ניתוח ביצועים

Windows Docker Containers vs Linux Docker Containers

על מנת להשוות את הביצועים של Containers מעל אירוח Windows לעומת Linux נצטרך סביבה שתומכת בתנאים הבאים:

1) שרתים בהם ניתן להריץ Docker Containers. כלומר מותקן בהן Docker Engine.
2) גם שרתי Windows וגם שרתי Linux עליהם יהיה ניתן לרוץ עם מעבד וזיכרון RAM זהה לחלוטין.
3) בחירת שפה ו-Framework ל-Benchmark - עקב הנחיתות של של Windows לעומת Linux, כמו לדוגמא, Java שלא מומלץ בשום אופן להריץ אותה על Windows ולכן גם אין לה אופטימזציות על סביבת Windows מה שיוביל לתוצאות לא שוויוניות בין מערכות ההפעלה.
4) כתיבת מקרי הבדיקות. הבדיקות צריכות להיות מצד אחד כללים, מצד שני צריכים לאתגר במיוחד את מערכת ההפעלה, את הגישה לקבצים, את הביצועים בבקשות רשת וכמובן את יעילות השימוש ב CPU המתבטאת בקוד שצורך הרבה זמן עיבוד וקוד שמאתגר את המעבד במקביליות הCPUs.

את כל אלה פתרתי כך:

1. בחרתי את Azure כענן שיודע להריץ לי Containers as a Service. תשתית זו נקרת ACI, היא לא מעל Kubernetes, ויודעת לרוץ גם מעל Windows וגם מעל Linux.
2. הענן Azure יודע להריץ את ה-ACI Containers גם מעל Windows וגם מעל Linux עם CPU ו-RAM לבחירה אישית, כך שניתן לבחור מעבד ו RAM זהים עבור שני מקרי הבדיקה.
3. בחרתי ב DotNet Core שמצד אחד על פי ה- best practice, מומלץ להריץ אותה על Linux, מצד שני Microsoft עובדת שעות נוספות על מנת לגרום לאנשים לרצות להריץ .net על Windows מה שמוביל לאופטימיזציה של .net גם על גבי Windows. בנוסף .Net זו שפה שיהיה לי נוח לפתח בה.
4. השתמשתי ב80% מקוד של פרוייקט: https://github.com/dotnet/performance. פרויקט זה הוא כחלק מפרויקט DotNet הגדול, בו Microsoft כותבים Benchmarks שוטפים עבור DotNet על מנת לוודא כי הביצועים של התשתית יציבים וכמצופה.
   בנוסף, כתבתי כמה טסטים שלי שמשלימים ניתוחים שרציתי לבדוק שלא היו בפרויקט של Microsoft כמו שימוש בMutex, Heavy load of Threads, Cpu intensive code, Heavy memory allocations.

Technical Specs

- הקוד רץ מעל .Net Core 2.2.301 שזוהי הגרסה האחרונה היציבה של .Net Core לזמן ביצוע Benchmarking
  - CoreCLR 4.6.27817.03, CoreFX 4.6.27818.02), 64bit RyuJIT)
- שני השרתים רצו עם המעבד הזה : Intel Xeon CPU E5-2673 v4 2.30GHz, 1 CPU, 4 logical and 2 physical cores
- Windows
  - Image - mcr.microsoft.com/dotnet/core/sdk:2.2-nanoserver-1809
  - Hosting - OS=Windows 10.0.17763.0 (1809/October2018Update/Redstone5)
- Linux
  - Image - mcr.microsoft.com/dotnet/core/sdk:2.2-stretch
  - Hosting - OS=debian 9
- האחסון הוא האחסון הבסיסי והמינימלי ש-Azure מספקים.
  - חלק מה Benchmarks עובדים עם הדיסק המסופק ע"י Azure.

שלבי ביצוע ה- Benchmark

- הכרה של DotNetBenchmark Framework, שעובד שונה מעל Containers
https://benchmarkdotnet.org/
- הבנה בקוד של Microsoft שכתבו Performance Benchmarks עבור DotNet.
https://github.com/dotnet/performance
- בניית Docker FIles מינימליים עבור Windows based ו Linux based.

