סוכני ה-AI שלכם לא צריכים לדעת איפה הם רצים

איך Factory בנתה מחדש את ה-agent שלה, Droid, כך שירוץ על כל compute בתוך הענן של הלקוח, ולמה שכבת ה-compute הזו תשוחרר כקוד פתוח

Nikhil Harithas · AI Engineer · 3 בנובמבר 2026
משך: כ-17 דקות · YouTube
סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו הרצאה קצרה (כ-17 דקות) מכנס AI Engineer בניו יורק. השקף הפותח מציין את התאריך 14 באוקטובר 2026, Expo Session ב-McCarthy Room. המרצה הוא Nikhil Harithas, שמוביל את צוות ה-forward deployed engineering ב-Factory, חברה שמפתחת coding agent בשם Droid. ההרצאה מבוססת על "notes from the field" מפריסה אצל Blackstone, מוסד פיננסי גדול עם רגולציה כבדה, שבו אף אחת מברירות המחדל של המוצר לא עבדה.

הטענה המרכזית: agent הוא בסך הכול אפליקציה קצת לא דטרמיניסטית. לכן אסור שימשיך "לשאול" את הזהות של המשתמש, וצריך לנתק אותו מהתשתית שעליה הוא רץ. הפתרון של Factory נקרא Open Compute. זו שכבה שמפרידה compute plane נפרד מה-control plane ומה-data plane, מנהלת את מחזור החיים של מכונות בתוך ה-VPC של הלקוח דרך drivers לכל סוג compute, ומתקשרת עם ה-backend רק בחיבורים יוצאים. החברה מתכננת לשחרר אותה כקוד פתוח.



שקף הפתיחה: Decoupling Agents from Infrastructure, הכותרת המקורית של ההרצאה בשקפים, עם הלוגו של Factory

איך לקרוא את המסמך

- **1. תקציר מנהלים:** הטענות המרכזיות עם חותמות זמן, ומה לא נאמר.
- **2. הלקוח שבו אף ברירת מחדל לא עבדה:** מקרה Blackstone והדרישות שלו.
- **3. זהות:** למה agents לא יכולים להמשיך לרוץ בזהות של המשתמש.
- **4. הארכיטקטורה:** compute gateway, relay, drivers ו-BYOK, שלב אחרי שלב.
- **5. Open Compute:** ה-API, snapshots, ו-forking, והחלקים הקשים בניהול compute.
- **6. סגירה:** agents לא צריכים לדעת איפה הם רצים.
- **7–8.** פעולות מומלצות, שאלות פתוחות ומילון מונחים.

הערת מקור המסמך נוצר אוטומטית מהתמלול ומהשקפים של ההקלטה. ייתכן ששמות ומונחים שובשו בזיהוי הדיבור (למשל, התמלול כותב פעם אחת "Laxone" במקום Blackstone). הציטוטים מועתקים מילה במילה ונבדקו מול חותמת הזמן שלהם. לא נוסף מידע ממקורות חיצוניים.

1. תקציר מנהלים

Factory נתקלה אצל לקוח רגולטורי גדול בשילוב של דרישות: אין להריץ agents על הלפטופ, ה-compute וה-inference יהיו של הלקוח, הזהות תהיה של הלקוח, והכול single tenant. המסקנה שלה היא שבעיית "איפה ה-agent רץ ובשם מי" חוזרת אצל כל לקוח enterprise, ולכן צריך לבנות לה שכבה ייעודית ולא לפתור אותה מחדש בכל פריסה.

