DEVOPS TRAINING COURSE

מרצה: אלכס גורבצ'וב

לו"ז שלנו

משך הקורס:

3.02.2025

24.02.2025

ימי לימוד:

יום ב'

9:00-13:00

היקף שעות:

16 שעות

מספר מפגשים:

4 מפגשים

סדר יום

חלק 1 - 9:00-10:15

הפסקה - 10:15-10:25

חלק 2 - 10:25-11:35

הפסקה - 11:35-11:45

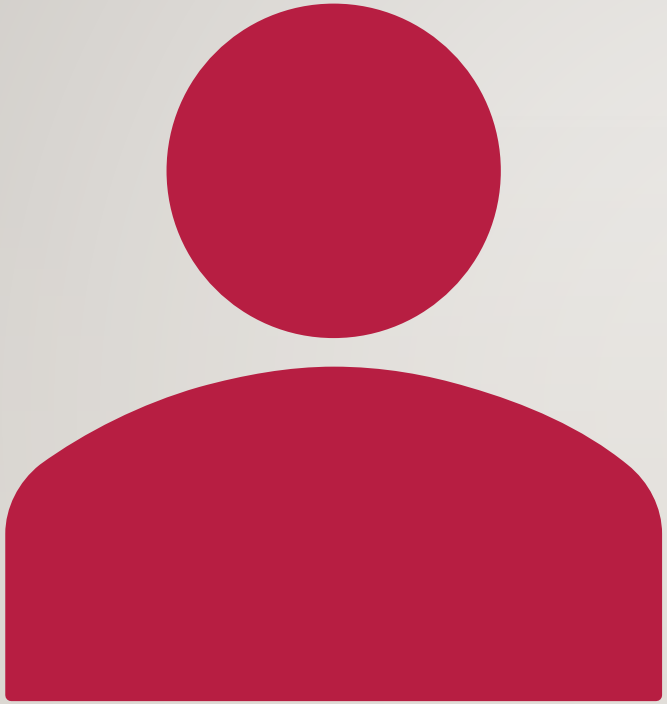
חלק 3 - 11:45-13:00

נושאי השיעור

- Git & GitHub Essentials
- Cloud Native Infrastructure
  - Infrastructure patterns
  - Service Mesh
  - API Gateways
  - Service Discovery

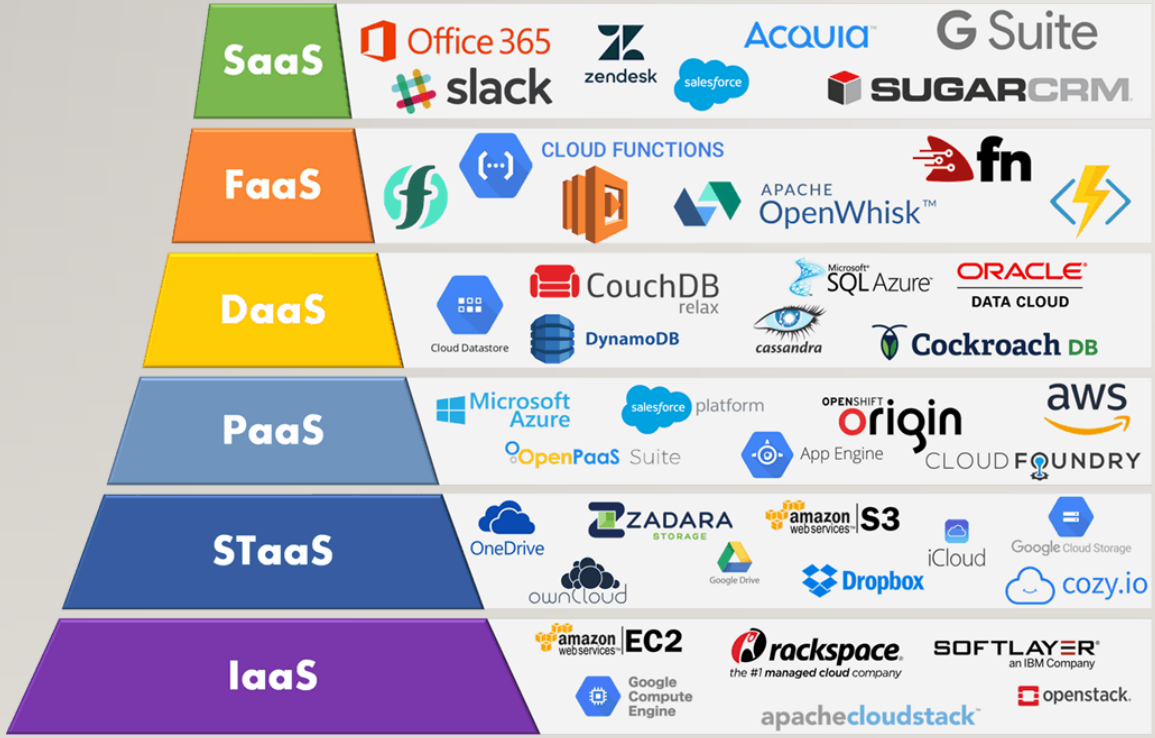
נושאי השיעור

- CI/CD Implementation
  - GitHub Actions
  - Pipeline creation
  - Automated testing
- Deployment Strategies
  - Blue/Green deployments
  - Canary releases



GIT & GITHUB ESSENTIALS

GIT & GITHUB ESSENTIALS



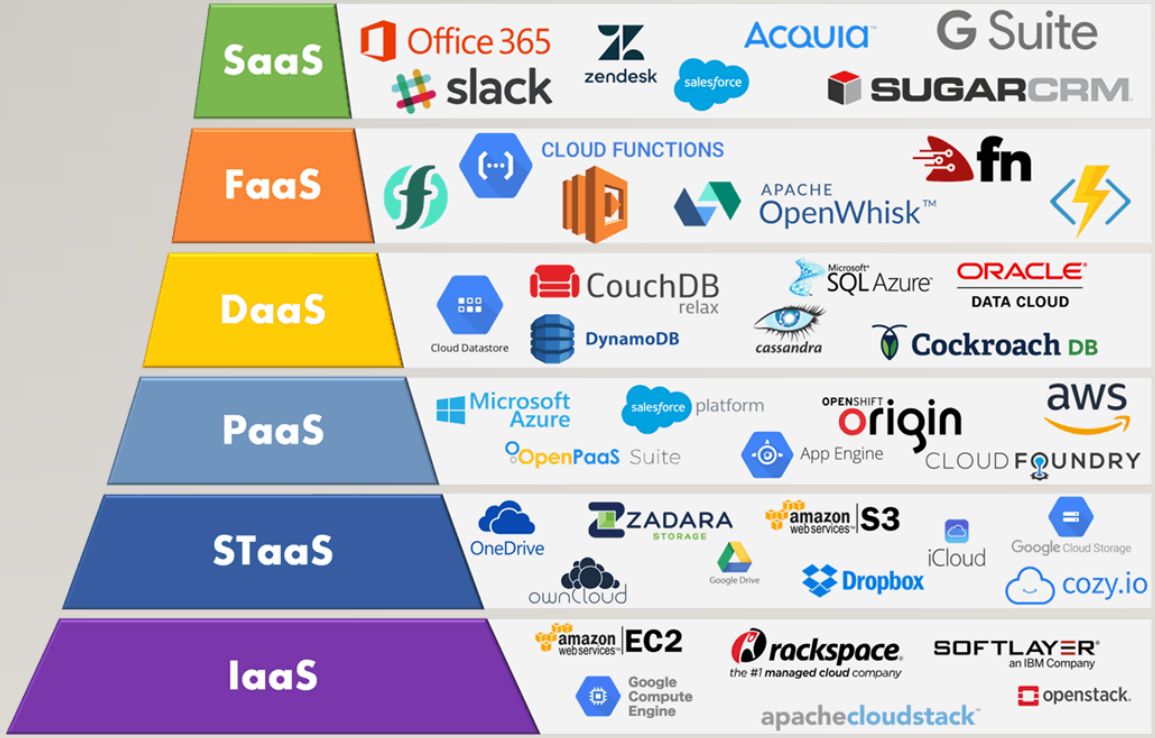
מודלי השירותים בענן

לפני שנעמיק בתשתיות Cloud Native, חשוב להבין את מודלי השירותים בענן:

- **תשתיות כשירות (IaaS - Infrastructure as a Service)** - השירות מספק למשתמשים בו משאבי מחשוב (למשל, נפח אחסון לצורך אחסון וגיבוי קבצים).
- **אחסון כשירות (STaaS - Storage as a Service)** - השירות מספק למשתמשים פתרונות אחסון לכל סוגי הנתונים.
- **פלטפורמה כשירות (PaaS - Platform as a Service)** - השירות מספק למשתמשים בו פלטפורמה המיועדת לפיתוח או להרצת אפליקציות. בפלטפורמה זו הוא יכול להשתמש על מנת לפתח מערכות לשימוש פנימי בארגון או כאלה המיועדות להפעלה בסביבת ענן מחשוב.

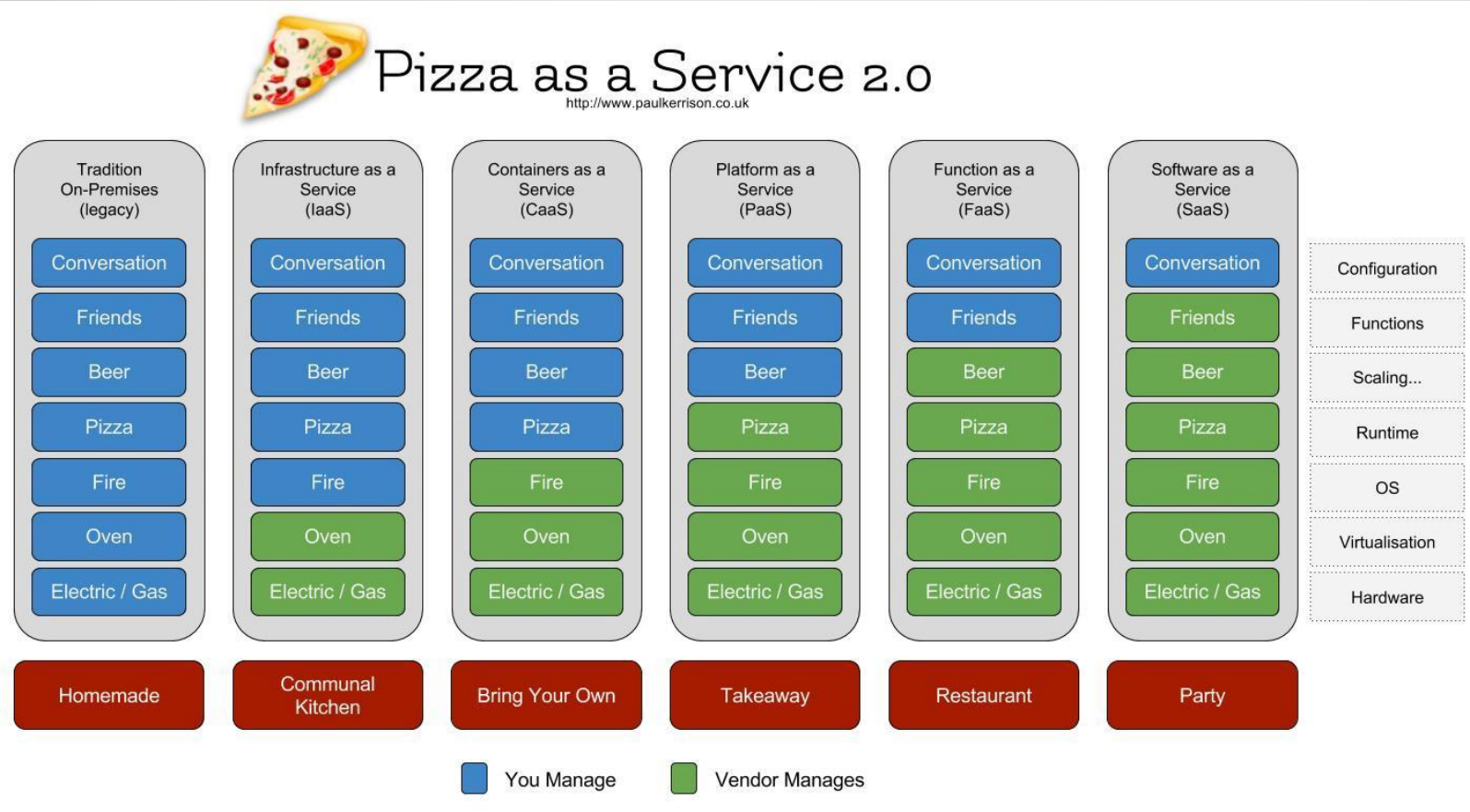
GIT & GITHUB ESSENTIALS

מודלי השירותים בענן



- **שולחן עבודה כשירות (DaaS – Data as a Service)** - מודל שבו ספק חיצוני מספק מידע לפי דרישה. במקום לאחסן ולנהל מידע באופן עצמאי, ארגונים יכולים לרכוש גישה למידע חיצוני לפי הצורך.
- **פונקציה כשירות (FaaS - Function as a Service)** - השירות מאפשר למפתחים לבצע פעולות מבלי לדאוג לשרתים, מכונות וירטואליות או משאבי מחשוב בסיסיים אחרים. המפתח יכול להעלות קוד לענן, ולהריץ אותו מיידית באמצעות כתובת HTTP.
- **תוכנה כשירות (SaaS - Software as a Service)** - השירות מספק למשתמשים בו שירותי שימוש בתוכנה, עדכונים ותחזוקה שוטפת. השירות מעמיד לרשות המשתמש את תשתית החומרה, הפלטפורמה וגם שירותים אפליקטיביים. שירות זה נפוץ במיוחד בתחום מערכות ה-CRM לארגונים.

GIT & GITHUB ESSENTIALS



מודלי השירותים בענן

GIT & GITHUB ESSENTIALS

מבוא ל-Git

ניהול גרסאות הוא הבסיס של פיתוח תוכנה מודרני. בעזרת Git & GitHub אנו שומרים על קוד נקי ומאורגן.

Git מאפשר לנו לעבוד בצוותים, לעקוב אחר שינויים ולהימנע ממצבים בהם קוד טוב הולך לאיבוד.

דוגמה:

Google Docs אפשר לראות גרסאות קודמות ולחזור אליהן במקרה הצורך.

GIT & GITHUB ESSENTIALS

למה צריך ניהול גרסאות (Version Control)?

מערכת ניהול גרסאות פותרת את הבעיות בכך שהיא מספקת דרך מסודרת לשמור ולנהל שינויים:

- **Backup & Restore** - אפשר לחזור לגרסה ישנה.
- **Synchronization** - עבודה משותפת מסונכרנת.
- **Undo** - חזרה אחורה במקרה של טעות.
- **Track Changes** - מעקב אחרי שינויים ומי עשה אותם.
- **Sandbox/Spike** - ניסויים בסביבה נפרדת.
- **Branch/Merge** - עבודה מקבילית על תכונות שונות.
- **לא רק לקוד** – מתאים למסמכים, קבצי תצורה ועוד.

GIT & GITHUB ESSENTIALS

התקנת Git ב-Windows

- **הורדת Git:**
  - יש לגלוש לאתר הרשמי של Git בכתובת: https://git-scm.com/downloads/win
  - בחרו את הגרסה המתאימה למערכת ההפעלה שלכם.
- **הפעלה והתקנה:**
  - הפעילו את קובץ ההתקנה שהורדתם.
  - עקבו אחר ההוראות המוצגות על המסך.
- **בדיקה:**
  - לאחר סיום ההתקנה, פתחו את שורת הפקודה (Command Prompt).
  - הקלידו את הפקודה הבאה ובדקו את הגרסה של Git שהותקנה: git --version

GIT & GITHUB ESSENTIALS

GitHub

GitHub הוא אתר אינטרנט ושירות אחסון מבוסס ענן המאפשר למפתחים לאחסן ולנהל את קוד התוכנה שלהם, לעקוב אחר שינויים שבוצעו בקוד, ולשתף פעולה עם מפתחים אחרים בפרויקטים.

יש ליצור בחשבון ב-GitHub:

- יש לגלוש לאתר https://github.com/
- בדף הבית, לחצו על כפתור "Sign up".
- מלאו את הפרטים הנדרשים, כמו שם משתמש, כתובת דואר אלקטרוני וסיסמה.
- עקבו אחר ההוראות לאימות החשבון.

GIT & GITHUB ESSENTIALS

הגדרת Git ראשונית ב-VS Code

- יש להגדיר את השם והדואר האלקטרוני עבור Git לשימוש בעת ביצוע commit:
  - git config --global user.name "Alex Gorbachov"
  - git config --global user.email agorbach@gmail.com
- יש להקליד git config --list, כדי לוודא ששינויים נקלטו.

GIT & GITHUB ESSENTIALS

עבודה עם Git

- יש להגדיר את השם והדואר האלקטרוני עבור Git לשימוש בעת ביצוע commit:
  - "git config --global user.name "Alex Gorbachov
  - git config --global user.email agorbach@gmail.com
- יש להקליד git config --list, כדי לוודא ששינויים נקלטו.

GIT & GITHUB ESSENTIALS

מושגי יסוד ב-Git

לפני שנתחיל להשתמש ב-Git, בוא להכיר את המונחים המרכזיים:

- Repository (מאגר) - אחסון מרכזי לקוד.
- Commit - שמירת שינוי בקוד.
- Branch - פיתוח במקביל בלי לפגוע בקוד הראשי.
- Merge - שילוב קוד מקבוצות שונות.
- Pull Request - בקשה למיזוג קוד עם GitHub.

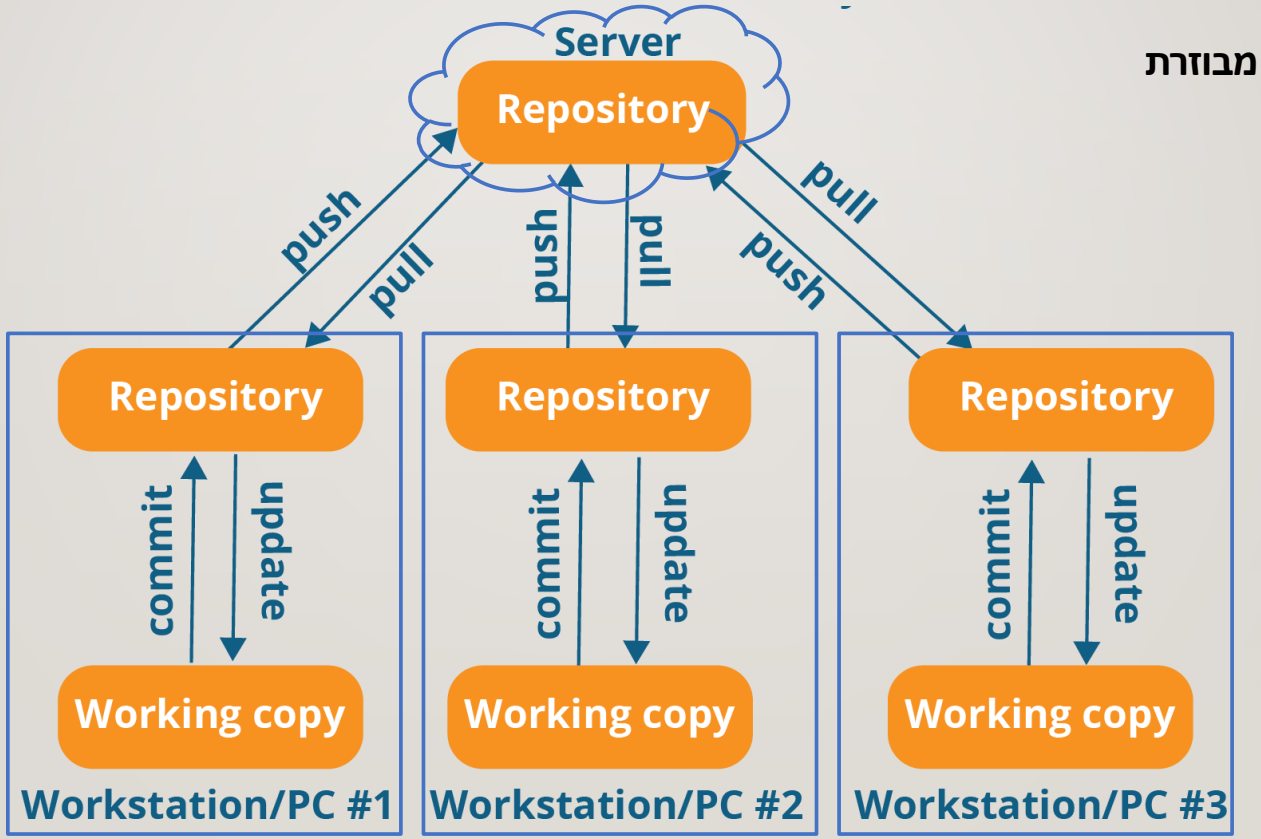
GIT & GITHUB ESSENTIALS

פקודות בסיסיות

command	description
`git clone` ***`url [dir]`***	copy a Git repository so you can add to it
`git add` ***`file`***	adds file contents to the staging area
`git commit`	records a snapshot of the staging area
`git status`	view the status of your files in the working directory and staging area
`git diff`	shows diff of what is staged and what is modified but unstaged
`git help` ***`[command]`***	get help info about a particular command
`git pull`	fetch from a remote repo and try to merge into the current branch
`git push`	push your new branches and data to a remote repository
others: `init, reset, branch, checkout, merge, log, tag`	

GIT & GITHUB ESSENTIALS

מערכת בקרת גרסאות מבוזרת



GIT & GITHUB ESSENTIALS

אסטרטגיות Branching

Branching מאפשרות ניהול יעיל של פיתוח תוכנה על ידי יצירת ענפים נפרדים לעבודה על תכונות חדשות או תיקון באגים.

כל ענף מייצג סביבת עבודה עצמאית, ומאפשר למפתחים לבצע שינויים מבלי להשפיע על הקוד הראשי.

לאחר השלמת העבודה, ניתן למזג את השינויים חזרה לענף הראשי.

שימוש נכון באסטרטגיות Branching תורם לשיפור שיתוף הפעולה בין מפתחים, ניהול גרסאות גמיש ובידוד שינויים.

GIT & GITHUB ESSENTIALS

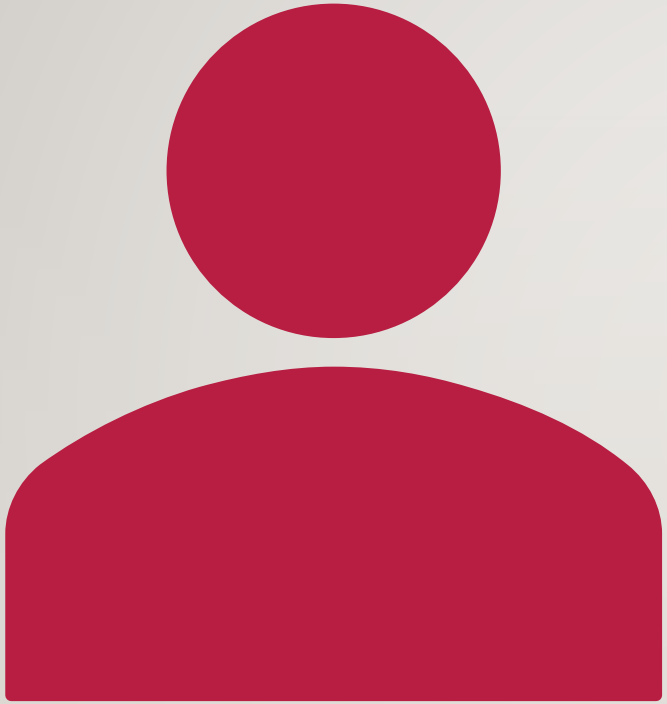
אסטרטגיות Branching

GIT & GITHUB ESSENTIALS

Version Control - Best Practices

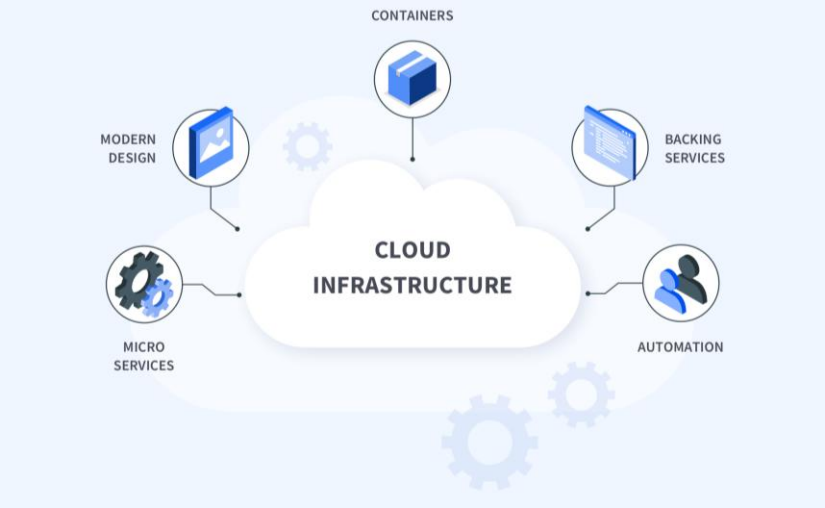
איך להשתמש נכון בניהול גרסאות? הנה כמה עקרונות חשובים:

- השתמשו בהערות טובות – תיאור ברור לכל שינוי.
- בצעו Commit בתדירות גבוהה – עדיף שינויים קטנים על פני גדולים.
- Repo יחיד מול ריבוי פרויקטים – איזון בין הפרדה לאיחוד.
- Branch/Tag במקום המתאים – סניפים לתכונות חדשות, תגיות לגרסאות יציבות.
- הימנעו מקבצים בינאריים גדולים – הם מאטים את המערכת.
- למדו לעבוד עם שורת הפקודה והכלים הגרפיים – שליטה טובה יותר במערכת.



CLOUD NATIVE INFRASTRUCTURE

CLOUD NATIVE INFRASTRUCTURE



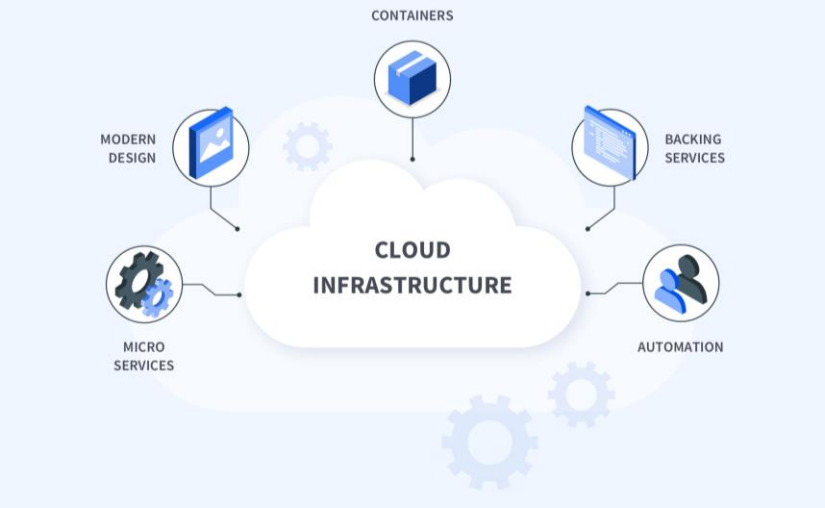
Cloud Native Infrastructure

Cloud Native Infrastructure כוללת:

- תשתית המתוכננת לעבודה בסביבת ענן
- תמיכה באוטומציה מלאה
- סקלאביליות דינמית
- גמישות ועמידות

דוגמאות: Terraform, הגדרת משאבים ב-Kubernetes, Auto-Scaling

CLOUD NATIVE INFRASTRUCTURE

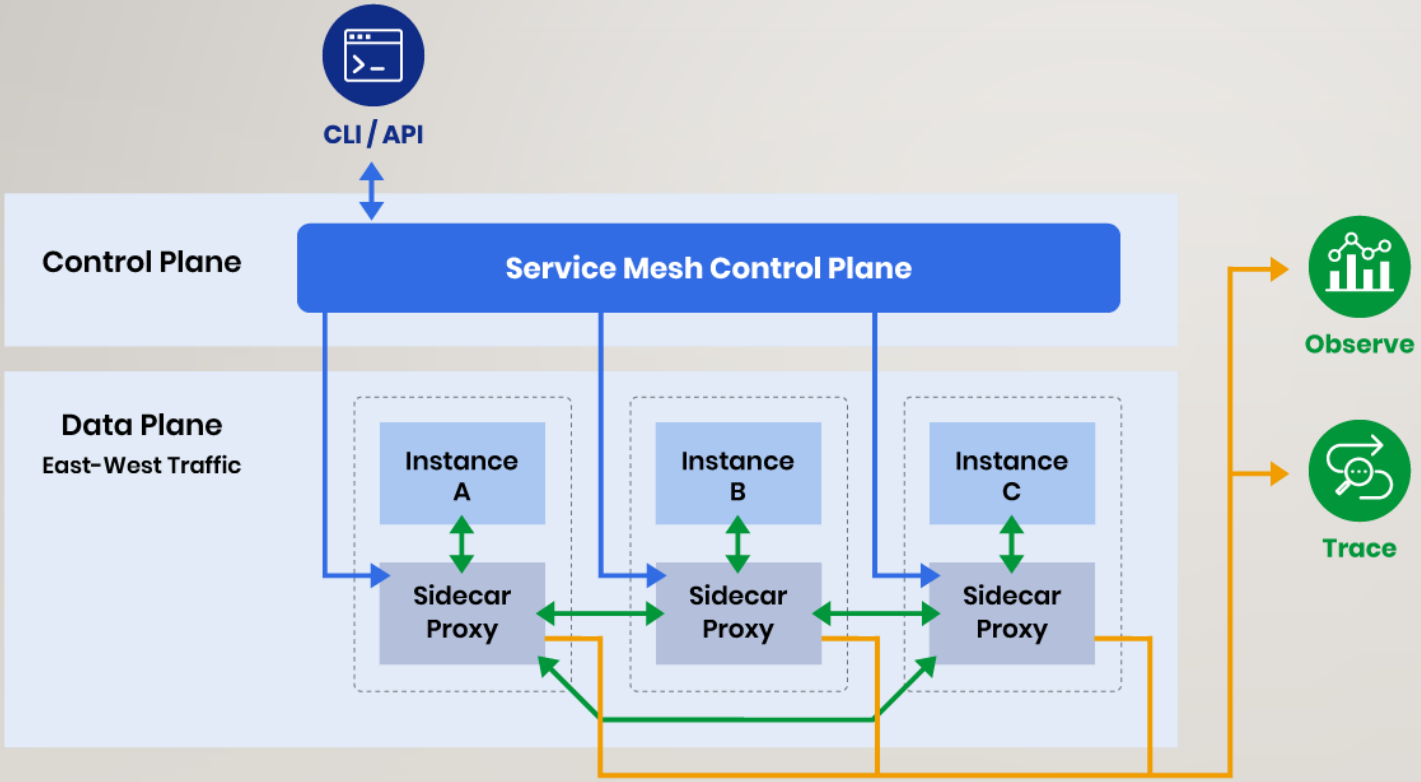


Cloud Native Infrastructure

יתרונות:

- חשיבות האוטומציה
- ניהול תצורה מרכזי
- מדידה וניטור מתמיד

CLOUD NATIVE INFRASTRUCTURE



Service Mesh

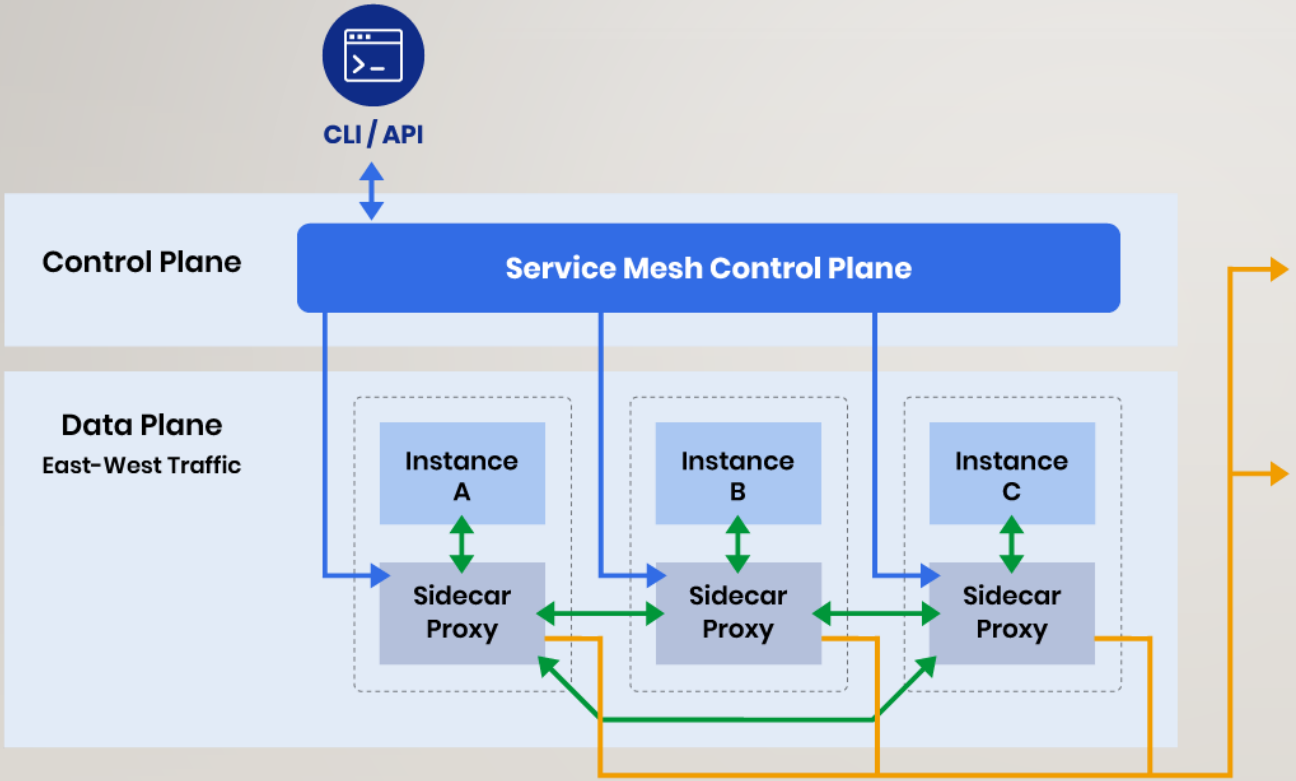
Service Mesh כולל:

- ❑ שכבת תשתית ייעודית לניהול תקשורת
- ❑ ניהול אבטחה ובקרת גישה
- ❑ ניטור וטלמטריה
- ❑ ניהול תעבורה

רכיבים מרכזיים:

- ❑ Control Plane
- ❑ Data Plane
- ❑ Sidecar Proxies
- ❑ API Gateway Integration

CLOUD NATIVE INFRASTRUCTURE

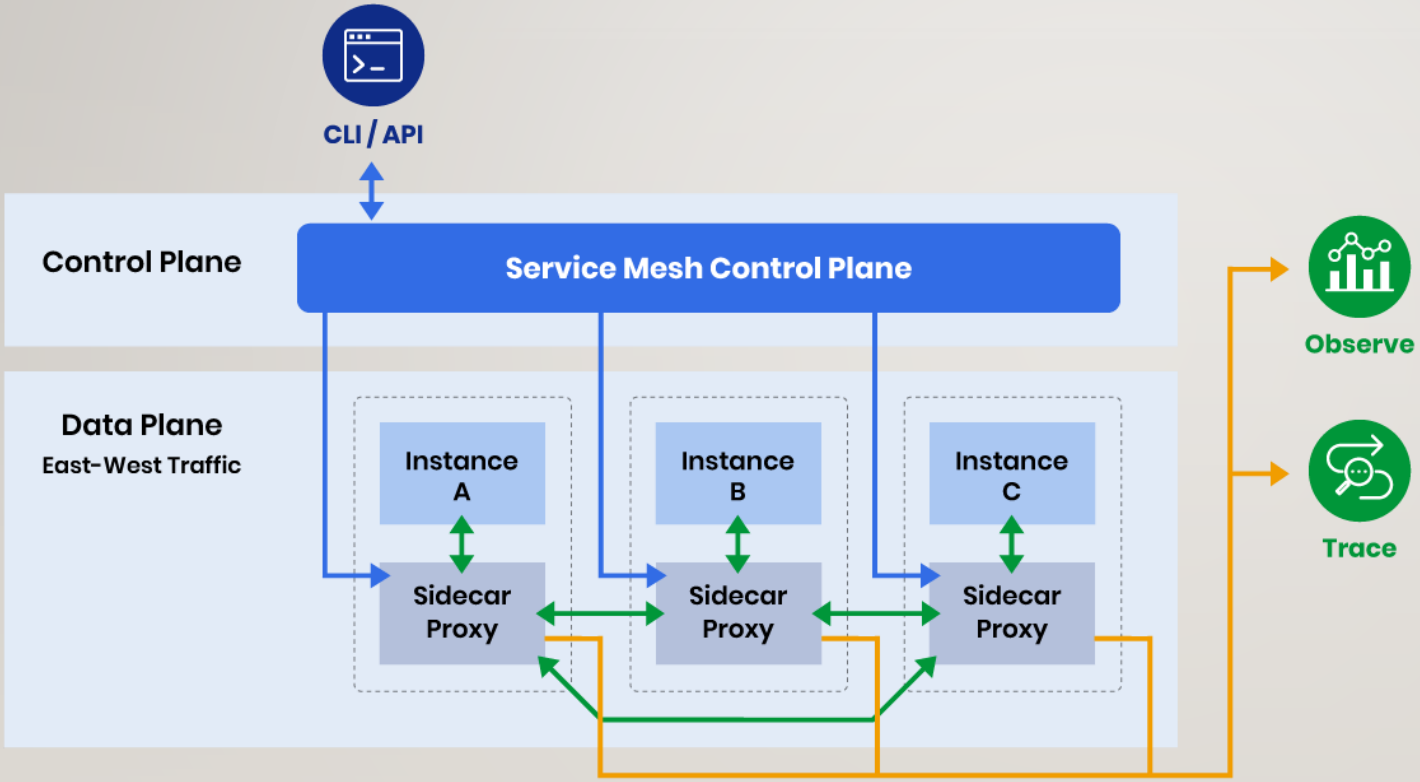


Service Mesh

Control Plane - מישור הבקרה. זהו ה"מוח" של המערכת. הוא מנהל ומגדיר את רשת השירותים (Service Mesh) תחשבו על זה כמגדל פיקוח שמפקח על כל מיקרו-שירות.

Data Plane - מישור הנתונים. כאן מתבצעת הפעולה האמיתית. הוא מורכב מפרוקסים קלים (Sidecars) שנפרסים ליד כל מופע של אפליקציה. הפרוקסים הללו מטפלים בכל התקשורת בין המופעים.

CLOUD NATIVE INFRASTRUCTURE

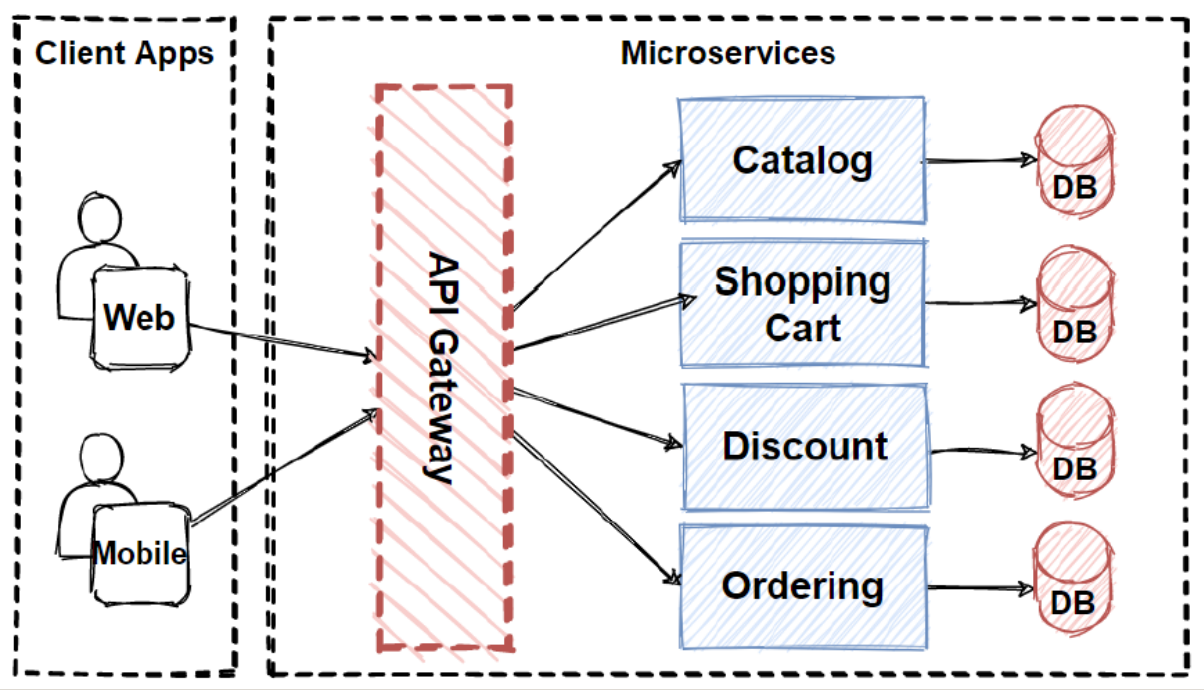


Service Mesh

Sidecar Proxies - פרוקסי קל שנפרס ליד כל מופע. הוא מנתב ומנהל את כל התעבורה הרשתית אל וממופע.

API Gateway Integration - ממשק שורת פקודה. המישור הבקרה ניתן להתממשק עם ממשקי שורת פקודה או APIs, מה שמאפשר אוטומציה ואינטגרציה עם מערכות אחרות.

CLOUD NATIVE INFRASTRUCTURE



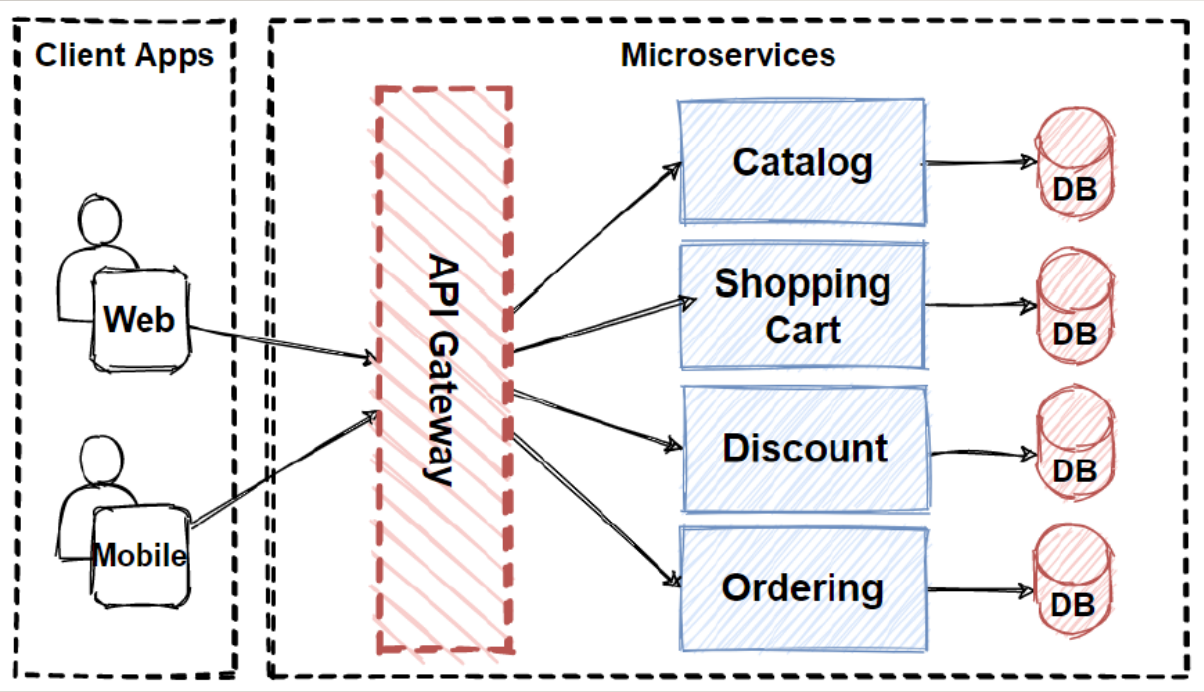
API Gateways

שער API הוא רכיב שיושב מול ממשקי ה-API שך השרות ופועל כנקודת כניסה יחידה לכל בקשות ה-API.

הוא יכול לטפל במשימות כמו אימות, הרשאה, הגבלת קצב ושמירה במטמון.

זה יכול לפשט את ארכיטקטורת ה-API של השירות ולהקל על ניהול ממשקי ה-API.

CLOUD NATIVE INFRASTRUCTURE

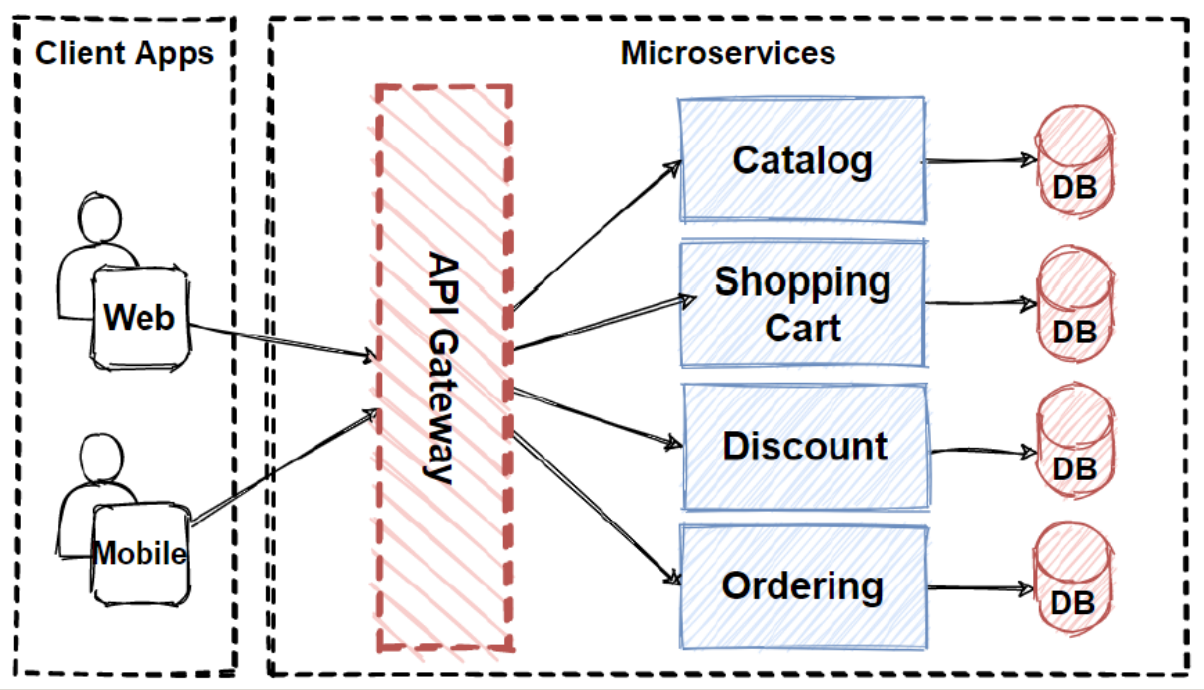


API Gateways

יתרונות:

- אבטחה משופרת: שערי API יכולים לעזור להגן על ממשקי ה-API מפני גישה לא מורשית.
- ביצועים מוגברים: שערי API יכולים לשמור בתשובות API במטמון, מה שיכול לשפר את הביצועים.
- ניהול פשוט: שערי API יכולים להקל על ניהול ממשקי ה-API.
- יכולת הרחבה משופרת: שערי API יכולים לעזור להרחיב את ממשקי ה-API.

CLOUD NATIVE INFRASTRUCTURE

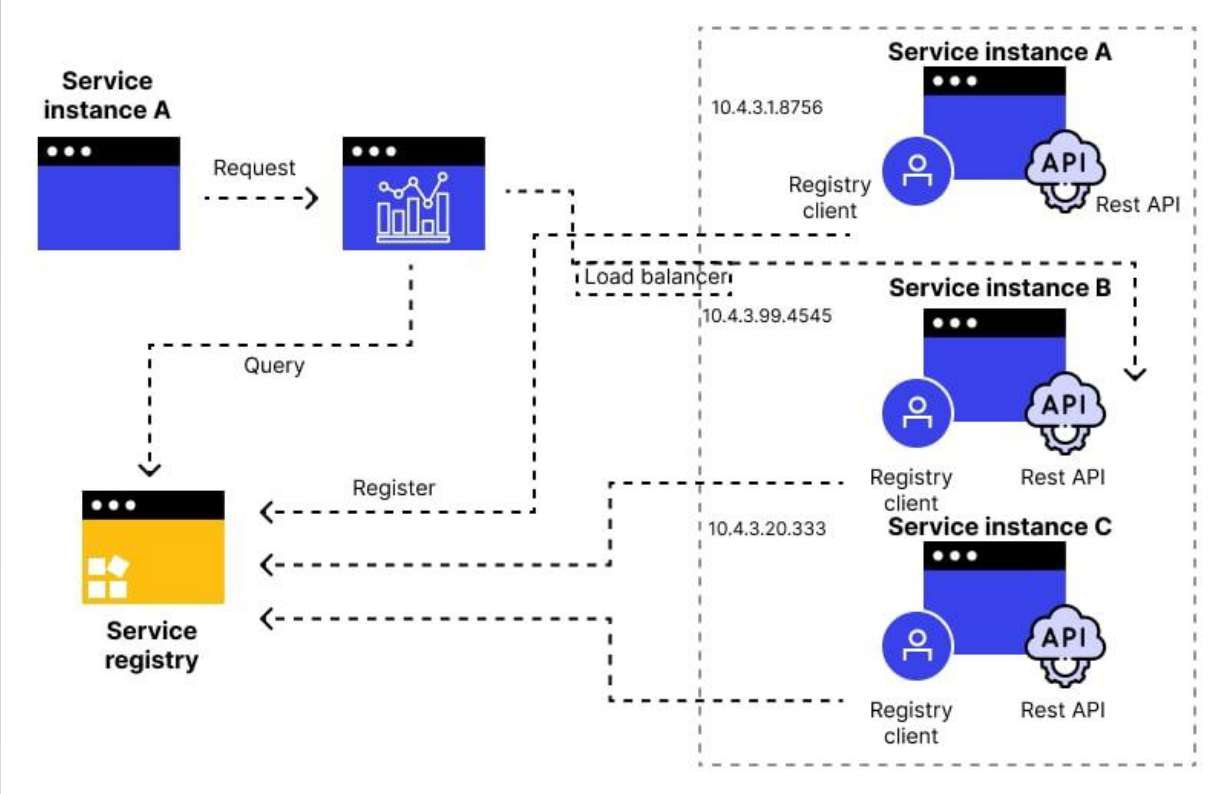


API Gateways

עוד נקודות חשובות:

- שערי API משמשים לעתים קרובות בארכיטקטורות של מיקרו-שירותים.
- ישנם פתרונות רבים ושונים לשערי API זמינים, הן בקוד פתוח והן מסחריים.

CLOUD NATIVE INFRASTRUCTURE



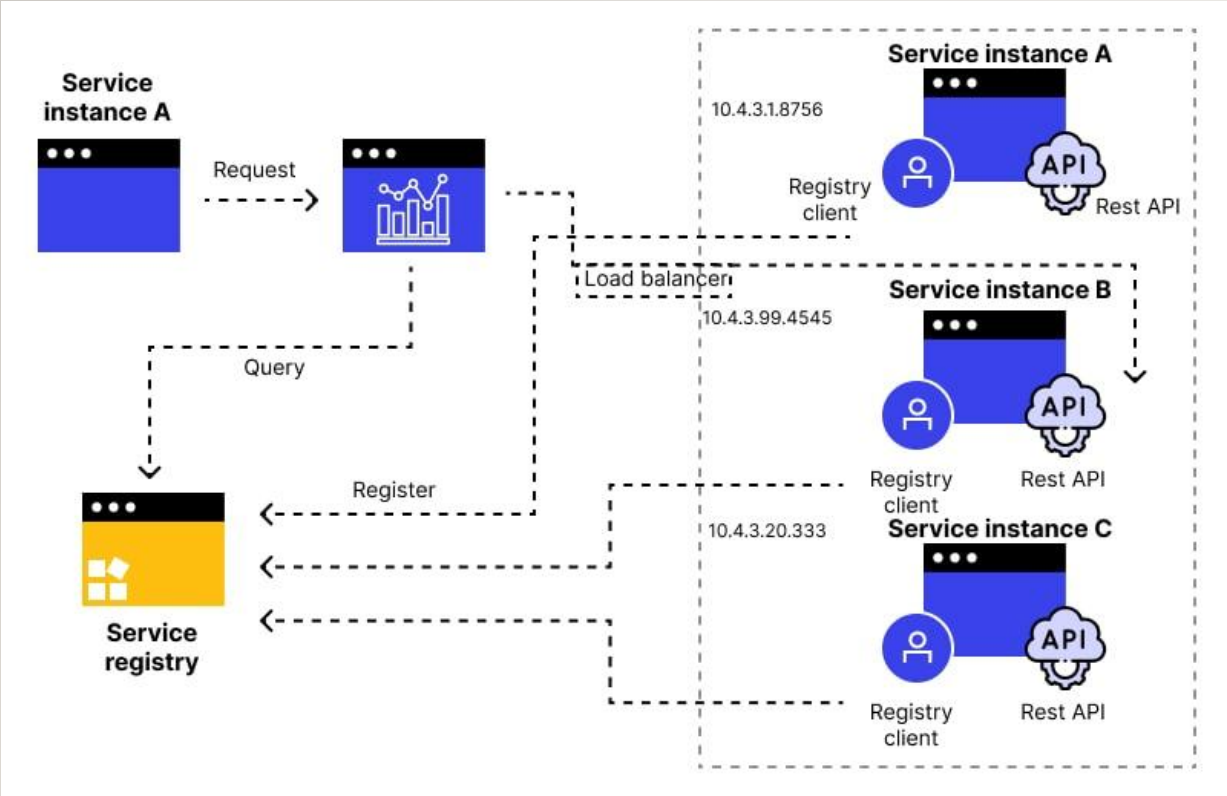
Service Discovery

גילוי שירותים (Service Discovery) הוא מנגנון שמאפשר למערכות שונות למצוא ולתקשר אחת עם השנייה באופן דינמי, ללא צורך בהגדרות קשיחות של כתובות IP ופורטים. גילוי שירותים הוא מרכיב חשוב בארכיטקטורות מבוזרות, והוא מאפשר גמישות ודינמיות בניהול שירותים. שילוב של רישום שירותים ובדיקות תקינות מבטיח זמינות גבוהה של השירותים.

עקרונות Service Discovery:

- Client-Side Discovery
- Server-Side Discovery
- Service Registry
- Health Checking

CLOUD NATIVE INFRASTRUCTURE

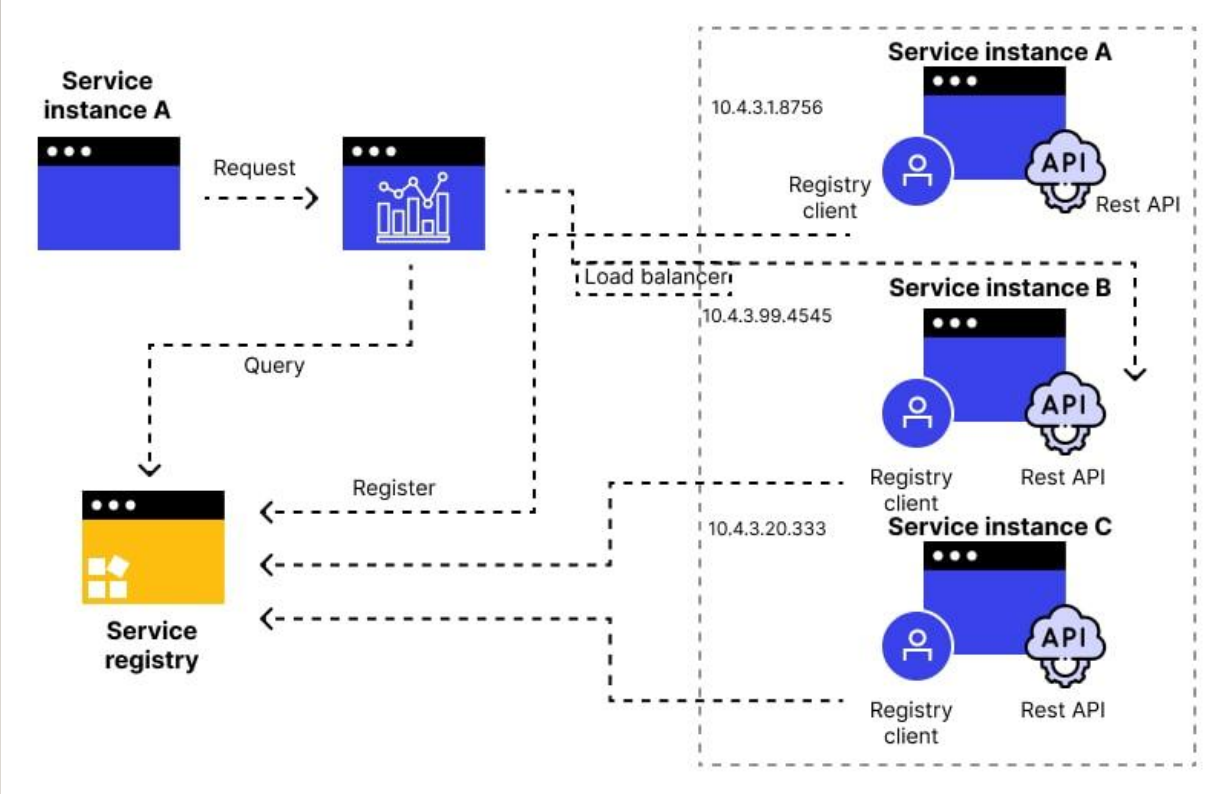


Service Discovery

Client-Side Discovery - הלקוח (למשל, שירות אחר) מחזיק רשימה של כל המופעים הזמינים של השירות שהוא רוצה לצרוך. הלקוח בוחר מופע באופן עצמאי.

Server-Side Discovery - הלקוח פונה לשרת ייעודי (Load Balancer או Proxy) שמודע לכל המופעים הזמינים של השירות. השרת מפנה את הבקשה לאחד המופעים (לפי אלגוריתם).

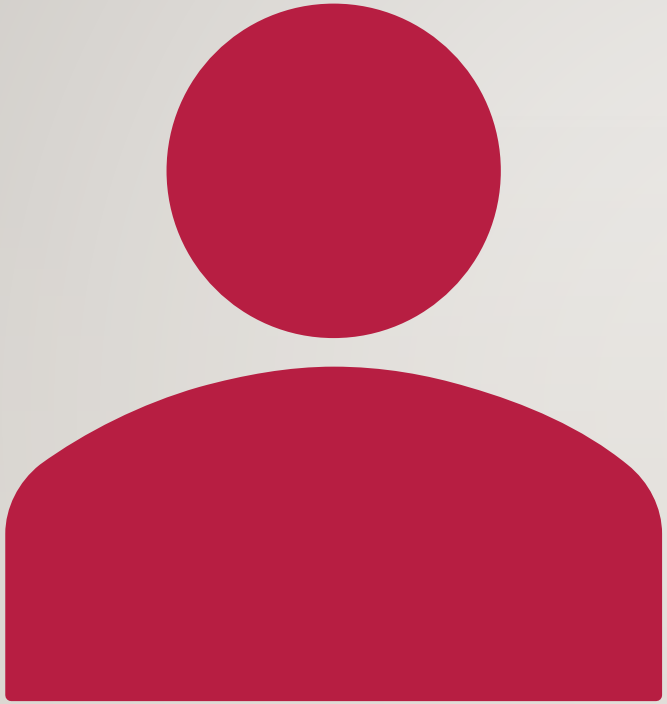
CLOUD NATIVE INFRASTRUCTURE



Service Discovery

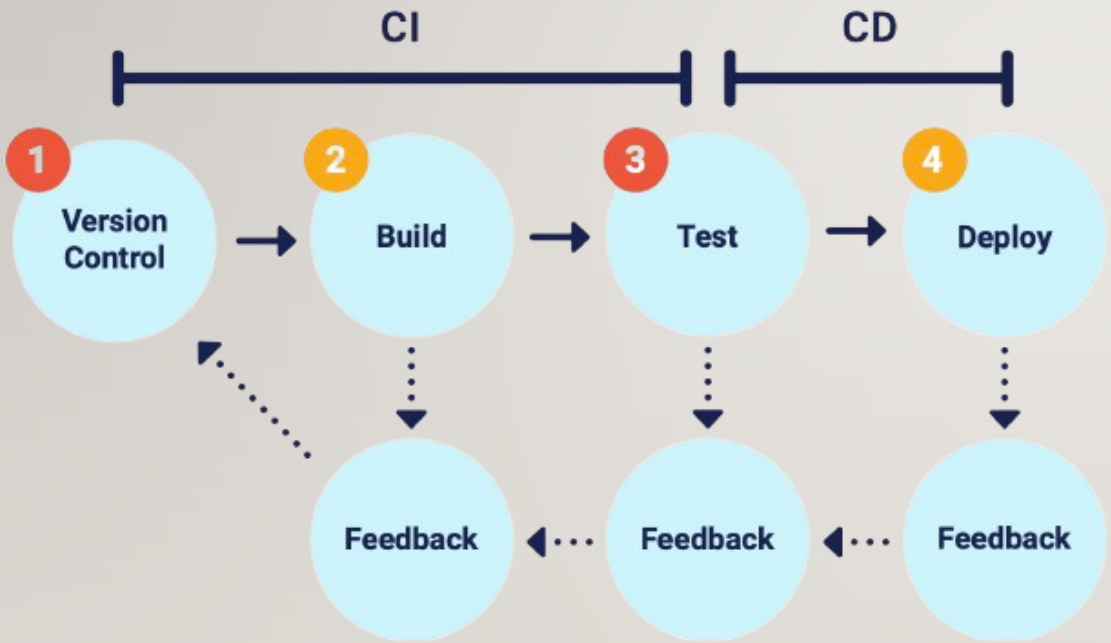
Service Registry - מאגר מרכזי שבו שירותים נרשמים עם המידע שלהם (כתובת, פורט, metadata). שירותים יכולים לרשום את עצמם בעת ההפעלה ולהסיר את עצמם בעת כיבוי.

Health Checking - מנגנון שבודק באופן קבוע את תקינות המופעים של השירות. מופעים לא תקינים מוסרים מה-Service Registry או מה- Load Balancer, כך שלקוחות לא ינסו לפנות אליהם. בדיקות תקינות יכולות להיות פשוטות (למשל, בדיקת חיבוריות) או מורכבות (למשל, בדיקות פונקציונליות).



