Nishant Shah (B00659531)

Assignment 1 – Lighweight vs. Heavyweight Virtualization Techniques

Google Cloud Instance Setup:

- Request coupon for Google Cloud Platform credits by clicking link provided, then entering University email ID
- Then redeem the coupon via normal gmail account
- Go to Vm Instances, click **Create Instance** button and create with following specifications:
  - **Machine type**: 2 vCPU and change memory size to 4 GB
  - In Boot disc section, click change and in **OS image**, select Ubuntu 16.04 LTS and change disc **size** to 20 GB
  - Keep other options as it is for now and click **Create** button.
- We can remotely connext to the instance via Browser directly.
- For cleaning up and avoiding unwanted charges to our account, we should use 'exit' or 'logout' command and then select instance and stop it.

Main steps to enable Docker container and important operations:

- We will install Docker CE by setting up Docker's repositories amd installing from them
- Update 'apt' package index by:
  - $ sudo apt-get update
- Install packages to allow 'apt' to use repository over HTTPS:
  - $ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl
    software-properties-common
- Add Docker's official GPG Key:
  - $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –
- Verify that we have key with fingerprint by searching last 8 characters of fingerprint:
  - $ sudo apt-key fingerprint 0EBFCD88
- Setup stable repository for amd64:
  - $ sudo add-apt repository \
    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) \
    stable"
- Install latest version of docker:
  - $ sudo apt-get install docker-ce

- Verify that Docker CE is installed correctly by running hello-world image:
    - $ sudo docker run hello-world
- The above command downloads a test image and runs it in a container. It prints Hello message and exits.
- Other important Operations:
    - $ sudo docker pull <image_name> - used to pull a docker image
    - $ sudo docker images – used to see currently installed images on our system
    - $ sudo docker run <image_name> - Docker daemon checks if image is already downloaded, if no it downloads and then runs the image
    - $ sudo docker run <image_name> ls -l – this command runs the I,age and then shows the listings
    - $ sudo docker ps – shows all currently running containers
    - if we add -a to above command, it shows all the containers that recently exited

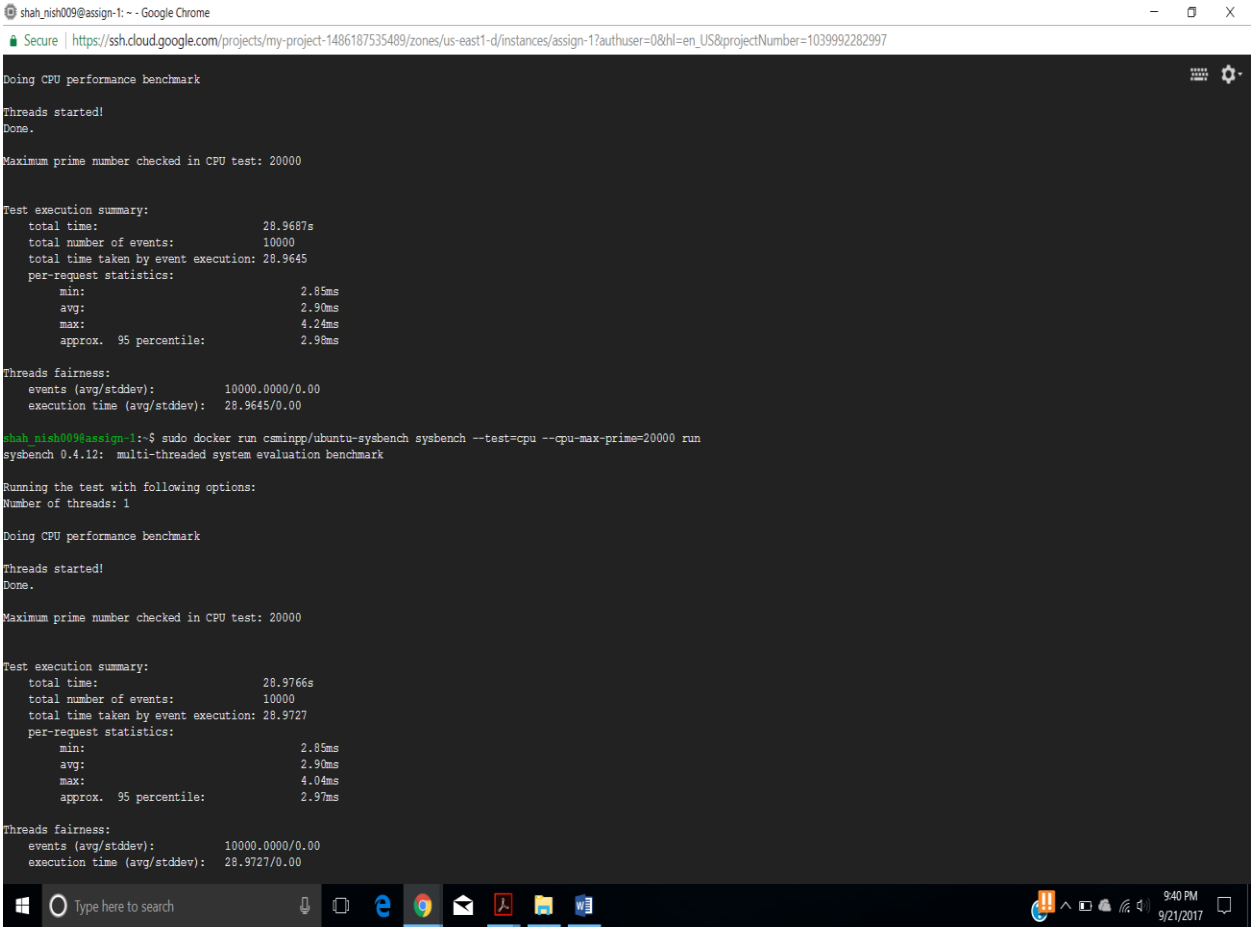
Main steps to enable QEMU VM with QEMU commands and VM configurations:

- Install QEMU by : sudo apt-get install qemu
- Download Ubuntu ISO image: $wget http://mirror.pnl.gov/releases/16.04/ubuntu-16.04.3-server-amd64.iso
- Create image before installing: $sudo qemu-img create ubuntu.img 10G
- Installing GUI for ubuntu on GCP:
    - install gnome components: $ sudo apt-get install gnome-core
    - install vnc server for interacting: $ sudo apt-get install vnc4server
    - start vnc server: $ vncserver – and set new passweord when prompted to.
    - open and edit startup file: $ vim .vnc/xstartup
    - #!/bin/sh
      def
      export XKL_XMODMAP_DISABLE=1
      unset SESSION_MANAGER
      unset DBUS_SESSION_BUS_ADDRESS
      metacity &
      gnome-settings-daemon &
      gnome-panel &
      nautilus &
      gnome-terminal &
    - navigate to Firewall rules in GCP and create a new firewall rule:
    - name: vnc-server
      network: default
      priority: 1000
      direction of traffic: ingress
      action on match: allow
      targets: all instances in the network

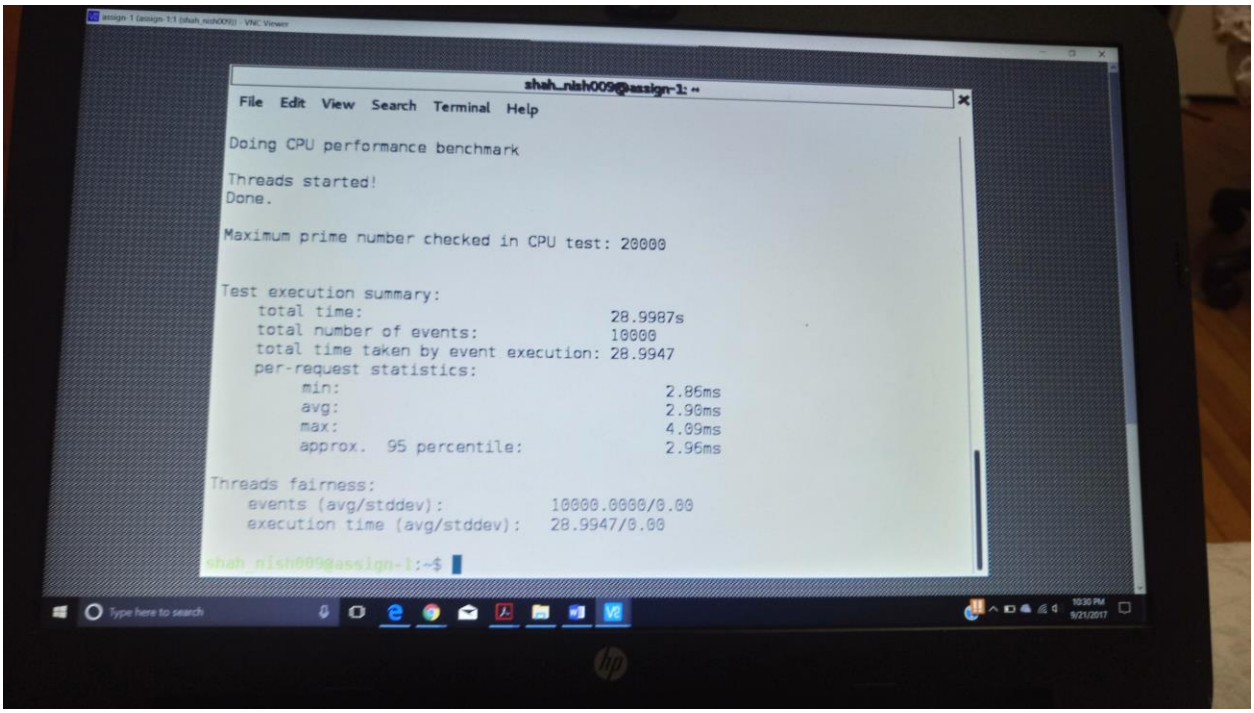
source filter: IP ranges
source IP ranges: 0.0.0.0/0
second source filter:none
protocols and ports: specified protocols and ports
tcp:5000-10000

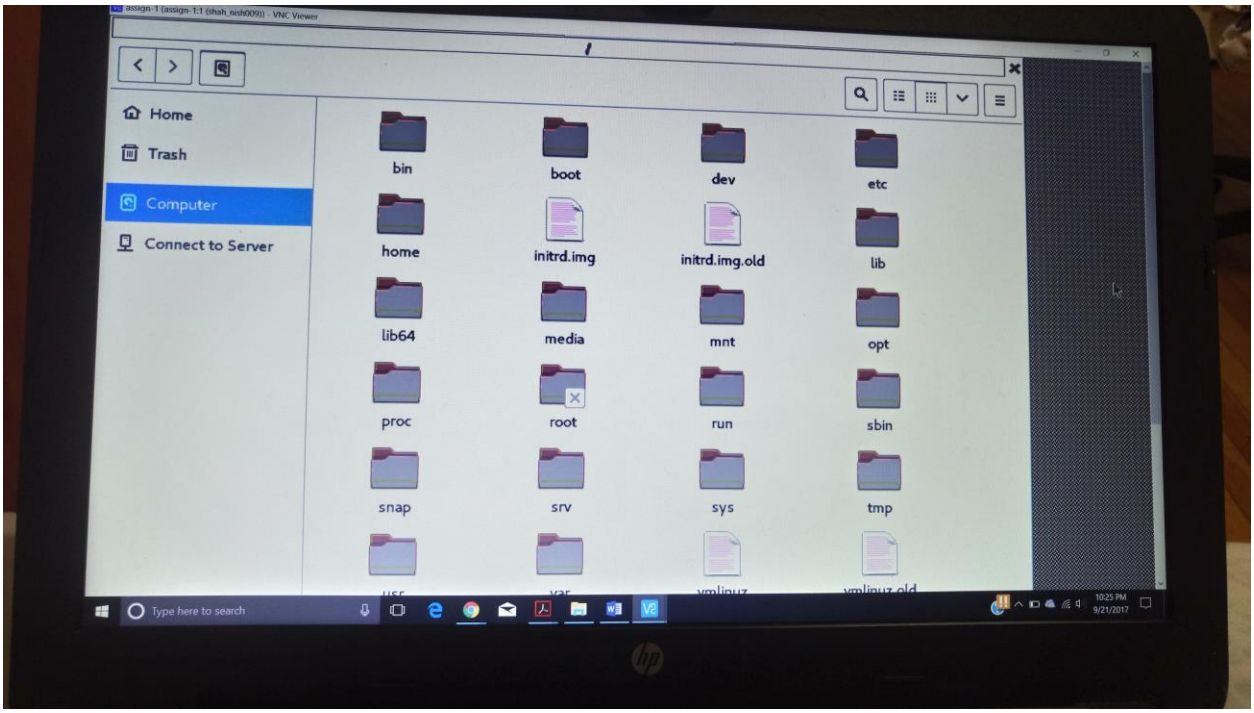
- click **create** button
- download vnc viewer
- start vnc server: $ vncserver
- connect it to gcp instance by: $ nc <external_ip> 5901
- start vnc viewer, make new connection using <external_ip> and vncviewer password
- This will show GUI for our GCP instance
- to stop vncserver anytime: vncviewer -kill :<task_number>
- open terminal from the recently created GUI
- then install QEMU by: $sudo qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom ./ubuntu-16.04.3-server-amd64.iso -m 1536
- Give following information/permissions when prompted:
  - username, password and language settings
  - keyboard and clock/time settings
  - disc partition settings
  - http proxy server could be left empty for now
  - make this system as root
- This will install ubuntu on QEMU VM and can access as GUI by VNC Server

