Into to DevOps

INT

המרכז הבינלאומי
ללימודי הייטק וחדשנות

מה זה DevOps

INT

המרכז הבינלאומי
ללימודי הייטק וחדשנות

הגדרה DevOps:

- DevOps=**Dev**(Development) + **Ops**(Operations)
- בעיני אנשים שונים ישנו מגוון של הגדרות ל) DevOpsבין היתר (SYS Admins
- הגדרה הבאה **מוויקיפדיה** היא נקודת התחלה טובה:

*"בפיתוח תוכנה, DevOpsהיא תרבות ארגונית ומתודולוגיית עבודה אשר שמה דגש על שיתוף הפעולה והתקשורת בין מפתחי תוכנה לבין שאר אנשי ה- IT שבחברה. השיטה דוגלת בביצוע **אוטומציה** של תהליכי אספקת תוכנה ושל שינויים בתשתיות.*

DevOps הינו...

- DevOps הוא בראש ובראשונה תרבות של שיתוף פעולה בין מפתחים וצוותי האופרציה (צוותי QA, צוותי IT , צוות אבטחה וכו')

- DevOps זה התרבות שהביאה לצמיחה של פרקטיקות העבודה המקובלות בענף

- DevOps הוא התנועה שגרמה לצמיחה של מערך מעשים פרקטי, על ידי **מקצוענים ועבור מקצוענים**.

DevOps אינו!

- DevOps **אינו** פשוט סט של כלים או טכנולוגיות - אלו כלים שחיוניים להצלחה ב-DevOps
- DevOps **אינו** תקן
- DevOps **אינו** מוצר
- DevOps **אינו** תפקיד ספציפי או משרה

היסטוריה של DevOps

INT

המרכז הבינלאומי
ללימודי הייטק וחדשנות

מודל פיתוח Agile

- הגילוי של DevOps צמח מתוך תנועת הפיתוח Agile
- Agile - מתודולוגית פיתוח תוכנה במחזורים קטנים ותדירים (ספרינטים) כדי לספק פונקציות ללקוחות במהירות ולהגיב מהר לשינויים במטרות עסקיות
- Agile ו DevOps נפגשים בדרך כלל יחד

שיטת ניהול Waterfall מול Agile

תהליכי הפיתוח המבוססים על שיטות המפלים (Waterfall) הוכחו כבעייתיים ואינם יעילים עבור פרויקטים תוכנה מורכבים. כמה בולטים המתגלים מתוך החוויות הקודמות הם:

- **רכיב אי הוודאות:** אנחנו לא באמת יודעים בתחילת הדרך את כלל מאפייני המערכת, וכאשר אנחנו מנסים לאפיין באופן חלוט אנו נוטים ליפול לדיסוננס של תכנון יתר שמקדם פיצ'רים שוליים על פני שימושיות ואפקטיביות.
- **מחיר הפיתוח**: פיתוח תוכנה הוא מאמץ יקר מאוד, ואם נפנה את המאמץ הזה בכיוון שגוי, נצטרך להוציא עוד יותר מאמץ ומשאבים כדי לתקן טעויות ובעיות שהתרחשו.
- **קושי תכנון באינטגרציה:** תכנון האינטגרציה בשלב מוקדם מאוד נהיה קשה מאוד, ובכך מגלים שהם מתעכבים (כי הרכיבים נבנו בצורה שמקשה על החיבור שלהם).
- **חוסר הבנה של משתמש צרכי והארגונים:** קשה מאוד להבין את המיקודים לפני שיש לנו נתונים אמיתיים של שימוש במערכת. נתונים אלו מצויים רק כאשר יש מוצר ממשי המגיע למשתמשי הקצה.

