בדיקות ביצועים ועומסים עם Apache JMeter 5.0

שי גינזבורג

בשנה החדשה צפוייה להתגבר המגמה של התחזקות המסחר האלקטרוני. אחד המאפיינים של המסחר הזה הוא גלי צונאמי של קונים חסרי סבלנות במועדים מוכרזים מראש. כמובן שאף אחד לא רוצה להיות בעל החנות שרק היא תהיה סגורה בבלאק פריידיי. מכאן ברורה החשיבות של בדיקת העומסים בתנאים קיצוניים, כאשר גולשים רבים נוחתים לתוך אתר אינטרנט מסחרי תוך זמן קצר. השאלה הראשונה- האם האתר ימשיך לספק את השירות או יקרוס? ואם לא יקרוס, האם זמן ההמתנה לקופה יהיה מקובל על הלקוחות, או שהם ינטשו את התור ויעברו לבדוק את המבצעים אצל המתחרים? בדפים הבאים אתאר לכם את המצב הזה, ואציג בפניכם את כלי הבדיקות Apache JMeter אשר באמצעותו ניתן לבצע את התהליך השלם של בדיקות העומסים והביצועים.

שלבי הבדיקה העיקריים הם: תכנון והקלטת תסריטי הבדיקה, עיבוד ההקלטה באמצעים הקיימים בכלי, הרצת התסריט המעובד בקבוצות גדולות שמפעילות דרישות עם עומס כבד על האתר הנבדק, ולבסוף, ניתוח מאוחר של הממצאים שנאספו תוך כדי ההרצה לצורך גילוי ופתיחת צווארי בקבוק במערכת הממוחשבת הנבדקת. כל השלבים האלה מבוצעים באמצעות פונקציות הקיימות בכלי המדובר, ואני אפרט אותם תוך כדי שאני מתמקד בחידושי הגרסה החדשה ביותר.

שי גינזבורג

מהנדס מכונות, תוכנה, וביו-רפואה בעל תואר שני בטכניון ותואר שני נוסף בהצטיינות באוניברסיטת תל-אביב. יועץ עצמאי לחברות היי-טק, סטארט-אפים, חברות ביו-רפואה, גופים כלכליים ומוסדות ממשלתיים בנושא בדיקות עומסים וביצועים של מערכות אינטרנט. בעברו היה מנהל QA של LOADRUNNER - מוצר הדגל של חברת מרקורי הישראלית אשר נכלל בשעתו במדד נאסד"ק 100.

הקדמה

בחודשים האחרונים הוזמנתי להרצות שלוש פעמים בנושא בדיקות עומסים וביצועים במסגרת המפגשים של קבוצת TestIL. המפגשים התקיימו בבאר שבע, רעננה וחיפה, מול קהל מגוון שהביע עניין רב בנושא והשאלות שלא זכו למענה בגלל קוצר הזמן, קיבלו מענה אישי דרך המייל שלי sginsbourg@gmail.com. כמו כן, סיכמתי בכתב את הנושאים אשר הוצגו במפגשים, ואני שמח להגיש אותם כאן לקהל הקוראים של המגזין. אתם מוזמנים להמשיך להפנות אלי שאלות באמצעות הדוא"ל. תודתי נתונה לכל אלה אשר עמלו בהתנדבות מלאה על-מנת לקיים את המפגשים המוצלחים ברחבי הארץ וכמובן גם לכל אלה, שבזכותם יש לקהילת התוכנה מגזין נגיש ומעודכן בשפה העברית, יישר כוחכם.

רקע

הביטו מסביבכם, בשנת 2019 צפויה להתחזק מגמת הרכישות המקוונות. המסחר במניות רובו ככולו יתבצע בצורה מקוונת. אתרים ממשלתיים ימשיכו להעביר את השירותים העיקריים שלהם לאינטרנט, הבנקים וחברות הביטוח ימשיכו לסגור סניפים תוך כדי שהם משכנעים אותנו לעשות פעולות במערכות אלקטרוניות מאובטחות (יותר או פחות). בזמן הפנוי שנותר לנו, נברח אל הרשתות החברתיות הצפופות. על מנת לתמוך בקצב השימוש הגובר באינטרנט, נצטרך להשתמש במערכות שיהיו עמידות בפני עליות ומורדות חדות של תנועות הצרכנים. למשל, black Friday שהצטרף לאחרונה אל רשימת מועדי ישראל, ויחד אתו cyber Monday, יום הרווקים, יום האהבה ועוד רשימה מתארכת של ספינים שיווקיים בעלי משמעויות מגוונות ברבדים טכנולוגים וחברתיים רבים.