DockerfileLinux

```
FROM mcr.microsoft.com/dotnet/core/sdk:2.2-stretch
WORKDIR /src
COPY ["DockerBenchmark/DockerBenchmark.csproj", "DockerBenchmark/"]

RUN dotnet restore "DockerBenchmark/DockerBenchmark.csproj"
COPY . .

WORKDIR "/src/DockerBenchmark"
RUN dotnet build "DockerBenchmark.csproj" -c Release

WORKDIR "bin/Release/netcoreapp2.2"
ENTRYPOINT ["dotnet", "DockerBenchmark.dll"]
```

DockerfileWindows

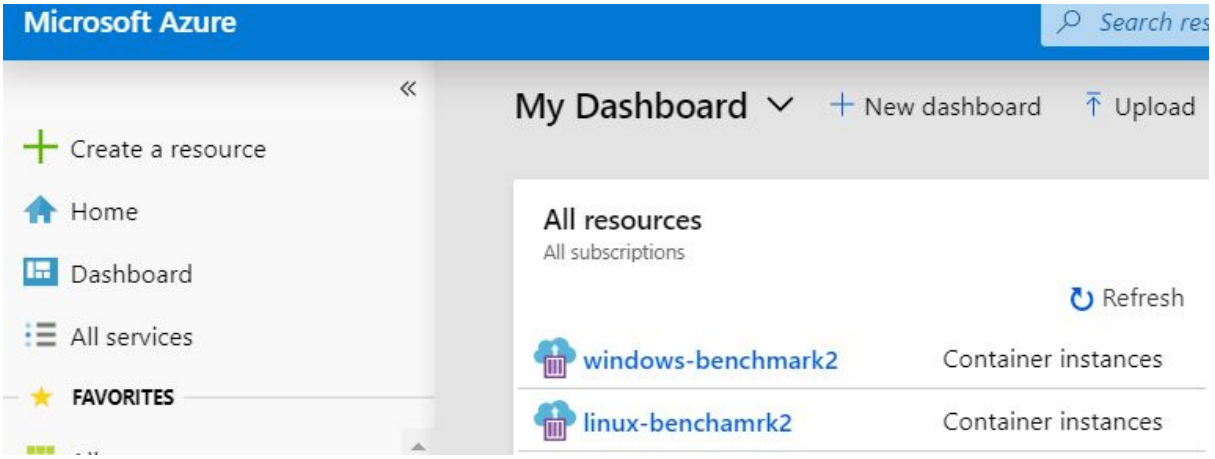
```
FROM mcr.microsoft.com/dotnet/core/sdk:2.2-nanoserver-1809 AS build
WORKDIR /src
COPY ["DockerBenchmark/DockerBenchmark.csproj", "DockerBenchmark/"]

RUN dotnet restore "DockerBenchmark/DockerBenchmark.csproj"
COPY . .

WORKDIR "/src/DockerBenchmark"
RUN dotnet build "DockerBenchmark.csproj" -c Release

WORKDIR "bin/Release/netcoreapp2.2"
ENTRYPOINT ["dotnet", "DockerBenchmark.dll"]
```

- בחירת מקרי בדיקה רצויים
- כתיבת מקרי בדיקה חסרים (Mutex,Memory Allocations, HeavyLoad MultiThreading)
- חלוקת מקרי הבדיקה לפי קטגוריות.

הקוד נמצא בRepo שלי בGithub:
https://github.com/YogiBear52/BenchmarkDockerWin-Linux

- הכרת Azure ותשתית הACI שלהם להרצת Containers
https://azure.microsoft.com/en-us/services/container-instances/

- הרצת הקוד והניסוי פעם מעל Linux Hosting ופעם מעל Windows Hosting.



- ייבוא תוצאות הניתוחים מה-Containers בAzure.

"az container exec --resource-group YogevResourceGroup --name linux-benchamrk2 --exec-command "/bin/bash"

- ניתוח התוצאות והבנה ראשונית של הפלטפורמה היעילה מבין השניים (Windows vs Linux)
- הבנת הניתוחים לעומק וכתיבת מסקנות ספציפיות בכל קטגוריה של ניתוחים.

צפי הניסויי

אני צופה כי הביצועים של Linux Containers יהיו גבוהים יותר. הסיבות כמובן לא מבוססות, ולשם זה נועד Benchmark זה.
הערכה זו מבוססות על ניתוח ביצועים מעל שרתי VM של שתי מערכות ההפעלה, ודעות ידועות על מערכות ההפעלה כגון:

- מערכת ההפעלה Windows כבדה הרבה יותר ולכן גם הביצועים עלולים לרדת. (Linux הרבה יותר Lightweight).
- מערכת ההפעלה Linux מחזיק ב-Kernel מרשים יותר, מה שעושה את רוב ההבדל בגישה לקבצים ושימוש ב CPU, RAM.

