Virtualization - A highlight on performance

V1.1 – Guillaume Urvoy-Keller

January 15, 2021

Important

- You must have done the first 7 questions of section 1 before the actual lab session
- The report must include screenshots showing your progress.

1 Vagrant VMs: Warm up

1. Install Vagrant on your machine. The lab works if the provider is Virtualbox. Other providers are possible, but this is under your responsibility.
2. Create a Vagrant folder and move to this folder
3. Browse the Atlas repository looking for a recent ubuntu box at `https://app.vagrantup.com/boxes/search`.
   Note that it must be an official box, i.e. a box pushed by the ubuntu community. Boxes are indexed with a repository name. The boxes pushed by the Ubuntu community are in the Ubuntu repository, `https://app.vagrantup.com/ubuntu`
4. Download the box by typing on the command line (adjust with the latest ubuntu release):
   vagrant box add ubuntu/xxx64
5. Check that the box is there by typing:
   vagrant box list
6. Create your first VM:
   (a) Create a Ubuntu1 folder under the Vagrant folder and move to this folder
   (b) Type: **vagrant init ubuntu/xxxx64**. The command creates a file called Vagrantfile that describes the VM parameters.
   (c) Start the machine by typing: **vagrant up**
   (d) If everything is ok, you should be able to ssh to the machine via a simple **vagrant ssh**
   (e) Check with the virtualbox GUI the type of network interface does the machine have?
   (f) What is the brand of the NIC of the machine?
   (g) When checking the user manual of Virtualbox concerning network interfaces (should be `https://www.virtualbox.org/manual/ch06.html#nichardware`), you observe that a limited number of interface types are available (forget about virtio-net). Explain why it is the case.
7. We are now going to modify the VM by
   - Installing a set benchmarks packages: sysbench (for CPU) and iperf (for network).
   - Adding a second interface in bridge mode
   - Forwarding 5001 port (the iperf server port) on the initial interface.
   - Installing docker and openvswitch

All these actions can be done via the Vagrantfile.

(a) Stop the VM by first exiting if you are connected to it and then from the command line: **vagrant halt**

(b) Edit the Vagrantfile and do the following:

- Uncomment the line with

  ```
  config.vm.network "public_network"
  ```

  so as to create a bridge interface. Normally, if everything is fine, i.e. the DHCP on the wifi interface of your machine is working appropriately, it should be ok. However, if you have doubts (for instance in the case of the unice hotspot!!), you can:

  - Assign an address manually on the public network

    ```
    config.vm.network "public_network", ip: "10.0.0.1"
    ```

  - Bridge on your Ethernet card and not your Wifi card (it works even if the Ethernet cable is not connected)

- Uncomment and modify the forwarded port line:

  ```
  config.vm.network "forwarded_port", guest: 5001, host: 5001 , auto_correct:  true
  ```

- Uncomment and modify the following lines that concern the provisioning of the machine:

  ```
  config.vm.provision "shell", inline: <<-SHELL
      sudo apt-get update
      sudo apt-get install -y sysbench iperf openvswitch-switch
      wget -qO- https://get.docker.com/ | sh
    SHELL
  ```

(c) Validate the modified vagrant file with **vagrant validate**.

(d) Restart the machine with **vagrant up**

(e) Provision the machine by typing: **vagrant provision**. Check the interfaces and the presence of the new packages with **dpkg -s package_name**. Check that Docker is available by running the hello world container: **sudo docker run hello-world**. You should see first the download of the many layers of the container from the docker hub (equivalent to Atlas), then the output of this simple container:

```
$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b901d36b6f2f: Pull complete
0a6ba66e537a: Pull complete
Digest: sha256:8be990ef2aeb16dbcb9271ddfe2610fa6658d13f6dfb8bc72074cc1ca36966a7
Status: Downloaded newer image for hello-world:latest

Hello from Docker.
This message shows that your installation appears to be working correctly.
[...]
```

2 Vagrant VMs: (moderately) advanced features

1. Vagrant is using Virtualbox has a provider in our set-up. Let us check what Virtualbox does when one suspends (to disk) a running VM.

   (a) First, locate the directory in which Virtualbox is maintaining the files of the Ubuntu1 VM. You can locate it by checking the GUI of Virtualbox for Ubuntu1. It should be something like $\sim /VirtualboxVM/Ubuntu1[....]$.

   (b) List the content of this directory

(c) Then instruct Virtualbox to pause the machine with a **vagrant suspend**.

(d) Check the directory again and infer which new elements has been added by Virtualbox. Given the name of this element, what is the technique used by the hypervisor here? Does it make sense with the objective (saving the execution state of the machine)?