כהערת אגב אציין, כי בישראל מדווחים לאחרונה שמערכת סליקת האשראי הארצית אינה מתוכננת כלל להסתערות של קונים תוך שעות ספורות על בתי המסחר במדינה, ובעלי עסקים לא מעטים נחרדו לראות אנשים שעזבו את התור לקופה רק בגלל שלא הייתה אפשרות לבצע רכישה באשראי, עקב עומסים גבוהים מאוד בזמנים צפופים מאד על מנגנון הסליקה המקוון.

החדשות הטובות הן, שהכלים לבדיקת עומסים הגיעו לדרגת בשלות גבוהה ביותר, ומי שיטרח להשתמש בהם מראש, יידע לתכנן את החומרה, לתקף את הארכיטקטורה של המערכות המורכבות, לתקף את הטופולוגיה האופטימלית, ולבצע שיפורי תוכנה הדרושים כדי שהיישום ימשיך להיות זמין לכל הלקוחות, גם כאשר העם הנבחר בציון רוצה להכפיל או לשלש את מספר הטאבלטים שברשותו תוך 24 שעות בגלל מבצע "שקר" כלשהו. בהשוואה לשנים הקודמות, בהן עדיין היה יתרון מסוים ברכישת רישיונות (יקרים מאוד) לכלי בדיקת עומסים, הרי שכעת כבר אין שום הצדקה לשלם על מערכת ההפעלה, על סביבת ההפעלה, על הכלים לבדיקה וכך הלאה עד אחרון התוספים כפי שיבואר להלן.

> **"כמובן שאף אחד לא רוצה להיות בעל החנות שרק היא תהיה סגורה בבלאק פריידיי"**

בעולם של קוד פתוח - הכל פתוח, הכל קריא וזמין, ובפרט אני מדבר כמובן על Apache JMeter 5.0 החדש, אותו ניתן להריץ על גבי שרת חלונות (אבל עדיף ומומלץ מעל שרת לינוקס 64 ביט), לאחר שמתקינים סביבת Java הולמת (אשר מורידים בחינם מהאינטרנט). במהלך בדיקת העומסים והביצועים, כל גולש וירטואלי אשר נצליח להריץ מהווה Active Thread, אשר רץ במסגרת process, שהוא תהליך המהווה מכונה וירטואלית של Java. המכונה מקבלת משאבים בהקצאות ממערכת ההפעלה העיקרית (Windows server או לינוקס). למעשה, כל מערכת הפעלה שיכולה להרים מעליה מכונת Java תתאים לנו בשלב הכנת התסריטים עבור הבדיקות.

בעבר היה מקובל לחשוב שכליי קוד פתוח הם אוסף של תוכנות מקולקלות, שאף חברה לא רוצה לקנות או להציג אותם כמוצרים מהשורה, ובהעדר יחסי ציבור ממומנים, לא הייתה מודעות או נכונות כלשהיא להתייחס ברצינות לאפשרות של שימוש בקוד פתוח בפרויקטים גדולים ויקרים מאוד של הבנקים הגדולים, חברות ביטוח מובילות, משרדי ממשלה וארגונים ביטחוניים מסווגים. לשמחתנו, המהפך הגדול כבר קרה, ואני יכול להעיד ממקור ראשון, שהיום לא רק חברות הזנק בעלות אפשרויות תקציביות מוגבלות מוכנות להשתמש בקוד פתוח. התוצאות מדברות בעד עצמן.

גם ללא יחסי ציבור מתוקשרים, ברור שהכלי JMeter מגיע אלינו מבית טוב מאוד: קרן אפאצ'י לתוכנה היא תאגיד אמריקאי ללא כוונת רווח, אשר תומך משנת 1999 בפרויקטים של משפחת המוצרים Apache. הקרן תומכת בקהילה מבוזרת של מפתחים, והתוכנה שהם מייצרים היא קוד פתוח בחינם, אשר מופצת תחת תנאי רישיון אפאצ'י. המוצרים המפורסמים זמינים לציבור בכל העולם והם כוללים את שרת אינטרנט HTTP אפאצ'י, OpenOffice, Jmeter, Maven, NetBeans, Groovy ועוד.