- **שתי שאלות חוזרות אצל כל לקוח enterprise:** "who is an agent acting on behalf of, and where does it run?". לדברי המרצה, מוצרים רבים בשנה–שנה וחצי האחרונות פשוט רכבו על הזהות האישית של המשתמש.
- **agent הוא אפליקציה, ואין צורך להמציא את הגלגל מחדש:** אפשר להשתמש במנגנונים מעשורים של הנדסת תוכנה ולהתאים אותם ל-agents.
- **התקנת agent על מכונה לא מספיקה:** מוצר ה-bring your own compute הקודם של Factory השאיר ללקוח את ניהול מחזור החיים של ה-compute, ו-"that's a pretty complex problem".
- **compute plane כשכבה עצמאית:** לצד ה-data plane וה-control plane, Factory מפרידה compute plane שרץ כולו ב-VPC של הלקוח.
- **רק חיבורים יוצאים:** המכונה בצד הלקוח פותחת WebSocket ל-relay של Factory, ואחרי ההקמה אין אף חיבור נכנס. ההסבר: לקוחות חוששים מ-ingress הרבה יותר מאשר מ-egress.
- **BYOK משאיר גם את קריאות ה-LLM בתוך ה-VPC:** ארגונים עם הסכם קיים מול מעבדה יכולים להריץ את הקריאות למשל דרך Bedrock.
- **מנהלים מחשבים ולא sessions:** יחידת העבודה היא מחשב, עם פעולות pause, resume, destroy, snapshot ו-fork. כך השכבה לא קשורה ל-Factory ומתאימה לשחרור כקוד פתוח בתוך "a few weeks".
- **fork הוא תשתית ל-swarms:** שמירת snapshot והעברת תת-משימה ל-agent אחר הופכת לדפוס מרכזי כש-sub-agents מאצילים ל-sub-agents.
- **Droid כותב drivers לבד:** ה-spec של ה-driver משמש גם כבדיקה, אז מספיק לבקש מ-Droid driver שעומד ב-spec.

מה לא נאמר

- **אין מספרים:** כמה סביבות פרוסות, latency של ה-relay, זמני הקמה של מכונה, עלויות. ההצלחה מתוארת רק כ-"pretty significant success".
- **ניהול זהות לא פורט:** ההרצאה פותחת בשאלה "בשם מי ה-agent פועל", אבל לא מסבירה איך Open Compute מנפיקה זהות ל-agent, איך היא מנהלת הרשאות או איך מתבצע ה-audit. היא מתמקדת כמעט רק בשאלה איפה ה-agent רץ.
- **אין תאריך, רישיון או repo לקוד הפתוח:** נאמר רק "in a few weeks".
- **ה-relay עדיין ב-control plane של Factory:** לא נאמר מה עובר דרכו ואם התוכן מוצפן מקצה לקצה. זו נקודה מהותית ללקוח רגולטורי.
- **תקרת שמונה השעות של MicroVMs ב-AWS:** נטענה בלי פירוט של איזה שירות מדובר.

2. הלקוח שבו אף ברירת מחדל לא עבדה



שקף הלקוח: Blackstone, עם שורת משנה קצרה שקשה לקרוא בתמונה הממוזערת

Harithas מתאר את צוות ה-forward deployed engineering כ-"sensory nervous system" של ארגון ההנדסה. זה הצוות שמגלה מה קשה אצל הלקוחות המורכבים ביותר, ומה שהוא מגלה מזין את אסטרטגיית המוצר. המקרה הוא Blackstone, שהוא מתאר כמוסד פיננסי עם טריליון דולר בנכסים, רגולציה ואילוצי אבטחה כבדים, ו-"really, really opinionated about how to deploy non-deterministic software".

הדרישות: לא להריץ agents על הלפטופ, להביא compute, inference וזהות משלהם, ו-single tenant בכל מקום. במוצר עצמו, Droid מקבל משימה דרך Slack, Jira או כל client אחר, ומחזיר PR. לדבריו, זה דורש "a lot of permissions". באמצע הפרויקט Factory התכוונה לעבוד עם אחד מספקי ה-sandbox, אבל שינוי באילוצים מפתח הוביל למעבר ל-EC2. הוא מציין שהתרחיש הזה חוזר אצל לקוחות: אחד רוצה Kubernetes, אחר EC2, אחר Google Cloud.

> **Nikhil Harithas** — And it's really one of those scenarios where none of our defaults worked. It's kind of like they couldn't run agents in their laptop.