Docker Running Environment:



QEMU Running Environment:





CPU Benchmarking:

For Native, following command:

- sysbench --test=cpu --cpu-max-prime=20000 run

For Docker, following command:

- sudo docker run csminpp/ubuntu-sysbench sysbench --test=cpu --cpu-max-prime=20000 run

For QEMU, following:

- open QEMU terminal
- sysbench --test=cpu --cpu-max-prime=20000 ru

TEST 1 for CPU Benchmarking:

Threads: 1 Max Prime: 20000 Number of events: 10000

Native (time in sec)	Docker (time in sec)	QEMU (time in sec)
29.8022	29.0252	29.0462

TEST 2 for CPU Benchmarking:

Threads: 1 Max Prime: 20000 Number of events: 10000

Native (time in sec)	Docker (time in sec)	QEMU (time in sec)
29.1136	28.9687	28.9716

TEST 3 for CPU Benchmarking:

Threads: 1 Max Prime: 20000 Number of events: 10000

Native (time in sec)	Docker (time in sec)	QEMU (time in sec)
29.0926	28.9766	28.9987

FileIO Benchmarking:

For Native, following commands:

- sysbench --num-threads=1 --test=fileio --file-total-size=5G --file-test-mode=rndrw prepare
- sysbench --num-threads=1 --test=fileio --file-total-size=5G --file-test-mode=rndrw run
- sysbench --num-threads=1 --test=fileio --file-total-size=5G --file-test-mode=rndrw cleanup

For Docker, following commands:

- sudo docker run csminpp/ubuntu-sysbench sysbench --num-threads=1 --test=fileio --file-total-size=5G --file-test-mode=rndrw prepare
- sudo docker run csminpp/ubuntu-sysbench sysbench --num-threads=1 --test=fileio --file-total-size=5G --file-test-mode=rndrw run
- sudo docker run csminpp/ubuntu-sysbench sysbench --num-threads=1 --test=fileio --file-total-size=5G --file-test-mode=rndrw cleanup

For Docker, following:

- open QEMU terminal
- sudo docker run csminpp/ubuntu-sysbench sysbench --num-threads=1 --test=fileio --file-total-size=5G --file-test-mode=rndrw prepare
- sudo docker run csminpp/ubuntu-sysbench sysbench --num-threads=1 --test=fileio --file-total-size=5G --file-test-mode=rndrw run
- sudo docker run csminpp/ubuntu-sysbench sysbench --num-threads=1 --test=fileio --file-total-size=5G --file-test-mode=rndrw cleanup

Threads: 1 File Size: 5 GB

Test 1 for FileIO Benchmarking:

Native	Docker	QEMU
32.4128	31.9428	32.0047

Test 2 for FileIO Benchmarking:

Native	Docker	QEMU
32.5178	32.0048	32.5469

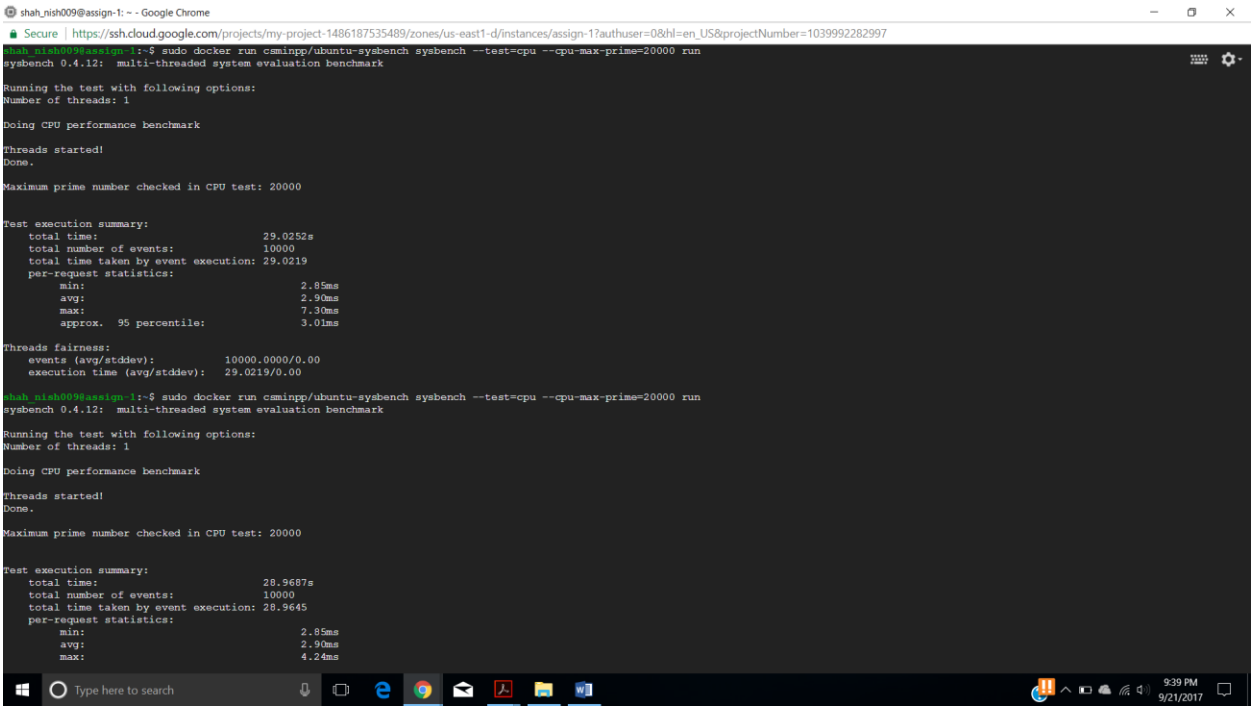
Test 3 for FileIO Benchmarking:

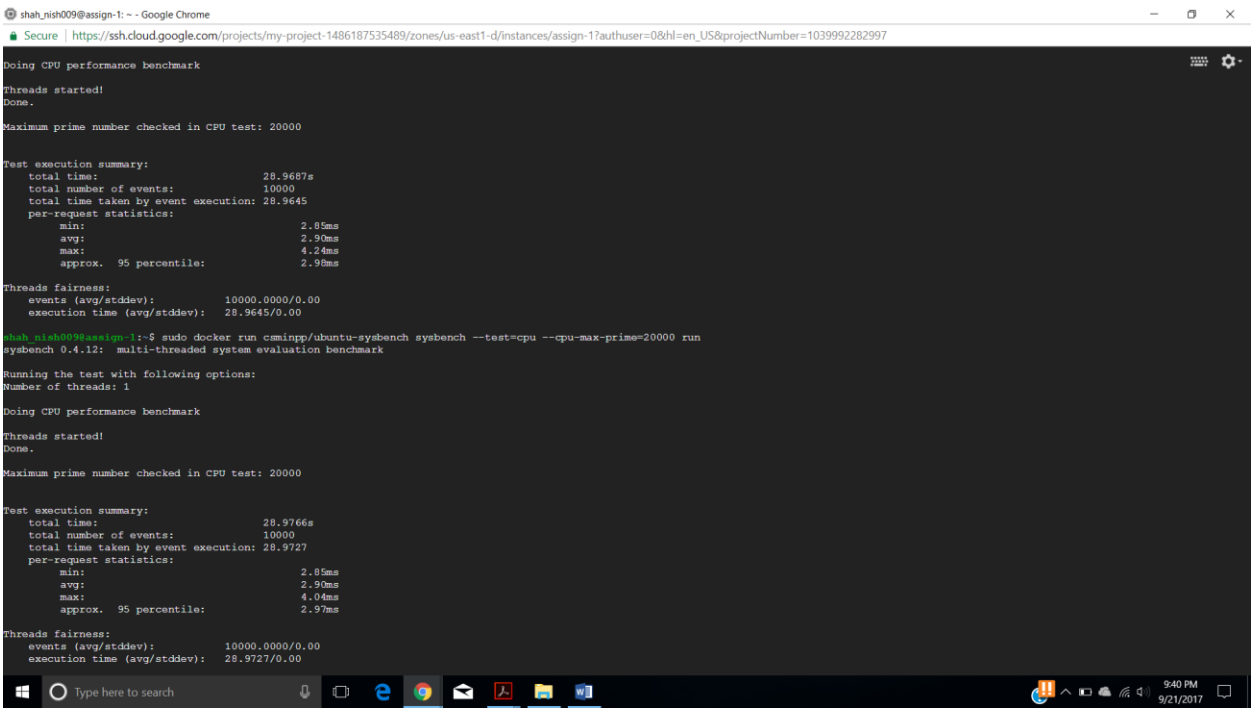
Native	Docker	QEMU
32.4289	32.0458	32.3498

OVERALL OBSERVATION FOR TESTS:

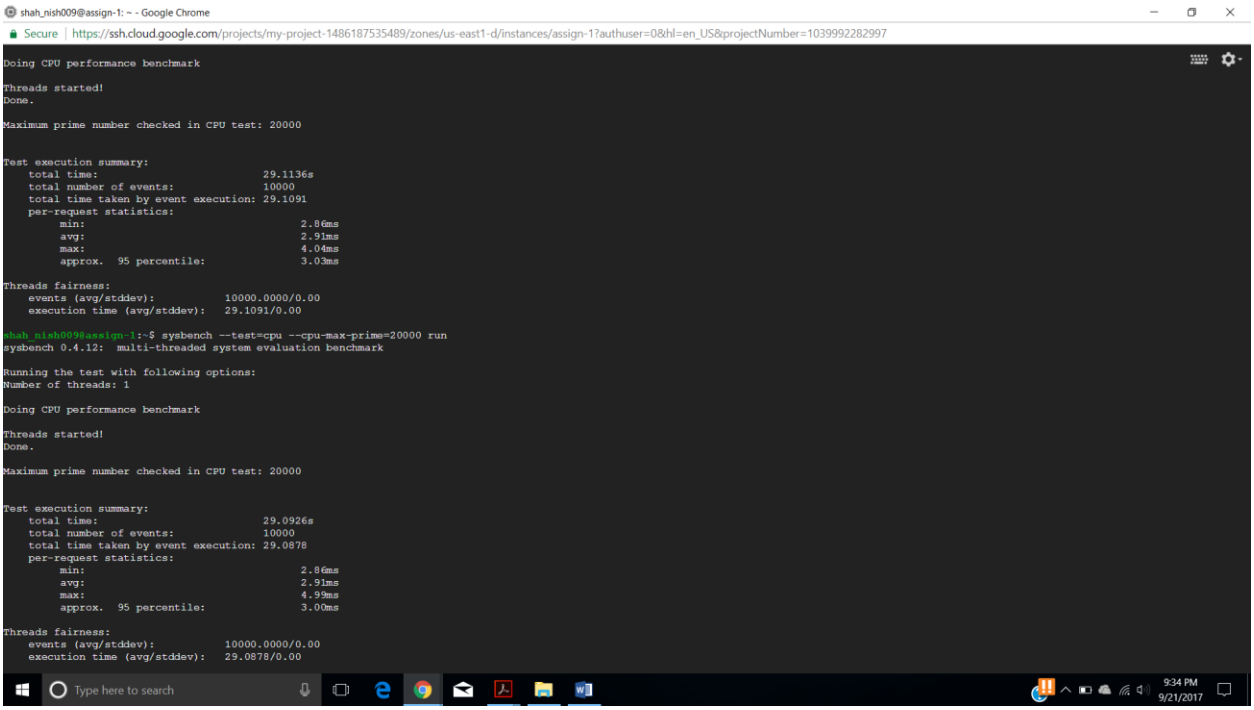
- From the above test results, we observe it takes around 29 seconds for CPU to do 10000 comparisons (max prime of 20000) for all the three methods
- If we see in detail, Docker and QEMU yeilds better results compared to the Native method. Why?
- Thos is where virtualization plays a role.
- While in Docker (Lightweight), we use pre-defined images with the functionality we want whicj restricts it's usage to specific type of tasks
- QEMU (heavyweight) virtualization, we only install the components needed, so it makes the system fast.
- Sameway with the FILEIO operations
- generally it takes around 32.5 seconds for 5 GB file for 1 thread.
- Threading improves speed but has a throughput
- Also, it's somewhat better in docker and QEMU because of virtualization.

