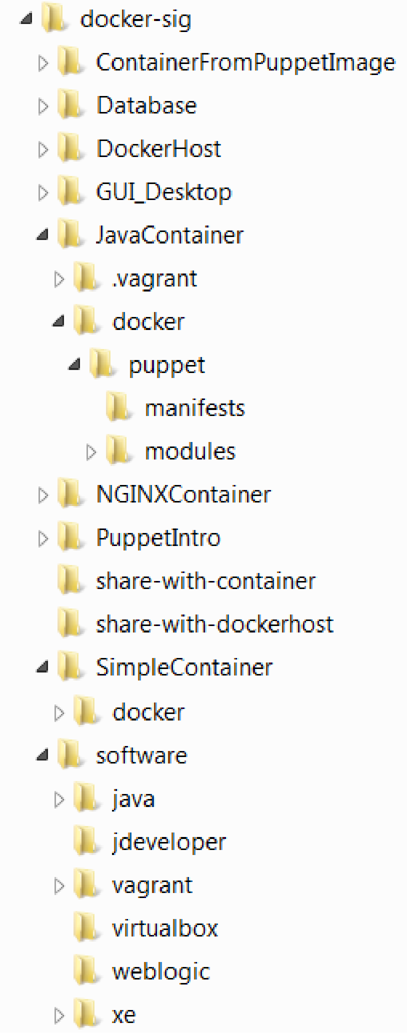
Efficient Environment Management Using Docker [and VirtualBox, Vagrant and Puppet]

In this hands-on, you will be introduced to Docker - a popular tool for running & managing, shipping and sharing, and building containers. Docker runs on a Linux host platform. In order to work with Docker on non-Linux platforms – such as Mac OS and Windows – we will use Vagrant to generate a Virtual Machine with VirtualBox and set it up as Docker Host.

Vagrant will also be used in some instances to build, start and stop Docker containers; in that case, Vagrant talks – from Windows or Mac OS - to the Docker Engine API inside the Docker Host VM to manipulate containers.

Using SSH, we can open a terminal in the Docker Host VM – based on Ubuntu Linux (14.04) – and use the Docker Command Line Interface directly, rather than through Vagrant.

Note: we assume in these instructions that you have copied the workshop materials from the USB stick to a folder called c:\docker-sig. This folder should look as is shown in the figure.



```
docker-sig
  ContainerFromPuppetImage
  Database
  DockerHost
  GUI_Desktop
  JavaContainer
    .vagrant
    docker
      puppet
        manifests
        modules
  NGINXContainer
  PuppetIntro
  share-with-container
  share-with-dockerhost
  SimpleContainer
    docker
  software
    java
    jdeveloper
    vagrant
    virtualbox
    weblogic
    xe
```

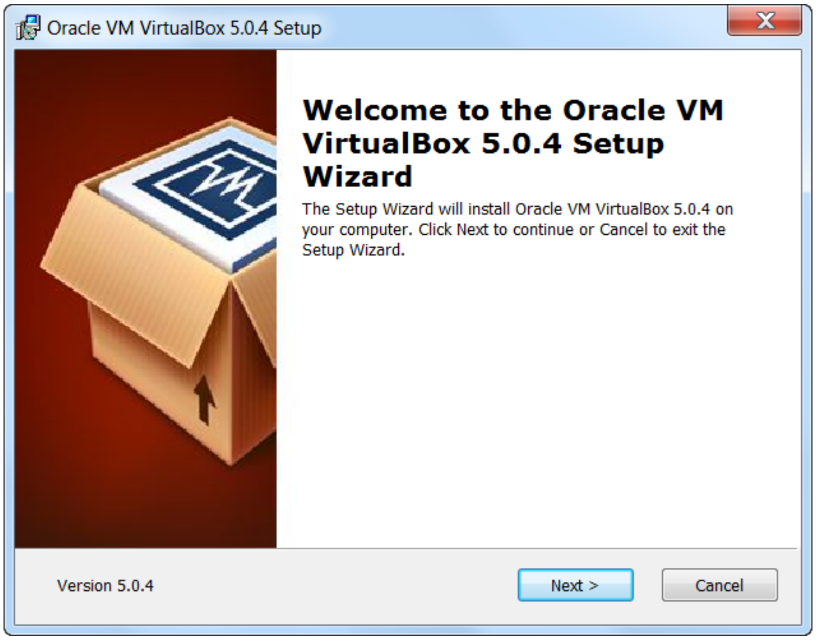
If you have copied the files to a different location, you will have to map the instructions in this document to whatever location you have chosen.

The instructions in this document are written from the perspective of Windows. Please adjust where necessary to the corresponding Mac OS actions if you happen to work from that Host operating system. Most actions are through the command line, with configuration files or inside the Docker Host VM and will be the same across all platforms.

1. Preparation of the host environment

The first steps in the hands-on are preparation: set up VirtualBox and Vagrant.

Please open folder c:\docker-sig\software\virtualbox and run the installer for your host platform in order to install Virtual Box 5.0.



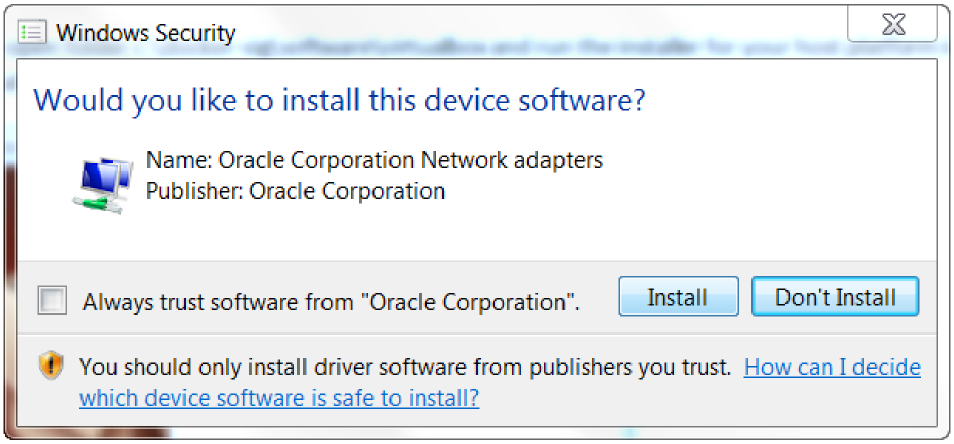
Oracle VM VirtualBox 5.0.4 Setup

Welcome to the Oracle VM VirtualBox 5.0.4 Setup Wizard

The Setup Wizard will install Oracle VM VirtualBox 5.0.4 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Version 5.0.4

Step through the installation wizard and accept all defaults. Please also press *install* for the three popups that will appear, similar to this next one:



Windows Security

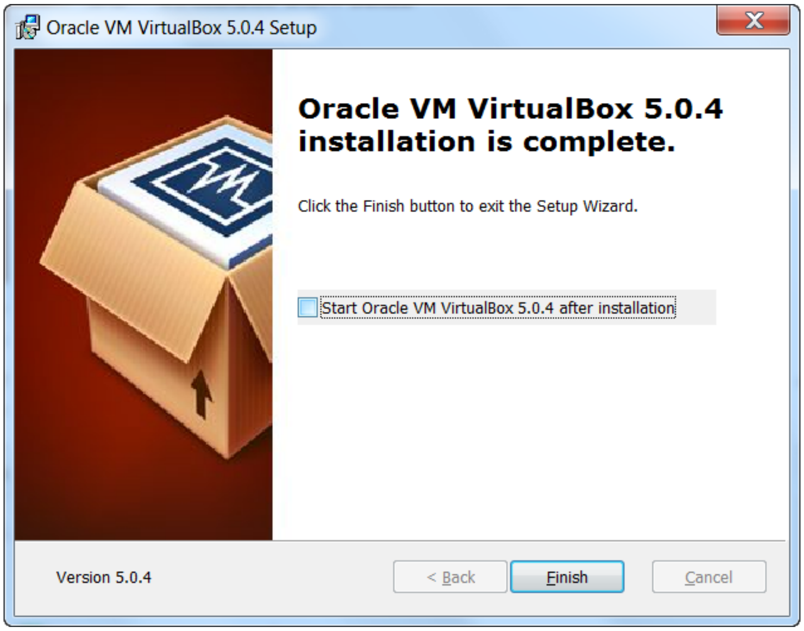
Would you like to install this device software?

Name: Oracle Corporation Network adapters
Publisher: Oracle Corporation

□ Always trust software from "Oracle Corporation".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

Finally, the installer completes. Do not run VirtualBox at this point – we will do so through Vagrant.



Oracle VM VirtualBox 5.0.4 Setup

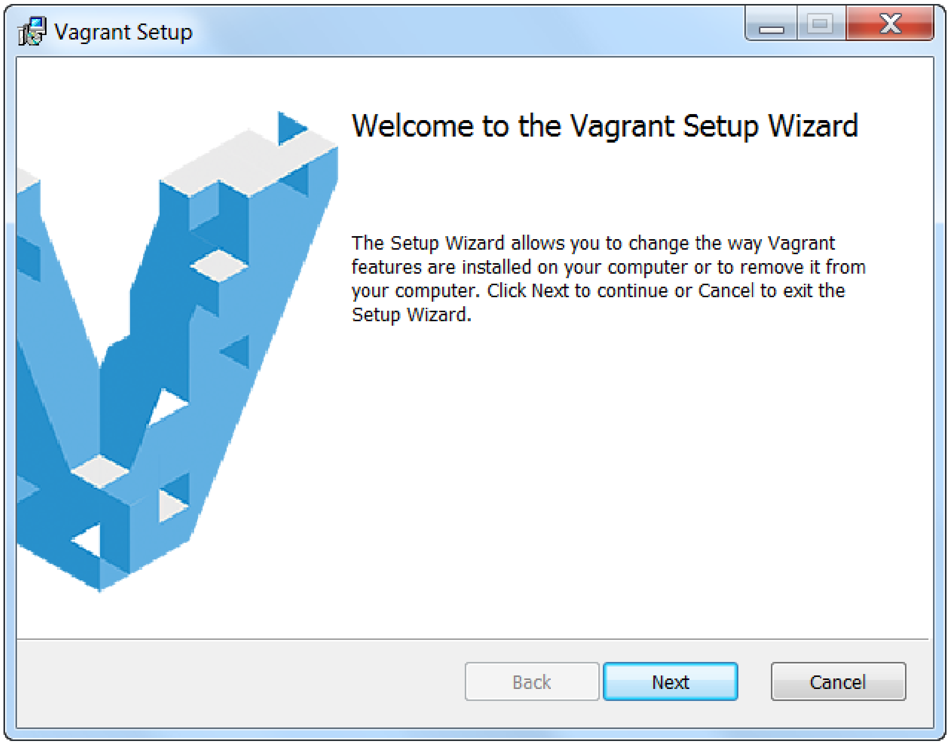
Oracle VM VirtualBox 5.0.4 installation is complete.

Click the Finish button to exit the Setup Wizard.

□ Start Oracle VM VirtualBox 5.0.4 after installation

Version 5.0.4

Next, navigate to folder C:\docker-sig\software\vagrant. Run the appropriate installer for Vagrant.



Vagrant Setup

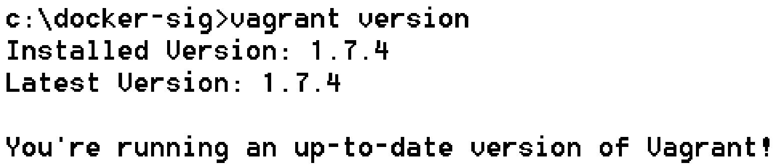
Welcome to the Vagrant Setup Wizard

The Setup Wizard allows you to change the way Vagrant features are installed on your computer or to remove it from your computer. Click Next to continue or Cancel to exit the Setup Wizard.

When the wizard completes, open a command window. Navigate to c:\docker-sig. Type:

```
vagrant version
```

and verify that Vagrant is running – and the correct version is installed.



```
c:\docker-sig>vagrant version
Installed Version: 1.7.4
Latest Version: 1.7.4

You're running an up-to-date version of Vagrant!
```

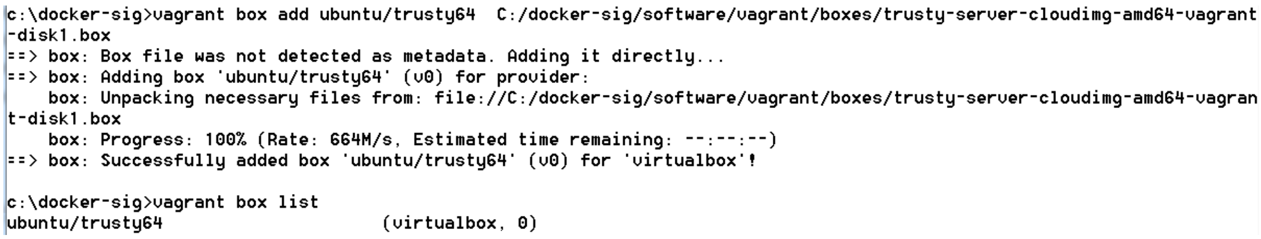
In order to speed up the process of creating Virtual Machines with Vagrant, we can prime the local Vagrant box repository. Directory C:\docker-sig\software\vagrant\boxes contains some boxes that we will be using later on. In order to prevent length downloading, please execute the following command that will add these boxes to the local Vagrant box repository (note: based on http://stackoverflow.com/questions/28399324/download-vagrant-box-file-locally-from-atlas-and-configuring-it)

```
vagrant box add ubuntu/trusty64  C:/docker-sig/software/vagrant/boxes/trusty-server-cloudimg-amd64-vagrant-disk1.box
```

After running this command, execute the following command to check whether the box was added successfully:

```
vagrant box list
```

Verify if the output includes the Ubuntu/trusty64 box, as shown in the figure.



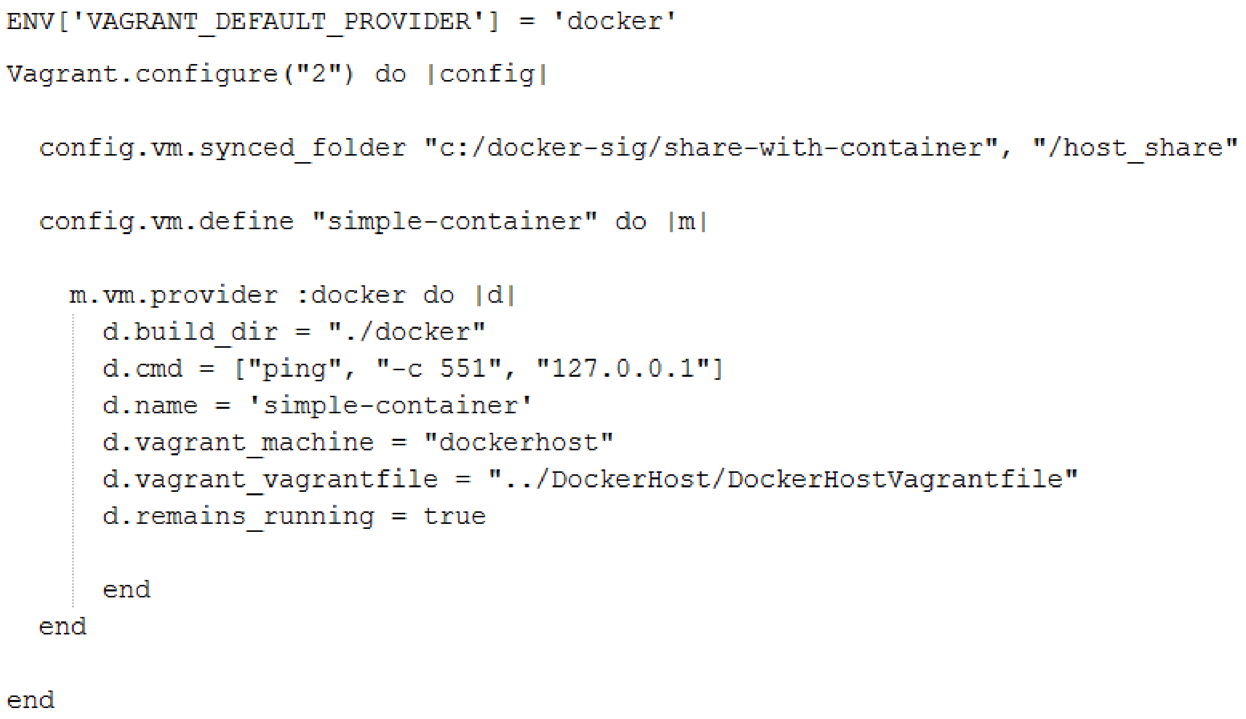
```
c:\docker-sig>vagrant box add ubuntu/trusty64  C:/docker-sig/software/vagrant/boxes/trusty-server-cloudimg-amd64-vagrant
-disk1.box
==> box: Box file was not detected as metadata. Adding it directly...
==> box: Adding box 'ubuntu/trusty64' (v0) for provider:
    box: Unpacking necessary files from: file://C:/docker-sig/software/vagrant/boxes/trusty-server-cloudimg-amd64-vagran
t-disk1.box
    box: Progress: 100% (Rate: 664M/s, Estimated time remaining: --:--:--)
==> box: Successfully added box 'ubuntu/trusty64' (v0) for 'virtualbox'!

c:\docker-sig>vagrant box list
ubuntu/trusty64                    (virtualbox, 0)
```

2. Running your first Docker Container

Folder c:\docker-sig\SimpleContainer contains a Vagrantfile that provides instructions to Vagrant to provision a Docker container, using the Dockerfile in folder C:\docker-sig\SimpleContainer\docker. This Dockerfile should look familiar from the demonstration you have just seen. Check out this file and try to understand what this file will tell Docker to do during the build process.

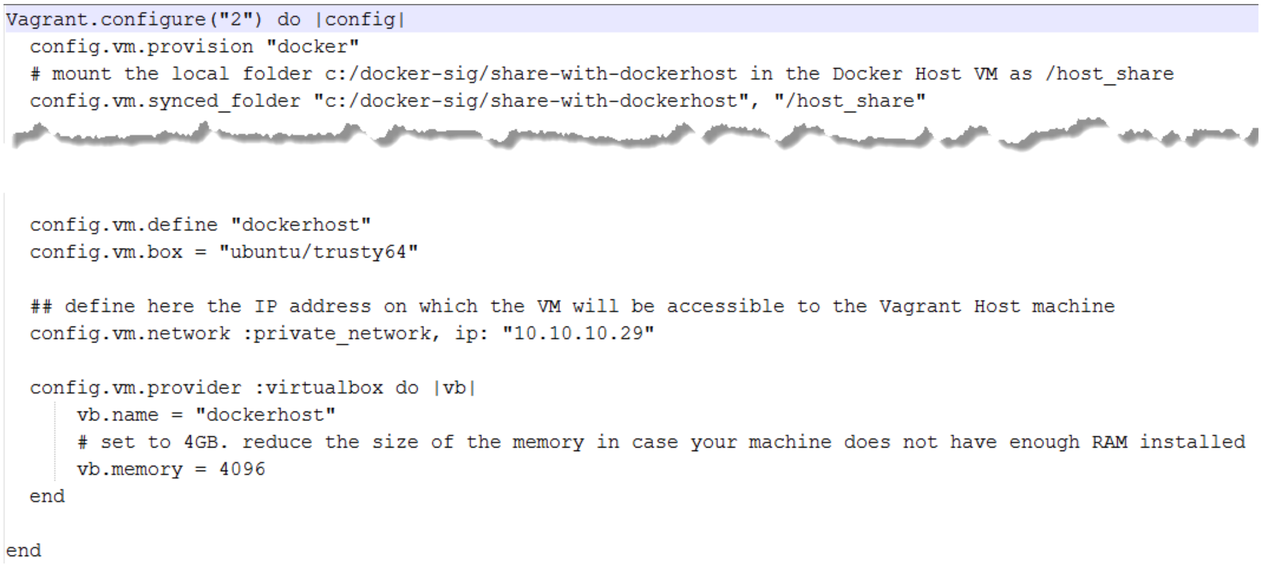
```
FROM ubuntu:14.04
ADD files/mynewfile.txt /tmp/
RUN mkdir /tmp/stuff
WORKDIR /tmp
RUN touch somefile.txt
WORKDIR stuff
RUN cp ../*.txt .
EXPOSE 8090
CMD echo "message from brand new container"
```

The Vagrantfile references a *dockerhost* – through settings d.vagrant_machine and d.vagrant_vagrantfile.



```
ENV['VAGRANT_DEFAULT_PROVIDER'] = 'docker'

Vagrant.configure("2") do |config|

  config.vm.synced_folder "c:/docker-sig/share-with-container", "/host_share"

  config.vm.define "simple-container" do |m|

    m.vm.provider :docker do |d|
      d.build_dir = "./docker"
      d.cmd = ["ping", "-c 551", "127.0.0.1"]
      d.name = 'simple-container'
      d.vagrant_machine = "dockerhost"
      d.vagrant_vagrantfile = "../DockerHost/DockerHostVagrantfile"
      d.remains_running = true

      end
  end

end
```

This dockerhost VM – a Linux VirtualBox VM – is described by the file DockerHostVagrantfile in directory C:\docker-sig\DockerHost.



```
Vagrant.configure("2") do |config|
  config.vm.provision "docker"
  # mount the local folder c:/docker-sig/share-with-dockerhost in the Docker Host VM as /host_share
  config.vm.synced_folder "c:/docker-sig/share-with-dockerhost", "/host_share"

  config.vm.define "dockerhost"
  config.vm.box = "ubuntu/trusty64"

  ## define here the IP address on which the VM will be accessible to the Vagrant Host machine
  config.vm.network :private_network, ip: "10.10.10.29"

  config.vm.provider :virtualbox do |vb|
      vb.name = "dockerhost"
      # set to 4GB. reduce the size of the memory in case your machine does not have enough RAM installed
      vb.memory = 4096
  end

end
```

Note that this file defines a folder mapping from directory C:\docker-sig\share-with-dockerhost into the dockerhost VM as well as a memory size of 4GB for the dockerhost VM. Depending on the available RAM memory on your laptop, you can decide to adjust this setting. Finally, not that a private network is defined, that sets the IP Address for the dockerhost to 10.10.10.29.

When the Docker container simple-container is started through Vagrant, the dockerhost is looked for. When it is found out that this VM does not yet exist, it will be created by Vagrant – and set up for running Docker.

Once the *dockerhost* VM is available, the Docker daemon will be called upon – by Vagrant, through the API – to build the container, based on the Dockerfile.