תוצאות

את המסקנות אציג לפי קטגוריות, ובכל אחד מהמבחנים אציג את ההשוואה בין התוצאות ב-Linux container אל מול Windows container.

File System:

Method	Units	Linux Mean	Windows Mean	Differences
Get Current Directory	ns	894.4	157.7	567.20%
Directory Exists	ns	1,745.40	12,430.50	712.20%
Get Logical Drives	ns	73,238.80	579.30	12642.60%
Create Directory	ns	3,351.90	18,794.00	560.70%
File Exists	us	1.76	13.11	744.00%
Delete File	us	10.97	281.49	2567.20%

כאן אפשר לראות הבדלים ניכרים בין שתי מערכות ההפעלה.
אם כי יש כמה פעולות ש Windows מנצח בהן, נראה שעדיין Linux מוביל בגישה למערכת הקבצים - בעיקר במיפוי, יצירה ומחיקה.
דבר זה לא מפתיע, הרי זה ידוע כי מערכת הקבצים של Linux **הרבה** יותר פשוטה ויעילה.

** מערכות הקבצים אינו זהות בין שתי מערכות ההפעלה, אך הן מהוות Best Practices עבור כל אחת, ולכן ניתן להניח כי ניתן להשוות בין הביצועים.

Networking:

Method	Units	Linux Mean	Windows Mean	Differences
Send Async Then Receive	ms	125.5	57.09	219.80%
Receive and Send Async	ms	325.70	531.03	163.00%
Send Async Then Receive SocketAsyncEventArgs	ms	118.10	52.34	225.60%
Receive and Send Async SocketAsyncEventArgs	ms	323.30	517.22	160.00%

בניתוח זה, בדקנו פעם אחת כאשר השרת קודם שולח חבילות ואז מקבל חבילות רשת, ובבדיקה השנייה בדקנו שהשרת גם מקבל וגם שולח חבילות בו-זמנית.
נראה שכאשר אנחנו או שולחים או מקבלים בזמן נתון, Windows מבצע זאת ביתר יעילות. אך כאשר גם מקבלים וגם שולחים בו-זמנית, שזה המקרה הנפוץ ביותר והחשוב יותר בו שרת מתמודד עם הרבה פניות בו זמנית וצריך בכל זמן נתון גם לשלוח חבילות רשת וגם לקבל, Linux מבצע זאת ביתר יעילות.
לסיכום, Linux מתמודד עם בקשות רשת בצורה יעילה יותר.

Memory Allocation:

Method	Units	Linux Mean	Windows Mean	Differences
Memory Allocation - Medium load	ms	256.8	231.4	111.00%
Memory Allocation - Heavy load	ms	1,481.70	1,295.60	114.40%

** Allocation Gen division where about the same

המנצח הברור והמפתיע של הניתוח זה הוא Windows, אשר מצליח להקצות זכרון פי 1.11-1.14 מהר מאשר לינוקס.
כאשר בשתי מערכות ההפעלה, ה GC תפקד בדיוק אותו דבר.

ניתוח זה מאוד מעניין הרי הקצאת זכרון הוא דבר שכיח מאוד בכל קטע קוד, דבר הקורה לעיתים תכופות ומעורב כמעט ובכל פעולה בסיסית, מה שמדגיש את רמת חשיבות היעילות בהקצאות זכרון.

מערכת ההפעלה Windows מנצחת, באופן מפתיע.

OS:

Method	Units	Linux Mean	Windows Mean	Differences
Lock with Mutex	ms	496.2	401.4	123.60%
Get current Cultureinfo	ms	11.81	14.55	123.20%
Get Current Date	ns	117.18	71.79	163.20%
Get Current Date UTC	ns	34.74	38.74	111.50%
Get one Environment Variable	ns	689.6	232.3	296.90%
Get all Environment Variable	ns	8,118.60	13,814.60	170.20%
Random Int	ns	14.91	15.96	107.00%
Random Bytes	ns	9,821.29	9,732.86	100.90%
Schedule Timer - Short	ns	288.10	323.80	112.40%
Schedule Timer - Long	ns	288.10	312.60	108.50%

ניסוי זה נראה מגוון בתוצאותיו וחד משמעי. נרד לפרטים בכל אחת מהבדיקות.

