Pat Stromberg - Final Project
Log Analytics

Problem Statement:
When moving solutions to the cloud, selecting appropriate sizes for your virtual machines is an issue from both a cost and performance perspective. Choose one too big, and you're paying more than you need to. Choose one too small, and your application will not scale properly. In this project, I use Log Analytics to collect various types of data during performance tests to help determine appropriate sizing of a VM.

Overview of the Technology:
To demonstrate the use of Log Analytics, we will run performance tests that simulate 5,000 search requests per second against four different virtual machines, each one housing a directory server (OpenDJ). Log Analytics is part of Operations Management Suite, which collects Linux OS-level data via the use of an agent installed on each VM. Additionally, the performance testing tool built into OpenDJ reports metrics, but not in a format understood by Log Analytics. I wrote a java tool that will watch the file that reports the performance testing results, transform them into JSON and use the Log Analytics Data Collector API to submit the results for analysis. We then will examine both dashboards and queries to visualize and compare the results.

High Level Steps:
1) Provision 4 Linux VMs of varying strength
2) Create Log Analytics objects in Azure portal and connect the VMs
3) Install OpenDJ and deploy custom java code
4) Run performance test using searchrate tool, submit to Log Analytics using custom java code
5) Analyze results via dashboard and queries in Azure Portal

Data Source:
- 50,000 sample users created automatically during OpenDJ install
- Performance data collected from VM and from the performance test results

Hardware Used:
Centos 7.4 Azure VM in four configurations over the following: 2 CPU, 8 CPU, hard disk drive, solid state drive (so, 2 CPU HDD, 2 CPU SSD, 8 CPU HDD, 8 CPU SSD)

Software Used:
ForgeRock OpenDJ 5.5 (https://go.forgerock.com/Registration-Trials-Download.html)
Azure Log Analytics
Any Java IDE (I used Eclipse)
Custom Java code (the jar is called FileTailer)

YouTube Links:
2 Min: https://youtu.be/wIE2RjrE-WM
15 Min: https://youtu.be/FeINq6da5II

GitHub Repo: https://github.com/patstromberg/da-final

Contents

Installation and Configuration

This will be divided into a few sections:

1. Creation of resource group and virtual machines
2. Installation and configuration of OpenDJ
3. Deployment of custom java code
4. Creation of Log Analytics and OMS related resources in Azure

Creation of Resource Group and Virtual Machines

Create a resource group via any means you wish. I used the Azure CLI and called it “rg-pls-dafp”

```
C:\Program Files\Microsoft SDKs\Azure\.NET SDK\v2.9>az group create -l eastus -g rg-pls-dafp
{
  "id": "/subscriptions/aeac8912-b8a9-44a7-a8d2-f3f30641cdcd/resourceGroups/rg-pls-dafp",
  "location": "eastus",
  "managedBy": null,
  "name": "rg-pls-dafp",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null
}
```

We will be attempting to determine how much power a virtual machine needs to have to support 5,000 searches per second. The machine selection will be fairly simple – if we were doing this for evaluation of a real production deployment, I'd add more machine selections, as well as expand my requirements. For this project we're using:

1. 2 CPU and a hard disk drive
2. 2 CPU and a solid state drive
3. 8 CPU and a hard disk drive
4. 8 CPU and a solid state drive

In other words, we're trying to see what moves the needle as far as directory server performance – CPU, SSD, or both.

We'll be using CentOS Linux for this. Version 7.4 is the latest free image that is available (you can see this in the template below).

Create the four virtual machines via any means you like. I used the Azure Portal, but you could use a template and the Azure CLI. Here is a template file for the 2 CPU / HDD machine (I won't reproduce all the templates in this document – it would be too lengthy, and they'll be included in the source code uploaded to GitHub). I created them to use a key-based login, so that's what you'll see in the rest of this documentation. You could just as easily use a password credential. It doesn't matter – the key was less typing once the VM was created.

```
{
  "$schema": "http://schema.management.azure.com/schemas/2015-01-
01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "location": {
      "type": "string"
    },
    "virtualMachineName": {
      "type": "string"
    },
    "virtualMachineSize": {
      "type": "string"
    },
    "adminUsername": {
      "type": "string"
    },
    "virtualNetworkName": {
      "type": "string"
    },
    "networkInterfaceName": {
      "type": "string"
    },
    "networkSecurityGroupName": {
      "type": "string"
    },
    "adminPublicKey": {
      "type": "string"
    },
    "diagnosticsStorageAccountName": {
      "type": "string"
    },
    "diagnosticsStorageAccountId": {
      "type": "string"
    },
    "diagnosticsStorageAccountType": {
      "type": "string"
    },
    "addressPrefix": {
      "type": "string"
    },
    "subnetName": {
      "type": "string"
    },
    "subnetPrefix": {
      "type": "string"
    },
    "publicIpAddressName": {
      "type": "string"
    },
    "publicIpAddressType": {
      "type": "string"
    },
    "publicIpAddressSku": {
      "type": "string"
    },
    "autoShutdownStatus": {
      "type": "string"
    },
    "autoShutdownTime": {
      "type": "string"
    },
    "autoShutdownTimeZone": {
      "type": "string"
    },
    "autoShutdownNotificationStatus": {
      "type": "string"
    }
  },
  "variables": {
      "vnetId": "[resourceId('rg-pls-dafp','Microsoft.Network/virtualNetworks',
parameters('virtualNetworkName'))]",
      "subnetRef": "[concat(variables('vnetId'), '/subnets/', parameters('subnetName'))]"
    },
    "resources": [
      {
        "name": "[parameters('virtualMachineName')]",
        "type": "Microsoft.Compute/virtualMachines",
        "apiVersion": "2016-04-30-preview",
        "location": "[parameters('location')]",
        "dependsOn": [
          "[concat('Microsoft.Network/networkInterfaces/', parameters('networkInterfaceName'))]",
          "[concat('Microsoft.Storage/storageAccounts/',
parameters('diagnosticsStorageAccountName'))]"
        ],
        "properties": {
          "osProfile": {
            "computerName": "[parameters('virtualMachineName')]",
            "adminUsername": "[parameters('adminUsername')]",
            "linuxConfiguration": {
              "disablePasswordAuthentication": "true",
              "ssh": {
                "publicKeys": [
                  {
                    "path": "[concat('/home/', parameters('adminUsername'),
'/.ssh/authorized_keys')]",
                    "keyData": "[parameters('adminPublicKey')]"
                  }
                ]
              }
            }
          },
          "hardwareProfile": {
            "vmSize": "[parameters('virtualMachineSize')]"
          },
          "storageProfile": {
            "imageReference": {
              "publisher": "OpenLogic",
              "offer": "CentOS",
              "sku": "7.4",
              "version": "latest"
            },
            "osDisk": {
              "createOption": "fromImage",
              "managedDisk": {
                "storageAccountType": "Standard_LRS"
              }
            },
                "dataDisks": []
            },
            "networkProfile": {
                "networkInterfaces": [
                    {
                        "id": "[resourceId('Microsoft.Network/networkInterfaces',
parameters('networkInterfaceName'))]"
                    }
                ]
            },
            "diagnosticsProfile": {
                "bootDiagnostics": {
                    "enabled": true,
                    "storageUri": "[reference(resourceId('rg-pls-dafp', 'Microsoft.Storage/storageAccounts',
parameters('diagnosticsStorageAccountName')), '2015-06-15').primaryEndpoints['blob']]"
                }
            }
        }
    },
    {
        "name": "[concat('shutdown-computevm-', parameters('virtualMachineName'))]",
        "type": "Microsoft.DevTestLab/schedules",
        "apiVersion": "2017-04-26-preview",
        "location": "[parameters('location')]",
        "properties": {
            "status": "[parameters('autoShutdownStatus')]",
            "taskType": "ComputeVmShutdownTask",
            "dailyRecurrence": {
                "time": "[parameters('autoShutdownTime')]"
            },
            "timeZoneId": "[parameters('autoShutdownTimeZone')]",
            "targetResourceId": "[resourceId('Microsoft.Compute/virtualMachines',
parameters('virtualMachineName'))]",
            "notificationSettings": {
                "status": "[parameters('autoShutdownNotificationStatus')]",
                "timeInMinutes": "30"
            }
        },
        "dependsOn": [
            "[concat('Microsoft.Compute/virtualMachines/', parameters('virtualMachineName'))]"
        ]
    },
    {
        "name": "[parameters('diagnosticsStorageAccountName')]",
        "type": "Microsoft.Storage/storageAccounts",
        "apiVersion": "2015-06-15",
        "location": "[parameters('location')]",
        "properties": {
            "accountType": "[parameters('diagnosticsStorageAccountType')]"
          }
        },
        {
          "name": "[parameters('virtualNetworkName')]",
          "type": "Microsoft.Network/virtualNetworks",
          "apiVersion": "2017-08-01",
          "location": "[parameters('location')]",
          "properties": {
            "addressSpace": {
              "addressPrefixes": [
                "[parameters('addressPrefix')]"
              ]
            },
            "subnets": [
              {
                "name": "[parameters('subnetName')]",
                "properties": {
                  "addressPrefix": "[parameters('subnetPrefix')]"
                }
              }
            ]
          }
        },
        {
          "name": "[parameters('networkInterfaceName')]",
          "type": "Microsoft.Network/networkInterfaces",
          "apiVersion": "2016-09-01",
          "location": "[parameters('location')]",
          "dependsOn": [
            "[concat('Microsoft.Network/virtualNetworks/', parameters('virtualNetworkName'))]",
            "[concat('Microsoft.Network/publicIpAddresses/', parameters('publicIpAddressName'))]",
            "[concat('Microsoft.Network/networkSecurityGroups/',
parameters('networkSecurityGroupName'))]"
          ],
          "properties": {
            "ipConfigurations": [
              {
                "name": "ipconfig1",
                "properties": {
                  "subnet": {
                    "id": "[variables('subnetRef')]"
                  },
                  "privateIPAllocationMethod": "Dynamic",
                  "publicIpAddress": {
                    "id": "[resourceId('rg-pls-dafp','Microsoft.Network/publicIpAddresses',
parameters('publicIpAddressName'))]"
                  }
                }
              }
            ],
            "networkSecurityGroup": {
              "id": "[resourceId('rg-pls-dafp', 'Microsoft.Network/networkSecurityGroups',
parameters('networkSecurityGroupName'))]"
            }
          }
        },
        {
          "name": "[parameters('publicIpAddressName')]",
          "type": "Microsoft.Network/publicIpAddresses",
          "apiVersion": "2017-08-01",
          "location": "[parameters('location')]",
          "properties": {
            "publicIpAllocationMethod": "[parameters('publicIpAddressType')]"
          },
          "sku": {
            "name": "[parameters('publicIpAddressSku')]"
          }
        },
        {
          "name": "[parameters('networkSecurityGroupName')]",
          "type": "Microsoft.Network/networkSecurityGroups",
          "apiVersion": "2017-06-01",
          "location": "[parameters('location')]",
          "properties": {
            "securityRules": [
              {
                "name": "default-allow-ssh",
                "properties": {
                  "priority": 1000,
                  "protocol": "TCP",
                  "access": "Allow",
                  "direction": "Inbound",
                  "sourceAddressPrefix": "*",
                  "sourcePortRange": "*",
                  "destinationAddressPrefix": "*",
                  "destinationPortRange": "22"
                }
              }
            ]
          }
        }
      ],
      "outputs": {
        "adminUsername": {
          "type": "string",
          "value": "[parameters('adminUsername')]"
        }
      }
  }
```

Once you have all four VMs, make sure they're up and running, and that you can SSH onto them

```
$ ssh -i /cygdrive/c/Users/e1vwcco/.ssh/id_rsa dirservd@52.168.122.157
Last login: Mon Feb  5 22:51:36 2018 from cpe-65-189-199-92.neo.res.rr.com
[dirservd@vm-pls-2core-hdd ~]$
```

Installation and Configuration of OpenDJ

You'll need to do this on all four VMs. Another way to do this would be to write the install and config in an automation tool like Ansible and have it provision the needed software for you. I went with the quicker brute force approach for this project.

First, get the OpenDJ software from ForgeRock. There are several ways to do this. You can obtain a free trial version from their website (forgerock.com) or you can pull an earlier version from source and build it yourself (instructions are on the ForgeRock website). I have access to the software already, so I just downloaded it from the site.

For this project, we're using OpenDJ 5.5. You could use any version from 2.6 onwards – they all have the same performance testing tools and sample data generation built in.

Next, copy the OpenDJ binary over to each VM. We'll use SCP to do this, and place it in the /tmp directory on the VM:

```
$ scp -i /cygdrive/c/Users/e1vwcco/.ssh/id_rsa DS-5.5.0.zip
dirservd@52.168.122.157:/tmp
DS-5.5.0.zip
100%   32MB 748.5KB/s   00:43
```

Now SSH onto the VM

```
$ ssh -i /cygdrive/c/Users/e1vwcco/.ssh/id_rsa dirservd@52.168.122.157
Last login: Mon Feb  5 22:51:36 2018 from cpe-65-189-199-92.neo.res.rr.com
[dirservd@vm-pls-2core-hdd ~]$
```

To install OpenDJ, you first need to have Java installed. The version that is easiest to obtain with the least amount of hassle is OpenJDK. You need to SSH onto the VM, as shown above, and then run:
`sudo yum install java-1.8.0-openjdk-devel`

Like so (output is truncated):

```
[dirservd@vm-pls-2core-hdd ~]$ sudo yum install java-1.8.0-openjdk-devel
Loaded plugins: fastestmirror, langpacks
base
| 3.6 kB  00:00:00
extras
| 3.4 kB  00:00:00
openlogic
| 2.9 kB  00:00:00
updates
| 3.4 kB  00:00:00
(1/5): openlogic/7/x86_64/primary_db
|  63 kB  00:00:00
(2/5): base/7/x86_64/group_gz
| 156 kB  00:00:00
```

```
(3/5): extras/7/x86_64/primary_db
| 166 kB  00:00:00
(4/5): base/7/x86_64/primary_db
| 5.7 MB  00:00:00
(5/5): updates/7/x86_64/primary_db
| 6.0 MB  00:00:00
Determining fastest mirrors
Resolving Dependencies
--> Running transaction check
---> Package java-1.8.0-openjdk-devel.x86_64 1:1.8.0.161-0.b14.el7_4 will be installed
…
…
…
Complete!
```

Next, create a directory where OpenDJ will live – we'll use /usr/local/software – change the owner of this directory to the OS user specified in the template and in our SSH login (dirservd)

```
[dirservd@vm-pls-2core-hdd ~]$ cd /usr/local/
[dirservd@vm-pls-2core-hdd local]$ sudo mkdir software
[dirservd@vm-pls-2core-hdd local]$ sudo chown dirservd:dirservd software
```

Unzip OpenDJ 5.5 in /usr/local/software (again, output truncated)

```
[dirservd@vm-pls-2core-hdd software]$ unzip /tmp/DS-5.5.0.zip
Archive:  /tmp/DS-5.5.0.zip
  inflating: opendj/lib/extensions/forgerock-backstage-connect-config.jar
   creating: opendj/bin/
   creating: opendj/bin/ControlPanel.app/
   creating: opendj/bin/ControlPanel.app/Contents/
…
…
…
```

Now navigate to /usr/local/software/opendj and run the following to setup OpenDJ with 50,000 sample users (you'll need to substitute in your own hostname). You should definitely NOT use "Test1234" as a root user password if you were doing this for any kind of permanent installation.

```
./setup directory-server \
          --instancePath /usr/local/software/opendj \
          --rootUserDn cn=Directory\ Manager \
          --rootUserPassword Test1234 \
          --hostname vm-pls-2core-hdd \
          --adminConnectorPort 4444 \
          --ldapPort 1389 \
          --sampleData 50000 \
          --baseDn dc=example,dc=com \
          --backendType je-backend \
          --acceptLicense
```

For example, using the terrible "Test1234" as the Directory Manager password (again, output truncated, specifically the license agreement in this case

```
[dirservd@vm-pls-2core-hdd opendj]$ ./setup directory-server \
>           --instancePath /usr/local/software/opendj \
>           --rootUserDn cn=Directory\ Manager \
>           --rootUserPassword Test1234 \
>           --hostname vm-pls-2core-hdd \
>           --adminConnectorPort 4444 \
>           --ldapPort 1389 \
>           --sampleData 50000 \
>           --baseDn dc=example,dc=com \
>           --backendType je-backend \
>           --acceptLicense


Validating parameters..... Done
Configuring certificates..... Done
Configuring server....... Done
Importing automatically-generated data (50000
entries)....................................................................
.......................... Done
Starting directory server.................................... Done

To see basic server status and configuration, you can launch
/usr/local/software/opendj/bin/status
```