Let’s see this in action.

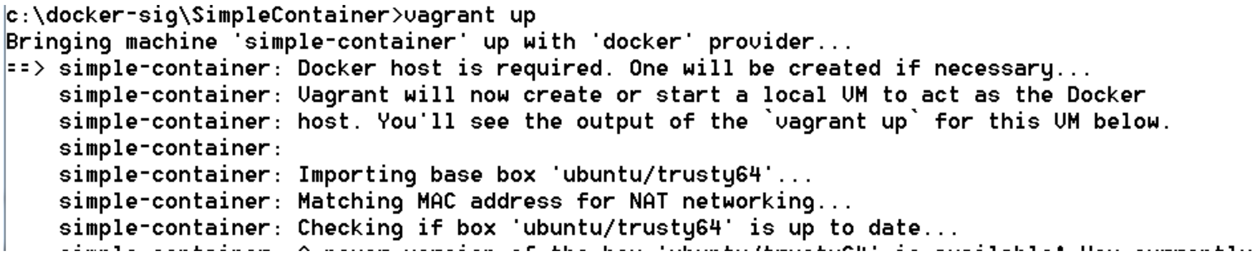
Open a command line window and navigate to C:\docker-sig\SimpleContainer.

Enter the command

```
vagrant up
```

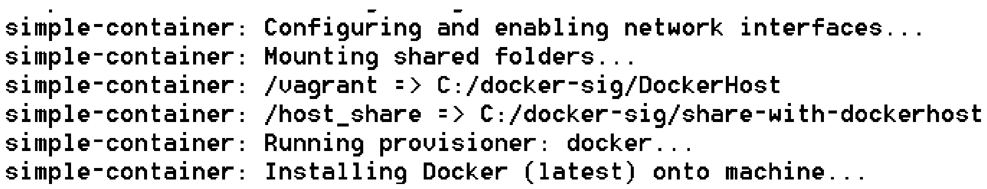
This will start the process of creating the dockerhost VM – if it does not already exist – and building and running the Docker container. This entire process can take a while.

The initial output will look like this:



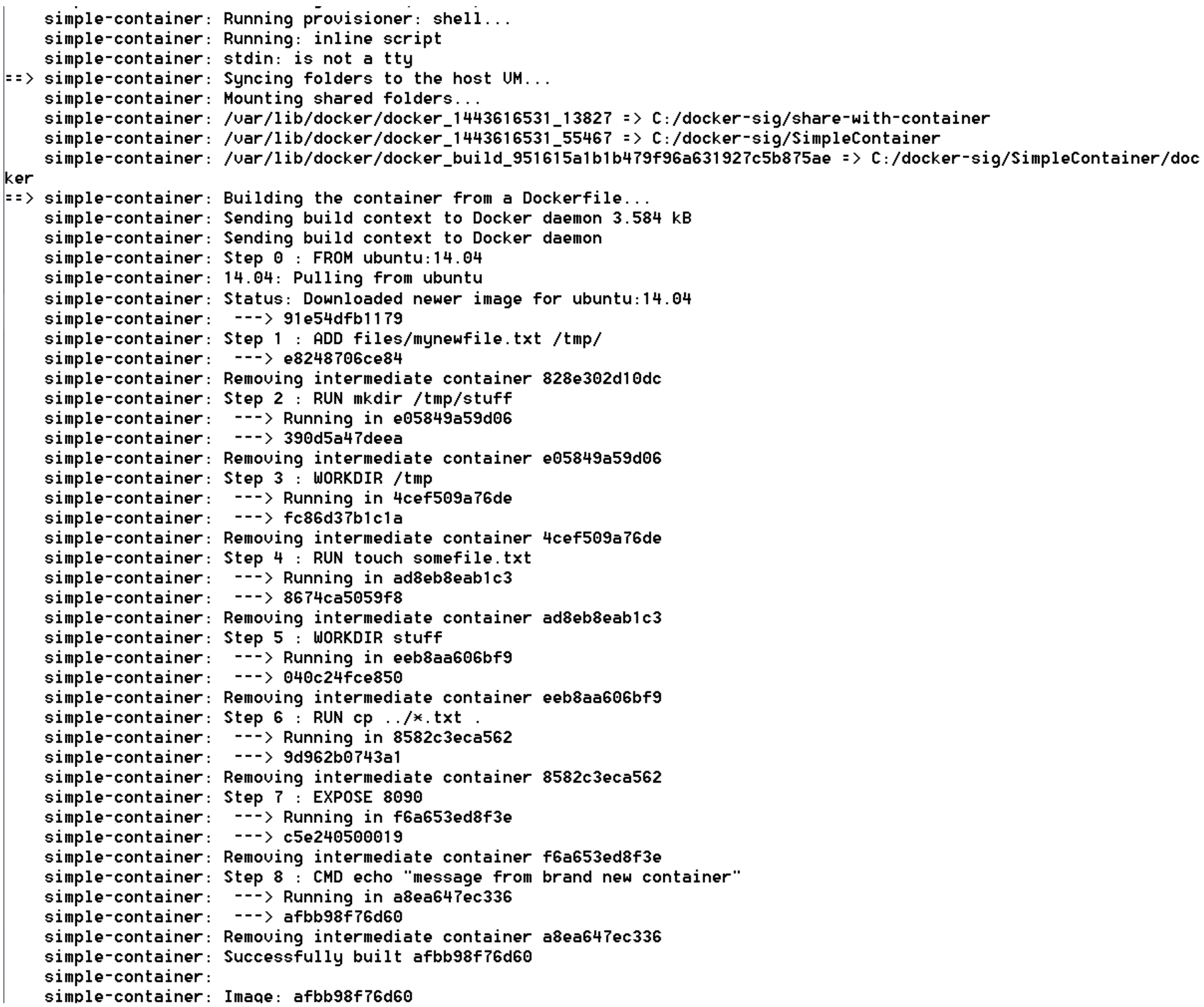
```
c:\docker-sig\SimpleContainer>vagrant up
Bringing machine 'simple-container' up with 'docker' provider...
==> simple-container: Docker host is required. One will be created if necessary...
    simple-container: Vagrant will now create or start a local VM to act as the Docker
    simple-container: host. You'll see the output of the `vagrant up` for this VM below.
    simple-container:
    simple-container: Importing base box 'ubuntu/trusty64'...
    simple-container: Matching MAC address for NAT networking...
    simple-container: Checking if box 'ubuntu/trusty64' is up to date...
```

After some time, when the dockerhost VM is up and running, including the folder mappings, Docker is installed into the VM.



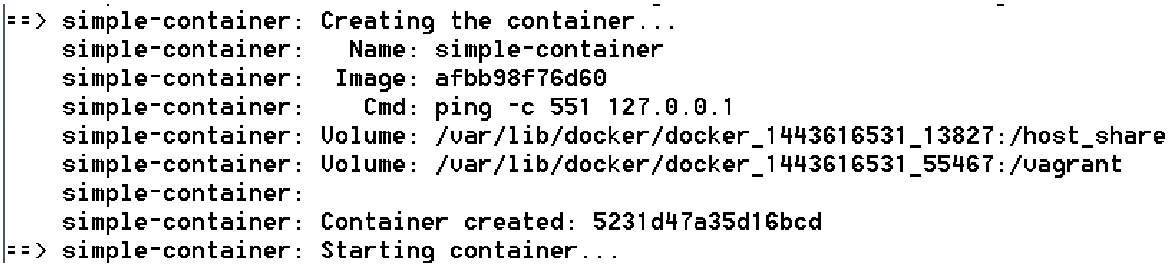
```
simple-container: Configuring and enabling network interfaces...
simple-container: Mounting shared folders...
simple-container: /vagrant => C:/docker-sig/DockerHost
simple-container: /host_share => C:/docker-sig/share-with-dockerhost
simple-container: Running provisioner: docker...
simple-container: Installing Docker (latest) onto machine...
```

Soon after, the container is built. The output on the command line should reflect the steps defined in the Dockerfile. Note that Vagrant creates a folder mapping for the directory c:/docker-sig/share-with-container on the Windows host to folder /host_share inside the Docker container.



```
    simple-container: Running provisioner: shell...
    simple-container: Running: inline script
    simple-container: stdin: is not a tty
==> simple-container: Syncing folders to the host VM...
    simple-container: Mounting shared folders...
    simple-container: /var/lib/docker/docker_1443616531_13827 => C:/docker-sig/share-with-container
    simple-container: /var/lib/docker/docker_1443616531_55467 => C:/docker-sig/SimpleContainer
    simple-container: /var/lib/docker/docker_build_951615a1b1b479f96a631927c5b875ae => C:/docker-sig/SimpleContainer/doc
ker
==> simple-container: Building the container from a Dockerfile...
    simple-container: Sending build context to Docker daemon 3.584 kB
    simple-container: Sending build context to Docker daemon
    simple-container: Step 0 : FROM ubuntu:14.04
    simple-container: 14.04: Pulling from ubuntu
    simple-container: Status: Downloaded newer image for ubuntu:14.04
    simple-container:  ---> 91e54dfb1179
    simple-container: Step 1 : ADD files/mynewfile.txt /tmp/
    simple-container:  ---> e8248706ce84
    simple-container: Removing intermediate container 828e302d10dc
    simple-container: Step 2 : RUN mkdir /tmp/stuff
    simple-container:  ---> Running in e05849a59d06
    simple-container:  ---> 390d5a47deea
    simple-container: Removing intermediate container e05849a59d06
    simple-container: Step 3 : WORKDIR /tmp
    simple-container:  ---> Running in 4cef509a76de
    simple-container:  ---> fc86d37b1c1a
    simple-container: Removing intermediate container 4cef509a76de
    simple-container: Step 4 : RUN touch somefile.txt
    simple-container:  ---> Running in ad8eb8eab1c3
    simple-container:  ---> 8674ca5059f8
    simple-container: Removing intermediate container ad8eb8eab1c3
    simple-container: Step 5 : WORKDIR stuff
    simple-container:  ---> Running in eeb8aa606bf9
    simple-container:  ---> 040c24fce850
    simple-container: Removing intermediate container eeb8aa606bf9
    simple-container: Step 6 : RUN cp ../*.txt .
    simple-container:  ---> Running in 8582c3eca562
    simple-container:  ---> 9d962b0743a1
    simple-container: Removing intermediate container 8582c3eca562
    simple-container: Step 7 : EXPOSE 8090
    simple-container:  ---> Running in f6a653ed8f3e
    simple-container:  ---> c5e240500019
    simple-container: Removing intermediate container f6a653ed8f3e
    simple-container: Step 8 : CMD echo "message from brand new container"
    simple-container:  ---> Running in a8ea647ec336
    simple-container:  ---> afbb98f76d60
    simple-container: Removing intermediate container a8ea647ec336
    simple-container: Successfully built afbb98f76d60
    simple-container:
    simple-container: Image: afbb98f76d60
```

Finally the following summary is presented:



```
==> simple-container: Creating the container...
    simple-container:   Name: simple-container
    simple-container:  Image: afbb98f76d60
    simple-container:    Cmd: ping -c 551 127.0.0.1
    simple-container: Volume: /var/lib/docker/docker_1443616531_13827:/host_share
    simple-container: Volume: /var/lib/docker/docker_1443616531_55467:/vagrant
    simple-container:
    simple-container: Container created: 5231d47a35d16bcd
==> simple-container: Starting container...
```

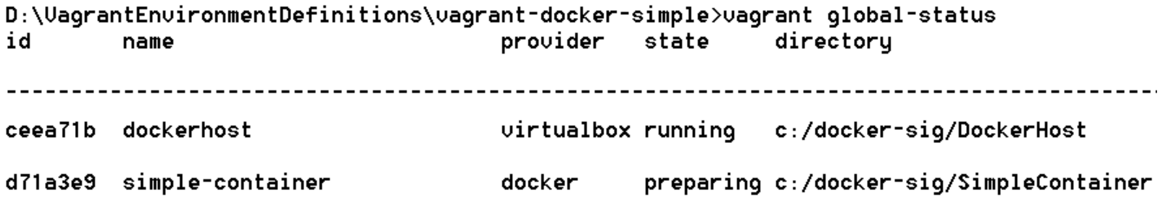
Presumably the container is running. But we cannot tell from the Windows command line.

Our next step is to open an SSH session to the dockerhost VM, using Vagrant.

First, type the command

```
vagrant global-status
```

This will list all Virtual Machines managed by Vagrant. Note: you will see different values for the id column.



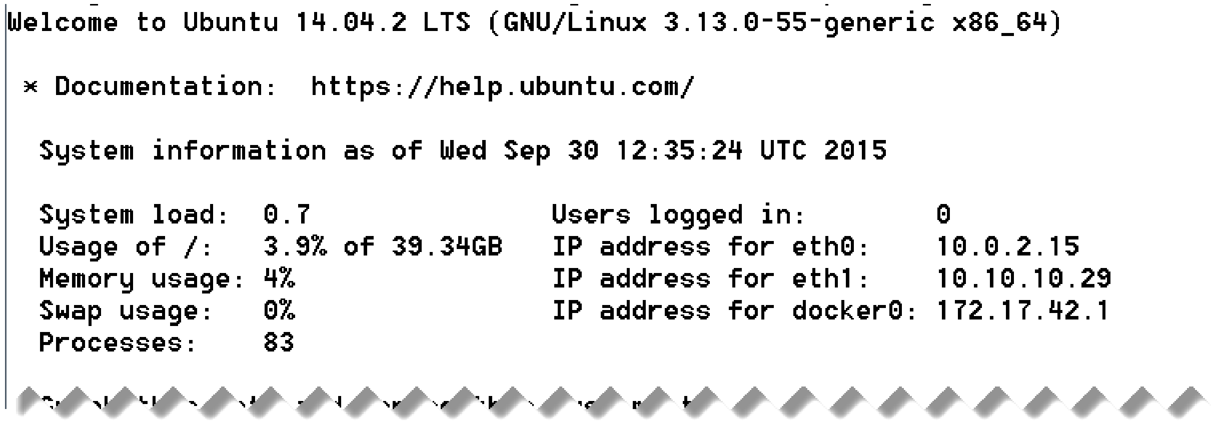
```
D:\VagrantEnvironmentDefinitions\vagrant-docker-simple>vagrant global-status
id       name                          provider   state     directory

-------------------------------------------------------------------------------------

ceea71b  dockerhost                    virtualbox running   c:/docker-sig/DockerHost

d71a3e9  simple-container              docker     preparing c:/docker-sig/SimpleContainer
```

Enter the command

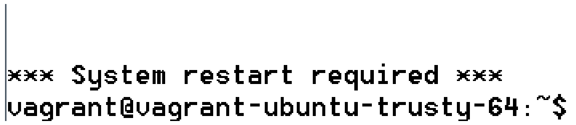
```
vagrant ssh <id for the dockerhost VM>
```

(in my case : vagrant ssh ceea71b).

You will enter the dockerhost VM: a secure shell is presented.



```
Welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-55-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Wed Sep 30 12:35:24 UTC 2015

  System load:  0.7               Users logged in:        0
  Usage of /:   3.9% of 39.34GB   IP address for eth0:    10.0.2.15
  Memory usage: 4%                IP address for eth1:    10.10.10.29
  Swap usage:   0%                IP address for docker0: 172.17.42.1
  Processes:    83
```

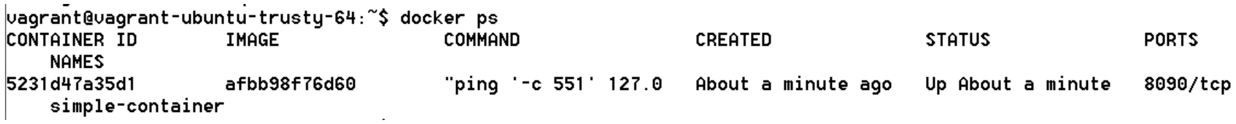


```
*** System restart required ***
vagrant@vagrant-ubuntu-trusty-64:~$
```

Type

```
docker ps
```

to list all running Docker containers. This should return the simple-container that we had Vagrant prepare for us.

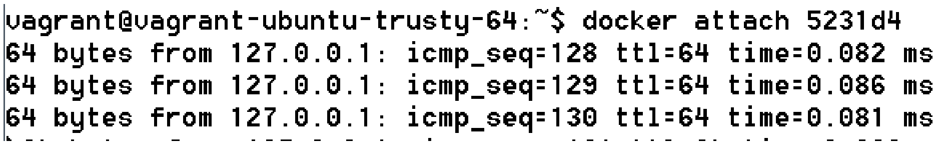


```
vagrant@vagrant-ubuntu-trusty-64:~$ docker ps
CONTAINER ID        IMAGE               COMMAND              CREATED             STATUS              PORTS
    NAMES
5231d47a35d1        afbb98f76d60        "ping '-c 551' 127.0  About a minute ago  Up About a minute   8090/tcp
    simple-container
```

Use the first few characters from the CONTAINER ID (in my case for example 5231d4) to manipulate the container. Let’s attach to it, to see what is going on inside:

```
docker attach <CONTAINER ID>
```

Output from the ping command is produced.



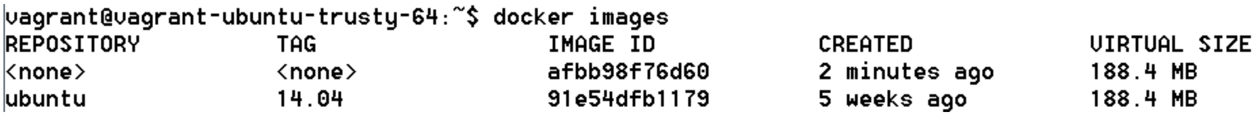
```
vagrant@vagrant-ubuntu-trusty-64:~$ docker attach 5231d4
64 bytes from 127.0.0.1: icmp_seq=128 ttl=64 time=0.082 ms
64 bytes from 127.0.0.1: icmp_seq=129 ttl=64 time=0.086 ms
64 bytes from 127.0.0.1: icmp_seq=130 ttl=64 time=0.081 ms
```

You will soon tire of this. Use CTRL+C to kill the container.

Type

```
docker images
```

to list all Docker images in the dockerhost VM.



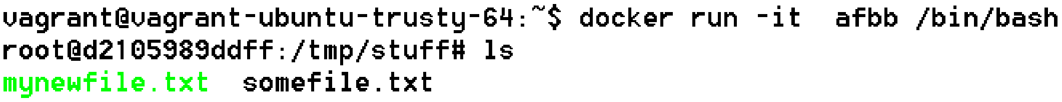
```
vagrant@vagrant-ubuntu-trusty-64:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
<none>              <none>              afbb98f76d60        2 minutes ago       188.4 MB
ubuntu              14.04               91e54dfb1179        5 weeks ago         188.4 MB
```

The image prepared while building the container should be listed, along with the official Ubuntu:14.04 image.

Type the following command to run [a container based on] the image created by the Vagrant & Docker combination:

```
docker run -it  IMAGE_ID /bin/bash
```

Note that a few identifying characters from the image id are enough to run the image; in my case this command suffices: docker run -it **afbb** /bin/bash. Check the contents of folder /tmp. There should be a directory /tmp/stuff, created during the container build process. It should contain two files – again, as per the build process.



```
vagrant@vagrant-ubuntu-trusty-64:~$ docker run -it  afbb /bin/bash
root@d2105989ddff:/tmp/stuff# ls
mynewfile.txt  somefile.txt
```

Note: you will not find the host folder that was mapped to the container. This folder mapping is applied to the container when it is started by Vagrant – it does not become part of the image. In order to benefit from the folder mapping in the container, we need to run the container through Vagrant.

Open a new Windows command line. Navigate to folder C:\docker-sig\SimpleContainer.

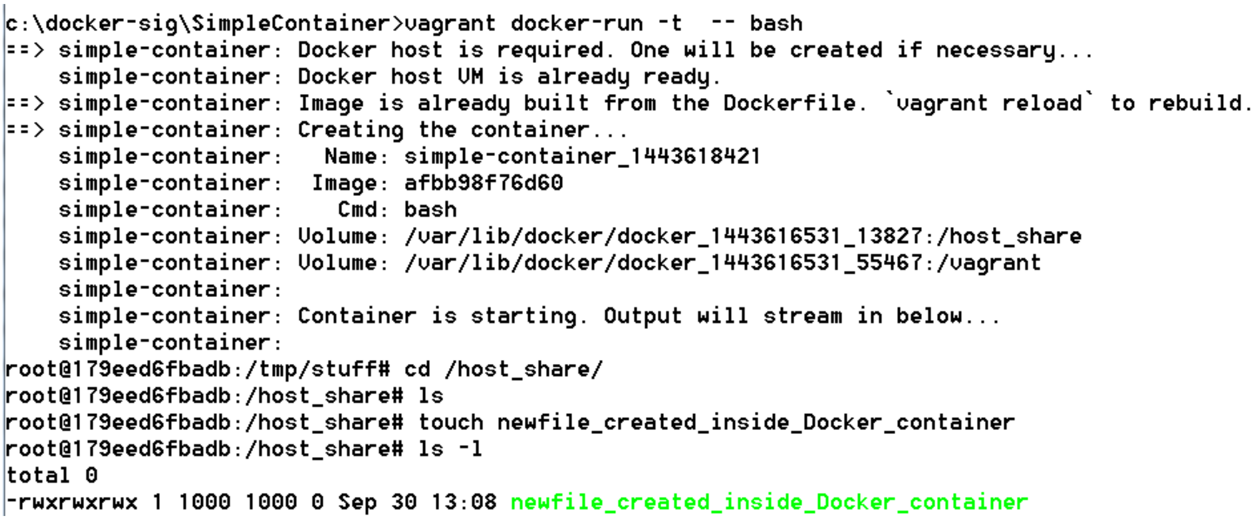
Type:

```
vagrant docker-run -t -- bash
```

This will run the container simple-container in interactive mode (because of the –t flag) and open a bash shell right into the Docker container, completely bypassing the dockerhost VM.

