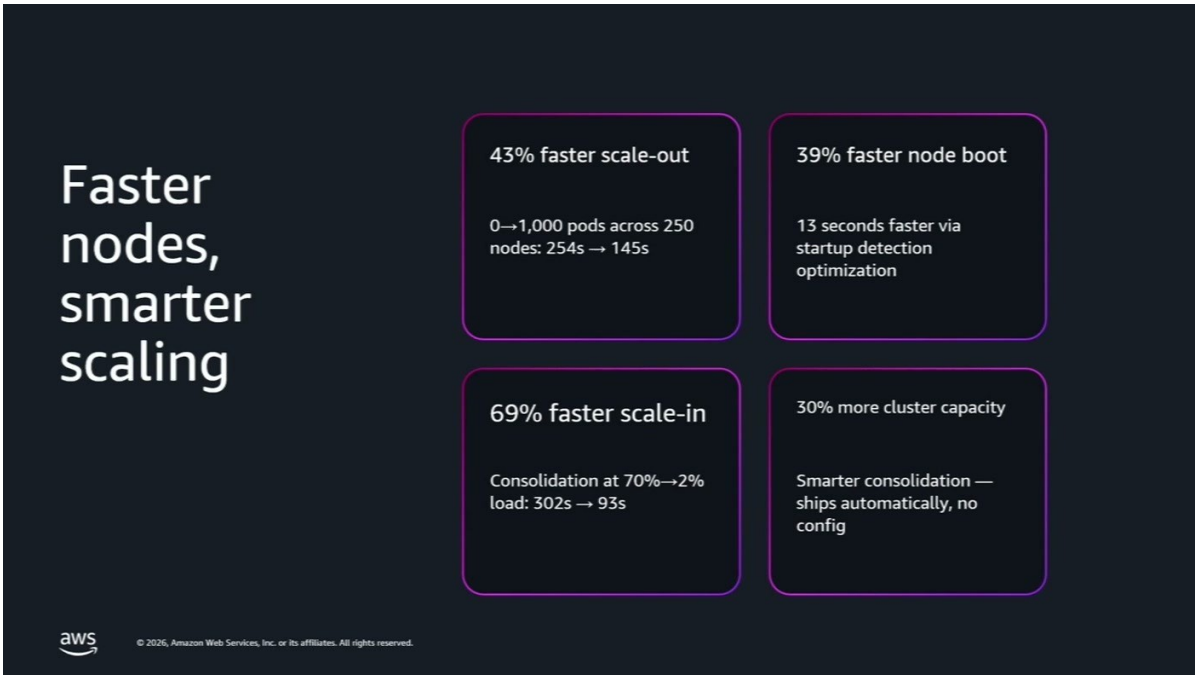
פיצ'ר מתקדם שמאפשר להזין ל-Karpenter מידע שהוא לא מכיר by default. ה-use case שהודגם הוא price adjustment. Karpenter בוחר לפי מחיר, אבל המחירים שהוא רואה הם list price. שתי דוגמאות שהוצגו:

- **Adjustment חיובי.** workload שיכול לרוץ על M7, M6 ו-M5, אבל על M5 הוא פחות יעיל — מוסיפים +10% למחיר של הדור הישן, כך ש-Karpenter יעדיף אותו רק כשזו הקפסיטי הזמינה [15:24].

- **Adjustment שלילי.** Savings Plan: המחיר שאתם משלמים בפועל נמוך מ-list price, ואתם רוצים ש-Karpenter יתחשב במחיר האמיתי בבחירה שלו [15:24].

8. מה השתפר בשנה האחרונה

השקופית האחרונה של החלק הראשון ריכזה שיפורי ביצועים שנכנסו ל-Karpenter ול-EKS Auto Mode במהלך השנה. הדובר לא עבר שינוי-שינוי, אלא השתמש בה כדי להדגים קצב פיתוח: "פסטר סקייל אאוט, פסטר סקייל אין, בעצם node boot time הרבה יותר מהיר, והכי חשוב בעצם ה-bin packing יותר חכם, smarter consolidation, שבסופו של דבר מביא לחיסכון של עלויות" [16:55].



