Building a Services Container

(Note: Use this guide in conjunction with the accompanying slide deck!)

Requirements

The end goal is to have a KVM virtual machine with applications (or capable of downloading and running application) that can be installed on a Cisco ISR 4000 series router.

It was decided to have a 4GB RAM sized KVM with 20GB of disk space.

To build a KVM virtual machine requires a Linux development environment with about 2GB more RAM than that needed for the KVM virtual machine, and disk space double the size of the KVM virtual machine plus 15GB. In other words 6GB RAM and 55GB disk space would be adequate for this example.

If you're developing on a PC then you could install the Linux development environment using VirtualBox or VMware Workstation. The latter was used for this example. It requires a PC with at least 8GB of RAM, but preferably 16GB or more.

If you don't have 55GB of local spare disk space then there are some solutions. An external USB connected hard disk/SSD could be used, or network attached storage (NAS). These options are slower though so ideally you want local disk space.

Caution

If you're following this guide, be aware that all characters are case-sensitive in Linux. Also, copy-and-paste doesn't always work, so if you copy-and-paste commands from this document and you immediately get an error then try to manually type the command. This is because this document may have stylized hyphens and quotation marks and other non-standard characters. When pasted into a Linux terminal it might appear as a normal character but actually it is a different character and therefore an error occurs.

Installing Linux

Obtain the long term support (LTS) version of Ubuntu 64-bit and install it on your PC using (say) VMware Workstation. With your newly created virtual machine powered off select 'Edit virtual machine settings' and click on the Processors selection and then select 'Virtualize Intel VT-x/EPT or AMD-V/RVI', and click OK.

Power up the VM. Click on the cog icon at the top-right and then System Settings. Select 'Text Entry'. Click on the '+' symbol and choose your keyboard for example 'English (UK)' and then close the dialog.

Click on the Ubuntu icon and type 'terminal' to open up a shell.

Type

```
egrep -c '(svm|vmx)' /proc/cpuinfo
```

and confirm the response is 1 or more.

Type the following and enter your password as requested:

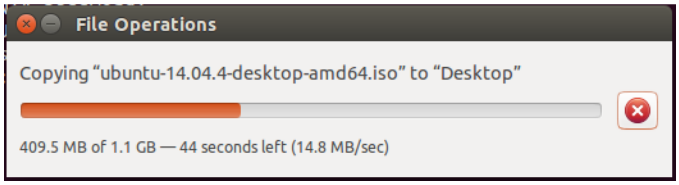
```
sudo su
```

Next, type the following and choose a root user password:

```
passwd
```

You also need to set up a shared folder so that you can access files between your laptop host and the VM. This is a setting in VMware Workstation for your VM (settings->Options-> Shared Folders) but it sometimes doesn't work if your copy of VMware Workstation is older. In that case, you can use drag-and-drop; that usually works.

You will need access to the Ubuntu install ISO file from inside the VM, so either set up a shared folder to access it, or drag the (1GB) file into your VM's desktop.



It should be on the desktop if that is where you dragged it but if you can't find it, use the find command, e.g. from the / folder:

```
find . -name "ubuntu*.iso" -print
```

2. Installing tools for KVM

Now staying as root user, install packages as shown here:

```
apt-get install qemu-kvm libvirt-bin bridge-utils virt-manager qemu-system
```

Reboot the VM (as root user just type reboot).

Once the machine comes back up, open up a terminal and then type the following:

```
virsh -c qemu:///system list
```

If all is well you should just see this:

```
Id    Name                           State
----------------------------------------------------
```

If you see this then skip on to the next section, 'Creating the KVM image'.

Otherwise if you get a permission denied error then do two thing;

First type the following to add your user to the libvirtd group:

```
adduser bob libvirtd
```

then, a file needs editing. Type the following to see what group number your user is in:

```
cat /etc/group | grep  ^bob
```

The output will be something like this:

```
bob:x:1000:
```

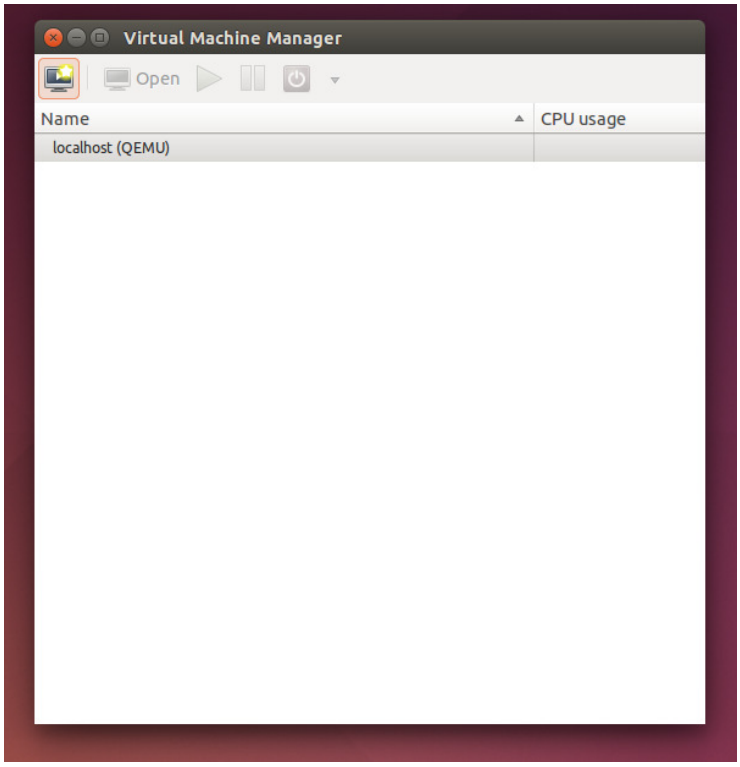
Now (still as root user) edit the file /etc/libvirt/libvirtd.conf and add the following lines if they do not already exist (check!):

```
unix_sock_group = "libvirtd"

unix_sock_ro_perms = "0777"

unix_sock_rw_perms = "0770"
```

After this do a reboot and then reattempt.

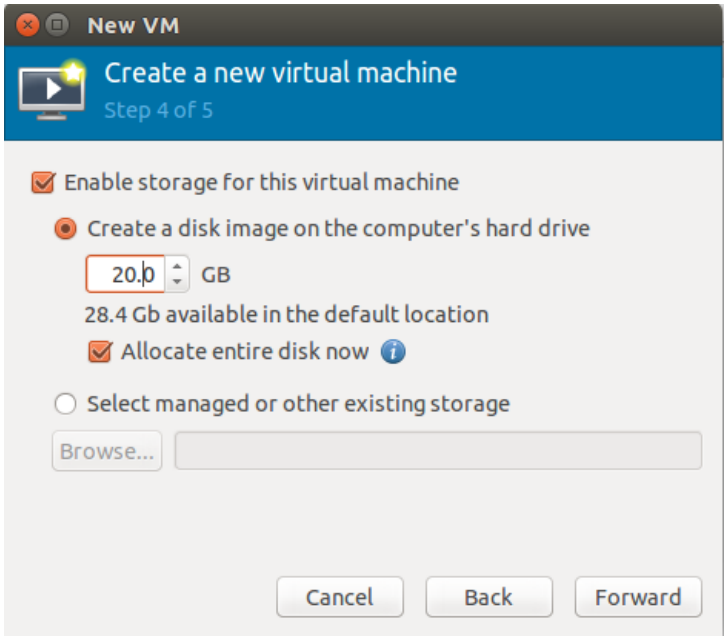
3. Creating the KVM image

Next, click on the Ubuntu icon and type virtual machine and start up the Virtual Machine Manager.

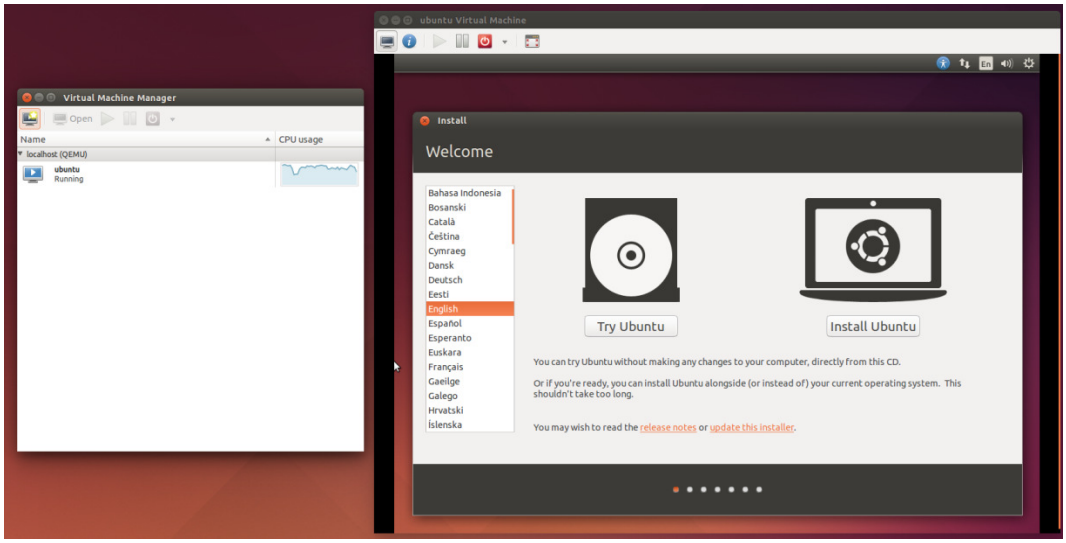


Click on the 'Create a new virtual machine' icon, type a name (e.g. ubuntu) and click Forward. Select the ISO image, and make sure OS type and Version are set to Linux and Ubuntu 14.04 LTS. Click Forward. For Memory and CPU, select 4096 MB and 1 CPU respectively.

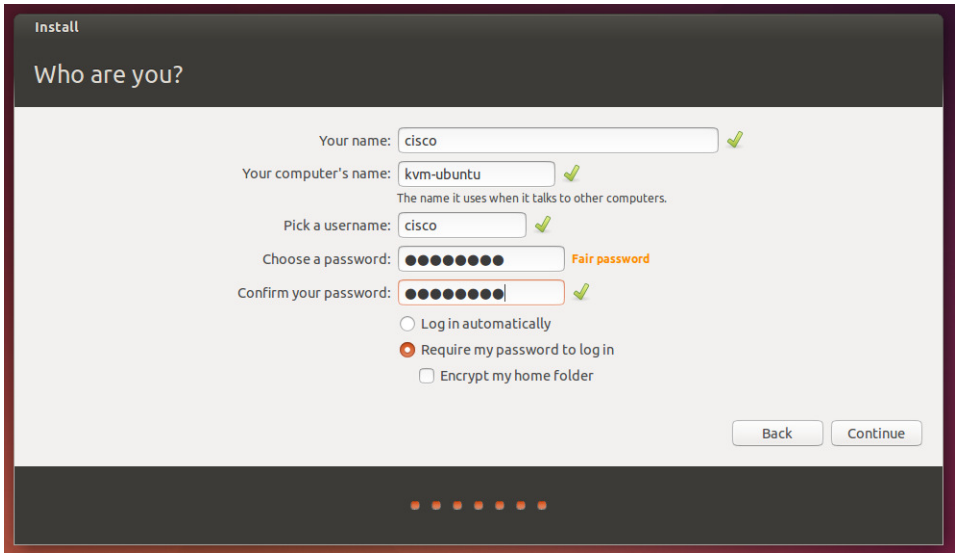
On the next screen select 20GB for the disk image size.



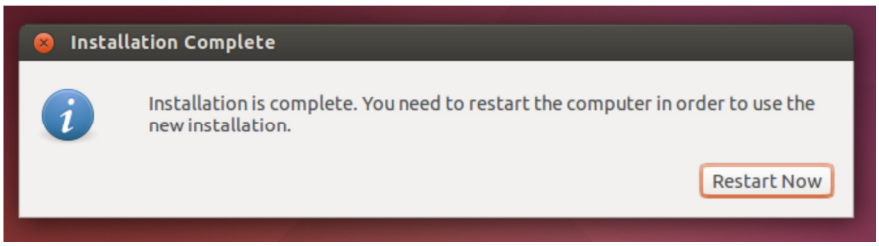
Eventually the KVM image will be created and you'll see Ubuntu trying to be installed:



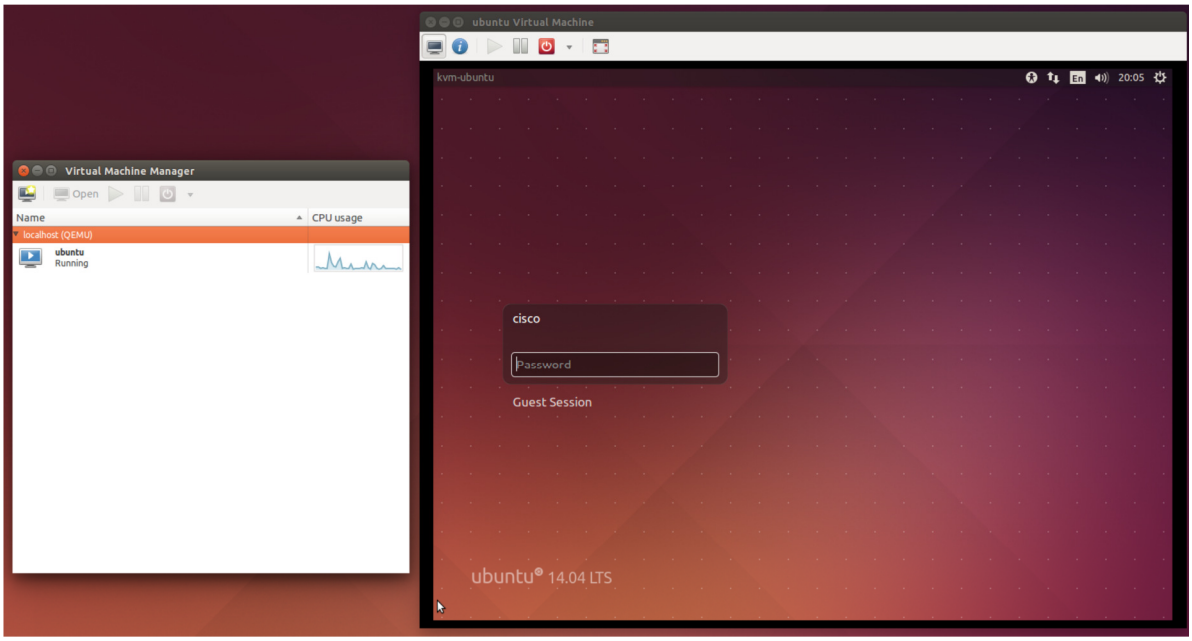
Click Install Ubuntu and accept all the defaults. You'll need to type some configuration details as shown here:



After a while the install will be complete, click Restart Now. It will then prompt you to remove any media and press Enter. Just press Enter.



If you named the KVM image ubuntu then the 20GB image file will be called ubuntu.img and will reside in /var/lib/libvirt/images (you need to be root user to go inside that folder). You are now at the position where you have your PC running a VM inside a VM:



4. Customizing the KVM Virtual Machine and Installing Applications

Log in and create a password for the root user (as before).

4.1 Enable SSH

Enable SSH using the following command as root user:

```
apt-get install openssh-server
```

Still as root user, edit the /etc/ssh/sshd_config file and change the line that says

PermitRootLogin without-password

to:

PermitRootLogin yes

Save the file, and then reboot (type reboot as root user)

After it restarts test that SSH works by typing

ssh root@127.0.0.1

If you're successful then type exit to quit the SSH session.

4.2 Enable Serial Console

Much like a real PC can have a serial or RS232 connection for a console session, it is possible to create a virtual one so that from IOS it becomes possible to connect to the Service Container without knowing its IP address, using the following command:

```
virtual-service connect name <my_service_container_name> console command
```

To do this, as root user create a file called /etc/init/ttyS0.conf containing the following:

```
# ttyS0 - getty
#
# This service maintains a getty on ttyS0


start on stopped rc RUNLEVEL=[2345] and (
            not-container or
            container CONTAINER=lxc or
            container CONTAINER=lxc-libvirt)


stop on runlevel [!2345]


respawn
```

```
exec /sbin/getty -L 9600 ttyS0 vt102
```

Save the file and then as root user type:

```
start ttyS0
```

4.3 Install any other Applications or Packages

You can now install any packages and applications as you desire.

If you wish to have an additional network interface you can create it using the virtual machine manager.

Finally shut down your KVM virtual machine and close the virtual machine manager application.

5. Building the Service Container

Now that the KVM virtual machine is in a good state, it can be assembled into a service container file which will be in an OVA format.

In you Linux development environment as a normal user create a folder called (say) container-build and inside there copy across the create_ova.sh file. Also create a folder there with the name of the KVM virtual machine (e.g. ubuntu) and copy across the package.yaml file there.

```
mkdir container-build

cd container-build

cp ~/templates/create_ova.sh .

mkdir ubuntu

cd ubuntu

cp ~/templates/package.yaml .
```

Type the following which will create a version.ver file containing just the text 1.0 (this matches the version number inside the package.yaml file; the grep command here displays it so you can check, and the echo command creates the file):

```
cd container-build/ubuntu

grep manifest-version package.yaml

echo 1.0 > version.ver
```

You can edit the package.yaml file to suit your needs.

Next, as root user type the following in the ubuntu folder:

```
qemu-img convert -p -c -f raw -o compat=0.10 -O qcow2
/var/lib/libvirt/images/ubuntu.img ubuntu.qcow2
```

(you need to be root user because the source /var/lib/libvirt/images folder is inaccessible otherwise). The process will take a while but progress will be displayed as a percentage.

Next, set the owner/group permissions for the newly created file into something suitable for the non-root user, such as typing:

```
chown bob:bob ubuntu.qcow2
```

Exit out of root user and then from the container-build folder issue the following command to generate the OVA file:

```
./create_ova.sh –mts 200000 –mfs 100000 ubuntu
```

The create_ova.sh script will create the ubuntu.ova file in the container-build/ubuntu folder. As part of that, the ubuntu.mf file will be created which contains a SHA1 hash per uncompressed file, and the script will then compress any file (using bzip2) larger than 100,000MB if the overall folder size is greater than 200,000MB. With these large numbers it effectively means that the ubuntu.qcow2 file (which is about 1.9GB in size) will not be compressed inside the OVA file. We want all the files to remain uncompressed.

In summary the script creates an OVA file contains the following files:

```
package.yaml

ubuntu.mf

ubuntu.qcow2

version.ver
```

Installing the Service Container

Copy the ubuntu.ova file onto a USB stick, insert it into the router and type:

```
copy usb0:ubuntu.ova harddisk:
```

Create a DHCP pool so that the service container Linux instance can acquire an IP address:

```
ip dhcp excluded-address 10.0.0.1

ip dhcp excluded-address 10.0.0.254

!

ip dhcp pool ubuntu-pool

 import all

 network 10.0.0.0 255.255.255.0

 default-router 10.0.0.1

 dns-server 192.168.1.254

 lease 0 5
```

Next, configure up the virtual service as shown here:

```
interface VirtualPortGroup1
 ip address 10.0.0.1 255.255.255.0
virtual-service
 signing level unsigned
virtual-service install
virtual-service ubuntu
 vnic gateway VirtualPortGroup1
```

If you like, turn on some debug:

```
debug virtual-service all
terminal monitor
```

Install the virtual service:

```
virtual-service install name ubuntu package harddisk:ubuntu.ova
```

You can check the status using:

```
show virtual-service list
```

Once that command shows the service is in Installed state, you can configure in IOS-XE for the service to be activated:

```
virtual-service ubuntu
activate
```

Check the state using the same command as before:

```
show virtual-service list
```

If successful the state will now be Activated.

You should be able to successfully ping the service container

```
ping 10.0.0.2
```

You should be able to SSH into it from the router like this:

```
ssh -l cisco 10.0.0.2
```

You can also connect to the virtual serial console using:

```
virtual-service connect name ubuntu console
```

To exit, type Ctrl-C three times.

Note: The virtual serial command only works if the ttyS0.conf file was created and start ttyS0 were executed as indicated in the 'Enable Serial Console' section earlier. If you are using the DCloud lab,

then the ubuntu.ova file is missing this. To add it, just SSH into the service container and then manually add the file.

Building Applications using the C Programming Language

The ubuntu image includes the Gnu compiler collection. You can create an example file called test.c containing the following:

```
#include <stdio.h>

int

main(void)

{

  printf("hello\n");

  return(0);

}
```

Save the file and build it using

```
gcc –o test test.c
```

Run the program by typing:

```
./test
```

Running Docker Applications

The example ubuntu.ova file available from Cisco has Docker installed. You can run an example Docker application by typing:

```
docker run docker/whalesay cowsay boo
```

Creating a Real Application

As an example, it was decided to create an application that would allow a user to see the round trip time (i.e. ping time) from the router to the user. In other words, a ping from the router to the user. For convenience this application should be usable via a browser.

This entails running a web server on the Service Container, and whenever someone tries to access the server using their browser, the Service Container should examine the IP address of the requesting browser, send a ping to it, and then send the ping result back in the served web page.

The application will be written and tested in the Development Environment first.

Create a file with the content below and name it basic_server.js – the file is a web server. By examining it you can see that it waits for a HTTP request on port 8090, and then it runs a ping command and then sends the result via HTTP.

```
#!/usr/bin/nodejs
```

```
console.log("Hello");
var app = require('http').createServer(handler);
var child = require('child_process');
app.listen(8090);


//HTML handler
function handler (req, res)
{
sel=req.url.substr(1);
remote_ip=req.connection.remoteAddress;
console.log("Got a request from "+remote_ip);
if (sel != "diskstat.html")
{
  sel = "diskstat.html"; // only allow this file for now
}

res.writeHead(200);
prog=child.exec("ping -c 4 "+remote_ip, function(error, data, stderr)
{
  res.end(data);
});

} // end of HTML handler
```

To run this simple program, a software platform called Node.js is needed. As root user type the following:

```
apt-get install nodejs
```

Exit out of root user, and then as a normal user type

```
chmod 755 basic_server.js
```

You can now test the program. First get the IP address of your development environment (type ifconfig –a) and make a note of it. Then execute the program by typing:

```
./basic_server.js
```

It should print 'Hello' and then it will wait for a browser to send a HTTP request.

From your PC in a web browser type http://xx.xx.xx.xx:8090 where xx.xx.xx.xx is the IP address of the development environment.

There should be a brief pause (while four ping messages are sent from the Service Container) and then the browser should display the result.

To stop the web server just press Ctrl-C.

Storing your Application

For convenience, upload your application to GitHub. This entails creating a GitHub account, downloading the GitHub application onto your PC and then creating a repository and uploading the basic_server.js file there.

If you want to save time, you can use the already existing file at https://raw.githubusercontent.com/shabaz123/ServiceContainers/master/basic_server.js

Deploying Applications using Docker

To keep things interesting, it was decided to deploy this application on the Service Container using Docker.

First, install Docker in the Development Environment as before (previously it was installed on the Service Container VM; now repeat those steps in the Development Environment).

Next, as a normal user create a folder somewhere called basic_server_dockerbuild. In that folder create a file called Dockerfile and it should contain the following:

```
FROM ubuntu

RUN apt-get -y update && apt-get install -y nodejs

RUN apt-get install -y wget

RUN mkdir -p /tmp/basic_server

WORKDIR /tmp/basic_server

RUN wget --directory-prefix /tmp/basic_server
https://raw.githubusercontent.com/shabaz123/ServiceContainers/master/basic_
server.js && chmod 755 basic_server.js

EXPOSE 8090

CMD [ "nodejs", "basic_server.js" ]
```

Build the Docker image by typing:

```
docker build –t basic_server .
```

Next, test it out by typing:

```
docker run –p 8090:8090 basic_server
```

If this works then stop it using Ctrl-C and then create an account at hub.docker.com .

(If you don't want to have to press Ctrl-C, the container can be run in detached mode with a –d switch on the command line, but then you won't see any logging output on the screen).

Click on Create Repository and enter some details. Type basic-server in the Enter Name box (it needs to match the name of the Docker image). The Visibility should be set to public.

An Image ID is needed for the Docker image. Type the following to see it:

```
docker images | grep basic_server
```

Next type (use your hub.docker.com account username and the Image ID you obtained):

```
docker tag d8dc7cbca23f bob/basic_server:latest

docker login –u=bob

docker push bob/basic_server
```

From hub.docker.com you should see that the image has been pushed!

To test a pull operation, first delete the image from your Development Environment by typing:

```
docker rmi –f bob/basic_server
```

Next, type the following:

```
docker run –p 8090:8090 shabaz/basic_server
```

If this all works then you're ready to run it on the router!

Log into the Service Container on the router, and type the same command:

```
docker run –p 8090:8090 shabaz/basic_server
```

Here is some example output:

```
cisco@kvm-ubuntu:~$ docker images

REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE

hello-world         latest              690ed74de00f        6 months ago
960 B

docker/whalesay     latest              6b362a9f73eb        10 months ago
247 MB
```

```
cisco@kvm-ubuntu:~$

cisco@kvm-ubuntu:~$

cisco@kvm-ubuntu:~$ docker run -p 8090:8090 shabaz/basic_server

Unable to find image 'shabaz/basic_server:latest' locally

latest: Pulling from shabaz/basic_server

759d6771041e: Pull complete

8836b825667b: Pull complete

c2f5e51744e6: Pull complete

a3ed95caeb02: Pull complete

250158b04904: Pull complete

c58248ebf101: Pull complete

6bec2a9e3a22: Pull complete

cfb5e5cd6aef: Pull complete

Digest:
sha256:156e524ad011fcd7c5dbce019524d2f35d40d755f5663f016075870ed8e5a496

Status: Downloaded newer image for shabaz/basic_server:latest

Hello

Got a request from 192.168.1.131

Got a request from 192.168.1.131
```

Note that you can also access this Docker container service directly from the router if desired since the service is exposed via HTTP.

At the router CLI type the following:

```
tclsh

more http://10.0.0.2:8090/index.html

tclquit
```

Running it directly from the router CLI doesn't make much sense for this particular ping service, but other services could make sense.

It would be desirable to be able to automatically run this image whenever the router and Service Container is powered up. To do this, type the following inside the Service Container:

```
docker run --restart=always -p 8090:8090 shabaz/basic_server
```

Now type Ctrl-C and then reboot the Service Container. Once the container comes back up, you won't need to issue any commands for the service to function.

To stop the container process, type the following to find the Container ID:

```
docker ps
```

Then, type (using the ID):

```
docker stop b271906c47a7
```

---X---