לפשט את מסע ה-Kubernetes עם EKS Capabilities

AWS Summit Tel Aviv 2026 ,סיכום מושן — TLV-SVS306

צחי מזרחי וגיא מנחם, Solutions Architects, AWS | דניאל רובין, Staff DevOps Engineer, Gong

10 בספטמבר 2026 | משך: כ-31 דקות | הופק מהקלטת המושן

מסלול: AWS for Containers and Platform Engineering | רמה: 300

על המסמך הזה

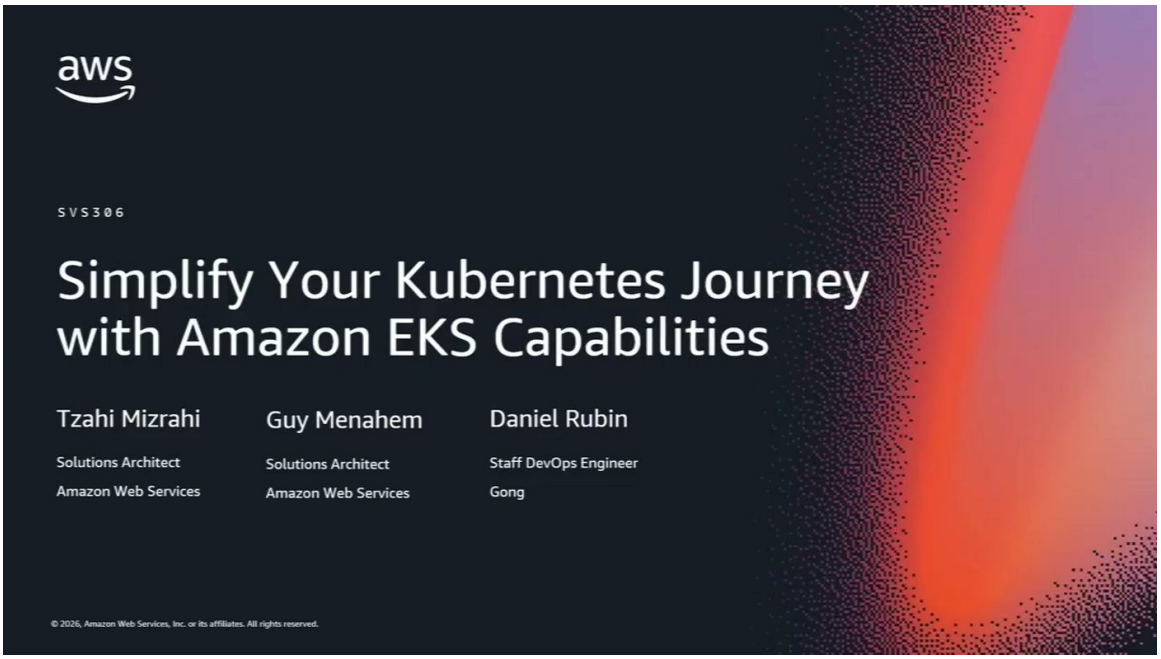
המסמך מסכם את הסשן TLV-SVS306 מתוך AWS Summit Tel Aviv 2026, בהתבסס על ההקלטה המלאה שפורסמה באתר הסאמיט. הוא נבנה מתמלול אוטומטי של ההקלטה ומהסליידים שחולצו מהווידאו עצמו, כך שאפשר לחזור לתוכן בלי לצפות שוב בשלושים ואחת הדקות. חותמות הזמן בגוף המסמך מפנות לנקודה המדויקת בהקלטה.

איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** ארבע דקות קריאה. מה הוכרז, מה הודגם, ומה רלוונטי לצוות פלטפורמה.
- **פרקים 2–3 — הרקע וההכרזה:** למה ניהול צי קלאסטרים נשבר, ומה בדיוק EKS Capabilities מציע.
- **פרק 4 — Argo CD מנוהל:** מה מקבלים, ואיך זה נראה מול הקלאסטר.
- **פרק 5 — הקייס של Gong:** 60 קלאסטרים, 2400 קונפיגורציות, וגישת Deploy-Merge.
- **פרקים 6–8 — ACK ו-kro:** ניהול משאבי AWS מתוך הקלאסטר, והרכבתם לאבסטרקציה אחת.
- **פרק 9 — מה לוקחים מכאן:** מסקנות ונקודות להמשך, ומילון מונחים.

> **הערת מקור** המסמך הופק מתמלול אוטומטי של ההקלטה. התמלול מדויק בקטעים המקצועיים, אך שמות ומונחים לועזיים מופיעים בו בשיבוש פונטי (למשל "קרו" = kro, "אק" = ACK, "סינגליידישן" = Singlidation) והם תוקנו בגוף המסמך מול הסליידים. הציטוטים מובאים כפי שהם נשמעים בהקלטה בחותמת הזמן המצוינת. כל המספרים במסמך מקורם בסליידים או בדברי הדוברים.

1. תקציר מנהלים



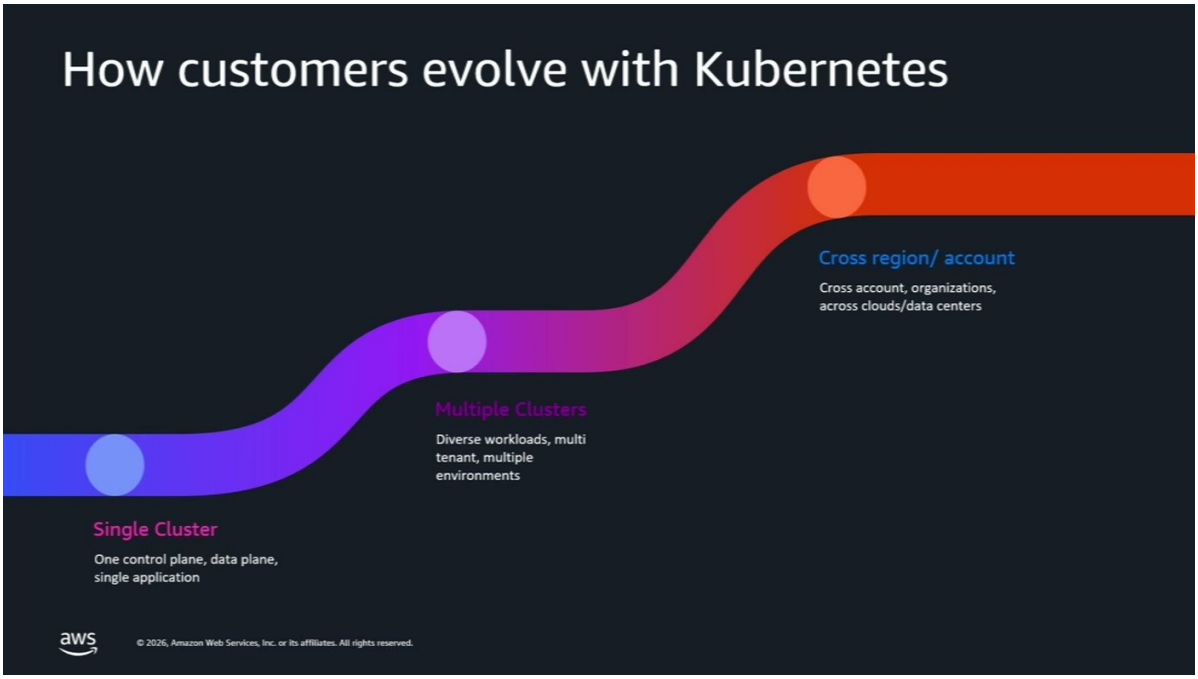
שקופית הפתיחה: שלושה דוברים — שניים מ-AWS ואחד מ-Gong.