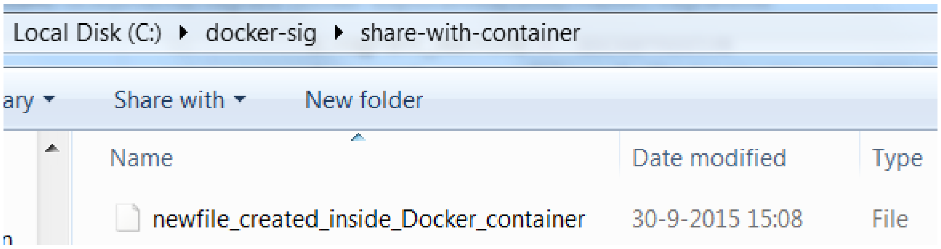
Inside the shell , navigate to directory /host_share – which is mapped from C:\docker-sig\share-with-container. Create a new file through:

```
touch newfile_created_inside_Docker_container
```



```
c:\docker-sig\SimpleContainer>vagrant docker-run -t  -- bash
==> simple-container: Docker host is required. One will be created if necessary...
    simple-container: Docker host VM is already ready.
==> simple-container: Image is already built from the Dockerfile. `vagrant reload` to rebuild.
==> simple-container: Creating the container...
    simple-container:   Name: simple-container_1443618421
    simple-container:  Image: afbb98f76d60
    simple-container:    Cmd: bash
    simple-container: Volume: /var/lib/docker/docker_1443616531_13827:/host_share
    simple-container: Volume: /var/lib/docker/docker_1443616531_55467:/vagrant
    simple-container:
    simple-container: Container is starting. Output will stream in below...
    simple-container:
root@179eed6fbadb:/tmp/stuff# cd /host_share/
root@179eed6fbadb:/host_share# ls
root@179eed6fbadb:/host_share# touch newfile_created_inside_Docker_container
root@179eed6fbadb:/host_share# ls -l
total 0
-rwxrwxrwx 1 1000 1000 0 Sep 30 13:08 newfile_created_inside_Docker_container
```

At this point, the file is created and visible inside the container. However, the file does not live in the container: it was created in the directory C:\docker-sig\share-with-container on the Windows host. Verify that this is indeed the case:



Name	Date modified	Type
newfile_created_inside_Docker_container	30-9-2015 15:08	File

You can modify the file on Windows and inspect those changes inside Docker container – and vice versa.

Type

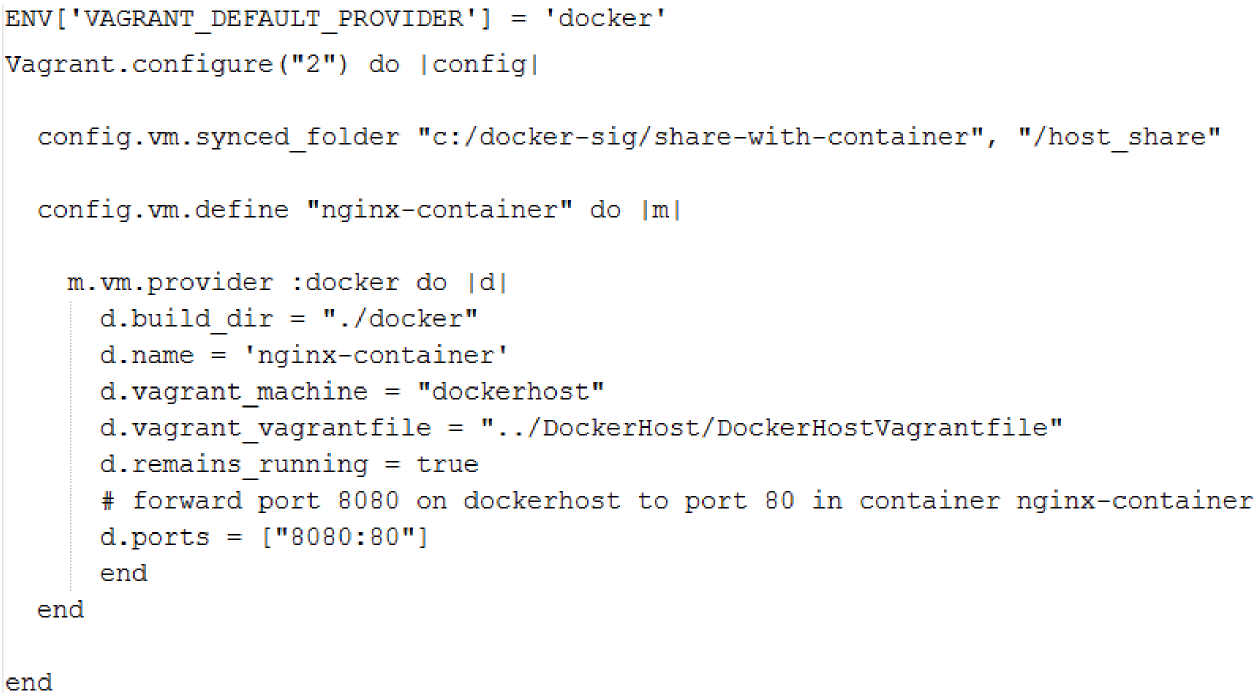
```
exit
```

to exit the bash shell. This will stop the container – in this case – and return you to the Windows command line.

3. Run a Docker Container with NGINX

In this section, we are going to run a Docker Container with NGINX inside. You could call this container a microservice: it handles web serving.

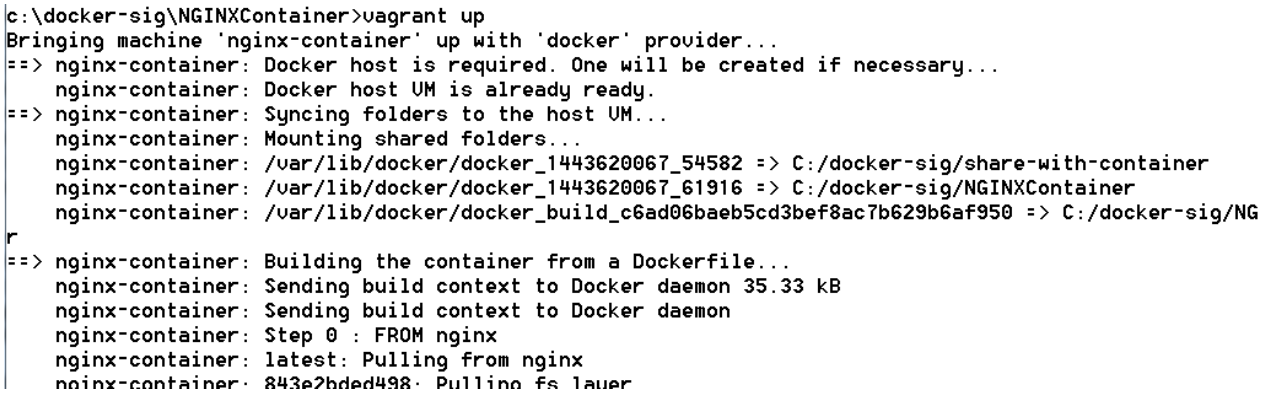
Open a command line in directory C:\docker-sig\NGINXContainer. It contains a Vagrantfile that references the same DockerHostVagrantfile we saw before (because the container should be created inside the same dockerhost VM). It also references a dockerfile that in turn contains the FROM nginx instruction. The Vagrantfile also specifies a port forwarding that maps port 8080 on the dockerhost VM to port 80 in the Docker container.



```
ENV['VAGRANT_DEFAULT_PROVIDER'] = 'docker'
Vagrant.configure("2") do |config|

  config.vm.synced_folder "c:/docker-sig/share-with-container", "/host_share"

  config.vm.define "nginx-container" do |m|

    m.vm.provider :docker do |d|
      d.build_dir = "./docker"
      d.name = 'nginx-container'
      d.vagrant_machine = "dockerhost"
      d.vagrant_vagrantfile = "../DockerHost/DockerHostVagrantfile"
      d.remains_running = true
      # forward port 8080 on dockerhost to port 80 in container nginx-container
      d.ports = ["8080:80"]
      end
  end

end
```

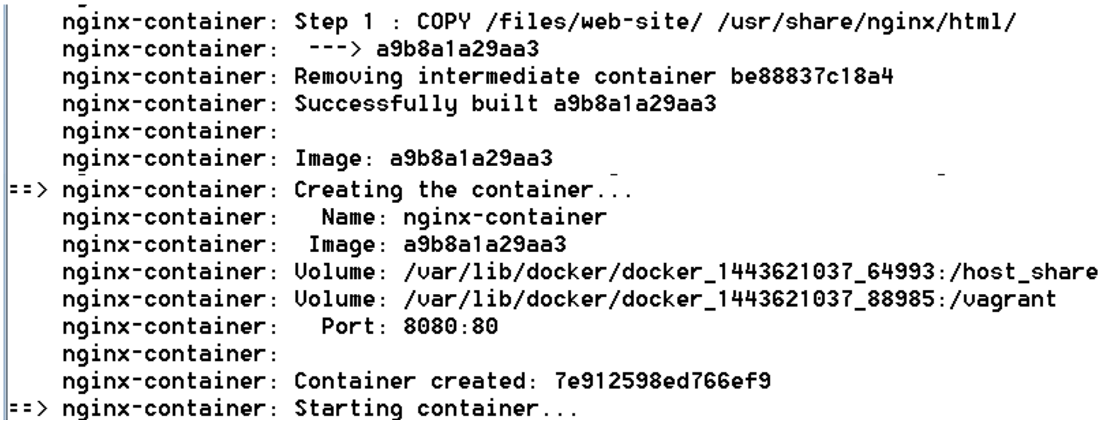
Run the command to build and start the Docker container:

```
vagrant up
```



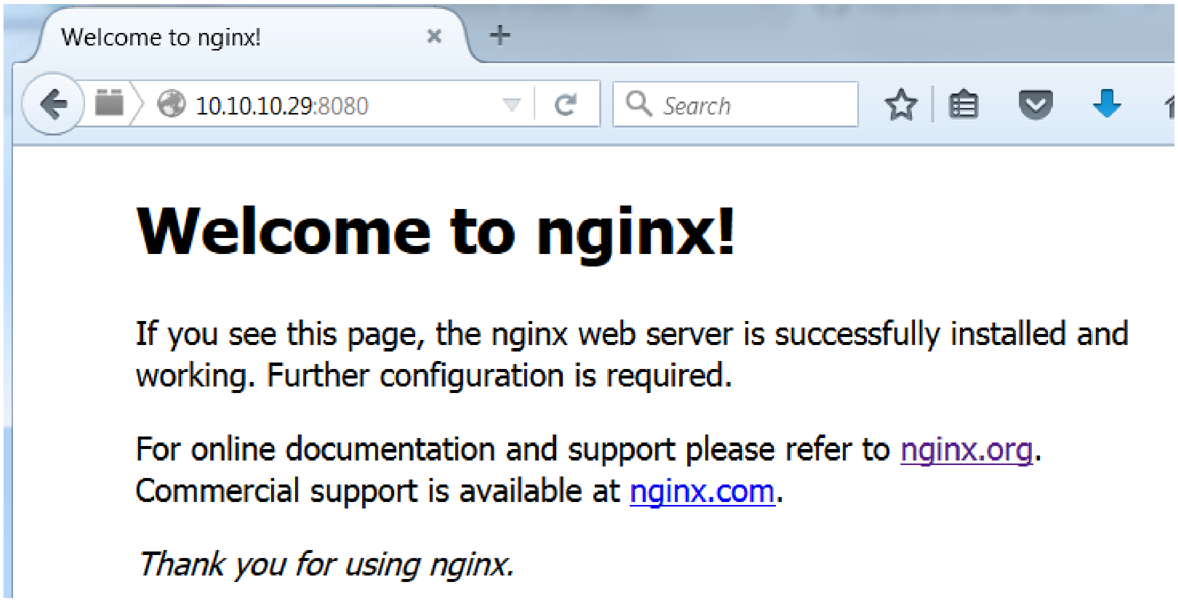
```
c:\docker-sig\NGINXContainer>vagrant up
Bringing machine 'nginx-container' up with 'docker' provider...
==> nginx-container: Docker host is required. One will be created if necessary...
    nginx-container: Docker host VM is already ready.
==> nginx-container: Syncing folders to the host VM...
    nginx-container: Mounting shared folders...
    nginx-container: /var/lib/docker/docker_1443620067_54582 => C:/docker-sig/share-with-container
    nginx-container: /var/lib/docker/docker_1443620067_61916 => C:/docker-sig/NGINXContainer
    nginx-container: /var/lib/docker/docker_build_c6ad06baeb5cd3bef8ac7b629b6af950 => C:/docker-sig/NG
r
==> nginx-container: Building the container from a Dockerfile...
    nginx-container: Sending build context to Docker daemon 35.33 kB
    nginx-container: Sending build context to Docker daemon
    nginx-container: Step 0 : FROM nginx
    nginx-container: latest: Pulling from nginx
    nginx-container: 843e2bded498: Pulling fs layer
```

When the base image is pulled, the files are added to the container and the image is created and the container started:



```
    nginx-container: Step 1 : COPY /files/web-site/ /usr/share/nginx/html/
    nginx-container:  ---> a9b8a1a29aa3
    nginx-container: Removing intermediate container be88837c18a4
    nginx-container: Successfully built a9b8a1a29aa3
    nginx-container:
    nginx-container: Image: a9b8a1a29aa3
==> nginx-container: Creating the container...
    nginx-container:   Name: nginx-container
    nginx-container:  Image: a9b8a1a29aa3
    nginx-container: Volume: /var/lib/docker/docker_1443621037_64993:/host_share
    nginx-container: Volume: /var/lib/docker/docker_1443621037_88985:/vagrant
    nginx-container:   Port: 8080:80
    nginx-container:
    nginx-container: Container created: 7e912598ed766ef9
==> nginx-container: Starting container...
```

On your host machine, open a browser and enter the following URL in the location bar: http://10.10.10.29:8080/ . This references port 8080 on the dockerhost VM which is forwarded to port 80 in the Docker container where NGINX is listening for such requests:



Welcome to nginx!

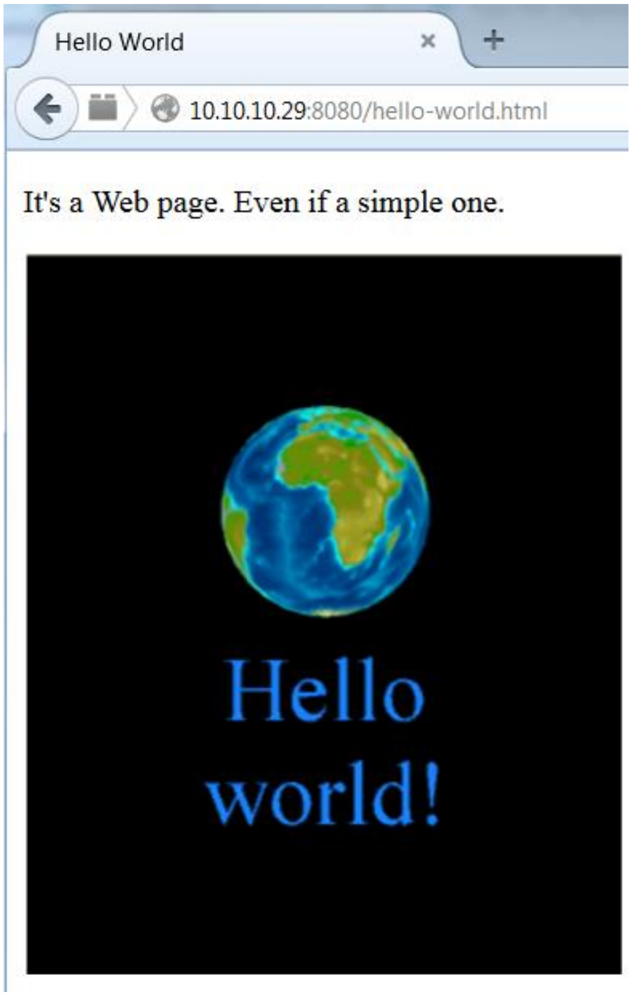
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

This proves that the container is running inside the dockerhost VM, listening to port 80 and serving its stuff.

Open the URL http://10.10.10.29:8080/hello-world.html in your host browser. This loads the files from NGINX that were copied into the Docker container during the build process from directory C:\docker-sig\NGINXContainer\docker\files\web-site.



It's a Web page. Even if a simple one.

4. Docker Container Provisioning with Puppet

As was discussed during the presentation prior to the hands on: building containers with just a Dockerfile is really a step back in term of configuration management. Tools such as Puppet, Chef, Ansible and Salt have evolved to make the fine grained configuration of environments a declarative process, not driven by manually created platform specific Shell scripts but by cross platform, easily parameterizable declarative configuration files.

It seems like a good approach to combine the two: use Docker build for the initial steps from a base image and some broad set up actions and then apply Puppet for the fine grained tuning of the configuration. The best separation of concerns is yet to be determined – and is a matter of personal taste as well.

In this section, we will first use Vagrant to make Docker build a container – based on a Dockerfile. This container is 'puppet enabled' in the build phase. Subsequently, the container is further provisioned using Puppet to apply fined grained configuration. Note: a more in depth discussion of the use of Puppet with Docker is provided in this article: https://technology.amis.nl/2015/08/26/vagrant-and-docker-followed-by-puppet-to-provision-complex-environments/ .

Open a command line window in directory C:\docker-sig\PuppetIntro. The Dockerfile in the subdirectory docker contains instructions for setting up Puppet during the Build. The directories C:\docker-sig\PuppetIntro\docker\manifests and C:\docker-sig\PuppetIntro\docker\modules contain the Puppet configuration files; these are copied into the container during the build operation.

Enter the command:

```
vagrant up
```

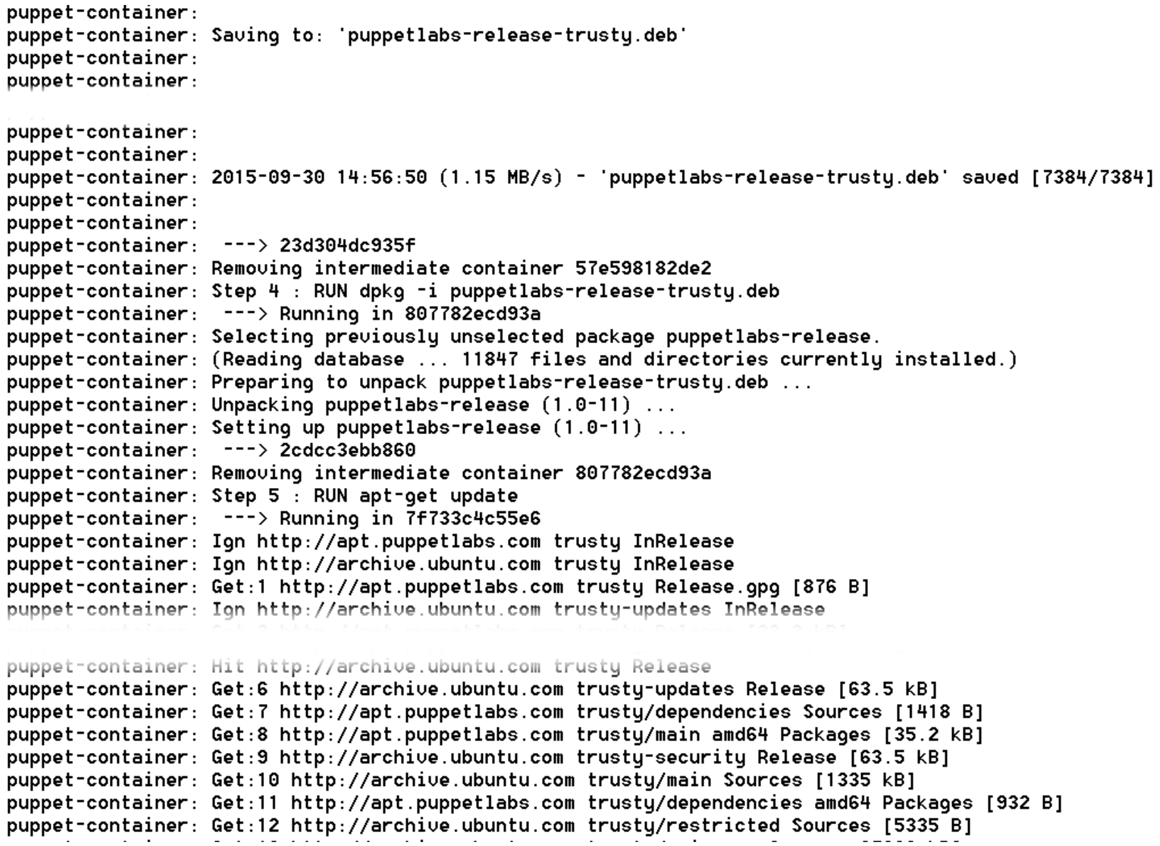
This will lead to the construction of the container.

The initial steps look familiar:

```
c:\docker-sig\PuppetIntro>vagrant up
Bringing machine 'puppet-container' up with 'docker' provider...
==> puppet-container: Docker host is required. One will be created if necessary...
    puppet-container: Docker host VM is already ready.
==> puppet-container: Building the container from a Dockerfile...
    puppet-container: Sending build context to Docker daemon 4.608 kB
    puppet-container: Sending build context to Docker daemon
    puppet-container: Step 0 : FROM ubuntu:14.04
    puppet-container:  ---> 91e54dfb1179
    puppet-container: Step 1 : RUN mkdir /u01 &&     chmod a+xr /u01
    puppet-container:  ---> Using cache
    puppet-container:  ---> 401cfd01a31f
    puppet-container: Step 2 : RUN apt-get install -q -y wget
```

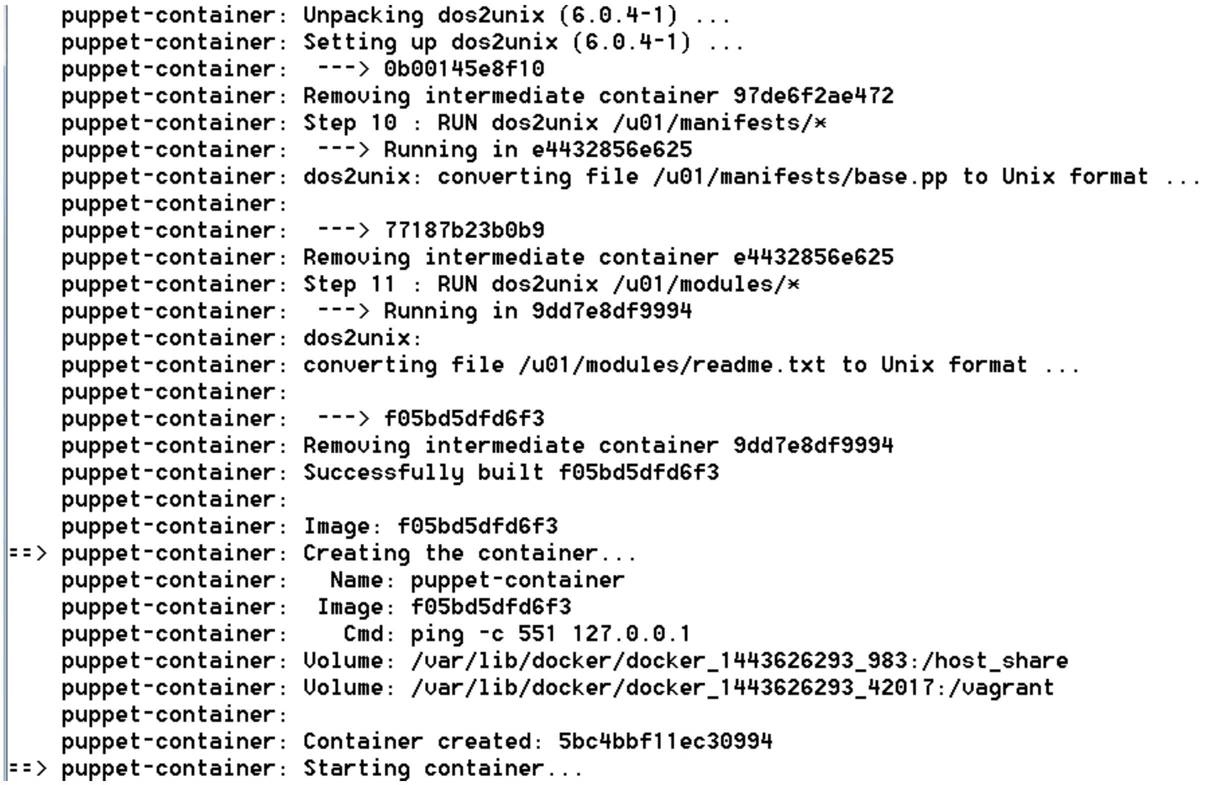
The Ubunu facility apt-get is used during the build to install some additional packages on top of the based Ubuntu image.

