



הקשר המובהק בין Docker לפלטפורמת המחשוב השלישית

איך אפליקציות הסמארטפון מקבלות עדכונים בתדירות גבוהה מאוד ע"מ להישאר רלוונטיות?

כיצד הן תמיד זמינות?

היום בעידן המובייל, אפליקציה שלא מעודכנת בתדירות גבוהה, שלא מקבלת פיצ'רים חדשים, נחשבת למיושנת.

קרוב לוודאי שמיד ניכנס לחנות ה App store ונוריד אפליקציה אחרת, טובה וחדשה יותר. הנושא רלוונטי לכל תחום שמעורב בו פיתוח. החל ממערכת ה CRM הבאה, או כול מוצר אשר מצריך פיתוח. כיצד ניתן לפתח, לבדוק, להוציא לייצור, להוציא תיקוני באגים בזמן כל כך מהיר ובתחרות מסחררת?

המושג **Time To Market** הפך להיות קריטי.

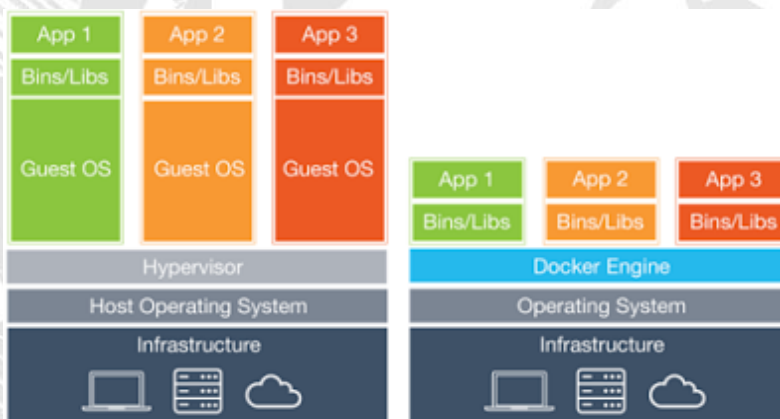
כיום, רוב תעשיית ה IT מושתתת ע"ג תשתיות סטנדרטיות. הכל ב High Availability מלא. כך גם האפליקציות של הארגון בנויות. ע"ג אותה ארכיטקטורה.

מה זו פלטפורמה שלישית במחשוב?

הענן הציבורי, "אפליקציות שנוגדו בענן". **Docker Containers** מדברים על ארכיטקטורה שונה לחלוטין ממה שקיים כיום בארגונים – אפליקציות אשר בנויות להיות "צפות", בתזוזה, בפריסה אוטומטית.

אפליקציות שמודעות שהתשתית הפיזית עליהן הן רוכבות היא לא ב 100% יתירות (High availability). אפליקציות שמחזור הפיתוח, השדרוג והכנסה לייצור מהירה פי עשרות מונים. ככל שתהיה זליגה של מערכות אלו לתוך הארגון, אימוץ הענן הציבורי, על שלל יכולותיו (לא רק תשתיות, DR, גיבוי) ואומר החלפת המערכות הישנות (legacy) באפליקציות אלו, כך השינוי יהיה חד ומהיר יותר.

"Containers - וירטואליזציה של מערכת הפעלה"



בעבר כל מערכת הפעלה יושבה על ברזל בודד. עם VMware לקחנו ברזל בודד והרצנו עליו עשרות מערכות הפעלה נפרדות. כל מערכת הפעלה משרתת אפליקציה בודדת. בפועל זה אומר שכל VM זו מערכת הפעלה שצורכת משאבים, היא גדולה, יש לבצע עדכון וטיפול שוטף שלה, וזה עוד לפני שהתחלנו להתקין עליה אפליקציות. **Containers** מאפשר לנו להגדיר על ברזל אחד, מערכת הפעלה אחת ואותה לדמות למספר רב של מערכות הפעלה.

"מערכת הפעלה אחת ויחידה"

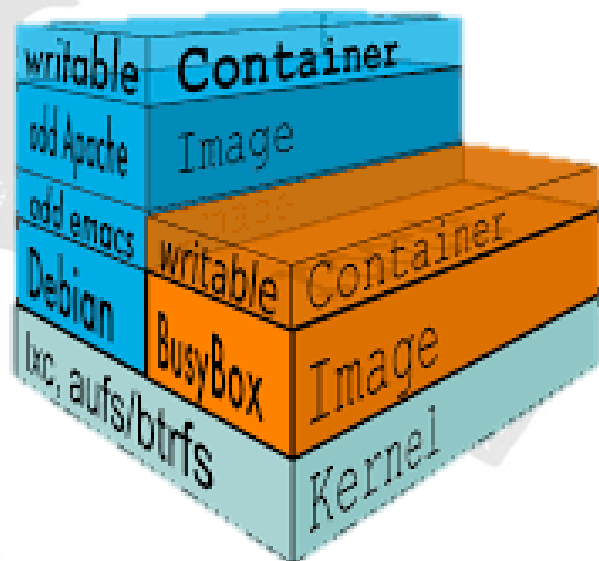
בצורה זו אפשר לדמות מערכת הפעלה יעודית לכל אפליקציה. ז"א שניתן להעמיס על שרת, ברזל, עשרות מערכות המורכבות מבסיסי נתונים, שרתי אפליקציה, מערכות קבצים ועדיין לנהל מערכת הפעלה אחת ויחידה. זה מאפשר לאפליקציות להיות קטנות משמעותית, מאפשר למפתחים לא לדאוג לשכבת מערכת ההפעלה וגורם לאחידות מערכות ללא קשר על איזה ברזל ופלטפורמה המפתח פיתח את האפליקציה, על איזו תשתית המערכת רצה בייצור והלכה למעשה פחות חיכוכים והאשמות הדדיות בין אנשי התשתיות לאנשי הפיתוח.

