רזיליינס בסקייל על Amazon EKS

AWS Summit Tel Aviv 2026 ,סיכום סשן — TLV-SVS401

אסף קנדר, Senior Manager, Solutions Architecture, AWS | אופיר כהן, Container Security Lead, Wiz

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

סיכום מאת קובי אלמוג

על המסמך הזה

המסמך מסכם את הסשן TLV-SVS401, "Resilience at Scale on Amazon EKS: Networking, Security, and Compute", מתוך AWS Summit Tel Aviv 2026 (מסלול AWS for Containers and Platform Engineering, רמה 400). הוא נבנה מתמלול אוטומטי של ההקלטה ומהסליידים שחולצו מהווידאו עצמו. חותמות הזמן בגוף המסמך מפנות לנקודה המדויקת בהקלטה.

הסשן הוא דו-שיח: אסף קנדר מ-AWS מציג את המסגרת ואת נקודות ההחלטה, ואופיר כהן מ-Wiz נכנס שלוש פעמים כדי לספר מה נבחר בפועל בסביבה שמנהלת מעל מיליון ריסורסים. זו לא הרצאת "עשו כך" — זו הרצאה על trade-offs.

איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. המסקנות המרכזיות והראיות להן.
- **פרקים 2–3 — המסגרת והבסיס:** שלושת סוגי הקונטרולים, שכבות הרזיליינס, ומה באמת מסתתר מאחורי "ארבע רפליקות".
- **פרקים 4–6 — Compute ו-Operations:** TopologySpread, Karpenter, Pod Disruption Budgets מול ניהול מחזור חיי nodes כפי ש-Wiz מיישמת אותו.
- **פרק 7 — Networking:** מ-VPC בסיסי, דרך subnets ייעודיים לפודים, ועד prefix mode.
- **פרק 8 — Security ו-Supply Chain:** מודל הגרף, toxic combinations, base images ו-admission control.
- **פרק 9 — מה לוקחים מכאן:** פעולות קונקרטיות ומקורות המשך.

> **הערת מקור** המסמך הופק מתמלול אוטומטי של ההקלטה. הסשן נאמר בעברית עם צפיפות גבוהה של מונחים באנגלית, והתמלול משבש חלק מהם ("קרפנטר" = Karpenter, "איקס" / "AKS" = EKS, "טופולג'י ספרד" = Topology Spread). המונחים נורמלו לאנגלית בגוף המסמך, אך הציטוטים מובאים כלשונם בהקלטה ואומתו מול חותמת הזמן המצוינת. מספרים ונתונים מצוטטים רק כפי שהופיעו על הסליידים או נאמרו במפורש.

1. תקציר מנהלים

הסשן לוקח את ההנחה הנפוצה — שהכל צריך להיות multi ולכן אנחנו מכוסים — ומפרק אותה. כל שקף מציג הגדרה שנראית סבירה, ואז שואל מה קורה כשמשהו נופל. התשובה כמעט תמיד היא שהבחירה הדיפולטית כבר קיבלה עבורכם החלטה, ובדרך כלל לא ידעתם מה היא.

- **רזיליינס בסקייל היא שאלה של קונטרולים, לא של אנשים.** "אנחנו לא יכולים לבנות על אנשים שיעשו את ה-operations ויעשו את ה-resiliency" [3:01]. המסגרת שהוצגה: preventive, corrective ו-detective controls — כשהסשן מתמקד בשניים הראשונים.
- **ה-defaults של Kubernetes כבר הכריעו את ה-trade-off במקומכם.** ברירת המחדל של TopologySpread היא ScheduleAnyway עם maxSkew 3 (hostname) ו-5 (zone) — כלומר המערכת מעדיפה להעלות את הרפליקה על פני לפזר אותה. אסף: "מה שבעצם באתי ואמרתי בסקדג'ול anyway זה שאני מעדיף שיהיו לי ארבע רפליקות על פני פיזור שלהם" [9:11].
- **ScheduleAnyway גם מנטרל את Karpenter בשקט.** הטריגר של Karpenter הוא pods במצב Pending. אם ה-scheduler הצליח להעלות את הפוד בכל מחיר, אין Pending — ולכן "גם אם קרפנטר עכשיו היה יכול לעזור לי והיה יכול אולי לייצר לי capacity בעוד מקום הוא לא ייכנס לפעולה" [10:42].
- **Karpenter הוא חבר צוות operations, לא כלי חיסכון.** Wiz מגדירה AMI pinning, expireAfter של 336h (14 יום) ו-disruption budget של 10% nodes. אופיר: "יש באמת את החיים לפני ואחרי קרפנטר" [39:45], ועל pinning: "בחיים, בחיים, בחיים, אל תעבדו על latest" [16:47].
- **רשת ואבטחה מושכות לכיוונים הפוכים, וזה נקודת עיצוב מפורשת.** מצד אחד "תעשה Micro-Segment Everything" [21:21], מצד שני מחסור בכתובות IP באנטרפרייז. הפתרון שהוצג: subnets נפרדים ל-nodes ולפודים, subnets ייעודיים לפי הקשר, ו-prefix mode שהופך כל secondary slot ל-16 כתובות [25:57].
- **Supply chain הוא blast radius בפני עצמו.** אפס חולשות ב-scan לא אומר שהבסיס נקי — "יכול להיות שהחולשה עצמה היא קריטיקל אבל הקונטקסט הוא לא מעניין" [30:36], ומנגד אימג' נקי יכול להיבנות ממקור לא מהימן. הקו האחרון הוא admission controller: "זה כמו אנטיבירוס מודרני" [36:45].

מה זה אומר לארגון

רוב התוכן ישים מיידית ולא דורש שירות חדש: בדיקת ה-defaults של TopologySpread בקלסטר, הוספת Pod Disruption Budgets, הפרדת CoreDNS ו-kube-proxy ל-node pool ייעודי, ותכנון ה-VPC לפני שנגמרות הכתובות. הפער העיקרי שמחייב החלטה ארגונית הוא בצד ה-supply chain — מאיפה מגיעות ה-base images ומי רשאי לדחוף לקלסטר.

2. המסגרת: שלושה קונטרולים ושכבות רזיליינס

אסף פתח בשני חיבורים. הראשון לשקף משנים קודמות — המספר 34, "34 דרכים מתועדות אז להריץ קונטיינרים על AWS", ומישהו מהקהל שהתעקש שזה צריך להיות 42. השני, רציני יותר, הוא לציטוט של Werner Vogels: "everything fails all the time". מכאן נגזר הכל — לא אם ייכשל, אלא מה קורה כשייכשל.

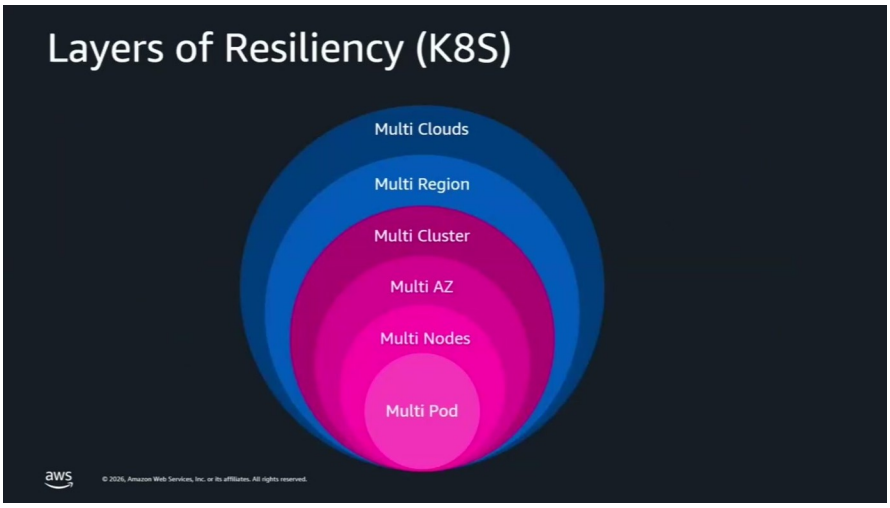
הסשן נשען במפורש על EKS Best Practices Guide ועל שני מסמכי Prescriptive Guidance נוספים (availability ו-scaling), אך עם הסתייגות מפורשת: "אין באמת one size fits all" — גם לא ב-best practices.

מה זה "scale" בהקשר הזה

- ריבוי קלסטרים וקלסטרים גדולים
- ריבוי locations ו-regions — פיזור גיאוגרפי
- ריבוי clouds, עם טכנולוגיות שאינן זהות
- diversity של אפליקציות — גם זה נחשב סקייל

שלוש משפחות הקונטרולים: מניעה לפני שקורה, תיקון כשצריך, וזיהוי והתרעה — הסשן התמקד בשתי הראשונות

ההתמקדות ב-preventive וב-corrective היא בחירה מודעת, לא אמירה על חשיבות: "אנחנו נתמקד בשני הסמליים אבל לא בלדל שהdetective controls לא חשוב אלא פשוט באמת כי אין לנו זמן לחסות את כולם" [4:35]. ההגדרה של preventive: "preventive controls באים ואומרים בואו ננסה לעצור דברים רעים לפני שהם קורים" [4:35].

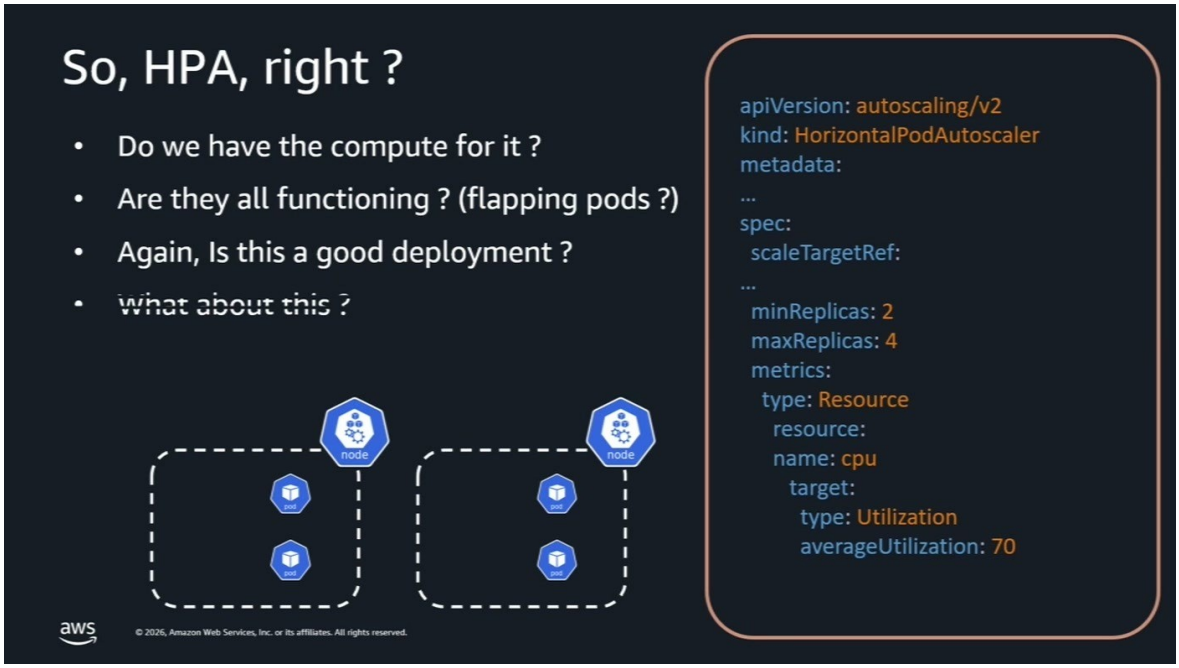


מודל השכבות המוכר — Multi Pod, Multi Nodes, Multi AZ, Multi Cluster, Multi Region, Multi Clouds

אסף הציג את השקף הזה ומיד ערער עליו: "עקרונית מאוד פשוט, נכון? הכל צריך להיות מולטי ומצבנו נהדר" [4:35] — ואז הצביע על מה שחסר בו. הצד האפליקטיבי, אבטחת ה-planes, הצד התפעולי והרשת אינם מיוצגים בעיגולים האלה, ובדיוק שם הסשן מתמקד.

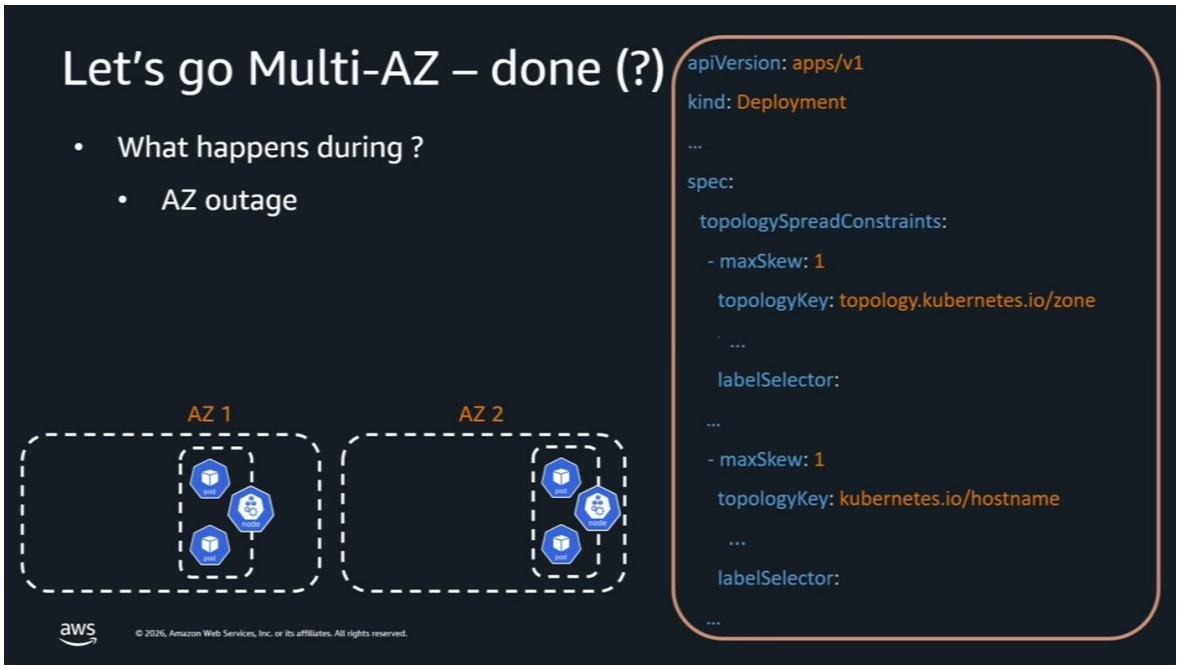
3. מארבע רפליקות ל-Multi-AZ: מה השקף לא מספר

החלק הזה בנוי כרצף שקפים שבו כל שלב נראה כמו פתרון, ואז נשאלת עליו שאלה אחת שמפילה אותו. ההתחלה: deployment עם ארבע רפליקות. "נהדר, סיימנו, יש לנו ארבע רפליקות, הכל מצוין" [6:07] — ומיד אחר כך שלוש שאלות: האם ארבע זה המספר הנכון (ואולי יש כאן waste, ואולי under performance), האם יש לנו בכלל את ה-compute להריץ אותן, והאם כל הפודים באמת מתפקדים.



אחרי הוספת HPA — השאלות שנשארות פתוחות: האם יש compute, האם הפודים מתפקדים, והאם זה deployment טוב

HPA פותר את כמות הרפליקות אבל לא את המיקום שלהן. אסף מדגים: "4 פרודים על אותו נוד, לא בטוח שזה דיפלוימנט טוב" [6:07]. גם המעבר לשני nodes אינו מספיק אם שניהם באותו AZ — "מה אם הם שניהם באותו איזי, שני הנודים אולי פחות רזיליינט" [7:39].

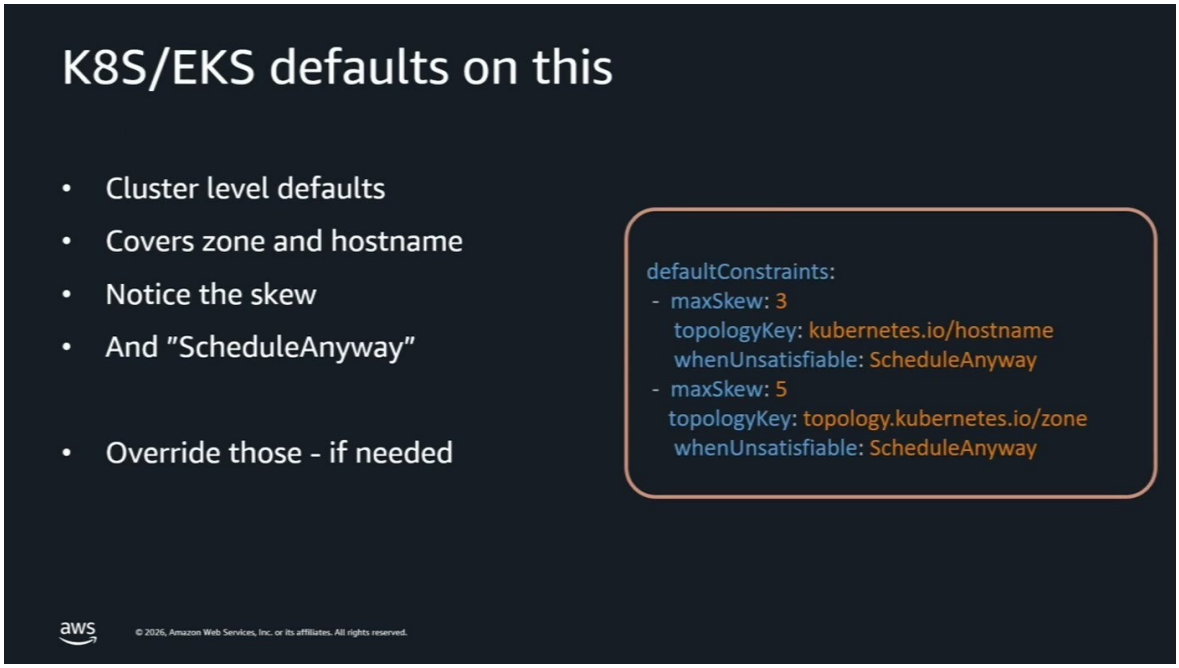


המעבר ל-Multi-AZ עם topologySpreadConstraints על zone ועל hostname — והשאלה מה קורה בזמן AZ outage

כאן מגיע התרחיש המעניין באמת. ב-AZ outage מלא Kubernetes מסתדר: נוצר node נוסף, שני פודים עליו, וה-Topology Spread עדיין מתקיים. אבל מה קורה ב-grey failure? "תשימו לב הנוד נשאר באוויר, אבל ה-AZ הזה כרגע לא זמין לי" [7:39] — אולי בעיה ב-Karpenter או ב-Cluster Autoscaler, ואולי "ב-AZ באמת נגמרה הקפסיטי של אין סטאנס טייפ שאנחנו רוצים" [7:39].

כל עוד צריך ארבע רפליקות הכל תקין. ברגע שה-HPA מבקש שש, ה-Topology Spread כבר לא יכול להתקיים, וה-scheduler צריך להכריע.

> **— אסף קנדר [9:11]** מה שבעצם באתי ואמרתי בסקדג'ול anyway זה שאני מעדיף שיהיו לי ארבע רפליקות על פני פיזור שלהם, או שש רפליקות במקרה הזה וארבע בעזי שתיים על פני פיזור



ברירות המחדל ברמת הקלסטר: maxSkew 3 ל-hostname, 5 ל-zone, ובשניהם whenUnsatisfiable: ScheduleAnyway

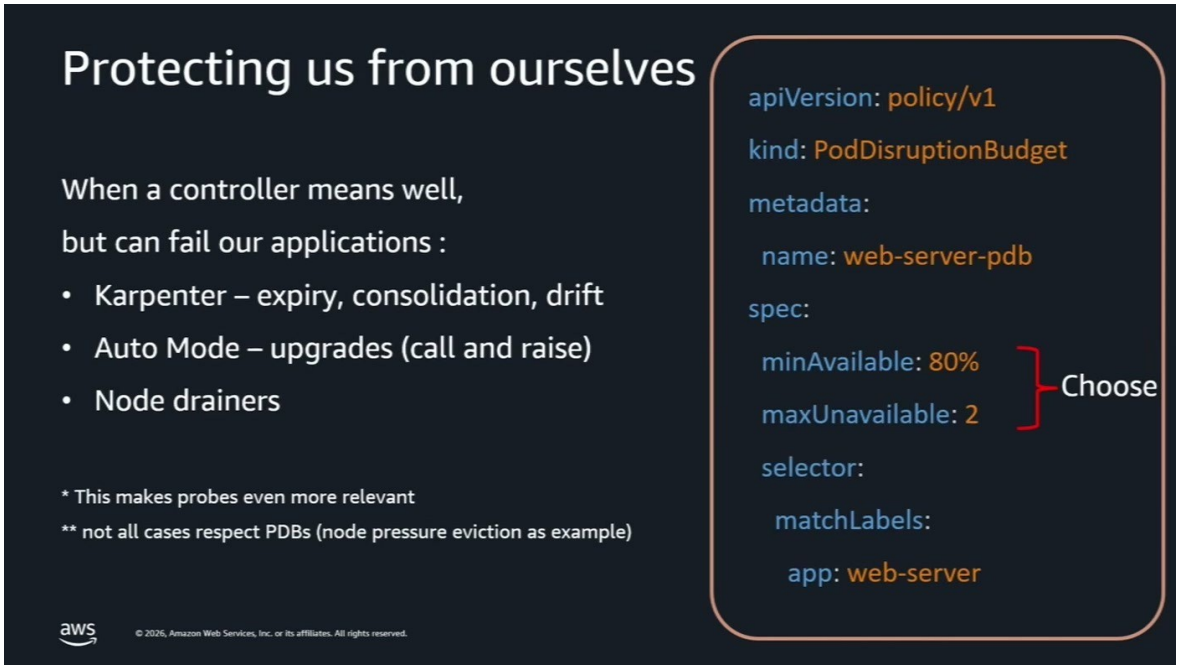
הנקודה החשובה: זו אינה החלטה שקיבלתם. "גם הדפולטים באו והניחו את מה שאני הנחתי שבטרייד-אוף, מה שמנצח זה בעצם שיעול התקמות הרפליקות" [9:11]. ה-maxSkew עצמו אינו בהכרח מתאים — "יכול להיות שזה מעט מדי, יכול להיות שזה יותר מדי" [9:11] — וההמלצה היא override מודע כשצריך.

4. TopologySpread, Karpenter ו-PDB: כשקונטרולר טוב שובר אפליקציה

האינטראקציה החשובה ביותר בחלק הזה היא בין ScheduleAnyway לבין Karpenter. Karpenter מופעל כאשר פודים נמצאים ב-Pending בגלל חוסר resources. אם ה-scheduler שובץ בכל מחיר, לא נוצר Pending — ולכן Karpenter לא ייצר capacity חדש.

> **— אסף קנדר [10:42]** גם אם קרפנטר עכשיו היה יכול לעזור לי והיה יכול אולי לייצר לי capacity בעוד מקום הוא לא ייכנס לפעולה

האלטרנטיבה, DoNotSchedule, הופכת את סדר העדיפויות: "עכשיו אני בא ואומר שישארו בפנטינג וצריך לטפל בסוגיית הקשל" [10:42]. אין כאן תשובה נכונה אחת — יש החלטה שצריך לקבל במודע ולהבין את השלכותיה.



Pod Disruption Budget כהגנה מפני קונטרולרים שפועלים לטובתנו — Karpenter (expiry, consolidation, drift), Auto Mode ו-node drainers

אסף הגדיר את PDB כהגנה עצמית: "pod disruption budgets עוזרים לי לשמור על עצמי מהקונטרולר הזה קרפנטר שאני רוצה שיעשה דברים טובים" [10:42]. השקף מציג את שתי האפשרויות — minAvailable או maxUnavailable — ומוסיף שתי הסתייגויות חשובות: זה הופך את ה-probes לרלוונטיים עוד יותר, ולא כל המקרים מכבדים PDB (eviction עקב node pressure כדוגמה).

אופיר חידד את הגבול של הכלי: "PDB לא דואג לדברים האלה, PDB דואג לסקדג'ואל" [18:19]. PDB מכסה node maintenance — voluntary disruptions, החלפת AMI, שדרוג קלסטר. הוא אינו מכסה involuntary disruptions, ולשם צריך Topology Spread ו-Multi-AZ.

> **שימו לב** נקודת החלטה לבדיקה בסביבה שלכם: מה ה-whenUnsatisfiable שלכם בפועל, ומה קורה ל-Karpenter תחתיו. השילוב ScheduleAnyway + Karpenter נראה תקין בכל דשבורד, ובדיוק לכן הוא מסוכן — הפודים רצים, אבל הפיזור שתכננתם לא קיים ואף אחד לא יצר capacity חלופי.

5. מהשטח: איך Wiz מנהלת מחזור חיי nodes

אופיר כהן, שמוביל Container Security ו-WizOS ב-Wiz, נכנס כדי לתת הקשר של סביבה אמיתית. הוא הקדים והסביר את המודל: Wiz נוסדה ב-2020 סביב גישת agentless — במקום להתקין agents על מכונות, לשאוב את הדאטה מה-control plane של הענן. "אם אנחנו cloud native, יש לנו הרבה מאוד דאטה שאנחנו יכולים להביא מהcontrol plane של הענן" [13:45].

- **מעל 50% מ-Fortune 100:** "מעל 50% מהפורצ'ן 100 זה לקוחות של וויז" [13:45], לצד סטארט-אפים ו-SMBs.
- **מעל מיליון ריסורסים ב-AWS:** הפריסה של Wiz עצמה, בארכיטקטורת cell-based לצורך isolation של blast radius ו-[13:45] tenant isolation.
- **230 מיליארד קבצים ביום:** היקף הסריקה — API, control plane, metadata, וגם דיסקים, volumes ו-registries. "זה נתון רק מלפני שנתיים" [15:16].

מכאן לנקודה התפעולית. אופיר שאל את הקהל מי שדרג EKS cluster בפרודקשן, ומי סיים את זה "בלי שערות לבנות על הראש" [15:16] — והידיים ירדו. הבעיה, לדבריו, היא שהתעשייה אימצה GitOps לאפליקציה ולאינפרה, אבל לא ל-node lifecycle: "כשבאים לשדרג איקס קלאסטר אז פתאום לפני העולמות של קרפנטר צריך שמישהו יחזיק לזה את היד" [16:47].

המלכודת, מה נשבר ומה מחזיק — שלוש גרסאות K8s בשנה, תמיכה של 3+3, ומול זה AMI pinning, expireAfter של 14 יום ו-budget של nodes 10%

שלושת המרכיבים בקונפיגורציה שהוצגה:

- **AMI pinning — predictability:** "אנחנו עושים פיננג ל-AMI בחיים, בחיים, בחיים, אל תעבדו על latest" [16:47]. הנימוק אינו אבטחתי בלבד: "resilience זה גם node — predictability שחוטף restart חוזר בדיוק לאותו AMI.
- **expireAfter — freshness:** מונע drift ו-snowflakes בצי גדול. "בכל חברה יש את המכונה של יוסי" [16:47]. הגישה: pets versus cattle.
- **disruption budget של 10% — zero downtime:** "אנחנו לא רוצים להפיל את כל המכונות בבת אחת אלא אנחנו רוצים לעשות gradual rollout" [18:19].

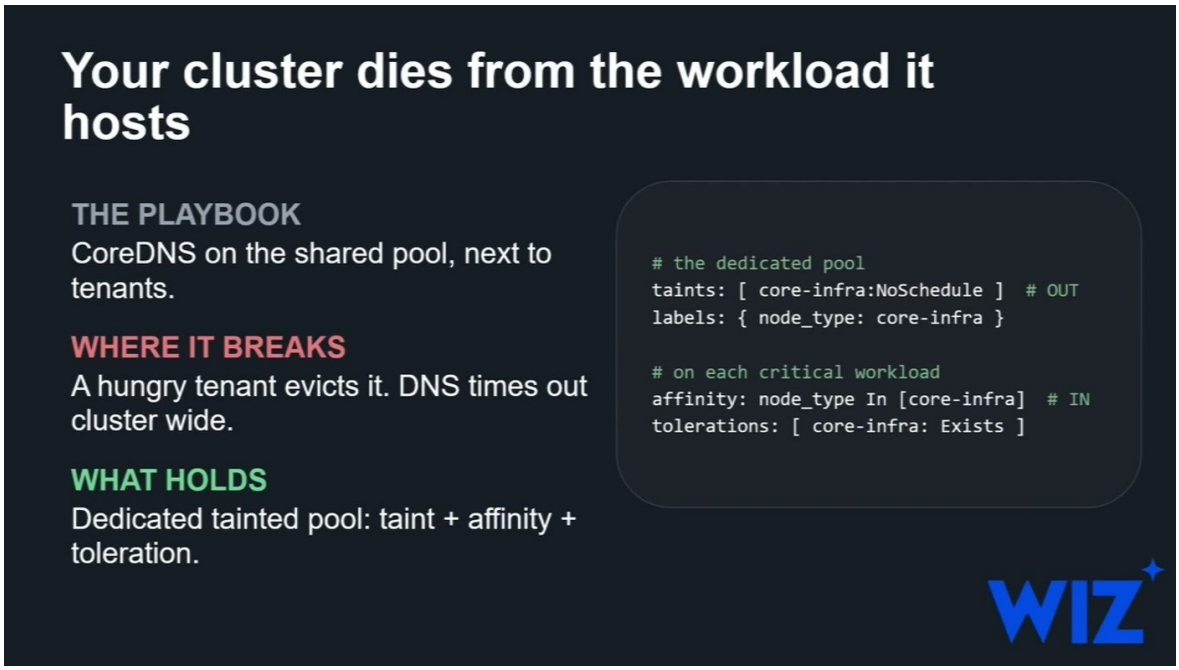
בסיכום הסשן אופיר חזר לנקודה הזו במונחי מוצר. הוא תיאר מדד נוסף ל-NPS — מה המשתמש היה עושה אילו היו לוקחים לו את המוצר — והשיב עליו בעצמו לגבי Karpenter: "אנחנו כנראה היפותטית נרצח את הבן אדם שיקח את זה מהיד, כי זה משנה חיים" [39:45].

— אופיר כהן [39:45] השקף של קרפנטר as an operations team member זה הדבר הכי נכון שיש להחליף עשרה, עשרים, שלושים הרבה עם אנשי דיו-אופס בקונטרולר שהוא סנג'ר והוא שואל את השאלות והוא עושה את ה-Continuous Drift ו-Reconciliation

6. הפרדת workloads קריטיים ו-rollout הדרגתי

הטענה המרכזית בחלק הזה: "לא כל ה-workloadים בקלסטר נולדו אותו דבר" [19:50]. CoreDNS, kube-proxy, VPC CNI ו-CSI drivers אינם באותו מעמד כמו מיקרו-סרוויס של האפליקציה, ואם הם יושבים על אותו node pool עם ה-tenants, אירוע במקום אחד מפיל את הקלסטר כולו.

התרחיש הקונקרטי: אפליקציה שמשובצת על אותו node וצורכת RAM ו-CPU בכמות גדולה, ה-kubelet וה-scheduler מבצעים preemption — "מחליטים להעיף את הקור די-אנס. ראינו מה קורה כשדי-אנס נופל, זה לא דבר כיף בחיים" [19:50].



CoreDNS על ה-pool המשותף לצד tenant — tenants רעב מפנה אותו ו-DNS נופל ברמת הקלסטר; מה שמחזיק הוא pool ייעודי עם taint

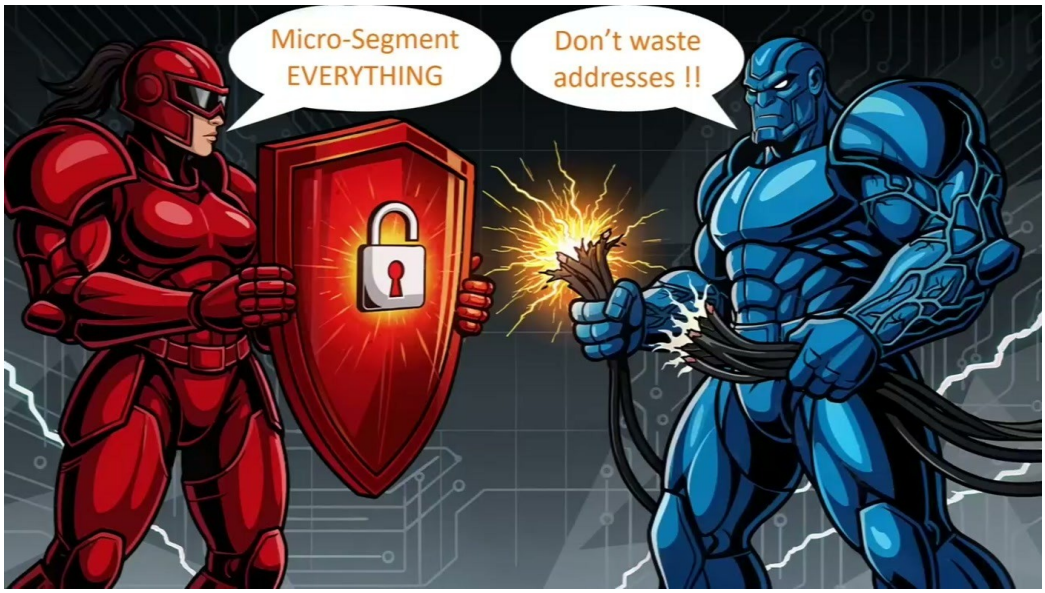
הפתרון הוא node pool ייעודי לאינפרסטרקטורה, באמצעות taints, tolerations ו-affinity: "אנחנו יכולים להגיד כל הדברים שהם mission critical תלכו לשם" [19:50].

אופיר גם חזר בחדות על ScheduleAnyway — מאותה סיבה: "schedule anyway אל תנסו את זה בבית", כי "זה לשים את כל הביצים בסל אחד" [19:50]. זו נקודה שבה מומחה ה-operations וה-Solutions Architect נחלקים בטון: אסף מציג אותה כ-trade-off לגיטימי שצריך להבין, אופיר מציג אותה כדיפולט שצריך לשנות.

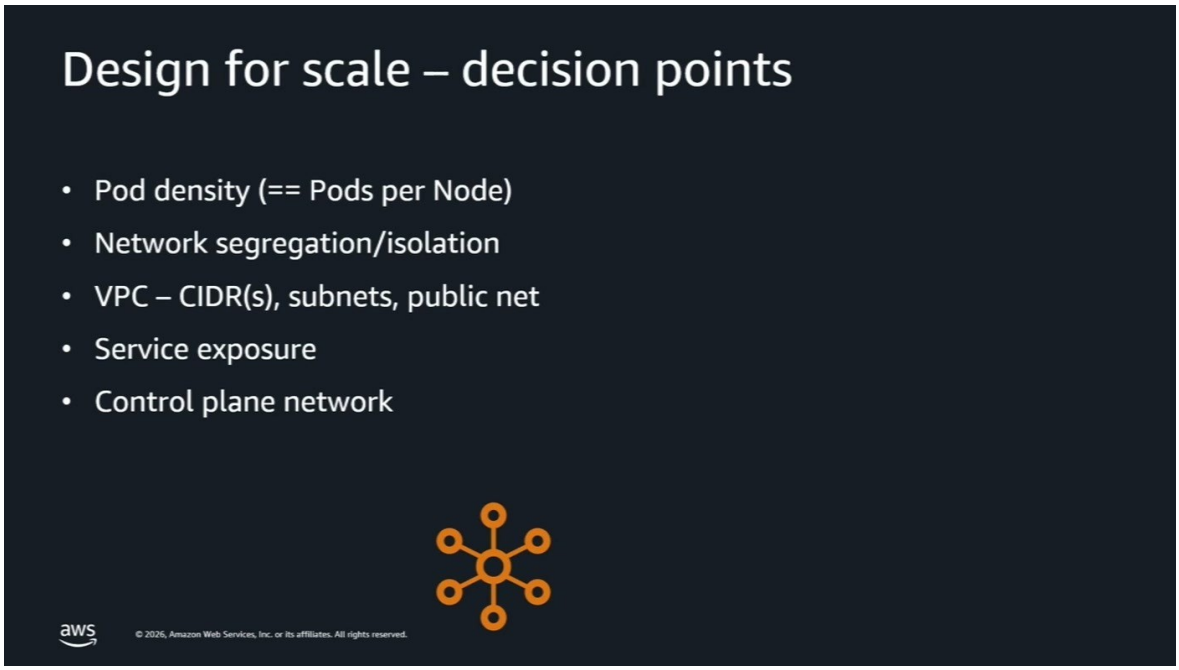
השקף האחרון שלו בסבב הזה חוזר לאפליקציה: גם אם התשתית מושלמת, מעבר מ-v1 ל-v2 צריך להיות gradual rollout — "זה ארגו rollouts שנותן לכם דוגמה" [21:21]. המשפט שנשאר על השקף בהמשך: rollback שלא תרגלתם הוא outage.

7. Networking: מ-VPC בסיסי ל-prefix mode

אסף פתח את החלק הזה בהצגת המתח הארגוני שמופיע בכל פרויקט. אנשי אבטחה: "תעשה Micro-Segment Everything הכל ב-Unique Area שלו ו-Isoleated". אנשי רשת: "אתה מבזבז לכתובות בוא, אני באנטרפריז, יש לי מחסור" [21:21]. השאלה היא איך מייצרים middle ground.



המתח כפי שהוצג על הבמה — שתי דרישות לגיטימיות שמושכות לכיוונים הפוכים



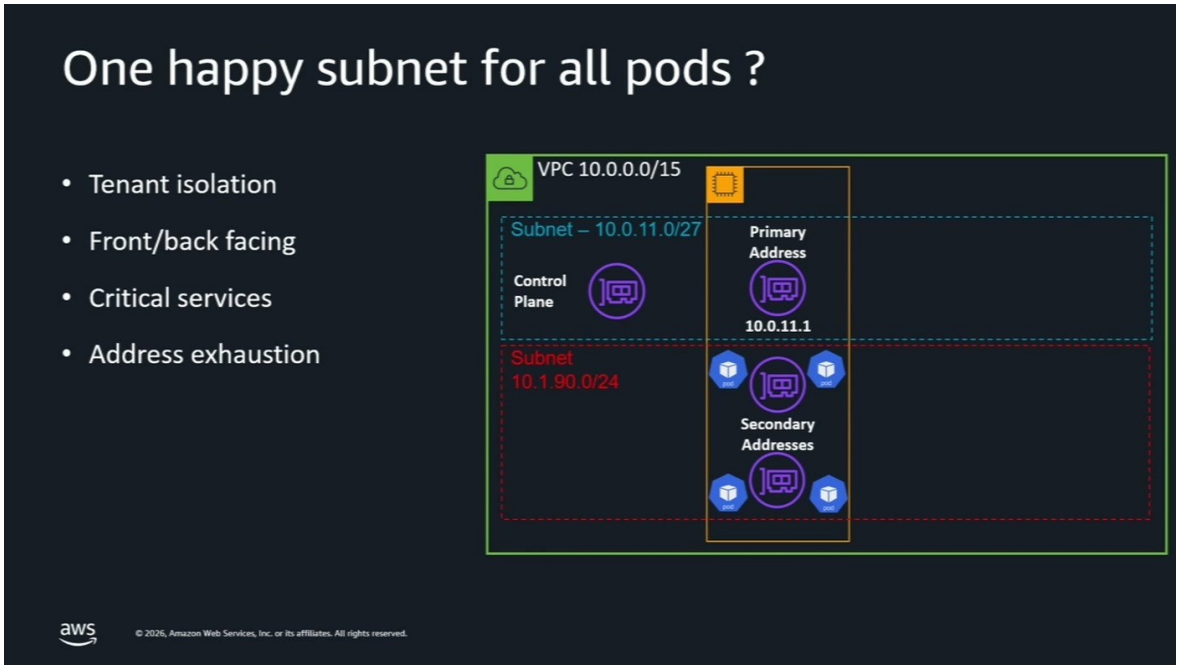
נקודות ההחלטה שעליהן נבנה החלק הזה: צפיפות פודים לנוד, הפרדת רשת, תכנון CIDR ו-subnets, חשיפת שירותים ורשת ה-control plane

הבנייה ההדרגתית (הדוגמאות מבוססות VPC CNI, והעקרונות ניתנים להעברה גם ל-CNI אחרים): VPC עם subnet, endpoints של ה-control plane, EC2 שמקבל ENI עם primary address — זו כתובת ה-node. ואז ה-CNI מקצה secondary IP addresses על ה-ENI, וכל פוד חדש מקבל כתובת מתוך ה-pool. כשה-pool נגמר, נוצר ENI נוסף.

השאלה שמפילה את המודל הזה היא שאלת הקשר, לא שאלת קיבולת:

> **— אסף קנדר [22:53]** נודים מדברים עם הקונטרול פליין נכון, זה פרוסס של מערכת הפעלה, אבל פודים זה כבר אפליקטיבי

הצעד הראשון הוא לכן subnet נפרד: ה-nodes על subnet אחד, הפודים מקבלים secondary ENIs על subnet אחר. אסף הוסיף גם התייחסות לביקורת הצפויה על בזבוז — בדוגמה שלו ה-subnet של הפודים רחב יותר וזה של ה-nodes צר יותר, "כי בהגדרה היו פחות נודים מפודים" [24:27]. זה לא פותר את הבעיה, אבל זה תכנון ולא מקריות.

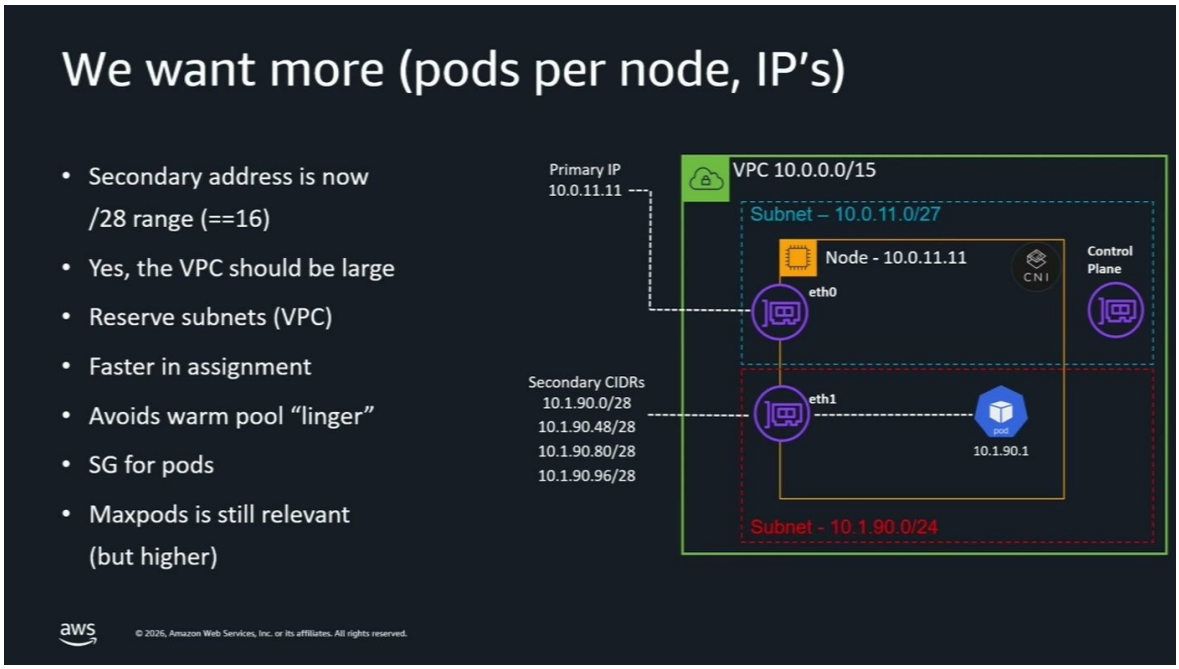


ארבע הסיבות לפצל את הפודים ליותר מ-subnet אחד: בידוד tenants, הפרדת front/back, שירותים קריטיים, ומיצוי כתובות

הצעד השני הוא subnets ייעודיים לפי הקשר. מבחינת המימוש ב-VPC CNI חשוב להבין את המגבלה: ההפרדה מתבצעת ברמת ה-node. "הדרך לייצר את זה זה בין נודים שונים" [25:57] — nodes שיושבים על הרשת האחת ו-nodes אחרים שיושבים על השנייה.

הצעד השלישי נועד לשבור את תקרת הכתובות:

> **— אסף קנדר [25:57]** מה אם אותו secondary IP address היה עכשיו לא אחד, אלא 16 כתובות, 1628, range, נקרא prefix mode



prefix mode — כל slot הופך ל-/28, בתמורה ל-VPC גדול, שריון subnets, ומגבלות שנשארות על SG for pods ו-maxPods

מה צריך לדעת לפני שמפעילים prefix mode

- **ה-VPC צריך להיות גדול.** כל slot של secondary address הופך ל-range של 16 כתובות, והצריכה עולה בהתאם.
- **חובה לשריין את ה-subnet:** "אפשר לעשות רזרבשן של הסאבנט וצריך לעשות את זה" [27:32] — כדי ש-AWS לא תקצה כתובות מתוך ה-prefixes האלה ל-EC2 אחרים ב-VPC.
- **assignment מהיר יותר ופחות תחרות:** ה-prefixes יושבים ברמת המכונה וה-CNI מקצה מתוכם, כך שאין מאבק בין nodes על secondary addresses.
- **התנגשות עם Security Groups for Pods:** אפשרי, אך דורש קונפיגורציה ספציפית מאוד — "צריך שנייה קיים בדוקומנטיישן" [27:32].
- **maxPods עדיין רלוונטי:** הסקריפט המוכר לחישוב מספר הפודים המקסימלי לאינסטנס ייתן מספר גבוה יותר במצב prefix, "זה לא [29:06] "unlimited.

8. Supply Chain ו-Security

הסבב השני של אופיר פתח בשאלה לקהל על supply chain attacks — Shai-Hulud, ה-Axios וה-Trivy — ורק כשליש מהידיים עלו. משם הוא בנה טיעון בשלושה שלבים: מיון החולשות לפי הקשר, אבטחת מקור החבילות והבסיס, ואכיפה ברגע הכניסה לקלסטר.

8.1 הקשר במקום ערימת אלרטים

הביקורת על הכלים המסורתיים הייתה חדה: הם מייצרים נפח שאי אפשר לפעול לפיו. "אתם מקבלים משהו כמו 100 אלף אלרטים ככה כלי SIM, SIEM היו עובדים הנה לוגים, אנחנו לא יודעים לעשות עבודת פרודקט טובה אז תתמודדו" [30:36].

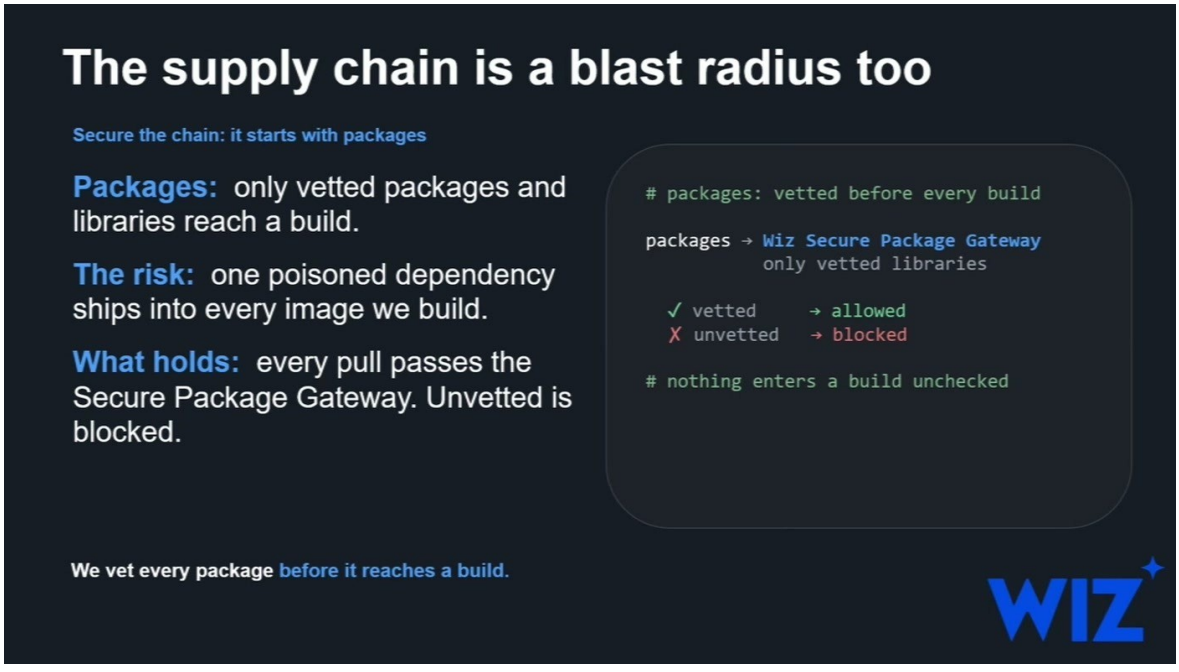
הגישה החלופית שהוצגה היא מודל גרף. "אנחנו מסתכלים על קוברנטיס כענן משלו" [30:36] — workloads, identities, הגדרות רשת, ומעליהם AWS — ואז שואלים שאלות על מסלולים בגרף, מנקודת מבט של תוקף מהאינטרנט הפומבי.

> **— אופיר כהן [30:36]** יכול להיות שהחולשה עצמה היא קריטיקל אבל הקונטקסט הוא לא מעניין כי אף אחד לא יכול להשמיש אותה

הדוגמה: Log4j בפוד שאינו חשוף — אין ingress controller, אין load balancer, אף אחד לא מגיע אליו. מנגד, אותה חולשה בפוד שנגיש מהאינטרנט ושממנו יש גישה ל-S3 עם PII "זה הדבר הכי גרוע שיש, כי זה Business Reputation." [30:36]. הקריטיות אינה תכונה של ה-CVE, אלא של המסלול.

8.2 מאיפה מגיע מה שאתם מתקינים

אופיר תיאר את מנגנון האמון המקובל — סטארים ב-GitHub, מספר הורדות ב-npm, קומיט שנראה תמים — ואת הפער בינו לבין מה שיורד בפועל: "יש מין ליפ אוף פייפ כזה, שאנחנו מאמינים שמה שאנחנו חושבים שאנחנו מתקינים, זה מה שאנחנו מתקינים, זה לא תמיד המצב" [32:09]. הדוגמאות שהוזכרו: גניבת npm token של maintainer, ו-compromised pull request target ב-CI/CD.



שער לחבילות: dependency מורעל אחד נכנס לכל אימג' שנבנה, ולכן כל pull עובר סינון לפני ה-build

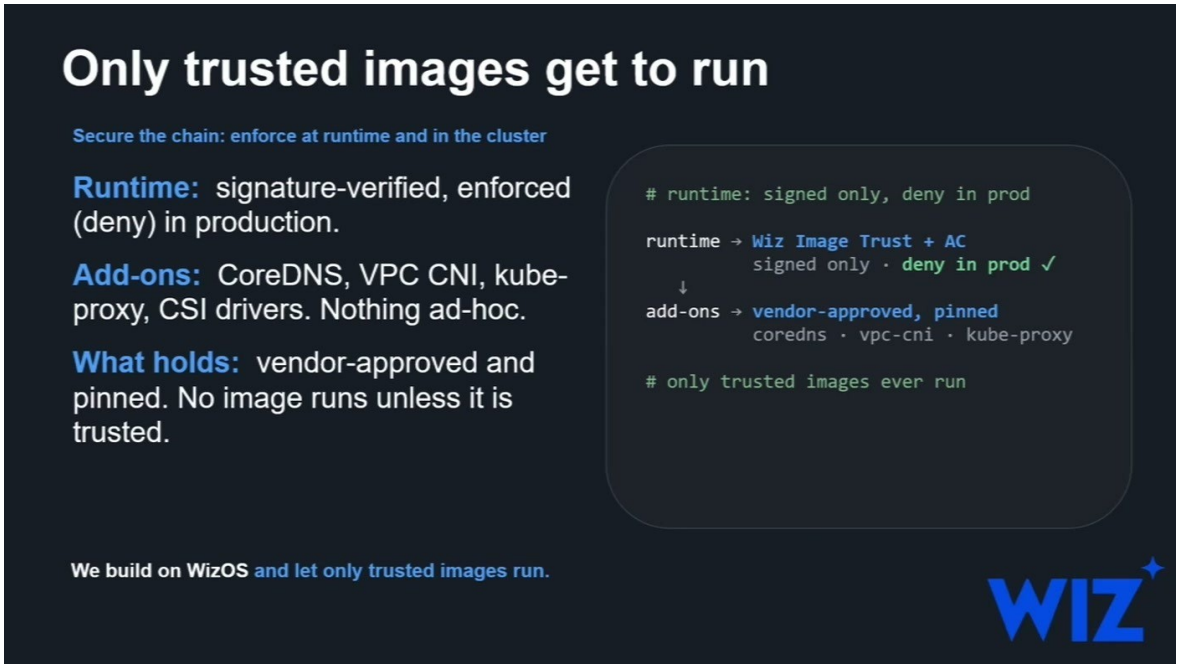
משם לשכבת ה-base image, והנקודה החזקה ביותר בחלק הזה. ארגונים שנדרשים ל-FedRAMP, SOC 2 או חוזי DoD מחויבים לאפס חולשות קריטיות ו-SLA לתיקון — אבל אפס חולשות ב-scan אינו מעיד על מקור האימג'. אופיר המחיש זאת בהיפותזה מכוונת-הגזמה: אימג' שנסרק ויצא נקי לחלוטין, אבל "אתם לא יודעים מי בנה אותו, ולמי יש גישה לדחוף" [33:42].

> **— אופיר כהן [35:14]** לשאול שאלות חברים, לשאול שאלות מאיפה הגיע סופטוור שמעליו אני בונה דברים, להיות חשדנים והחיים קצרים מדי כדי לתקן חולשות

המסקנה המעשית שהוצגה: עדיף base images מהימנים מאשר "צוות של 100 אנשים שמתקנים חולשות" [35:14]. פחות חבילות בבסיס פירושן פחות CVEs לאורך כל הצי.

8.3 אכיפה בכניסה לקלסטר

גם shift left מושלם אינו מספיק, כי Kubernetes אינו יודע איך נבנה מה שדוחפים אליו. התרחיש: עובד עם service account token שמריץ kubectl apply מהבית — "קוברנטיס לא אכפת לו שאתם בניתם בצורה שהיא "secure" [35:14].



אכיפה ב-runtime: חתימות מאומתות במצב deny בפרודקשן, ו-add-ons מאושרי ספק ונעוצים בגרסה — כולל CoreDNS, VPC CNI, kube-proxy ו-CSI

הכלי הוא admission controller — "זה פשוט policy engine זה כמו אנטיבירוס מודרני, כן, לקוברנטיס" [36:45]. כל apply פונה אליו ומקבל תשובה בינארית על הפוד ועל האימג'. המסגרת הכוללת, בניסוחו: "דיפנס אינדפט, ככה עובד סקירותי אנחנו עושים סקירות בשכבות" [36:45].

9. מה לוקחים מכאן

אסף סגר בארבעה עקרונות לאימות רזיליינס, שמסכמים גם את הטון של כל הסשן — הנחות לא נבדקות אינן רזיליינס.

- **לבדוק, לא להניח:** "אנחנו רוצים לבדוק את המערכת ולא להגיד כן כן רזילינט הכל בסדר בלי שבדקתי" [38:15].
- **יותר מפעם אחת:** הסביבה משתנה כל הזמן, ולכן בדיקה חד-פעמית אינה מספיקה.
- **אוטומטי ו-always on:** בסקייל אי אפשר להסתמך על הרצה ידנית מדי חודש — testing רציף, ובדיקת רזיליינס בלייב.
- **ללמוד מכל משבר:** "[38:15] to let a good crisis go to waste so don't" — אירוע הוא בדיוק המקום ללמוד ממנו ולשפר ארכיטקטורה ו-operations.

פעולות קונקרטיות שנגזרות מהסשן

תחום	מה לבדוק / לעשות
Scheduling	לבדוק את ערכי maxSkew ו-whenUnsatisfiable בפועל בקלסטר, ולהחליט במודע בין ScheduleAnyway ל-DoNotSchedule
Karpenter	לוודא ש-ScheduleAnyway אינו מנטרל את הטריגר; להגדיר AMI pinning, expireAfter ו-disruption budget
PDB	להוסיף Pod Disruption Budgets לכל workload קריטי, ולוודא ש-probes מוגדרים נכון
Node pools	להוציא CoreDNS, kube-proxy, VPC CNI ו-CSI drivers ל-pool ייעודי עם taint + toleration + affinity
Networking	subnet נפרד ל-nodes ולפודים; subnets ייעודיים לפי הקשר; לשקול prefix mode עם שריון subnet
Supply chain	להגדיר מקור מהימן ל-base images ול-third party; סריקה ב-CI/CD אם עדיין לא קיימת
Admission	admission controller שאוכף חתימות ו-add-ons מאושרים, במצב deny בפרודקשן
Deployment	gradual rollout (למשל Argo Rollouts) ותרגול rollback לפני שצריך אותו

מקורות שהוצגו בסשן

- EKS Best Practices Guide
- Scaling Amazon EKS infrastructure to optimize compute, workloads, and network performance (Prescriptive Guidance)
- Designing for high availability and resiliency in Amazon EKS applications (Prescriptive Guidance)

10. מילון מונחים

מונח	פירוש בהקשר הסשן
Topology Spread Constraints	מנגנון ב-Kubernetes לפיזור פודים על פני zones ו-hosts, לפי maxSkew ומדיניות whenUnsatisfiable
maxSkew	הפער המותר בכמות הפודים בין topology domains. דיפולט ברמת קלסטר: 3 ל-hostname, 5 ל-zone
ScheduleAnyway / DoNotSchedule	שתי המדיניויות ב-whenUnsatisfiable: לשבץ בכל מחיר גם על חשבון הפיזור, או להשאיר את הפוד ב-Pending
Karpenter	קונטרולר להקצאת nodes ב-EKS. מופעל כתגובה לפודים ב-Pending, ומטפל ב-expiry, consolidation ו-drift
PDB (Pod Disruption Budget)	הגבלה על כמות הפודים שניתן להפיל ב-voluntary disruption. אינו מגן מפני תקלה בלתי-רצונית
Voluntary / Involuntary disruption	הפרעה יזומה (שדרוג, node maintenance) מול הפרעה שאינה בשליטתנו (תקלת חומרה, אובדן AZ)
Grey failure	כשל חלקי שבו רכיב נראה זמין אך אינו מתפקד — למשל node שעדיין רץ ב-AZ שאין בו capacity

מונח	פירוש בהקשר הסשן
VPC CNI	תוסף הרשת של EKS שמקצה לפודים כתובות IP מתוך ה-VPC, דרך secondary addresses על ENI
ENI / Secondary IP	ממשק רשת אלסטי וכתובות משניות עליו — המנגנון שממנו הפודים מקבלים IP
Prefix mode	הקצאת /28 (16 כתובות) לכל slot במקום כתובת בודדת, כדי להגדיל צפיפות פודים לנוד
Taint / Toleration / Affinity	המנגנונים להפניית workloads קריטיים ל-node pool ייעודי והרחקת אחרים ממנו
Admission controller	policy engine שמאשר או דוחה כל בקשת apply לקלסטר — לדוגמה אכיפת חתימות על אימג'ים
Toxic combination	צירוף של ממצאים שכל אחד לבדו אינו קריטי, אך יחד מייצר מסלול תקיפה ממשי
Supply chain attack	פגיעה בשרשרת האספקה של הקוד — חבילה, maintainer token או pipeline — לפני שהקוד בכלל הגיע אליכם