Various packages are installed to enable the Puppet support in the container:



```
puppet-container:
puppet-container: Saving to: 'puppetlabs-release-trusty.deb'
puppet-container:
puppet-container:

puppet-container:
puppet-container:
puppet-container: 2015-09-30 14:56:50 (1.15 MB/s) - 'puppetlabs-release-trusty.deb' saved [7384/7384]
puppet-container:
puppet-container:
puppet-container:  ---> 23d304dc935f
puppet-container: Removing intermediate container 57e598182de2
puppet-container: Step 4 : RUN dpkg -i puppetlabs-release-trusty.deb
puppet-container:  ---> Running in 807782ecd93a
puppet-container: Selecting previously unselected package puppetlabs-release.
puppet-container: (Reading database ... 11847 files and directories currently installed.)
puppet-container: Preparing to unpack puppetlabs-release-trusty.deb ...
puppet-container: Unpacking puppetlabs-release (1.0-11) ...
puppet-container: Setting up puppetlabs-release (1.0-11) ...
puppet-container:  ---> 2cdcc3ebb860
puppet-container: Removing intermediate container 807782ecd93a
puppet-container: Step 5 : RUN apt-get update
puppet-container:  ---> Running in 7f733c4c55e6
puppet-container: Ign http://apt.puppetlabs.com trusty InRelease
puppet-container: Ign http://archive.ubuntu.com trusty InRelease
puppet-container: Get:1 http://apt.puppetlabs.com trusty Release.gpg [876 B]
puppet-container: Ign http://archive.ubuntu.com trusty-updates InRelease

puppet-container: Hit http://archive.ubuntu.com trusty Release
puppet-container: Get:6 http://archive.ubuntu.com trusty-updates Release [63.5 kB]
puppet-container: Get:7 http://apt.puppetlabs.com trusty/dependencies Sources [1418 B]
puppet-container: Get:8 http://apt.puppetlabs.com trusty/main amd64 Packages [35.2 kB]
puppet-container: Get:9 http://archive.ubuntu.com trusty-security Release [63.5 kB]
puppet-container: Get:10 http://archive.ubuntu.com trusty/main Sources [1335 kB]
puppet-container: Get:11 http://apt.puppetlabs.com trusty/dependencies amd64 Packages [932 B]
puppet-container: Get:12 http://archive.ubuntu.com trusty/restricted Sources [5335 B]
```

The final steps in the build process:



```
    puppet-container: Unpacking dos2unix (6.0.4-1) ...
    puppet-container: Setting up dos2unix (6.0.4-1) ...
    puppet-container:  ---> 0b00145e8f10
    puppet-container: Removing intermediate container 97de6f2ae472
    puppet-container: Step 10 : RUN dos2unix /u01/manifests/*
    puppet-container:  ---> Running in e4432856e625
    puppet-container: dos2unix: converting file /u01/manifests/base.pp to Unix format ...
    puppet-container:
    puppet-container:  ---> 77187b23b0b9
    puppet-container: Removing intermediate container e4432856e625
    puppet-container: Step 11 : RUN dos2unix /u01/modules/*
    puppet-container:  ---> Running in 9dd7e8df9994
    puppet-container: dos2unix:
    puppet-container: converting file /u01/modules/readme.txt to Unix format ...
    puppet-container:
    puppet-container:  ---> f05bd5dfd6f3
    puppet-container: Removing intermediate container 9dd7e8df9994
    puppet-container: Successfully built f05bd5dfd6f3
    puppet-container:
    puppet-container: Image: f05bd5dfd6f3
==> puppet-container: Creating the container...
    puppet-container:   Name: puppet-container
    puppet-container:  Image: f05bd5dfd6f3
    puppet-container:    Cmd: ping -c 551 127.0.0.1
    puppet-container: Volume: /var/lib/docker/docker_1443626293_983:/host_share
    puppet-container: Volume: /var/lib/docker/docker_1443626293_42017:/vagrant
    puppet-container:
    puppet-container: Container created: 5bc4bbf11ec30994
==> puppet-container: Starting container...
```

When the container is ready, we need to use vagrant ssh to get into the dockerhost. Inside the host, we will run the container and perform the configuration through Puppet.

Use:

```
vagrant global-status
```

to learn the id for the dockerhost VM.

Type:

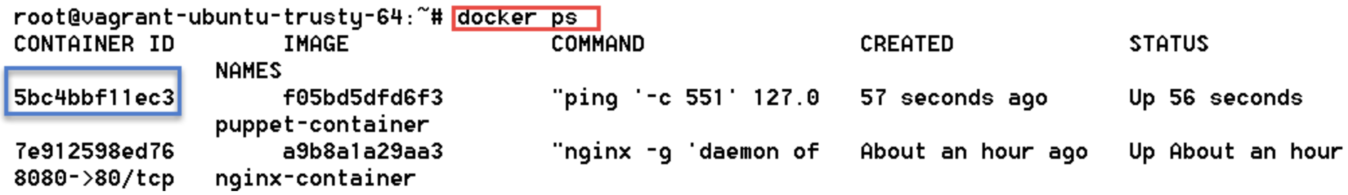
```
vagrant ssh <dockerhost VM ID>
```

to open a terminal session into the dockerhost VM.

Use

```
docker ps
```

to learn about the container id for the puppet-container.



```
root@vagrant-ubuntu-trusty-64:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
              NAMES
5bc4bbf11ec3        f05bd5dfd6f3        "ping '-c 551' 127.0     57 seconds ago      Up 56 seconds
              puppet-container
7e912598ed76        a9b8a1a29aa3        "nginx -g 'daemon of     About an hour ago   Up About an hour
8080->80/tcp  nginx-container
```

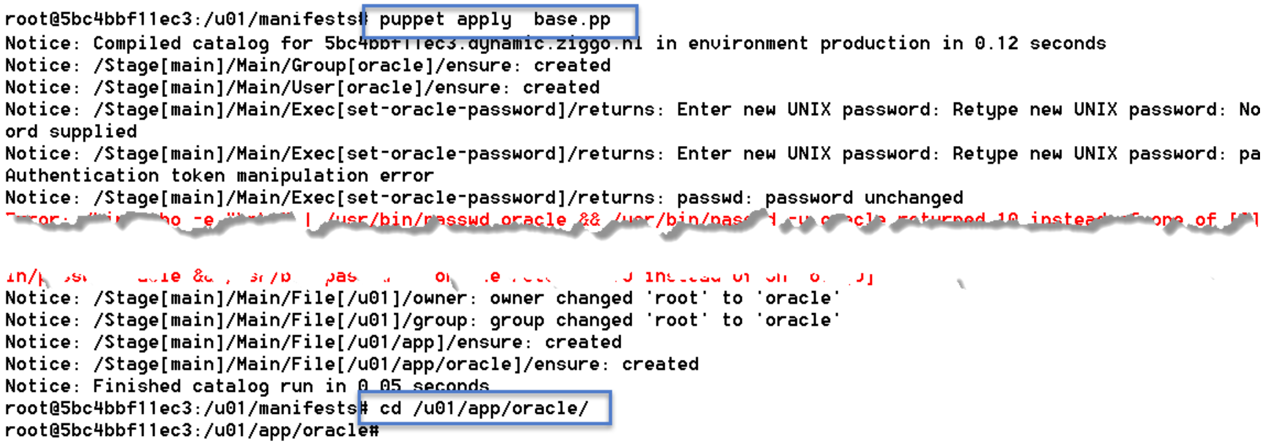
Then use

```
docker exec -it <container id> bash
```

to open a shell in the container. Once inside the container, the following command will make Puppet complete the configuration based on the declarations in the base.pp manifest file.

```
puppet apply /u01/manifests/base.pp
```

Once Puppet completes, the container is in the desired state. Verify that folders /u01/app and /u01/app/oracle were created – as specified in the base.pp file.



```
root@5bc4bbf11ec3:/u01/manifests# puppet apply  base.pp
Notice: Compiled catalog for 5bc4bbf11ec3.dynamic.ziggo.nl in environment production in 0.12 seconds
Notice: /Stage[main]/Main/Group[oracle]/ensure: created
Notice: /Stage[main]/Main/User[oracle]/ensure: created
Notice: /Stage[main]/Main/Exec[set-oracle-password]/returns: Enter new UNIX password: Retype new UNIX password: No
ord supplied
Notice: /Stage[main]/Main/Exec[set-oracle-password]/returns: Enter new UNIX password: Retype new UNIX password: pa
Authentication token manipulation error
Notice: /Stage[main]/Main/Exec[set-oracle-password]/returns: passwd: password unchanged
Error: [illegible] | /usr/bin/passwd oracle && /usr/bin/pas[illegible] oracle returned 10 instead of one of [0]
in/[illegible] instead of one of [0]
Notice: /Stage[main]/Main/File[/u01]/owner: owner changed 'root' to 'oracle'
Notice: /Stage[main]/Main/File[/u01]/group: group changed 'root' to 'oracle'
Notice: /Stage[main]/Main/File[/u01/app]/ensure: created
Notice: /Stage[main]/Main/File[/u01/app/oracle]/ensure: created
Notice: Finished catalog run in 0.05 seconds
root@5bc4bbf11ec3:/u01/manifests# cd /u01/app/oracle/
root@5bc4bbf11ec3:/u01/app/oracle#
```

We now need to preserve this state in the form of a new image.

Exit the container – by typing

```
exit
```

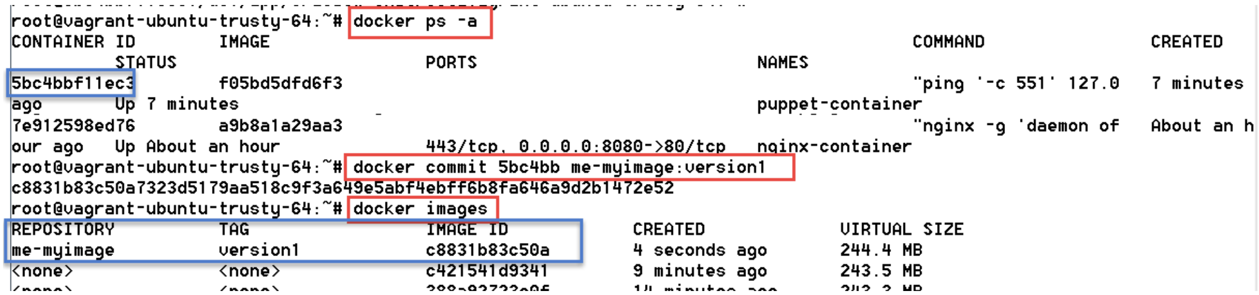
Then user docker commit to turn the container into a tagged image:

```
docker commit <container id> me-myimage:version1
```

List all images with

```
docker images
```

and find the new image in the list



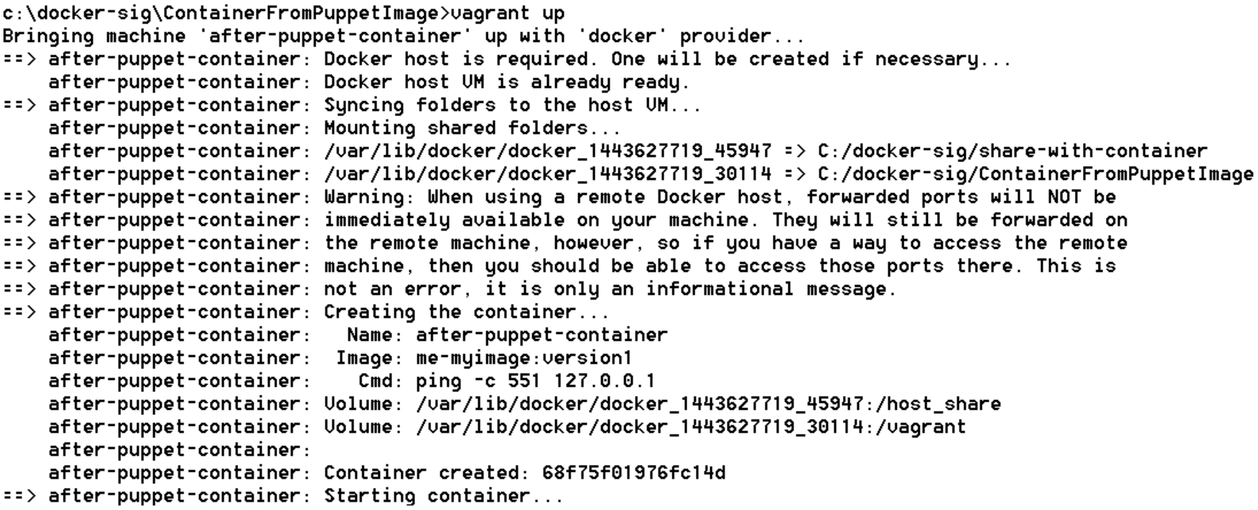
```
root@vagrant-ubuntu-trusty-64:~# docker ps -a
CONTAINER ID        IMAGE                                                   COMMAND              CREATED
             STATUS                       PORTS                          NAMES
5bc4bbf11ec3        f05bd5dfd6f3                                            "ping '-c 551' 127.0   7 minutes
ago          Up 7 minutes                                                puppet-container
7e912598ed76        a9b8a1a29aa3                                            "nginx -g 'daemon of   About an h
our ago      Up About an hour             443/tcp, 0.0.0.0:8080->80/tcp   nginx-container
root@vagrant-ubuntu-trusty-64:~# docker commit 5bc4bb me-myimage:version1
c8831b83c50a7323d5179aa518c9f3a649e5abf4ebff6b8fa646a9d2b1472e52
root@vagrant-ubuntu-trusty-64:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
me-myimage          version1            c8831b83c50a        4 seconds ago       244.4 MB
<none>              <none>              c421541d9341        9 minutes ago       243.5 MB
```

At this point we can create a new container based on our image. Since that image is created based on the Dockerfile complemented by the Puppet manifest, any container started from it is completely configured.

Return to the Windows command line. Navigate to C:\docker-sig\ContainerFromPuppetImage. This directory only contains a Vagrantfile. This file refers to the newly created image me-myimage:version1 that was the final result after Docker build and Puppet apply. We can now have Vagrant create and run a fresh container based on that image.

Type:

```
vagrant up
```



```
c:\docker-sig\ContainerFromPuppetImage>vagrant up
Bringing machine 'after-puppet-container' up with 'docker' provider...
==> after-puppet-container: Docker host is required. One will be created if necessary...
    after-puppet-container: Docker host VM is already ready.
==> after-puppet-container: Syncing folders to the host VM...
    after-puppet-container: Mounting shared folders...
    after-puppet-container: /var/lib/docker/docker_1443627719_45947 => C:/docker-sig/share-with-container
    after-puppet-container: /var/lib/docker/docker_1443627719_30114 => C:/docker-sig/ContainerFromPuppetImage
==> after-puppet-container: Warning: When using a remote Docker host, forwarded ports will NOT be
==> after-puppet-container: immediately available on your machine. They will still be forwarded on
==> after-puppet-container: the remote machine, however, so if you have a way to access the remote
==> after-puppet-container: machine, then you should be able to access those ports there. This is
==> after-puppet-container: not an error, it is only an informational message.
==> after-puppet-container: Creating the container...
    after-puppet-container:   Name: after-puppet-container
    after-puppet-container:  Image: me-myimage:version1
    after-puppet-container:    Cmd: ping -c 551 127.0.0.1
    after-puppet-container: Volume: /var/lib/docker/docker_1443627719_45947:/host_share
    after-puppet-container: Volume: /var/lib/docker/docker_1443627719_30114:/vagrant
    after-puppet-container:
    after-puppet-container: Container created: 68f75f01976fc14d
==> after-puppet-container: Starting container...
```

The container has started and contains all the effects from both the Docker build and the Puppet apply.

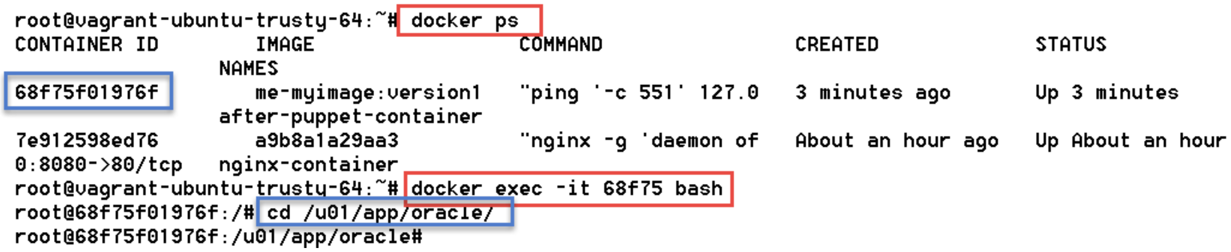
One way to check this out is from the dockerhost VM. List the running containers with

```
docker ps
```

Find the container id for the container just started by Vagrant. Get into this container using

```
docker exec -it <container id> bash
```

Verify whether the directories to be created by Puppet are indeed in the container.



```
root@vagrant-ubuntu-trusty-64:~# docker ps
CONTAINER ID        IMAGE                COMMAND               CREATED             STATUS
                 NAMES
68f75f01976f           me-myimage:version1   "ping '-c 551' 127.0   3 minutes ago       Up 3 minutes
                 after-puppet-container
7e912598ed76           a9b8a1a29aa3          "nginx -g 'daemon of   About an hour ago   Up About an hour
0:8080->80/tcp   nginx-container
root@vagrant-ubuntu-trusty-64:~# docker exec -it 68f75 bash
root@68f75f01976f:/# cd /u01/app/oracle/
root@68f75f01976f:/u01/app/oracle#
```

This should give us some confidence to try our hand at more complex Puppet configuration scenarios – involving Java, Database and Application Server perhaps.

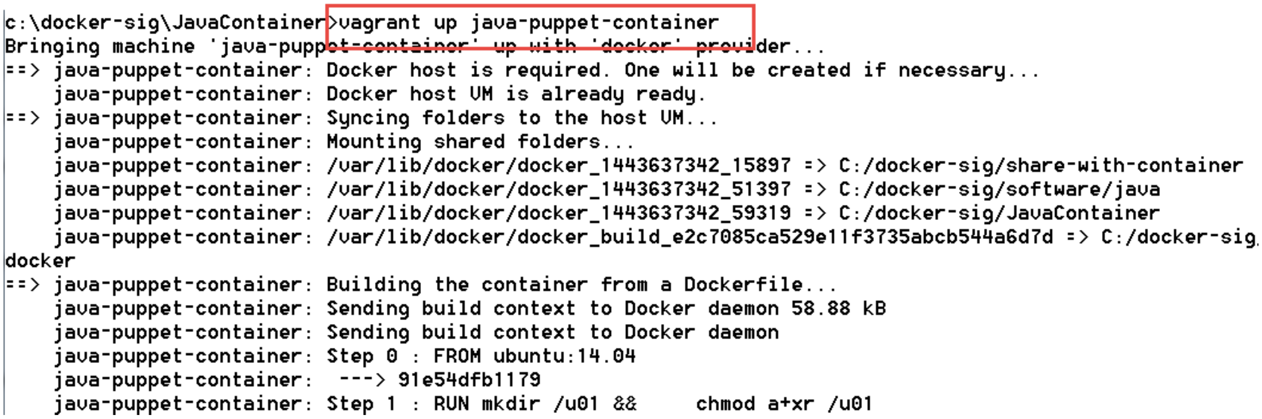
5. Provision Docker Container with Java 7 (JDK 7U79)

In this section, we will make use of a somewhat more complex Puppet manifest that will install the Java Development Kit (JDK) into a Docker container. The steps are by and large the same as in the previous section – however with a little bit more complexity and result.

Open a command window and navigate to directory C:\docker-sig\JavaContainer. You may want to check the Vagrantfile in this directory. It defines two Docker Containers: the first one that is the intermediate build vehicle and the second one that is the final result – the reusable Java Container.