CI/CD IMPLEMENTATION

CI/CD IMPLEMENTATION



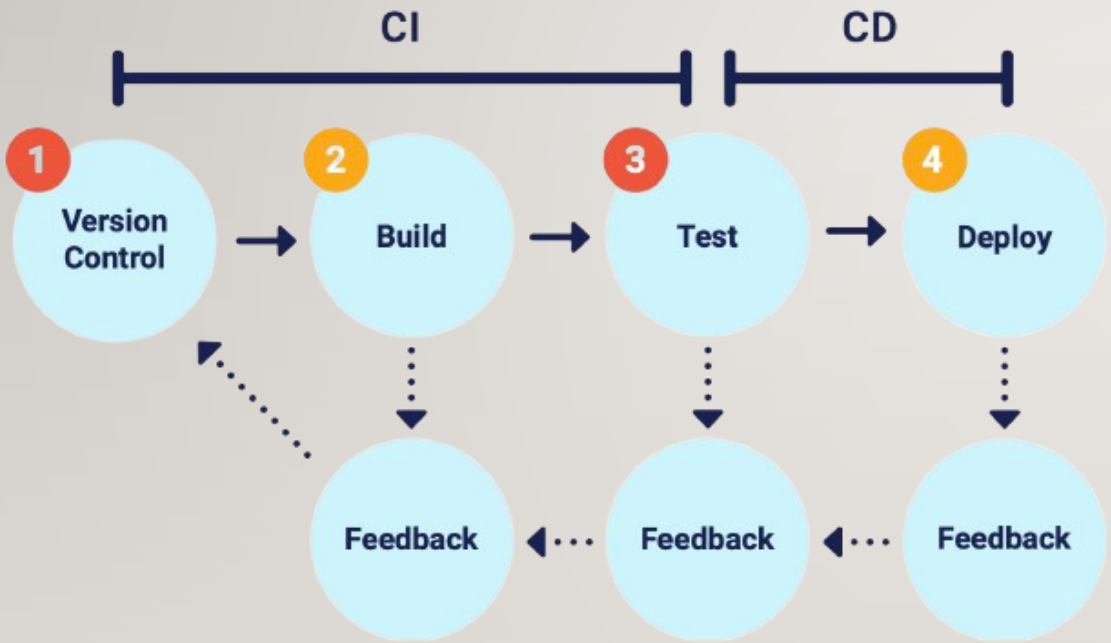
CI/CD מבוא

CI/CD מאפשר לנו לכתוב קוד, לבדוק אותו ולהעלות אותו לפרודקשן אוטומטית.

Continuous Integration (CI) - שילוב שינויים כל הזמן.

Continuous Deployment (CD) - פריסה אוטומטית של גרסאות חדשות.

CI/CD IMPLEMENTATION



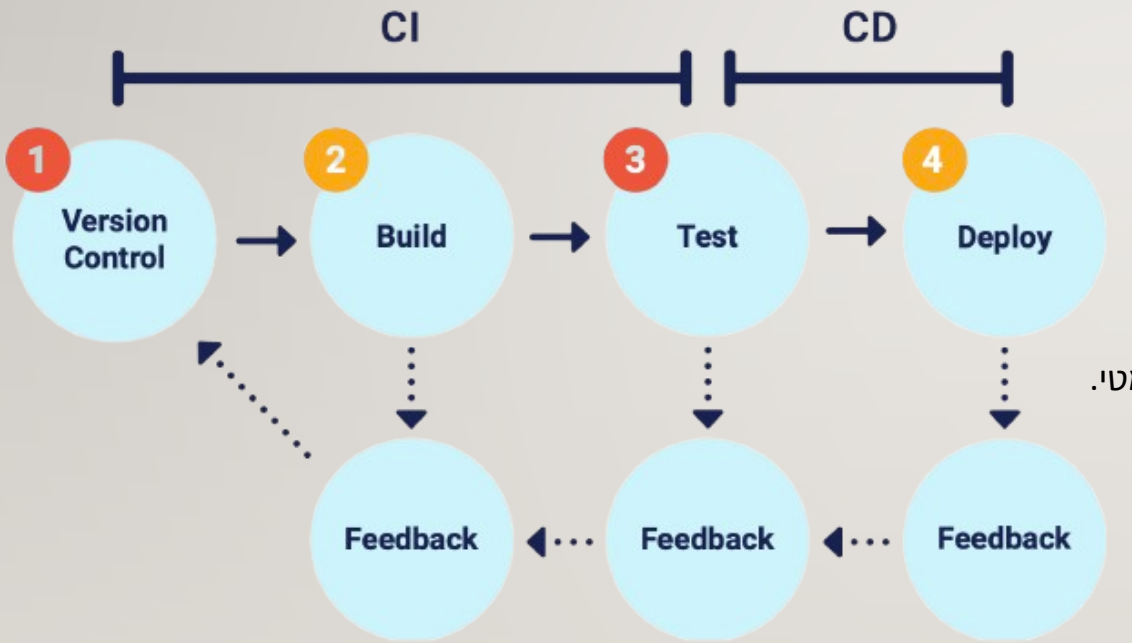
Continuous Integration (CI)

המטרה: למנוע בעיות אינטגרציה על ידי שילוב תכוף של קוד חדש.

תהליך:

- מפתחים כותבים קוד חדש.
- הקוד החדש עובר בדיקות אוטומטיות (יחידה, אינטגרציה).
- הקוד החדש משולב למאגר המרכזי.
- מערכת CI בונה את האפליקציה ובודקת אותה.
- אם הבנייה נכשלת, המפתחים מקבלים התראה וצריכים לתקן את הבעיה.

CI/CD IMPLEMENTATION



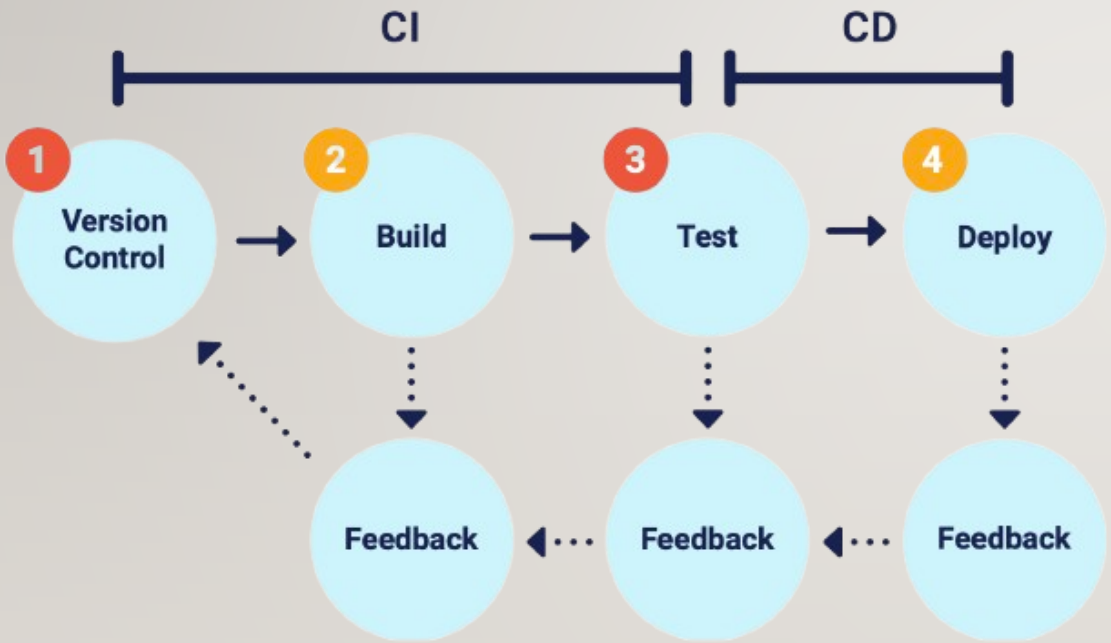
Continuous Delivery (CD)

המטרה: להפוך את תהליך ההפצה של התוכנה לאוטומטי ומהיר יותר.

תהליך:

- לאחר האינטגרציה הרציפה, הקוד עובר תהליך בנייה, בדיקה והפצה אוטומטי.
- התוכנה מוכנה לפריסה בכל רגע נתון.
- ניתן לפרוס את התוכנה לסביבת הייצור בלחיצת כפתור.

CI/CD IMPLEMENTATION



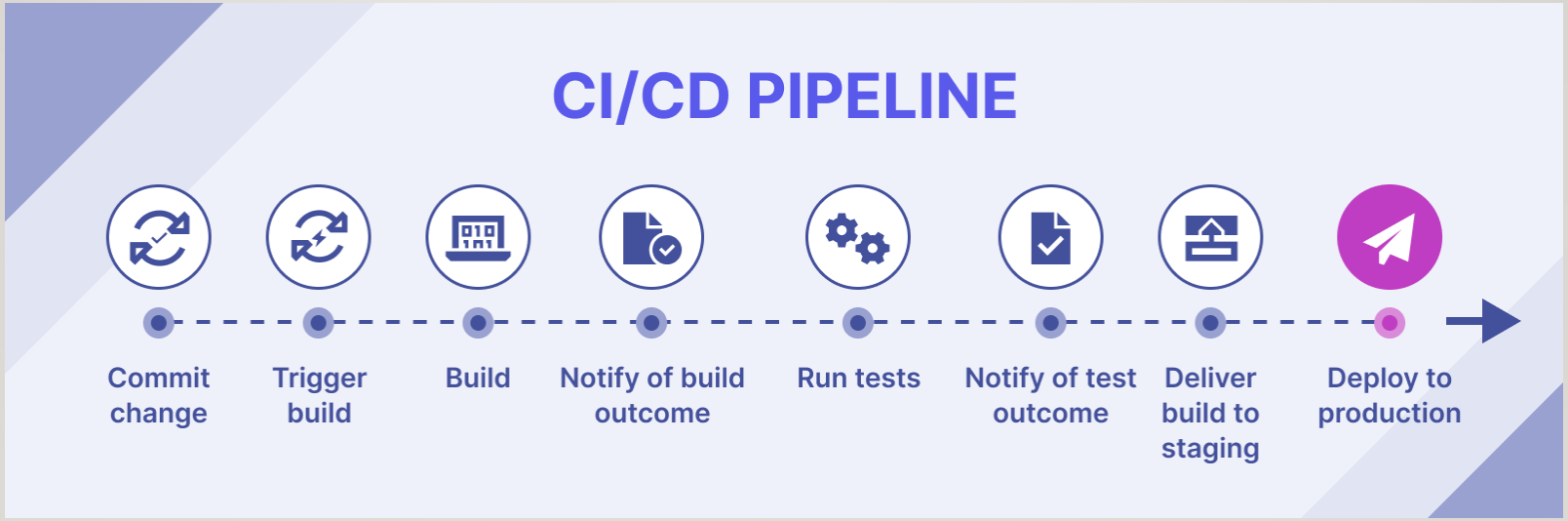
CI/CD Pipeline

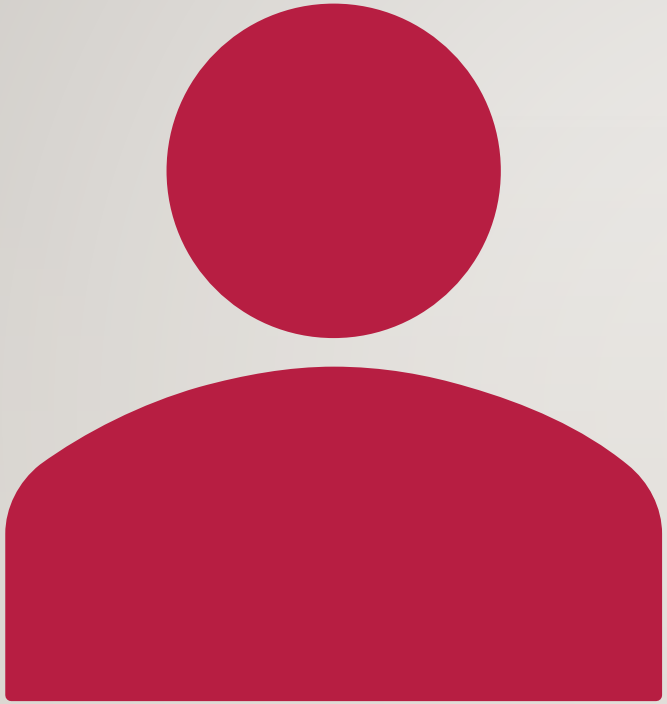
סדרה של צעדים אוטומטיים שנועדו להפוך את תהליך הפיתוח, הבנייה, הבדיקה והפריסה של תוכנה ליעיל, מהיר ואמין יותר.