הסשן הציג יכולת חדשה בשם EKS Capabilities: סט רכיבי פלטפורמה מעולם ה-CNCF — Argo CD, ACK ו-kro — שרצים כשירות מנוהל בתוך ה-EKS Control Plane במקום על הנודים של הלקוח. המסר המרכזי הוא שהשכבה שבין Kubernetes המנוהל לבין האפליקציה, אותה "פלטפורמה" שכל ארגון בונה לעצמו, היא נקודת החיכוך האמיתית — ו-AWS מציעה לקחת אותה עליה.

- **נקודת השבירה היא בין 10 ל-40 קלאסטרים.** עד עשרה קלאסטרים גישת GitOps רגילה עובדת; מכאן והלאה מתחילים לרדוף אחרי שדרוגי גרסאות ופאצ'ים. הדובר מ-AWS: "השיטות שבחרנו לעבוד בהם הם כבר לא כל כך מספיקות" [3:02].
- **הרכיבים יוצאים מהקלאסטר אל ה-Control Plane.** "היכולות האלה ירוצו ב-EKS Control Plane ולא על הקלאסטר שלכם" [4:34]. הלקוח רואה רק את ה-CRDs — ה-API — והקונטרולרים עצמם מנוהלים ומתוקנים על ידי AWS.
- **אין נעילת ספק בכוונה.** השקופית מגדירה את הגישה כ-"Loosely opinionated": כלים בסטנדרט פתוח, משאבים סטנדרטיים ב-Kubernetes, ו-"easy migrations, no proprietary components".
- **Gong מריצה את הדפוס הזה היום על 60+ קלאסטרים.** דניאל רובין, Staff DevOps Engineer: "תדמינו שהגענו לסביבות של 60 קלסטרים עם לפחות איזה 40 2400 — [12:11] "infra components+ קונפיגורציות לפי השקופית.
- **התובנה התפעולית של Gong: לפרוס לפני ה-merge, לא אחריו.** הם בנו ApplicationSet Generator משלהם שמריץ את השינוי מענף ה-PR על קבוצות קלאסטרים מדורגות. "אני יודע בוודאות שמה שפרסתי ובדקתי תקין" [16:43].
- **הדמו: מערכת אג'נטית שלמה ב-14 שורות YAML.** שמונה משאבים נפרדים נעטפו בשלושה RGDs ובאבסטרקציה אחת מעליהם — "מניפסט פשוט של קוברנטיס 14 שורות של ימל וזה הכל" [22:46].

2. איפה נשבר מודל ניהול הקלאסטרים

הפתיחה הייתה היסטורית: פעם ניהלנו הכול מהדאטה סנטר ועד האפליקציה, ואז הגיעו EC2 ו-VPC, ואז EKS עם Control Plane ונודים מנוהלים. "מה שנשאר לנו זה באמת לבנות את הפלטפורמה" [0:00] — השכבה שמעל Kubernetes ומתחת לאפליקציה. הטענה היא שהשכבה הזו היא שקובעת אם הארגון יגדל: "המפתחים מתעסקים בעיקר בביזנס הארגוני, ולא בתשתיות או בלרדוף אחרי פאצ'ים או סקיוריטי" [1:30].



שלושת השלבים בהתפתחות לקוחות: קלאסטר יחיד, ריבוי קלאסטרים, ואז פריסה חוצת אזורים וחשבונות.

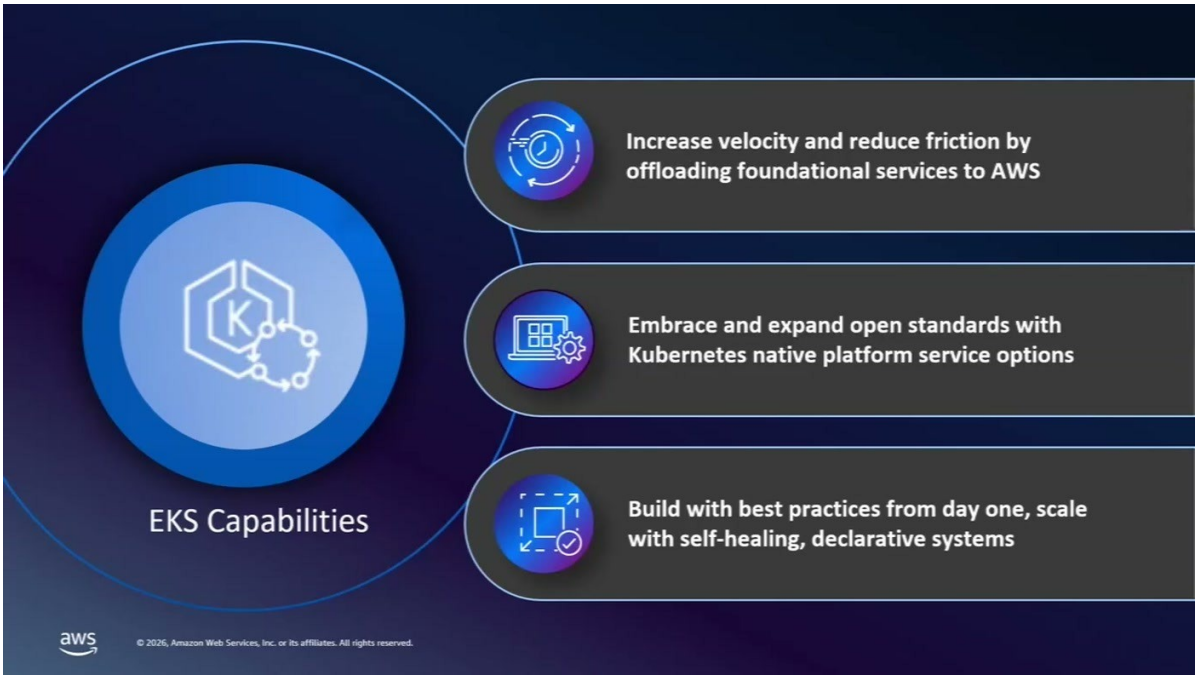
התיאור המעשי: ארגון מתחיל עם קלאסטר אחד או שניים — prod, dev, test — ומנהל אותם דקלרטיבית ב-Git עם Terraform, Pulumi, CDK או CloudFormation. הצמיחה מוסיפה קלאסטרים, חשבונות ואזורים, ואז "השיטה הזאת של לבוא ולשים קלסטר עם הקונפיגורציה שלו והאפליקציה שרץ עליו, קצת מתחילה להישבר" [3:02]. נקודת החיכוך המשמעותית ביותר, לפי הדובר, היא המעבר מחמישה עד שמונה קלאסטרים לעשרים, שלושים וארבעים.