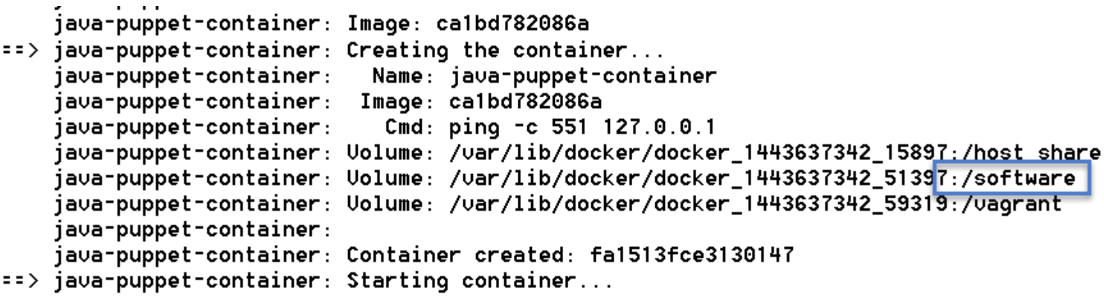
Start the creation of the intermediate container by typing

```
vagrant up java-puppet-container
```



```
c:\docker-sig\JavaContainer>vagrant up java-puppet-container
Bringing machine 'java-puppet-container' up with 'docker' provider...
==> java-puppet-container: Docker host is required. One will be created if necessary...
    java-puppet-container: Docker host VM is already ready.
==> java-puppet-container: Syncing folders to the host VM...
    java-puppet-container: Mounting shared folders...
    java-puppet-container: /var/lib/docker/docker_1443637342_15897 => C:/docker-sig/share-with-container
    java-puppet-container: /var/lib/docker/docker_1443637342_51397 => C:/docker-sig/software/java
    java-puppet-container: /var/lib/docker/docker_1443637342_59319 => C:/docker-sig/JavaContainer
    java-puppet-container: /var/lib/docker/docker_build_e2c7085ca529e11f3735abcb544a6d7d => C:/docker-sig
docker
==> java-puppet-container: Building the container from a Dockerfile...
    java-puppet-container: Sending build context to Docker daemon 58.88 kB
    java-puppet-container: Sending build context to Docker daemon
    java-puppet-container: Step 0 : FROM ubuntu:14.04
    java-puppet-container:  ---> 91e54dfb1179
    java-puppet-container: Step 1 : RUN mkdir /u01 &&     chmod a+xr /u01
```

The build process progresses through the build steps and after a little while arrives at:



```
    java-puppet-container: Image: ca1bd782086a
==> java-puppet-container: Creating the container...
    java-puppet-container:   Name: java-puppet-container
    java-puppet-container:  Image: ca1bd782086a
    java-puppet-container:    Cmd: ping -c 551 127.0.0.1
    java-puppet-container: Volume: /var/lib/docker/docker_1443637342_15897:/host_share
    java-puppet-container: Volume: /var/lib/docker/docker_1443637342_51397:/software
    java-puppet-container: Volume: /var/lib/docker/docker_1443637342_59319:/vagrant
    java-puppet-container:
    java-puppet-container: Container created: fa1513fce3130147
==> java-puppet-container: Starting container...
```

Note the mapping to the container (mounted as /software) from the host directory C:\docker-sig\software\java. This folder contains the installer for the JDK and a supporting file.

As before, from a terminal in the dockerhost VM (use vagrant global-status to find the ID of the VM, then use vagrant ssh <VM ID> to open the secure shell), check for the running containers:

```
docker ps
```

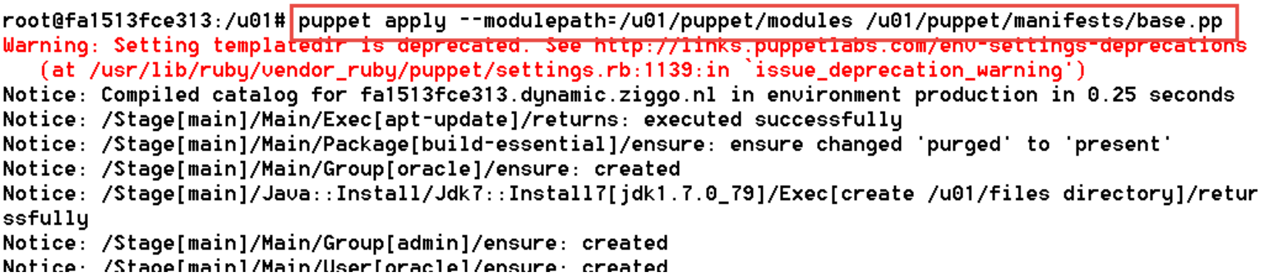
Learn the container id for the java-puppet-container. Next, use

```
docker exec -it <container id> bash
```

to open a shell into this container.

With the following command Puppet is activated to perform the configuration of the container, based on the base.pp manifest:

```
puppet apply --modulepath=/u01/puppet/modules
/u01/puppet/manifests/base.pp
```

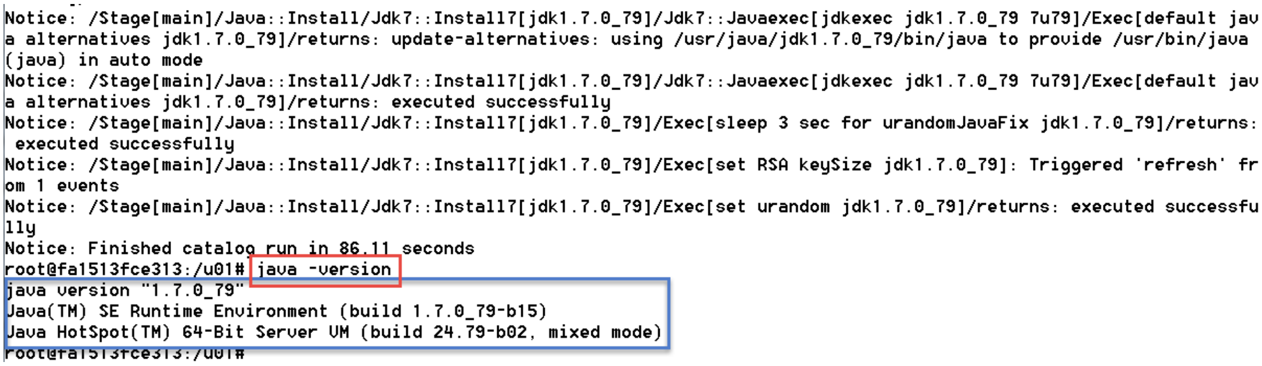


```
root@fa1513fce313:/u01# puppet apply --modulepath=/u01/puppet/modules /u01/puppet/manifests/base.pp
Warning: Setting templatedir is deprecated. See http://links.puppetlabs.com/env-settings-deprecations
   (at /usr/lib/ruby/vendor_ruby/puppet/settings.rb:1139:in `issue_deprecation_warning')
Notice: Compiled catalog for fa1513fce313.dynamic.ziggo.nl in environment production in 0.25 seconds
Notice: /Stage[main]/Main/Exec[apt-update]/returns: executed successfully
Notice: /Stage[main]/Main/Package[build-essential]/ensure: ensure changed 'purged' to 'present'
Notice: /Stage[main]/Main/Group[oracle]/ensure: created
Notice: /Stage[main]/Java::Install/Jdk7::Install7[jdk1.7.0_79]/Exec[create /u01/files directory]/retur
ssfully
Notice: /Stage[main]/Main/Group[admin]/ensure: created
Notice: /Stage[main]/Main/User[oracle]/ensure: created
```

after a little while, Puppet completes. With

```
java -version
```

you can test whether the installation of the JDK was successful.



```
Notice: /Stage[main]/Java::Install/Jdk7::Install7[jdk1.7.0_79]/Jdk7::Javaexec[jdkexec jdk1.7.0_79 7u79]/Exec[default jav
a alternatives jdk1.7.0_79]/returns: update-alternatives: using /usr/java/jdk1.7.0_79/bin/java to provide /usr/bin/java
(java) in auto mode
Notice: /Stage[main]/Java::Install/Jdk7::Install7[jdk1.7.0_79]/Jdk7::Javaexec[jdkexec jdk1.7.0_79 7u79]/Exec[default jav
a alternatives jdk1.7.0_79]/returns: executed successfully
Notice: /Stage[main]/Java::Install/Jdk7::Install7[jdk1.7.0_79]/Exec[sleep 3 sec for urandomJavaFix jdk1.7.0_79]/returns:
 executed successfully
Notice: /Stage[main]/Java::Install/Jdk7::Install7[jdk1.7.0_79]/Exec[set RSA keySize jdk1.7.0_79]: Triggered 'refresh' fr
om 1 events
Notice: /Stage[main]/Java::Install/Jdk7::Install7[jdk1.7.0_79]/Exec[set urandom jdk1.7.0_79]/returns: executed successfu
lly
Notice: Finished catalog run in 86.11 seconds
root@fa1513fce313:/u01# java -version
java version "1.7.0_79"
Java(TM) SE Runtime Environment (build 1.7.0_79-b15)
Java HotSpot(TM) 64-Bit Server VM (build 24.79-b02, mixed mode)
root@fa1513fce313:/u01#
```

Exit the container – by typing

```
exit
```

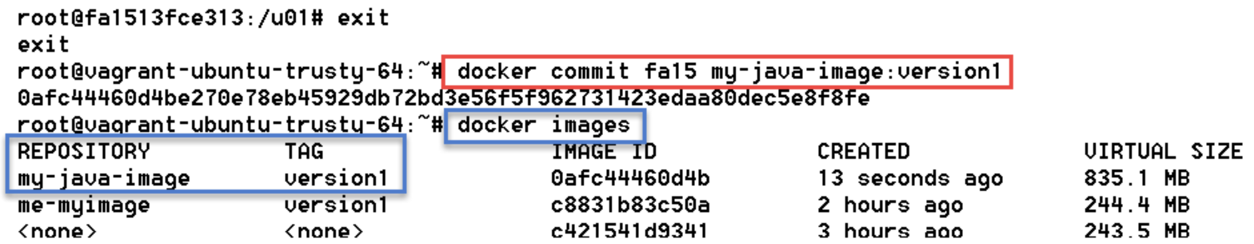
Then user docker commit to turn the container into a tagged image:

```
docker commit <container id> me-java-image:version1
```

List all images with

```
docker images
```

and find the new image in the list

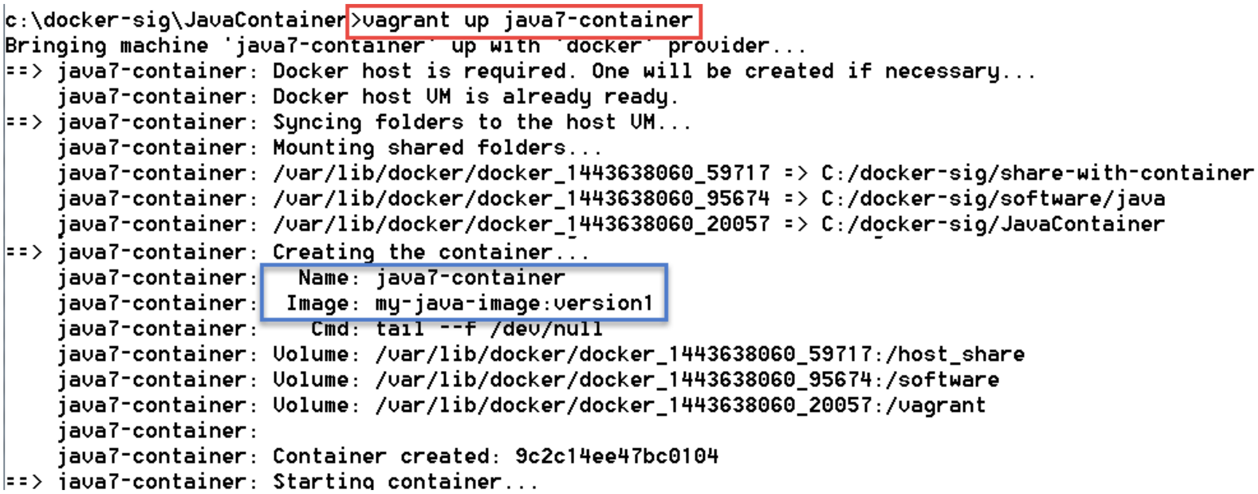


```
root@fa1513fce313:/u01# exit
exit
root@vagrant-ubuntu-trusty-64:~# docker commit fa15 my-java-image:version1
0afc44460d4be270e78eb45929db72bd3e56f5f962731423edaa80dec5e8f8fe
root@vagrant-ubuntu-trusty-64:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
my-java-image       version1            0afc44460d4b        13 seconds ago      835.1 MB
me-myimage          version1            c8831b83c50a        2 hours ago         244.4 MB
<none>              <none>              c421541d9341        3 hours ago         243.5 MB
```

From the Windows Command Line, run a new container based on the new image:

```
vagrant up java7-container
```

This refers to the second Docker container definition in the Vagrantfile. This one is based on the image my-java-image:version1 that we have created after the Puppet apply action was complete.



```
c:\docker-sig\JavaContainer>vagrant up java7-container
Bringing machine 'java7-container' up with 'docker' provider...
==> java7-container: Docker host is required. One will be created if necessary...
    java7-container: Docker host VM is already ready.
==> java7-container: Syncing folders to the host VM...
    java7-container: Mounting shared folders...
    java7-container: /var/lib/docker/docker_1443638060_59717 => C:/docker-sig/share-with-container
    java7-container: /var/lib/docker/docker_1443638060_95674 => C:/docker-sig/software/java
    java7-container: /var/lib/docker/docker_1443638060_20057 => C:/docker-sig/JavaContainer
==> java7-container: Creating the container...
    java7-container:   Name: java7-container
    java7-container:  Image: my-java-image:version1
    java7-container:    Cmd: tail -f /dev/null
    java7-container: Volume: /var/lib/docker/docker_1443638060_59717:/host_share
    java7-container: Volume: /var/lib/docker/docker_1443638060_95674:/software
    java7-container: Volume: /var/lib/docker/docker_1443638060_20057:/vagrant
    java7-container:
    java7-container: Container created: 9c2c14ee47bc0104
==> java7-container: Starting container...
```

In the secure shell in the dockerhost, we can check for the running containers:

```
docker ps
```

You should see the java7-container running. With

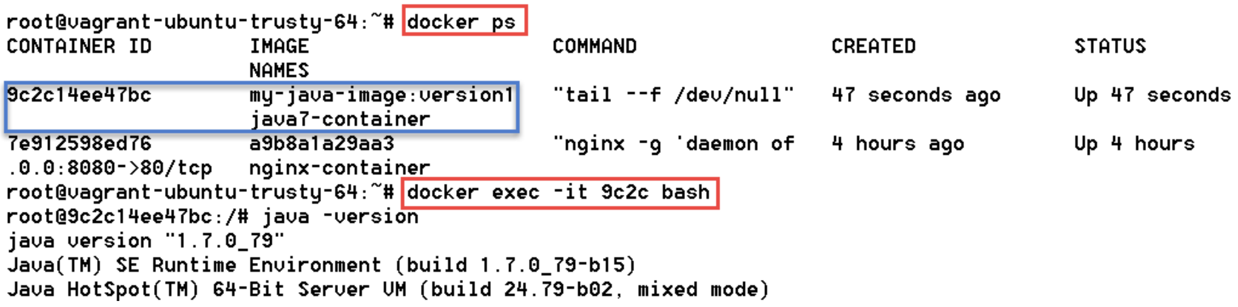
```
docker –it <container id> bash
```

can open a shell in the running container.

Type

```
java –version
```

‘to verify if the JDK has indeed been installed:



```
root@vagrant-ubuntu-trusty-64:~# docker ps
CONTAINER ID        IMAGE                    COMMAND                 CREATED             STATUS
                    NAMES
9c2c14ee47bc        my-java-image:version1   "tail --f /dev/null"    47 seconds ago      Up 47 seconds
                    java7-container
7e912598ed76        a9b8a1a29aa3             "nginx -g 'daemon of    4 hours ago         Up 4 hours
.0.0:8080->80/tcp   nginx-container
root@vagrant-ubuntu-trusty-64:~# docker exec -it 9c2c bash
root@9c2c14ee47bc:/# java -version
java version "1.7.0_79"
Java(TM) SE Runtime Environment (build 1.7.0_79-b15)
Java HotSpot(TM) 64-Bit Server VM (build 24.79-b02, mixed mode)
```

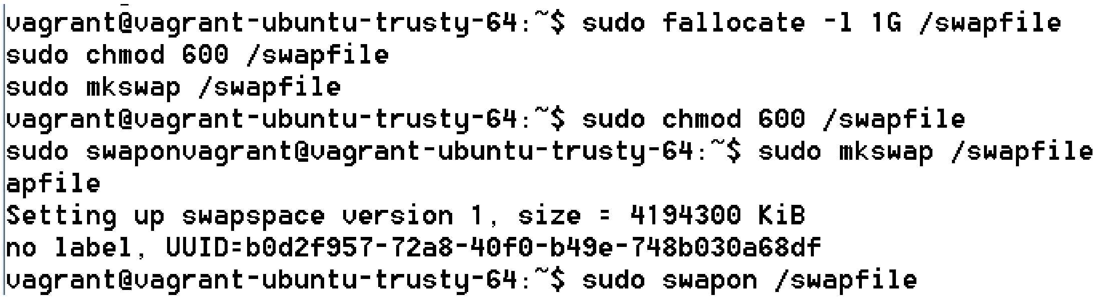
6. Building a Docker Container with the Oracle 11g XE Database

Creating a Docker Container that contains an Oracle Database (in this case the 11g XE database) is the subject for this section. Several Puppet module and manifest combinations are available for declaratively configuring a database on a Linux host. Using the same two-step approach we adopted in the previous two sections: Docker build (initiated through Vagrant) and Puppet apply inside the intermediate container for the advanced configuration, we should be able to create the oraclexe-container.

However, it turns out there are some difficulties with this approach. Specifically, the installer for Oracle Database 11g XE tries to do several things that are not allowed inside a Docker Container. So instead of doing this the proper but rather hard way – we will leverage the work of our peers and run our container with the Oracle XE database from a predefined image. Note: this image infringes on Oracle's copyright in a sense that it ships Oracle software that we can download from OTN ourselves after accepting the OTN license conditions, but that the author of the Docker Image is not allowed to distribute. I have not been able to find out a simple, foolproof way to configure XE on a Docker Container, so for now we will benefit from the slight rule bending and consume the image.

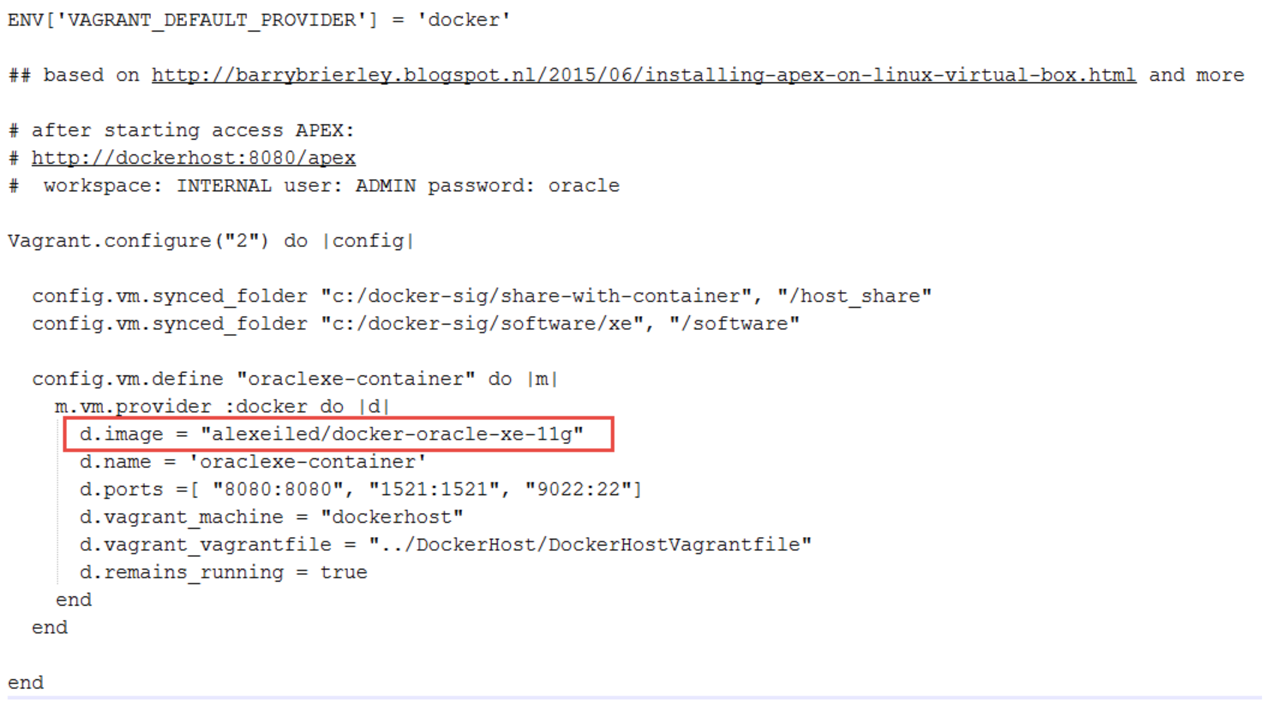
Before attempting to run this image, we first need to ensure that the Linux dockerhost has a swapfile set up. In the shell in the dockerhost VM, execute these statements, that set up the swapfile. Note that the container will inherit this swapfile configuration (and setting up a swapfile inside a container does not seem to be possible or even desirable)

```
sudo fallocate -l 3G /swapfile

sudo chmod 600 /swapfile

sudo mkswap /swapfile

sudo swapon /swapfile
```



```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo fallocate -l 1G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
vagrant@vagrant-ubuntu-trusty-64:~$ sudo chmod 600 /swapfile
sudo swaponvagrant@vagrant-ubuntu-trusty-64:~$ sudo mkswap /swapfile
apfile
Setting up swapspace version 1, size = 4194300 KiB
no label, UUID=b0d2f957-72a8-40f0-b49e-748b030a68df
vagrant@vagrant-ubuntu-trusty-64:~$ sudo swapon /swapfile
```

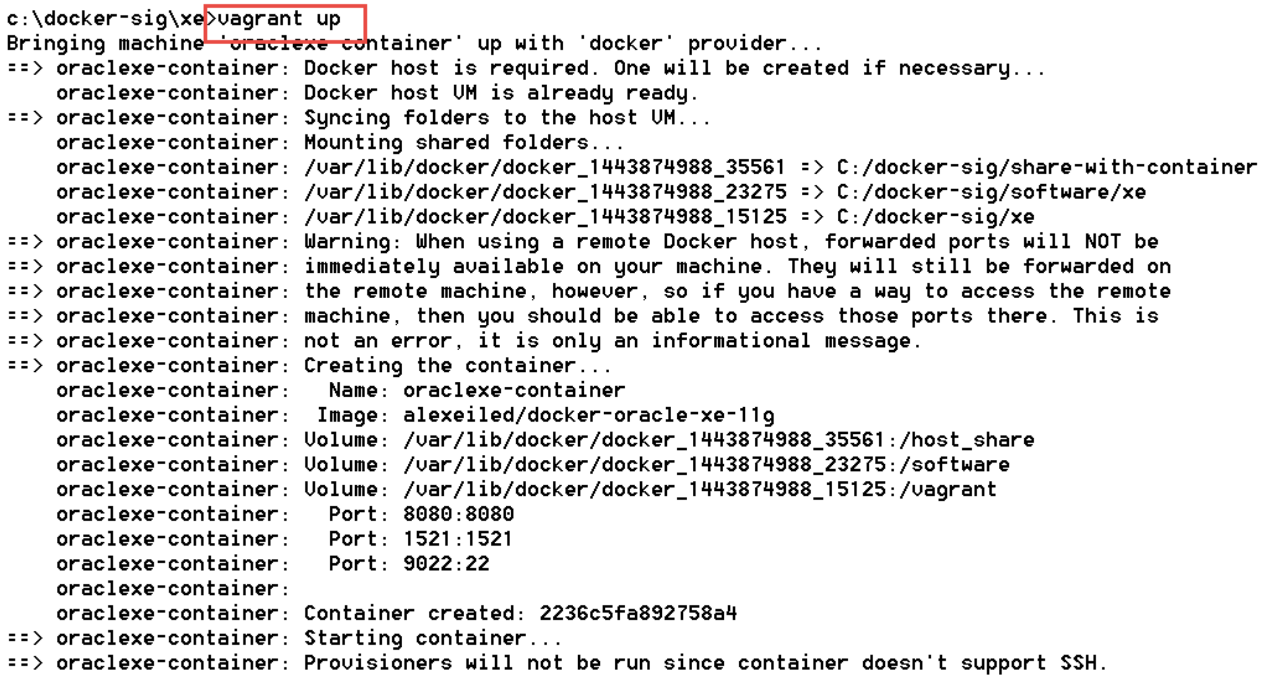
Open a command line window and navigate to directory C:\docker-sig\xe. The Vagrantfile in this directory – the only thing really in this directory – instructs Vagrant to tell Docker to create and run a container from the image alexeiled/docker-oracle-xe-11g. Note how the port mappings are specified – mapping ports 8080 (the HTTP port in the XE database) to the port 8080 on the dockerhost and likewise with the database SQL*Net port of 1521. Noe that you can easily change these mappings, for example if you want to run a second container from this image.



```
ENV['VAGRANT_DEFAULT_PROVIDER'] = 'docker'

## based on http://barrybrierley.blogspot.nl/2015/06/installing-apex-on-linux-virtual-box.html and more

# after starting access APEX:
# http://dockerhost:8080/apex
#  workspace: INTERNAL user: ADMIN password: oracle

Vagrant.configure("2") do |config|

  config.vm.synced_folder "c:/docker-sig/share-with-container", "/host_share"
  config.vm.synced_folder "c:/docker-sig/software/xe", "/software"

  config.vm.define "oraclexe-container" do |m|
    m.vm.provider :docker do |d|
      d.image = "alexeiled/docker-oracle-xe-11g"
      d.name = 'oraclexe-container'
      d.ports =[ "8080:8080", "1521:1521", "9022:22"]
      d.vagrant_machine = "dockerhost"
      d.vagrant_vagrantfile = "../DockerHost/DockerHostVagrantfile"
      d.remains_running = true
    end
  end

end
```

To build and run the container, type:

```
vagrant up
```



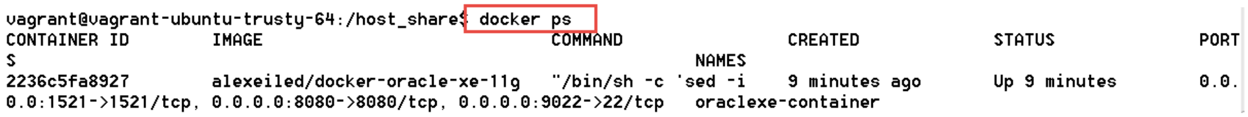
```
c:\docker-sig\xe>vagrant up
Bringing machine 'oraclexe-container' up with 'docker' provider...
==> oraclexe-container: Docker host is required. One will be created if necessary...
    oraclexe-container: Docker host VM is already ready.
==> oraclexe-container: Syncing folders to the host VM...
    oraclexe-container: Mounting shared folders...
    oraclexe-container: /var/lib/docker/docker_1443874988_35561 => C:/docker-sig/share-with-container
    oraclexe-container: /var/lib/docker/docker_1443874988_23275 => C:/docker-sig/software/xe
    oraclexe-container: /var/lib/docker/docker_1443874988_15125 => C:/docker-sig/xe
==> oraclexe-container: Warning: When using a remote Docker host, forwarded ports will NOT be
==> oraclexe-container: immediately available on your machine. They will still be forwarded on
==> oraclexe-container: the remote machine, however, so if you have a way to access the remote
==> oraclexe-container: machine, then you should be able to access those ports there. This is
==> oraclexe-container: not an error, it is only an informational message.
==> oraclexe-container: Creating the container...
    oraclexe-container:   Name: oraclexe-container
    oraclexe-container:  Image: alexeiled/docker-oracle-xe-11g
    oraclexe-container: Volume: /var/lib/docker/docker_1443874988_35561:/host_share
    oraclexe-container: Volume: /var/lib/docker/docker_1443874988_23275:/software
    oraclexe-container: Volume: /var/lib/docker/docker_1443874988_15125:/vagrant
    oraclexe-container:   Port: 8080:8080
    oraclexe-container:   Port: 1521:1521
    oraclexe-container:   Port: 9022:22
    oraclexe-container:
    oraclexe-container: Container created: 2236c5fa892758a4
==> oraclexe-container: Starting container...
==> oraclexe-container: Provisioners will not be run since container doesn't support SSH.
```

The first time you build a container from the image alexeiled/docker-oracle-xe-11g, it will take a long time: this image is 2.5 GB in size. Once this image is loaded into the dockerhost VM, running containers from that image will be lightning fast.

In the shell inside the dockerhost, just like previously, use

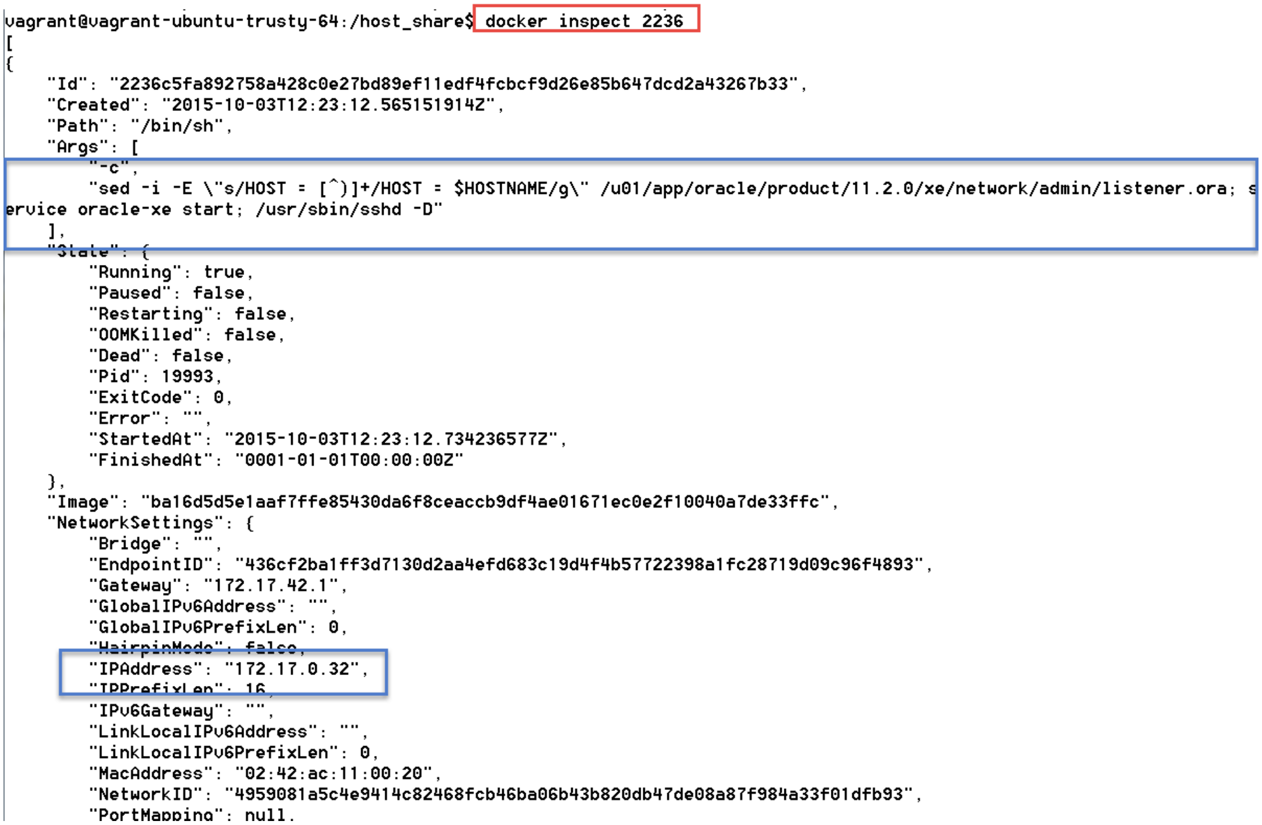
```
docker ps
```

to learn the container id for the oraclexe-container.



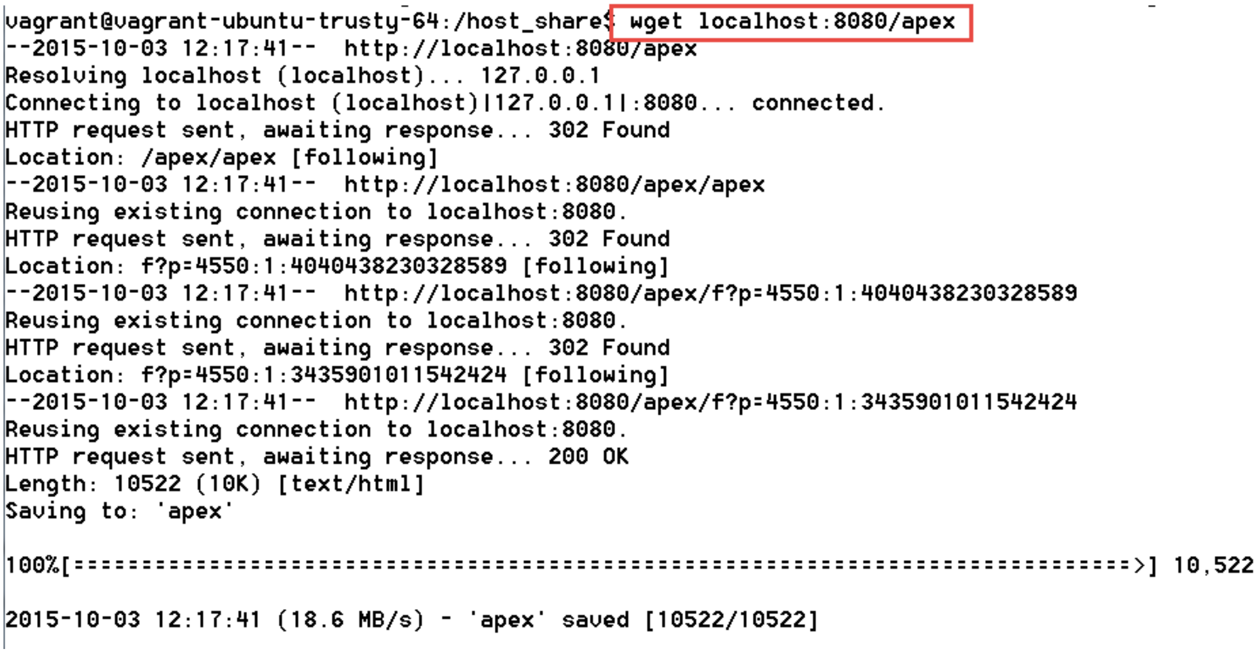
```
vagrant@vagrant-ubuntu-trusty-64:/host_share$ docker ps
CONTAINER ID        IMAGE                               COMMAND                  CREATED             STATUS              PORT
S                                                                      NAMES
2236c5fa8927        alexeiled/docker-oracle-xe-11g      "/bin/sh -c 'sed -i     9 minutes ago       Up 9 minutes        0.0.
0.0:1521->1521/tcp, 0.0.0.0:8080->8080/tcp, 0.0.0.0:9022->22/tcp   oraclexe-container
```

Next, use docker inspect to learn some more details for the container, such as the default CMD used to start it or the IP address:



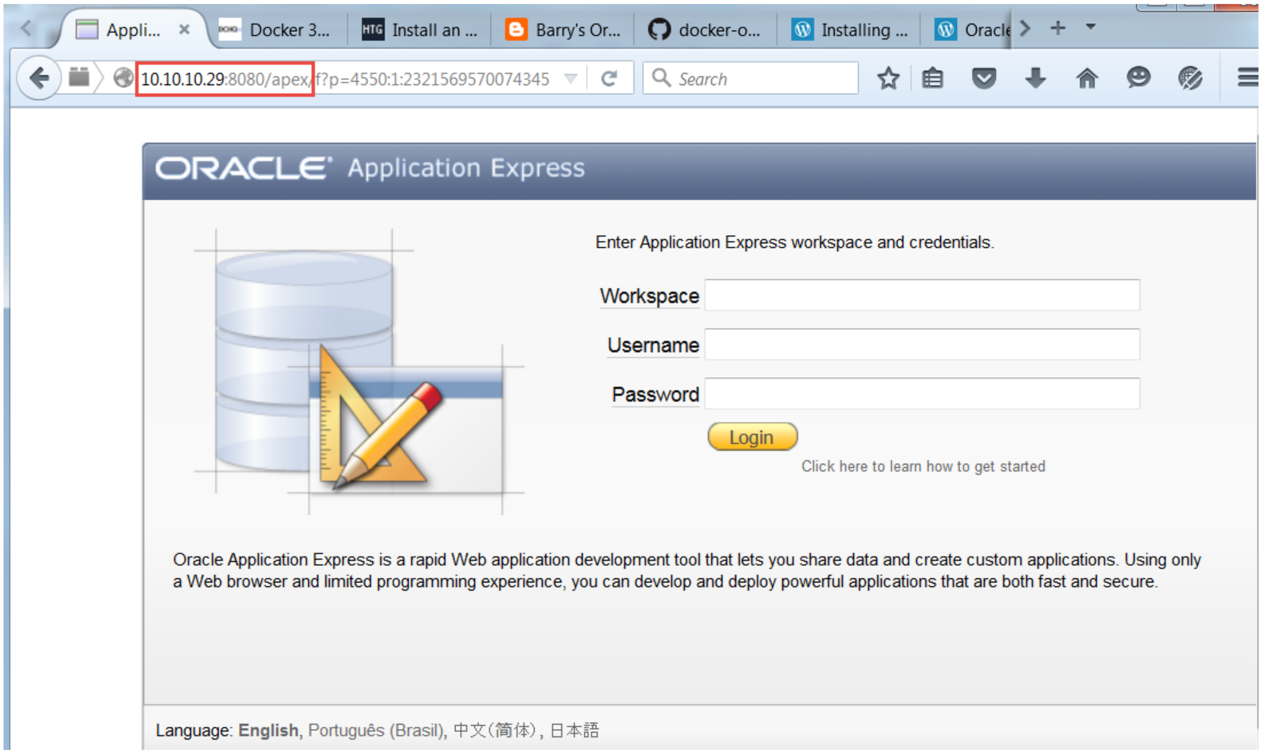
```
vagrant@vagrant-ubuntu-trusty-64:/host_share$ docker inspect 2236
[
{
    "Id": "2236c5fa892758a428c0e27bd89ef11edf4fcbcf9d26e85b647dcd2a43267b33",
    "Created": "2015-10-03T12:23:12.565151914Z",
    "Path": "/bin/sh",
    "Args": [
        "-c",
        "sed -i -E \"s/HOST = [^)]+/HOST = $HOSTNAME/g\" /u01/app/oracle/product/11.2.0/xe/network/admin/listener.ora; s
ervice oracle-xe start; /usr/sbin/sshd -D"
    ],
    "State": {
        "Running": true,
        "Paused": false,
        "Restarting": false,
        "OOMKilled": false,
        "Dead": false,
        "Pid": 19993,
        "ExitCode": 0,
        "Error": "",
        "StartedAt": "2015-10-03T12:23:12.734236577Z",
        "FinishedAt": "0001-01-01T00:00:00Z"
    },
    "Image": "ba16d5d5e1aaf7ffe85430da6f8ceaccb9df4ae01671ec0e2f10040a7de33ffc",
    "NetworkSettings": {
        "Bridge": "",
        "EndpointID": "436cf2ba1ff3d7130d2aa4efd683c19d4f4b57722398a1fc28719d09c96f4893",
        "Gateway": "172.17.42.1",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "HairpinMode": false,
        "IPAddress": "172.17.0.32",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "LinkLocalIPv6Address": "",
        "LinkLocalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:11:00:20",
        "NetworkID": "4959081a5c4e9414c82468fcb46ba06b43b820db47de08a87f984a33f01dfb93",
        "PortMapping": null,
```

On the dockerhost, we can ping to the container, and access its HTTP port. Type wget localhost:8080/apex to see if you can reach the APEX engine inside the container:



```
vagrant@vagrant-ubuntu-trusty-64:/host_share$ wget localhost:8080/apex
--2015-10-03 12:17:41--  http://localhost:8080/apex
Resolving localhost (localhost)... 127.0.0.1
Connecting to localhost (localhost)|127.0.0.1|:8080... connected.
HTTP request sent, awaiting response... 302 Found
Location: /apex/apex [following]
--2015-10-03 12:17:41--  http://localhost:8080/apex/apex
Reusing existing connection to localhost:8080.
HTTP request sent, awaiting response... 302 Found
Location: f?p=4550:1:4040438230328589 [following]
--2015-10-03 12:17:41--  http://localhost:8080/apex/f?p=4550:1:4040438230328589
Reusing existing connection to localhost:8080.
HTTP request sent, awaiting response... 302 Found
Location: f?p=4550:1:3435901011542424 [following]
--2015-10-03 12:17:41--  http://localhost:8080/apex/f?p=4550:1:3435901011542424
Reusing existing connection to localhost:8080.
HTTP request sent, awaiting response... 200 OK
Length: 10522 (10K) [text/html]
Saving to: 'apex'

100%[=============================================================================>] 10,522

2015-10-03 12:17:41 (18.6 MB/s) - 'apex' saved [10522/10522]
```

This is also available in a web browser on the Windows host, using the IP address of the dockerhost VM (10.10.10.29):



10.10.10.29:8080/apex/f?p=4550:1:2321569570074345

Enter Application Express workspace and credentials.

Workspace

Username

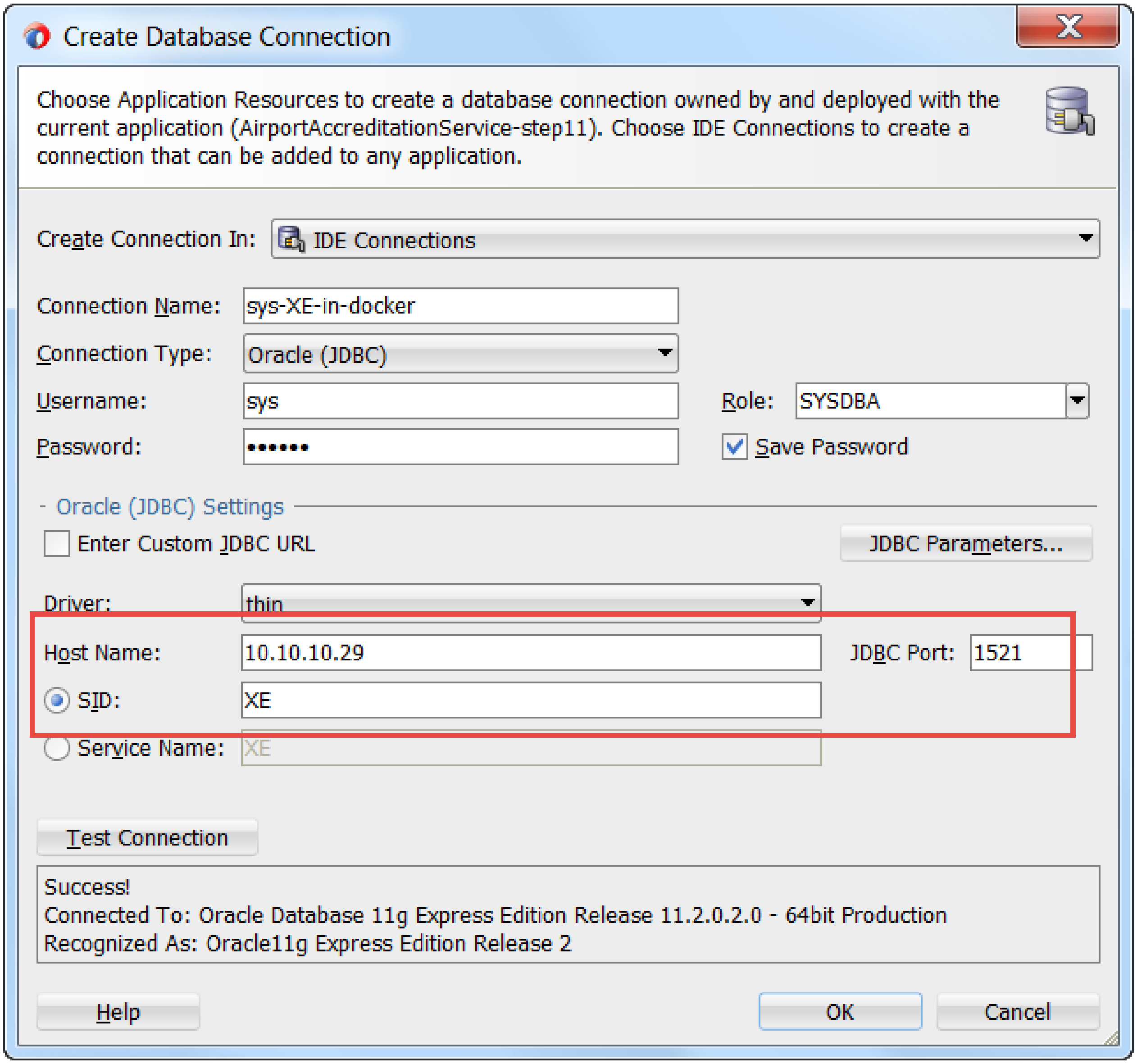
Password

Oracle Application Express is a rapid Web application development tool that lets you share data and create custom applications. Using only a Web browser and limited programming experience, you can develop and deploy powerful applications that are both fast and secure.