תחשבו על זה כמו פס ייצור של תוכנה, שבו כל שלב מבוצע באופן אוטומטי ומוביל לשחרור מהיר ואיכותי יותר של התוכנה.

CI/CD IMPLEMENTATION

CI/CD Pipeline





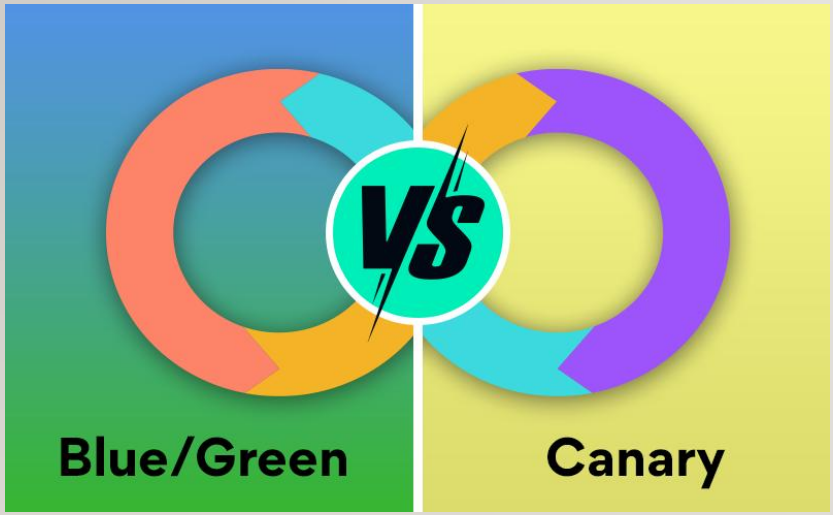
DEPLOYMENT STRATEGIES

DEPLOYMENT STRATEGIES

מבוא לאסטרטגיות פריסה

אסטרטגיות פריסה הן שיטות שונות להפצת גרסה חדשה של תוכנה למשתמשים.

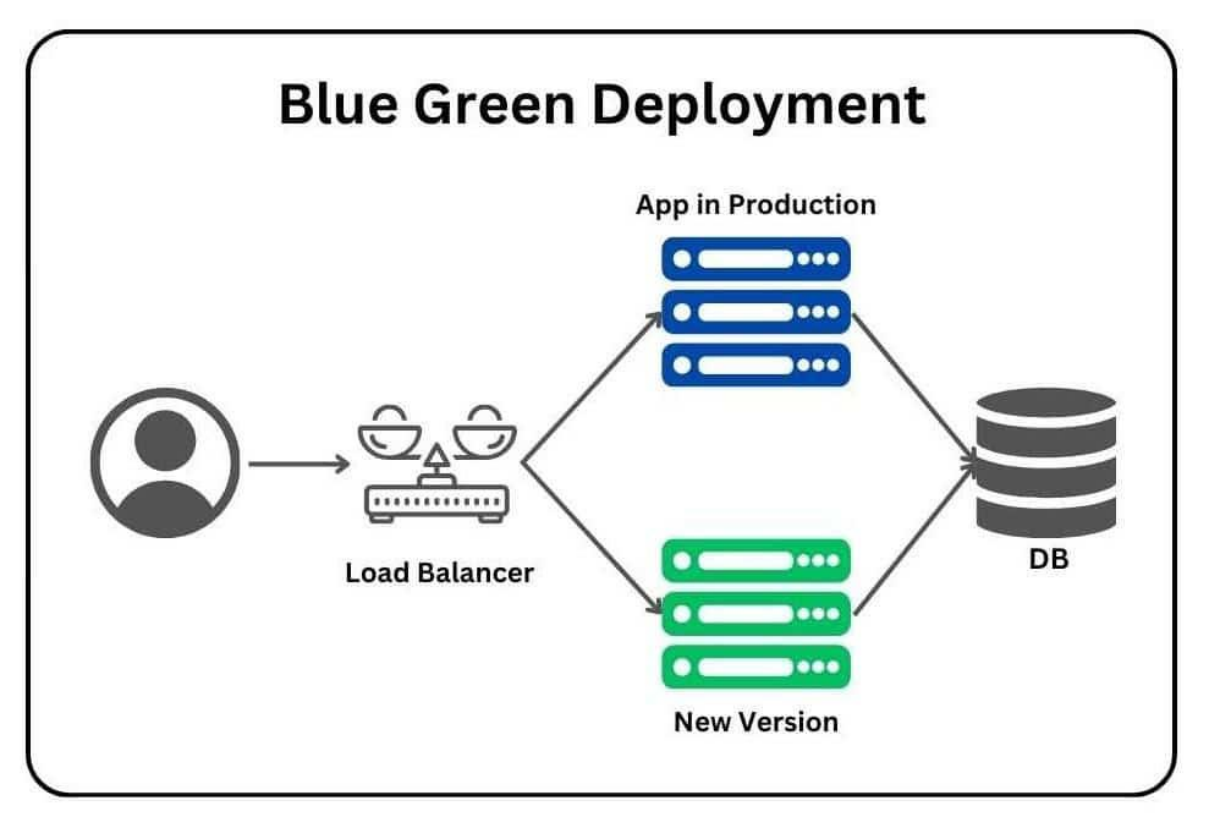
מטרתן היא למזער את זמן ההשבתה, להפחית סיכונים, ולאפשר חזרה מהירה לגרסה קודמת במידת הצורך.



למה זה חשוב?

בחירת אסטרטגיית פריסה נכונה היא קריטית להצלחת שחרור גרסה חדשה. היא משפיעה על חוויית המשתמש, יציבות המערכת, ומהירות התגובה לתקלות.

DEPLOYMENT STRATEGIES



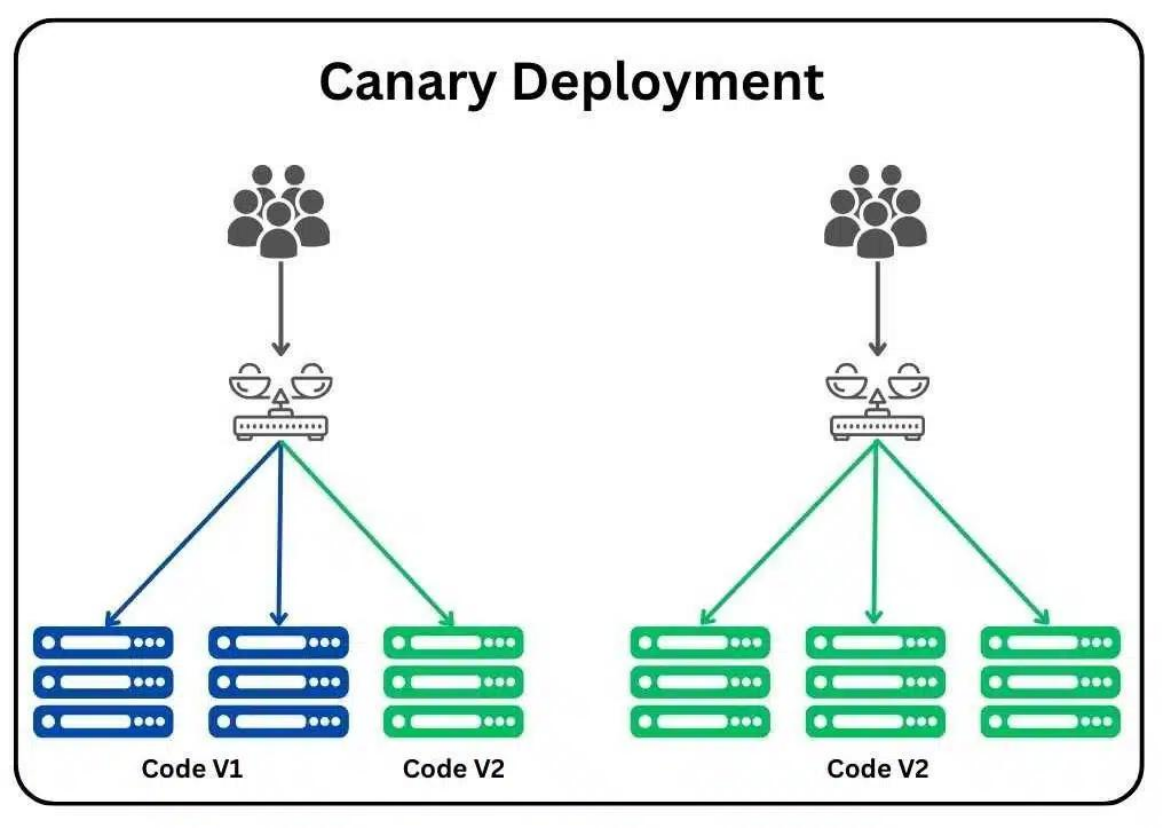
Blue/Green Deployments

- שתי סביבות ייצור: Blue ו-Green.
- אחת פעילה ומשרתת משתמשים, השנייה מעודכנת.
- החלפה בין הסביבות בלחיצת כפתור.

יתרונות: זמן השבתה מינימלי, חזרה מהירה.

חסרונות: דורש כפילות של סביבות.

DEPLOYMENT STRATEGIES



Canary Releases

- פריסה הדרגתית למספר קטן של משתמשים.
- איסוף משוב וניטור ביצועים.
- הרחבת הפריסה בהדרגה לשאר המשתמשים.

יתרונות: זיהוי בעיות מוקדם, מזעור השפעה.

חסרונות: מורכבות בניהול ובמעקב.

שאלות?