בדרך הוצגו שני שלבי ביניים שרוב הארגונים מכירים: Helm כ"כלי הכי נפוץ לדיפלוימנט של third-party tools" [3:02], ואחריו מנגנון ה-add-ons של EKS שמנהל את ה-life cycle דרך ה-API. החסרון שנותר בשניהם זהה — "עדיין התשתית עצמה ירוץ על הנודים שלנו" [4:34], כלומר הלקוח עדיין משלם קומפיוט, מתחזק ומטליא.

המספר שכדאי לזכור לפי השקופית, Amazon EKS מריצה עשרות מיליוני קלאסטרים ברחבי העולם. זהו הבסיס לטענה שהתשתית המרכזית שרצה עבור כולם כדאית יותר מאותה תשתית שכל ארגון מתחזק בעצמו.

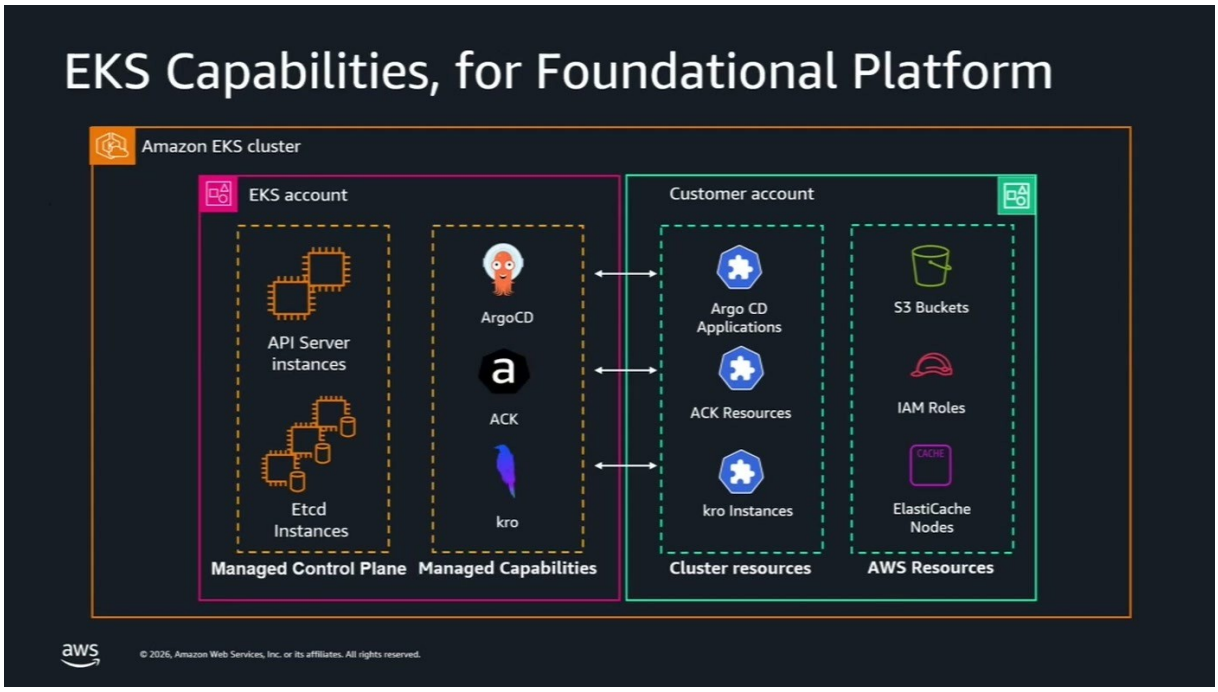
3. מה זה EKS Capabilities

ההגדרה שניתנה: סט יכולות מנוהלות במלואן, דקלרטיביות וגיטופסיות, שרצות ב-EKS Control Plane ונצרכות דרך API. שלושת העקרונות המובילים שהוצגו הם הורדת חיכוך מצוותי הפיתוח, הישענות על open source ולא על פתרונות פרופרייטריים, ואספקת הרכיבים לא ישר מה-Git repo של הפרויקט [6:05] אלא עם best practices אפויים פנימה ואינטגרציות מוכנות לשירותי AWS.



שלושת עקרונות היסוד כפי שהוצגו: הפחתת חיכוך, אימוץ סטנדרטים פתוחים, ובנייה עם best practices מהיום הראשון.