Language: **English**, Português (Brasil), 中文(简体), 日本語

Use the settings:
Workspace: INTERNAL
Username: ADMIN
Password: oracle

From the Windows host (or any other node) we can create a database connection to the XE database. In JDeveloper, this looks as follows:



Create Database Connection

Choose Application Resources to create a database connection owned by and deployed with the current application (AirportAccreditationService-step11). Choose IDE Connections to create a connection that can be added to any application.

Create Connection In: IDE Connections

Connection Name: sys-XE-in-docker

Connection Type: Oracle (JDBC)

Username: sys

Role: SYSDBA

Password: ••••••

☑ Save Password

Oracle (JDBC) Settings

□ Enter Custom JDBC URL

Driver: thin

Host Name: 10.10.10.29

JDBC Port: 1521

● SID: XE

○ Service Name: XE

Success!
Connected To: Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
Recognized As: Oracle11g Express Edition Release 2

Note how again the dockerhost ip address is used, along with the port number 1521 on the dockerhost (that forwards to port 1521 in the container). The password for SYS (and SYSTEM) is set to *oracle* in this container.

7. Building a Docker Container with GUI Applications

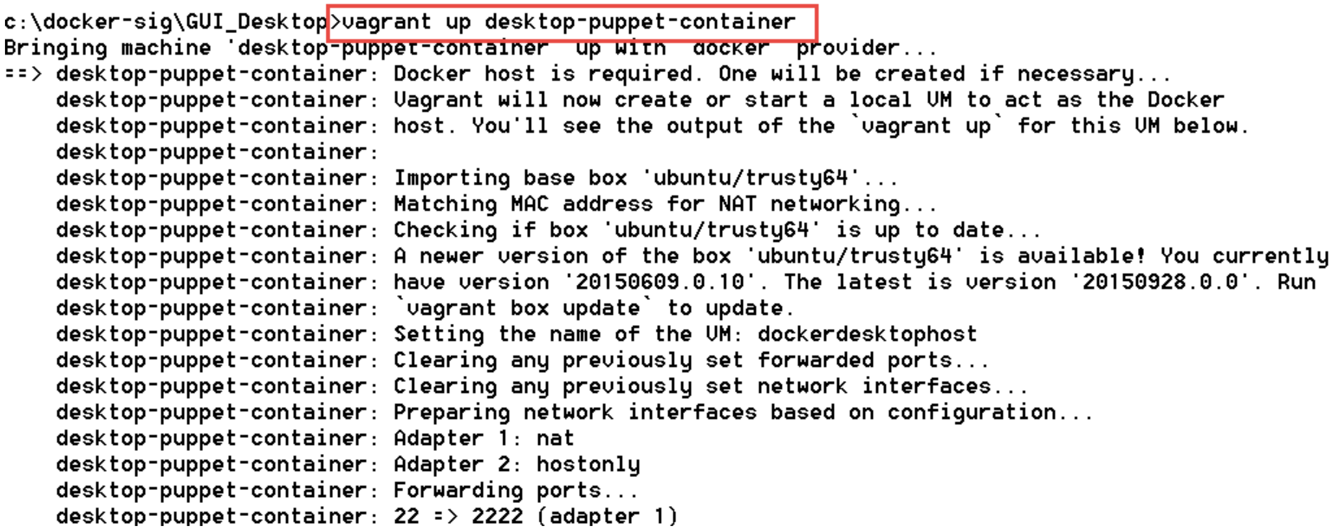
Although perhaps not a common occurrence, I would still like to be able to sometimes run GUI applications inside a Docker container. This may be required because an installer does not provide a silent option and has a GUI wizard I need to click my way through in order to set up the software that the container subsequently will run. It can also be because my development environment management is fully containerized, including management of IDEs and other graphical development tools. Whatever the motivation for running GUI in containers – it is the topic of this section. More specifically: running GUI applications in Docker containers that are managed by Vagrant

see: https://technology.amis.nl/2015/08/29/vagrant-docker-virtualbox-and-the-graphical-desktop-for-gui-applications-in-docker-containers/

The intermediate container is created from the directory C:\docker-sig\GUI_Desktop. Type:

```
vagrant up desktop-puppet-container
```

This starts the creation of the dockerdesktophost – a new VirtualBox VM with desktop support. You will see Virtual Box starting up and doing things. You can ignore this – Vagrant is in the driving seat.



```
c:\docker-sig\GUI_Desktop>vagrant up desktop-puppet-container
Bringing machine 'desktop-puppet-container' up with 'docker' provider...
==> desktop-puppet-container: Docker host is required. One will be created if necessary...
    desktop-puppet-container: Vagrant will now create or start a local VM to act as the Docker
    desktop-puppet-container: host. You'll see the output of the `vagrant up` for this VM below.
    desktop-puppet-container:
    desktop-puppet-container: Importing base box 'ubuntu/trusty64'...
    desktop-puppet-container: Matching MAC address for NAT networking...
    desktop-puppet-container: Checking if box 'ubuntu/trusty64' is up to date...
    desktop-puppet-container: A newer version of the box 'ubuntu/trusty64' is available! You currently
    desktop-puppet-container: have version '20150609.0.10'. The latest is version '20150928.0.0'. Run
    desktop-puppet-container: `vagrant box update` to update.
    desktop-puppet-container: Setting the name of the VM: dockerdesktophost
    desktop-puppet-container: Clearing any previously set forwarded ports...
    desktop-puppet-container: Clearing any previously set network interfaces...
    desktop-puppet-container: Preparing network interfaces based on configuration...
    desktop-puppet-container: Adapter 1: nat
    desktop-puppet-container: Adapter 2: hostonly
    desktop-puppet-container: Forwarding ports...
    desktop-puppet-container: 22 => 2222 (adapter 1)
```

The creation of the VM takes quite a while.

When Vagrant is done, the next step is to get into the new VM – dockerdesktophost – through SSH. To that end, use

```
vagrant global-status
```

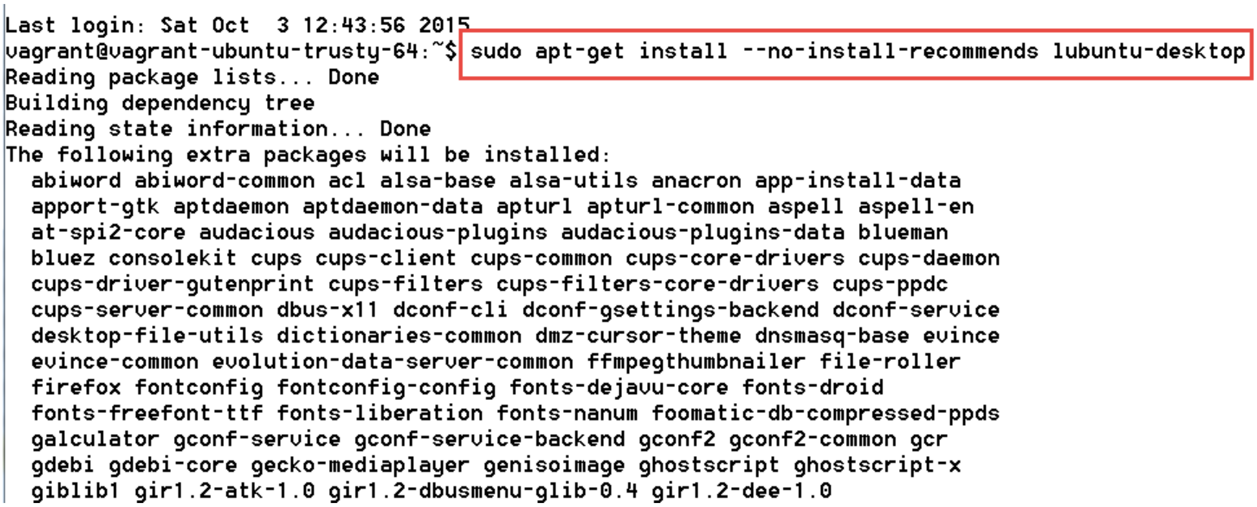
to get a listing of all Vagrant controlled VMs and Containers. Find the ID for the dockerdesktophost.

Next, type

```
vagrant ssh <ID for dockerdesktop>
```

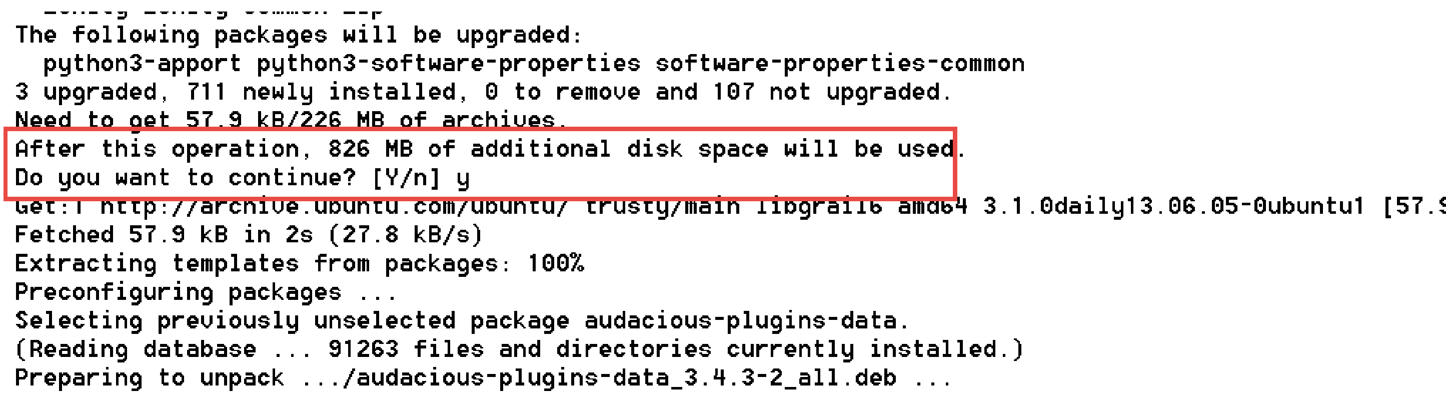
Inside the secure shell, type the following command to add a dektop (Lubuntu in this case) to the Linux environment.

```
sudo apt-get install --no-install-recommends lubuntu-desktop
```



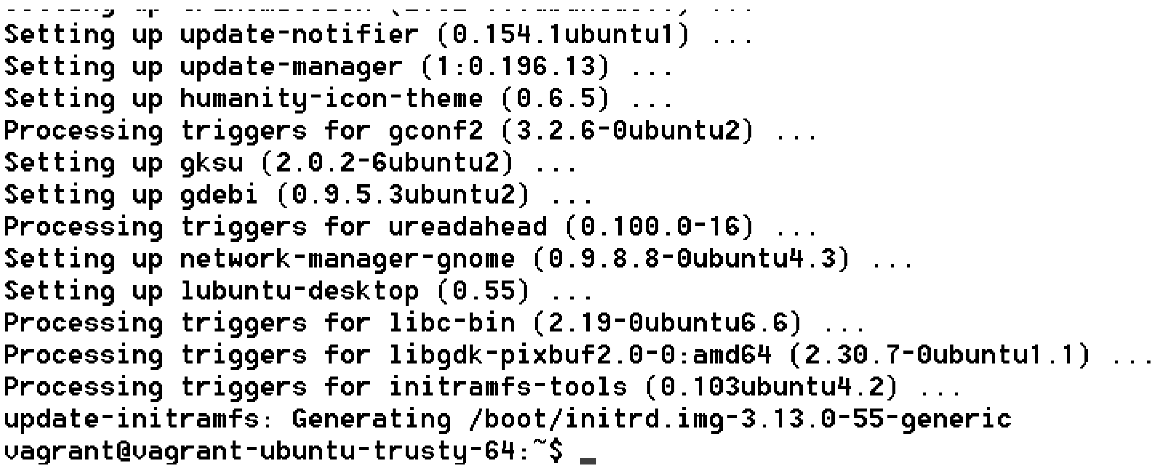
```
Last login: Sat Oct  3 12:43:56 2015
vagrant@vagrant-ubuntu-trusty-64:~$ sudo apt-get install --no-install-recommends lubuntu-desktop
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  abiword abiword-common acl alsa-base alsa-utils anacron app-install-data
  apport-gtk aptdaemon aptdaemon-data apturl apturl-common aspell aspell-en
  at-spi2-core audacious audacious-plugins audacious-plugins-data blueman
  bluez consolekit cups cups-client cups-common cups-core-drivers cups-daemon
  cups-driver-gutenprint cups-filters cups-filters-core-drivers cups-ppdc
  cups-server-common dbus-x11 dconf-cli dconf-gsettings-backend dconf-service
  desktop-file-utils dictionaries-common dmz-cursor-theme dnsmasq-base evince
  evince-common evolution-data-server-common ffmpegthumbnailer file-roller
  firefox fontconfig fontconfig-config fonts-dejavu-core fonts-droid
  fonts-freefont-ttf fonts-liberation fonts-nanum foomatic-db-compressed-ppds
  galculator gconf-service gconf-service-backend gconf2 gconf2-common gcr
  gdebi gdebi-core gecko-mediaplayer genisoimage ghostscript ghostscript-x
  giblib1 gir1.2-atk-1.0 gir1.2-dbusmenu-glib-0.4 gir1.2-dee-1.0
```

You have to confirm that this is indeed the installation you want to have happen. The installation subsequently takes place. This can take quite some time (a desktop is several 100MBs worth of file download and installation).



```
The following packages will be upgraded:
  python3-apport python3-software-properties software-properties-common
3 upgraded, 711 newly installed, 0 to remove and 107 not upgraded.
Need to get 57.9 kB/226 MB of archives.
After this operation, 826 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu/ trusty/main libgrail6 amd64 3.1.0daily13.06.05-0ubuntu1 [57.9
Fetched 57.9 kB in 2s (27.8 kB/s)
Extracting templates from packages: 100%
Preconfiguring packages ...
Selecting previously unselected package audacious-plugins-data.
(Reading database ... 91263 files and directories currently installed.)
Preparing to unpack .../audacious-plugins-data_3.4.3-2_all.deb ...
```

After this process completes,



```
Setting up update-notifier (0.154.1ubuntu1) ...
Setting up update-manager (1:0.196.13) ...
Setting up humanity-icon-theme (0.6.5) ...
Processing triggers for gconf2 (3.2.6-0ubuntu2) ...
Setting up gksu (2.0.2-6ubuntu2) ...
Setting up gdebi (0.9.5.3ubuntu2) ...
Processing triggers for ureadahead (0.100.0-16) ...
Setting up network-manager-gnome (0.9.8.8-0ubuntu4.3) ...
Setting up lubuntu-desktop (0.55) ...
Processing triggers for libc-bin (2.19-0ubuntu6.6) ...
Processing triggers for libgdk-pixbuf2.0-0:amd64 (2.30.7-0ubuntu1.1) ...
Processing triggers for initramfs-tools (0.103ubuntu4.2) ...
update-initramfs: Generating /boot/initrd.img-3.13.0-55-generic
vagrant@vagrant-ubuntu-trusty-64:~$ _
```

we focus again on the Docker container. Use

```
docker ps
```

to find the container identifier for the desktop-puppet-container. Next, use

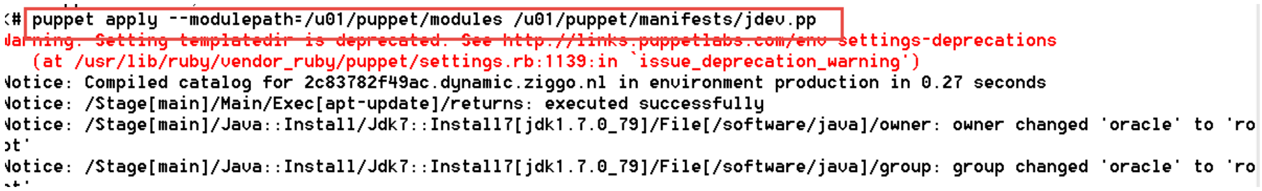
```
docker exec -it <container id> bash
```

to open a shell into this container.

With the following command Puppet is activated to perform the configuration of the container, based on the jdev.pp manifest to install the JDK and JDeveloper 12.1.3 (SOA Suite edition):

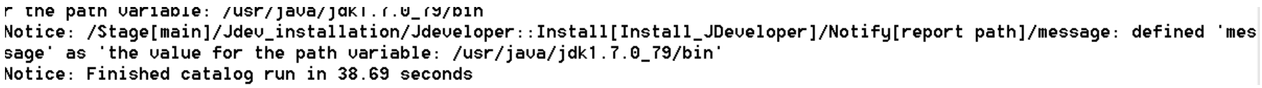
```
puppet apply --modulepath=/u01/puppet/modules
/u01/puppet/manifests/jdev.pp
```

to install JDeveloper into the container.



```
<# puppet apply --modulepath=/u01/puppet/modules /u01/puppet/manifests/jdev.pp
Warning: Setting templatedir is deprecated. See http://links.puppetlabs.com/env-settings-deprecations
   (at /usr/lib/ruby/vendor_ruby/puppet/settings.rb:1139:in `issue_deprecation_warning')
Notice: Compiled catalog for 2c83782f49ac.dynamic.ziggo.nl in environment production in 0.27 seconds
Notice: /Stage[main]/Main/Exec[apt-update]/returns: executed successfully
Notice: /Stage[main]/Java::Install/Jdk7::Install7[jdk1.7.0_79]/File[/software/java]/owner: owner changed 'oracle' to 'ro
ot'
Notice: /Stage[main]/Java::Install/Jdk7::Install7[jdk1.7.0_79]/File[/software/java]/group: group changed 'oracle' to 'ro
```

Wait, wait, and wait some more…



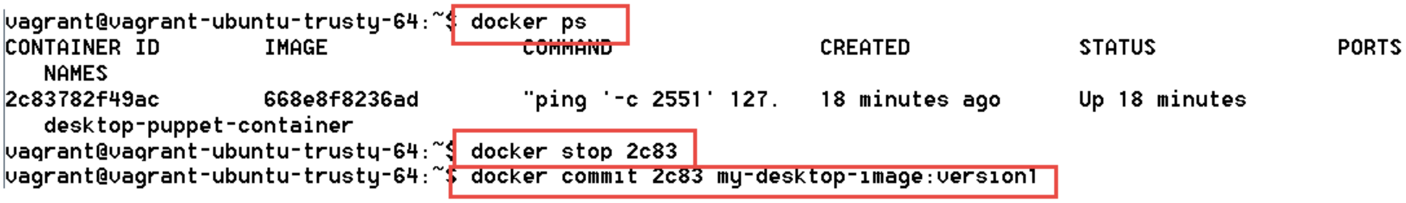
```
r the path variable: /usr/java/jdk1.7.0_79/bin
Notice: /Stage[main]/Jdev_installation/Jdeveloper::Install[Install_JDeveloper]/Notify[report path]/message: defined 'mes
sage' as 'the value for the path variable: /usr/java/jdk1.7.0_79/bin'
Notice: Finished catalog run in 38.69 seconds
```

Then Puppet completes, exit the container – by typing

```
exit
```

Then user docker commit to turn the container into a tagged image:

```
docker commit <container id> my-desktop-image:version1
```



```
vagrant@vagrant-ubuntu-trusty-64:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
   NAMES
2c83782f49ac        668e8f8236ad        "ping '-c 2551' 127.   18 minutes ago      Up 18 minutes
   desktop-puppet-container
vagrant@vagrant-ubuntu-trusty-64:~$ docker stop 2c83
vagrant@vagrant-ubuntu-trusty-64:~$ docker commit 2c83 my-desktop-image:version1
```

List all images with

```
docker images
```

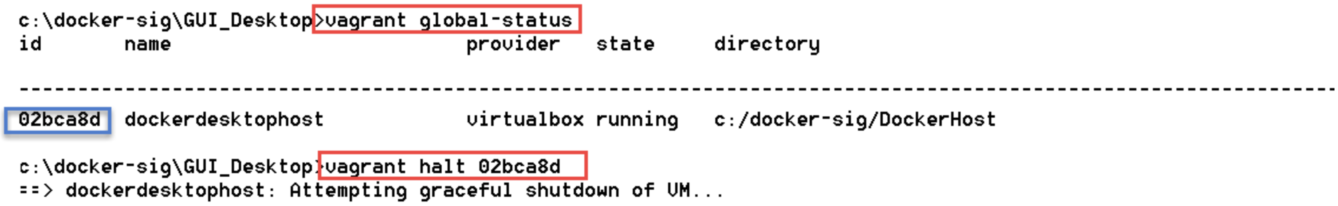
and find the new image in the list.