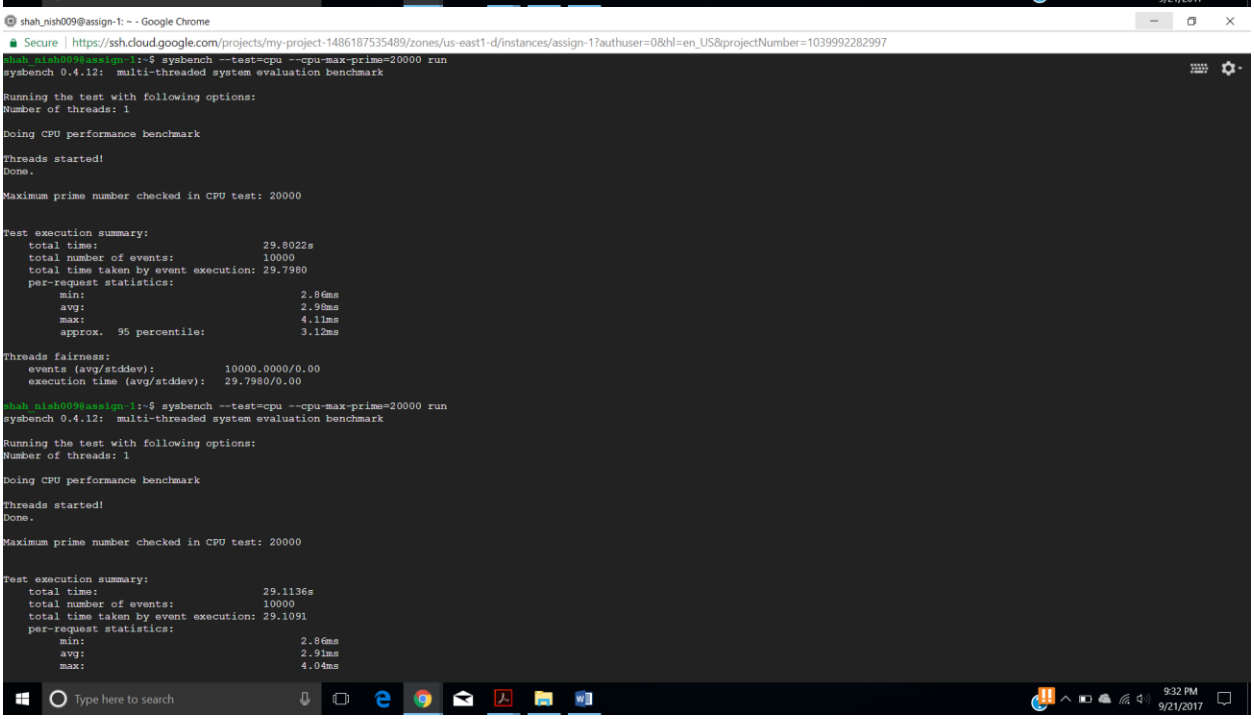
Images for Docker tests:





Images for Native tests:





Images for QEMU Tests: