Docker and VirtualBox networking

# Introduction

In this workshop we are going to use a Bitnami VM image which we will import in VirtualBox. We will configure a NAT network and expose port 80 (on which an application runs) to the host.

We will configure this image with additional packages and clone it so we have 2 VMs. These Virtual Machines will be able to connect to each other by using the NAT network. On these machines, we are going to run a Docker container each. These Docker containers will use the Docker overlay network to communicate with each other.

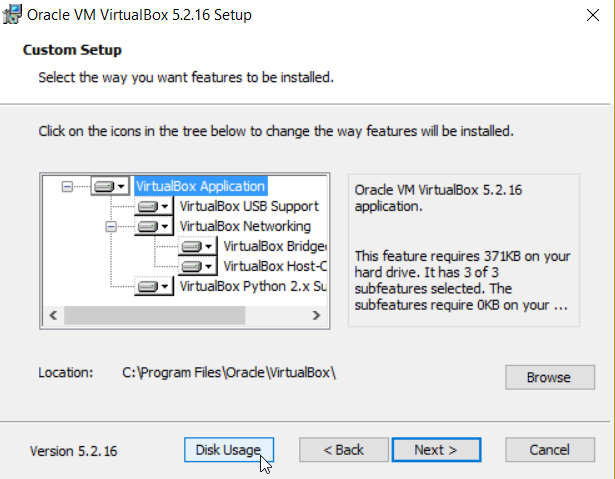
# Preparations

## VirtualBox

Install VirtualBox 5.2.16: <https://www.virtualbox.org/wiki/Downloads>.

* For Windows: <https://download.virtualbox.org/virtualbox/5.2.16/VirtualBox-5.2.16-123759-Win.exe>
* For OSX: <https://download.virtualbox.org/virtualbox/5.2.16/VirtualBox-5.2.16-123759-OSX.dmg>
* For Linux the installation differs per distribution: <https://www.virtualbox.org/wiki/Linux_Downloads>

Make sure you select both the VirtualBox Networking options: Bridged and Host-Only. They should be selected by default.



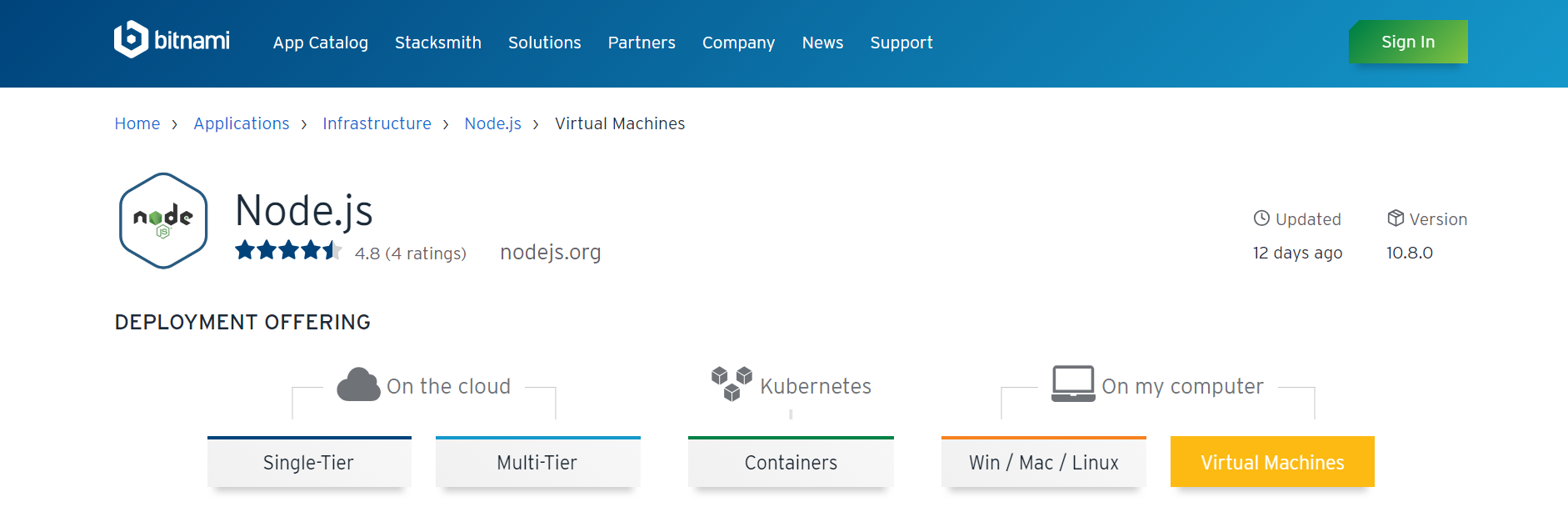
After the VirtualBox installation, install the Extension Pack.

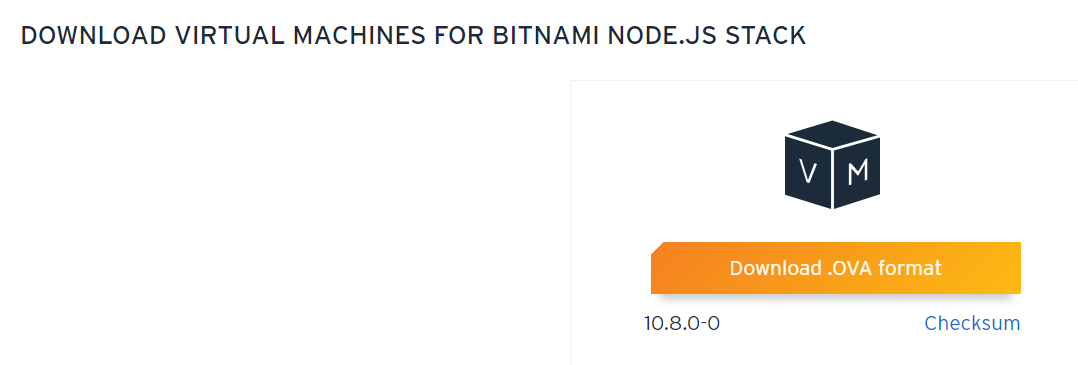
<https://download.virtualbox.org/virtualbox/5.2.16/Oracle_VM_VirtualBox_Extension_Pack-5.2.16.vbox-extpack>

# Bitnami

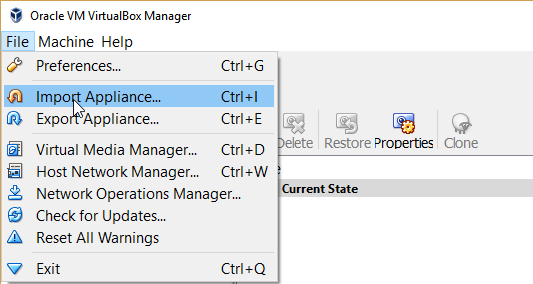
## Download the OVA

Download the Bitnami Node.js VM OVA from: <https://bitnami.com/stack/nodejs/virtual-machine>

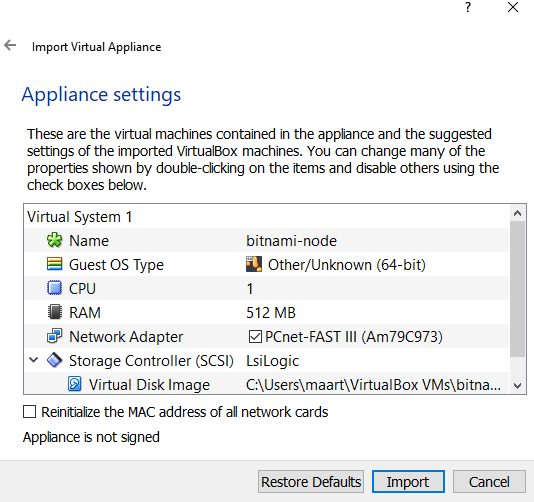




## Import the OVA in VirtualBox



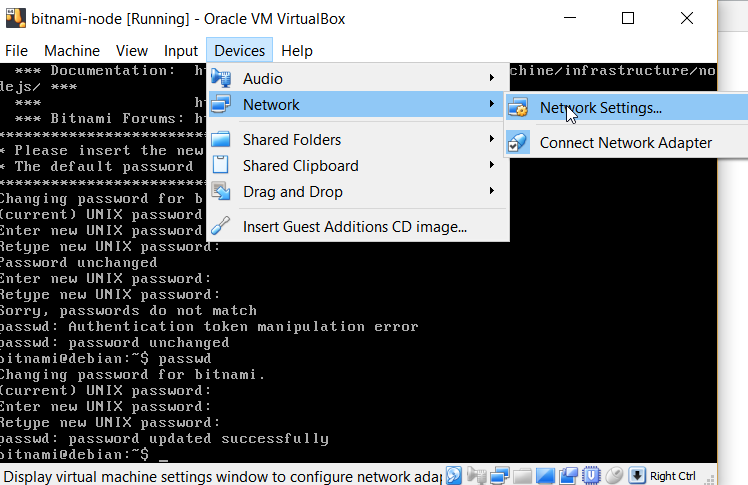
Accept the default settings

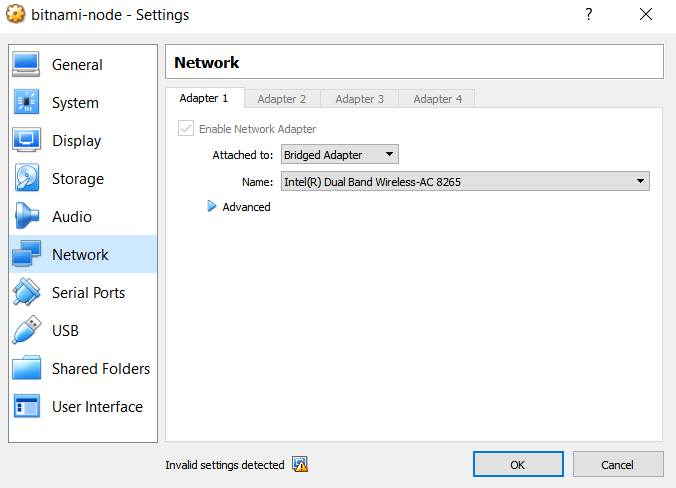


Login with user bitnami, password bitnami

Change the password to Welcome01

Check the VirtualBox network interface



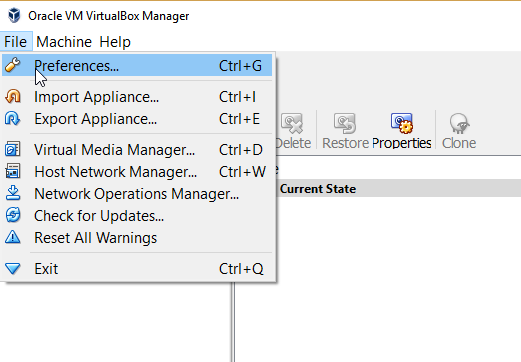


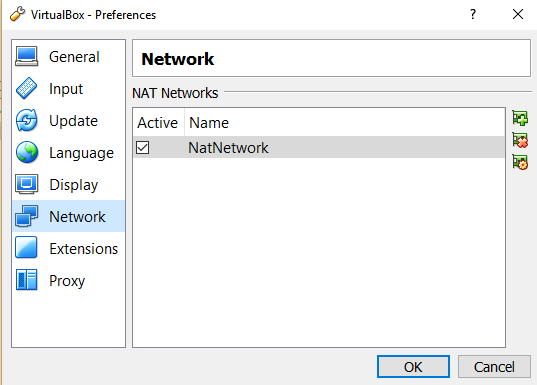
Why would this not be suitable for trying out the Docker overlay network?

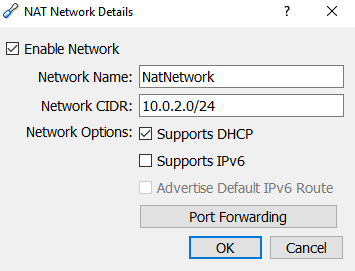
# Prepare the VM

## Create a NAT network

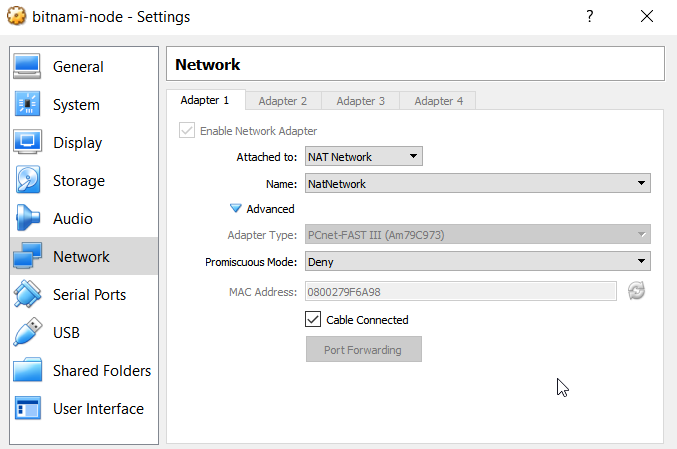
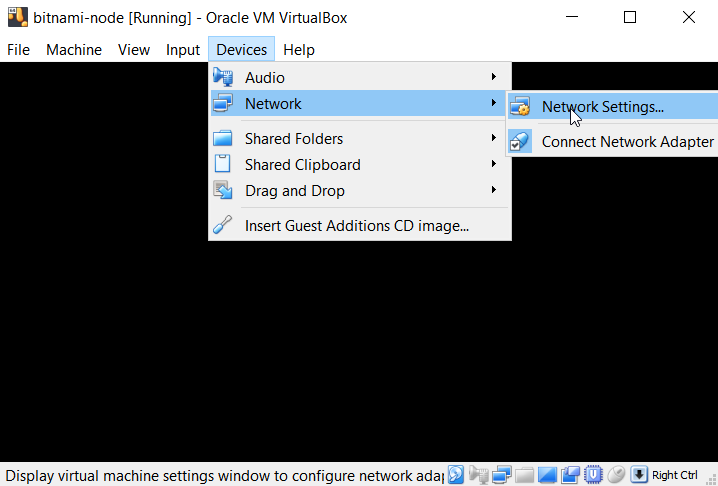
Make sure you have a NAT network defined. If not, create one.



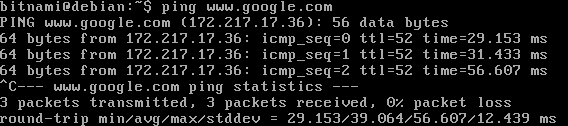




## Assign the NAT network to the guest interface



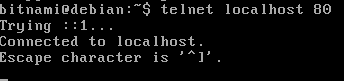
Login (bitnami, Welcome01) and confirm internet connectivity



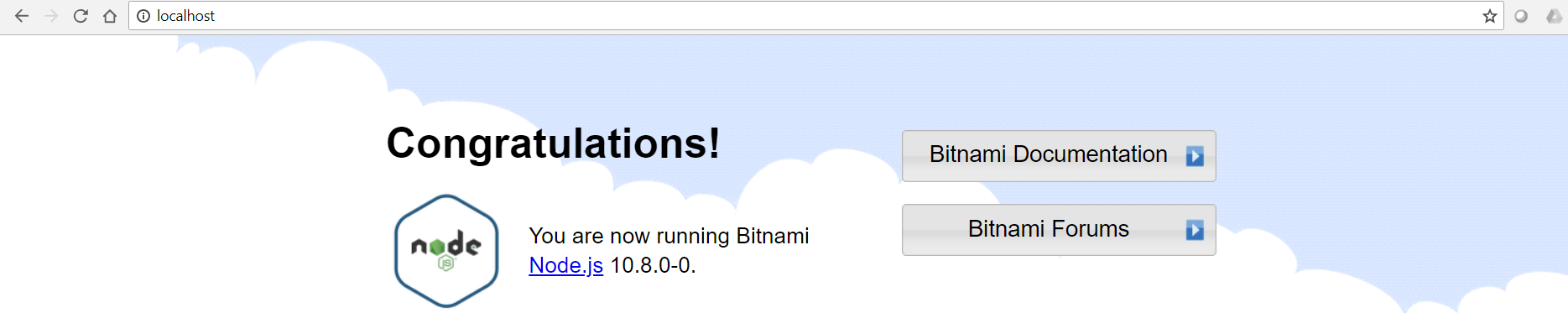
## Port forwarding and access the guest from the host

Confirm something is running on port 80 of the machine;

telnet localhost 80



Assignment: Expose port 80 to the host by creating a port forwarding rule in the NAT network configuration and check with a browser what is running on port 80.



## Easy copy and paste

We are going to execute several commands. You don’t want to type all of them manually. The VM does not have a GUI. To allow easy copy/paste actions, you can access it by using SSH

First;

sudo rm /etc/ssh/sshd\_not\_to\_be\_run

sudo systemctl start ssh.service

sudo systemctl enable ssh.service

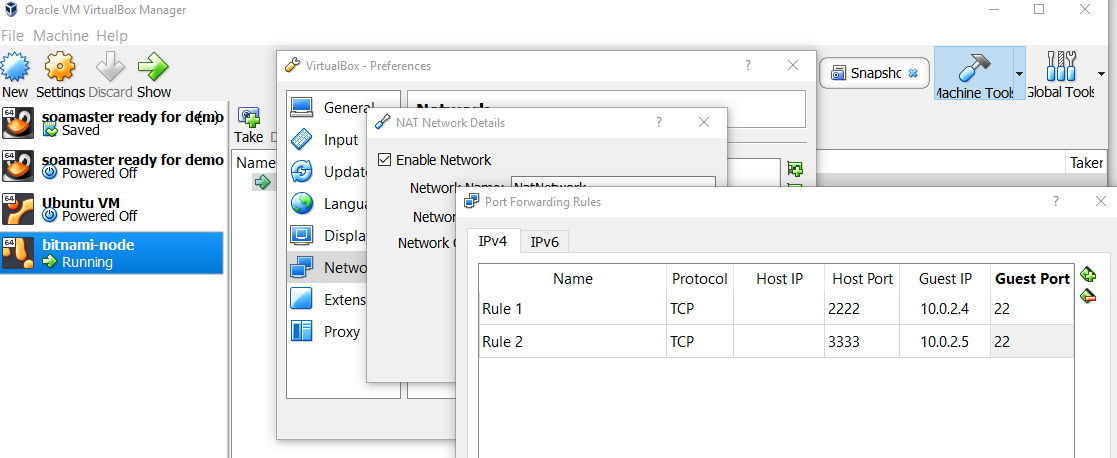
You can check if it is running with

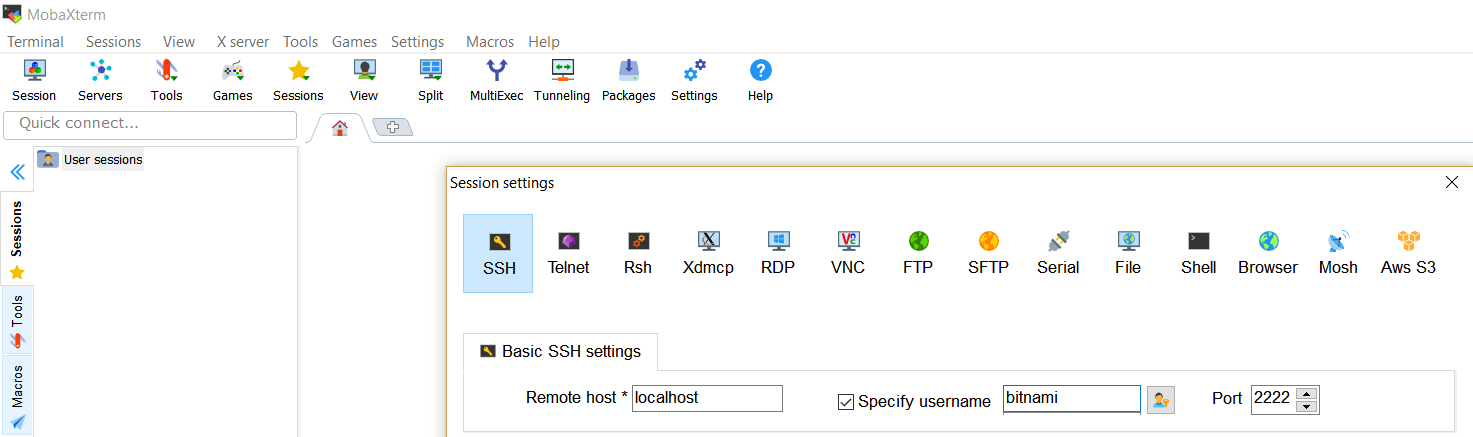
sudo systemctl status ssh.service

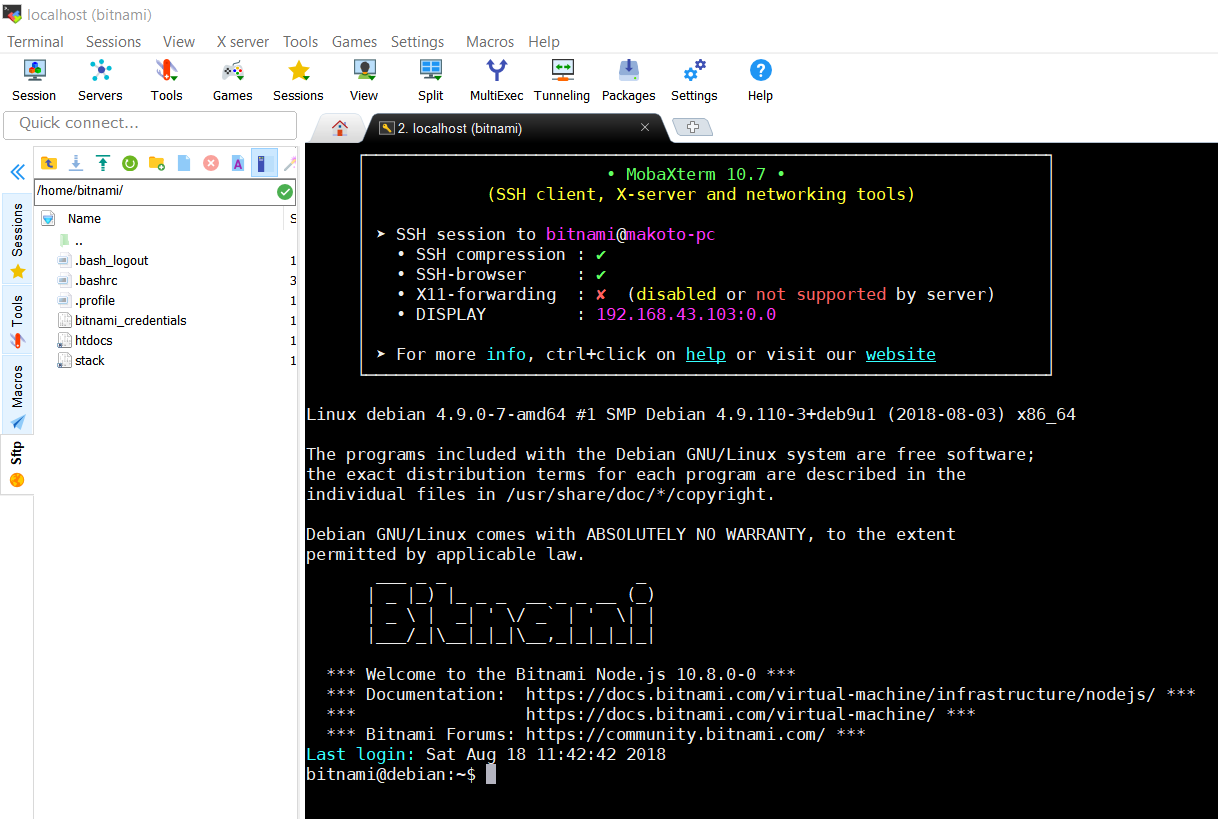
or

sudo netstat -tulpn

Create a port forwarding rule in your NAT network and use an SSH client such as (simple) Putty <https://www.putty.org/> or MobaXTerm https://mobaxterm.mobatek.net/ (more features) to connect to the client.







## Install telnet and docker

You should now be able to copy and paste to the VM via the SSH client.

sudo apt-get update

sudo apt-get install apt-transport-https ca-certificates curl gnupg2 software-properties-common telnet

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

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb\_release -cs) stable"

sudo apt-get update

sudo apt-get install docker-ce

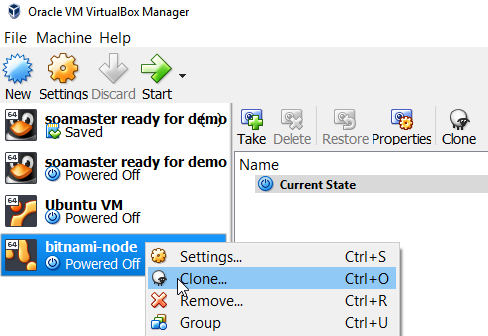
sudo usermod -aG docker bitnami

Test docker:

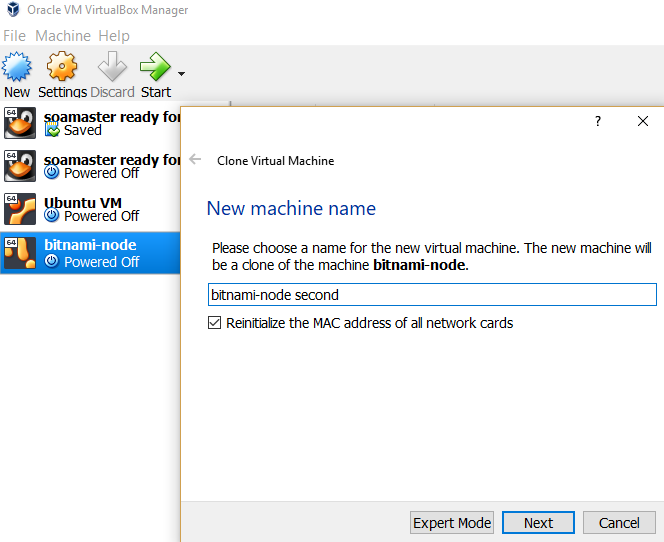
sudo docker run hello-world

## Clone the VM

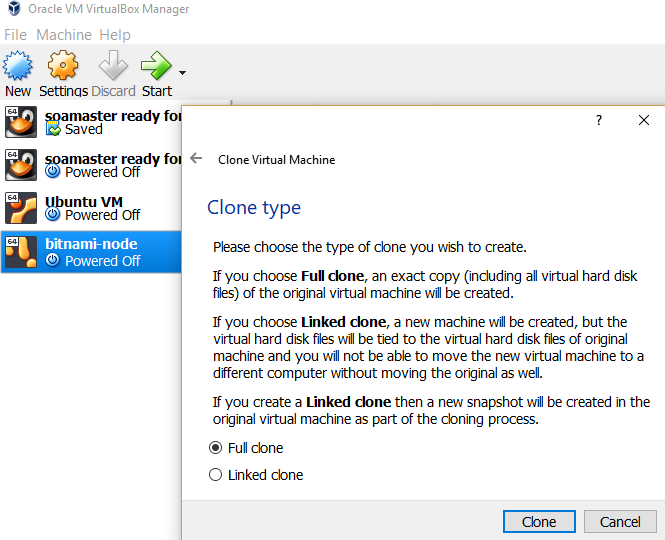
Power off the machine, clone it



Make sure to reinitialize the MAC address



Assignment: why would you need to reinitialize the MAC for the clone?



Assignment: Start both machines, determine their IPs and check if they can access each other

# Getting started with Docker overlay networking

Based on: <https://docs.docker.com/network/network-tutorial-overlay/>

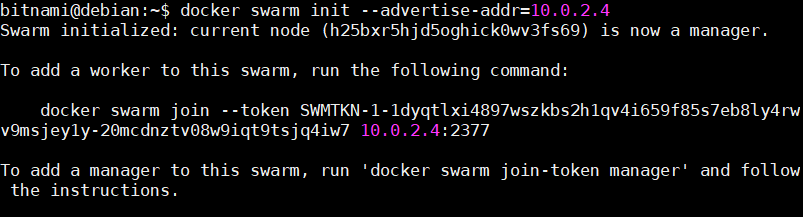
The Docker overlay network is used by Docker swarm. Docker provides a native Docker engine clustering feature called swarm. This is a limited, simple, lightweight but less feature rich alternative to Kubernetes.

The (VirtualBox) hosts which run the clustered Docker engines, should have different hostnames. Edit (vi, see <https://www.howtogeek.com/102468/a-beginners-guide-to-editing-text-files-with-vi/> if you are unfamiliar with it) /etc/hosts and /etc/hostname on both hosts and give them different hostnames. E.g. replace debian with debian1 on the first host and debian with debian2 on the second host.

On the first host start a master.

For this you will need the IP of the machine. 10.0.2.4 in the below command is an example.

docker swarm init



Disable the firewall to allow connections to the host (for a worker to join the swarm, port 2377 must be accessible)

sudo ufw disable

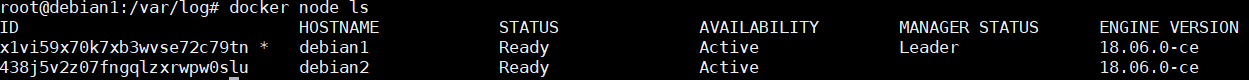
On the other host (the worker) do the same and join the swarm (described below)

Copy the command as shown above (with the token) and add the advertise address of the worker. In this case:

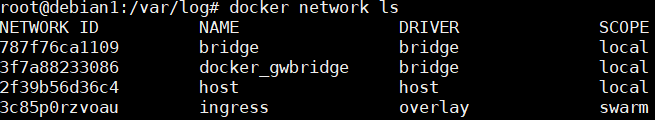
docker swarm join --token SWMTKN-1-1dyqtlxi4897wszkbs2h1qv4i659f85s7eb8ly4rwv9msjey1y-20mcdnztv08w9iqt9tsjq4iw7 10.0.2.4:2377

You should see a message: This node joined a swarm as a worker

On the master list the nodes;



docker network ls



The docker\_gwbridge connects the ingress network to the Docker host’s network interface so that traffic can flow to and from swarm managers and workers. If you create swarm services and do not specify a network, they are connected to the ingress network.

The same as with regular networks, you should create separate named networks to provide isolation.

On the manager create a new overlay network:

docker network create -d overlay nginx-net

This is only required on the master.

Create a 5-replica Nginx cluster:

docker service create \

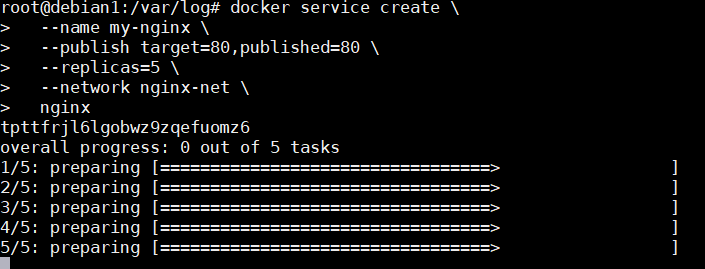
--name my-nginx \

--publish target=80,published=80 \

--replicas=5 \

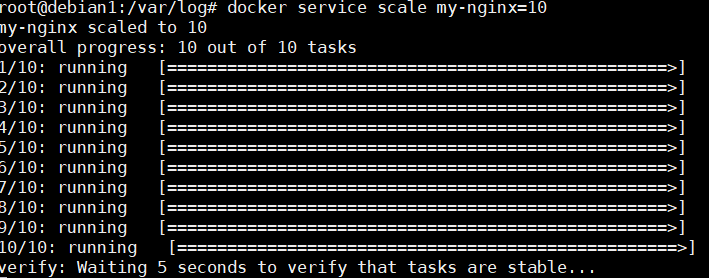
--network nginx-net \

nginx

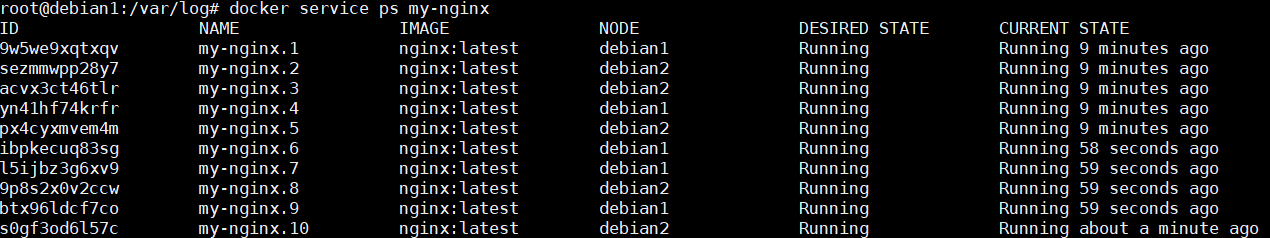


Execute docker ps -a on both hosts. Confirm 5 replica’s are running distributed over the 2 hosts.

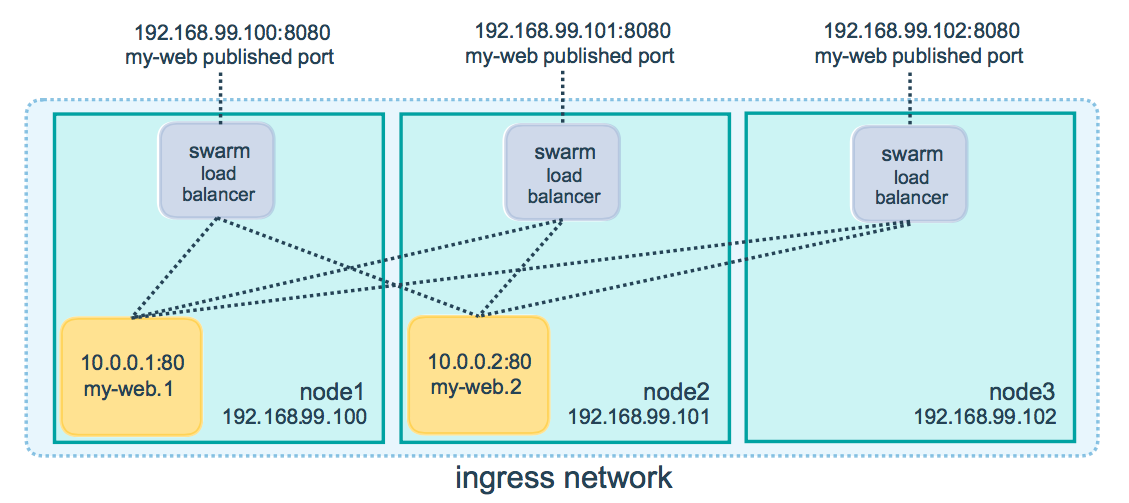
Scale up to 10 nodes:



Check where the nodes are running:



Docker swarm does load balancing. See <https://docs.docker.com/engine/swarm/ingress/#publish-a-port-for-a-service>. Below is an example, IP’s differ.



Assignment: Test this by configuring the NAT network to forward port 80 on the master to port 80 on the (VirtualBox) host and opening port 80 in a browser on the host.

Scale down to 3 hosts

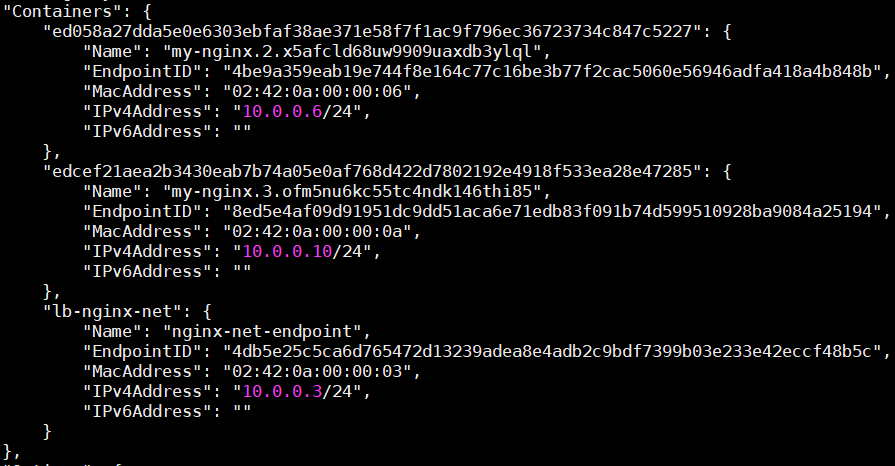
docker service scale my-nginx=3

Check the nginx network:

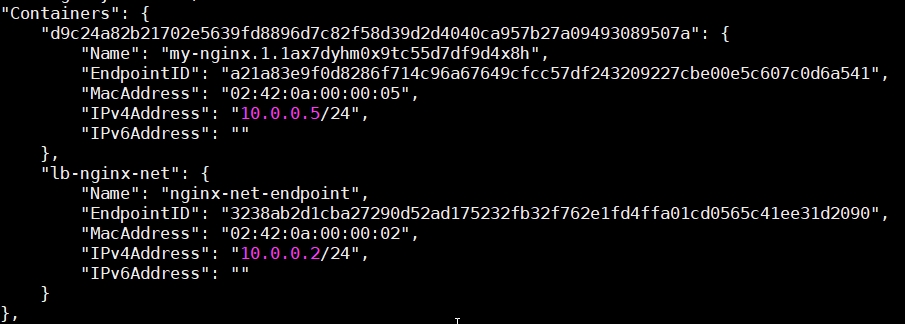
docker network inspect nginx-net

Determine the IP’s of the containers on both hosts (below is an example)

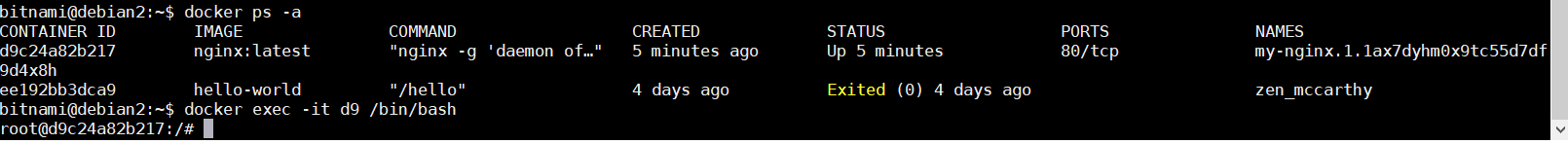
Host 1:



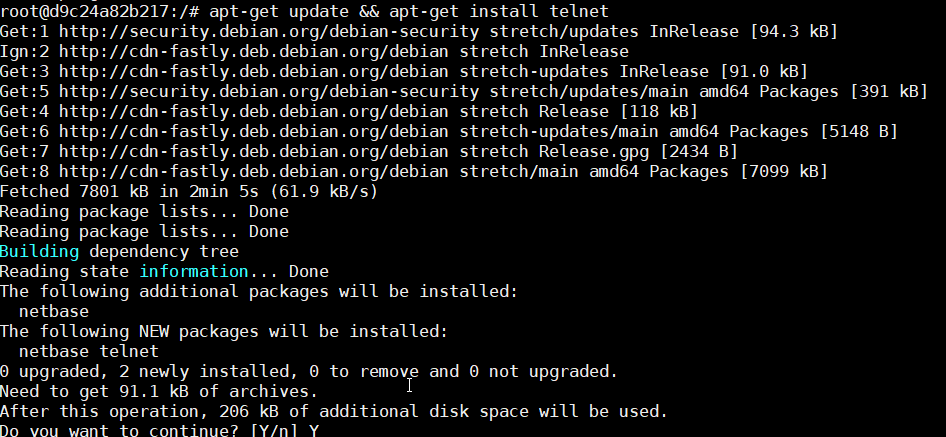
Host 2:



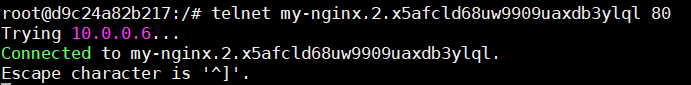
Enter a container. Determine its ID and start a bash shell in it. You don’t have to specify the entire container ID, just enough to make it unique.