In order to run a container *and* actually see the graphical display, we need to use the graphical view into the VM in Virtual Box. Therefore, we need to exit the secure shell into the *dockerdesktop*, by typing

```
exit
```

and we need to halt both the container and the dockerdesktophost VM, through Vagrant:

```
vagrant halt desktop-puppet-container
```

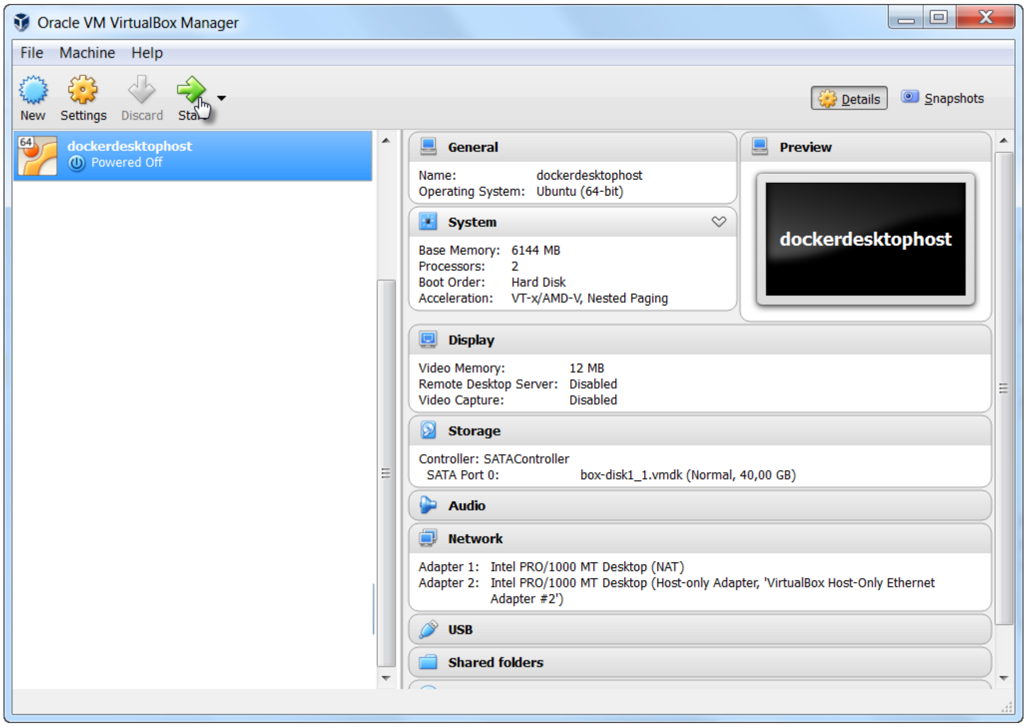


```
c:\docker-sig\GUI_Desktop>vagrant global-status
id       name                          provider   state     directory
-------------------------------------------------------------------------------------------------------------------
02bca8d  dockerdesktophost             virtualbox running   c:/docker-sig/DockerHost

c:\docker-sig\GUI_Desktop>vagrant halt 02bca8d
==> dockerdesktophost: Attempting graceful shutdown of VM...
```

```
vagrant halt <VM id for dockerdesktophost>
```

In order to work with JDeveloper – a GUI application – we need to run the container from within the DockerDesktopHost that is capable of sharing its X-Windowing with the container. The command we need to use for running the container is somewhat complex. It has to forward the display system, run the JDevelopet startup script etc.

Start Docker Host VM through VirtualBox GUI – this will launch the GUI (the desktop).



General

Name: dockerdesktophost
Operating System: Ubuntu (64-bit)

System

Base Memory: 6144 MB
Processors: 2
Boot Order: Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging

Display

Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage

Controller: SATAController
SATA Port 0: box-disk1_1.vmdk (Normal, 40,00 GB)

Audio

Network

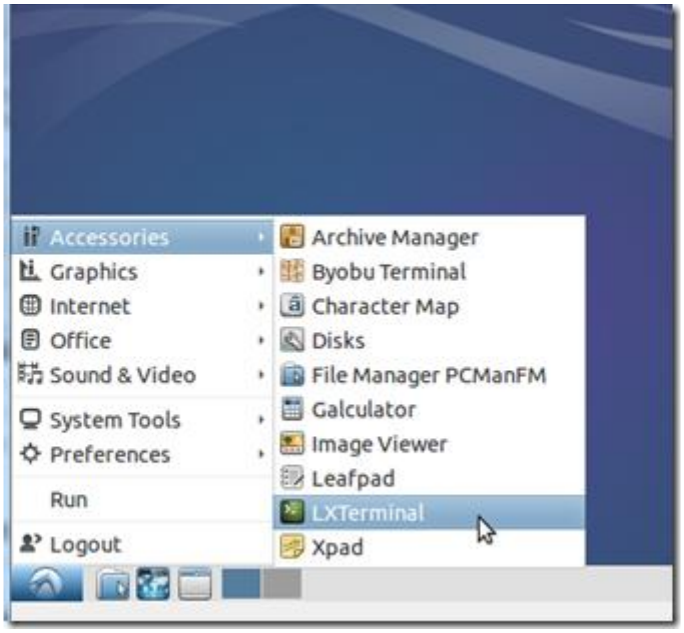
Adapter 1: Intel PRO/1000 MT Desktop (NAT)
Adapter 2: Intel PRO/1000 MT Desktop (Host-only Adapter, 'VirtualBox Host-Only Ethernet Adapter #2')

USB

Shared folders

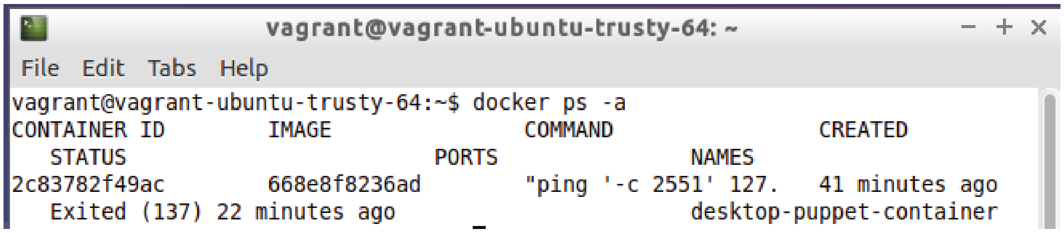
Login – vagrant/vagrant:

And open a terminal:



You can check whether Docker is available and what the situation is for containers:

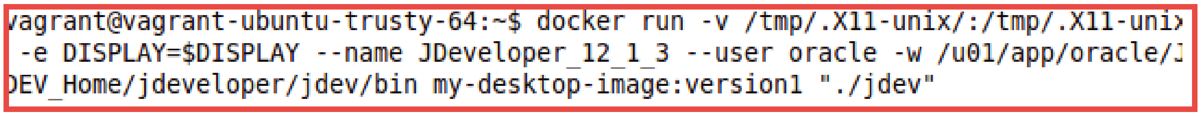
```
docker ps -a
```



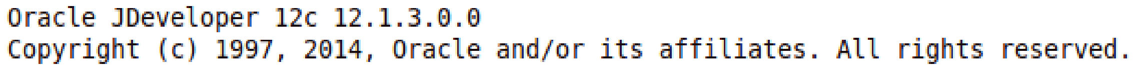
```
vagrant@vagrant-ubuntu-trusty-64:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                CREATED
   STATUS                      PORTS               NAMES
2c83782f49ac        668e8f8236ad        "ping '-c 2551' 127.   41 minutes ago
   Exited (137) 22 minutes ago                     desktop-puppet-container
```

Launch a Docker container from the terminal window for the new Docker image using special options: -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix – and run the GUI application inside Docker Container:

```
docker run -v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=$DISPLAY
--name JDeveloper_12_1_3 --user oracle
-w /u01/app/oracle/JDEV_Home/jdeveloper/jdev/bin
my-desktop-image:version1 “./jdev”
```

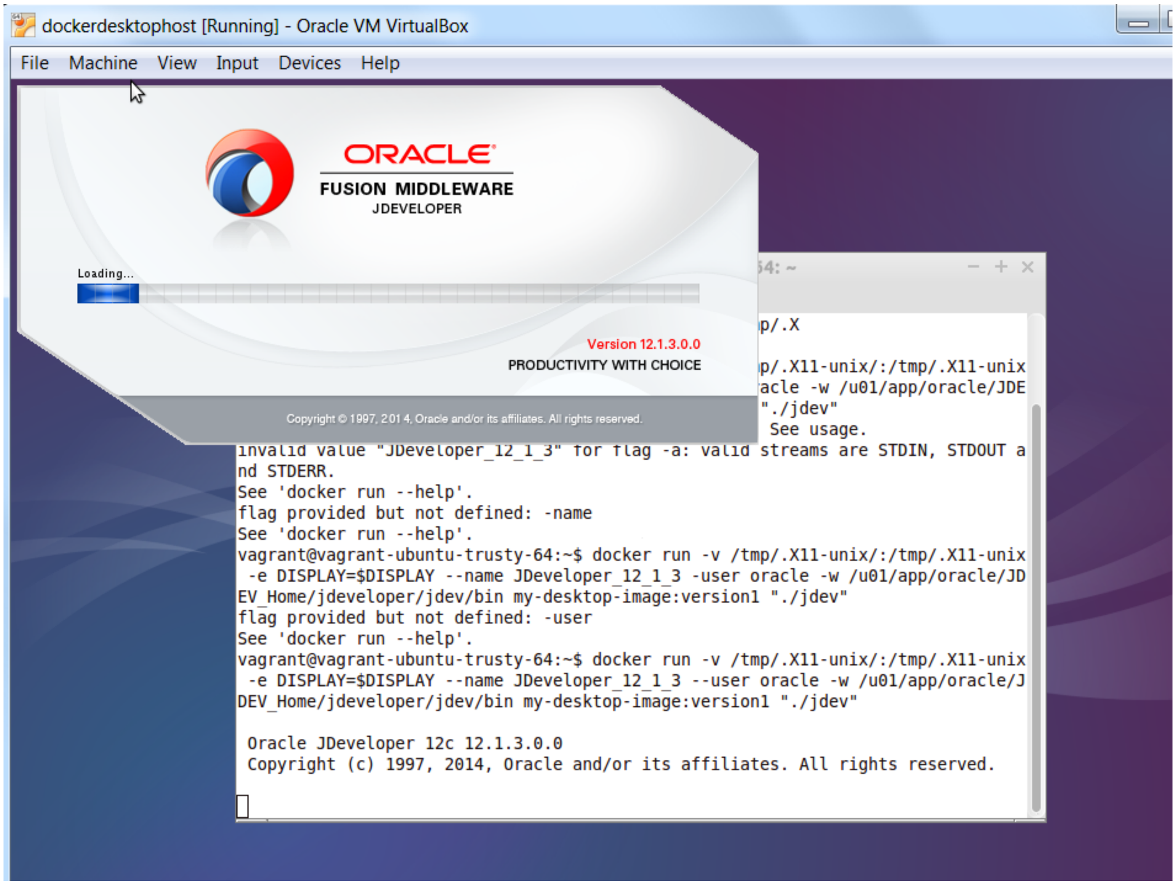


```
vagrant@vagrant-ubuntu-trusty-64:~$ docker run -v /tmp/.X11-unix/:/tmp/.X11-unix
 -e DISPLAY=$DISPLAY --name JDeveloper_12_1_3 --user oracle -w /u01/app/oracle/J
DEV_Home/jdeveloper/jdev/bin my-desktop-image:version1 "./jdev"
```



```
 Oracle JDeveloper 12c 12.1.3.0.0
 Copyright (c) 1997, 2014, Oracle and/or its affiliates. All rights reserved.
```

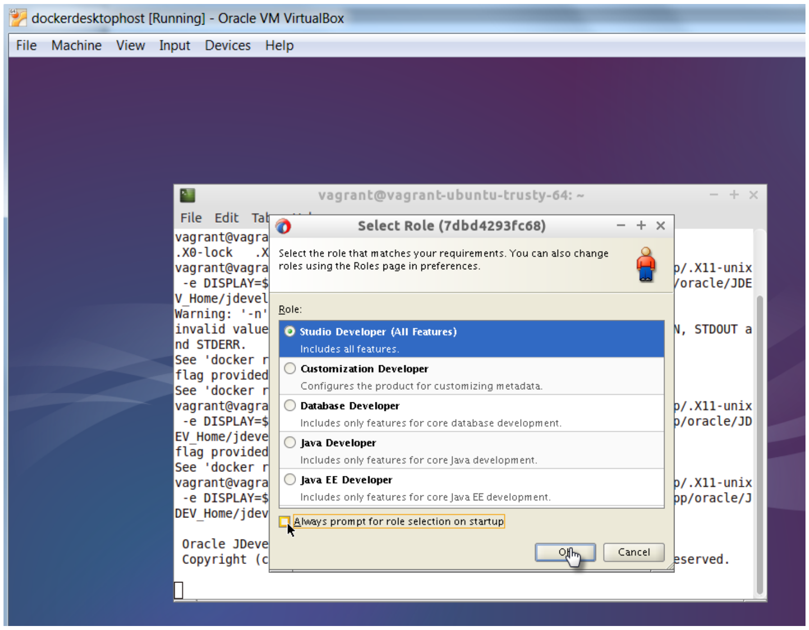
This figure shows how the JDeveloper GUI is launched inside the dockerdesktophostvm when the Docker container is run using this statement:


dockerdesktophost [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
ORACLE
FUSION MIDDLEWARE
JDEVELOPER
Loading...
Version 12.1.3.0.0
PRODUCTIVITY WITH CHOICE
Copyright © 1997, 2014, Oracle and/or its affiliates. All rights reserved.
54: ~

```
p/.X

p/.X11-unix/:/tmp/.X11-unix
acle -w /u01/app/oracle/JDE
"./jdev"
 See usage.
invalid value "JDeveloper_12_1_3" for flag -a: valid streams are STDIN, STDOUT a
nd STDERR.
See 'docker run --help'.
flag provided but not defined: -name
See 'docker run --help'.
vagrant@vagrant-ubuntu-trusty-64:~$ docker run -v /tmp/.X11-unix/:/tmp/.X11-unix
 -e DISPLAY=$DISPLAY --name JDeveloper_12_1_3 -user oracle -w /u01/app/oracle/JD
EV_Home/jdeveloper/jdev/bin my-desktop-image:version1 "./jdev"
flag provided but not defined: -user
See 'docker run --help'.
vagrant@vagrant-ubuntu-trusty-64:~$ docker run -v /tmp/.X11-unix/:/tmp/.X11-unix
 -e DISPLAY=$DISPLAY --name JDeveloper_12_1_3 --user oracle -w /u01/app/oracle/J
DEV_Home/jdeveloper/jdev/bin my-desktop-image:version1 "./jdev"

 Oracle JDeveloper 12c 12.1.3.0.0
 Copyright (c) 1997, 2014, Oracle and/or its affiliates. All rights reserved.

```

See details in https://technology.amis.nl/2015/08/30/generate-docker-containerized-run-time-and-design-time-for-oracle-streamexplorer-event-processor-and-jdeveloper-using-vagrant-puppet-and-virtualbox/

Accept the Studio Developer Role and press OK.



dockerdesktophost [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

vagrant@vagrant-ubuntu-trusty-64: ~

```
vagrant@vagra
.X0-lock   .X
vagrant@vagra                                  p/.X11-unix
 -e DISPLAY=$                                  /oracle/JDE
V_Home/jdevel
Warning: '-n'
invalid value                                  N, STDOUT a
nd STDERR.
See 'docker r
flag provided
See 'docker r
vagrant@vagra                                  p/.X11-unix
 -e DISPLAY=$                                  p/oracle/JD
EV_Home/jdeve
flag provided
See 'docker r
vagrant@vagra                                  p/.X11-unix
 -e DISPLAY=$                                  pp/oracle/J
DEV_Home/jdev

 Oracle JDeve
 Copyright (c                                  eserved.
```

Select Role (7dbd4293fc68)

Select the role that matches your requirements. You can also change roles using the Roles page in preferences.

Role:

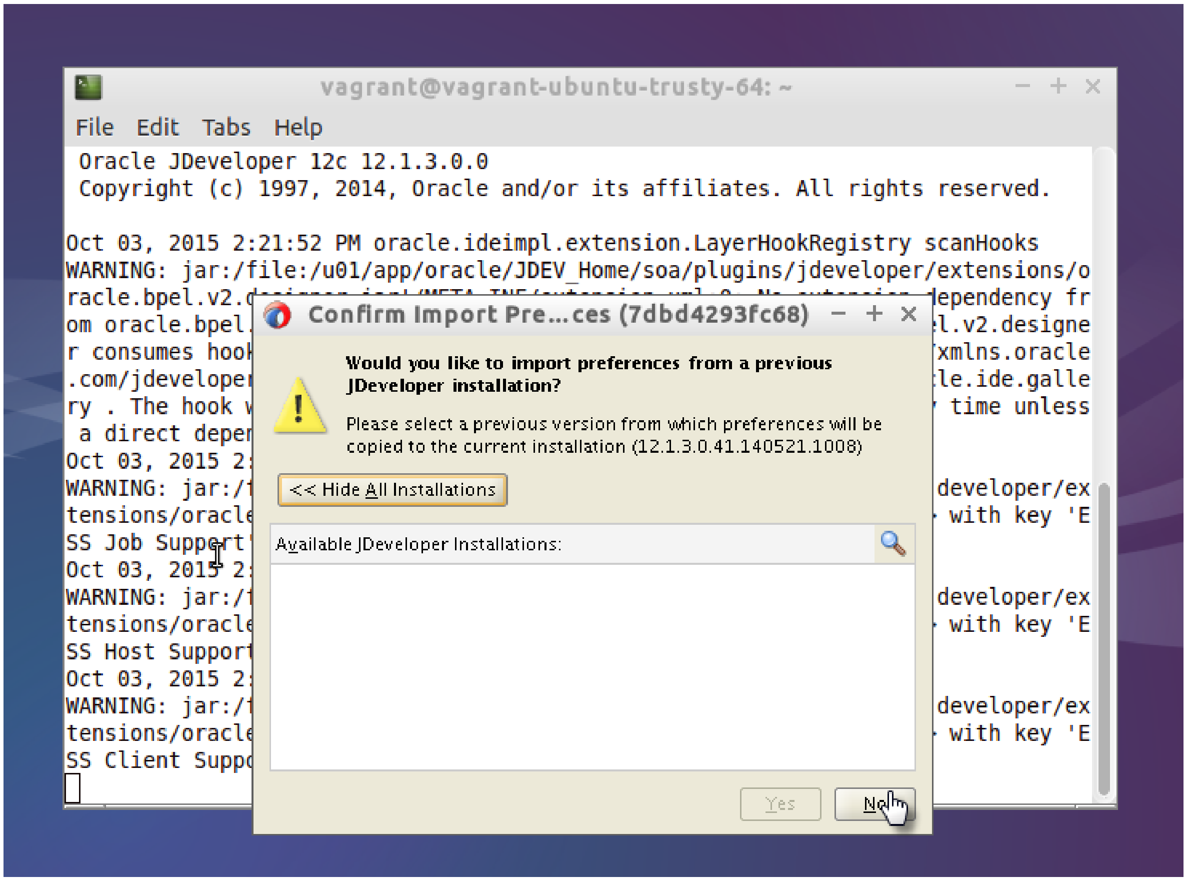
- ● **Studio Developer (All Features)** Includes all features.
- ○ **Customization Developer** Configures the product for customizing metadata.
- ○ **Database Developer** Includes only features for core database development.
- ○ **Java Developer** Includes only features for core Java development.
- ○ **Java EE Developer** Includes only features for core Java EE development.

□ Always prompt for role selection on startup

OK Cancel

JDeveloper continues starting up.

Press No to decline the import of preferences.



vagrant@vagrant-ubuntu-trusty-64: ~

File Edit Tabs Help

```
 Oracle JDeveloper 12c 12.1.3.0.0
 Copyright (c) 1997, 2014, Oracle and/or its affiliates. All rights reserved.

Oct 03, 2015 2:21:52 PM oracle.ideimpl.extension.LayerHookRegistry scanHooks
WARNING: jar:/file:/u01/app/oracle/JDEV_Home/soa/plugins/jdeveloper/extensions/o
racle.bpel.v2.d                                                    dependency fr
om oracle.bpel                                                    l.v2.designe
r consumes hoo                                                    xmlns.oracle
.com/jdevelope                                                    le.ide.galle
ry . The hook w                                                    time unless
 a direct depe
Oct 03, 2015 2
WARNING: jar:/f                                                    developer/ex
tensions/oracle                                                    with key 'E
SS Job Support
Oct 03, 2015 2
WARNING: jar:/f                                                    developer/ex
tensions/oracle                                                    with key 'E
SS Host Support
Oct 03, 2015 2
WARNING: jar:/f                                                    developer/ex
tensions/oracle                                                    with key 'E
SS Client Supp
```

Confirm Import Pre...ces (7dbd4293fc68)

Would you like to import preferences from a previous JDeveloper installation?

Please select a previous version from which preferences will be copied to the current installation (12.1.3.0.41.140521.1008)

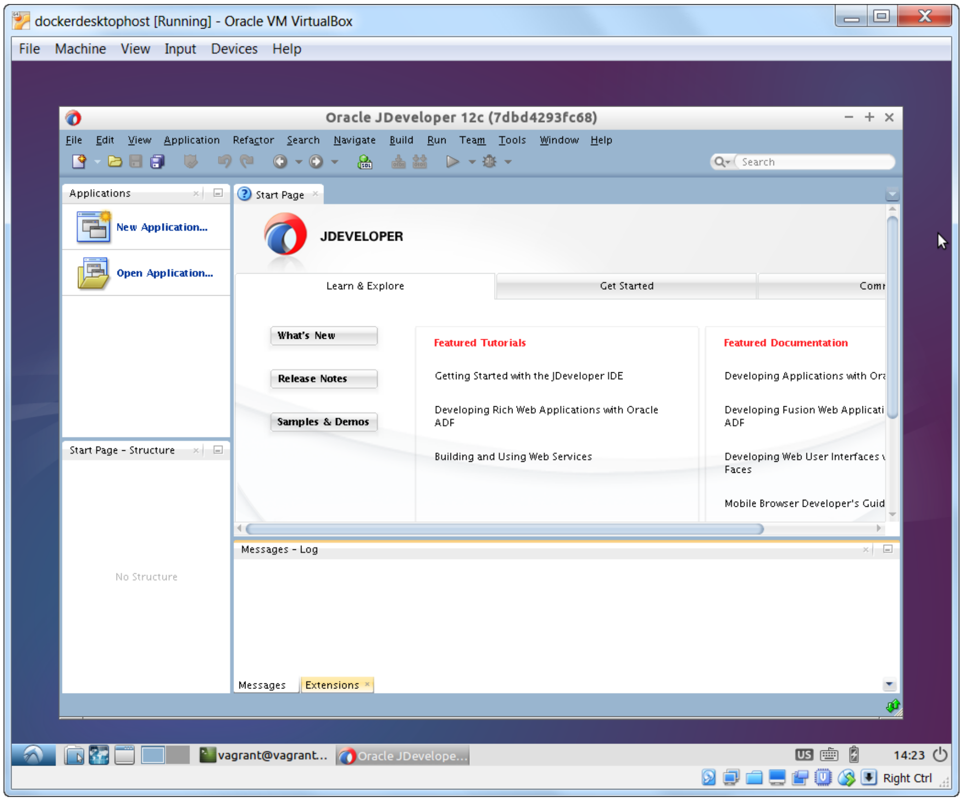
<< Hide All Installations

Available JDeveloper Installations:

Yes No

And the installation continues.

And JDeveloper is open and available to us to start using.



Featured Tutorials

Getting Started with the JDeveloper IDE

Developing Rich Web Applications with Oracle ADF

Building and Using Web Services

Featured Documentation

Developing Applications with Ora

Developing Fusion Web Applicati ADF

Developing Web User Interfaces v Faces

Mobile Browser Developer's Guid