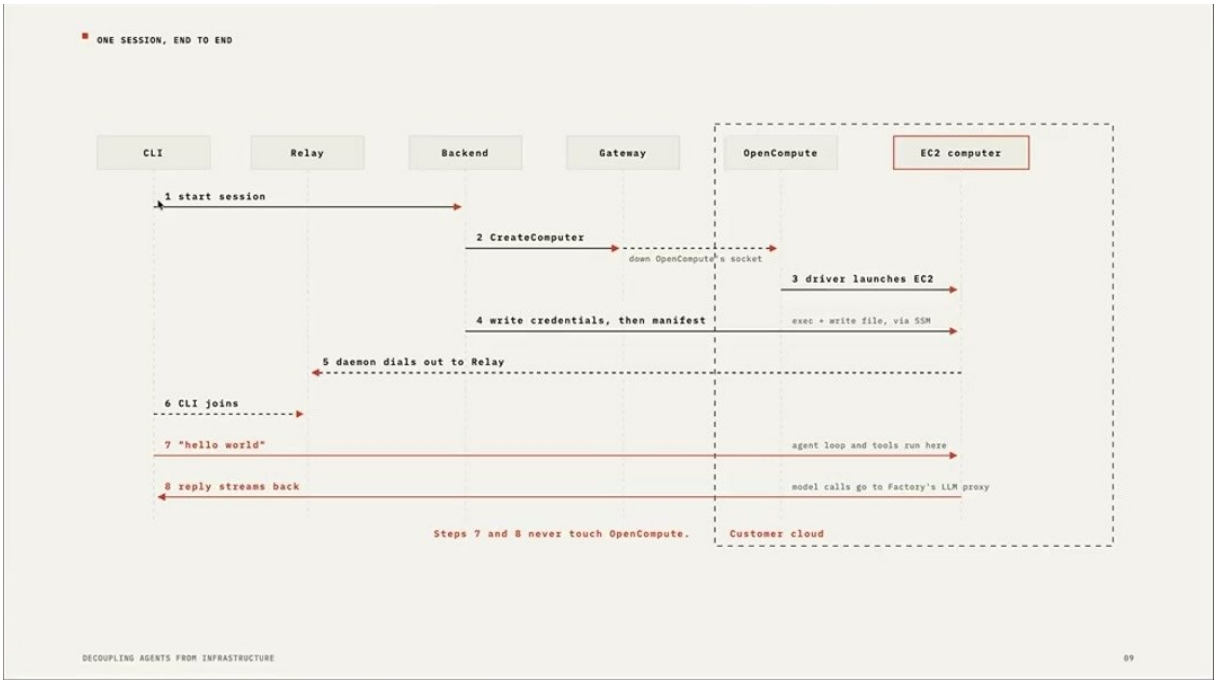
3. סוכנים לא יכולים להמשיך לשאול את הזהות שלך

לפי Harithas, הדפוס הנפוץ, שבו sandbox רץ עם הזהות של המשתמש שהקים אותו ומשתמש בהרשאות שלו, היה פתרון זמני. agent צריך להיות מנוהל כמו כל אפליקציה אחרת: "Agents are basically just slightly non-deterministic applications". המסקנה היא לקחת את המערכות הקיימות ולמקם אותן מחדש, לא להמציא מנגנון חדש.

מכאן עולה גם ההכרה שמוצר ה-bring your own compute הקודם לא הספיק. התקנה של agent על מכונה מעבירה ללקוח את כל ניהול מחזור החיים: הקמה, כיבוי, שחזור, ניתוב. מכאן נולד Open Compute, פרויקט שמיועד לשחרור כקוד פתוח ומפשט את מחזור החיים של ה-compute עבור כל agent, לא רק Droid. ההבטחה ללקוח היא "custom-filled suit", כלומר פריסה לפי אילוצי האבטחה שלו, על ה-compute שבחר, בלי שהנתונים יוצאים מה-VPC. את רוב היתרונות של on-prem מקבלים בלי להעביר אליו גם את ה-control layer.