Install telnet and test connectivity



Confirm you can access the IP/hostname from a docker container on one host to the other via the overlay network you have created.

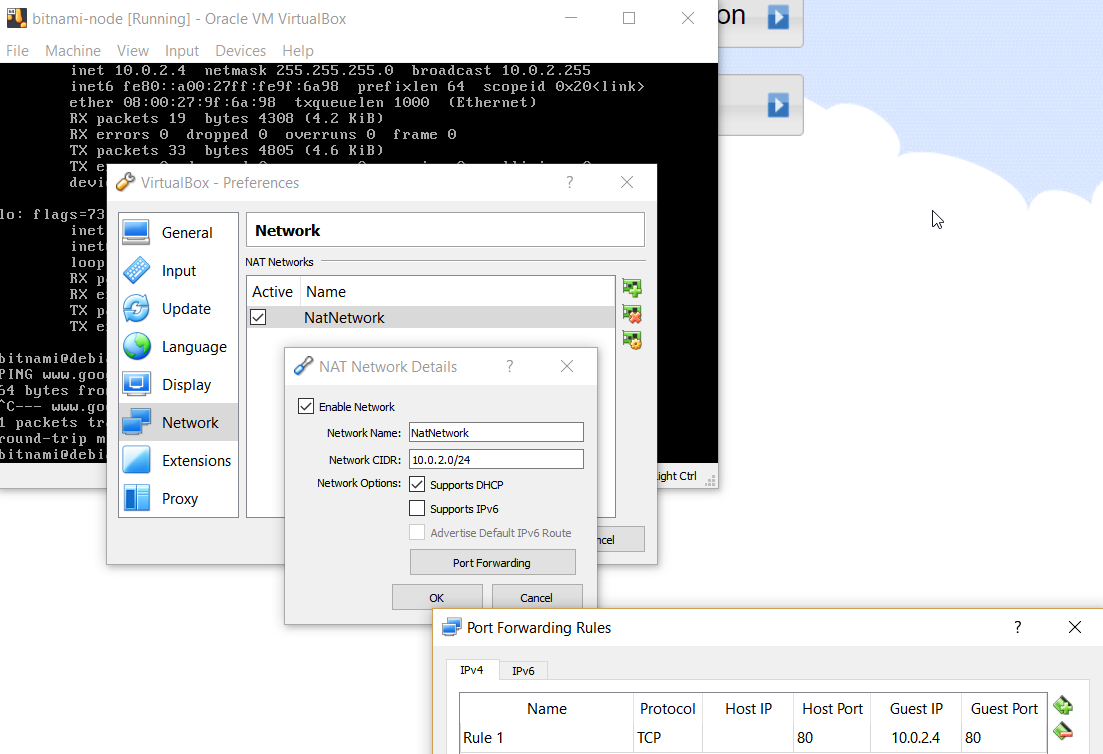
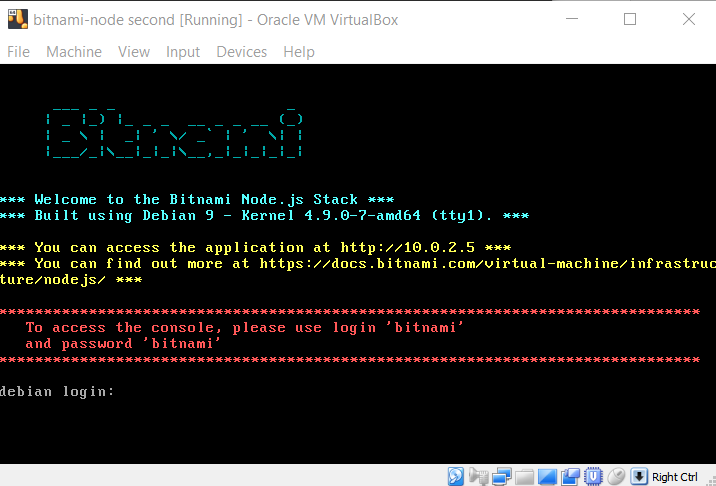
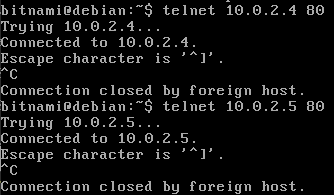


Leave the swarm:

docker swarm leave on the worker and docker swarm leave --force on the manager

Confirm no containers are left running with docker ps -a

# Solutions

* Why would this not be suitable for trying out the Docker overlay network?  
  The bridged network would directly attach the VM’s interface to the hosts interface. This would mean that even for getting an IP, the VM is dependent on the host network. Some company networks might not give out an IP since the MAC address which is used is different from the hosts MAC address. A better solution would be to use a NAT network which provides local DNS to the host and still provides internet access for downloading packages.
* Assignment: Expose port 80 to the host by creating a port forwarding rule and check with a browser what is running on port 80. In order to do this, you will need to configure a port forwarding rule in the NAT network configuration. This is not done on host interface level such as with NAT.  
  First determine the IP of the guest: /sbin/ifconfig (look at the inet entry of the interface which isn’t lo, top line in the screenshot)  
  Next configure the port forwarding rule accordingly;  
  
* First determine the IP. It’s on the first screen displayed. See below; 10.0.2.5 in my case.  
  Next check connectivity between the VM’s  
  
* Assignment: why would you need to reinitialize the MAC for the clone? If the two guests have the same MAC, they will receive the same IP from the NAT network DHCP server and it will be a challenge to connect them.
* Assignment: Test this by configuring the NAT network to forward port 80 on the master to port 80 on the host  
  