Test case Solution:

Setting up docker on Vagrant hosting

To describe a container we need a Dockerfile:

```
FROM ubuntu:14.04

# Install dev tools: jdk, git etc...

RUN apt-get update

RUN apt-get install -y openjdk-7-jdk git wget

# jdk7 is the default jdk

RUN ln -fs /usr/lib/jvm/java-7-openjdk-amd64/jre/bin/java /etc/alternatives/java

# Install vertx

RUN \

    mkdir -p /usr/local/vertx && cd /usr/local/vertx && \

    wget http://dl.bintray.com/vertx/downloads/vert.x-2.1.2.tar.gz -qO - | tar -xz

# Add vertx to the path

ENV PATH /usr/local/vertx/vert.x-2.1.2/bin:$PATH

RUN mkdir -p /usr/local/src

WORKDIR /usr/local/src

CMD ["bash"]
```

Build the Docker image:

```
$ sudo docker build -t=vertxdev
```

fetch the source code from Github:

```
$ sudo docker run -t --rm -v /src/vertx/:/usr/local/src vertxdev git clone
https://github.com/vert-x/vertx-examples.git
```

==========

Setting up the environment

1. The first step in installing Odoo is to install the EPEL repository that provides a set of extra packages for enterprise Linux. But first, be sure to update the system as shown.

```
$ sudo dnf update
```

2. Once the update of the system is complete, install the EPEL repository as shown.

```
$ sudo dnf install epel-release
```

3. Next, install Python and other requisite dependencies that are required by Odoo as shown.

```
$ sudo dnf install python36 python36-devel git gcc wget nodejs libxslt-devel bzip2-devel openldap-devel libjpeg-devel freetype-devel
```

4. We need to install PostgreSQL for Odoo and to do this, run the command.

```
$ sudo dnf install postgresql-server postgresql-contrib
```

5. Next, initialize a new PostgreSQL database cluster.

```
$ sudo postgresql-setup initdb
```

6. Once the database cluster has been initialized, restart, and enable PostgreSQL as shown.

```
$ sudo systemctl restart postgresql
```

```
$ sudo systemctl enable postgresql
```

7. To confirm that the database is up and running, execute.

```
$ sudo systemctl status postgresql
```

8. For Odoo to print PDF reports, it requires a package called Wkhtmltopdf. This is used to render HTML to PDF and other image formats.

```
$ sudo dnf install
https://github.com/wkhtmltopdf/wkhtmltopdf/releases/download/0.12.5/wkhtmltox-0.12.5-1.centos8.x86_64.rpm
```

9. We will add a new system user that we will use to run the Odoo service. The home directory is located in the /opt/odoo directory.

```
$ sudo useradd -m -U -r -s /bin/bash odoo -d /opt/odoo
```

10. To begin installing Odoo, first switch to the Odoo user that we created above.

```
$ sudo su - odoo
```

11. Then clone the git repository.

$ git clone https://www.github.com/odoo/odoo --depth 1 --branch 13.0 /opt/odoo/odoo13

12. Next, clone the virtual environment as shown.

$ cd /opt/odoo

$ python3 -m venv odoo13-venv

13. Once the virtual environment is created, activate it using the following command.

$ source odoo13-venv/bin/activate

14. Inside the virtual environment, install the required Python modules.

$ pip3 install -r odoo13/requirements.txt

15. Once the installation of the Python modules is complete, exit the virtual environment and go back to the sudo user.

$ deactivate && exit

16. Create a directory for custom modules and later assign the directory ownership to the 'Odoo' user.

$ sudo mkdir /opt/odoo/odoo13-custom-addons

$ sudo chown -R odoo:odoo /opt/odoo/odoo13-custom-addons

17. We will create a custom log directory and log file as shown.

$ sudo mkdir /var/log/odoo13

$ sudo touch /var/log/odoo13/odoo.log

$ sudo chown -R odoo:odoo /var/log/odoo13/

18. create a custom configuration file for Odoo as shown.

$ sudo vim /etc/odoo.conf

Paste the following configuration and save the file.

admin_passwd = strong_password

db_host = False

db_port = False

db_user = odoo

db_password = False

xmlrpc_port = 8069

; longpolling_port = 8072

logfile = /var/log/odoo13/odoo.log

logrotate = True

addons_path = /opt/odoo/odoo13/addons,/opt/odoo/odoo13-custom-addons

Replace the strong_password with your preferred password.

