Stephen Tambussi

April 18th, 2022

COEN 241

HW 1

Configurations

System	Desktop PC	QEMU VM/Docker Container		
Type	NA	Low-end	Mid-end	High-end
CPU	Ryzen 5 2600 6-core (12-threads) @ 3.4 GHz	Cores: 1 System Threads: 2	Cores: 2 System Threads: 4	Cores: 4 System Threads: 8
Memory	16 GB of DDR4 @ 3000 MHz	2 GB	4 GB	8 GB
Storage	256 GB NVMe SSD for Windows 500 GB SATA3 SSD for additional storage 256 GB SATA3 SSD for Ubuntu	- VM has 30 GB dedicated - Container shares storage with host OS (Ubuntu)		
Notes	System is dual-booted with Windows 10 Pro and Ubuntu Desktop 20.04 LTS	- QEMU VM runs on Ubuntu 20.04 LTS Desktop - VM OS is Ubuntu 20.04 Server - Docker Container runs on Ubuntu 20.04 LTS Desktop - Each CPU Core has 2 Threads (Hyperthreading)		

Running a QEMU Virtual Machine

When a VM has already been installed, it is quite easy to spin it up via the command line:

```
sudo qemu-system-x86_64 -m 4096 -boot d -accel kvm -cpu EPYC -smp
cores=2,threads=2 -boot strict=on -hda ubuntu.img
```

The above command will start the VM in a graphical window which allows direct access to its command line interface. As seen in the command's arguments, this particular VM is allocated 4

GB of RAM and KVM (Kernel-based Virtual Machine) is the accelerator which allows the kernel to act as a hypervisor (type-1). The CPU model is EPYC because QEMU does not have a preset model for Ryzen 2000 series processors and they have similar architectures. The number of cores allocated to this VM is 2, with 2 threads per core, for a total of 2 cores and 4 threads. The OS is loaded from the "ubuntu.img" file which was written to from an ISO file during installation. These configuration options will change to see how the VM's performance is affected.

QEMU VM Performance Testing with Varying Configurations

As shown in the configuration table, 3 performance scenarios were selected for each virtualization technology. Classified from "Low-end" to "High-end", each successive scenario doubles the system resources allocated to it. The Sysbench tests utilized were CPU and File I/O testing modes. For each of these testing modes, subtests were devised to analyze how performance scales with each scenario. These subtests are the same across both virtualization platforms and every subtest was run on each scenario 5 times.

CPU Tests				**File I/O Tests**		
Test Number	**1**	**2**	**3**	**Test Number**	**1**	**2**
threads	2	4	8	**file-num**	128	128
time(s)	30	30	30	**file-total-size**	10 GB	10 GB
cpu-max-prime	10000	10000	10000	**file-io-mode**	seqwr	seqrd

For the sysbench CPU tests, all the parameters were kept the same except for the number of process threads to determine how well the VM can scale in performance with more resources. The file I/O tests were kept the same except for the I/O mode to measure both read and write storage performance of the VM.

To run the VM with its performance scenarios and then execute the sysbench tests, shell scripts were created to automate the process. These scripts can be found below:

Script 1: Run VM Performance Scenarios

```
#!/bin/bash

#Arg = 0 - low-end config
#Arg = 1 - middle-end config
#Arg = 2 - high-end config
```

```
vm_config=$1

if [ $vm_config -eq 0 ];
then
   sudo qemu-system-x86_64 -m 2048 -boot d -accel kvm -cpu EPYC -smp
cores=1,threads=2 -nic user,hostfwd=tcp::2222-:22 -boot strict=on -hda
ubuntu.img

elif [ $vm_config -eq 1 ];
then
   sudo qemu-system-x86_64 -m 4096 -boot d -accel kvm -cpu EPYC -smp
cores=2,threads=2 -nic user,hostfwd=tcp::2222-:22 -boot strict=on -hda
ubuntu.img

elif [ $vm_config -eq 2 ];
then
   sudo qemu-system-x86_64 -m 8192 -boot d -accel kvm -cpu EPYC -smp
cores=4,threads=2 -nic user,hostfwd=tcp::2222-:22 -boot strict=on -hda
ubuntu.img

else
   echo "Incorrect VM Configuration selected."
   exit 1
fi
```

Script 2: Run Sysbench CPU Tests

```
#!/bin/bash

#Stephen Tambussi - COEN241 - HW1
#Script to run sysbench CPU tests

#Get current directory
dir=$(pwd)
#Check if file exists and create it if it does not
filename=$dir/vm_sysbench_cpu_results.txt
if [ ! -f $filename ]
then
   touch $filename
   echo "QEMU VM Sysbench CPU Test Results" >> $filename
fi


#This gets the number of CPU cores allocated to the VM to detect current VM
Config
cpus=$(lscpu | grep "Core(s) per socket:" | grep -Eo '[0-9]')

if [ $cpus -eq 1 ]; #Low-end config - 1 core, 2 threads
then
   echo "Low-end VM Configuration Tests: 1 core, 2 threads, 2 GB Mem" >>
$filename
   echo "============================================================" >>
$filename

elif [ $cpus -eq 2 ]; #Middle-end config - 2 cores, 4 threads
then
```

```
   echo "Middle-end VM Configuration Tests: 2 cores, 4 threads, 4 GB Mem" >>
$filename
   echo "============================================================" >>
$filename

elif [ $cpus -eq 4 ]; #High-end config - 4 cores, 8 threads
then
   echo "High-end VM Configuration Tests: 4 cores, 8 threads, 8 GB Mem" >>
$filename
   echo "============================================================" >>
$filename

else
   echo "Incorrect number of cores to run sysbench, needs to match preset
configs"
   exit 1
fi

#3 types of sysbench cpu tests with varying threads(2,4,8). Each of these tests
are run 5 times on each VM config.
echo "CPU Test with 2 Threads" >> $filename
echo "-----------------------" >> $filename
for ((counter=1; counter<6; counter++))
do
   echo "TEST $counter" >> $filename
   sysbench --test=cpu --threads=2 --cpu-max-prime=10000 --time=30 run | grep
"events per second:\|total number of events:" >> $filename
done

echo "CPU Test with 4 Threads" >> $filename
echo "-----------------------" >> $filename
for ((counter=1; counter<6; counter++))
do
   echo "TEST $counter" >> $filename
   sysbench --test=cpu --threads=4 --cpu-max-prime=10000 --time=30 run | grep
"events per second:\|total number of events:" >> $filename
done

echo "CPU Test with 8 Threads" >> $filename
echo "-----------------------" >> $filename
for ((counter=1; counter<6; counter++))
do
   echo "TEST $counter" >> $filename
   sysbench --test=cpu --threads=8 --cpu-max-prime=10000 --time=30 run | grep
"events per second:\|total number of events:" >> $filename
done
```

Script 3: Run Sysbench File I/O Tests

```
#!/bin/bash

#Stephen Tambussi - COEN241 - HW1
#Script to run sysbench file io tests

#Get current directory
dir=$(pwd)
#Check if file exists and create it if it does not
filename=$dir/vm_sysbench_io_results.txt
```

```
if [ ! -f $filename ]
then
   touch $filename
   echo "QEMU VM Sysbench File IO Test Results" >> $filename
fi

#This gets the number of CPU cores allocated to the VM to detect current VM
config
cpus=$(lscpu | grep "Core(s) per socket:" | grep -Eo '[0-9]')

if [ $cpus -eq 1 ]; #Low-end config - 1 core, 2 threads
then
   echo "Low-end VM Configuration Tests: 1 core, 2 threads, 2 GB Mem" >>
$filename
   echo "============================================================" >>
$filename

elif [ $cpus -eq 2 ]; #Middle-end config - 2 cores, 4 threads
then
   echo "Middle-end VM Configuration Tests: 2 cores, 4 threads, 4 GB Mem" >>
$filename
   echo "============================================================" >>
$filename

elif [ $cpus -eq 4 ]; #High-end config - 4 cores, 8 threads
then
   echo "High-end VM Configuration Tests: 4 cores, 8 threads, 8 GB Mem" >>
$filename
   echo "============================================================" >>
$filename

else
   echo "Incorrect number of cores to run sysbench, needs to match preset
configs"
   exit 1
fi

echo "File IO Test in Sequential Write Mode" >> $filename
echo "-------------------------------------" >> $filename
for ((counter=1; counter<6; counter++))
do
   echo "TEST $counter" >> $filename
   #Create the files to test
   sysbench --test=fileio --file-num=128 --file-total-size=10G prepare
   #Run the actual test
   sysbench --test=fileio --file-num=128 --file-total-size=10G
--file-test-mode=seqwr run \
   | grep "Throughput:\|read, MiB/s:\|written, MiB/s:\|Latency
(ms):\|min:\|avg:\|max:\|95th percentile:\|sum:" >> $filename
   #Cleanup the test files
   sysbench --test=fileio cleanup
   #Drop the cache
   sudo sh -c 'echo 3 > /proc/sys/vm/drop_caches'
done

echo "File IO Test in Sequential Read Mode" >> $filename
echo "-------------------------------------" >> $filename
```

```
for ((counter=1; counter<6; counter++))
do
   echo "TEST $counter" >> $filename
   #Create the files to test
   sysbench --test=fileio --file-num=128 --file-total-size=10G prepare
   #Run the actual test
   sysbench --test=fileio --file-num=128 --file-total-size=10G
--file-test-mode=seqrd run \
   | grep "Throughput:\|read, MiB/s:\|written, MiB/s:\|Latency
(ms):\|min:\|avg:\|max:\|95th percentile:\|sum:" >> $filename
   #Cleanup the test files
   sysbench --test=fileio cleanup
   #Drop the cache
   sudo sh -c 'echo 3 > /proc/sys/vm/drop_caches'
done
```

QEMU VM Performance Testing Results

CPU Tests

As previously mentioned, all 3 CPU subtests were run 5 times on each VM scenario for a total 15 sysbench runs on one scenario or 45 sysbench runs across all 3 scenarios. Screenshots of each subtest are shown below:

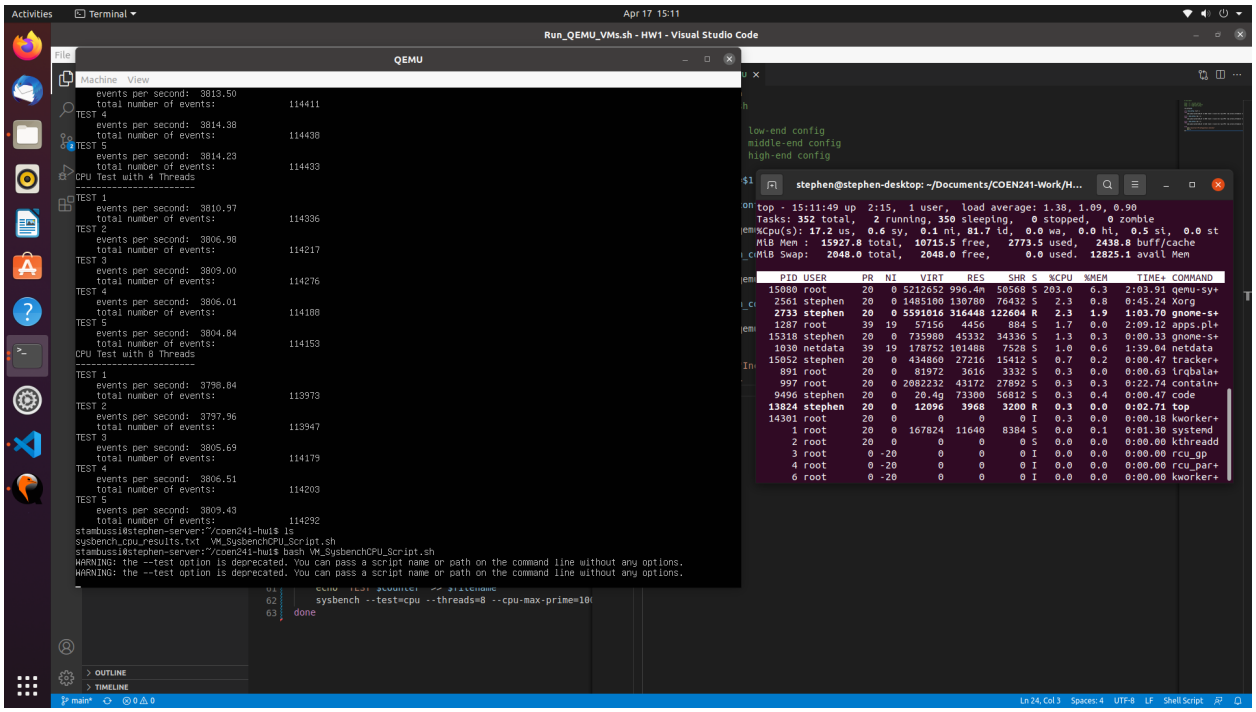

Figure 1: VM CPU Test with 2 Threads

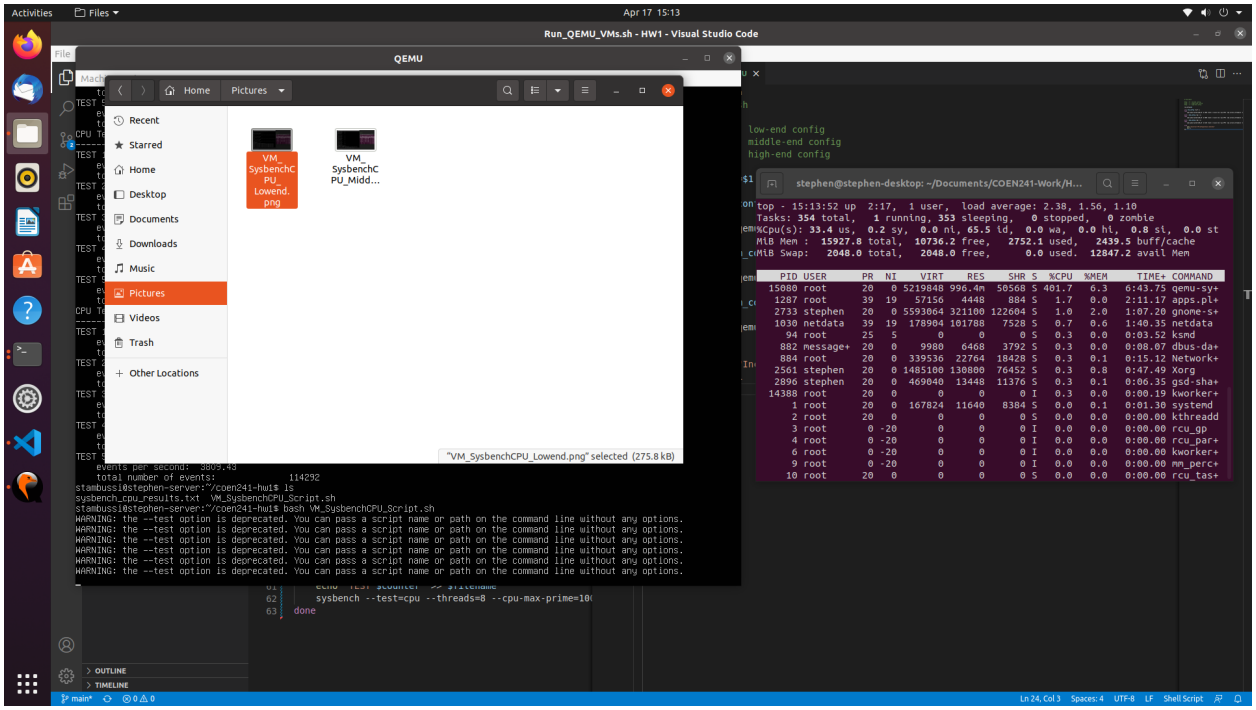

Figure 2: VM CPU Test with 4 Threads

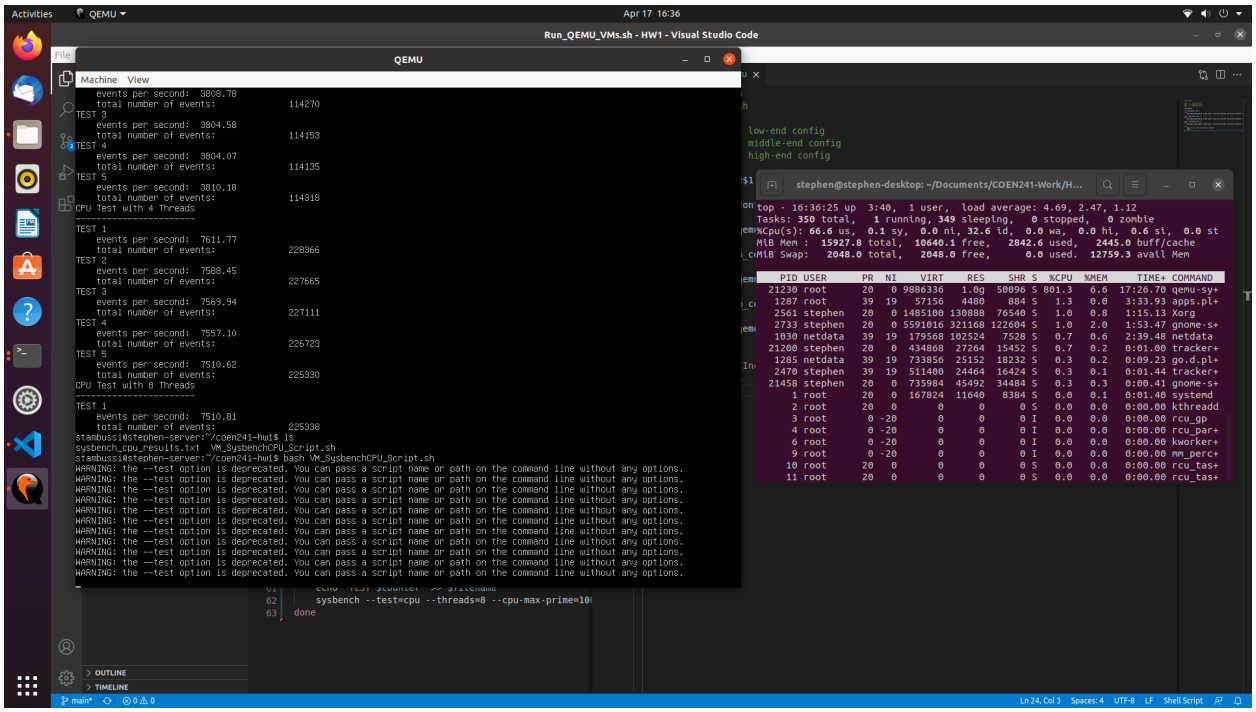

Figure 3: VM CPU Test with 8 Threads

The performance data collected for each test is events per second, total number of events, kernel-level CPU utilization, and user-level CPU utilization. Sysbench reports events per second and total events, while the CPU utilization was collected using the ‘top’ command line tool.

Low-end

2 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**3812.00**	**114366.00**	**0.22**	**16.92**
min	**3805.04**	**114157.00**	**0.20**	**16.70**
max	**3814.38**	**114438.00**	**0.30**	**17.20**
std	**3.94**	**118.33**	**0.04**	**0.19**

4 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**3807.56**	**114234.00**	**0.24**	**16.86**
min	**3804.84**	**114153.00**	**0.20**	**16.50**
max	**3810.97**	**114336.00**	**0.30**	**17.20**
std	**2.44**	**72.69**	**0.05**	**0.27**

8 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**3803.69**	**114118.80**	**0.26**	**16.94**
min	**3797.96**	**113947.00**	**0.20**	**16.70**
max	**3809.43**	**114292.00**	**0.30**	**17.20**
std	**5.03**	**151.23**	**0.05**	**0.21**

Mid-end

2 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**3807.32**	**114231.60**	**0.24**	**16.94**
min	**3804.07**	**114135.00**	**0.20**	**16.40**
max	**3810.18**	**114318.00**	**0.40**	**17.50**
std	**2.79**	**82.14**	**0.09**	**0.43**

4 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**7567.58**	**227039.00**	**0.24**	**33.20**
min	**7510.62**	**225330.00**	**0.20**	**32.80**

max	**7611.77**	**228366.00**	**0.40**	**33.70**
std	**37.91**	**1137.82**	**0.09**	**0.37**

8 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**7491.55**	**224761.20**	**0.26**	**33.12**
min	**7475.20**	**224271.00**	**0.20**	**32.90**
max	**7516.76**	**225518.00**	**0.30**	**33.50**
std	**20.44**	**613.01**	**0.05**	**0.27**

High-end

2 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**3815.36**	**114468.40**	**0.30**	**16.72**
min	**3812.77**	**114397.00**	**0.20**	**16.30**
max	**3816.89**	**114513.00**	**0.50**	**17.20**
std	**1.69**	**47.73**	**0.12**	**0.33**

4 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**7614.06**	**228437.00**	**0.18**	**33.16**
min	**7599.07**	**227985.00**	**0.10**	**32.80**
max	**7624.43**	**228746.00**	**0.30**	**33.50**
std	**9.35**	**280.83**	**0.08**	**0.30**

8 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**10926.84**	**327827.40**	**0.18**	**66.76**
min	**10914.25**	**327449.00**	**0.10**	**66.50**
max	**10937.55**	**328152.00**	**0.30**	**67.00**
std	**10.56**	**317.88**	**0.08**	**0.21**

As shown in the above tables, the sysbench performance does not improve with more threads unless the VM has more threads to spare. This is made evident by the “Low-end” scenario not

showing any discernible improvement in performance when the sysbench thread count is increased.

File I/O Tests

The file I/O tests, sequential read and write, were run 5 times on each VM scenario for a total of 10 sysbench runs on one scenario or 30 sysbench runs across all 3 scenarios. The performance data collected for each test is read and write speed (MiB/s), latency (ms), and disk utilization (%). Sysbench reports the I/O throughput and latency, while disk utilization was collected from the netdata tool.

Low-end

Sequential Write				
	Read (MiB/s)	**Write (MiB/s)**	**Latency (ms)**	**Disk Util. (%)**
average	-	**68.36**	**0.10**	**37.80**
min	-	**67.05**	**0.10**	**34.00**
max	-	**70.26**	**0.10**	**40.00**
std	-	**1.47**	**0.00**	**2.28**

Sequential Read				
	Read (MiB/s)	**Write (MiB/s)**	**Latency (ms)**	**Disk Util. (%)**
average	**438.75**	-	**0.04**	**99.20**
min	**414.02**	-	**0.03**	**97.00**
max	**506.10**	-	**0.04**	**100.00**
std	**38.91**	-	**0.00**	**1.30**

Mid-end

Sequential Write				
	Read (MiB/s)	**Write (MiB/s)**	**Latency (ms)**	**Disk Util. (%)**
average	-	**65.43**	**0.11**	**38.80**
min	-	**63.76**	**0.10**	**38.00**
max	-	**68.80**	**0.11**	**40.00**
std	-	**1.96**	**0.01**	**0.84**

Sequential Read				
	Read (MiB/s)	**Write (MiB/s)**	**Latency (ms)**	**Disk Util. (%)**

average	**430.24**	-	**0.04**	**99.60**
min	**399.41**	-	**0.03**	**99.00**
max	**456.05**	-	**0.04**	**100.00**
std	**20.93**	-	**0.00**	**0.55**

High-end

Sequential Write				
	Read (MiB/s)	**Write (MiB/s)**	**Latency (ms)**	**Disk Util. (%)**
average	-	**64.00**	**0.11**	**38.00**
min	-	**63.35**	**0.10**	**37.00**
max	-	**65.06**	**0.11**	**39.00**
std	-	**0.65**	**0.00**	**0.71**

Sequential Read				
	Read (MiB/s)	**Write (MiB/s)**	**Latency (ms)**	**Disk Util. (%)**
average	**412.42**	-	**0.04**	**98.80**
min	**379.42**	-	**0.04**	**95.00**
max	**428.42**	-	**0.04**	**100.00**
std	**19.81**	-	**0.00**	**2.17**

The I/O performance across every scenario is very similar. This conveys that I/O performance is independent of allocated system resources.