המענה של תפיסת הפיתוח Agile

במידה רבה, הבנות אלו הן הביטוי המיידי של הפער בין פיתוח חומרה לפיתוח תוכנה. על בסיס רק מבוכות אלו, החלו להתפתח החל מראשית שנות ה-90' וביתר שאת בעשורים האחרונים שיטות ניהול אחרות וגמישות יותר, שמתבקשות להתמודד עם קריסת מתודולוגיית הפרויקט המסורתית. גישות אלו, שזכו לתיאור הכולל והמוכר " Agile Software Developement", מבקשות להציב מענה ראוי לבעיות האינהרנטיות שבגישת ה-Waterfall.

המענה של תפיסת הפיתוח Agile

- **מיקוד ב-Minimum Viable Product** - המוצר החיוני המינימלי (MVP) מדבר על פיתוח המוצר ברמה המינימלית ההכרחית כדי להשיק אותו. ה-MVP מבקש להגדיר מוצר מצומצם ככל הניתן, שממנו ימשך הפיתוח לשלבים הבאים
- **אינטגרציה מתמשכת (CI: Continuous Integration)** - על רקע הקושי באינטגרציה בין הרכיבים, גישה ה-CI מבקשת למנוע את יצירת הפערים בין המשתתפים, על ידי ניהול שוטף של האינטגרציה על כל שלבי הפיתוח ולא רק עם הסיום.
- **פיתוח בסבבים מהירים (איטרציות או ספרינטים) ועדכוני גרסה תכופים** - בעולם התוכנה התנועה היא מהירה מאוד מגרסה לגרסה, תוך גילום מנגנון למידה מהירה, ותכנון-ביצוע בטווח קצר. באופן זה ניתן לצמצם את המורכבות הרבה של תהליכי הפיתוח הארוכים וכן להביא את מערכות הפיתוח להרבה יותר. את הפיתוח מבלוק לבלוק מחליף פיתוח מבוסס איטרציות קצרות.
- **צוותי פיתוח אחודים** - בעוד שהפיתוח בגישת WATERFALL התבסס לרוב על צוותים נפרדים ומבודלים שכל אחד מהם אחראי על אספקט אחר של המוצר (קשרי לקוח/ניתוח מערכות, פיתוח) או רכיבים נפרדים, פרוייקטים אג'יליים מתבססים על צוותים אחודים המקדמים את המוצר או נדבך שלם במוצר בכל איטרציה.
- **פידבק ישיר משתמשים** - לנוכח הקשיים הקוגניטיביים לצפות מראש את התנהגות המשתמשים, הגישה מקדמת שיח ישיר ומתמשך עם הלקוח, על גבי המערכת הנבנית, לצורך הבנת הצורך האמיתי. יתרה מכך, גישה זו נובעת גם מכך שקשה יותר ויותר לצפות את הצורך האמיתי של המשתמש, או את התאימות של הפתרון לשוק ככל שמדובר במוצר היוצר שיבוש שוק.

DevOps היום

תנועת DevOps לא הפסיקה לצמוח מאז 2009, והיא כבר לא תנועה או נישה קטנה.

מושג ה-DevOps מאז 2009:

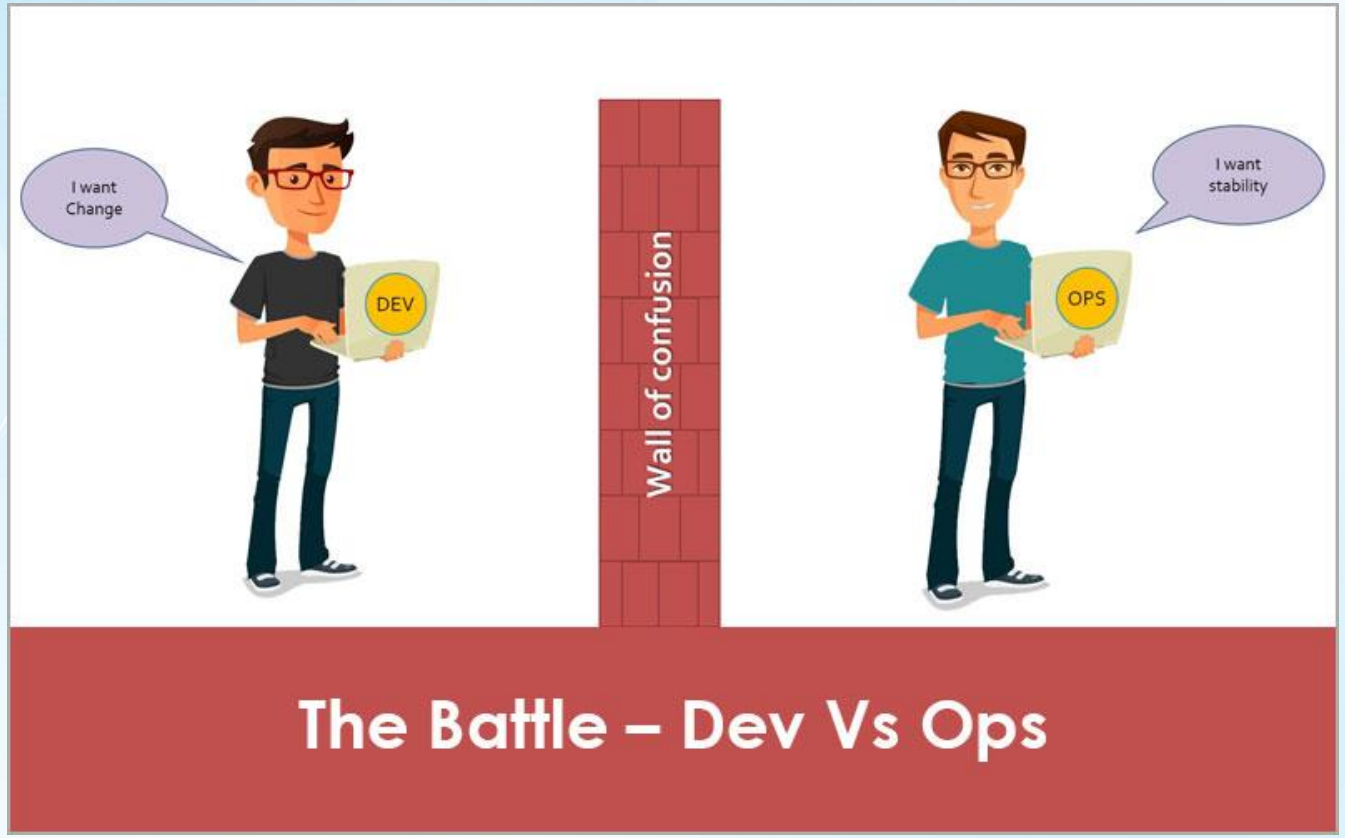
- הפך למיינסטרים.
- הוליד מגוון גדול של כלים.
- שינה לחלוטין את תעשיית ה- IT "לנצח" (למשל בתחום של אוטומציה של התהליכים).