כאשר אנו מתחברים למחשב מרוחק, אנחנו פונים אליו באמצעות פרוטוקולי תקשורת, שעוברים הלוך וחזור מאחורי הקלעים בין המחשב שלנו (צד לקוח) אל המחשב המרכזי שנותן שירות לכולם (צד שרת). עד שהאינטרנט פרץ לתודעתנו, שכליי בדיקות עומסים נמדדו במספר הפרוטוקולים הרלוונטיים בהם הם היו מסוגלים לדמות עומסים. בימים הרחוקים של שנות ה-90, הייתה תחרות של פרוטוקולים ובסיסי נתונים, שכליי בדיקות נאלצו לתמוך בהם על-מנת לתפוס גזרות חמות בשוק בדיקות הביצועים והעומסים. בוקר בהיר אחד השתנו כל החוקים, וכל בסיסי הנתונים והפרוטוקולים שלהם הפכו להיות שכבה תחתונה ומוצנעת של טופולוגיה מורכבת מאד עם פנים חדשות שנקראו HTML. כל תוכנות צד הלקוח נזנחו לטובת הדפדפנים החדשים, שנועדו לדפדף בין קישורים, שאנחנו הגולשים מקבלים בחיפושים אצל מנוע צעיר ומבטיח שקראו לו גוגל. אנשי המכירות של כלי בדיקות העומסים נאלצו להעמיד פנים, שהתמיכה באינטרנט היא טבעית לכלי שהם ניסו להפיח בו חיים חדשים, כאשר למעשה כל הארכיטקטורות השתנו וכל מדדי העומס המוכרים הפכו לגרוטאות מחלידות.

JMeter נולד בעצם אל תוך העידן החדש הזה ולכן יש בו תמיכה טבעית בכל הפרוטוקולים שבאמת חשובים מאז אותה הפיכת החצר, שהובילה לשינוי והסתיימה בכך שכל מה שהיום צריך להתקין על מחשב זה דפדפן.

בדיקות ביצועים ועומסים עם Apache JMeter 5.0

שי גינזבורג

הרשימה המכובדת הזו כוללת את: HTTP – Web, HTTPS (Java, NodeJS, PHP, ASP.NET), SOAP / REST Webservices, FTP, Database via JDBC, LDAP, Message-oriented middleware via JMS (MOM), Mail - SMTP(S), POP3(S) and IMAP(S), Native commands or shell scripts, TCP, Java Objects.

שקט מקליטים!

בכל מקרה, האיכות של כלי בדיקת עומסים וביצועים נמדדת במספר מישורים, ובמקרי קצה (שלא יפורטו כאן) יש גם סתירות בין הדרישות במישורים השונים (ואז נדרש לבנות סולם עדיפויות ולקבל החלטות הנדסיות). המישורים הראשונים בהם בוחנים את כלי הבדיקות הם המספר, האיכות והשימושיות של האמצעים שהכלי מספק לנו עבור שלב ההקלטה ובישול התסריטים. במישורים האלה, חלו לא מעט תמורות בכלי, שראוי לפרט אותם כאן. באופן היסטורי, JMeter תמיד כלל Virtual proxy שמקליט תעבורה בפרוטוקול השולט HTTP ויוצר עבורנו טיוטה מוקלטת של תסריט. הטיוטה (בפורמט JMX) מכילה כמובן קבועי זמן וקבועי אינטראקציה עם השרת שאותם צריך לנקות מההקלטה. הזמן בין תחילת ההקלטות לבין רגע מוכנות התסריט להרצת עומסים, תלוי במורכבות היישום ובכל מקרה הוא אינו שלב זניח בתכנון הלו"ז.

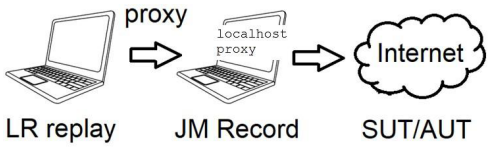
לרוב נקבל בתחילת העבודה תיאור של תסריט בסיסי, שאותו מבקשים לבחון תחת משטר עומסים. אחרי שלמדנו את התסריט לבדיקת עומס, ננסה להקליטו. הקלטת פרוקסי היא ברירת מחדל, ונותנת מענה במקרים רבים, אלא, שלא תמיד היא קלה ליישום, ולא תמיד היא תהליך הכרחי. בהקלטת פרוקסי אנחנו מאזינים בשפת פרוטוקול HTTP לכל בקשה שהלקוח שולח לשרת, ולכל מענה ששולח השרת ללקוח. במערכות ללא אבטחה מוגברת, התהליך של ההקלטה קל לביצוע ונותן מענה מושלם. כלומר, בסוף ההקלטה יש לנו טיוטה טובה, אשר תהפוך בסופו של דבר להיות Bot (בוט הוא גולש וירטואלי באינטרנט).

לעומת זאת, בניסיון להקליט מערכות מאובטחות מאוד ישנם מכשולים בדרך, אבל מצד שני, ישנם קיצורי דרך בשלב ההקלטה, שכדאי לפרט אותם. למשל, אם אנחנו לא רוצים להקליט בכלל, אז נבצע את התהליך שהיינו אמורים להקליט מבלי להפעיל כלל את פונקציית ההקלטה של JMeter, אלא את פונקציית ההקלטה של הדפדפן שבו אנחנו משתמשים כרגיל במחשב שלנו. לכל דפדפן יש אפשרויות מתקדמות להקליט תקשורת network, לשמור את ההקלטה בקובץ, ואנחנו נשתמש ביכולת הזאת. אפשר לשלוח לוגים של הדפדפן (בפורמט HAR) לאתרים מקוונים, שהופכים את הקובץ ששלחנו לקובץ דומה מאוד לזה שהיינו יכולים לקבל כתוצאה מההקלטה (בפורמט JMX).

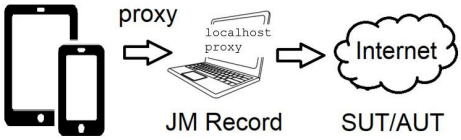
> **JMeter נולד בעצם אל תוך העידן החדש הזה ולכן יש בו תמיכה טבעית בכל הפרוטוקולים שבאמת חשובים מאז אותה הפיכת החצר, שהסתיימה בכך שכל מה שהיום צריך להתקין על מחשב זה דפדפן.**

במקרים אחרים, אם עבדנו בעבר עם כלי עומסים אחר, ויש לנו כבר תסריטים טובים, ניתן להמיר את התסריטים שלנו באופן אוטומטי לתסריטים של JMeter. זה מומלץ בעיקר אם חייבים בהווה לרכוש רשיונות יקרים כדי להריץ בדיקות עומס נוספות. יש לפחות שתי אפשרויות טכניות לבצע זאת: 1) לחפש בגוגל מי עושה את זה מקוון בחינם, להעלות את התסריט הקיים ולהוריד אחרי כמה שניות את התסריט שכבר עבר המרה. 2) במקרים בהם אין לנו גישה לאינטרנט (מה שקורה במפעלים ביטחוניים לדוגמה), אפשר לעבוד עם Double Proxy (ראו איור בהמשך). בשיטה זו יש לנגן את התסריט ממחשב אחד, כאשר הפרוקסי שלו הוא מחשב אחר, שבו מותקן JMeter, שהפרוקסי שלו מקליט את התעבורה אל השרתים בשני הכיוונים כמובן.

> **החדשות הטובות הן, שהכלים לבדיקת עומסים הגיעו לדרגת בשלות גבוהה ביותר, ומי שיטרח להשתמש בהם מראש, ידע לתכנן את החומרה, לתקף את הארכיטקטורה של המערכות המורכבות, לתקף את הטופולוגיה האופטימלית, ולבצע שיפורי תוכנה הדרושים.**



השיטה הזו, שקראתי לה Double Proxy מתאימה מאוד גם אם רוצים להקליט יישום, אשר מותקן ופעיל מתוך מכשיר נייד המופעל באמצעות מערכת Android או iOS. נכון לעכשיו, על המכשירים הניידים האלה אי אפשר להתקין Java ולכן גם אין אפשרות להריץ שם JMeter, אבל אפשר לסדר שהפרוקסי שלהם יהיה מכוון אל Host שבו עובד JMeter כמו בדוגמה למעלה. פתרון זה אינו מסובך מדי, והוא פותח את כל האפשרויות להקלטת יישומים מבלי להתקין סימולטורים של פלטפורמות ניידות, ומבלי שהם מכניסים את הרעש המיותר שלהם לתוך סביבת העבודה שלנו.



