**LAB 1**

**RESOURCE VIRTUALIZATION**

|  |
| --- |
| Fullname: Trần Minh Khang  Student ID: B1606805 |

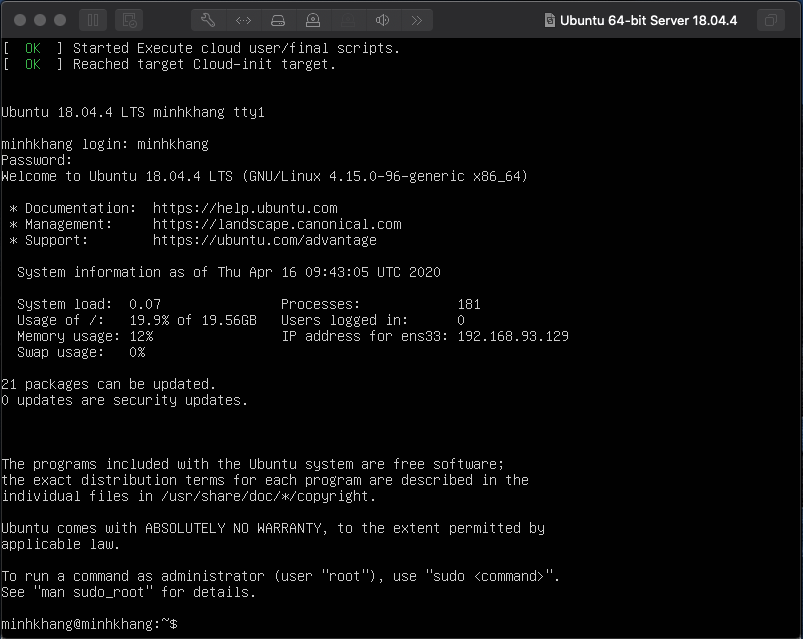
* Note: screenshots need to be clear and good-looking; submissions must be in PDF format.

1. **Ubuntu installation**
   1. Create a virtual machine using VirtualBox/VMWare.

* The network setting of the VM is bridge mode.
* Enable PAE/NX and Enable Nested VT-x/AMD-v (Setting/System/Processor). If the option is grey out, entering the following command

$ VBoxManage modifyvm "vm name" --nested-hw-virt on

* 1. Install Ubuntu server 18.04 LTS as the OS of the VM. (take a screenshot after finishing the installation)



1. **Machine level virtualization with KVM + Libvirt + Kimchi 2.5**

* *Kernel-based Virtual Machine (KVM)* is a virtualization module in the Linux kernel that allows the kernel to function as a hypervisor
* The *libvirt* library is used to interface with different virtualization technologies.
* *Kimchi*, which is a free, open-source web-based virtualization management platform so we can manage our hypervisor easily through the web browser.
  1. Make sure your hardware supports the necessary virtualization extensions for KVM.

$sudo apt install cpu-checker

$kvm-ok

A message will be printed informing us if our CPU *does* or *does not* support hardware virtualization.

* 1. Install the necessary packages for KVM and Libvirt

$sudo apt update

$sudo apt install qemu qemu-kvm libvirt-bin bridge-utils virtinst

* 1. Add the user used to manage virtual machines will need to be added to the libvirt group

$sudo adduser $USER libvirt

$su - ${USER}

After the installation we can use commands such as virt-install, virsh list, virsh start, virsh reboot, virsh shutdown, etc. to create, list, start, reboot, shutdown VMs. (optional, try it by yourself)

* 1. Install the necessary packages

$sudo apt update

$sudo apt install python-paramiko python-pil novnc python-libvirt python-ethtool python-ipaddr python-guestfs libguestfs-tools spice-html5 spice-html5 python-magic keyutils libnfsidmap2 libtirpc1 nfs-common rpcbind python-configobj python-parted -y

**Note:** If you receive an error about packages not found. You must enable the Universe Repository by running.

$sudo add-apt-repository universe

$sudo apt update

* 1. Install Kimchi

# Install nginx

$sudo apt install nginx -y

# Install Wok

$wget https://github.com/kimchi-project/kimchi/releases/download/2.5.0/wok-2.5.0-0.noarch.deb

$sudo apt install -y ./wok-2.5.0-0.noarch.deb

#Install Ginger (optional)

sudo apt-get install hddtemp python-libuser python-ethtool python-augeas python-ipaddr python-magic python-netaddr python-crypto python-apt

$wget $http://kimchi-project.github.io/gingerbase/downloads/latest/ginger-base.noarch.deb

$wudo apt install -y ./ginger-base.noarch.deb

$wget http://kimchi-project.github.io/ginger/downloads/latest/ginger.noarch.deb

$sudo apt install -y ./ginger.noarch.deb

# Install Kimchi

$wget https://github.com/kimchi-project/kimchi/releases/download/2.5.0/kimchi-2.5.0-0.noarch.deb

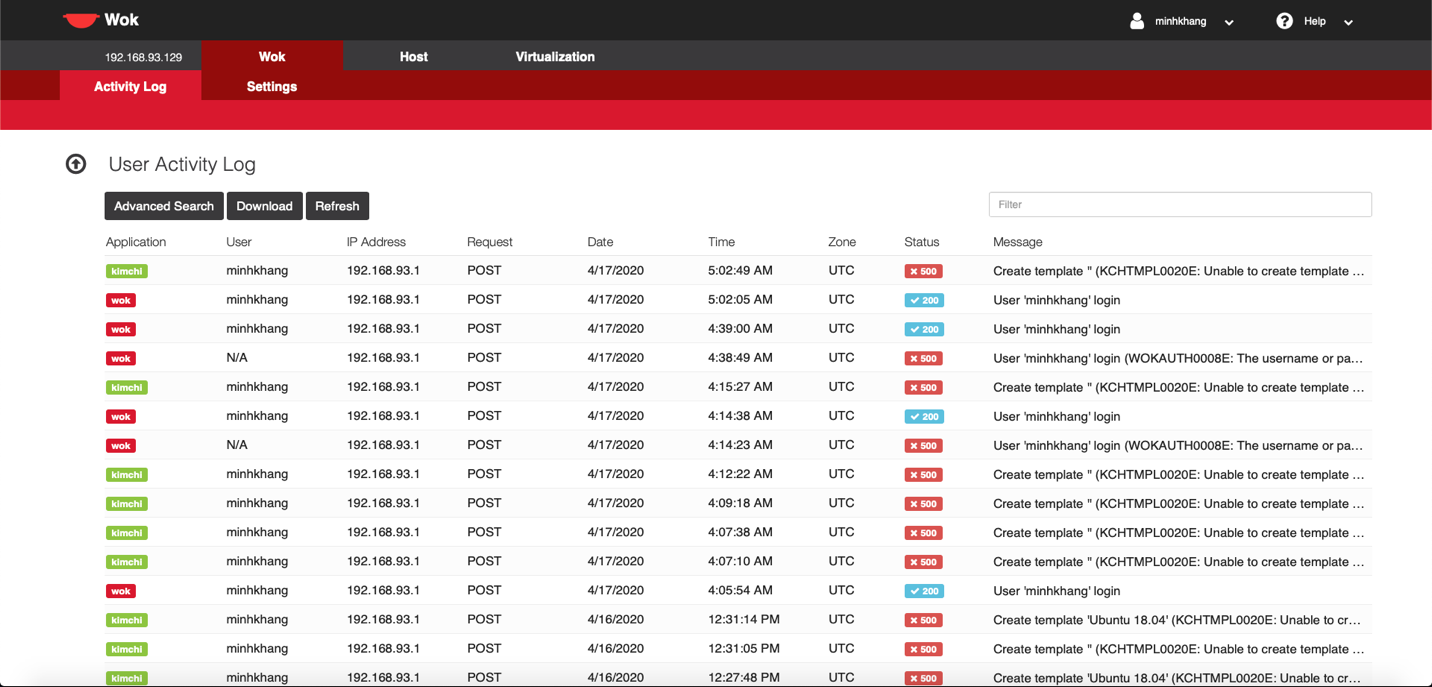
$sudo dpkg --ignore-depends=python-imaging -i kimchi-2.5.0-0.noarch.deb

* 1. **Start the service and enjoy!**

$sudo service wokd start

Access Kimchi at https://<IP of VM>:8001

* 1. **Login and create a VM . Then view the console of the VM. (take a screenshot of the console)**

****

**Note :**

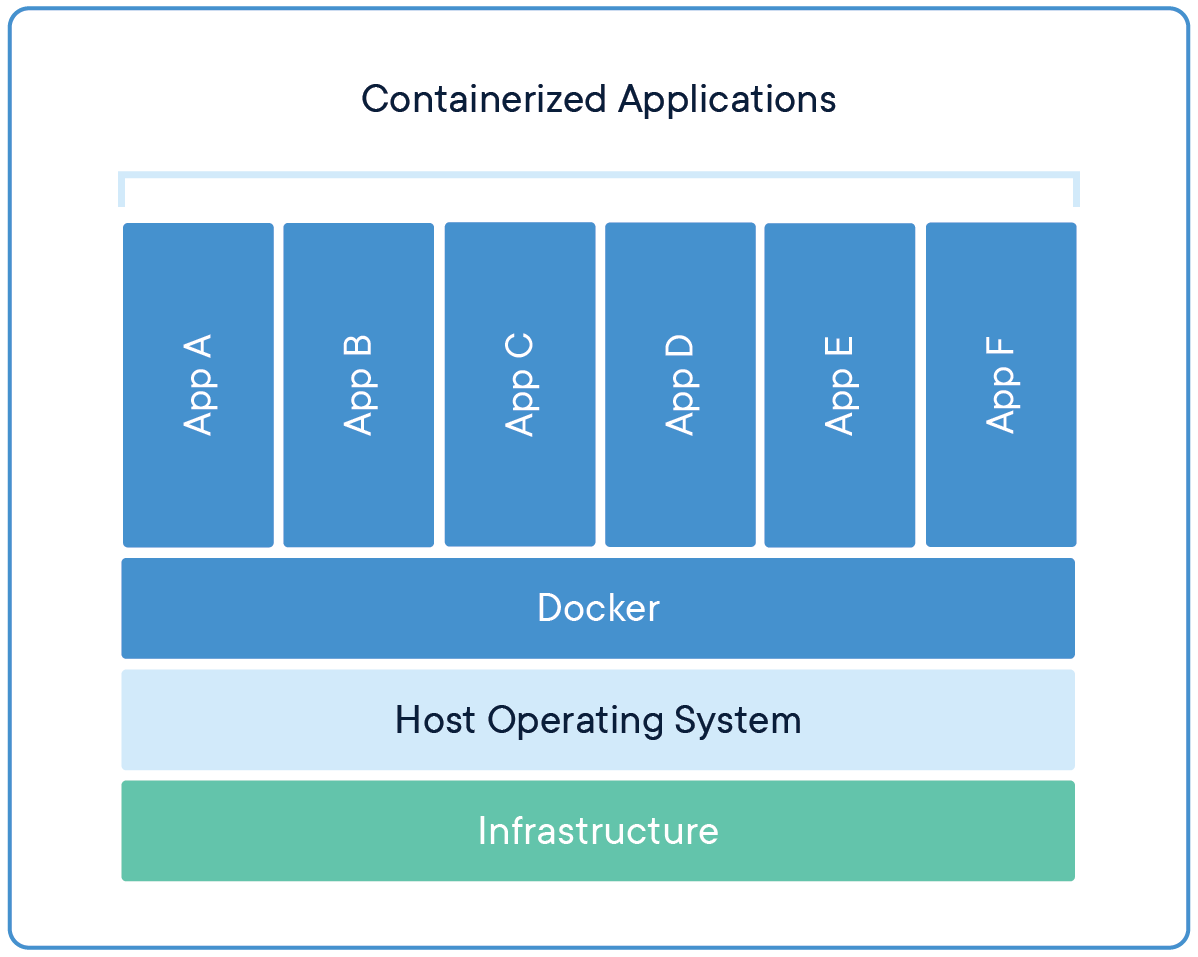
For problems (e.g. missing virtualization tab) troubleshooting, please check:

<https://github.com/kimchi-project/wok/wiki/Troubleshooting>

<https://github.com/kimchi-project/kimchi/wiki/Troubleshooting>

1. **OS level virtualization with Docker**

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

An image is a file that represents a packaged application with all the dependencies needed to run correctly (a container is an instance of an image).

* 1. **Installing Docker**

- Install packages to allow apt to use a repository over HTTPS:

$sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common

- Add Docker’s official GPG key:

$curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

- Add the Docker repository to APT sources:

$sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) stable"

- Update the apt package index:

$sudo apt-get update

- Install the latest version of Docker Engine - Community and containerd

$sudo apt-get install docker-ce docker-ce-cli containerd.io

- Add current user to the docker group:

$sudo usermod -aG docker ${USER}

$su - ${USER}

- Check whether you can access and download images from Docker Hub

$docker run hello-world

The output will indicate that Docker is working correctly:

Hello from Docker!

This message shows that your installation appears to be working correctly.

* 1. **Working with Docker images and containers**

- Search for the Ubuntu image from Docker hub

$docker search ubuntu

- Download the official Ubuntu image

$docker pull ubuntu

- To see the images that have been downloaded

$docker images

- Running a Docker Container using ubuntu image

$docker run -it --name webserver -p 5000:80 ubuntu

-it: gives us interactive shell access into the container

--name webserver: name the container as “webserver”

-p 5000:80: map port 80 of the container to port 5000 of host machine

- Now we can run any command inside the container. Install Apache2 to the container.

#apt update

#apt install apache2

#exit

- To see all containers have been created

$docker container ls -a

* 1. **Commit changes in a container to a new image**

- Start the container *webserver*

$docker start webserver

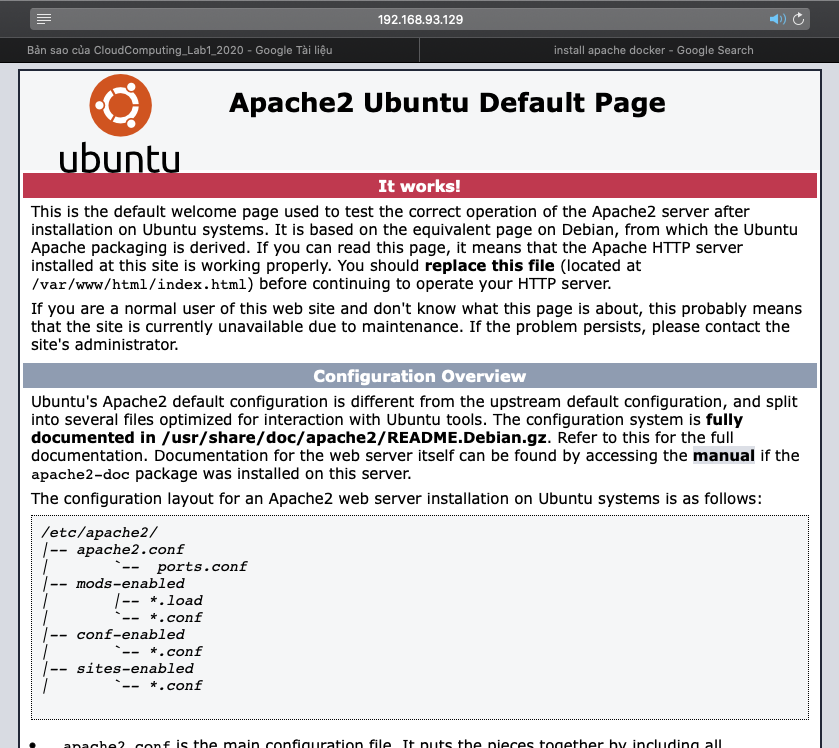
- To get a bash shell in the container *webserver*

$docker exec -it webserver /bin/bash

- Start apache2 of the container

#service apache2 start

- Access Apache2 from a browser ([http://Host\_IP:5555](http://host_ip:5555)). The homepage of Apache must be shown. **(take a screenshot)**

****

- Commit changes in the container to a new docker image

$docker commit -m "added Apache2" -a "tmtuan" webserver tmtuan/ubuntu-apache

-m "added Apache2": description - what you did to the image

-a "tmtuan": author name is "tmtuan"

webserver: container name

tmtuan/ubuntu-apache: repository/new\_image\_name

- To see the images that have been downloaded or created

$docker images

* 1. **Pushing Docker images to a Docker repository**

- To push an image to Docker Hub or any other Docker registry, we must have an account there (<https://hub.docker.com/>)

- Log into Docker Hub

$docker login -u docker-username

You’ll be prompted to authenticate using your Docker Hub password. If you specified the correct password, authentication should succeed.

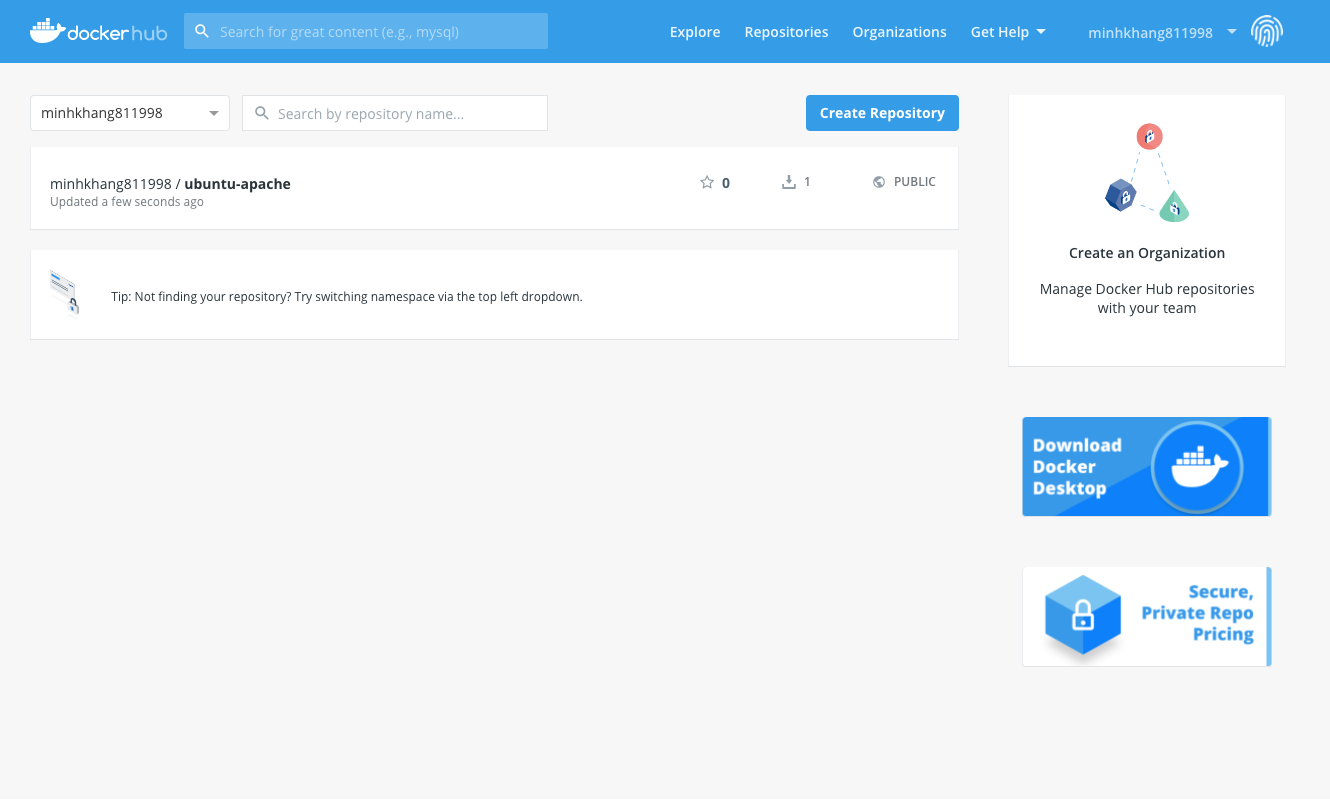
- If your Docker username is different from the local username we used to create the image, we will have to tag our image with your docker username.

$docker tag local-username/ubuntu-apache docker-username/ubuntu-apache

- To push the ubuntu-nodejs image to the docker-username repository, the command would be:

$docker push docker-username/ubuntu-apache

- Go to Docker Hub (<https://hub.docker.com/>) to check if our image is in docker-username repository **(take a screenshot)**

****

---END---