Docker Containers הינו המנוע שמאפשר לכל האופרציה הזו לעבוד. הוא עובד בצורה מדהימה ומקצר את תהליך הפיתוח עד לייצור בעשרות מונים!

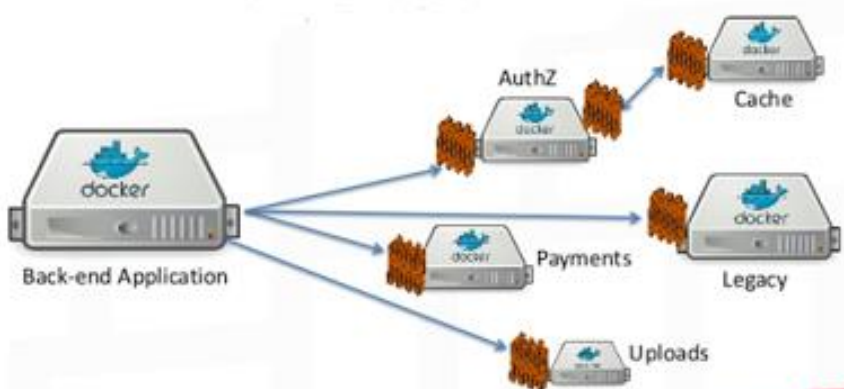
בקצרה, כל הפיתוח מבוצע בשכבות של שירותים. כל שכבה מנוהלת ומעודכנת בנפרד ונשמרת במיקום מרכזי - Repository. כל שכבה כזו מדמה גרסה משלה. כל פעם שמפתח צריך לקבל אליו את הגרסה החדשה – הוא מושך רק את אותה שכבה שנוספה ולא את כל המערכת מחדש.

בכל תהליך כזה נוצר טמפלט שזה למעשה האפליקציה על כל ההגדרות, כל הרכיבים מוכנים. בתצורה הזו אין קבצי התקנה, אין התקנות וקנפוגים של אפליקציות. המשמעות היא שבמידה ומשהו השתבש באפליקציה אני לא צריך לתקן או להתקין מחדש אלא פורס מחדש את האפליקציה. בתצורה זו אין התקנה של אפליקציות. אני מקבל גרסה מלאה ואחידה. לא משנה אם אני מפתח על MAC, על לינוקס, מעביר לסביבת בדיקות על שרת ישן או על מערכות בייצור.

תמיד זו תהיה אותה מערכת עם אותן הגדרות ב ד י ו ק! **Build - Ship – Run**



Containers נקרא לפעמים Micro Services עקב השימוש בשכבות. הארכיטקטורה של בניית האפליקציה נחשבת למסובכת וקשה מפני שבתצורה זו אנו מפרקים את האפליקציה למספר שירותים, Services, נפרדים אשר בנויים ומטופלים בנפרד. מצד שני, הפיתוח, שינוי גרסאות, שדרוג, טיפול בתקלות, פריסת מערכות בייצור, נחשב לקליל ומהיר לאין שיעור מאפליקציות legacy - אחת הסיבות לפופולאריות בקרב קהילת הפיתוח.



בנוסף לארכיטקטורת ה-Micro services, בה אנו מפרקים את האפליקציה למספר רכיבים, יש אפשרות לנקוט בגישה מונוליטית. משמע, לקחת אפליקציה סטנדרטית קיימת ואותה להעביר לסביבת Docker Container.

הערך המוסף - גמישות!

כיוון שבודדתי את האפליקציה ממערכת ההפעלה והתשתית עליה היא רוכבת בתצורה זו האפליקציה "צפה", האפליקציה יכולה לעבור מסביבת VMWare אצלי ב Data Center, לסביבת ענן ציבורי או לכל מקום בעולם אשר עובד בטכנולוגיית **Docker Containers** ולא משנה אם זה לפטופ, Azure, Google, או אצלי ב Data Center.

אין משמעות לתשתית עליה היא רוכבת (VMWare, KVM, Azure) – כל עוד זו מערכת לינוקס נתמכת - הכל פיקס!

האם המשמעות היא שמערכות אחסון לא יהיו רלוונטיות ?

Docker Containers הם למעשה טמפלטים של מערכות ואפליקציות שלמות - Stateless. הם אינם מחזיקים את ה Data. כאשר ה Container יכבה, כל ה DATA תמחק. אם נרצה לקשר לאותן אפליקציות מידע כגון בסיסי נתונים, קבצי רשת או כל מידע חשוב שחייב להישמר לאחר כיבוי המערכת - חובה להשתמש באותן יכולות של מערכת אחסון מרכזית.

NetApp Docker Volume Plugin (ndvp) מאפשר ליצור ולקשר ווליומים מתוך ממשק Docker אל ה Container עצמו ובכך להנות מה State Data וגם לאפשר פריסה אוטומטית של ה Data בעזרת כלי ניהול, כגון kubernetes.

באם סביבת ה Docker שלכם עובדת מקומית, בענן הציבורי או בסביבה היברידית, **NetApp** מאפשרת לנו סט כלים שלם לניהול מחזור חיי המידע ולהנות מעילות מקסימלית. מוצרים חדשניים לסביבות הענן ההיברידי כגון **ONTAP Cloud, Cloud-Sync, StorageGrid (Object Storage)** יאפשרו את פריסת ה DATA כגון קבצים, בסיסי נתונים וה Registry (Image Repository) בכל הסביבות - בין ענן ציבורי ובין האתר המקומי.