עיבוד התסריט

אחרי שכל המאמצים להקליט ולקבל בדרך זו או אחרת טיוטה של התסריט שלנו, מתחילה העבודה הקשה יותר של הפיכת הטיוטה לתסריט פעיל, עם התאמות (קורלציות), גורמים (פרמטרים) מסוגים שונים, Assertions, נקודות תיקוף (ולידציה), ביטויים רגולריים, קישורים לגליונות אלקטרונים וכדומה. אני מזכיר, בסופו של דבר אנחנו אמורים להפיק Bot שצריך להעמיס את השרתים באופן אמיתי (ריאליסטי), כלומר התסריט המעובד צריך לעבוד כמו גולש חרוץ מאד. הגרסאות החדשות של JMeter עוזרות לנו מאד בכל המשימות המהותיות האלה, שיהפכו את ההקלטות המעובדות שלנו בזמן ריצה להיות Bot.

Bot לא יכול לנגן את אותם הנתונים אשר השרת שלח לנו בזמן ההקלטה. דבר העלול להיות כמו לנהל שיחה עם אדם שבמקום לענות לנו, משמיע תשובות שהוא הקליט לנו מזמן. Bot חייב לשלוף מידע חדש מתוך התשובות של השרת, ורק אז לכלול אותם בבקשות הבאות שלו מהשרת. במקרים רבים, ביצוע אמיתי ומדויק של התסריט שלנו צריך לכלול שליפת מידע מתוך עמוד ה-HTML

שהשרת שולח. לשם כך, עומדים לרשותנו אמצעים מגוונים לתכנת את שליפת המידע. נסקור כאן כמובן את המעודכנים שבהם. ראשית כל, בגרסאות החדשות של JMeter קיימת תמיכה מובנית בפעולת שליפת הנתונים על-סמך ידיעת המיקום היחסי שלהם בתוך התשובה של השרת. את המיקום היחסי אפשר להגדיר בשפת XPath או XPath2, לדוגמה: שליפה של שם של ספר מתוך חנות ספרים שאנחנו גולשים בה.

XPath2 Extractor
Name: XPath2 Extractor
Comments:
Apply to: Main sample and sub-samples | Main sample only | Sub-samples only | JMeter Variable Name to use
Name of created variable: bookTitle
XPath query: //title/text()
Match No. (0 for Random): 1
Default Value: bookTitle_NOT_FOUND

Sampler result | Request | Response data
Response Body | Response headers

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
</bookstore>
```

```
JMeterVariables:
=Shay Ginsbourg
Downloads=C:\Users\Shay Ginsbourg\Dropbox\Studio\H
JMeterThread.last_sample_ok=true
JMeterThread.pack=org.apache.jmeter.threads.SampleP
START.HMS=105852
START.MS=1540108732624
START.YMD=20181021
TESTSTART.MS=1540108747322
__jm__Thread Group__idx=0
bookTitle=Harry Potter
bookTitle_1=Harry Potter
bookTitle_matchNr=1
```

לכל מי שנרתע מביטויים רגולריים (כי הם באמת מקבלים צורות מפחידות לעיתים קרובות), יש כלי שחוסך את כל כאב הראש הזה. כל מה שצריך לעשות במקום לתאר את תבנית החיפושים שלנו, זה לכתוב את מה שיש בצד שמאל של התבנית (Left Boundary), ובנפרד לכתוב את מה שיש בצד ימין של התבנית (Right Boundary) שאנו מחפשים. הפטנט עובד מצוין ולא דורש ידע מוקדם. לדוגמה, אם מחפשים keywords בתוך עמוד HTML, אפשר להשתמש בתבנית מהסוג הבא. (שימו לב, שבדוגמה זו נשמר התוכן בתוך משתנה שקראתי לו Fake News).

Boundary Extractor
Name: Boundary Extractor
Comments:
Apply to: Main sample and sub-samples | Main sample only | Sub-samples only | JMeter Variable Name to use
Field to check: Body | Body (unescaped) | Body as a Document | Response Headers | Request Headers | URL | Response Code | Response Message
Name of created variable: FakeNews
Left Boundary: <meta name="keywords" content="
Right Boundary: ">
Match No. (0 for Random): 1
Default Value: FakeNews_NOT_FOUND | Use empty default value

Sampler result | Request | Response data

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
        <title>Welcome to Ginsbourg.com - Ginsbourg.com</title>
                <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
                <meta name="keywords" content="Managed Services">
                <meta name="description" content="Managed Services">
                <!-- master box -->
                <link rel="canonical" href="http://www.ginsbourg.com/"/>

<meta property="og:site_name" content="Ginsbourg.com
" />
<meta property="og:title" content="Managed Services" />
```

```
JMeterVariables:
FakeNews=Managed Services
JMeterThread.last_sample_ok=true
```