מטרות ויעדים של DevOps

תרבות DevOps

תרבות DevOps עוסקת בשיתוף פעולה בין Dev ל-.Ops

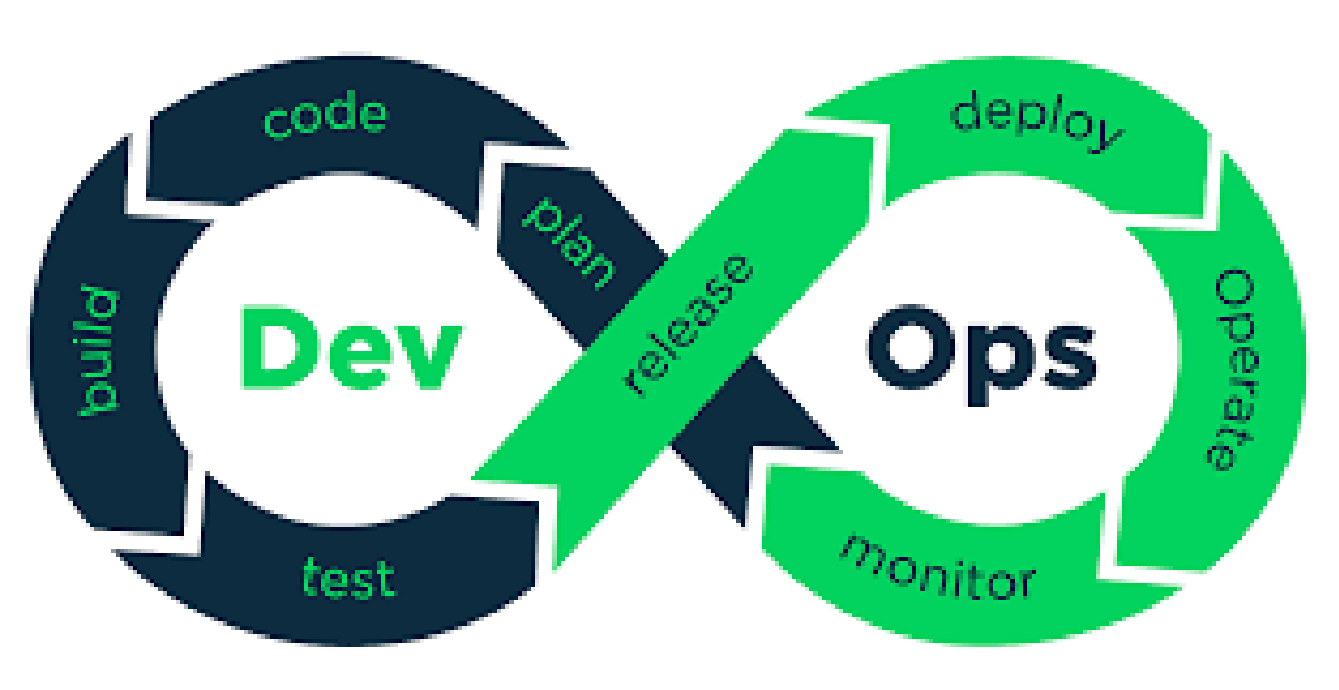
לפי ההפרדה המסורתית ל- Dev ול- Ops יש מטרות **שונות ומנוגדות**.



תרבות DevOps

על פי תרבות ה-DevOps:

Dev-ו-Ops עובדים יחד וחולקים את אותן מטרות.



מטרות תרבות DevOps

כאמור ,בתוך תרבות ה DevOps-ישנה עבודה משותפת של צוותי Dev ו-Ops .

בתרבות ה ,DevOps-למפתחים **אכפת מיציבות** בנוסף למהירות, ולאנשי האופרציה **אכפת ממהירות** בנוסף ליציבות.

התפקידים המסורתיים של מפתחים ומהנדסים תפעוליים יכולים אפילו להיטשטש תחת) DevOpsכמו למשל מנהלי רשתות.(

במקום "**לזרוק קוד על הקיר**", מפתחים ואנשי האופרציות עובדים יחד כדי ליצור ולהשתמש בכלים ותהליכים התומכים הן במהירות והן ביציבות.

ההנחה הרווחת היא ש- Dev ו- Ops חזקים יותר כשהם ביחד!

מטרות תרבות DevOps

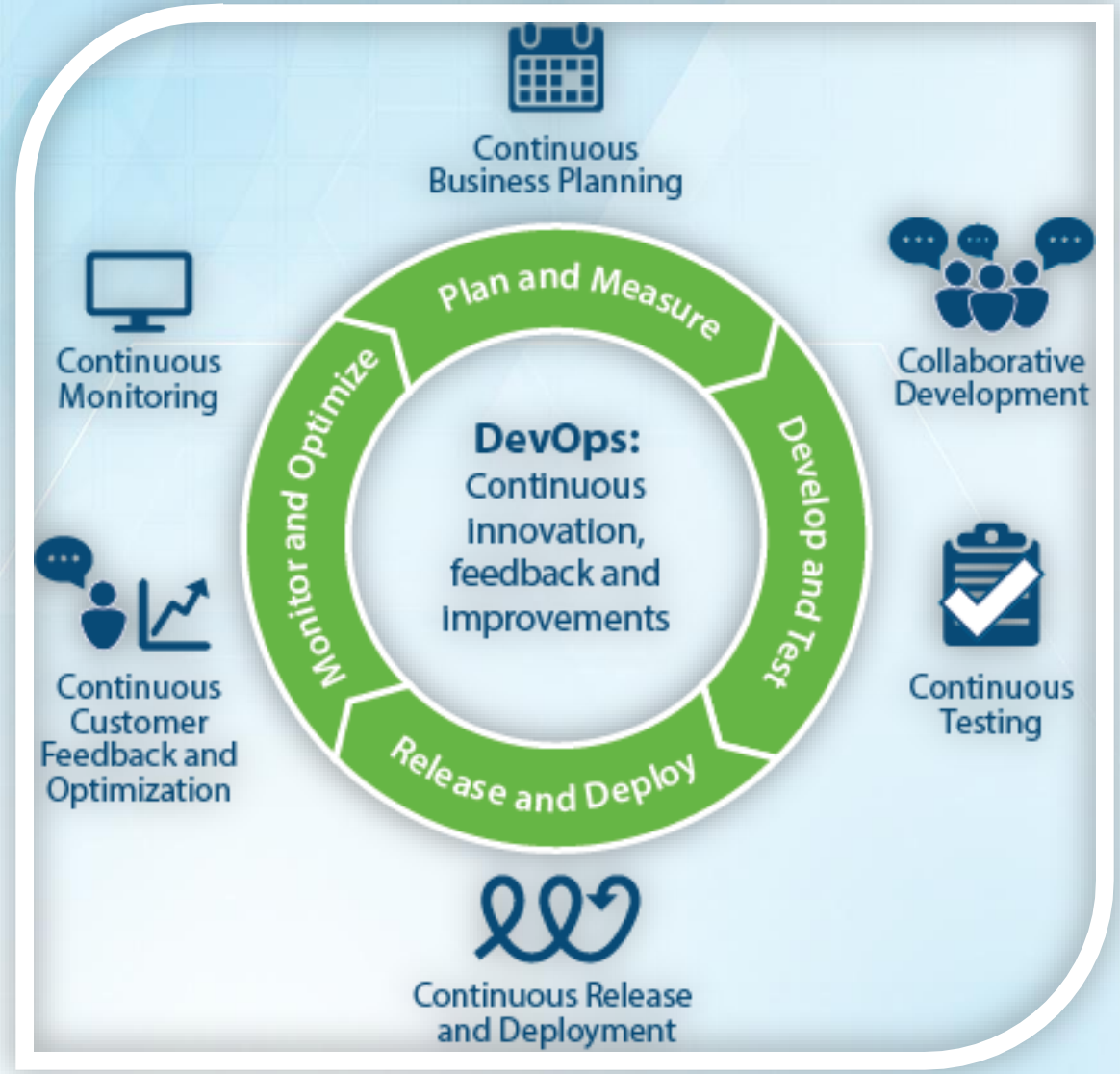
עם DevOps:

- Dev ו- Ops משחקים באותה קבוצה
- Dev ו Ops חולקים את אותן מטרות

יעדים אלה כוללים דברים כמו:

- זמן הגעה מהיר לשוק (Time To Market)
- מעט מאוד כשלים בתהליך הייצור
- התאוששות כמעט מיידית מתקלות

מטרות תרבות DevOps



אוטומציה לתהליך הבילד

INT

המרכז הבינלאומי
ללימודי הייטק וחדשנות

מה היא אוטומציה לתהליך הבילד?

אוטומציה של תהליך הכנת הקוד לפריסה לסביבת ה-LIVE

בניית אוטומציה:

- בהתאם לשפות שבהן נעשה שימוש, יש לקמפל את הקוד, לקשר, לשנות, לעשות בדיקות יחידה וכו'.
- בניית אוטומציה פירושה נקיטת צעדים אלה וביצועם באופן עקבי ואוטומטי באמצעות סקריפטים או כלים
- הכלים של בניית האוטומציה שונים לעתים קרובות בהתאם לשפות התכנות והמסגרות בהן משתמשים, אך יש להם דבר אחד במשותף: **אוטומציה**!

איך נראית פריסת קוד אוטומטית?

- בדרך כלל, אוטומציה של בנייה נראית כמו הפעלת כלי שורת פקודה שבונה קוד באמצעות קבצי תצורה ו/או סקריפטים המטופלים כחלק מקוד המקור
- אוטומציה של בנייה אינה תלויה ב- IDE) סביבות עבודה)
- גם אם אתה יכול לבנות בתוך ה- IDE, זה אמור להיות מסוגל לעבוד באותו אופן מחוץ ל- IDE) למשל בתוך ענן כלשהוא)
- ככל האפשר, אוטומציית בנייה צריכה להיות אגנוסטית לתצורת המכונה עליה היא בנויה
- הקוד שלך אמור להיות מסוגל לבנות על מחשב של מישהו אחר באותו אופן שבו הוא נבנה על שלך

למה אנו צריכים פריסת קוד אוטומטית?

- אוטומציה של בנייה היא מהירה - אוטומציה מטפלת במשימות שאחרת היו צריכים להיעשות באופן ידני.
- אוטומציה של בנייה עקבית - הבנייה מתרחשת באותה צורה בכל פעם, ובכך היא מסירה בעיות ובלבול שיכולים לקרות תוך כדי בנייה ידנית.
- אוטומציה היא רפטטיבית - ניתן לבצע את הבנייה מספר פעמים עם אותה תוצאה. תמיד ניתן להפוך כל גרסה של קוד המקור לקוד בר פריסה בצורה עקבית.
- אוטומציה של בנייה היא ניידת - הבנייה יכולה להתבצע באותו אופן בכל מכונה. כל אחד בצוות יכול לבנות על המחשב שלו, כמו גם על שרת בנייה משותף. קוד בנייה אינו תלוי באנשים או במכונות ספציפיות.
- אוטומציה של בנייה אמינה יותר - יהיו פחות בעיות הנגרמות על ידי בנייה גרועה.

Continuous Integration

INT

המרכז הבינלאומי
ללימודי הייטק וחדשנות

מה הוא Continuous Integration (CI)?

- אינטגרציה מתמשכת :(CI) פרקטיקה של מיזוג תדיר של שינויי קוד שנעשו על ידי מפתחים
- באופן מסורתי, מפתחים היו עובדים בנפרד, אולי במשך שבועות בכל פעם, ואז ממזגים את כל הקוד העבודה שלהם ביחד בסוף במאמץ אחד גדול
- אינטגרציה מתמשכת פירושה התמזגות מתמדת לאורך כל היום, בדרך כלל עם ביצוע בדיקות אוטומטיות לאיתור בעיות שנגרמו מהמיזוג
- מיזוג מתמיד יכול לייצור המון עבודה, לכן על מנת למנוע זאת התהליך הזה חייב להיות אוטומטי

איך נראית אינטגרציה מתמשכת?

- אינטגרציה רציפה נעשית בדרך כלל בעזרת שרת CI.
- כאשר מפתח מבצע שינוי קוד, שרת ה- CI רואה את השינוי ובאופן אוטומטי מבצע בנייה, גם מבצע בדיקות אוטומטיות ועוד המון תהליכים אפשריים.
- זה יכול להתרחש עשרות פעמים פעמים ביום - והכל בצורה אוטומטית!.
- אם יש בעיה כלשהי בבנייה, שרת ה- CI מודיע באופן מידי ואוטומטי למפתחים (**הבילד נפל**!).
- אם מישהו מבצע קוד ש"שובר את הבנייה" הוא אחראי לתקן את הבעיה או לבטל את השינויים שלו באופן מידי כדי שמפתחים אחרים יוכלו להמשיך לעבוד.
- בהרבה מקרים - סטים של בדיקות (למשל סטטיות) מונעים מיזוג של קוד לא טוב והכל מתוקן בשלב מוקדם והפרטני (בראנצ' פרטי).

למה אנו צריכים אינטגרציה מתמשכת?

- **זיהוי מוקדם של סוגים מסוימים של באגים** - אם הקוד לא מתחבר או בדיקה אוטומטית נכשלת, המפתחים מקבלים הודעה ויכולים לתקן זאת מיד. ככל שהבאגים הללו מתגלים מוקדם יותר, כך קל יותר (וזול יותר) לתקן אותם!
- מבטל את הטירוף למיזוג הקוד רגע לפני שחרור הגרסה - הקוד מתמזג כל הזמן, כך שאין צורך לבצע מיזוג גדול בסוף, לקראת השחרור עצמו וכך גם מבצעים את הבדיקות לפני המיזוג (למשל CR) לא בלחץ.
- מאפשר שחרורים תכופים - הקוד נמצא תמיד במצב שניתן לפרוס לבילד סופי ומוכן לשימוש.
- מאפשר בדיקה רציפה - מכיוון שניתן תמיד להפעיל את הקוד, בודקי QA יכולים לשים את ידם לאורך תהליך הפיתוח, לא רק בסופו.
- מעודד שיטות קידוד טובות - התחייבויות תכופות מעודדות קוד פשוט ומודולרי.

Continuous Delivery and Continuous Deployment

מה היא אספקה\מסירה רציפה?

- מסירה רציפה :(CD) פרקטיקה של שמירה מתמדת על קוד במצב ניתן לפריסה.
- ללא קשר אם התקבלה ההחלטה לפרוס או לא, הקוד נמצא תמיד במצב שהוא מוכן לפריסה.
- חלק גדול מתעשייה משתמשים במונחים אספקה רציפה (Continuous Delivery)

ולסירוגין, פריסה רציפה (Continuous Deployment) , או פשוט משתמשים בקיצור CD.

מה היא פריסה רציפה?

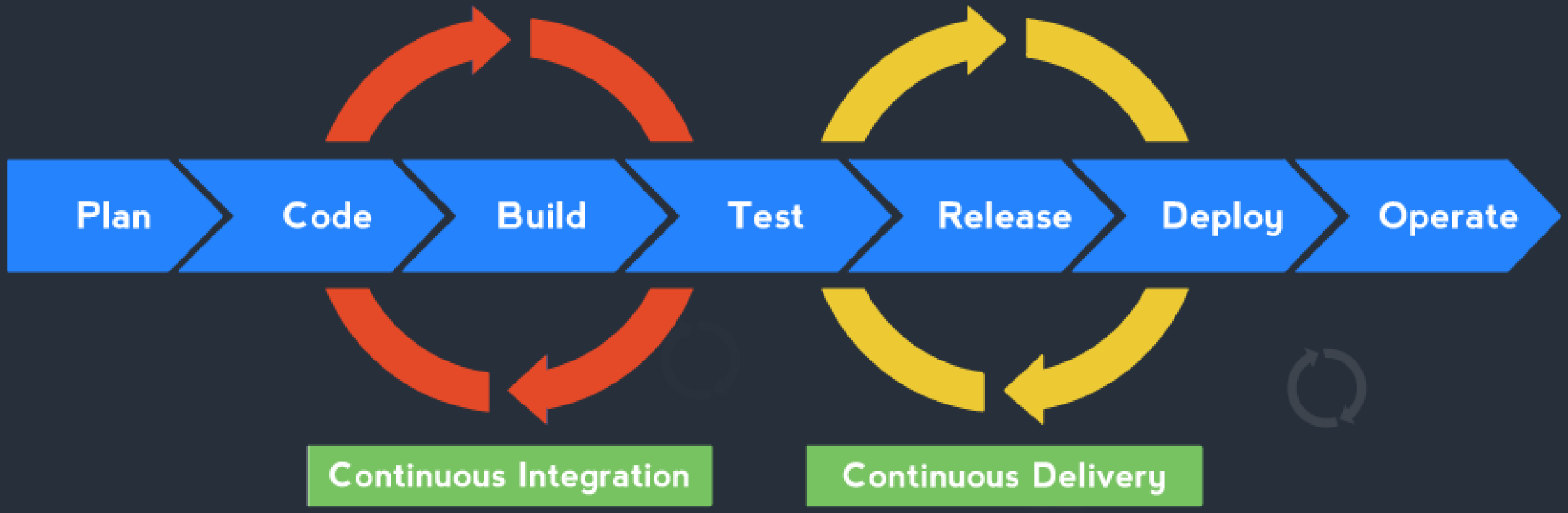
- **פריסה רציפה**: הנוהג של פריסה תכופה של שינויי קוד קטנים ל-production (סביבה חיה של הלקוח כמו אפליקציה או אתר משל מכירות).
- **מסירה רציפה** היא שמירה על הקוד במצב בר פריסה. פריסה רציפה היא למעשה עושה את הפריסה לעתים קרובות.
- **חברות מסוימות** שעושות פריסה רציפה פורסות ל-production מספר פעמים ביום.
- **אין תקן מוגדר** לתדירות הפריסה, אך בכללי, ככל שמפריסים יותר תדיר, זה יותר טוב.
- **עם פריסה רציפה**, פריסות ל-production זה אירוע שגרתי ולא אירוע גדול ומפחיד.