Running a Docker Container

Once docker has been installed, it is easy to run a container:

```
sudo docker run --rm -it --cpuset-cpus="0-1" -m 2G --entrypoint /bin/sh
zyclonite/sysbench
```

The command above will start the container with image *zyclonite/sysbench* which is used to run sysbench. This container is limited to 1 CPU core, or 2 logical CPU cores (threads), and is allocated 2 GB of RAM. The *-it* parameter runs the container interactively, while the *–entrypoint* parameter specifies the container's shell for interaction. The *–rm* will automatically remove the container when it exits.

The command below lists all the currently running containers along with associated information:

```
sudo docker ps -a
```

This command will display all the container images currently installed on the system:

```
sudo docker images -a
```

Finally, to remove a container image from the system run the below command with the desired image ID:

```
sudo docker rmi <image_id>
```

Docker Container Performance Testing with Varying Configurations

As specified for the QEMU VM, the Docker container was run with 3 scenarios and a suite of tests to analyze its performance across these scenarios. The same table from the VM section was copied for reference.

CPU Tests				**File I/O Tests**		
Test Number	**1**	**2**	**3**	**Test Number**	**1**	**2**
threads	2	4	8	**file-num**	128	128
time(s)	30	30	30	**file-total-size**	10 GB	10 GB
cpu-max-prime	10000	10000	10000	**file-io-mode**	seqwr	seqrd

Shell scripts were written to automate the container performance scenarios and run the sysbench CPU tests. However, due to the extra steps required by the file I/O tests, no script was written for these tests and instead they were performed manually.

Script 4: Run Container Performance Scenarios

```
#!/bin/bash

#This is to run the docker container for the file I/O tests
#Arg = 0 - low-end config
#Arg = 1 - middle-end config
#Arg = 2 - high-end config

contain_config=$1

if [ $contain_config -eq 0 ];
then
   sudo docker run --rm -it --cpuset-cpus="0-1" -m 2G --entrypoint /bin/sh
zyclonite/sysbench

elif [ $contain_config -eq 1 ];
then
   sudo docker run --rm -it --cpuset-cpus="0-3" -m 4G --entrypoint /bin/sh
zyclonite/sysbench

elif [ $contain_config -eq 2 ];
then
```

```
   sudo docker run --rm -it --cpuset-cpus="0-7" -m 8G --entrypoint /bin/sh
zyclonite/sysbench

else
   echo "Incorrect Container Configuration selected."
   exit 1
fi
```

Script 5: Run Sysbench CPU Tests

```
#!/bin/bash

#Stephen Tambussi - COEN241 - HW1
#Script to run sysbench CPU tests

#Get current directory
dir=$(pwd)
#Check if file exists and create it if it does not
filename=$dir/container_sysbench_cpu_results.txt
if [ ! -f $filename ]
then
   touch $filename
   echo "Docker Container Sysbench CPU Test Results" >> $filename
fi

echo "Low-end Container Configuration Tests: 1 core, 2 threads, 2 GB Mem" >>
$filename
echo "============================================================" >> $filename
echo "Low-end tests"
echo "CPU Test with 2 Threads" >> $filename
echo "-----------------------" >> $filename
for ((counter=1; counter<6; counter++))
do
   echo "TEST $counter" >> $filename
   sudo docker run --rm --cpuset-cpus="0-1" -m 2G zyclonite/sysbench --test=cpu
--threads=2 --cpu-max-prime=10000 --time=30 run | grep "events per
second:\|total number of events:" >> $filename
done

echo "CPU Test with 4 Threads" >> $filename
echo "-----------------------" >> $filename
for ((counter=1; counter<6; counter++))
do
   echo "TEST $counter" >> $filename
   sudo docker run --rm --cpuset-cpus="0-1" -m 2G zyclonite/sysbench --test=cpu
--threads=4 --cpu-max-prime=10000 --time=30 run | grep "events per
second:\|total number of events:" >> $filename
done

echo "CPU Test with 8 Threads" >> $filename
echo "-----------------------" >> $filename
for ((counter=1; counter<6; counter++))
do
   echo "TEST $counter" >> $filename
   sudo docker run --rm --cpuset-cpus="0-1" -m 2G zyclonite/sysbench --test=cpu
--threads=8 --cpu-max-prime=10000 --time=30 run | grep "events per
second:\|total number of events:" >> $filename
done
```

```
echo "Middle-end Container Configuration Tests: 2 cores, 4 threads, 4 GB Mem"
>> $filename
echo "============================================================" >> $filename
echo "Middle-end tests"
echo "CPU Test with 2 Threads" >> $filename
echo "-----------------------" >> $filename
for ((counter=1; counter<6; counter++))
do
   echo "TEST $counter" >> $filename
   sudo docker run --rm --cpuset-cpus="0-3" -m 4G zyclonite/sysbench --test=cpu
--threads=2 --cpu-max-prime=10000 --time=30 run | grep "events per
second:\|total number of events:" >> $filename
done

echo "CPU Test with 4 Threads" >> $filename
echo "-----------------------" >> $filename
echo "CPU Test with 4 Threads -- CPU utilization should become higher"
for ((counter=1; counter<6; counter++))
do
   echo "TEST $counter" >> $filename
   sudo docker run --rm --cpuset-cpus="0-3" -m 4G zyclonite/sysbench --test=cpu
--threads=4 --cpu-max-prime=10000 --time=30 run | grep "events per
second:\|total number of events:" >> $filename
done

echo "CPU Test with 8 Threads" >> $filename
echo "-----------------------" >> $filename
for ((counter=1; counter<6; counter++))
do
   echo "TEST $counter" >> $filename
   sudo docker run --rm --cpuset-cpus="0-3" -m 4G zyclonite/sysbench --test=cpu
--threads=8 --cpu-max-prime=10000 --time=30 run | grep "events per
second:\|total number of events:" >> $filename
done

echo "High-end Container Configuration Tests: 4 cores, 8 threads, 8 GB Mem" >>
$filename
echo "============================================================" >> $filename
echo "High-end tests"
echo "CPU Test with 2 Threads" >> $filename
echo "-----------------------" >> $filename
for ((counter=1; counter<6; counter++))
do
   echo "TEST $counter" >> $filename
   sudo docker run --rm --cpuset-cpus="0-7" -m 8G zyclonite/sysbench --test=cpu
--threads=2 --cpu-max-prime=10000 --time=30 run | grep "events per
second:\|total number of events:" >> $filename
done

echo "CPU Test with 4 Threads" >> $filename
echo "-----------------------" >> $filename
echo "CPU Test with 4 Threads -- CPU utilization should become higher"
for ((counter=1; counter<6; counter++))
do
   echo "TEST $counter" >> $filename
```

```
   sudo docker run --rm --cpuset-cpus="0-7" -m 8G zyclonite/sysbench --test=cpu
--threads=4 --cpu-max-prime=10000 --time=30 run | grep "events per
second:\|total number of events:" >> $filename
done

echo "CPU Test with 8 Threads" >> $filename
echo "-----------------------" >> $filename
echo "CPU Test with 8 Threads -- CPU utilization should become higher"
for ((counter=1; counter<6; counter++))
do
   echo "TEST $counter" >> $filename
   sudo docker run --rm --cpuset-cpus="0-7" -m 8G zyclonite/sysbench --test=cpu
--threads=8 --cpu-max-prime=10000 --time=30 run | grep "events per
second:\|total number of events:" >> $filename
done
```

Docker Container Performance Testing Results

CPU Tests

Similar to the VM testing, a total of 45 sysbench CPU tests were conducted for the container testing. Screenshots of each subtest are shown below:

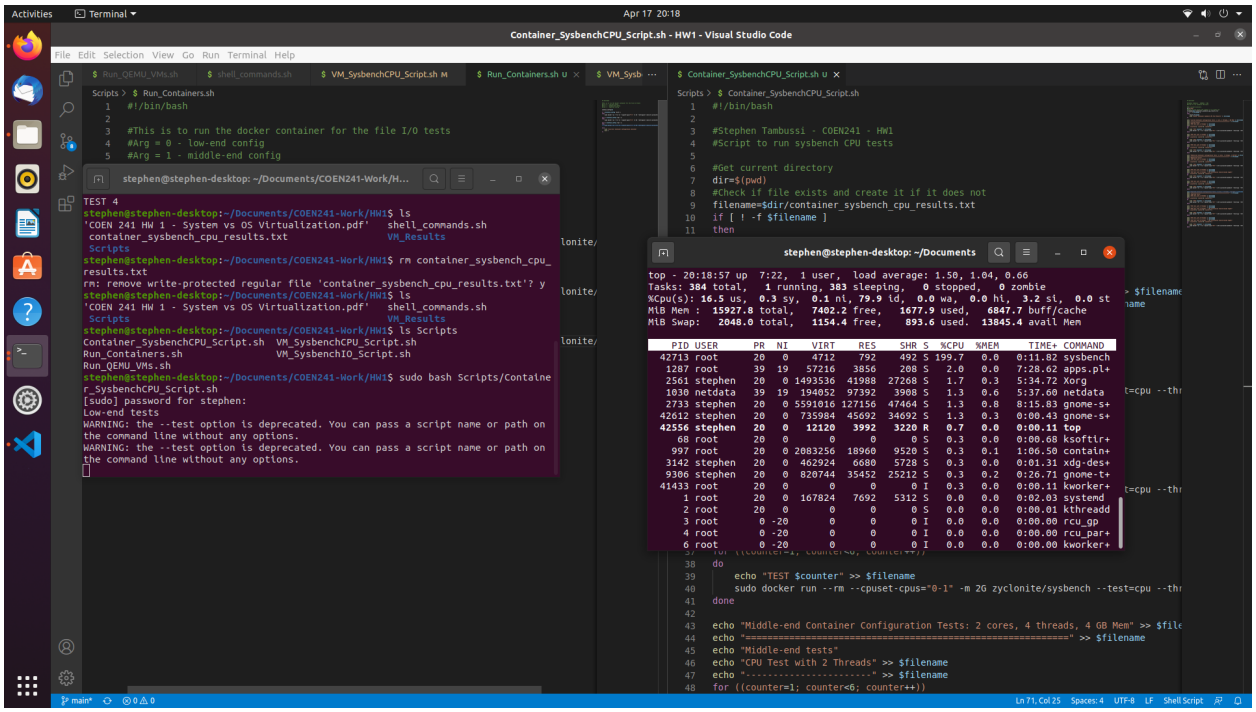

Figure 4: Container CPU Test with 2 Threads

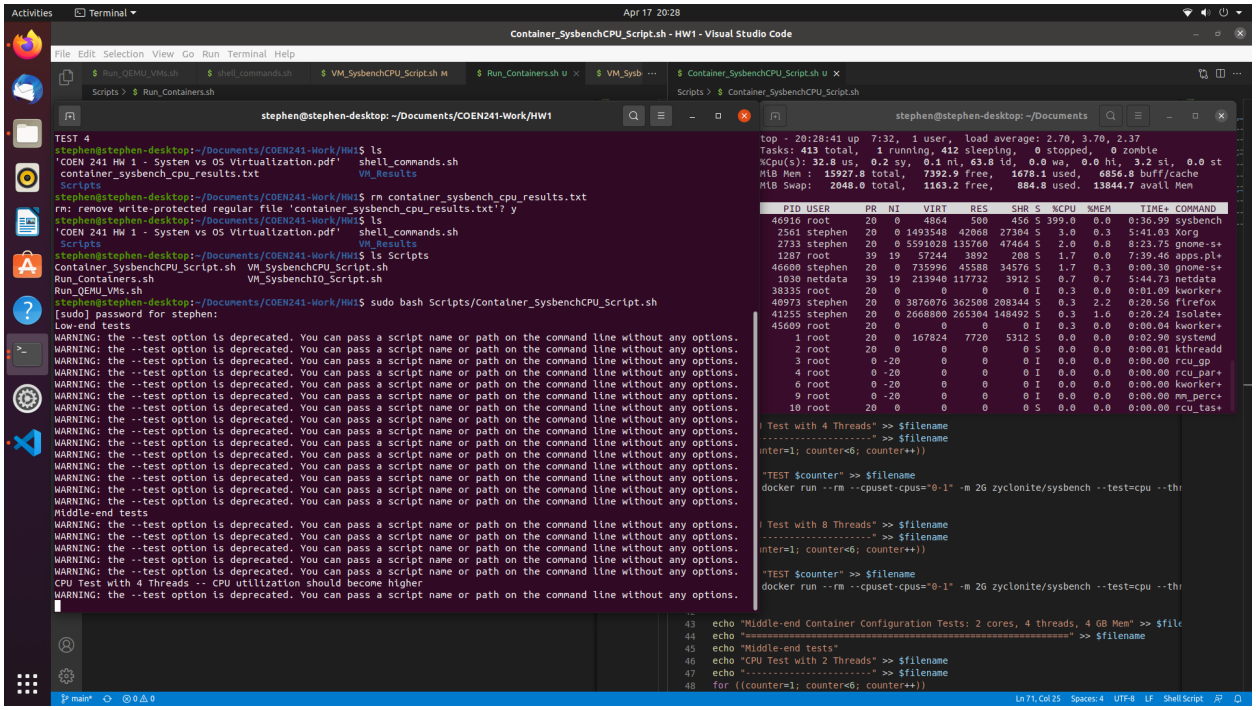

Figure 5: Container CPU Test with 4 Threads

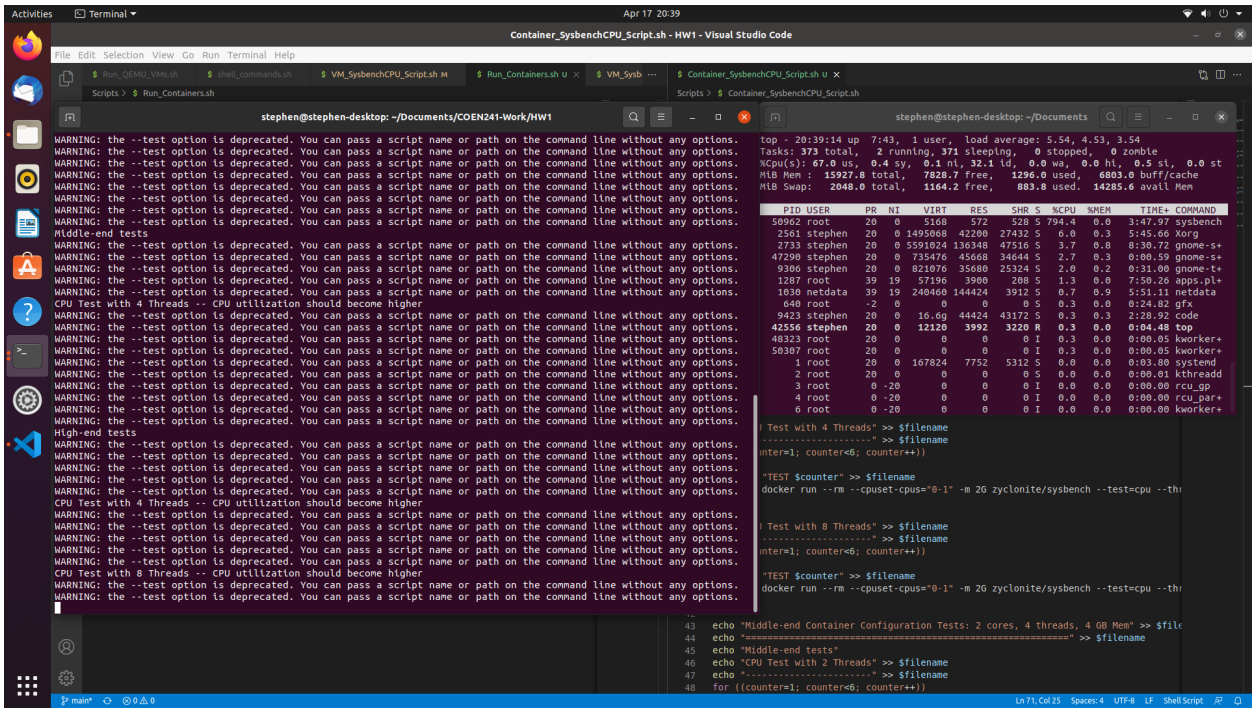

Figure 6: Container CPU Test with 8 Threads

The performance data collected for each test is the same as the VM testing with the ‘top’ tool being used to collect CPU utilization.

Low-end

2 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**3481.23**	**104442.40**	**0.28**	**16.64**
min	**3480.87**	**104432.00**	**0.20**	**16.50**
max	**3481.77**	**104459.00**	**0.30**	**16.80**
std	**0.34**	**10.45**	**0.04**	**0.13**

4 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**3483.99**	**104525.80**	**0.30**	**16.64**
min	**3480.50**	**104421.00**	**0.30**	**16.60**
max	**3492.06**	**104768.00**	**0.30**	**16.70**
std	**4.79**	**143.62**	**0.00**	**0.05**

8 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**3481.63**	**104457.80**	**0.22**	**16.66**
min	**3480.61**	**104426.00**	**0.10**	**16.50**
max	**3484.32**	**104539.00**	**0.30**	**16.70**
std	**1.54**	**46.73**	**0.08**	**0.09**

Mid-end

2 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**3480.83**	**104433.80**	**0.30**	**16.64**
min	**3480.06**	**104413.00**	**0.30**	**16.50**
max	**3481.50**	**104450.00**	**0.30**	**16.80**
std	**0.58**	**14.97**	**0.00**	**0.11**