> **Nikhil Harithas** — But ultimately, we really think that that is, was a kind of temporary idea, because ultimately, agents can't just keep borrowing your identity.

4. הארכיטקטורה: control plane, compute gateway ו-relay



דיאגרמת רצף של פתיחת session. עמודות (קריאה משוערת מהתמונה המוזערת): CLI, Relay, Backend, Gateway, OpenCompute ומכונת EC2 שהוקמה. האזור המקווקו מימין מסמן את מה שרץ ב-VPC של הלקוח

Harithas עובר על הזרימה לאט, צעד אחר צעד–. ה-client יכול להיות ה-CLI של Factory, אפליקציית ה-web, Slack או מוצר של הלקוח עצמו. ב-backend של Factory יש שני רכיבים: compute gateway ו-relay. ה-gateway הוא עצמו שכבת הפשטה, כי מאחוריו יכול להיות Open Compute או ספק sandbox חיצוני.

- **1. בקשה:** ה-client מבקש לפתוח session חדש, והבקשה מגיעה ל-backend.
- **2. compute gateway:** פונה ל-controller של Open Compute ב-VPC של הלקוח, שכבר מחזיק חיבור פתוח ל-control plane, ומבקש "I want to create a new computer".
- **3. driver:** ה-controller פונה ל-driver, עטיפה לסוג compute מסוים: EC2, Kubernetes (או pod ב-EKS) או ספק sandbox כמו Daytona או Modal.
- **4. הקמה:** ה-driver מקים, בדוגמה שבהרצאה, מכונת EC2 חדשה, וה-daemon של ה-agent עולה עליה.
- **5. חיבור יוצא:** המכונה פותחת WebSocket ל-relay שב-control plane של Factory ומשאירה אותו פתוח.
- **6. שיחה:** הודעה מה-CLI, מ-Slack או מה-web עוברת דרך ה-relay ל-sandbox, ושני הצדדים מנהלים דרכו שיחה הלוך ושוב.

> **Nikhil Harithas** — So basically, what we promise in the system is that once set up, we never need to make an inward connection. It's always the customer that's making an outward connection and establishing WebSocket connection.

השכבה האחרונה היא BYOK. ארגונים בשלים, לדבריו, שכבר יש להם הסכם מול אחת המעבדות, מביאים מפתח משלהם, למשל דרך Bedrock, וכך גם קריאות ה-LLM נשארות ב-VPC. התוצאה, כך הוא טוען, היא שליטה בחלק הלא דטרמיניסטי של ה-agent loop, observability מלאה, ו-blast radius של כל פעולה "too spicy for an agent" שנשאר בתוך ה-VPC של הלקוח. ובטווח הארוך: "owning your sovereign" לא יחול רק על המודל אלא על כל ה-stack, כולל evals (בתמלול: "EVAS"), מודל, compute ואירוח המוצר.

5. Open Compute: מחשבים, לא sessions

לפי Harithas, ההפשטות של compute בניהול עוסקות: "Open Compute have nothing to do with factory". יחידת העבודה היא מחשב. הוא מציג את ה-spec שה-compute gateway של Factory קורא בצורה נוחה ל-agent. המכונות מוקמות מתוך templates שהלקוח שולט בהם: מה מותקן, לו, כלומר החוזה שכל מימוש צריך לעמוד בו. את ה-snapshots אפשר לשמור ב-S3 או בבסיס נתונים באיזה SHA (בתמלול: "what shy"), ובאיזה container. לבחירת הלקוח, ובהמשך גם להזין secrets מה-secret store שלו.

פעולה	מה היא נותנת לפי ההרצאה
start / destroy	הקמה והריסה של מכונה מתוך template
pause / resume / suspend	השהיה וחזרה בלי לאבד מצב
snapshot	שמירת המצב המדויק, באחסון שבבעלות הלקוח (S3 או DB)
fork	שכפול snapshot והעברת תת-משימה ל-agent אחר, הבסיס ל-sub-agents ול-swarms

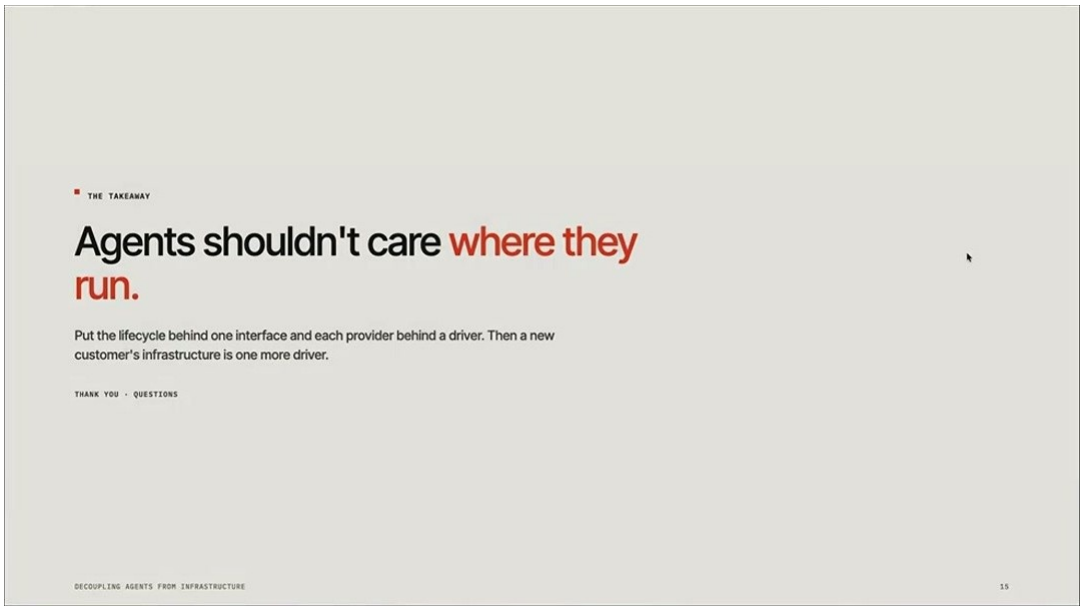
> **Nikhil Harithas** — And I think that an important idea here is that this is a project to manage computers and not sessions.

ההחלטה הזו מכוונת. ברגע שהמושג session, במובן של שיחה ב-Factory, נכנס לשכבה, היא הופכת לספציפית ל-Factory, וללקוח קשה יותר להרגיש בנוח לארח אותה. שכבה שמנהלת רק מחשבים יכולה לשרת גם מוצרים אחרים.

החלקים הקשים

- **bookkeeping:** AWS מאשר שקיבל את הבקשה אבל ה-sandbox לא מגיע, או ש-ack נאבד באמצע ה-session. צריך לעקוב איזה sandbox הוקם, איך קוראים לו, איך לנתב אליו את הבקשות של משתמש מסוים, ומתי לבצע retry כדי שה-client וה-sandbox יסכימו איפה הם בשיחה.
- **תקרת זמן:** "MicroVMs and AWS have an eight-hour session cap", בזמן ש-Missions, מוצר ה-multi-agent orchestration של Factory, רץ ימים. הפתרון הוא snapshot, rehydrate למכונה חדשה והחלפה במקום "as if nothing happened".
- **הפצה לפני קוד פתוח:** בתקופת ה-private preview הם ניהלו גרסאות של חבילות Terraform ושיתפו אותן כדי שלקוחות יוכלו להקים את המערכת.
- **drivers חדשים:** ה-spec משמש גם כבדיקה, בגרסה טובה של test-driven ו-spec-driven development. מבקשים מ-Droid "just make me a driver that fits the spec", ואם הוא עומד ב-spec הוא מוכן לשימוש.

6. סוכנים לא צריכים לדעת איפה הם רצים



שקף הסיכום: Agents shouldn't care where they run, ומתחתיו בערך: כל ה-lifecycle מאחורי ממשק אחד וכל ספק מאחורי driver, כך שהתשתית של לקוח חדש היא עוד driver אחד (הנוסח המדויק לא קריא לגמרי)

הסגירה קצרה: זו הבעיה שחוזרת אצל כולם, וזו הדרך שבה Factory פותרת אותה ונותנת גמישות גבוהה בפריסות ללקוחות ברחבי העולם. לפי השקף, ברגע שמחזור החיים מאחורי ממשק אחד, התמיכה בתשתית של לקוח חדש מסתכמת בכתיבת driver נוסף, ואת זה, כאמור, Droid יכול לעשות בעצמו מול ה-spec.

7. מה לעשות עכשיו

- **בדקו בשם מי ה-agents שלכם רצים:** אם sandbox יורש את ההרשאות של המשתמש שהקים אותו, זה הדפוס ש-Harithas מגדיר כזמני. תנו ל-agent זהות של אפליקציה.
- **הפרידו compute plane:** גם בלי Open Compute, כדאי לבנות ממשק אחד לניהול מחזור החיים (start, pause, snapshot, fork, destroy) ומימוש נפרד לכל ספק.
- **תכננו חיבורים יוצאים בלבד:** WebSocket מה-sandbox ל-relay מקל מאוד על אישור אבטחה בארגון רגולטורי.
- **תכננו מראש ל-snapshot ו-rehydrate:** כל משימה ארוכה תיתקל בתקרת זמן של ה-compute.
- **עקבו אחרי השחרור:** Open Compute אמור לצאת כקוד פתוח "in a few weeks". בדקו רישיון, drivers זמינים ותיעוד ה-spec.

שאלה פתוחה	למה היא חשובה
מה עובר דרך ה-relay ואם הוא מוצפן מקצה לקצה	ה-relay נמצא ב-control plane של Factory, מחוץ ל-VPC של הלקוח
איך מנוהלת הזהות של ה-agent בפועל	ההרצאה פתחה בשאלה הזו אבל לא ענתה עליה
מתי ובאיזה רישיון ישוחרר Open Compute	נאמר רק "in a few weeks"
אילו drivers יהיו זמינים ביום השחרור	הוזכרו EC2, Kubernetes/EKS, Daytona ו-Modal כדוגמאות
איך מטופל כשל של ה-relay באמצע session ארוך	רלוונטי במיוחד ל-Missions שרץ ימים

8. מילון מונחים

מונח	משמעות
Forward deployed engineering	צוות הנדסה שעובד צמוד ללקוחות מורכבים ומזין את אסטרטגיית המוצר
Droid	ה-coding agent של Factory
Missions	מוצר של Factory ל-multi-agent orchestration שרץ ימים

מונח	משמעות
Open Compute	שכבת ניהול compute ל-agents שמיועדת לשחרור כקוד פתוח
Compute plane	שכבה נפרדת מה-control plane ומה-data plane, שמנהלת את המכונות שעליהן ה-agents רצים
Compute gateway	רכיב ב-backend של Factory שמבקש compute, מ-Open Compute או מספק sandbox
Relay	רכיב ב-control plane שמעביר הודעות בין ה-client ל-sandbox דרך WebSocket פתוח
Driver	עטיפה לסוג compute מסוים (EC2, Kubernetes, Daytona, Modal)
BYOK	Bring Your Own Key: הלקוח מביא מפתח משלו לספק המודל, למשל דרך Bedrock
VPC	רשת פרטית וירטואלית של הלקוח בענן
Ingress / Egress	תעבורה נכנסת ותעבורה יוצאת
Snapshot / Fork	שמירת מצב מכונה, ושכפול שלה לטובת תת-משימה
Blast radius	היקף הנזק האפשרי מפעולה שגויה
Single tenant	סביבה ייעודית ללקוח אחד בלבד