19. Now, create a systemd unit file for Odoo.

$ sudo vim /etc/systemd/system/odoo13.service

Description=Odoo13

#Requires=postgresql-10.6.service

#After=network.target postgresql-10.6.service

[Service]

Type=simple

SyslogIdentifier=odoo13

PermissionsStartOnly=true

User=odoo

Group=odoo

ExecStart=/opt/odoo/odoo13-venv/bin/python3 /opt/odoo/odoo13/odoo-bin -c /etc/odoo.conf

StandardOutput=journal+console

WantedBy=multi-user.target

20. Reload systemd of the new changes made to the file.

$ sudo systemctl daemon-reload

21. Start and enable Odoo as shown.

$ sudo systemctl start odoo13

$ sudo systemctl enable odoo13

22. Confirm the status of Odoo:

```
$ sudo systemctl status odoo13
```

23. Use the netstat command to check if Odoo is listening on port 8069.

```
$ sudo netstat -pnltu | grep 8069
```

24. Open the port across the firewall and make Odoo accessible on a browser.

```
$ sudo firewall-cmd --add-port=8069/tcp --zone=public --permanent
$ sudo firewall-cmd --reload
```

25. Install the Nginx web server which will act as a reverse proxy to our Odoo instance.

```
$ sudo dnf install nginx
```

26. Create a new virtual host file.

```
$ sudo vim /etc/nginx/conf.d/odoo13.conf
```

Paste the following configuration to the new host file.

```
upstream odoo {
  server 127.0.0.1:8069;
}
server {
        listen 80;
        server_name server-IP;
        access_log /var/log/nginx/odoo13.access.log;
        error_log /var/log/nginx/odoo13.error.log;
                location / {
                proxy_set_header X-Forwarded-Host $host;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-Forwarded-Proto $scheme;
                proxy_set_header X-Real-IP $remote_addr;
```

```
        proxy_redirect off;
        proxy_pass http://odoo;
    }
location ~* /web/static/ {
        proxy_cache_valid 200 90m;
        proxy_buffering on;
        expires 864000;
        proxy_pass http://odoo;
    }
    gzip_types text/css text/less text/plain text/xml application/xml application/json application/javascript;
    gzip on;
}
```

Save and exit the configuration file.

27. Now start and enable Nginx webserver.

```
$ sudo systemctl start nginx
$ sudo systemctl enable nginx
```

28. Confirm that Nginx is running as shown.

```
$ sudo systemctl status nginx
```

29. Launch a web browser and visit the server's IP.

http://server-ip/

For the master password, use the password specified in Step 5

30. This ushers you to Odoo's dashboard showing the various applications that can be installed.

Vagrant has some important advantages:

Configure it once and run it everywhere : Vagrant is just a Docker wrapper on systems that support Docker natively while it spins up a « host VM » to run containers on systems that don't support it. User

don’t have to bother whether Docker is supported natively or not : the same configuration will work on every OS.

Docker hosts are not limited to boot2docker (a Virtualbox image of Tiny Core Linux) but Debian, Ubuntu, CoreOS and other Linux distros are supported too. And can run can run on more stable VM managers than Virtualbox (e.g. VMWare).

Vagrant can orchestrate Docker containers: run multiple containers concurrently and link them together

---

Using vagrant to make Docker containers portable

Vagrant supports Docker both as provider and as a provisioner. But to let it automatically spin up a Docker host VM on Windows and Mac it should be used as a provider.

We are going to reuse the same Dockerfile we saw above. And, as above, we will run two Docker containers to execute git clone and vertx run. But Vagrant commands will used instead of Docker’s.

Install Vagrant and Virtualbox to be able to run the samples.

Vagrantfile

Vagrantfiles describe Vagrant boxes. We will use the following one along this section:

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

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

  config.vm.define "vertxdev" do |a|

    a.vm.provider "docker" do |d|

      d.build_dir = "."

      d.build_args = ["-t=vertxdev"]

      d.ports = ["8080:8080"]

      d.name = "vertxdev"

      d.remains_running = true

      d.cmd = ["vertx", "run", "vertx-examples/src/raw/java/httphelloworld/HelloWorldServer.java"]

      d.volumes = ["/src/vertx/:/usr/local/src"]

    end

  end
```

```
end
```

ENV'VAGRANT_DEFAULT_PROVIDER' = 'docker' saves us to specify, at every Vagrant command, that the provider is Docker (default provider is Virtualbox). The rest of the file has options that Vagrant will use to build the Docker image and run a container. For more information refer to Vagrantfilehttps://docs.vagrantup.com/v2/vagra… and Docker provider documentation.

Get the source code

Once we have copied the Vagrantfile in Dockerfile folder we can run git clone to fetch the source code:

```
$ vagrant docker-run vertxdev -- git clone https://github.com/vert-x/vertx-examples.git
```

Like before, the container will be destroyed when git clone ends its execution. Note that we haven't built the image, Vagrant did it automatically. On manual step less.

Build and run the application

We are able to build and run the HTTP Hello World server:

```
$ vagrant up
```

Under the hoods Vagrant will execute docker run and the command to start the container is specified by the d.cmd option.

To get the output of the vertx run command:

```
$ vagrant docker-logs
```

Testing

On a linux platform just run:

```
$ curl localhost:8080
```

Hello World

On Windows and Mac port 8080 is not forwarded from Docker host VM to the main vagrant host (although Docker container port is forwarded to Docker host). As a consequence we need to ssh into Docker host VM to connect to the HTTP server. Let's retrieve the id of Vagrant default Docker host:

```
$ vagrant global-status
id          name        provider    state     directory
---------------------------------------------------------------------------------------------------
c62a174     default     virtualbox running /Users/mariolet/.vagrant.d/data/docker-host
```

Once the box id retrieved we can test the HTTP server:

```
$ vagrant ssh c62a174 -c "curl localhost:8080"
```

Hello World

==========

Docker host VM Vagrantfile

We will use a new Vagrantfile to define the Docker host VM. The following one is based on Ubuntu Server 14.04 LTS:

```
Vagrant.configure("2") do |config|
  config.vm.provision "docker"
  # The following line terminates all ssh connections. Therefore
  # Vagrant will be forced to reconnect.
  # That's a workaround to have the docker command in the PATH
  config.vm.provision "shell", inline:
    "ps aux | grep 'sshd:' | awk '{print $2}' | xargs kill"
  config.vm.define "dockerhost"
  config.vm.box = "ubuntu/trusty64"
  config.vm.network "forwarded_port",
    guest: 8080, host: 8080
  config.vm.provider :virtualbox do |vb|
      vb.name = "dockerhost"
  end
end
```

Run Docker containers in a custom Docker host

Next specify to use this new VM as Docker host instead of the default one adding two new lines to the a.vm.provider block:

```
config.vm.define "vertxdev" do |a|
    a.vm.provider "docker" do |d|
        [...]
        d.vagrant_machine = "dockerhost"
        d.vagrant_vagrantfile = "./DockerHostVagrantfile"
    end
end
```

Note that that configuring a custom docker host has another benefit: we can now specify custom forwarded ports:

```
config.vm.network "forwarded_port",
                guest: 8080, host: 8080
```

Configure two Docker containers within a new Vagrantfile:

```
ENV['VAGRANT_DEFAULT_PROVIDER'] = 'docker'
DOCKER_HOST_NAME = "dockerhost"
DOCKER_HOST_VAGRANTFILE = "./DockerHostVagrantfile"
Vagrant.configure("2") do |config|
    config.vm.define "vertxreceiver" do |a|
        a.vm.provider "docker" do |d|
            d.build_dir = "."
            d.build_args = ["-t=vertxreceiver"]
            d.name = "vertxreceiver"
            d.remains_running = true
            d.cmd = ["vertx", "run",
"vertx-examples/src/raw/java/eventbus_pointtopoint/Receiver.java","-cluster"]
            d.volumes = ["/src/vertx/:/usr/local/src"]
            d.vagrant_machine = "#{DOCKER_HOST_NAME}"
```

```
        d.vagrant_vagrantfile = "#{DOCKER_HOST_VAGRANTFILE}"
      end
  end
  config.vm.define "vertxsender" do |a|
      a.vm.provider "docker" do |d|
        d.build_dir = "."
        d.build_args = ["-t=vertxsender"]
        d.name = "vertxsender"
        d.remains_running = true
        d.cmd = ["vertx", "run",
"vertx-examples/src/raw/java/eventbus_pointtopoint/Sender.java","-cluster"]
        d.volumes = ["/src/vertx/:/usr/local/src"]
        d.vagrant_machine = "#{DOCKER_HOST_NAME}"
        d.vagrant_vagrantfile = "#{DOCKER_HOST_VAGRANTFILE}"
      end
  end
end
```

==========

For both docker containers, vagrant_mahchine, the id of the Docker host VM, is dockerhost. Vagrant will be smart enough to reuse the same instance of dockerhost to run both containers.

To start vertxsender and vertxreceiver replace the Vagrantfile with this one and run vagrant up:

```
$ vagrant up
...
$ vagrant docker-logs
==> vertxsender: Starting clustering...
==> vertxsender: No cluster-host specified so using address 172.17.0.18
```

```
==> vertxsender: Succeeded in deploying verticle

==> vertxreceiver: Starting clustering...

==> vertxreceiver: No cluster-host specified so using address 172.17.0.19

==> vertxreceiver: Succeeded in deploying verticle

==> vertxreceiver: Received message: ping!

==> vertxsender: Received reply: pong

==> vertxreceiver: Received message: ping!

==> vertxreceiver: Received message: ping!

==> vertxsender: Received reply: pong

==> vertxsender: Received reply: pong

...
```

We first run a Docker container (vertxdev) that starts up the HelloWorld web server we saw previously. Then a second container (vertxdev-client) will do an HTTP request using wget:

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

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

  config.vm.define "vertxdev" do |a|

    a.vm.provider "docker" do |d|

      d.image = "vertxdev:latest"

      d.ports = ["8080:8080"]

      d.name = "vertxdev"

      d.remains_running = true

      d.cmd = ["vertx", "run", "vertx-examples/src/raw/java/httphelloworld/HelloWorldServer.java"]

      d.volumes = ["/src/vertx/:/usr/local/src"]

      d.vagrant_machine = "dockerhost"

      d.vagrant_vagrantfile = "./DockerHostVagrantfile"
```

```
    end
  end
  config.vm.define "vertxdev-client" do |a|
    a.vm.provider "docker" do |d|
      d.image = "vertxdev:latest"
      d.name = "vertxdev-client"
      d.link("vertxdev:vertxdev")
      d.remains_running = false
      d.cmd = ["wget","-qO", "-","--save-headers","http://vertxdev:8080"]
      d.vagrant_machine = "dockerhost"
      d.vagrant_vagrantfile = "./DockerHostVagrantfile"
    end
  end
end
```

==========