4 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**6891.84**	**206766.60**	**0.22**	**32.90**
min	**6830.84**	**204937.00**	**0.20**	**32.70**
max	**6950.69**	**208533.00**	**0.30**	**33.10**

std	**50.34**	**1510.55**	**0.04**	**0.16**

8 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**6830.20**	**204919.20**	**0.18**	**32.92**
min	**6822.83**	**204698.00**	**0.10**	**32.80**
max	**6840.10**	**205216.00**	**0.20**	**33.00**
std	**6.84**	**204.82**	**0.04**	**0.08**

High-end

2 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**3479.68**	**104402.20**	**0.26**	**16.62**
min	**3478.34**	**104362.00**	**0.20**	**16.50**
max	**3482.50**	**104486.00**	**0.30**	**16.70**
std	**1.70**	**50.73**	**0.05**	**0.08**

4 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**6916.45**	**207509.80**	**0.30**	**32.90**
min	**6884.56**	**206547.00**	**0.20**	**32.80**
max	**6949.31**	**208504.00**	**0.40**	**33.00**
std	**29.31**	**884.12**	**0.10**	**0.07**

8 Threads				
	events/s	**Total events**	**Kern. CPU (%)**	**User CPU (%)**
average	**10083.35**	**302519.80**	**0.34**	**66.92**
min	**10065.28**	**301976.00**	**0.30**	**66.80**
max	**10110.19**	**303325.00**	**0.40**	**67.00**
std	**17.39**	**521.40**	**0.05**	**0.08**

File I/O Tests

Similar to the VM testing, a total of 30 sysbench file I/O tests were conducted for the container testing. The performance data collected for each test is the same as the VM testing with the netdata tool being used to collect disk utilization.

Low-end

Sequential Write				
	Read (MiB/s)	**Write (MiB/s)**	**Latency (ms)**	**Disk Util. (%)**
average	-	**144.90**	**0.05**	**99.80**
min	-	**143.48**	**0.05**	**99.00**
max	-	**145.70**	**0.05**	**100.00**
std	-	**0.87**	**0.00**	**0.45**

Sequential Read				
	Read (MiB/s)	**Write (MiB/s)**	**Latency (ms)**	**Disk Util. (%)**
average	**488.72**	-	**0.03**	**99.80**
min	**486.80**	-	**0.03**	**99.00**
max	**489.40**	-	**0.03**	**100.00**
std	**1.09**	-	**0.00**	**0.45**

Mid-end

Sequential Write				
	Read (MiB/s)	**Write (MiB/s)**	**Latency (ms)**	**Disk Util. (%)**
average	-	**144.52**	**0.05**	**99.70**
min	-	**137.75**	**0.05**	**99.00**
max	-	**146.91**	**0.05**	**100.00**
std	-	**3.84**	**0.00**	**0.45**

Sequential Read				
	Read (MiB/s)	**Write (MiB/s)**	**Latency (ms)**	**Disk Util. (%)**
average	**489.19**	-	**0.03**	**99.94**
min	**488.92**	-	**0.02**	**99.70**
max	**489.46**	-	**0.03**	**100.00**
std	**0.22**	-	**0.00**	**0.13**

High-end

Sequential Write				
	Read (MiB/s)	**Write (MiB/s)**	**Latency (ms)**	**Disk Util. (%)**
average	-	**151.29**	**0.05**	**100.00**

min	-	**147.64**	**0.04**	**100.00**
max	-	**157.84**	**0.05**	**100.00**
std	-	**4.75**	**0.01**	**0.00**

Sequential Read				
	Read (MiB/s)	**Write (MiB/s)**	**Latency (ms)**	**Disk Util. (%)**
average	**488.76**	-	**0.03**	**99.90**
min	**487.14**	-	**0.03**	**99.50**
max	**489.42**	-	**0.03**	**100.00**
std	**0.92**	-	**0.00**	**0.22**

QEMU VM vs. Docker Container Performance

A comparison between performance results of each virtualization technology can be found in the tables below. The “High-end” scenario was chosen for comparison because it scales proportionally to the number of threads selected for the sysbench tests.

VM CPU Performance vs. Container CPU Performance						
	QEMU VM High-end			Docker Container High-end		
Averages	**2 Threads**	**4 Threads**	**8 Threads**	**2 Threads**	**4 Threads**	**8 Threads**
events/s	3815.36	7614.06	10926.84	3479.68	6916.45	10083.35
total events	114468	228437	327827	104402	207509	302519.80
Kernel CPU (%)	0.3	0.18	0.18	0.26	0.30	0.34
User CPU (%)	16.72	33.16	66.76	16.62	32.90	66.92

Overall, the QEMU VM is the better performing on the sysbench CPU tests. While the container scales proportionally to the number of threads used for the test, it is still 7-9% slower than the VM. All VM scenarios were run with the KVM accelerator (reduces overhead to near host-OS level), with only the RAM and CPU cores varied. The container scenarios had the CPU cores and RAM varied in the same amount as the VM. Therefore, the hypothetical performance between the two should have been the same. However, one possible reason for the difference in performance is due to the *zyclonite/sysbench* image. This image could incorporate some configurations that hinder performance as containers are typically supposed to be faster than VMs.

VM I/O Performance vs. Container I/O Performance				
	QEMU VM High-end		Docker Container High-end	
Averages	**Seq. Write**	**Seq. Read**	**Seq. Write**	**Seq. Read**
Read (MiB/s)	-	412.42	-	488.76
Write (MiB/s)	64.00	-	151.29	-
Latency (ms)	0.11	0.04	0.05	0.03
Disk Util. (%)	38.00	98.80	100.00	99.90

The I/O performance results are very interesting. Both the read and write performance is significantly faster on the container. The read performance is approximately 15% faster on the container and the write performance is more than twice as fast on the container. These performance differences make sense when considering the amount of storage allocated to the VM. The container has access to the entire storage capacity of the host OS's drive which in this case is a SATA3 SSD. The VM is allocated a small fraction of this capacity and due to SSD technology limitations it would have slower I/O throughput (specifically write speeds).