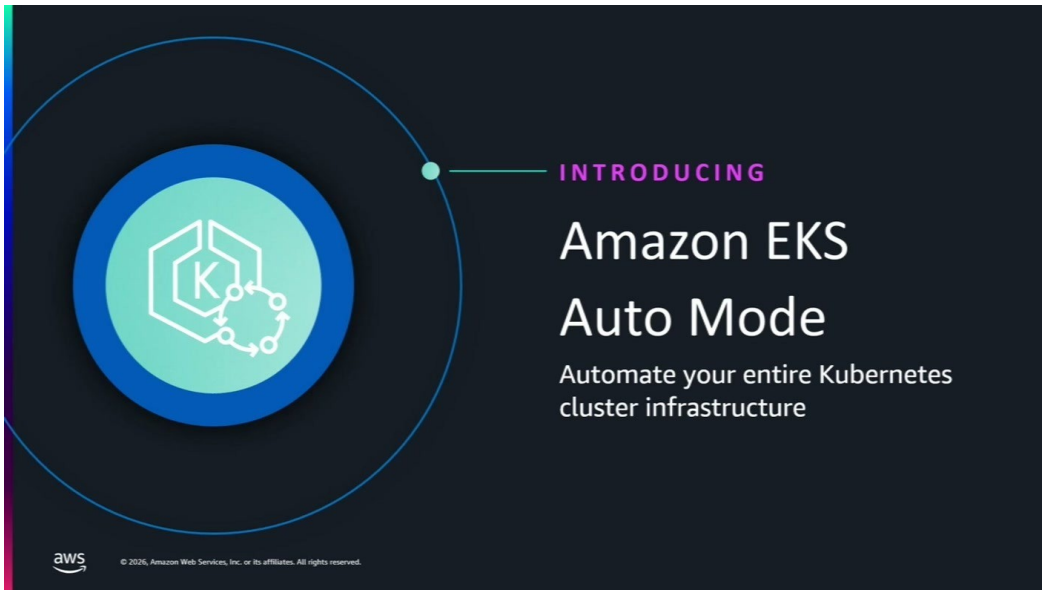
ארבעת המספרים כפי שהופיעו על השקופית: 43% scale-out מהיר יותר, 39% boot מהיר יותר, 69% scale-in מהיר יותר, ו-30% יותר קיבולת קלאסטר

שיפור (מהשקופית)	הפירוט שעל השקופית
43% faster scale-out	0→1,000 pods across 250 nodes: 254s → 145s
39% faster node boot	13 seconds faster via startup detection optimization
69% faster scale-in	Consolidation at 70%→2% load: 302s → 93s
30% more cluster capacity	Smarter consolidation — ships automatically, no config

הערת קריאה המספרים האלה הופיעו על השקופית ולא הוסברו בעל פה. שימו לב במיוחד לשורה האחרונה: השיפורים האלה מגיעים אוטומטית עם הגרסה, בלי קונפיגורציה — כלומר מי שמריץ Karpenter מעודכן נהנה מהם גם בלי לשנות דבר.

9. EKS Auto Mode: מה עובר לאחריות AWS

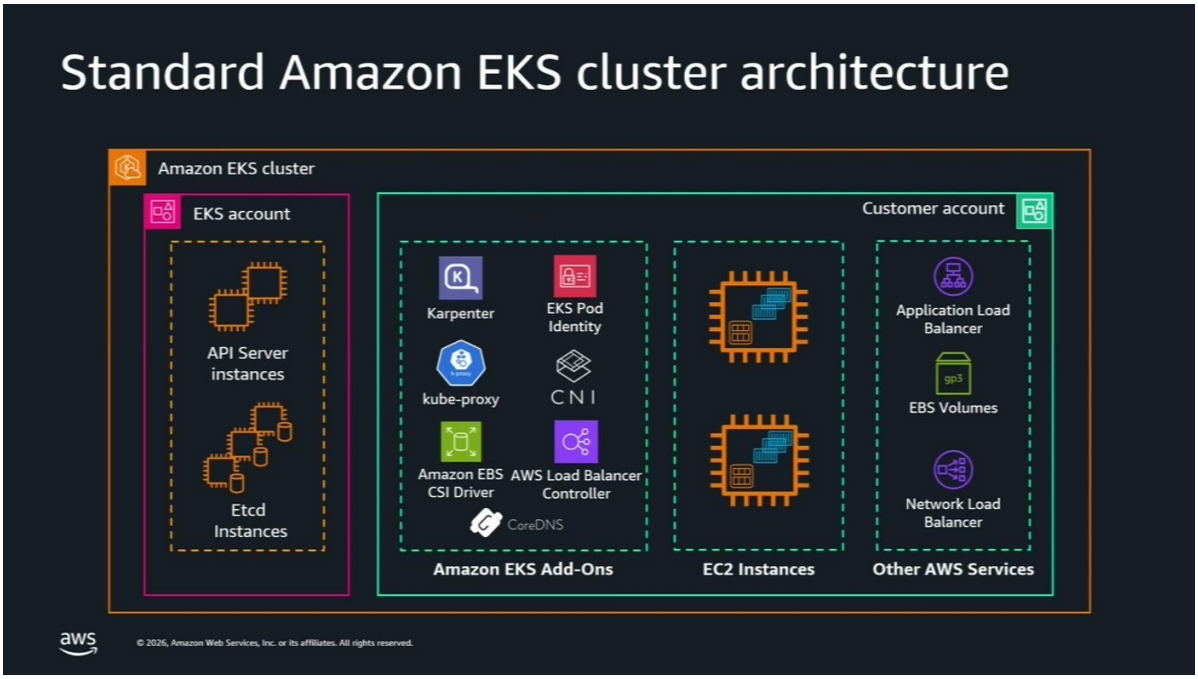
Eliran Mesika, Senior Solutions Architect ב-AWS, לקח את החלק השני. נקודת המוצא שלו: "אם אתם פה, זה אומר שאתם מכירים קוברנטיס, ואתם מבינים את היתרונות הגדולים שיש לו. אבל אתם גם יודעים שניהול קוברנטיס, יש בו קצת כאב ראש, במיוחד באזורי התחזוקה" [16:55]. EKS Auto Mode, שנבנה בעקבות פידבקים חוזרים מלקוחות, הוצג כ"הדרך הפשוטה ביותר להריץ קוברנטיס ב-AWS" — והוא עצמו משתמש ב-Karpenter מתחת למכסה [16:55].