(e) What do you expect to find in the newly created file (no need to edit it, it is a binary file)? item Restart the VM with **vagrant resume** (that will restart the machine from its saved state) or **vagrant reload** (that discards the saved state and do a normal reboot)

2. Let us now create an image out of the current VM.

   (a) Stop the VM **vagrant halt**

   (b) Dump the current VM: **vagrant package --output Myubuntu**

   (c) Add the box: **vagrant box add --name ubuntu_docker Myubuntu**

   (d) Check that the box is now available with **vagrant box list**

3 Containers and Namespaces

Use your Ubuntu1 VM or a second VM created out of the image created at the end of the previous section if your want.

1. . Containers are sets of processes packed together with their own interfaces, file system, etc. Just like Vagrant, a repository exists where one can push/pull container images.

   (a) Browse the docker hub `https://hub.docker.com/`

   (b) Find the official ubuntu repository and download the latest version: **docker pull ubuntu**. You must be root to run docker. The vagrant user is in the sudo group.

   (c) Check the image is correctly downloaded with **docker images**

2. Let us a start a ubuntu container: **docker run -ti --name=ubuntu ubuntu /bin/bash**.

   (a) What is the process id (PID) of the bash process inside the container (**echo $$** to get its PID)?

   (b) We want to look at the PID of this bash from within the VM (i.e., outside the container). For this, let us first leave the container without stopping it. Do this by typing *ˆP ˆQ* (Ctrl+P Ctrl+Q). Then use **docker top ubuntu** to find the PID of the bash within the VM.

   (c) Using the PID of the process (called $PID in the next command), we can find all the namespaces associated to the container under /proc/$PID/ns/. The namespace numbers are presented as symbolic links. You need to do a **ls -al** to see them.

   (d) Show that the namespaces of the current shell in which you are logged (i.e., outside the container) are the same as the init process (whose PID is 1) and differ from the ones of the bash within the ubuntu container.

4 CPU benchmarking

4.1 Vagrant / Virtualbox

The CPU benchmark we consider consists in an algorithm to compute prime number. You run it with:

- With sysbench 0.4.x

```
sysbench --test=cpu  --cpu-max-prime=10000 --num-threads=1 run | grep "total time:"
```

- With sysbench 1.x

```
sysbench cpu  --threads=1 --max-requests=10000 run | grep "total time:"
```

It performs 1000 independent computations. If it runs over N CPUs, its duration should be divided by N **if** you adjust the number of threads (via the parameter –num-threads=) to N.

1. Modify the configuration of your VM by editing the Vagrantfile, find the part on Virtualbox provisioning that start with

   ```
   config.vm.provider "virtualbox" do |vb|
   ```

   Vary the number of vCPUs with the parameter

   ```
   vb.cpus
   ```

   Consider values 1 and 4 (to adjust depending on your hardware). After each change in the Vagrantfile, do a **vagrant reload**. Check that the configuration was effective by checking the GUI of Virtualbox or, from inside the VM, check the number of cpu with **more /proc/cpuinfo**.

2. For each configuration, perform 10 tests and report the raw values as well as the average in a table.

3. If your host is a linux box, compare the performance of sysbench on your virtual machine and your physical host. **Even if you can't do the test on the physical machine because you have a Windows box**, you should be able to infer the result by looking at the kind of workload sysbench is generating and referring to the course.

4.2 Docker

Remark: with docker, you must kill a stopped container before restarting it, i.e. if you have done a **docker run -ti --name=ubuntu ubuntu /bin/bash** and then exited from this container, you must do a docker rm ubuntu before doing a **docker run -ti --name=ubuntu ubuntu /bin/bash** again.

1. Provision the VM with 2 vCPU.

2. Run our first container: **docker run -ti --name=ubuntu ubuntu /bin/bash**, install the sysbench package and benchmark the CPU with sysbench.

3. How many cpus sees the container?

4. Compare the performance with the one of the (host) VM if you can and comment.

5. Stop the container (type **exit**). Check the active containers with **docker ps** and also the list of stopped containers with **docker ps -a** (you should see your recently killed container) and delete it: **docker rm ubuntu**

6. Restart a similar container with the command:

   ```
   docker run -ti --cpus=1 --name=ubuntu ubuntu /bin/bash
   ```

   and redo the test. What do you obtain (explain!) ?

7. The CPU limitation is implemented using cgroups by Docker. Cgroups are visible as a file system mounted in the /sys/fs/cgroup directory. Under this directory, you see sub-directories that correspond to hierarchies.

   (a) Which hierarchy shall we focus on for the above Docker container according to you?

   (b) Under a hierarchy, there is a always a file called cgroup.procs that lists the processes falling under. Find the process id of the bash of the docker container and check that it is *NOT* in the cgroup.procs of the hierarchy you think

   (c) Explain how Docker implements itself in the hirearchy, by searching in the subdirectories of the hierarchy using the following hint: the internal name used by docker for your container is (partially) listed when using the **docker ps** command.