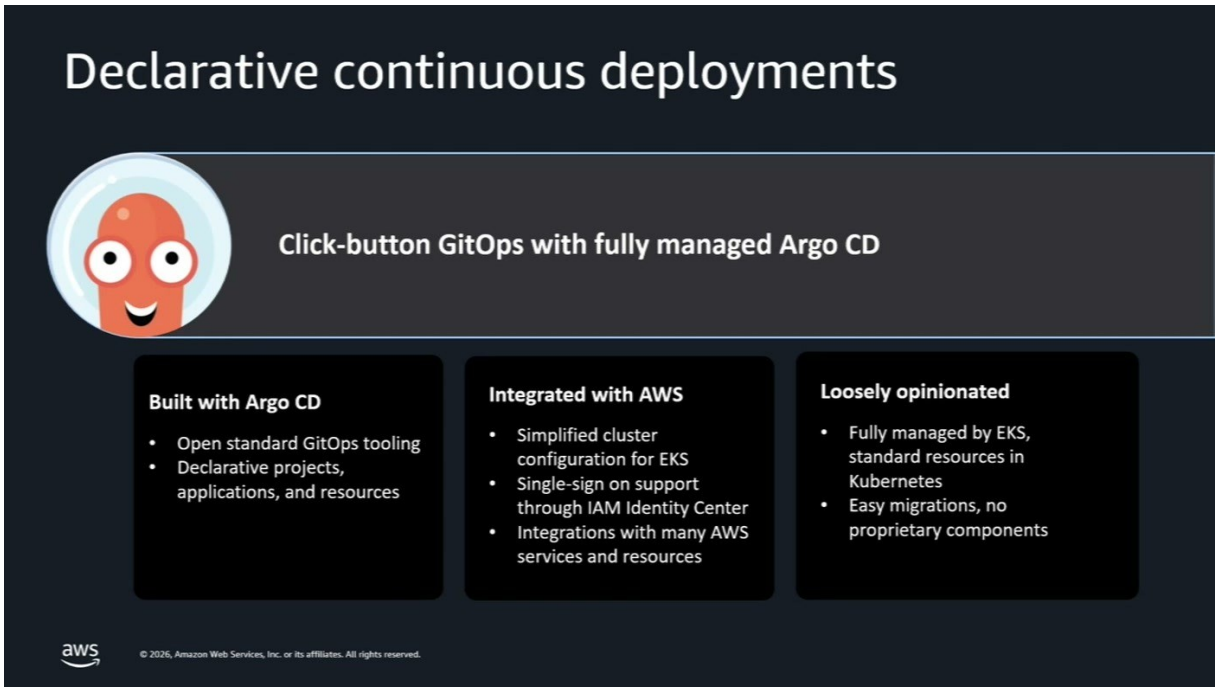
שלוש היכולות שנמנו בסשן: Argo CD ל-GitOps, ACK (AWS Controllers for Kubernetes) לניהול משאבי AWS כאובייקטים של Kubernetes, ו-kro (Kube Resource Orchestrator) לאורקסטרציה של משאבים לאבסטרקציה אחת.



הקונטרולרים של Argo CD, ACK ו-kro יושבים לצד ה-API Server וה-etcd בחשבון של EKS; בחשבון הלקוח נשארים רק המשאבים.

ההשוואה לתרשים הקודם, שבו אותם רכיבים רצים self-managed בתוך namespace אצל הלקוח, היא לב הטיעון: כשמתגלה security vulnerability באימג' או ב-dependency, "אנחנו צריכים ללכת עכשיו לעשרות רבות של קלסטרים וממש לעשות את הפאץ'" [7:36]. במודל המנוהל "הרכיבים האלה יוצאים החוצה ומנועלים ב-Control Plane, אתם לא רואים אותם, אתם יכולים להשתמש ב-API שלהם" [7:36].

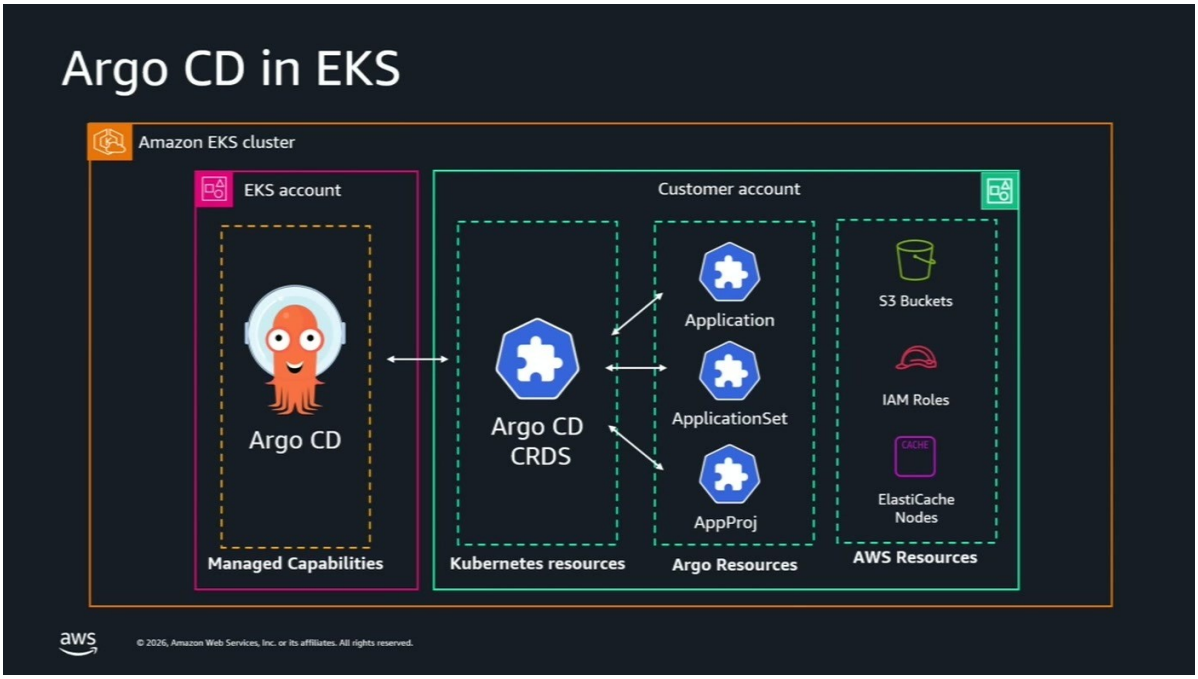
4. Argo CD כיכולת מנוהלת



שלוש עמודות המפרטות מה נשאר תקני, מה מתווסף מצד AWS, ומה מכוון להגירה חלקה.

Argo CD הוצג כ"אחד הכלים הכי נפוצים היום בעולמות ה-CD של ה-CNCF" [6:05], והבחירה בו כיכולת הראשונה נגזרת מהשאלה "מה הכלי הראשון שאנחנו מתקינים על הקלאסטר?". מה שמוסיפה הגרסה המנוהלת: גישה לקלאסטרים מרוחקים בלי לפתור בעיות נטוורקינג בדרך, ו"אינטגרציה מלאה עם IAM או עם ECR" [7:36]. המנגנון עצמו נשאר זהה, ולכן ההגירה פשוטה — "בגלל שאנחנו משתמשים באותו API עם הדברים הרבה יותר פשוטים" [7:36].

תזכורת למי שלא מכיר: Argo CD מסתכל על ה-Git repo, שולף את המצב בפועל מהקלאסטר, משווה ומתקן. הדוגמה שניתנה מהבמה: "אם בקלסטר רצות שתי רפליקות של אחד הפודים שלנו וביקשנו שלוש, ארגו-CD ידעה אוטומטית ללכת לשם ולעשות את התיקון" [9:08]. בתצורת hub-and-spoke, מופע אחד מנהל טווח רחב של קלאסטרים — dev, staging ו-production — ממקום אחד.



ה-CRDs שנחשפים בקלאסטר — Application, ApplicationSet ו-AppProject — מול הקונטרולר שרץ מחוץ לו.

- **Application:** סט שלם של משאבים שה-Argo CD יודע להחיל על הקלאסטר — deployments, services וכל משאב אחר.
- **ApplicationSet:** "מאפשר לנו להריץ אפליקציות בסקייל בעזרת ג'נרטורים" [9:08] — במקום לכתוב אפליקציות מראש, מייצרים אותן.
- **AppProject:** ניהול פרויקטים והרשאות למשתמשים, ובכך מולטי-טננטיות בתוך סביבת Argo CD [10:40].

5. הקייס של Gong: Zero-Risk Infrastructure at Scale

דניאל רובין, Staff DevOps Engineer ב-Gong, עלה לבמה בדקה 10:40 כדי לתאר מימוש קיים. הרקע: Gong משרתת מעל 5,000 חברות כפלטפורמת revenue מבוססת AI. ההתפתחות שלהם זהה לדפוס שתואר בפרק 2 — קלאסטר אחד, ואז staging, preprod ו-playground, ו"כל סביבה שחליטו" [10:40] — עד שניהול הגרסאות והיררכיית הערכים הפך לכאב.

היקף הבעיה בצד של Gong: קלאסטרים בכמה אזורים גאוגרפיים, כל אחד עם עשרות שירותי תשתית.

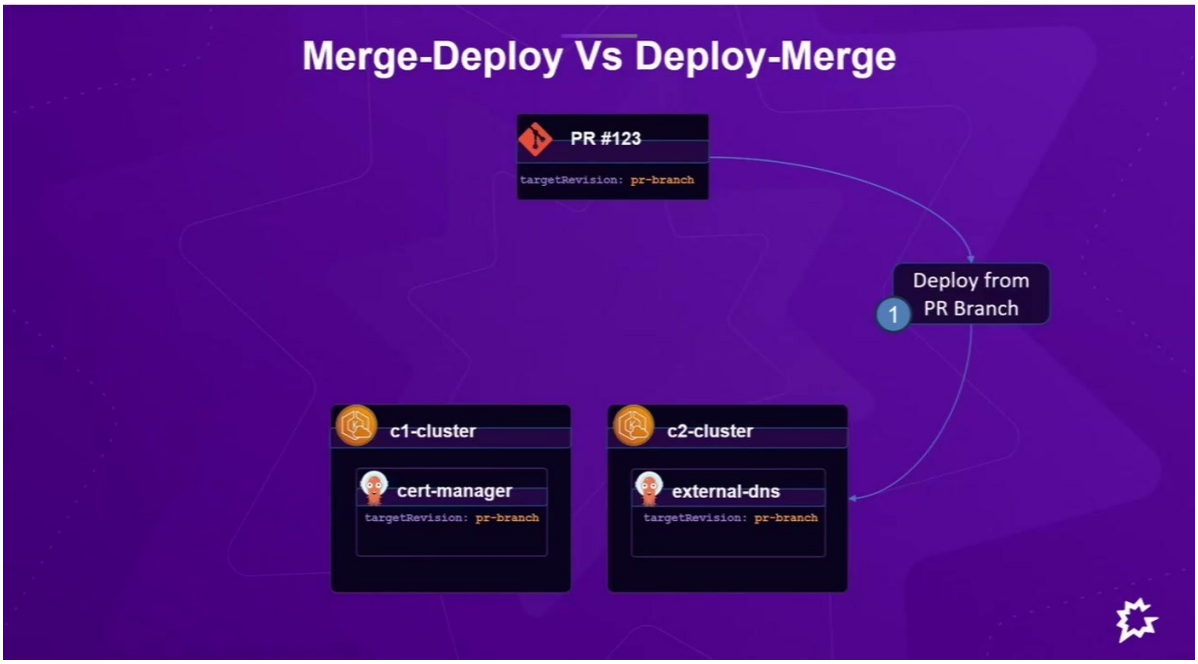
רובין מנה את הכאבים במפורש: ניהול versioning, כי "מגוון סבא מאוד מאוד אטרוגנית יוצר לנו המון המון בגים" [12:11]; היררכיית קונפיגורציות; ודרישות security, audit trail ו-guard rails. הפתרון שנבחר נקרא Singlidation — repo אחד כמקור אמת יחיד לכל קונפיגורציות הקלאסטרים.

מבנה ה-repo: תיקייה לכל namespace ורכיב תשתית, Chart.yaml אחד לרכיב, ו-values גלובלי עם override לכל קלאסטר.

מעל המבנה הזה הם לא יצרו Applications ידנית אלא ApplicationSet לכל namespace, עם ה-generator הנייטיבי git files by path: הוא סורק את תיקיית ה-namespace ו"מחלצים משם את האפליקציות של הקלאסטר המסוים" [13:41]. התוצאה — App-of-ApplicationSets.

5.1 הבעיה שנשארה: קצב השינויים

לפי רובין, שדרוג של רכיב תשתית אחד, למשל external-dns מ-1.15 ל-1.16, הוא אירוע שחוזר על עצמו: "הדבר הזה קורה אצלנו בשבוע לפחות 150 פעמים" [13:41]. בשיטה הטבעית — merge ל-main ואז auto-sync של כל האפליקציות — ה-blast radius גבוה מדי.



ההיפוך: השינוי נפרס מענף ה-PR אל קלאסטרים נבחרים, וה-merge קורה רק אחריו.

Deploy-Merge פירושו להעביר אפליקציות לענף, להריץ functional tests ו-end-to-end, ורק אחרי שכל הקלאסטרים עברו — לבצע merge. המימוש: pipeline ("בכל תול שתחליטו, "action, ergo workflow, whatever [15:13]) יוצר status check על ה-Pull Request, וקבוצות phase של קלאסטרים — playground, QA, staging וכן הלאה — מורידות את ה-blast radius.

החלק הטכני הוא generator מותאם אישית ל-ApplicationSet: ה-ApplicationSet מחזיק targetRevision מותנה, הקונטרולר פונה ל-generator שכתבו ב-Gong, וזה "מסתכל על כל הפיארים הפתוחים" [15:13], מחליט אם ה-phase שהוא משרת אמור לצאת לענף, ומחזיר מפה. "הוא בעצם מייצר לנו את האפליקציות אבל אחת מהם הוא מבין שהוא צריך להוציא לברנץ'" [16:43].

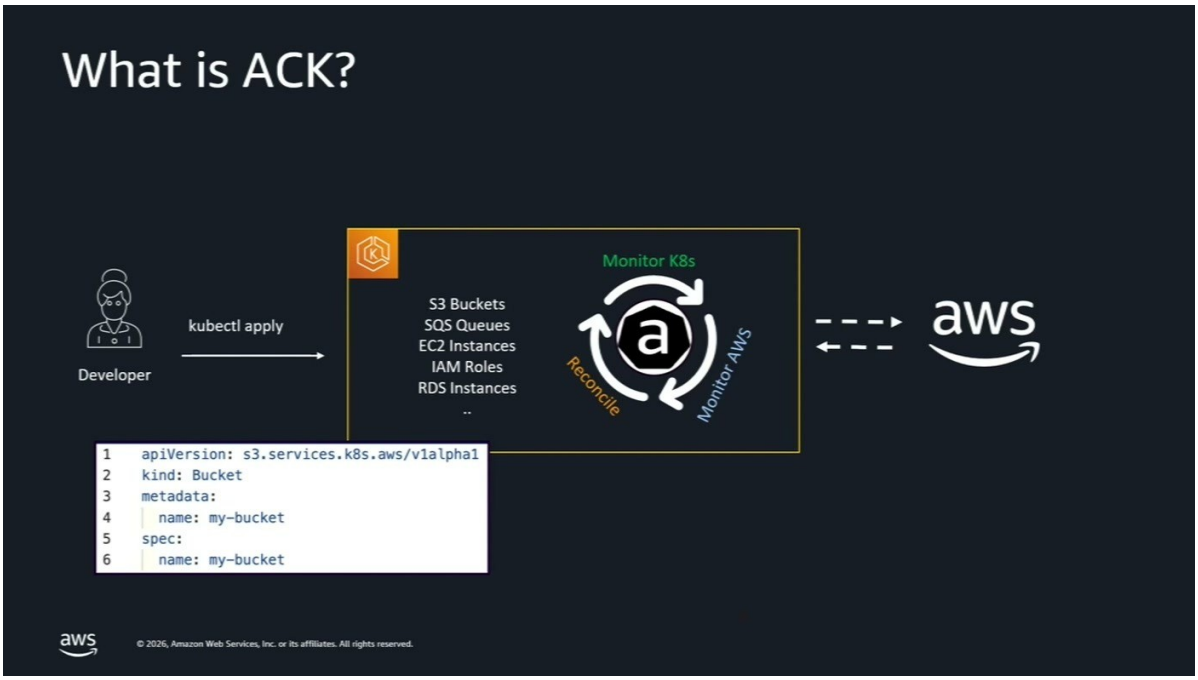
> **— דניאל רובין, Gong [16:43]** חברה, ורז'נינג, קונפיגורציות בין קלאסטרים זה אקוסיסטם אחר זה לוד אחר של טרפיק גם אם נצא לדמות אף פעם לא נוכל להיות 100% בשיטה הזאת אני יודע בוודאות שמה שפרסתי ובדקתי תקין

ההנמקה שלו לא הייתה טכנית אלא אנושית: הוא סיפר שדיבר עם לא מעט maintainers ב-CNCF — "בסוף זה אנשים כמוני כמוכם, זה לא העבודה שלהם" [16:43], הם תורמים קוד ומכניסים באגים — ומכאן הסיכום שלו, "Trust but Verify".

המסקנה	מה זה אומר בפועל
Singlidation	ריפו GitOps אחד כמקור אמת, מבנה תיקיות אחיד ודפוס values-override
Gradual Deployments	פריסה בשלבים מסביבות נמוכות כלפי מעלה, אחרי בדיקות
Validate Before Merge	בדיקה בפועל על קלאסטרים אמיתיים לפני ה-merge, לא אחריו

6. ACK: משאבי AWS כאובייקטים של Kubernetes

הדובר השלישי חזר לשאלה שנשארה פתוחה: "מה קורה עם כל מה שרץ מחוץ לקלאסטר" [18:14] — משאבי ענן כמו SQS, S3 buckets ובסיסי נתונים. הטיעון פשוט — Kubernetes כבר מספק מנגנון דקלרטיבי שבו מגדירים מצב רצוי במניפסט והקונטרולרים דואגים לשאר, אז למה שלא יחול גם על משאבי הענן.



המפתח מריץ kubectl apply, והקונטרולר קורא ל-API של AWS ומשמר reconciliation מתמשך.

הדוגמה שהוצגה: מניפסט מסוג Bucket שבו מגדירים שם ואפשר להוסיף הגדרות כמו lifecycle rules. מחילים אותו על הקלאסטר עם kubectl apply או דרך Argo CD, והקונטרולר של ACK מבצע את הקריאה ל-API. היופי, לדבריו, הוא שזה לא נעצר אחרי היצירה: הקונטרולר עובד "כל הזמן כדי לוודא שה-desire state תואם למצב הריסורס בפועל" [19:45].

ההשלכה המעשית אם מישהו נכנס דרך ה-Console ושינה משהו ב-S3 bucket, "ACK יזהה את הדריפט ויחזיר את הריסורס למצב הקודם" שהוגדר במניפסט [21:15] — מה ש"הופך את קוברנטיס להיות ה-source of truth של הקונפיגוריישן שלכם". לפי הסליידים, ACK כבר תומך בלמעלה מ-65 שירותי AWS.

7. kro: הרכבה לאבסטרקציה אחת

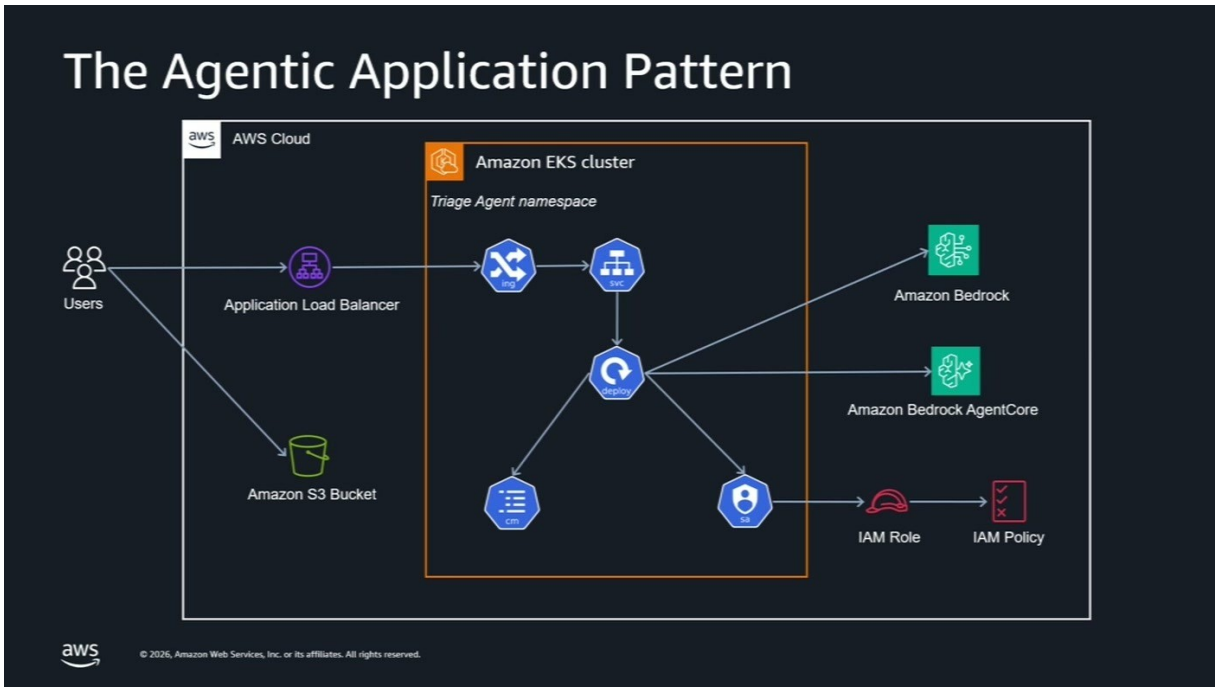
ההצדקה ל-kro נובעת ישירות מההצלחה של ACK: ברגע שמערכת שלמה כוללת Deployment, Service, ConfigMap, Account וגם משאבי ענן, "אנחנו מתחילים לדבר על לא מעט רסורסס שאנחנו צריכים לפרוס בשביל מערכת שלמה" [21:15].

מהנדס הפלטפורמה מגדיר RGDs; המפתח מקבל משאב יחיד להחיל.

kro מאפשר לצוות הפלטפורמה להרכיב יחד כמה משאבים — גם אובייקטים נייטיביים של Kubernetes וגם משאבי ACK — ליחידה אחת שנחשפת כ-CRD יחיד בקלאסטר, ונגישה דרך ה-API הרגיל של Kubernetes. זה מה ש"מאפשר לנו ולמפתחים לפרוס מערכת שלמה" באמצעות משאב בודד על הקלאסטר [22:46].

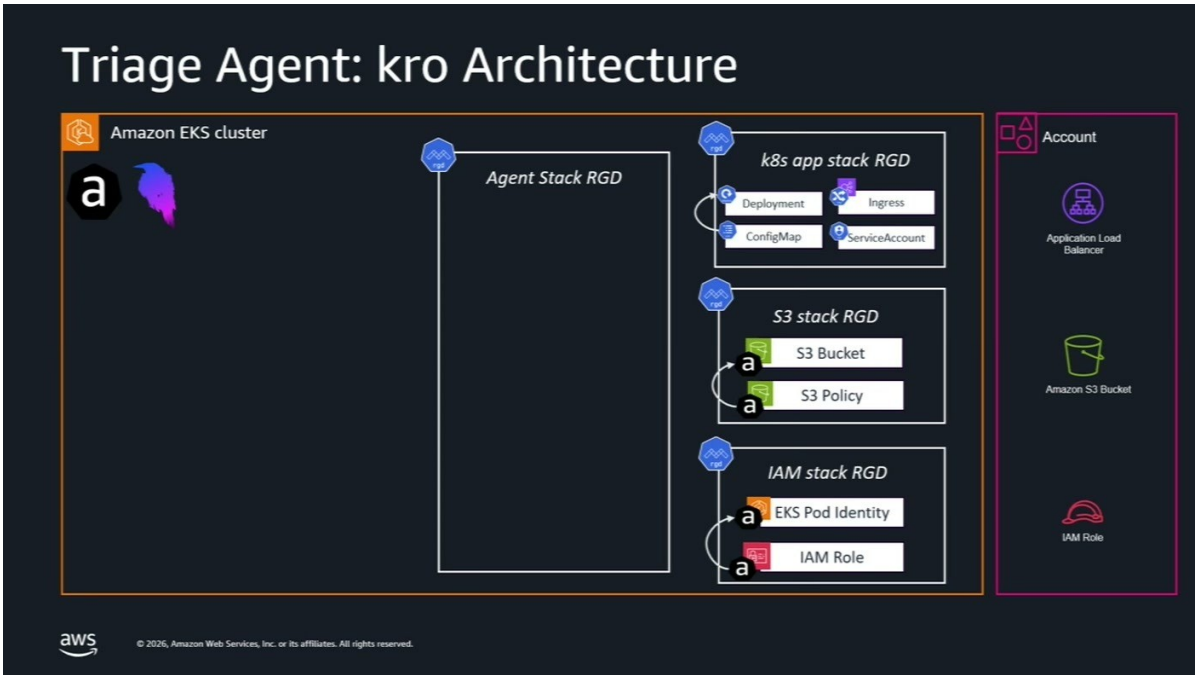
8. הדמו: מערכת אג'נטית ב-14 שורות

כדי להמחיש, הוצג מניפסט של 14 שורות YAML מסוג AgentStack: שם האג'נט, public access מופעל, model id של Anthropic Sonnet 4.5, ופרמטר להזרקת ה-system prompt. 14 שורות, kubectl apply אחד, "והמערכת רצה, נשמע כמו קסם" [22:46].



מאחורי המניפסט: deployment ו-ingress לחשיפת האג'נט, ConfigMap ו-Service Account, ואינטגרציה עם Amazon Bedrock ו-Bedrock AgentCore.

הפירוק שהוצג: בתוך הקלאסטר יושב Deployment שמריץ את האג'נט ברציפות, Service ו-Ingress שחושפים אותו דרך Application Load Balancer, ConfigMap להגדרות סביבה ו-Service Account להרשאות. מחוץ לקלאסטר — Amazon Bedrock לשירותי LLM, ו-Amazon Bedrock AgentCore לתשתית אג'נטית כמו MCP Gateway. השאלה שהוצגה: מה קורה כ"שאנחנו צריכים אג'נט מספר 2 ואג'נט מספר 3" [24:22] — "מדובר פה בלא מעט עבודה שאנחנו צריכים לעשות כל פעם מחדש".

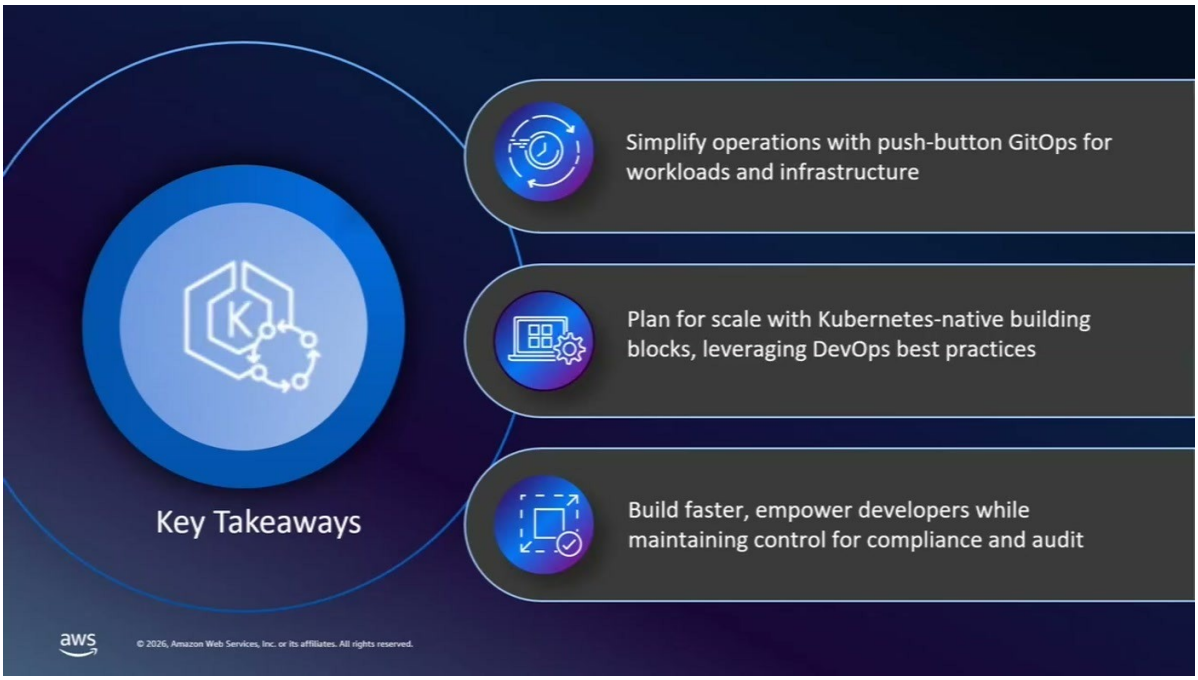


שלושה Kubernetes RGDs — אחסון והרשאות — ומעליהם Agent Stack RGD שמרכיב אותם בהיררכיה.

הבנייה מחדש, שלב אחר שלב: ארבעת משאבי ה-Kubernetes נעטפים ב-Kubernetes App Stack RGD; S3 bucket ו-S3 policy ב-S3 Stack RGD; ורשומת EKS Pod Identity המקושרת ל-IAM Role ב-IAM Stack RGD. הדובר ציין שהחלוקה הזו תואמת חלוקת אחריות ארגונית — אולי צוות ה-security יהיה אחראי על ה-IAM [27:27].

מעל שלושת אלה נוצר Agent Stack RGD, שמייצר אותם היררכית לפי ה-dependencies: ה-Service Account תלוי ב-IAM Role, ו"עד שה-IAM role לא נוצר ולא יספק לו את ה-ARN, הסרוויס אקאונט לא יכול להיווצר" [27:27]. הסגירה: "שמונה resources, שלושה RGDs, agents stack 1, המפתח כותב בסך הכל 14 שורות" [28:57], ו-kro דואג ל-dependencies לפני הפרוביז'נינג ול-life cycle אחריו.

9. מה לוקחים מכאן



שלוש נקודות הסיכום שהדובר ביקש לזכור מהסשן.

- **פישוט התפעול בלחיצת כפתור.** אפשר לפרוס על הקלאסטר Argo CD ל-GitOps ו-ACK שמרחיב את הגישה הדקלרטיבית גם למשאבי ענן, "ככה ששניהם יכולים לחיות יחד באותו ריפרוזיטוריז ולהשתמש באותם "workflows [28:57].
- **להתכונן לסקייל עם בילדינג בלוקס קטנים.** מערכת גדולה שמורכבת מחלקים קטנים מאפשרת להתקדם מהר ולעשות שינויים בצורה בטוחה יותר.
- **להעצים מפתחים בלי לוותר על ממשל.** RGD אחד שמכיל את כל המשאבים הנחוצים, יחד עם policies ו-boundaries שמגדיר הארגון, מספק למפתחים "API אחד פשוט וקל" [30:33].
- **הפער שכדאי לבדוק לפני אימוץ.** הסשן לא הציג תמחור, לא SLA ולא רשימת אזורים או גרסאות נתמכות — נקודות שיידרשו בירור לפני החלטה על הגירה מ-Argo CD self-managed.
- **הלקח מ-Gong תקף גם בלי EKS Capabilities.** דפוס ה-Deploy-Merge ומבנה ה-repo האחיד הם החלטות ארכיטקטורה שאפשר ליישם עם Argo CD רגיל, ללא תלות ביכולת המנוהלת.

מילון מונחים

מונח	פירוש
EKS Capabilities	סט יכולות מנוהלות במלואן שרצות ב-EKS Control Plane ונחשפות ללקוח כ-CRDs בלבד
ACK	AWS Controllers for Kubernetes — יצירה וניהול של משאבי AWS כאובייקטים נייטיביים בקלאסטר, עם reconciliation מתמשך
kro	Kube Resource Orchestrator — הרכבת כמה משאבים ליחידה אחת שנחשפת כ-CRD יחיד
RGD	Resource Graph Definition — ההגדרה ב-kro שמתארת אילו משאבים נכללים ביחידה ומה התלויות ביניהם
ApplicationSet	משאב של Argo CD שמייצר אפליקציות בסקייל באמצעות generators במקום להגדיר כל אחת ידנית
AppProject	משאב של Argo CD לניהול פרויקטים והרשאות — הבסיס למולטי-טננטיות
Singlidation	השם שנתנו ב-Gong לריפו ה-GitOps המאוחד שלהם, המשמש מקור אמת יחיד לכל הקלאסטרים
Deploy-Merge	פריסת השינוי מענף ה-PR לקלאסטרים ובדיקתו שם, ורק אחר כך merge ל-main — היפוך של Merge-Deploy

מונח	פירוש
EKS Pod Identity	מנגנון שמקשר Service Account בקלאסטר ל-IAM Role לצורך הרשאות בחשבון AWS