If you go to the /bin directory of your OpenDJ install, we can now get a status for our directory server (notice the 50,000+ entries)

```
[dirservd@vm-pls-2core-hdd bin]$ ./status


>>>> Specify OpenDJ LDAP connection parameters

How do you want to trust the server certificate?

    1)  Automatically trust
    2)  Use a truststore
    3)  Manually validate

Enter choice [3]: 1

Administrator user bind DN [cn=Directory Manager]:

Password for user 'cn=Directory Manager':

          --- Server Status ---
Server Run Status:        Started
Open Connections:         1

          --- Server Details ---
Host Name:                vm-pls-2core-hdd
Administrative Users:     cn=Directory Manager
Installation Path:        /usr/local/software/opendj
Version:                  ForgeRock Directory Services 5.5.0
Java Version:             1.8.0_161
Administration Connector: Port 4444 (LDAPS)

          --- Connection Handlers ---
Address:Port : Protocol : State
-------------:----------:---------
--           : LDIF     : Disabled
0.0.0.0:161  : SNMP     : Disabled
0.0.0.0:1389 : LDAP     : Enabled

          --- Data Sources ---
Base DN:     dc=example,dc=com
Backend ID:  userRoot
Entries:     50002
Replication:
```

One last change before starting the performance test – increase the amount of RAM available to OpenDJ to 2 GB (which our VM can easily support). To do this, edit the following file - /usr/local/software/opendj/config/java.properties and edit the start-ds line so it reads as follows

```
start-ds.java-args=-server -Xms2g -Xmx2g
```

If you're using OpenDJ 5.0 or later, just stop and start OpenDJ so the new memory settings take effect. If not, run bin/dsjavaproperties to enforce them, then stop and start OpenDJ. Stop and start are in the bin subdirectory in OpenDJ. You can see if your changes were correct by watching the startup for the following line

```
[06/Feb/2018:00:56:17 +0000] category=JVM severity=NOTICE msgID=19 msg=JVM
Arguments: "-Xms2g", "-Xmx2g", "-Dorg.opends.server.scriptName=start-ds"
```

You can verify that a performance test will run correctly by trying one now – we'll use the searchrate tool that comes with OpenDJ, in the /bin directory. We'll be aiming for 5,000 requests per second. Here's an example of what one should look like

```
[dirservd@vm-pls-2core-hdd bin]$ ./searchrate -D "cn=Directory Manager" -w Test1234
-b "dc=example,dc=com" -F -M 5000 -c 4 -t 4 -p 1389 -g "rand(0,2000)"
"(uid=user.%d)"
-------------------------------------------------------------------------------------
--------
|     Throughput    |                 Response Time                |
Additional      |
|    (ops/second)   |                (milliseconds)                |
Statistics      |
|   recent  average |   recent  average    99.9%   99.99%  99.999% |  err/sec
Entries/Srch |
-------------------------------------------------------------------------------------
--------
|    200.0    200.0 |   75.855   75.855   270.53   293.60   293.60 |      0.0
1.1 |
|    368.4    284.2 |   43.352   54.789   263.19   293.60   293.60 |      0.0
1.0 |
|    692.2    420.2 |   23.090   37.378   259.00   270.53   293.60 |      0.0
1.0 |
|    804.2    516.2 |   19.731   30.505   250.61   270.53   293.60 |      0.0
1.0 |
|    776.0    568.1 |   20.505   27.775   238.03   270.53   293.60 |      0.0
1.0 |
|    932.4    628.8 |   17.093   25.136   233.83   270.53   293.60 |      0.0
1.0 |
|    834.7    658.2 |   19.083   24.040   230.69   270.53   293.60 |      0.0
1.0 |
|    990.0    699.7 |   16.146   22.643   230.69   263.19   293.60 |      0.0
1.0 |
```

Deployment of Custom Java Code

The code that I wrote as part of this project will be more fully described below. For now, just know that you'll have two JAR files you need to move over to the VMs – one is the custom code, and another is a third party library

Take the FileTalier jar file, plus the json-20090211.jar and move them to your VMs using SCP

```
$ scp -i /cygdrive/c/Users/e1vwcco/.ssh/id_rsa *.jar dirservd@52.186.28.79:/tmp
FileTailer-0.0.1-SNAPSHOT.jar
100% 6193   111.6KB/s   00:00
json-20090211.jar
100%   45KB 559.4KB/s   00:00
```

You can validate that you've placed the code correctly by trying to run it without any arguments – it will tell you how it should be used if things are set up

```
[dirservd@vm-pls-2core-hdd bin]$ java -cp "/tmp/json-20090211.jar:/tmp/FileTailer-
0.0.1-SNAPSHOT.jar" LogReader
Usage: LogReader
               -fileName <file to monitor for searchrate results>
               -transmit <Y or N to indicate whether to attempt to transmit data to
Log Analytics>
               -description <label to identify this machine / run in Log Analytics
               -logName <the name of the custom log in Log Analytics
               -customerId <Log Analytics customer ID - obtained on the Azure
portal>
```

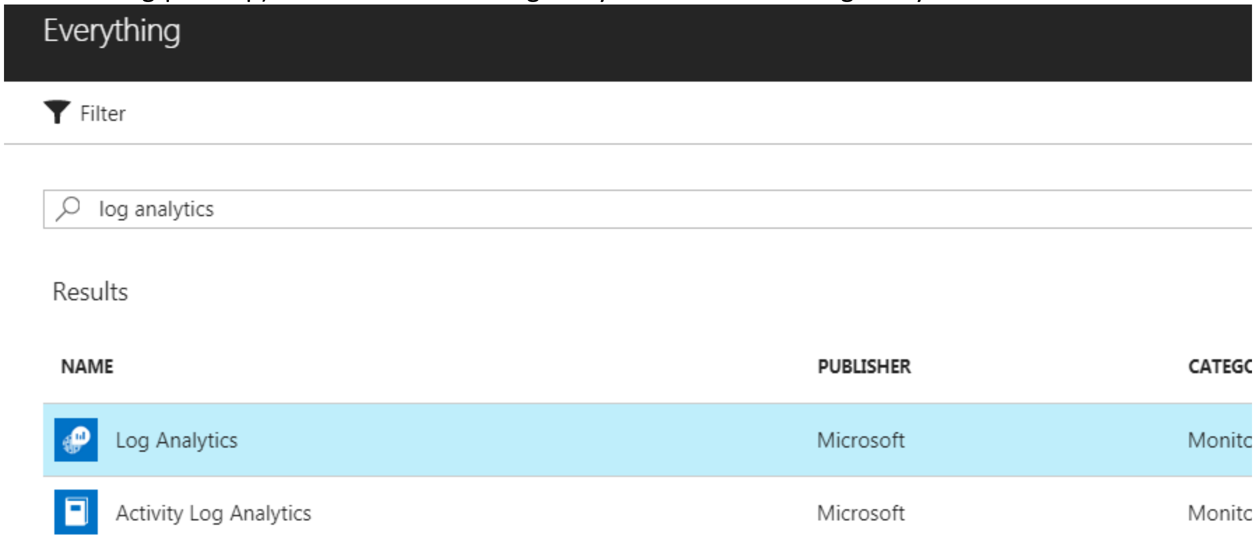
```
               -sharedKey <key required for transmission - also acquired from Log
Analytics in Azure portal>
               -verbose [if Y then print debug output]
```

Notice that there is a transmit flag and a verbose flag if you should need to debug what it is doing. In other words, if you run into an error, turn on verbose, and possibly shut off transmit to see what the issue is.

The code is now in place and ready for when we'll run our performance tests (which is later).

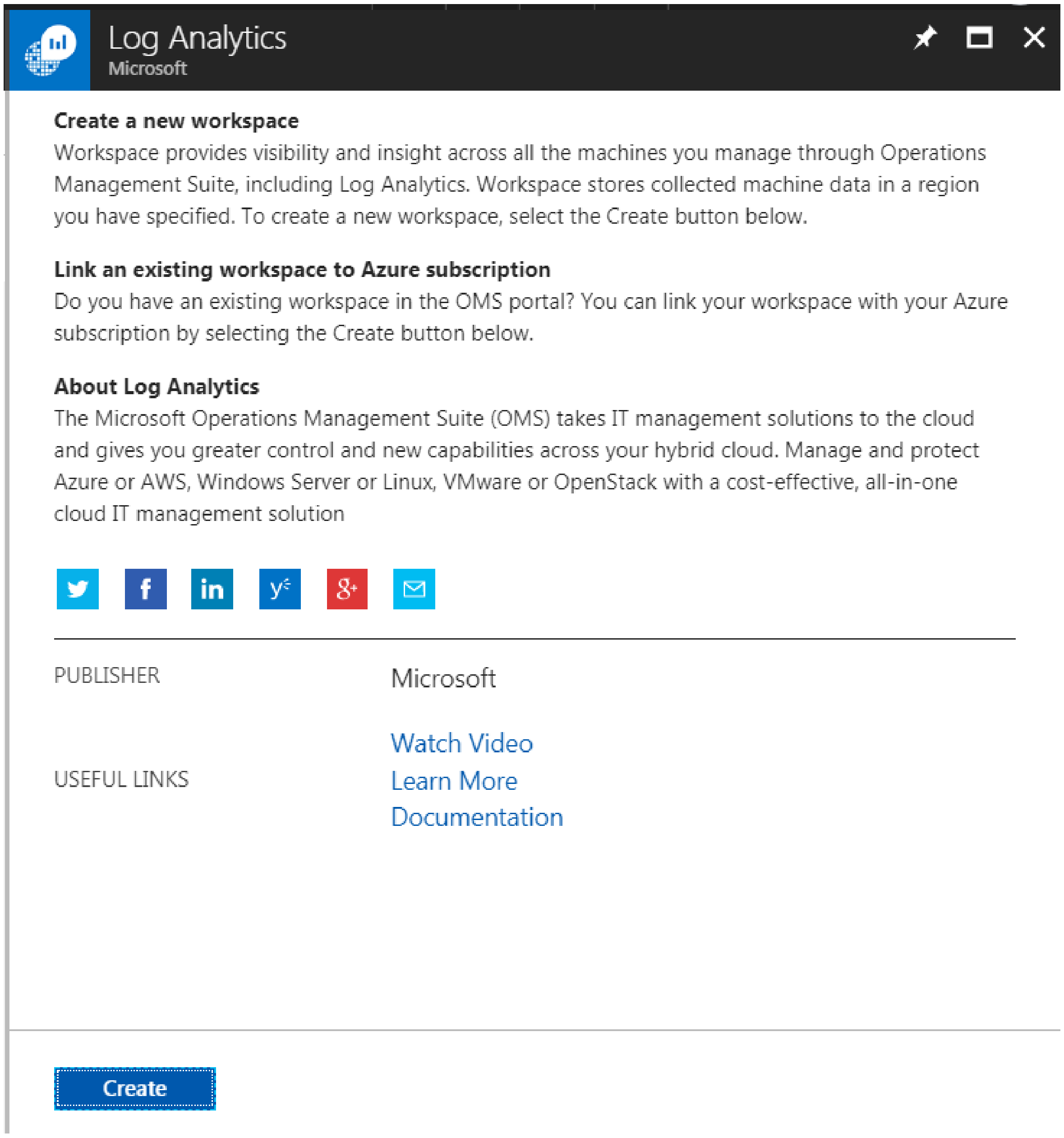
Creation of Log Analytics and OMS related resources in Azure

Now we'll need to add Log Analytics to our resource group. Go into your resource group (remember, this one is rg-pls-dafp) select Add. Enter "log analytics" and choose Log Analytics from the list.



NAME	PUBLISHER	CATEGO
Log Analytics	Microsoft	Monitc
Activity Log Analytics	Microsoft	Monitc

Then choose Create



Log Analytics
Microsoft

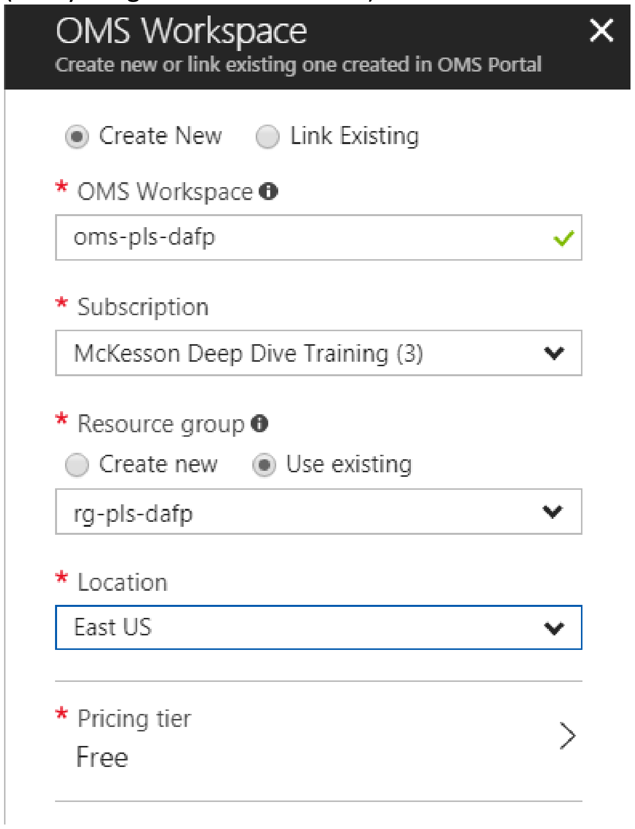
Create a new workspace
Workspace provides visibility and insight across all the machines you manage through Operations Management Suite, including Log Analytics. Workspace stores collected machine data in a region you have specified. To create a new workspace, select the Create button below.

Link an existing workspace to Azure subscription
Do you have an existing workspace in the OMS portal? You can link your workspace with your Azure subscription by selecting the Create button below.

About Log Analytics
The Microsoft Operations Management Suite (OMS) takes IT management solutions to the cloud and gives you greater control and new capabilities across your hybrid cloud. Manage and protect Azure or AWS, Windows Server or Linux, VMware or OpenStack with a cost-effective, all-in-one cloud IT management solution

PUBLISHER	Microsoft
USEFUL LINKS	Watch Video Learn More Documentation

Create a new OMS Workspace, place it into your resource group and select the appropriate location (everything I have in is East US). I named mine to match the RG name – oms-pls-dafp.


OMS Workspace
Create new or link existing one created in OMS Portal

[✓ Create New] Link Existing

* OMS Workspace: oms-pls-dafp

* Subscription: McKesson Deep Dive Training (3)

* Resource group: Create new [✓ Use existing]
rg-pls-dafp

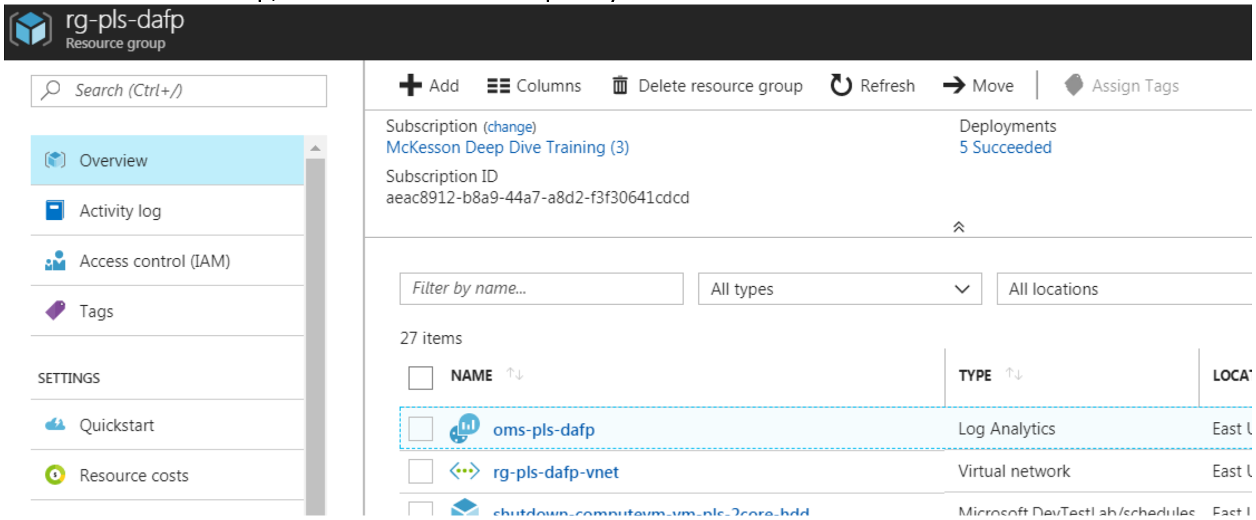
* Location: East US

* Pricing tier: Free

For simplicity, I chose the Free pricing tier.

Next, take each of the four VMs you created and link them to the new OMS workspace you've created.

In the Resource Group, select the OMS workspace you've created

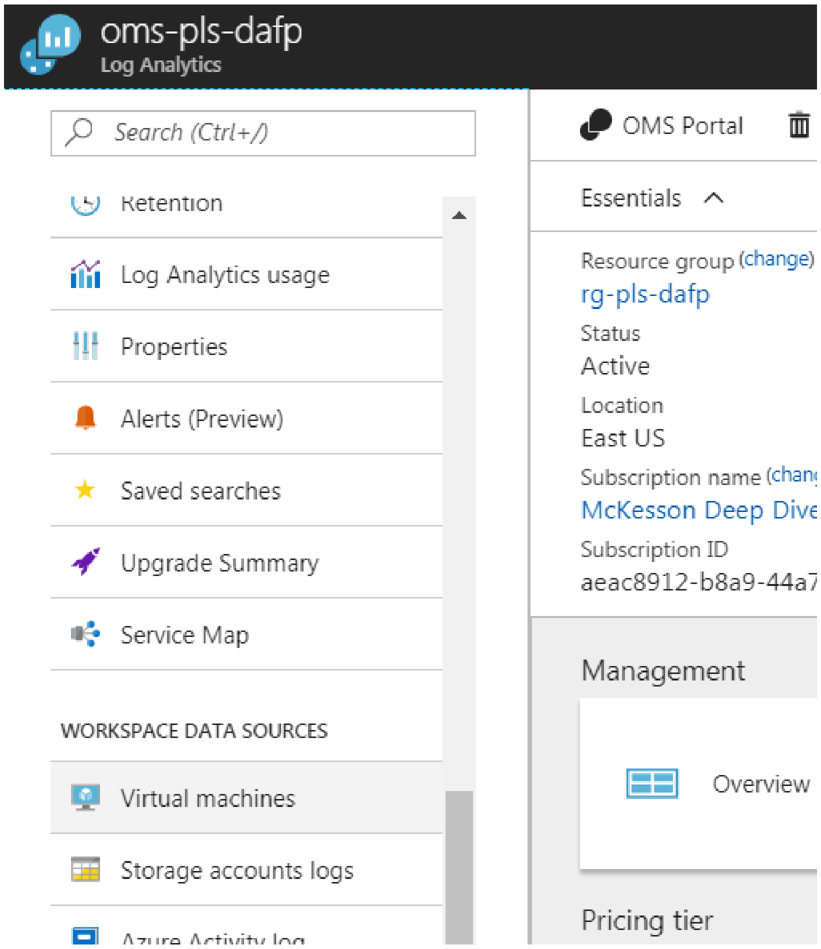

rg-pls-dafp
Resource group

Subscription (change): McKesson Deep Dive Training (3)
Subscription ID: aeac8912-b8a9-44a7-a8d2-f3f30641cdcd
Deployments: 5 Succeeded

27 items

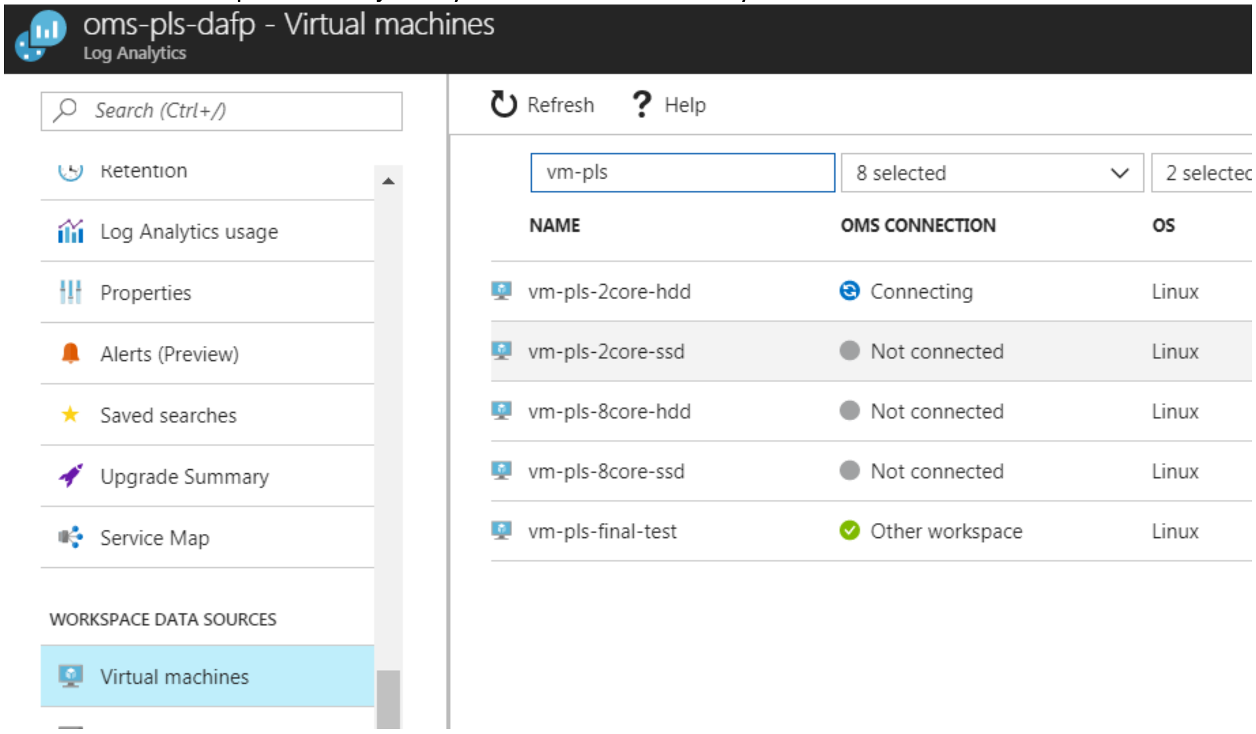
NAME	TYPE	LOCA
oms-pls-dafp	Log Analytics	East U
rg-pls-dafp-vnet	Virtual network	East U
shutdown-computevm-vm-pls-2core-hdd	Microsoft.DevTestLab/schedules	East U

Scroll down on the list and choose Virtual Machines



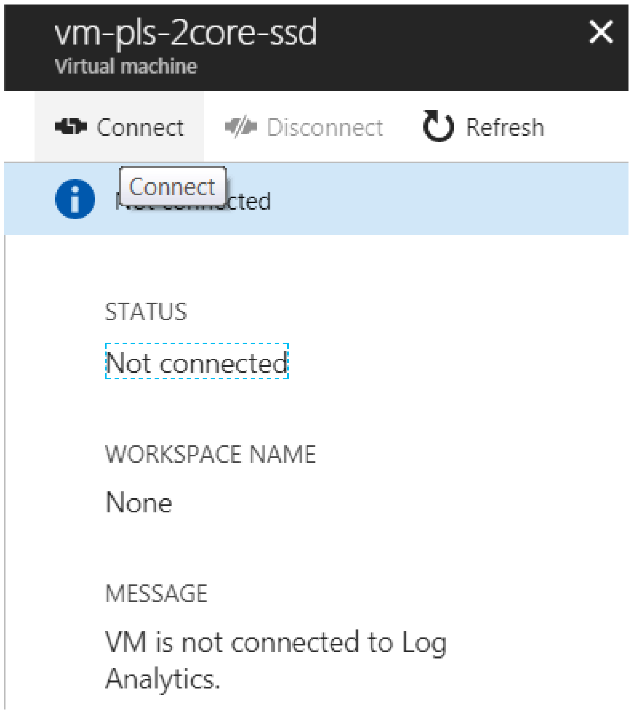
Resource group (change)
rg-pls-dafp
Status
Active
Location
East US
Subscription name (chan
McKesson Deep Dive
Subscription ID
aeac8912-b8a9-44a7

I filtered the subsequent list to just my VMs – choose the VM you want to connect



NAME	OMS CONNECTION	OS
vm-pls-2core-hdd	Connecting	Linux
vm-pls-2core-ssd	Not connected	Linux
vm-pls-8core-hdd	Not connected	Linux
vm-pls-8core-ssd	Not connected	Linux
vm-pls-final-test	Other workspace	Linux

Click on the Connect button

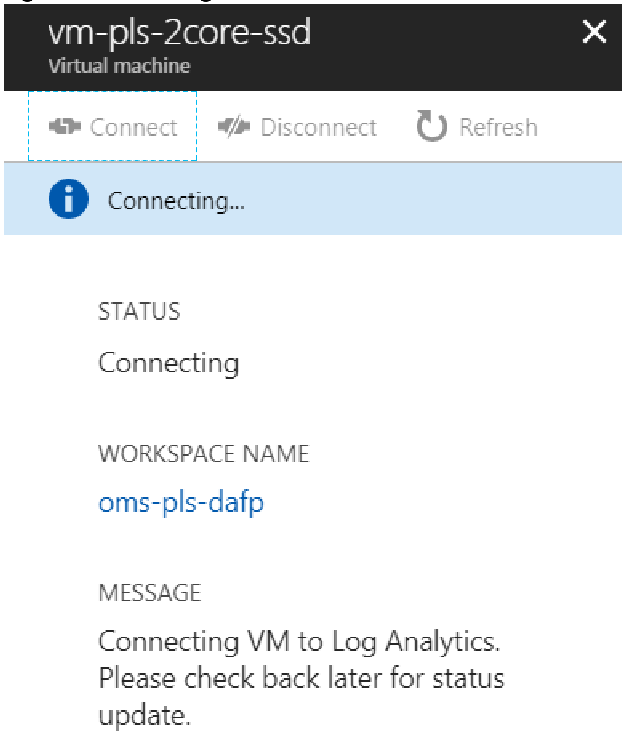


STATUS
Not connected

WORKSPACE NAME
None

MESSAGE
VM is not connected to Log Analytics.

You'll get a message saying that it is connecting. What it is doing at this point is installing the OMS Agent on the target VM


vm-pls-2core-ssd
Virtual machine

Connecting...

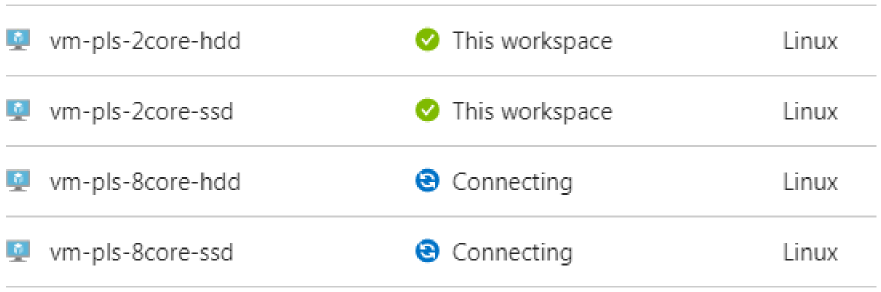
STATUS
Connecting

WORKSPACE NAME
oms-pls-dafp

MESSAGE
Connecting VM to Log Analytics. Please check back later for status update.

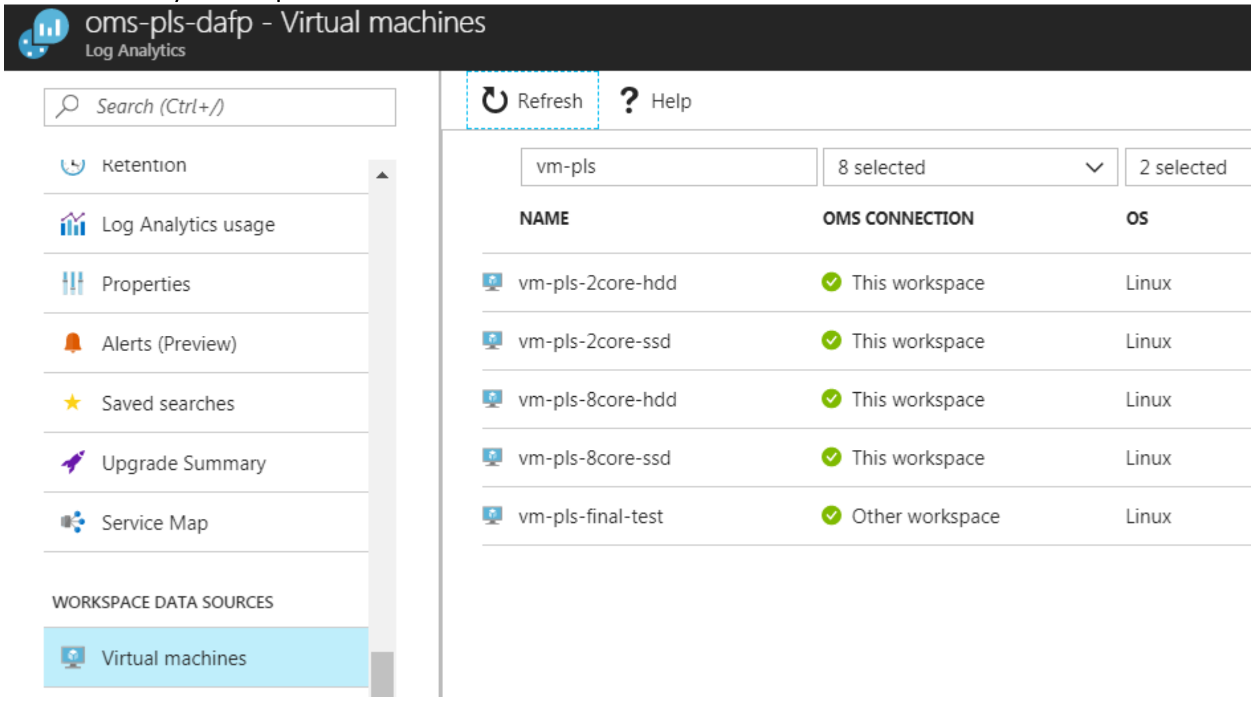
Do this for all four VMs. When this is complete, you'll be able to look at the Virtual Machines tab of the OMS blade and see that they're all connected to "This workspace".

Here's what it looks like when they're in the process of connecting:



vm-pls-2core-hdd	✓ This workspace	Linux
vm-pls-2core-ssd	✓ This workspace	Linux
vm-pls-8core-hdd	Connecting	Linux
vm-pls-8core-ssd	Connecting	Linux

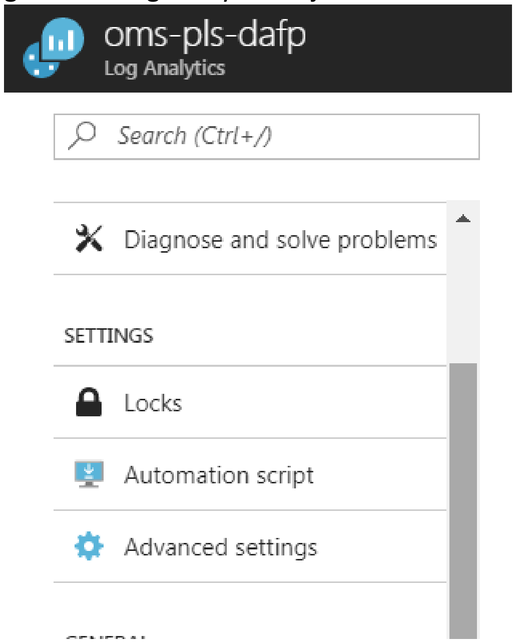
And when they're complete:



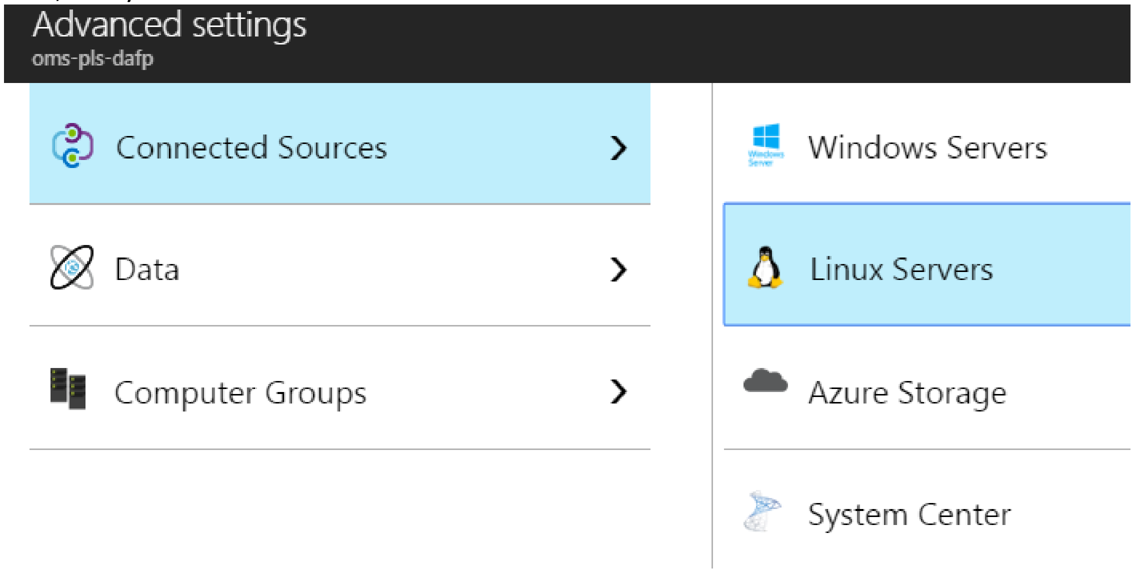
NAME	OMS CONNECTION	OS
vm-pls-2core-hdd	This workspace	Linux
vm-pls-2core-ssd	This workspace	Linux
vm-pls-8core-hdd	This workspace	Linux
vm-pls-8core-ssd	This workspace	Linux
vm-pls-final-test	Other workspace	Linux

Next, we're going to build a dashboard to show OS data like CPU and disk utilization.

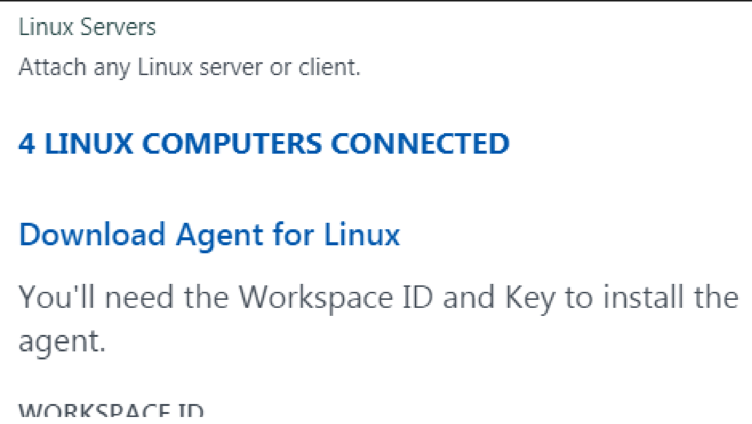
We need to tell the OMS agent to start collecting Linux Performance Counters. In the resource group, go to the Log Analytics object and choose Advanced Settings



First, verify that the four VMs are connected. Choose Connected Sources and then Linux Servers


Advanced settings
oms-pls-dafp

Then verify that the panel that pops up says there are 4 linux computers connected


Linux Servers
Attach any Linux server or client.

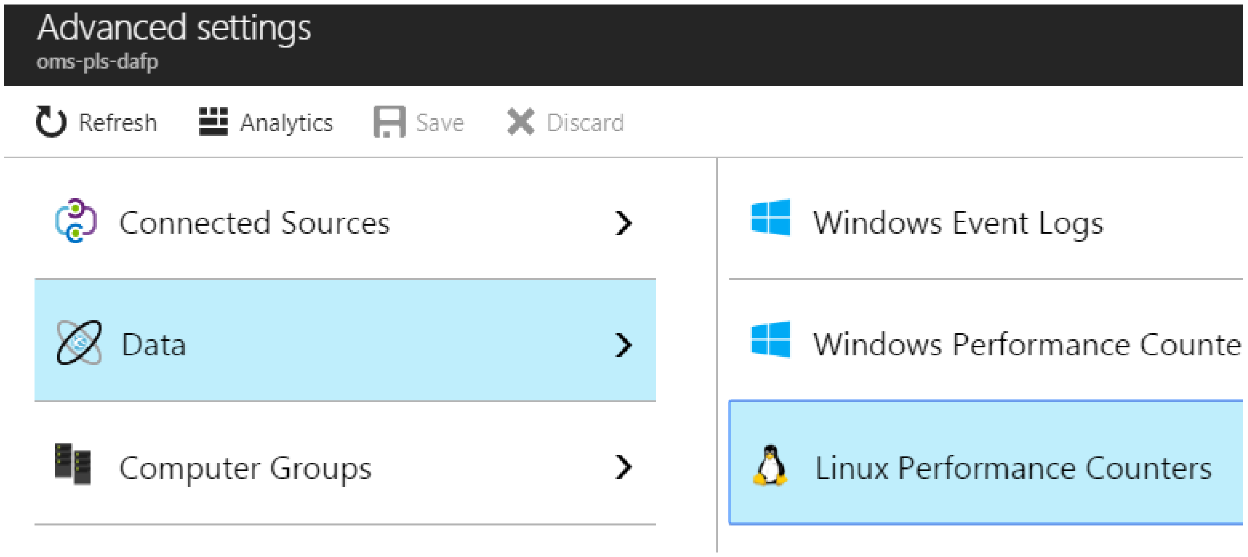
4 LINUX COMPUTERS CONNECTED

Download Agent for Linux

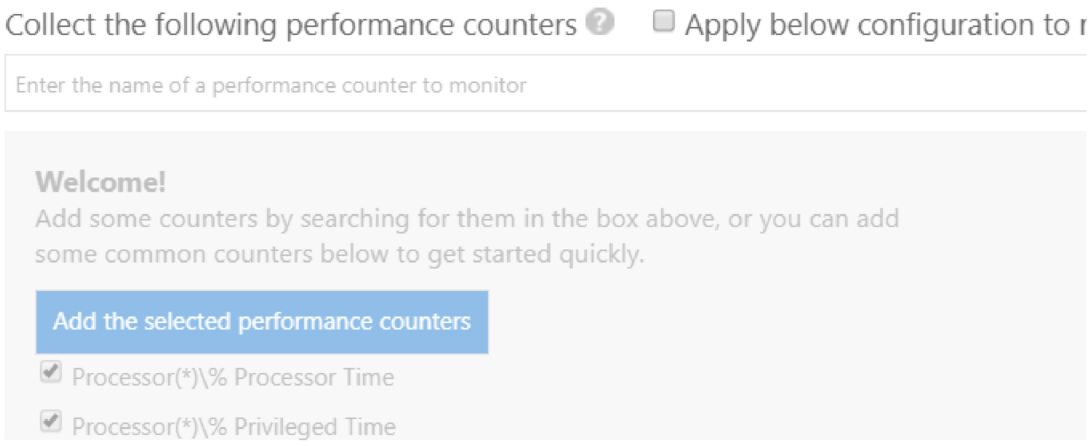
You'll need the Workspace ID and Key to install the agent.

WORKSPACE ID

Now click on Data and then Linux Performance Counters


Advanced settings
oms-pls-dafp

Then click on the “Apply below” checkbox, followed by the blue “Add the selected performance counters” button



Collect the following performance counters ☐ Apply below configuration to r

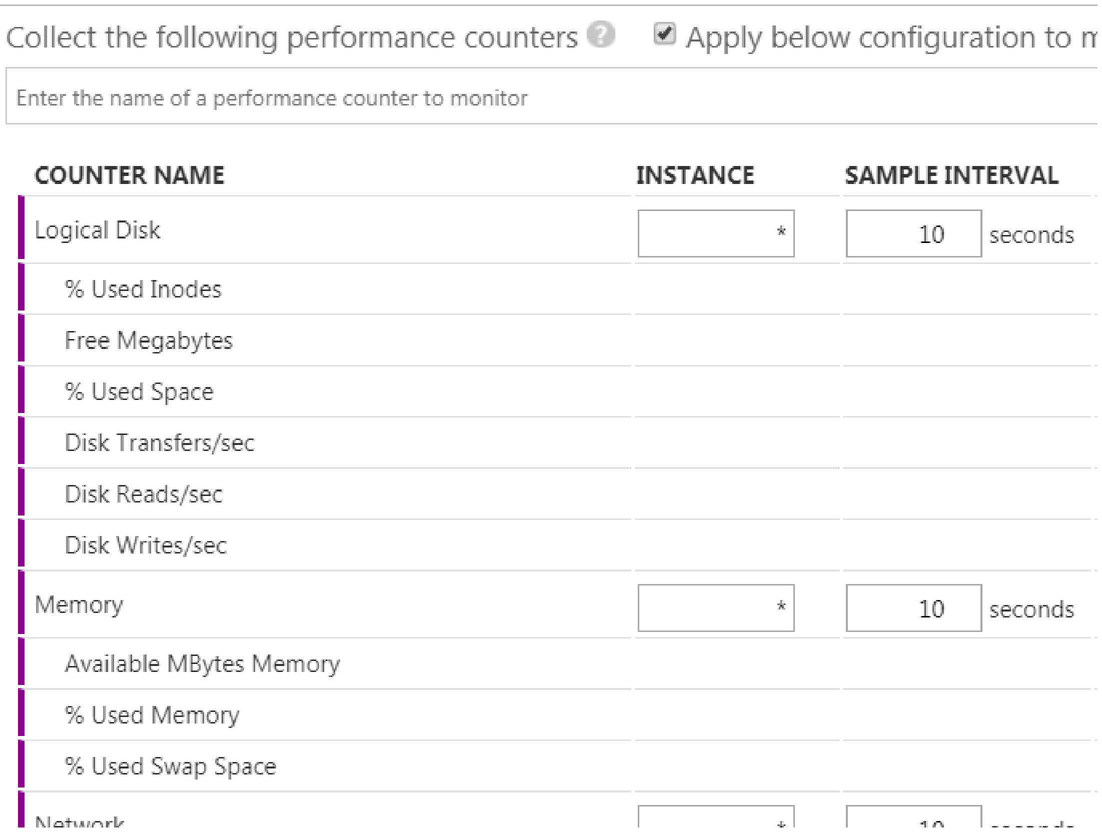
Enter the name of a performance counter to monitor

Welcome!
Add some counters by searching for them in the box above, or you can add some common counters below to get started quickly.

✓ Processor(*)\% Processor Time

✓ Processor(*)\% Privileged Time

The screen will look like this

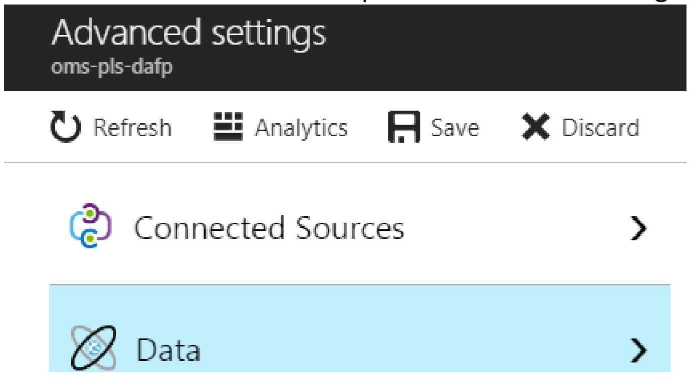


Collect the following performance counters ✓ Apply below configuration to m

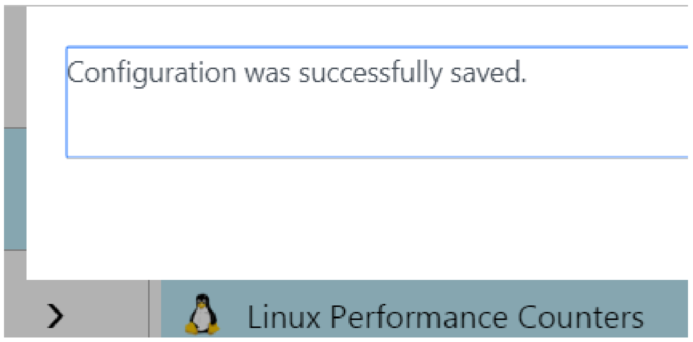
Enter the name of a performance counter to monitor

COUNTER NAME	INSTANCE	SAMPLE INTERVAL
Logical Disk	*	10 seconds
% Used Inodes		
Free Megabytes		
% Used Space		
Disk Transfers/sec		
Disk Reads/sec		
Disk Writes/sec		
Memory	*	10 seconds
Available MBytes Memory		
% Used Memory		
% Used Swap Space		
Network	*	10 seconds

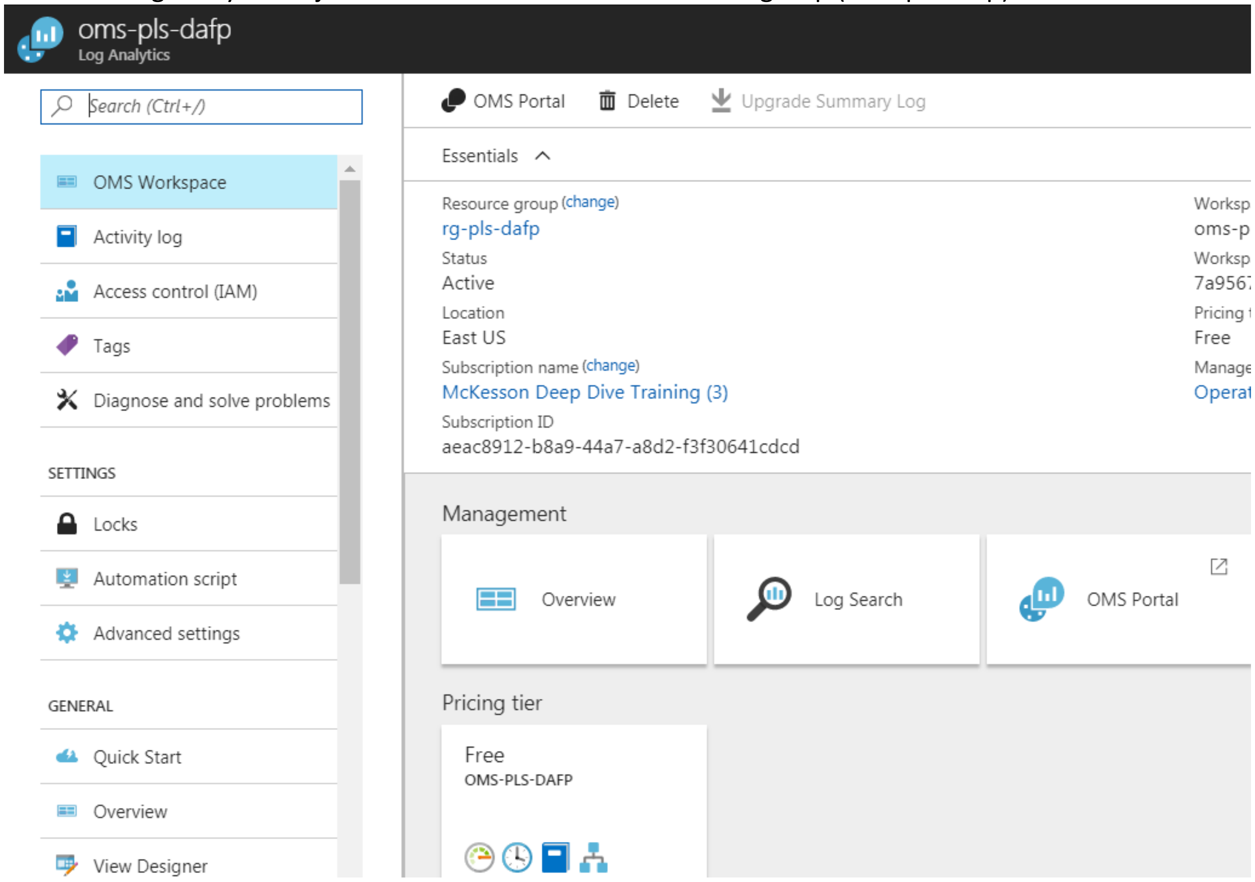
Then click on Save near the top of the Advanced Settings blade



You'll get a message that the configuration was successfully saved.


Configuration was successfully saved.

Go to the Log Analytics object that was created in the resource group (oms-pls-dafp)

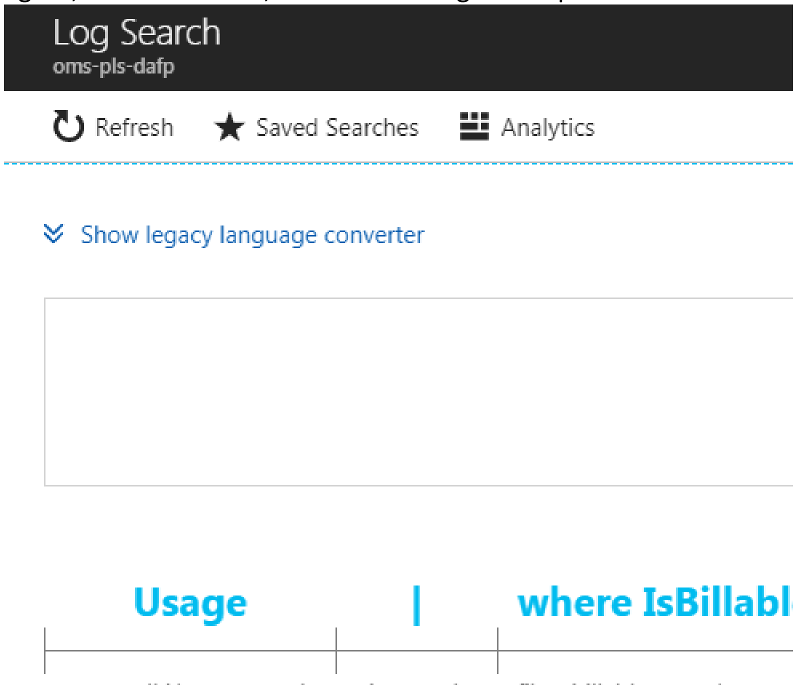


Resource group (change)
rg-pls-dafp
Status
Active
Location
East US
Subscription name (change)
McKesson Deep Dive Training (3)
Subscription ID
aeac8912-b8a9-44a7-a8d2-f3f30641cdcd

Worksp
oms-p
Worksp
7a9567
Pricing t
Free
Manage
Operat

Free
OMS-PLS-DAFP

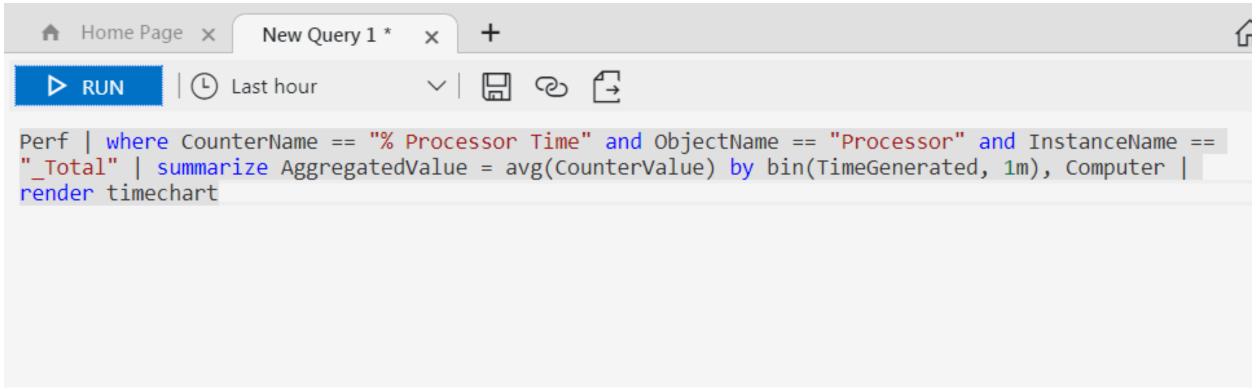
Click on the Log Search – from here you can run a query against the metrics being recorded by the OMS agent, but in this case, we'll be running some queries to build a dashboard. Click on Analytics



In the Analytics portal, we'll enter a query (as an example) that shows the processor utilization on each of the four VMs. This is the query

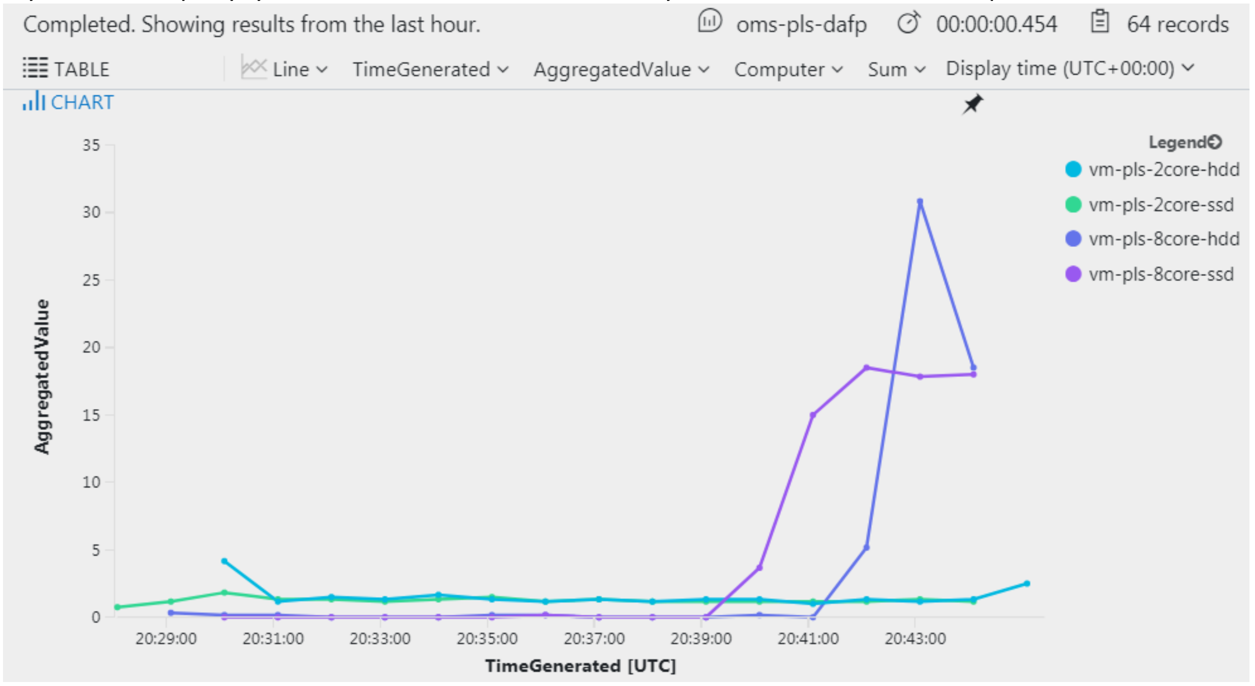
```
Perf | where CounterName == "% Processor Time" and ObjectName == "Processor"
and InstanceName == "_Total" | summarize AggregatedValue = avg(CounterValue) by
bin(TimeGenerated, 1m), Computer | render timechart
```

From the home page tab of the Analytics portal, hit the New Tab plus sign and paste in the query. It'll look like this



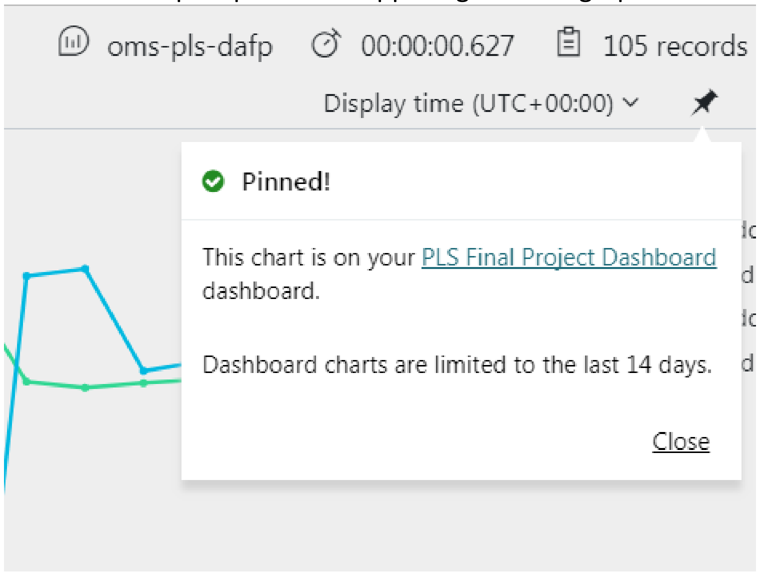
```
Perf | where CounterName == "% Processor Time" and ObjectName == "Processor" and InstanceName ==
"_Total" | summarize AggregatedValue = avg(CounterValue) by bin(TimeGenerated, 1m), Computer |
render timechart
```

If you run the query, you'll see results at the bottom of your screen start to show up



Completed. Showing results from the last hour. oms-pls-dafp 00:00:00.454 64 records

Then click the pushpin on the upper right of the graph itself and pin the chart to your new dashboard



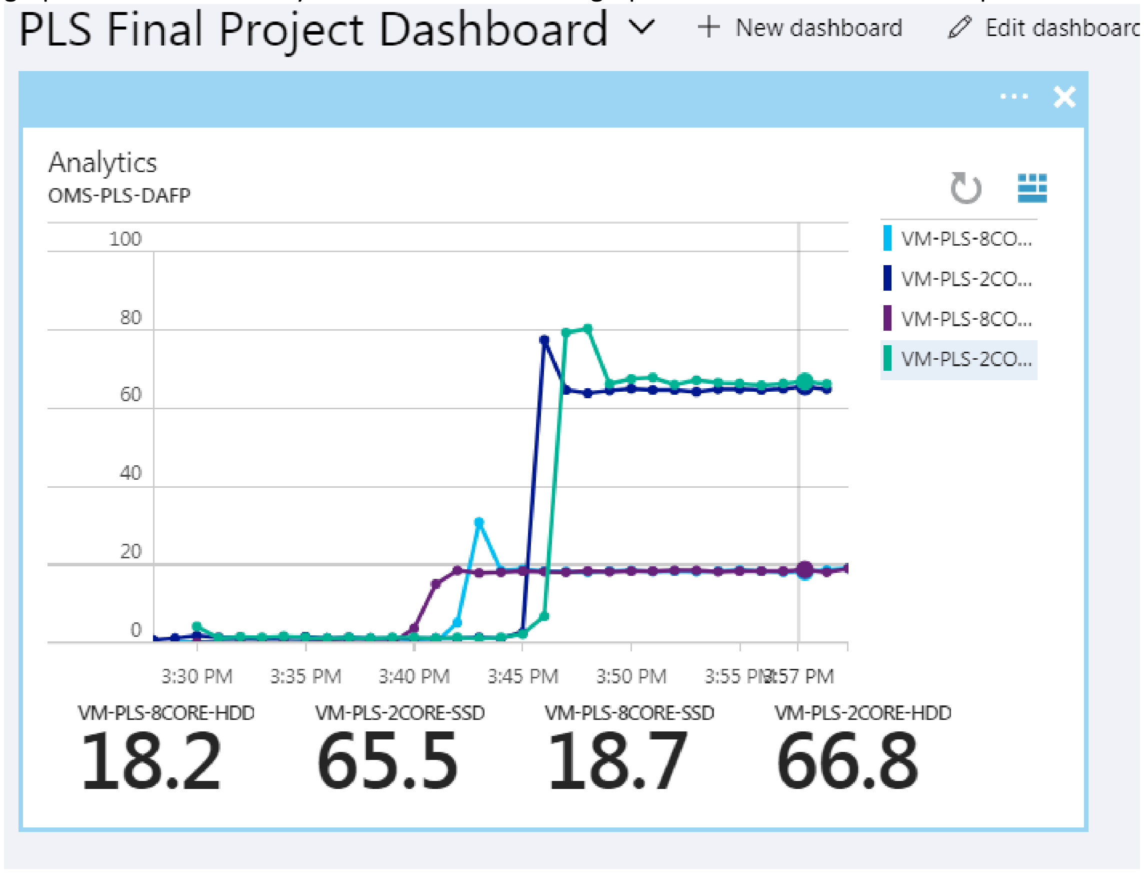
Pinned!

This chart is on your PLS Final Project Dashboard dashboard.

Dashboard charts are limited to the last 14 days.

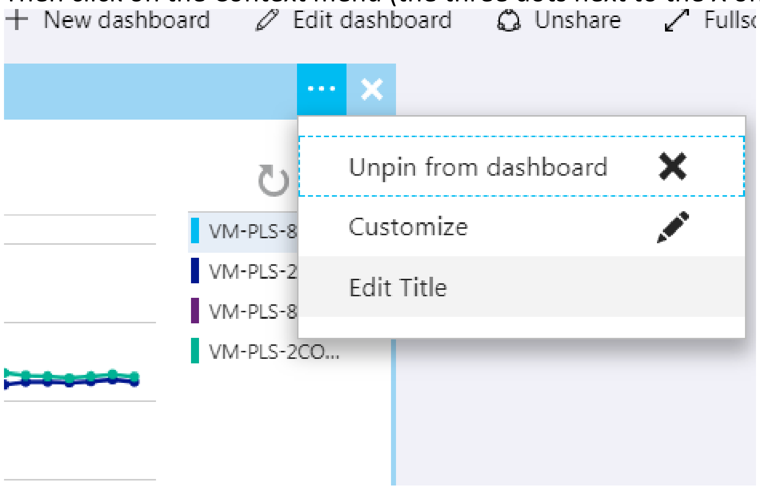
Close

Then, back in the Azure portal, go to Dashboards and choose your dashboard. So far, it has just one graph on it. Notice that you can mouse-over the graph and show the data at that point

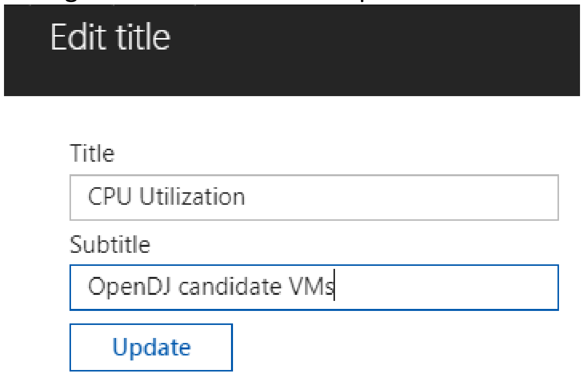


VM-PLS-8CORE-HDD	VM-PLS-2CORE-SSD	VM-PLS-8CORE-SSD	VM-PLS-2CORE-HDD
18.2	65.5	18.7	66.8

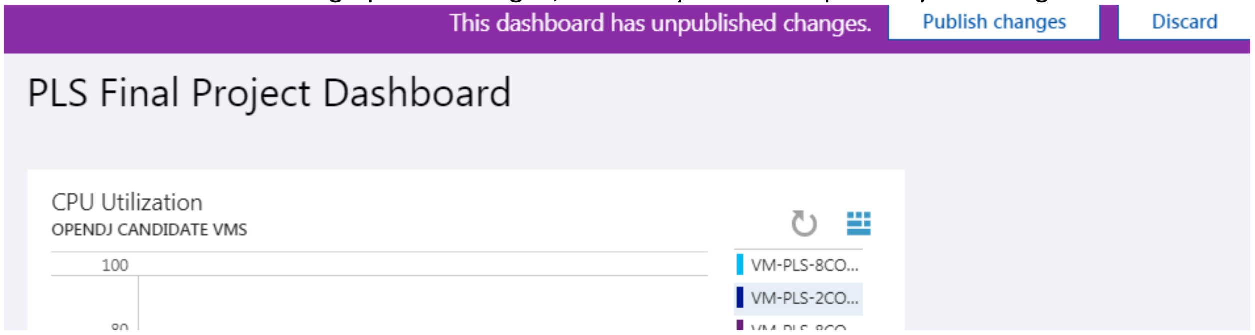
Then click on the Context menu (the three dots next to the X on the graph) and choose Edit Title



Change the title and choose update



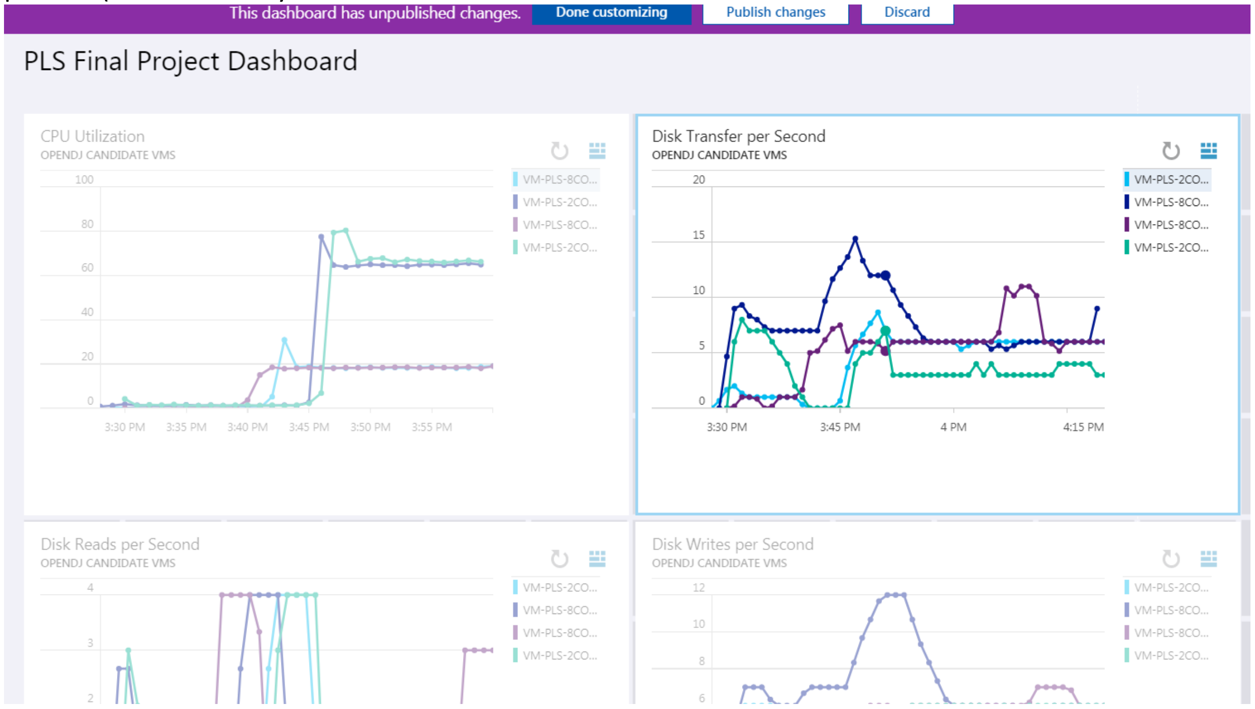
Notice that the title on the graph has changed, and that you need to publish your changes



Likewise, go through and alter the query, pinning to the dashboard, for each of the following counters:

Disk Writes/sec Disk Reads/sec Disk Transfers/sec

After you have all four graphs in place, with their titles edited, go to your dashboard and choose Edit Dashboard (it is to the right of the title of the dashboard). Rearrange the graphs so they are in a square pattern (to make it easy to see all four at once.



Make sure to publish your changes

Custom Java Code

Next we need to talk about the custom java code that was written to capture the performance testing results. The data that is produced by the OpenDJ performance test simply records to the command prompt on standard out. When we run the performance test, we'll output the results to a file instead. Then we'll run Java code that does three things:

1. Watches the file where the searchrate tool is sending the output
2. Transforms the raw data into a valid JSON object that LogAnalytics will understand
3. Transmits the results to LogAnalytics using the DataCollector API

The whole purpose of the code is to take performance test results that look like this:

```
|     200.0     200.0 |    75.855    75.855    270.53    293.60    293.60 |
```

And change them into a format that looks like this so they can be submitted to Log Analytics

```
{"response_average":31,"reponse_999":143.65,"response_recent":31,"throughput_recent"
:483,"throughput_average":483,"machine_name":"2cpuhdd"}
```

And it needs to do this in real time as the test is running.

Here are code snippets showing each of those sections. Full source code is available in the GitHub repo, or at the end of this document.

Watch the output file – this wakes up every 3 seconds to read the file where we're sending the searchrate output

```
canBreak = false;
String line;
try {
    LineNumberReader lnr = new LineNumberReader(new FileReader(filename));
    while (!canBreak)
    {
        line = lnr.readLine();
        if (line == null) {
            if (verbose.equals("Y")) System.out.println("waiting");
            Thread.sleep(3000);
            continue;
        }
        processLine(line);
    }
    lnr.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

Transform the raw data into JSON – notice that the numbers in the switch / case statement correspond to the columns in the searchrate output. I'm sure there is a more elegant way to do this, but it wasn't necessary for what this project was trying to do.

```
JSONObject jo = new JSONObject();
if (verbose.equals("Y")) System.out.println("line = " + s);
StringTokenizer st = new StringTokenizer(s," ");
int colNum = 1;
while (st.hasMoreTokens()) {
        String value = st.nextToken();
        if (value.indexOf("|") < 0) {
                // this is not a column break - it is a number
                Double dVal = new Double(value);
                switch (colNum) {
                case 1:
                        jo.put("throughput_recent", dVal);
                case 2:
                        jo.put("throughput_average", dVal);
                case 3:
                        jo.put("response_recent",dVal);
                case 4:
                        jo.put("response_average", dVal);
                case 5:
                        jo.put("reponse_999", dVal);
                }
                colNum++;
        }
}
jo.put("machine_name", description);
if (transmit.equals("Y")) {
        if (verbose.equals("Y")) System.out.println("posting " + jo.toString());
                postMessage(jo.toString());
        }
        else {
            if (verbose.equals("Y")) System.out.println("DEBUG " +
jo.toString());
        }
}
```

Finally, transmit the results to LogAnalytics using the Data Collector API

```
String Signature = "";
String encodedHash = "";
String url = "";

// Date object
Date date = new Date();

// Todays date input for OMS Log Analytics
SimpleDateFormat sdf = new SimpleDateFormat("E, dd MMM YYYY HH:mm:ss zzz");
sdf.setTimeZone(TimeZone.getTimeZone("GMT"));
String timeNow = sdf.format(date);

// String for signing the key
String stringToSign="POST\n" + json.length() + "\napplication/json\nx-ms-
date:"+timeNow+"\n/api/logs";

try {
        byte[] decodedBytes = Base64.getDecoder().decode(sharedKey);

        Mac hasher = Mac.getInstance("HmacSHA256");
        hasher.init(new SecretKeySpec(decodedBytes, "HmacSHA256"));
        byte[] hash = hasher.doFinal(stringToSign.getBytes());

        encodedHash = DatatypeConverter.printBase64Binary(hash);
        Signature = "SharedKey " + customerId + ":" + encodedHash;

        url = "https://" + customerId + ".ods.opinsights.azure.com/api/logs?api-
version=2016-04-01";
        URL objUrl = new URL(url);
        HttpsURLConnection con = (HttpsURLConnection) objUrl.openConnection();
        con.setDoOutput(true);
        con.setRequestMethod("POST");
        con.setRequestProperty("Content-Type", "application/json");
        con.setRequestProperty("Log-Type",logName);
        con.setRequestProperty("x-ms-date", timeNow);
        con.setRequestProperty("Authorization", Signature);

        DataOutputStream wr = new DataOutputStream(con.getOutputStream());
        wr.writeBytes(json);
        wr.flush();
        wr.close();

        int responseCode = con.getResponseCode();
        if (verbose.equals("Y")) {
                System.out.println("\nSending 'POST' request to URL : " + url);
                System.out.println("Post parameters : " + json);
                System.out.println("Response Code : " + responseCode);
        }
}
catch (Exception e) {
      System.out.println("Catch statement: " + e);
}
```

You'll see in the GitHub repo that I used Maven to manage my dependencies (specifically the external JSON library), but that isn't necessary. If you find and download the JSON jar file (json-20090211.jar), you can add it to the classpath of your project in your IDE of choice and the code will compile. Here's the Maven snippet if you want to use that:

```
  <dependencies>
      <dependency>
        <groupId>org.json</groupId>
        <artifactId>json</artifactId>
        <version>20090211</version>
        <scope>runtime</scope>
      </dependency>
  </dependencies>
```

I also used Maven to produce a jar file for me – that wasn't necessary either. All you'd really need is just the compiled code and the json-20090211.jar file and you'd be good to go.

At this point, you should have either a jar file produced by Maven, or just a compiled Java class (mine is called LogReader). You can take these artifacts and copy them over to the VMs according to the steps shown above in Deployment of Custom Java Code.

I had several sources I based my source code on. All of them were adapted into the complete code that is in the repo.

First, for continuously monitoring a file:
https://stackoverflow.com/questions/21037994/continuous-file-reading

Second, for writing Java code that will call a REST-based API
http://crunchify.com/create-very-simple-jersey-rest-service-and-send-json-data-from-java-client/

Finally, for fulfilling the Data Collector API requirements for the shared key and transmitting it
http://systemcenterme.com/send-data-to-oms-log-analytics-with-java-code/

The official Microsoft documentation on the Data Collector API was also useful
https://docs.microsoft.com/en-us/azure/log-analytics/log-analytics-data-collector-api

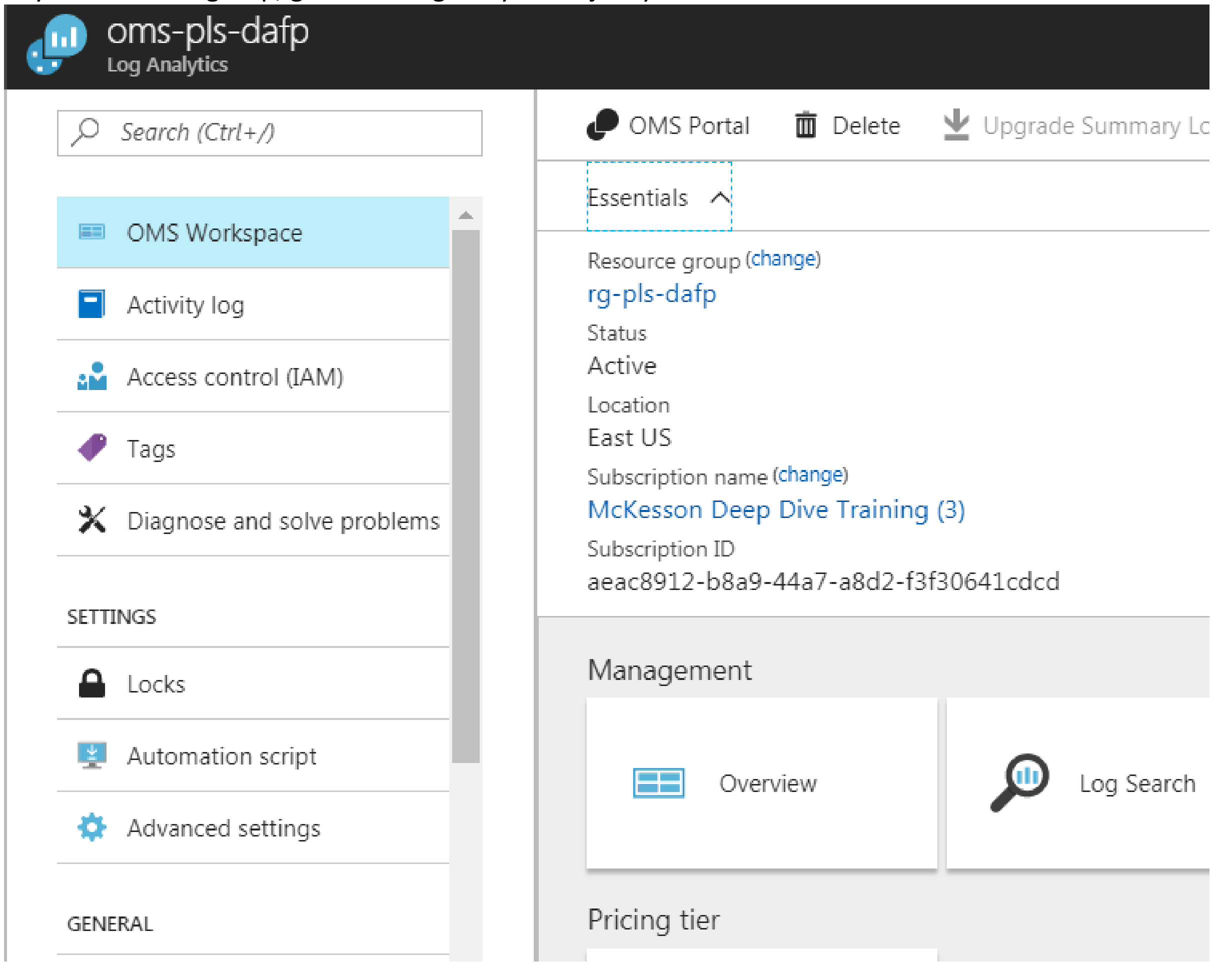
Running the Performance Test and Collecting Data

Now we have all the pieces in place:

- OpenDJ installed and running on each VM
- Custom java code deployed onto each VM
- LogAnalytics hooked up to each VM to collect OS data

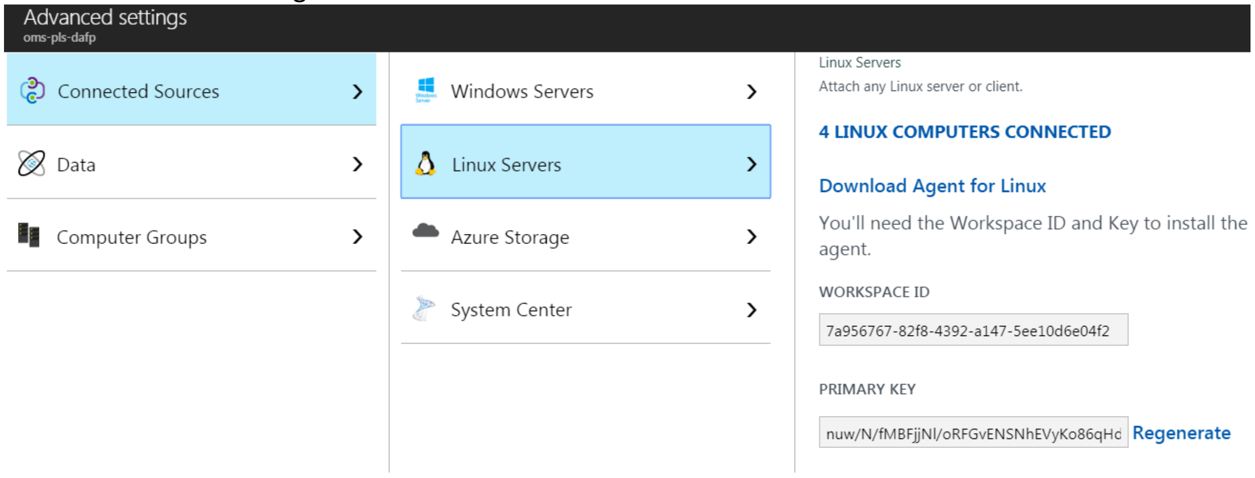
The first thing you'll need to do is collect the customer ID and shared key from the Azure portal – you'll need them when running the test, so the Java code can submit results to the Data Collector API. Here's how to get them.

In your resource group, go to the LogAnalytics object you've created



Resource group (change)
rg-pls-dafp
Status
Active
Location
East US
Subscription name (change)
McKesson Deep Dive Training (3)
Subscription ID
aeac8912-b8a9-44a7-a8d2-f3f30641cdcd

Choose Advanced Settings and click on Linux Servers



Linux Servers
Attach any Linux server or client.

4 LINUX COMPUTERS CONNECTED

Download Agent for Linux

You'll need the Workspace ID and Key to install the agent.

WORKSPACE ID
7a956767-82f8-4392-a147-5ee10d6e04f2

PRIMARY KEY
nuw/N/fMBFjjNI/oRFGvENSNhEVyKo86qHd

The customer ID is the WORKSPACE ID and the shared key is the PRIMARY KEY. Collect each of those and save them – they will be arguments provided to your custom Java code.

So we're ready to run our test!

We'll use the following strategy:

1. Pre-seed a "results" file so we can start our Java code ahead of time (we could improve the Java code so this would not be necessary)
2. Start the Java code as a background process
3. Start the OpenDJ searchrate tool and send the output to the "results" file

Pre-seed a "results" file

This is simple – just use vi or even echo anything you like into /tmp/results.txt

```
[dirservd@vm-pls-2core-ssd ~]$ echo "results" > /tmp/results.txt
```

Start the Java code as a background process

Take your customer ID and shared key and replace them as parts of this command line. Additionally, provide a value for "logName" – this is what you'll use to look up the results in a query in LogAnalytics in Azure. Notice that we're running this as a background process by adding "&" to the end of the line. This means we can use the same terminal session to launch our performance test.

You'll get an immediate exception – this is OK (quick and dirty code, remember?) – it is just complaining that it cannot find any statistical results to transform and submit.

```
[dirservd@vm-pls-2core-ssd bin]$ java -cp "/tmp/json-20090211.jar:/tmp/FileTailer-
0.0.1-SNAPSHOT.jar" LogReader -fileName /tmp/results.txt -transmit Y -description
2cpussd -logName opendjperf -customerId "CUSTOMER-ID-GOES-HERE" -sharedKey "SHARED-
KEY-GOES-HERE" &
[1] 15686
[dirservd@vm-pls-2core-ssd bin]$ java.lang.NumberFormatException: For input string:
"results"
        at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:2043)
        at sun.misc.FloatingDecimal.parseDouble(FloatingDecimal.java:110)
        at java.lang.Double.parseDouble(Double.java:538)
        at java.lang.Double.<init>(Double.java:608)
        at LogReader.processLine(LogReader.java:199)
        at LogReader.startReading(LogReader.java:122)
        at LogReader.main(LogReader.java:93)
```

Start the OpenDJ searchrate tool

Next, make sure OpenDJ is running

```
[dirservd@vm-pls-2core-ssd ~]$ cd /usr/local/software/opendj/bin
[dirservd@vm-pls-2core-ssd bin]$ ./start-ds
[08/Feb/2018:19:44:48 +0000] category=CORE severity=NOTICE msgID=134 msg=ForgeRock
Directory Services 5.5.0 (build 20171019141329, revision number
3cc42334c8c36135c3098d963e09665ae0a3ea24) starting up
[08/Feb/2018:19:44:49 +0000] category=JVM severity=NOTICE msgID=21 msg=Installation
Directory:  /usr/local/software/opendj
[08/Feb/2018:19:44:49 +0000] category=JVM severity=NOTICE msgID=23 msg=Instance
Directory:      /usr/local/software/opendj
[08/Feb/2018:19:44:49 +0000] category=JVM severity=NOTICE msgID=17 msg=JVM
Information: 1.8.0_161-b14 by Oracle Corporation, 64-bit architecture, 2058354688
bytes heap size
[08/Feb/2018:19:44:49 +0000] category=JVM severity=NOTICE msgID=18 msg=JVM Host: vm-
pls-2core-ssd, running Linux 3.10.0-693.11.6.el7.x86_64 amd64, 4125949952 bytes
physical memory size, number of processors available 2
```

```
[08/Feb/2018:19:44:49 +0000] category=JVM severity=NOTICE msgID=19 msg=JVM
Arguments: "-Xms2g", "-Xmx2g", "-Dorg.opends.server.scriptName=start-ds"
[08/Feb/2018:19:44:54 +0000] category=BACKEND severity=NOTICE msgID=513 msg=The
database backend userRoot containing 50002 entries has started
[08/Feb/2018:19:44:55 +0000] category=EXTENSIONS severity=NOTICE msgID=221
msg=DIGEST-MD5 SASL mechanism using a server fully qualified domain name of: vm-pls-
2core-ssd
[08/Feb/2018:19:44:56 +0000] category=PROTOCOL severity=NOTICE msgID=276 msg=Started
listening for new connections on Administration Connector 0.0.0.0 port 4444
[08/Feb/2018:19:44:56 +0000] category=PROTOCOL severity=NOTICE msgID=276 msg=Started
listening for new connections on LDAP Connection Handler 0.0.0.0 port 1389
[08/Feb/2018:19:44:56 +0000] category=CORE severity=NOTICE msgID=135 msg=The
Directory Server has started successfully
[08/Feb/2018:19:44:56 +0000] category=CORE severity=NOTICE msgID=139 msg=The
Directory Server has sent an alert notification generated by class
org.opends.server.core.DirectoryServer (alert type
org.opends.server.DirectoryServerStarted, alert ID org.opends.messages.core-135):
The Directory Server has started successfully
```

And finally, start the performance test using the searchrate tool. The specifics of the switches is outside the scope of this project, but suffice it to say that we're going for 5,000 searches per second using four concurrent connections.

We're sending the output of this command to our results.txt file so it can be picked up by our java process

```
[dirservd@vm-pls-2core-hdd bin]$ ./searchrate -D "cn=Directory Manager" -w Test1234
-b "dc=example,dc=com" -F -M 5000 -c 4 -t 4 -p 1389 -g "rand(0,50000)"
"(uid=user.%d)" > /tmp/results.txt
```

Allow this to run for a while to generate data and submit it to Azure Log Analytics.

Debugging

If any portion of this does not go properly, you can debug by trying a few different things.

First, for searchrate, run it separately without sending the output to results.txt. you can do this just by removing "> results.txt" from the command line. It'll look like this:

```
[dirservd@vm-pls-8core-ssd bin]$ ./searchrate -D "cn=Directory Manager" -w Test1234
-b "dc=example,dc=com" -F -M 5000 -c 4 -t 4 -p 1389 -g "rand(0,50000)"
"(uid=user.%d)"
------------------------------------------------------------------------------------
--------
|     Throughput    |                   Response Time                |
Additional      |
|    (ops/second)   |                  (milliseconds)                |
Statistics      |
|   recent  average |   recent  average    99.9%   99.99%  99.999% |  err/sec
Entries/Srch |
------------------------------------------------------------------------------------
--------
|   5006.4   5006.4 |    0.850    0.850    15.01    23.72    28.18 |      0.0
1.0 |
|   5000.8   5003.6 |    0.497    0.673    11.86    19.40    27.53 |      0.0
1.0 |
|   4999.4   5002.2 |    0.483    0.610    10.62    18.61    27.53 |      0.0
1.0 |
```

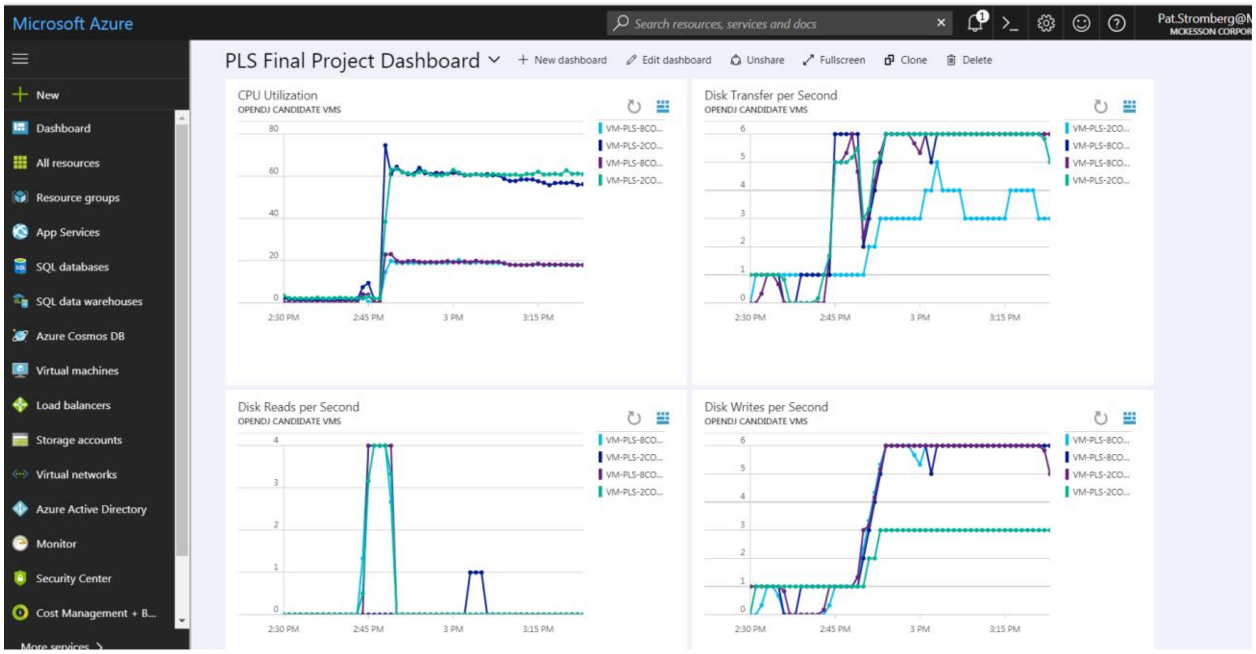
For the java side, you can set the "transmit" flag to "N" and set the "verbose" flag to "Y". This will show you what it would send, without actually sending it. Once you're happy that looks OK, you can set the “transmit” to “Y” (leaving verbose as Y) and see what status code you get back. It should be an HTTP 200. For instance:

```
[dirservd@vm-pls-2core-hdd bin]$ java -cp "/tmp/json-20090211.jar:/tmp/FileTailer-
0.0.1-SNAPSHOT.jar" LogReader -fileName /tmp/results.txt -transmit Y -description
2cpuhdd -logName testlog2 -customerId "***" -sharedKey "***" -verbose Y

line = results
java.lang.NumberFormatException: For input string: "results"
        at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:2043)
        at sun.misc.FloatingDecimal.parseDouble(FloatingDecimal.java:110)
        at java.lang.Double.parseDouble(Double.java:538)
        at java.lang.Double.<init>(Double.java:608)
        at LogReader.processLine(LogReader.java:199)
        at LogReader.startReading(LogReader.java:122)
        at LogReader.main(LogReader.java:93)
waiting
waiting
waiting
waiting
waiting
waiting
waiting
waiting
waiting
line = |    483.0    483.0 |   31.000   31.000   143.65   164.63   164.63 |      0.0
1.0 |
posting
{"response_average":31,"reponse_999":143.65,"response_recent":31,"throughput_recent"
:483,"throughput_average":483,"machine_name":"2cpuhdd"}

Sending 'POST' request to URL : https://7a956767-82f8-4392-a147-
5ee10d6e04f2.ods.opinsights.azure.com/api/logs?api-version=2016-04-01
Post parameters :
{"response_average":31,"reponse_999":143.65,"response_recent":31,"throughput_recent"
:483,"throughput_average":483,"machine_name":"2cpuhdd"}
Response Code : 200
line = |    775.4    629.1 |   20.494   24.532   153.09   158.33   164.63 |      0.0
1.0 |
posting
{"response_average":24.532,"reponse_999":153.09,"response_recent":20.494,"throughput
_recent":775.4,"throughput_average":629.1,"machine_name":"2cpuhdd"}

Sending 'POST' request to URL : https://7a956767-82f8-4392-a147-
5ee10d6e04f2.ods.opinsights.azure.com/api/logs?api-version=2016-04-01
Post parameters :
{"response_average":24.532,"reponse_999":153.09,"response_recent":20.494,"throughput
_recent":775.4,"throughput_average":629.1,"machine_name":"2cpuhdd"}
Response Code : 200
waiting
line = |    886.8    714.9 |   17.894   21.790   143.65   158.33   164.63 |      0.0
1.0 |
posting
{"response_average":21.79,"reponse_999":143.65,"response_recent":17.894,"throughput_
recent":886.8,"throughput_average":714.9,"machine_name":"2cpuhdd"}

Sending 'POST' request to URL : https://7a956767-82f8-4392-a147-
5ee10d6e04f2.ods.opinsights.azure.com/api/logs?api-version=2016-04-01
Post parameters :
{"response_average":21.79,"reponse_999":143.65,"response_recent":17.894,"throughput_
recent":886.8,"throughput_average":714.9,"machine_name":"2cpuhdd"}
Response Code : 200
waiting
```

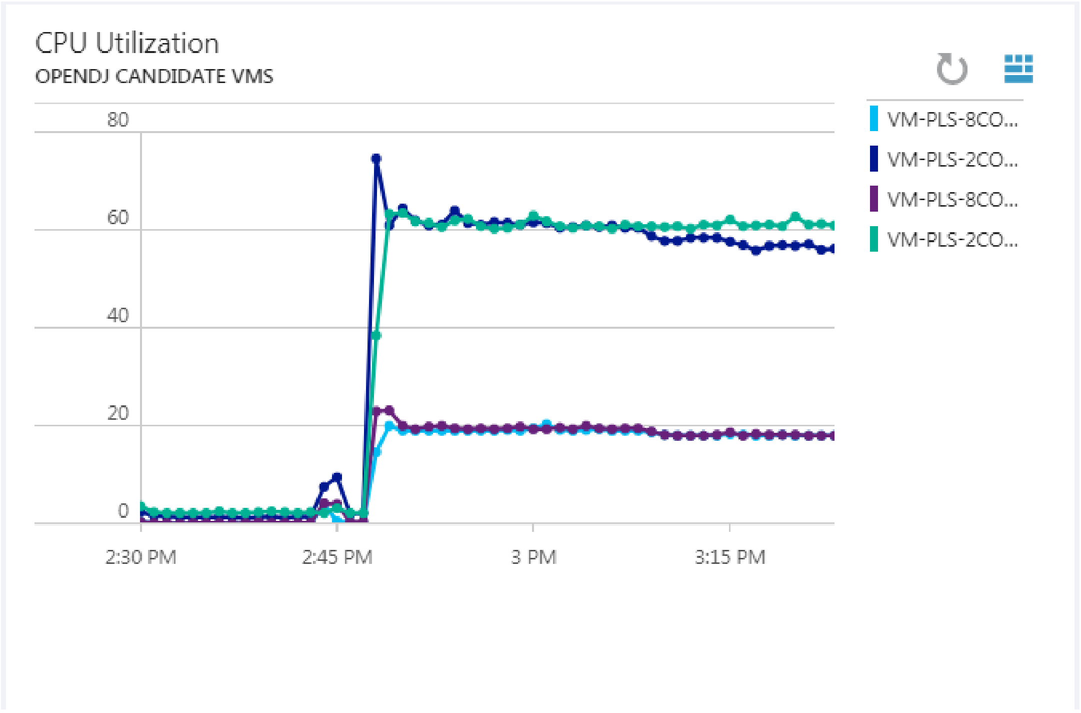
Analyzing the Results

Now that we have some data, both OS and application specific, flowing into our Log Analytics in Azure, we can hop over to the portal.

The first thing to check out is the dashboard we created previously. In running this test simultaneously across all four VMs, I arrived at a dashboard that looked like this:

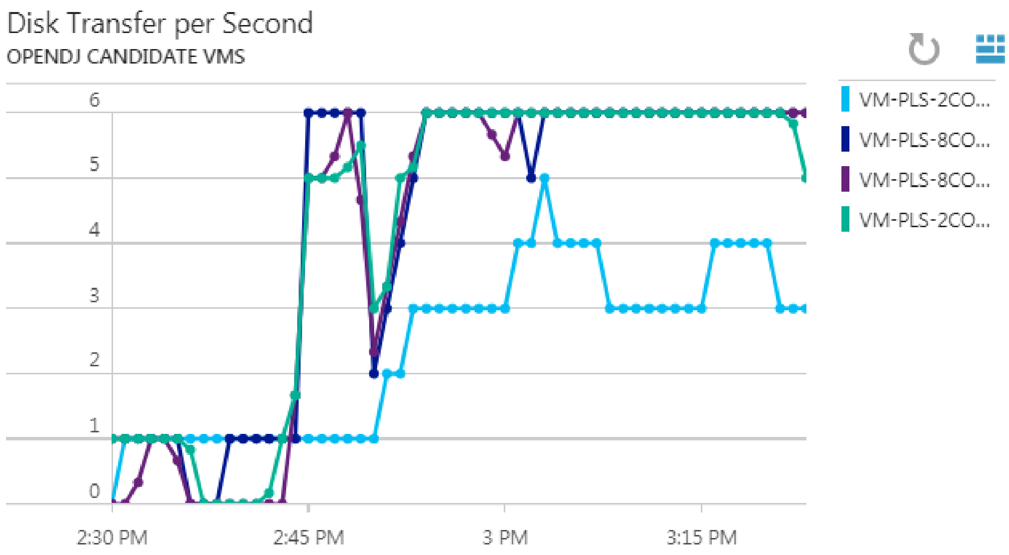


To break down each of those graphs, the CPU Utilization looked like this:



It may be hard to tell from this still image, but the 2 CPU boxes had much higher CPU utilization than the 8 CPU boxes. This is to be expected.

Disk utilization turned out to be largely a non-factor. We're cheating with our criteria here in that we're optimizing for just searches. If we were doing this in the real world, we'd also have to account for binds (authentications) and modifications, both of which can be more disk intensive. But for this project, we'll just who the Disk Transfer graph in detail and note that the 2 CPU hard disk drive box was the lowest, but that was because it was not able to get up to 5,000 requests per second (we'll see this when we run some ad-hoc queries)



Next, let's move on to some ad-hoc queries of the performance test data (response time and throughput) that we sent into the Data Collector API. We could have built a dashboard for these as well, but I decided to demonstrate some of the queries you can run in the Log Analytics in Azure.

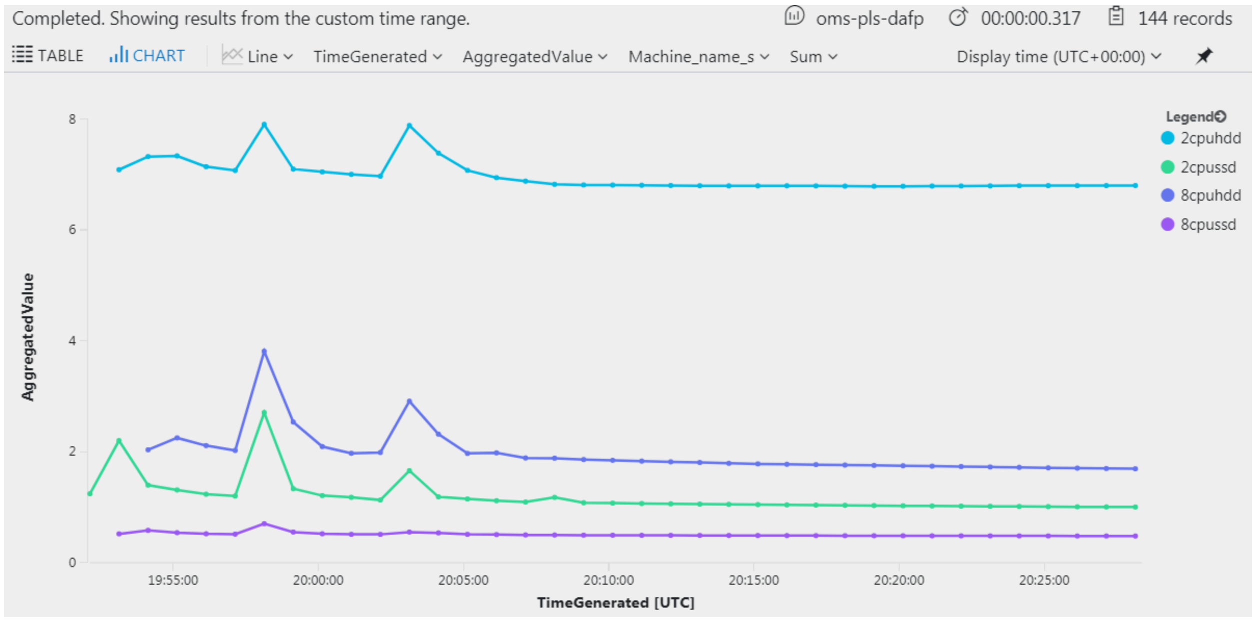
If you want to run ad-hoc queries, you can do so a few places. I following the process defined above when creating the dashboard – go into Log Search and then click on the Analytics button. Then open a new tab.

Now we'll run a few queries and ask for the results in a graphical format. For each of these, I've specified a custom time range for while I had the tests running on all four VMs.

First, the average response time

```
opendjperf_CL | summarize AggregatedValue = avg(response_average_d) by
bin(TimeGenerated, 1m), machine_name_s | render timechart
```

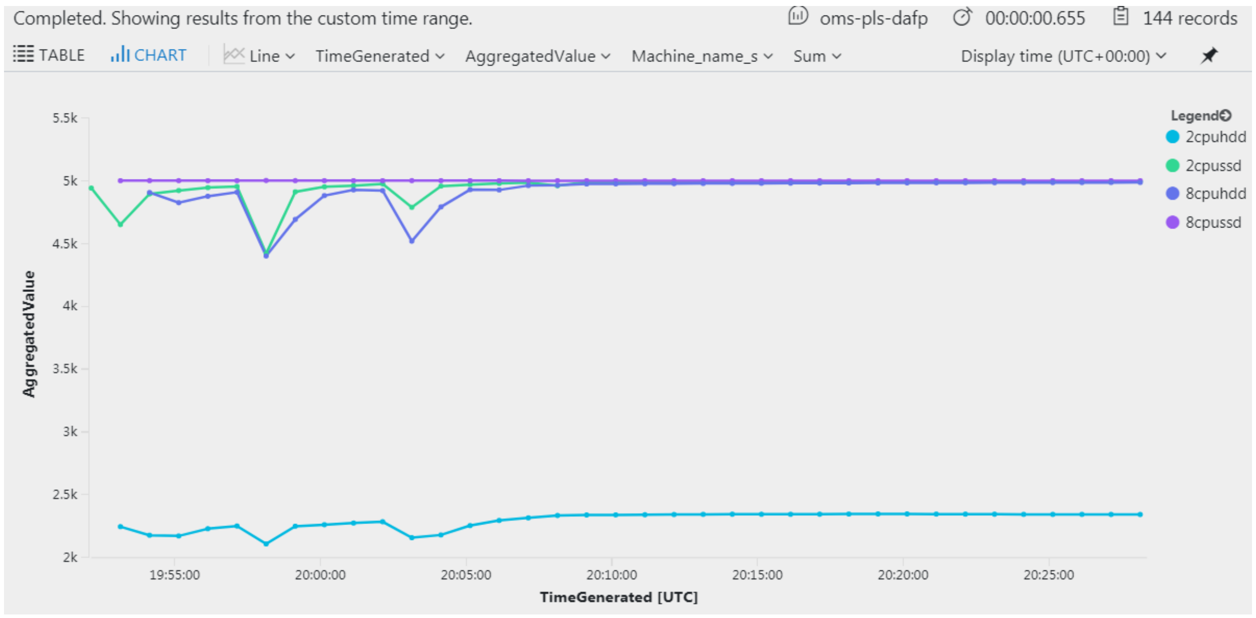
Which produces the following graph



We can see that the 2 CPU HDD box is the most expensive, but interestingly, the 2 CPU SSD is able to outperform the 8 CPU HDD!

Next, we'll discover something interesting in looking at the throughput

```
opendjperf_CL | summarize AggregatedValue = avg(throughput_average_d) by
bin(TimeGenerated, 1m), machine_name_s | render timechart
```



Here we can see that not only is the 2 CPU HDD a laggard in response time, it can't get anywhere close to 5,000 requests per second. Also notice that the 2 CPU SSD and 8 CPU HDD need some warmup time before they're at full capacity. This could be due to our testing tool (it is running on the same VM as OpenDJ, which is not ideal), or it could be due to some need to cache data.

Conclusions from the testing

- The 2 CPU HDD box is not viable. It will not support the required load.
- The 2 CPU SSD box is an interesting low-cost alternative. It might require some amount of warmup before being placed into service, but it was able to outperform the 8 CPU HDD
- If cost is not an issue, or high levels of service are required, the 8 CPU SSD is the way to go.
- I'd recommend further analysis with other configurations (4 CPU?)

Lessons Learned

I have the following takeaways from this exercise:

- I would change the directory tree structure in OpenDJ to more closely model the desired production use-case. It is not common that all users are thrown into a big pile in a single location in the directory tree. I'd also test the limits of group membership and attempt to strain the indexing strategy. There was no time to accomplish this for this assignment.
- I would automate the provisioning of the software onto the VMs using a tool like Ansible or Puppet. This would allow you to take just the binaries for OpenDJ and the custom Java code and run an automated setup against the VM. You could likely even tie the VM creation into the automated setup.
- The cloud is a fantastic performance testing lab, providing you're able to pay the bills. The ability to create n-many configurations and test them quickly and easily is priceless.
- I'm not sold on the use of Dashboards in Azure for things like this. They're nice and flashy, but you can likely get the job done using ad-hoc queries.

References

Several of these were mentioned inline above.

ForgeRock (maintainers of OpenDJ)
www.forgerock.com

Log Analytics documentation, with examples
https://docs.microsoft.com/en-us/azure/log-analytics/

The official Microsoft documentation on the Data Collector API
https://docs.microsoft.com/en-us/azure/log-analytics/log-analytics-data-collector-api

For continuously monitoring a file:
https://stackoverflow.com/questions/21037994/continuous-file-reading

For writing Java code that will call a REST-based API
http://crunchify.com/create-very-simple-jersey-rest-service-and-send-json-data-from-java-client/

For fulfilling the Data Collector API requirements for the shared key and transmitting it
http://systemcenterme.com/send-data-to-oms-log-analytics-with-java-code/

Full Source Code

```
import java.io.DataOutputStream;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.LineNumberReader;
import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.Base64;
import java.util.Date;
import java.util.StringTokenizer;
import java.util.TimeZone;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.net.ssl.HttpsURLConnection;
import javax.xml.bind.DatatypeConverter;

import org.json.JSONObject;

public class LogReader {

        // see printUsage method below for descriptions of
        // these variables
        static String fileName;
        static String transmit;
        static String description;
        static String logName;
        static String customerId;
        static String sharedKey;
        static String verbose = "N";

  private static void printUsage() {
    System.out.println("Usage: LogReader");
    System.out.println("        -fileName <file to monitor for searchrate results>");
    System.out.println("        -transmit <Y or N to indicate whether to attempt to transmit data to
Log Analytics>");
    System.out.println("        -description <label to identify this machine / run in Log Analytics");
    System.out.println("        -logName <the name of the custom log in Log Analytics");
    System.out.println("        -customerId <Log Analytics customer ID - obtained on the Azure
portal>");
    System.out.println("        -sharedKey <key required for transmission - also acquired from Log
Analytics in Azure portal>");
    System.out.println("        -verbose [if Y then print debug output]");
  }

  private static boolean validateArgs(String args[]) {
        // make sure we have the appropriate switches in place
        // otherwise, do not run
        boolean valid = true;
    for (int i = 0; i < args.length - 1;) {
      if (args[i].equals("-transmit")) {
        transmit = args[i + 1];
        i += 2;
      } else if (args[i].equals("-fileName")) {
        fileName = args[i + 1];
        i += 2;
      } else if (args[i].equals("-description")) {
        description = args[i + 1];
        i += 2;
      } else if (args[i].equals("-logName")) {
        logName = args[i + 1];
        i += 2;
      } else if (args[i].equals("-customerId")) {
        customerId = args[i + 1];
        i += 2;
      } else if (args[i].equals("-sharedKey")) {
        sharedKey = args[i + 1];
        i += 2;
      } else if (args[i].equals("-verbose")) {
        verbose = args[i + 1];
        i += 2;
      } else {
        i++;
      }
    }
    if (transmit == null
                    || transmit.length() == 0
                    || fileName == null
                    || fileName.length() == 0
                    || description == null
                    || description.length() == 0
                    || customerId == null
                    || customerId.length() == 0
                    || sharedKey == null
                    || sharedKey.length() == 0
                    || logName == null
                    || logName.length() == 0) {
      valid = false;
    }
    return valid;
  }

        public static void main(String[] args) {
                if (!validateArgs(args)) {
```

```
                printUsage();
            }
            else {
                try {
                    startReading(fileName);
                }
                catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }

private static boolean canBreak = false;

public static void startReading(String filename) throws InterruptedException, IOException {
  canBreak = false;
  String line;
  try {
    LineNumberReader lnr = new LineNumberReader(new FileReader(filename));
    while (!canBreak)
    {
      line = lnr.readLine();
      if (line == null) {
        if (verbose.equals("Y")) System.out.println("waiting");
        Thread.sleep(3000);
        continue;
      }
      processLine(line);
    }
    lnr.close();
  } catch (FileNotFoundException e) {
    e.printStackTrace();
  }
}

public static void stopReading() {
  canBreak = true;
}

private static void postMessage(String json) {

            String Signature = "";
            String encodedHash = "";
            String url = "";

            // Date object
            Date date = new Date();
		// Todays date input for OMS Log Analytics
		SimpleDateFormat sdf = new SimpleDateFormat("E, dd MMM YYYY HH:mm:ss zzz");
		sdf.setTimeZone(TimeZone.getTimeZone("GMT"));
		String timeNow = sdf.format(date);

		// String for signing the key
		String stringToSign="POST\n" + json.length() + "\napplication/json\nx-ms-
date:"+timeNow+"\n/api/logs";

		try {
			byte[] decodedBytes = Base64.getDecoder().decode(sharedKey);

			Mac hasher = Mac.getInstance("HmacSHA256");
			hasher.init(new SecretKeySpec(decodedBytes, "HmacSHA256"));
			byte[] hash = hasher.doFinal(stringToSign.getBytes());

			encodedHash = DatatypeConverter.printBase64Binary(hash);
			Signature = "SharedKey " + customerId + ":" + encodedHash;

			url = "https://" + customerId + ".ods.opinsights.azure.com/api/logs?api-
version=2016-04-01";
			URL objUrl = new URL(url);
			HttpsURLConnection con = (HttpsURLConnection) objUrl.openConnection();
			con.setDoOutput(true);
			con.setRequestMethod("POST");
			con.setRequestProperty("Content-Type", "application/json");
			con.setRequestProperty("Log-Type",logName);
			con.setRequestProperty("x-ms-date", timeNow);
			con.setRequestProperty("Authorization", Signature);

			DataOutputStream wr = new DataOutputStream(con.getOutputStream());
			wr.writeBytes(json);
			wr.flush();
			wr.close();

			int responseCode = con.getResponseCode();
			if (verbose.equals("Y")) {
				System.out.println("\nSending 'POST' request to URL : " + url);
				System.out.println("Post parameters : " + json);
				System.out.println("Response Code : " + responseCode);
			}
		}
		catch (Exception e) {
			System.out.println("Catch statement: " + e);
		}
    }

    private static void processLine(String s) {
```

```
    // this method will process a line that was read from the
        // performance testing results file
        try {
            JSONObject jo = new JSONObject();
            if (verbose.equals("Y")) System.out.println("line = " + s);

            // break the line up into tokens - space is the separator
            StringTokenizer st = new StringTokenizer(s," ");
            int colNum = 1;
            while (st.hasMoreTokens()) {
                String value = st.nextToken();
                if (value.indexOf("|") < 0) {
                    // this is not a column break - it is a number
                    Double dVal = new Double(value);

                    // the column number will determine
                    // which metric this is
                    switch (colNum) {
                    case 1:
                        jo.put("throughput_recent", dVal);
                    case 2:
                        jo.put("throughput_average", dVal);
                    case 3:
                        jo.put("response_recent",dVal);
                    case 4:
                        jo.put("response_average", dVal);
                    case 5:
                        jo.put("reponse_999", dVal);
                    }
                    colNum++;
                }
            }
            jo.put("machine_name", description);
            if (transmit.equals("Y")) {
                if (verbose.equals("Y")) System.out.println("posting " + jo.toString());
                postMessage(jo.toString());
            }
            else {
                if (verbose.equals("Y")) System.out.println("DEBUG " + jo.toString());
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
  }
}
```