- פעולה 'Get Current Date UTC' הרבה יותר שכיחה ובסיסית מאשר מקבילה ללא 'Get Current Date' ולכן למרות היתרון של Windows ב 'Get Current Date', מערכת ההפעלה Linux מהירה יותר בפעולה היותר שכיחה 'Get Current Date UTC'.
- הפעולה 'Get one Environment Variable' הרבה יותר שכיחה מאשר לקבל את כל משתני הסביבה בבת אחת 'Get all Environment Variable' ולכן יש פה יתרון ברור ל Windows שיעיל **פי-3** מאשר Linux.
- את הפעולה 'Lock With Mutex' החשבתי תחת בדיקות של מערכות הפעלה למרות שגם אפשר להחשיב אותה תחת בדיקות MuliThread. בדיקה זו מראה יתרון קל לWindows בשימוש בנעילות בתוך מערכת ההפעלה.

- בניתוחי הגרלת מספרים ותזמון שעון, Linux מוביל עם יתרון קל.

לסיכום, אין מנצח ברור, אך נראה ש Windows נותן פייט לא קטן ל-Linux ואפילו יעיל יותר בחלק מהמקרים.

Multitasking:

Method	Units	Linux Mean	Windows Mean	Differences
Empty Async Method Invocation	ns	28.43	28.3	100.50%
Single Yield Method Invocation	ns	1,731.89	1,760.86	101.70%
MultiTasking with lock	ns	45.99	41.19	111.70%
Concurrent Insertions To Dictionary	ns	106.30	107.60	101.20%
Running Multiple Tasks Synchronously	ns	172.2	164.4	104.70%

עבודה מקבילית זו אחת הבדיקות החשובות ביותר, שכן תוכנות מודרניות נוטות להשתמש יותר בניצול המקביליות יותר מאשר ניצול כוח עיבוד של Core בודד.
ניתן לראות כי המדדים מאוד קרובים בין Linux ל-Windows. יותר מכך, נראה שהעבודה עם Lock מהירה יותר ב-Windows, שזה מדד חשוב מאוד הרי משתמשים בו בהרבה מאוד על מנת להבטיח קוד Thread-Safe.

המנצח בניתוח חשוב זה הוא Windows.

CPU:

Method	Units	Linux Mean	Windows Mean	Differences
Regex Match	ms	2.375	2.4	101.10%
Regex Match - Heavy	ns	197.33	204.07	103.40%
Regex Match - Heavy - Multithreaded	ns	229.71	210.71	109.00%

בחרתי להשתמש בRegex כיודע בתור טוחן CPU כמדד לניצול כוח עיבוד.
לא נראה שיש מנצח בניתוח זה.

סיכום ומסקנות Benchmark

מסקנות כלליות:

- קוד שעושה שימוש רב במערכת הקבצים, עדיף שיתארח מעל Linux Containers, אשר יתן ביצועים כללים טובים **הרבה** יותר.
- קוד המבצע הרבה שליחת וקבלת בקשות רשת מקביליות, עדיף שיתארח מעל Linux Containers.
- קוד הממקסם על Multitasking והקצאת זכרון, עדיף שיתארח מעל Windows Containers.

הגעתי למחקר זה כאשר הצפי היה ליתרון מוחלט ובולט עבור מערכת ההפעלה Linux.
נראה שאכן במקרים רבים, Linux יעילה ומהירה יותר בביצועיה, אך, Windows Containers בחלק מהמקרים מתעלה על ביצועי Linux Containers, אך לרוב לא יורד משמעותית מביצועיה.

המסקנה שהייתי רוצה שיקחו ממחקר זה היא ש Windows Containers משתפר מאוד עם הזמן, Microsoft מייצרת כאן מוצר שלא נופל מLinux האגדית ושאולי יום אחד אפילו יתעלה על Linux בביצועיה.

כמובן שזהו Benchmark שבודק ביצועים נטו, ועל מנת להשוות באופן מוחלט בין Windows Containers לבין Linux Containers נצטרך לבדוק דברים נוספים כגון פשטות שימוש, שרידות לאורך זמן, משקל Containers, שיקול Security ועוד...

שורה תחתונה, Windows Containers נותן פייט רציני ל-Linux Containers, אך עדיין, למרות השיפור הניכר ב-Windows, הרצת Linux Containers עדיפה מבחינת ביצועים.

יוגב מזרחי 205707672
קורס ניתוח ביצועים, 2019 סמסטר ב'.