בדוגמה הבאה, התסריט שלנו צריך לשלוף מידע מתוך JSON אשר שולח השרת. יש תמיכה מובנית של JMeter לשליפת ביטוי באמצעות JSON path. במקרה זה, אדגים שליפה מורכבת של מידע מתוך JSON שנמסר לנו כחלק מעמוד HTML שהשרת שלח. ראשית אפרק את המורכבות לגורמים, וכך אוכל לבצע שליפה מדויקת. כלומר, הפתרון הוא ללכוד קודם את כל ה-JSON לתוך משתנה שקראתי לו More Fake News ורק אחר כך להרכיב JSON extractor שקורא את כל ה-JSON מתוך משתנה (במקום מתוך HTML שנשלח אלינו מהשרת) ושולף מתוך התכולה של המשתנה החדש רק את השם של "משפחת סימפסון" לתוך משתנה נוסף שקראתי לו JSON Fake News.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>

<body>

<script>

var jsonData = {
  "name":"Simpsons family",
  "members":[
            {"firstName":"Homer", "lastName":"Simpson"},
            {"firstName":"Marge", "lastName":"Simpson"},
            {"firstName":"Bart", "lastName":"Simpson"},
            {"firstName":"Lisa", "lastName":"Simpson"},
            {"firstName":"Maggie", "lastName":"Simpson"}
        ]
};

</script>

</body>

</html>
```

Test Plan
- Thread Group
  - HTML 200
    - Boundary Extractor > MoreFakeNews
    - JSON Extractor > JsonFakeNews
    - Debug PostProcessor
    - Precise Throughput Timer
  - Aggregate Graph
  - View Results Tree
  - jp@gc - Response Codes per Second

Boundary Extractor
Name: Boundary Extractor > MoreFakeNews
Comments:
Apply to: Main sample and sub-samples | Main sample only | Sub-samples only | JMeter
Field to check: Body | Body (unescaped) | Body as a Document | Response Headers | Re
Name of created variable: MoreFakeNews
Left Boundary: var jsonData =
Right Boundary: ;
Match No. (0 for Random): 1
Default Value: MoreFakeNews_NOT_FOUND | Use empty default value

JSON Extractor
Name: JSON Extractor > JsonFakeNews
Comments:
Apply to: Main sample and sub-samples | Main sample only | Sub-samples only | JMeter Variable Name to use | MoreFakeNews
Names of created variables: JsonFakeNews
JSON Path expressions: $.name
Match No. (0 for Random): 1
Compute concatenation var (suffix _ALL):
Default Values: JsonFakeNews_NOT_FOUND

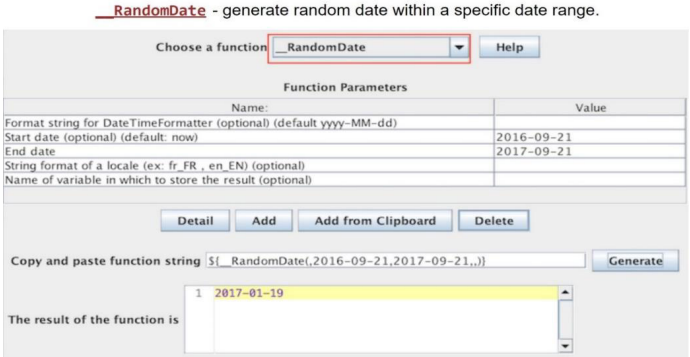
Sampler result | Request | Response data

```
JMeterVariables:
JMeterThread.last_sample_ok=true
JMeterThread.pack=org.apache.jmeter.threads.SamplePackage@43ec346e
JsonFakeNews=Simpsons family
JsonFakeNews_matchNr=1
MoreFakeNews={
  "name":"Simpsons family",
  "members":[
            {"firstName":"Homer", "lastName":"Simpson"},
            {"firstName":"Marge", "lastName":"Simpson"},
            {"firstName":"Bart", "lastName":"Simpson"},
            {"firstName":"Lisa", "lastName":"Simpson"},
            {"firstName":"Maggie", "lastName":"Simpson"}
        ]
}
```

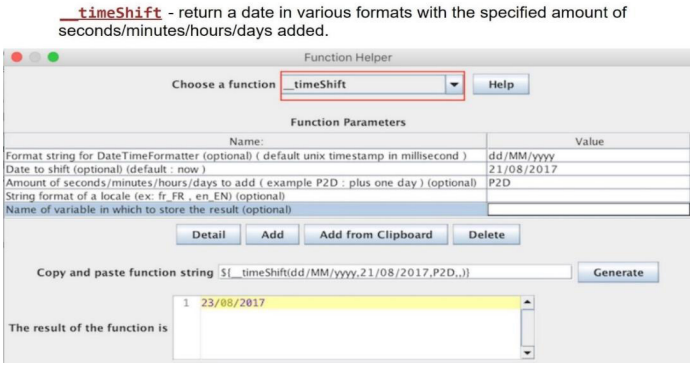
בגרסאות החדשות של JMeter, עומדים לרשותנו לא מעט כלים קטנים אך שימושיים ביותר בדרך להפיכת ההקלטה שלנו לתסריט פעיל (דינמי) בעל מידה של אקראיות, כלומר תסריט שאינו חוזר על עצמו באופן שבו השרת יכול לנבא בקלות את המשך הריצה שלו, ולהעביר את התשובות הבאות מראש אל מנגנון cache. למשל, בכל פעם שיש שימוש בתאריכים – וכמובן שיש שימוש רב בתאריכים במערכות ממוחשבות – אפשר להפוך את התאריך לאקראי בתוך טווח ידוע ובפורמט ידוע, שניתן לבחור מראש בקלות באמצעות היישום הפשוט RandomDate.

בדיקות ביצועים ועומסים עם Apache JMeter 5.0

שי גינזבורג



אם אנו רוצים להתחכם יותר אבל גם לשמור על חוקיות מסודרת, אפשר לקדם את התאריכים בקצב קבוע ובפורמט שאנחנו בוחרים באמצעות היישום הפשוט timeShift. כמו בסעיף הקודם, גם כאן את התאריך שמקבלים אפשר לשלוח אל השרת ולגרום לו לעבוד קשה יותר.



למקומות, היכון, רוץ!

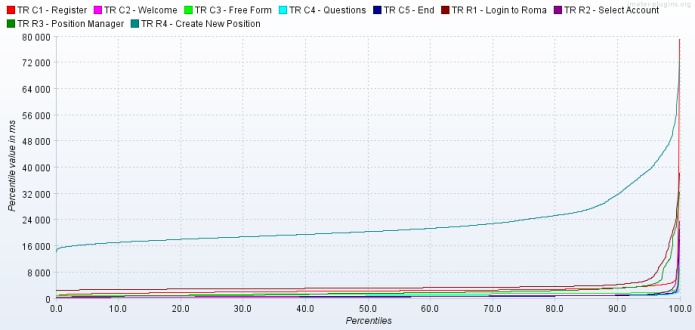
אחרי העבודה המורכבת על הקלטות של התסריטים, אנחנו אמורים לקבל תסריט שמתפקד כמו Bot שעושה הדמיה מציאותית מאוד של גולשים. השאיפה הגדולה היא, שבזמן הרצת בדיקות העומסים והביצועים ייחשפו לאור צוואר הבקבוק הנכונים שאנחנו צריכים לטפל בכלי תוכנה או חומרה, ולא צוואר בקבוק מדומים (ארטיפקט) שאינם מייצגים מצבים שקורים במציאות העמוסה.

שלב ההרצה הוא השלב המרכזי בתהליך בדיקות העומס והביצועים. קיימות בספרות מספר שיטות בנושא בחירת משטר העומס הרצוי במהלך הריצות השונות. שימו לב, עולם המושגים אינו מגובש וספרים שונים משתמשים במונחים שונים על-מנת לתאר משטרים דומים (או שונים לחלוטין) של העמסה באמצעות כלי בדיקות אבסטרקטי כלשהו.

תוכלו למצוא בספרות המקצועית הגדרות לא אחידות לבדיקת מסוג Scalability, Stability, DR, Volume, Stress ועוד. למרות שהתאוריה והמינוחים לא עברו הרמוניזציה בין המקורות הספרותיים השונים, ובפרט שאין לי שום כוונה להכתיב טרמינולוגיה נוספת של בדיקות עומסים או משטרי העמסת יישומים אינטרנטיים לצורכי בדיקות, אני רוצה בנקודה הזו לייעץ לגבי משטר הפעלת העומס ואופן קביעת הגורמים בשלב ההרצה של בדיקות הביצועים. בקיצור, תחקרו את Google Analytics (או מקורות דומים) ותבינו מראש מהו עומס הבקשות שהמערכת שלכם תצטרך להתמודד אתו בזמן רגיעה ובזמן שיא. תשתדלו לעבוד ברזולוציה של דקות ועדיף שניות. אחר כך, נסו ליצור עומסים רלוונטיים, קונסיסטנטיים וריאליסטיים באמצעות הכלי אשר עומד לרשותכם. אל תתייחסו בחשיבות מופרזת אל מספר המשתמשים הווירטואליים, אשר נדרשים לכם בו-זמנית על-מנת להגיע לעומס המתוכנן, ובמקום זה תתמקדו באחוז השגיאות הרגעי וכמובן בזמן התגובה הרלוונטי של המערכת. לסיכום, אם אחוז השגיאות שאתם מקבלים הוא אפסי, כאשר זמן התגובה הוא טוב, ומדד העומס מראה שאתם בסביבה הנכונה של זרם הבקשות, הרי שהמערכת שלכם עברה בהצלחה את בדיקת העומס. בכל מקרה אחר, נדרש לצלול אל עומקם של הנתונים על-מנת להבין היכן מצוי צוואר הבקבוק.

> **אל תתייחסו בחשיבות מופרזת אל מספר המשתמשים הווירטואליים, אשר נדרשים לכם בו-זמנית על-מנת להגיע לעומס המתוכנן, ובמקום זה תתמקדו באחוז השגיאות הרגעי וכמובן בזמן התגובה הרלוונטי של המערכת.**

במהלך ההרצה, נאספים נתונים רבים וחשובים על הביצועים של המערכות הממוחשבות תחת עומס שמחוללים הגולשים הווירטואליים שלנו. המוח האנושי לא בנוי ולא מסוגל להתמודד עם כמויות מידע עצומות, ולכן על-מנת לעכל את המידע הרב, אנחנו חייבים לצייר גרפים צבעוניים ברורים וגם לסכם בטבלאות קריאות את הסטטיסטיקה של כל התוצאות שנאספו תוך כדי הריצה. גם במישור הזה עשה JMeter קפיצה ענקית קדימה ובגרסאות החדשות כל התוצאות עוברות ניתוח סטטיסטי מובנה מראש והן מוצגות באופן מסודר מאד בתוך HTML report.



הדו"ח האוטומטי של JMeter מכין את עצמו תוך מספר שניות, והוא מציג בתבנית HTML את כל הגרפים שאתם צריכים ואת פילוח השגיאות והסטטיסטיקה של כל הביצועים, שבשבילם עשינו את כל הדרך הארוכה. הגרפים האלה מציגים את ההתנהגות של המערכת, אשר נבדקה על ציר הזמן לאורך כל הריצה: מספר המשתמשים הווירטואליים (מספר ה-Bots), זמן התגובה הממוצע, פילוח זמן התגובה לפי אחוזים (ראו דוגמה מציאותית בהמשך) ועוד. כמו כן, מתקבלים בדו"ח הזה גרפים של קצב העומס: TPS, HPS, CPS ועוד. במישור אחר, מקבלים גרף (שאני מרבה להשתמש בו להסקת מסקנות) המתאר את זמן התגובה הממוצעת של המערכת כפונקציה של מספר ה-Bots, כלומר active threads. ראוי לציין, שזה ניתוח שבו הציר האופקי של הגרף אינו ציר הזמן (ברירת המחדל הטבעית שלנו בהבנת תהליכים) ולכן הוא פותח לנו חלון חדש להבנת המערכת והיכולות המובנות בה על-מנת להתמודד עם ריבוי משתמשים בו-זמנית. בהצלחה!