השקופית שפותחת את החלק השני של הסשן

גבול האחריות היום

השקופית הבאה הראתה ארכיטקטורה של קלאסטר EKS סטנדרטי, עם קו הפרדה ברור. AWS כבר מנהלת את ה-control plane — החלק האפור. "אבל כל המסגרת הירוקה היא עדיין באחריות שלכם, הלקוחות" [18:26]: תחזוקה של רכיבים קריטיים כמו VPC CNI ו-CoreDNS, הגדרות הקומפיוט, והבאת רכיב autoscaling כמו Karpenter.



המסגרת הירוקה מסמנת את מרחב האחריות של הלקוח: Karpenter, kube-proxy, CNI, ה-add-ons, ומופעי ה-EC2

העלות של זה היא עלות סיכון, לא רק עלות זמן: "כל שדרוג או עדכון גרסה, או טעות קטנה בקונפיגורציה, עלולה לתקוע פודים או לפגוע בגישה לסטורג'" [18:26] — עבודה שלא מייצרת ערך עסקי. זה מה ש-AWS מכנה undifferentiated heavy lifting, וזה מה ש-Auto Mode בא לפתור.

מה מקבלים בפועל

- **אין ניהול של autoscaling.** ה-control plane מתחזק, משדרג ומגדיל את התשתית בזמן אמת לפי הדרישה [19:57].
- **הקלאסטר מגיע עם best practices מובנות.** מהגדרות ה-Security Group ועד הרשאות IAM ברמת הנוד [19:57].
- **Add-ons out of the box.** במקום להתקין ולעדכן VPC CNI או EBS Driver — "הכל מגיע מנוהל כחלק מהפלטפורמה" [19:57].

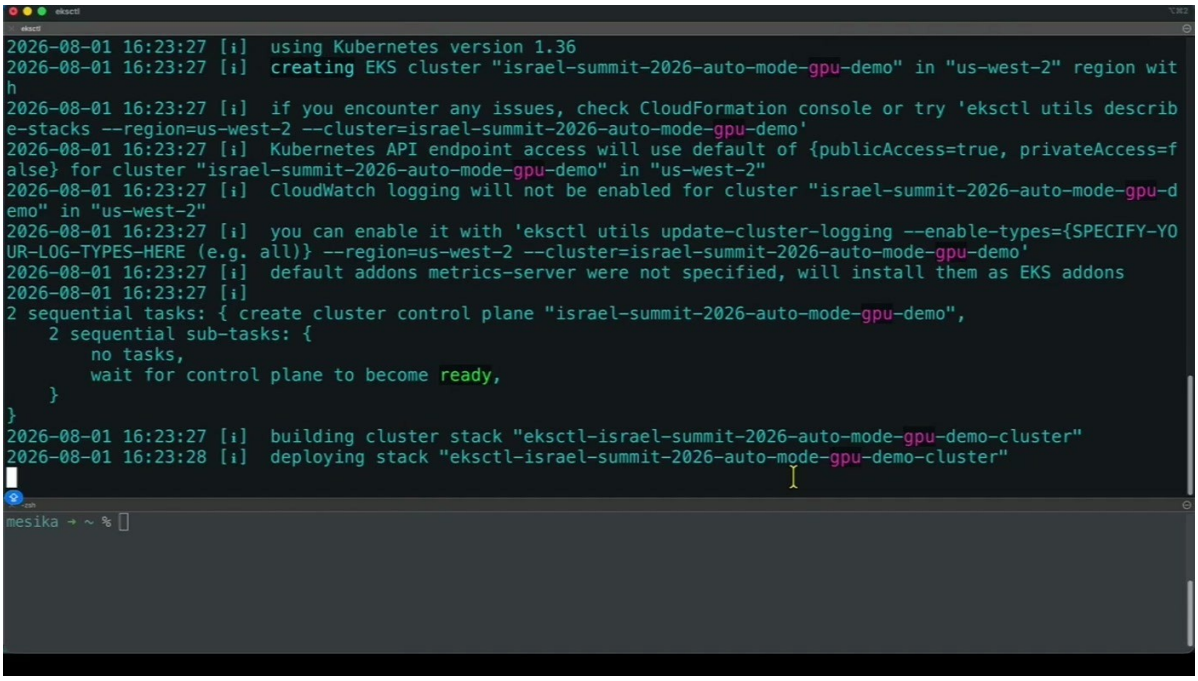
> **Eliran Mesika [19:57] —** עכשיו עד עכשיו מצגות, זה אחלה, זה נחמד. בואו נראה איך זה עובד באמת.

10. הדמו: מאפס ל-LLM inference

הדמו היה מוקלט, והמטרה שהוגדרה בתחילתו: להקים קלאסטר EKS Auto Mode חדש עם כל ה-best practices, ולפרוס עליו את המודל Qwen3 בגודל 35 מיליארד פרמטרים עם inference מלא — "וכל זה בפחות מ-15 דקות" [18:26]. ההדגשה החשובה: "אני לא הולך להגדיר ולתחזק מראש שרתי GPU או להתקין דרייברים ייעודיים. אנחנו נפרוס את האפליקציה, Auto Mode יבין מה אנחנו צריכים, יקצה את התשתית המתאימה, והמודל שלנו ירוץ" [19:57].

שלב 1 — בדיקת זמינות Spot והקמת הקלאסטר

הצעד הראשון היה לוודא שיש מכונות Spot זמינות ב-region (us-west-2), ורק אז להתחיל את היצירה [21:32]. במקביל הוסבר מה זה Spot למי שלא מכיר: "מכונות ספוט אלה מכונות שאנחנו יכולים לקבל בעד 90% הנחה" — עם ה-trade-off הידוע, שהמכונה יכולה להילקח בהתראה של שתי דקות [21:32].

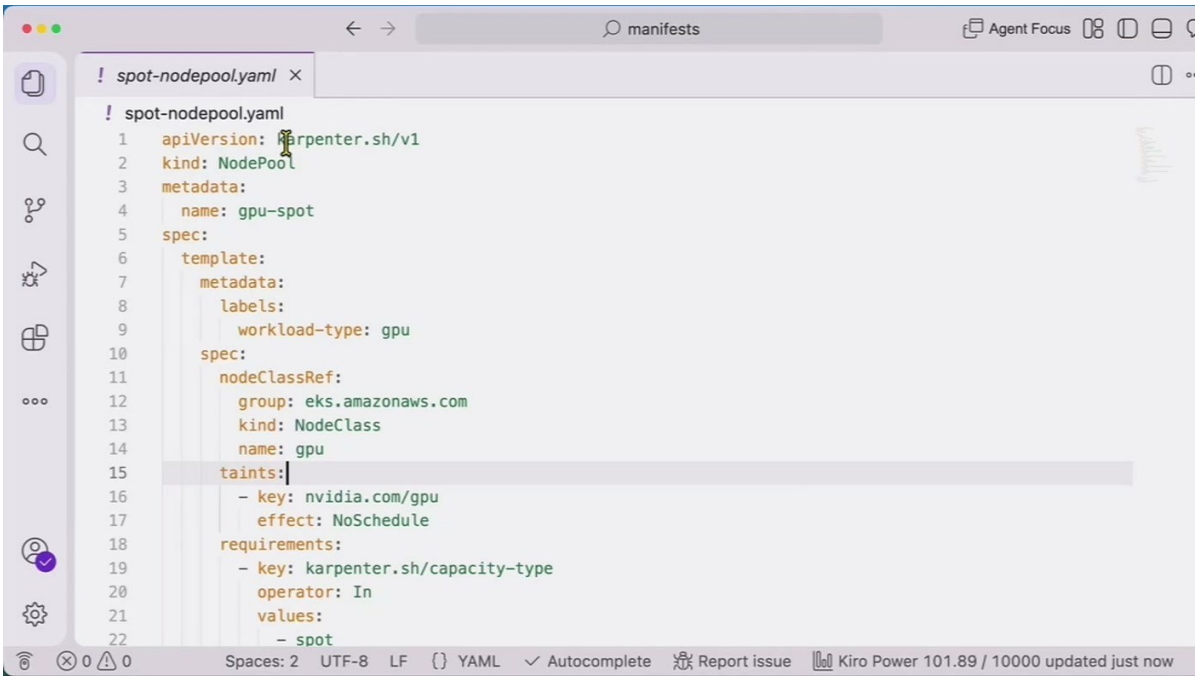


פלט eksctl בזמן יצירת הקלאסטר ב-us-west-2, כולל האזהרה שברירות המחדל של ה-add-ons יותקנו כ-EKS add-ons

בתחתית המסך רץ לאורך כל הדמו כלי בשם eks-node-viewer, שמראה כמה נודים חיים ואילו פודים רצים עליהם [21:32]. בשלב הזה הוא הראה אפס נודים ואפס פודים — הקלאסטר קם, ריק.

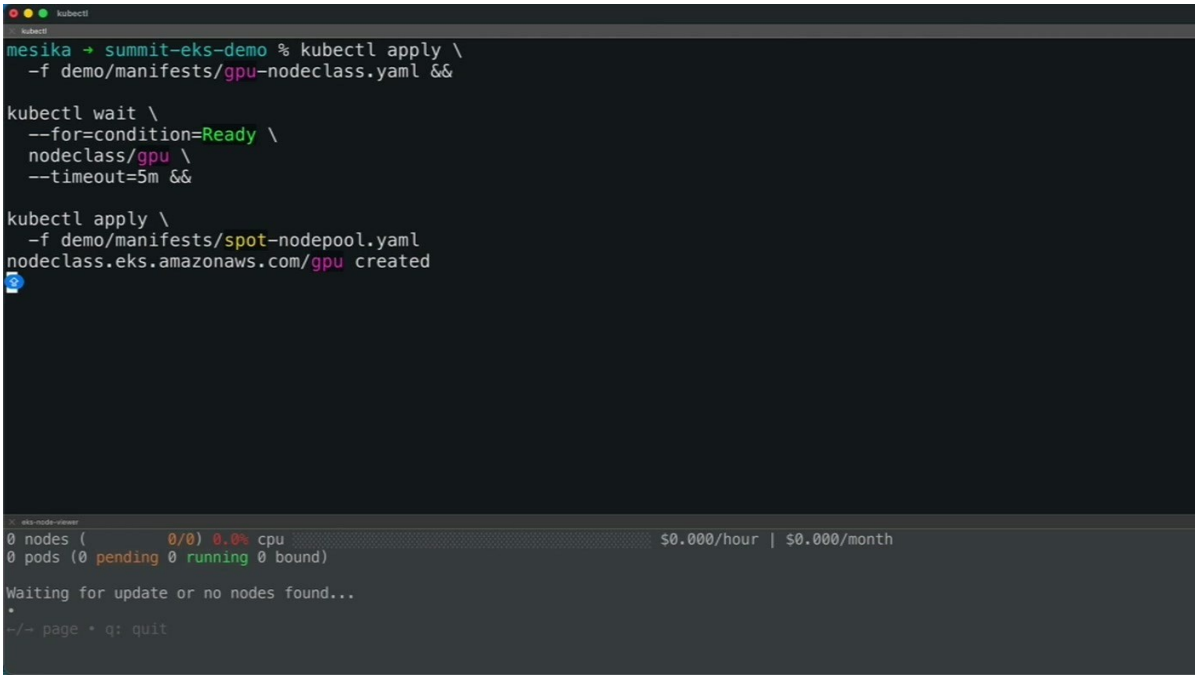
שלב 2 — NodePool ל-GPU, נכתב עם Kiro

את הקונפיגורציה הוא כתב בעזרת Kiro, ה-IDE האג'נטי של AWS [21:32]. ה-NodePool שנוצר נקרא gpu-spot, מבקש מכונות עם GPU של NVIDIA, מסוג Spot, ו-instance type ספציפי — g6e.2xlarge — כי "נתתי שזאת ספציפית מתאימה ל-workload" [23:03].



קובץ spot-nodepool.yaml: apiVersion karpenter.sh/v1, nodeClassRef לקבוצה eks.amazonaws.com, taint על nvidia.com/gpu ודרישת capacity-type בערך spot

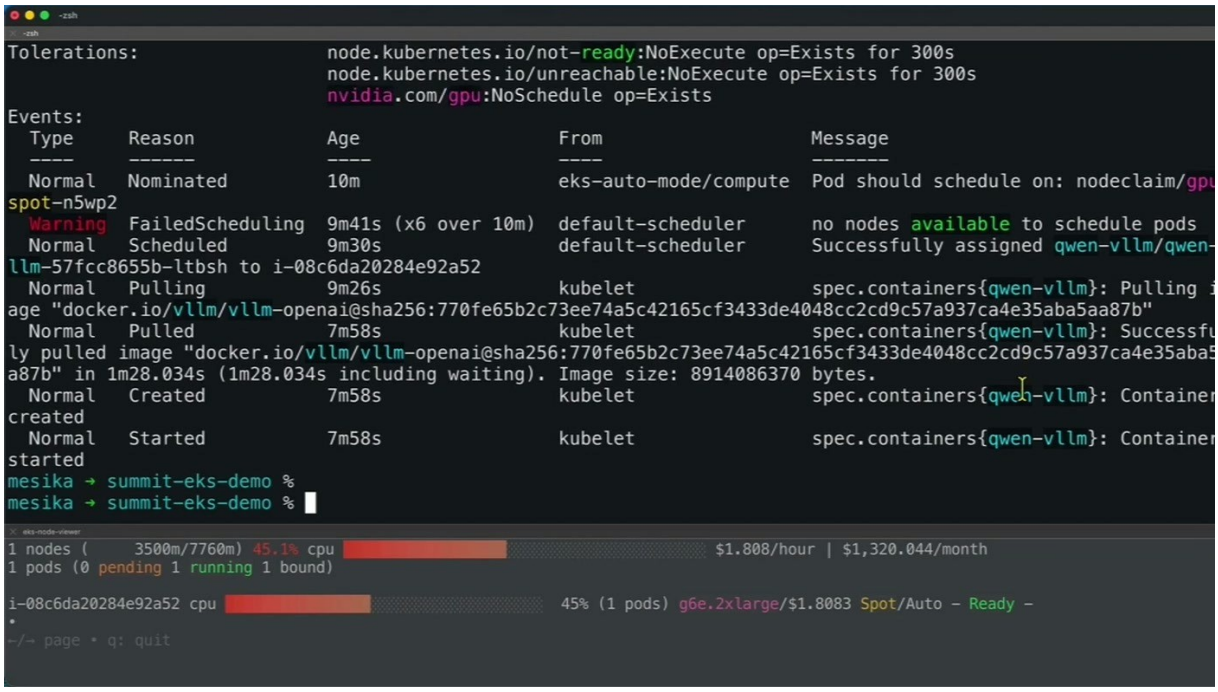
הנקודה של הדמו "אבל שימו לב מה לא עשיתי פה, וזה הכי מעניין — לא התקנתי פה דרייברים. בתור מי שיצא לו להקים שרתי GPU ב-EKS יודע שזה יכול להיות כאב ראש: צריך להתקין פה את ה-AMI הספציפי, להתקין DaemonSet של NVIDIA ולדאוג לתאימות. עם Auto Mode, הכל מגיע [23:03] "out of the box.



הרצת kubectl apply — ה-NodePool וה-NodeClass נוצרו שניהם בהצלחה, בעוד eks-node-viewer עדיין מראה אפס נודים

שלב 3 — פריסת המודל והמדידה

אחרי ה-apply של הדיפלוימנט, שתי דקות של הכנת מכונה והורדת image בגודל של כ-9–10 ג'יגה. ה-node viewer הראה שהמכונה שנמצאה היא אכן g6e.2xlarge מסוג Spot, עם המחיר שלה, והפוד ב-[24:35] Running.



פלט kubectl describe: אירועי Nominated, FailedScheduling, Scheduled, Pulling ו-Started — ומתחת, נוד g6e.2xlarge ב-$1.8083 לשעה, Spot/Auto, Ready

המספר שהוא הצביע עליו: "זמן הטעינה הוא דקה וחצי, וההורדה של האימג' באזור 9 ג'יגה". ההסבר: "הקסם שקורה פה של הורדה של 9 ג'יגה בדקה וחצי, זה בגלל יכולת של SOCI — זה ספריית אופן-סורס של AWS, שבגדול מה שהיא מאפשרת לנו זה הורדה פרללית של השכבות של האימג', ודרך זה לחסוך זמן" [24:35].

ההקשר שהוא נתן למספר הזה רלוונטי לכל מי שמתכנן AI בפרודקשן: "הדבר הזה הוא קריטי בעולם ה-AI. המודלים הם ענקיים, וכשיגיע פתאום איזשהו פיק ואני צריך לעשות סקייל, אנחנו נהיה חייבים להפחית את זמן העלייה" [24:35].

שלב 4 — שאילתה למודל

אחרי port-forward לפוד, הוא שאל את המודל — שנפרס ללא כל מודיפיקציה — למה תעשיית ההייטק הישראלית מתקדמת כל כך מהר. התשובה שהוצגה על המסך, בתרגום לעברית:

> **— התשובה של Qwen3 בדמו [26:08]** ההייטק הישראלי זז כל כך מהר כי אנחנו המקום היחיד בעולם שבו סטארטאפ יכול להפוך מרעיון ראשוני ליוניקורן לפני שהקפה יספיק להתקרר — וכל זה בזמן שאנחנו מסבירים למשקיעים בחו"ל כמה חוות השרתים שלנו מוגנת מטילים.

הסיכום שלו לדמו: "אנחנו רצינו להריץ LLM inference על קלאסטר חדש שיצרנו, ועשינו את זה במהירות. יצרנו NodePool שיתמוך ב-workload, יצרנו את הדיפלוימנט והרצנו. וזה היה פשוט" [26:08].

11. סיפור לקוח: Bluevine

המעבר לחלק השלישי נעשה עם השאלה המתבקשת: "אתם בטח אומרים לעצמכם, אה, דמו, בדמו הכל טוב. זה יכול להחזיק מים בעולם האמיתי? מה יקרה שנזרוק את EKS Auto Mode למים העמוקים של פרודקשן — מערכות עם סקייל גבוה, עם דרישות רגולציה, עם דרישות אבטחה מחמירות?" [26:08]. כדי לענות, עלה לבמה Fadi Tuma, Senior DevOps Engineer ב-Bluevine.

פרופיל החברה כפי שהוצג: פינטק ל-SMB banking בארה"ב, מעל מיליון עסקים, כ-600 עובדים

Bluevine היא חברת פינטק שנותנת שירותים פיננסיים לעסקים קטנים ובינוניים בארצות הברית — checking, lending ועוד — ליותר ממיליון לקוחות, עם יותר משני מיליארד דולר בפיקדונות. החברה פועלת מארה"ב, ישראל והודו, וכ-600 עובדים [27:40]. מבחינת הנדסה: cloud native, ו-Kubernetes הוא חלק מרכזי בפלטפורמה — לא רק להרצת שירותים, אלא גם כחלק מרכזי מזרימת ה-CI/CD.

שני ה-use cases שהכריעו

- **CI/CD — הכבד מבין השניים.** מעל 600 ג'ובים של Jenkins פרוסים על כמה סביבות, "ויש לנו 4,000 CI jobs שרצים מדי יום, ויותר מ-170 דוקר אימג'ים שנבנים ביום" [29:11]. הכל בפודים שעולים ויורדים דינמית — compute on demand.
- **והם לא פודים קלים.** "אנחנו משתמשים גם ב-DinD קונטיינר, למי שמכיר, Docker in Docker — ככה שזה קונטיינר מאוד כבד שדורש משאבים" [29:11], וגם עובדים במקביל עם AMD ועם ARM.
- **Monitoring ו-Observability.** סטאק של Prometheus ו-Grafana עם עשרות exporters, שמנטר גם כלים פנימיים וגם שירותי AWS — "שכבת מוניטורינג מאוד כבדה שרצה גם כן על קוברנטיס" [30:45].

שלושת האתגרים עם EKS סטנדרטי

"עם EKS הסטנדרטי, אנחנו מצאנו את עצמנו משקיעים לא מעט זמן באופרציה ובתחזוקה של הקלאסטר, ופחות זמן בפיתוח ובדליברי" [30:45]. שלושת המוקדים:

- **ניהול node groups ו-instance types.** כשיש workloads שחלקם bursty, חלקם ארוכי חיים וחלקם דורשים סוגי קומפיוט שונים — "מהר מאוד נכנסים למורכבות תפעולית" [30:45].
- **Autoscaling.** עם Cluster Autoscaler צריך כל הזמן לכוונן ולהבין זמני responsiveness. בעולם כמו Jenkins, "זמן scale-up זה לא נשאר בעיה תיאורטית, זה הופך לג'ובים שמחכים בתור" [30:45] — וזה משפיע ישירות על הדליברי.
- **Add-ons.** CoreDNS, kube-proxy, EBS CSI ואחרים — "כל אחד מהם בעצם ניתן לניהול, אבל השדרוגים וה-dependencies לאורך הזמן, זה הופך להיות עוד שכבה שאנחנו צריכים לתחזק" [32:16].

Fadi Tuma, Bluevine [32:16] — ואם אני צריך לסכם את זה משפט אחד — it's the best configuration, it's the less configuration.

מה שמשך אותם, לדבריו, לא היה רק מהירות הסקיילינג אלא רוחב האחריות: שכבת הקומפיוט, שכבת ה-networking ושכבת ה-storage של הקלאסטר עוברות ל-AWS. "אנחנו כבר לא מתעסקים עם node group upgrades ולא עם פאצ'ינג, וגם לא מתעסקים יותר עם autoscaler" [32:16]. בנוסף קיבלו תמיכה ב-AMD וב-Graviton, גם על Spot וגם על On-Demand [33:50].

המספרים שהם מדדו

מדד	השיפור שדווח	חותמת זמן
Operational overhead בניהול הקלאסטר	כמעט 60% פחות	[33:50]
זמן provisioning לסביבה חדשה	ירידה של כ-75% — מכמה שעות, ולעיתים יום שלם, ל-50–55 דקות	[33:50]
בעיות סקיילינג ותאימות add-ons	ירידה של כ-40%, "הייתי אומר אפילו טיפה יותר"	[35:20]
עיסוק ב-node upgrades ו-security patches	100% — בתנאי שמכוונים policy ו-NodePool נכונים ל-use case	[35:20]

הסייג ה-100% לא הגיע חינם. הדובר ציין במפורש: "היה לנו אתגר בשלב הזה, בסקופ הזה, אבל אחרי שמבינים את ה-use case שלכם, מה אתם מריצים — האם זה deployment, האם זה StatefulSet — שמים את הפוליסי המתאים ואת ה-NodePool המתאים, יכולים להגיע למספר הזה" [35:20]. כלומר, הבנת ה-workload עדיין באחריותכם.

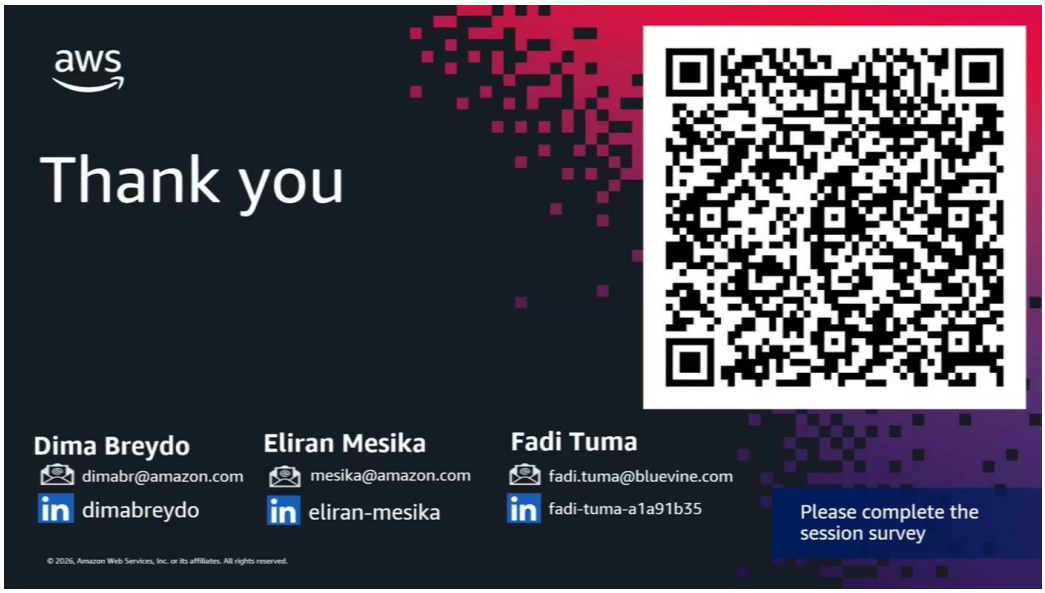
והמספר שהוא עצמו סימן כמשמעותי ביותר לא היה מספר אחוזים אלא תוצאה: ההפחתה בתחזוקת הקלאסטר אפשרה לצוותי ה-DevOps וה-SRE להשקיע יותר זמן ב-business value, ב-reliability, בשיפור התשתית ובייעול תהליכי הפיתוח [35:20]. "בסוף ההחלטה לא הייתה רק החלטה טכנית" [35:20].