איך נראית מסירה ופריסה רציפה?

- כל גרסה של הקוד עוברת סדרה של שלבים כמו בנייה אוטומטית, בדיקות אוטומטיות ובדיקות קבלה ידניות. התוצאה של תהליך זה היא artifact או חבילה שניתן לפרוס.
- כאשר מתקבלת ההחלטה על פריסה, הפריסה מתבצעת אוטומטית. איך נראית הפריסה האוטומטית תלוי בארכיטקטורה, אבל לא משנה מהי הארכיטקטורה, הפריסה היא אוטומטית.
- אם פריסה גורמת לבעיה, היא מוחזרת (**rollback**) במהירות ובאמינות באמצעות תהליך אוטומטי (בתקווה לפני שלקוח בכלל יבחין בבעיה!)
- החזרות לאחור הן לא עניין גדול מכיוון שהמפתחים יכולים לפרוס מחדש גרסה קבועה ברגע שתהיה להם אחת זמינה.
- גם אם הפריסה אכן גורמת לבעיה - בדרך כלל השדרוג נעשה לא בשעות הפעילות של האתר ולכן תמיד אפשר לחזור לגרסה שעובדת עד שהבעיה תתוקן. ולכן אין מה לפחד - זה תהליך מבוקר.

למה אנו צריכים מסירה ופריסה רציפה?

- **זמן יציאה מהיר יותר לשוק** - קבלת עדכונים ותיקונים אצל הלקוחות מהיר יותר במקום לחכות לתהליך פריסה ארוך שלא קורה לעתים קרובות.
- **פחות בעיות -** הנגרמות כתוצאה מתהליך הפריסה - מכיוון שתהליך הפריסה נמצא בשימוש תדיר, כל בעיות בתהליך מתגלות בקלות ובמהירות רבה יותר.
- **סיכון נמוך יותר** - ככל שנפרסים יותר שינויים בבת אחת, כך הסיכון גבוה יותר. פריסה תכופה של שינויים בודדים בלבד היא פחות מסוכנת.
- **רולבקים אמינים** - אוטומציה אמינה ביחד עם אפשרות לעשות החזרה (Rollback) הן דרכים אמינות כדי לוודא יציבות בישביל לקוחות, ורולבקים אינם כואבים למפתחים כי הם יכולים להפיץ תיקון ברגע שהוא מוכן.
- **פריסות ללא פחד** - אוטומציה חזקה בתוספת היכולת לחזור במהירות, פירושה שהפריסה היא אירועים שגרתיים, יומיומיים ולא אירועים גדולים ומפחידים.


Plan
Code
Build
Test
Release
Deploy
Operate
Continuous Integration
Continuous Delivery