12. מה לוקחים מכאן

- **אם אתם עדיין על Cluster Autoscaler — זו הנקודה לבדוק מעבר.** המודל ה-groupless חוסך את ניהול ה-ASGs ומקצר את זמן התגובה לפודים ב-Pending. הדובר ציין שהוא מכיר לקוחות שעדיין משתמשים ב-Cluster Autoscaler, אבל כל הסשן נבנה סביב ההנחה שזה לא המקום להישאר בו [1:31].
- **התחילו מ-NodePool אחד רחב.** זו ההמלצה המפורשת לצעד הראשון [9:18]. NodePools נוספים עם taints נכנסים רק כשיש דרישה אמיתית — GPU, AMI אחר, או הפרדת tenants.
- **consolidation policy הוא כפתור העלות המשמעותי.** בחירה בין WhenEmpty לבין WhenEmptyOrUnderutilized היא בחירה בין יציבות לחיסכון. Balanced הוא הפשרה. אל תשכחו PDBs — ואל תהדקו אותם מדי [3:06].
- **Graviton הוא הרווח המהיר, אם ה-images מוכנים.** עד 40% שיפור ב-price performance [6:11], אבל התנאי הוא multi-architecture builds לכל ה-workload, כולל DaemonSets.
- **Auto Mode מתאים למי שמוכן לוותר על שליטה בדאטה פליין.** אם אתם צריכים AMI מותאם, agents משלכם על הנוד, או שליטה מלאה בתזמון שדרוגים — בדקו את זה לפני שאתם זזים.
- **לסביבות AI: SOCI הוא מה שהופך scale-out לאפשרי.** הורדה מקבילית של שכבות ה-image הביאה 9 ג'יגה בדקה וחצי בדמו [24:35]. בלי זה, cold start של מודל גדול הוא חסם אמיתי.
- **Capacity Buffer ו-Node Overlay עדיין ב-alpha.** שווה להכיר ולעקוב, אבל לא לבנות עליהם פרודקשן. לגבי Node Overlay יש אזהרה מפורשת מהבמה [15:24].

13. מונחון

מונח	משמעות
Karpenter	פרויקט קוד פתוח של AWS תחת CNCF, שמקצה נודים ל-Kubernetes לפי פודים ב-Pending
Groupless	אין Auto Scaling Groups — כל נוד נבחר בנפרד, בלי הנחה שכל המכונות בקבוצה זהות
Consolidation	שם כולל לשלושת מנגנוני האופטימיזציה: הורדת נודים ריקים, איחוד פודים, והחלפה בזול יותר
EC2NodeClass	CRD שמגדיר את הצד ה-AWS של הנוד: subnets, security groups, user data, דיסק ו-AMI
NodePool	CRD שמגדיר את מרחב האפשרויות: משפחות, דורות, AZ, ארכיטקטורות ו-capacity types
Drift	פיצ'ר שמזהה שיצאה גרסת AMI חדשה ומגלגל את המכונות בהתאם
Capacity type	אופן הרכישה: Spot, On-Demand, או Reserved (ODCR / Capacity Blocks for ML)
Baseline Capacity	כמות נודים קבועה שרצה תמיד, ללא גדילה דינמית; ה-limit משמש כ-headroom להחלפות
Capacity Buffer	פודים וירטואליים ששומרים מקום פנוי מראש כדי לקצר cold start (alpha)
Node Overlay	הזנת מידע ש-Karpenter לא מכיר, למשל תיקון מחיר חיובי או שלילי (alpha)
EKS Auto Mode	מצב מנוהל של EKS שבו AWS מתחזקת את הדאטה פליין, ה-add-ons והשדרוגים
SOCI	ספריית קוד פתוח של AWS להורדה מקבילית של שכבות container image
eks-node-viewer	כלי CLI שמציג נודים חיים, פודים עליהם, ועלות לשעה — שימש לאורך הדמו



שקופית הסיום עם פרטי הקשר של שלושת הדוברים