[Nginx Docker install via Ansible](http://www.bogotobogo.com/DevOps/Ansible/Ansible-Deploy-Nginx-to-Docker.php)

Note

This tutorial is experimental or just to play with Ansible and Docker. Actually, we need to setup ssh within the container, and then Ansible installs nginx from outside, which is really weird situation.

Also, trying to install/run ssh on a container may not be a proper way of start using Docker, and as showed in my other tutorials (listed below) are probably the ways we want to use Ansible and Docker together:

1. Launch micro services (wordpress, mysql, varnish, proxy server separately on each container):  
   [Deploying Wordpress micro-services with Docker containers on Vagrant box via Ansible](http://www.bogotobogo.com/DevOps/Ansible/Docker-WordPress-Microservices-with-Nginx-reverse-proxy-Varnish-Mysql-Deployed-via-Ansible.php)
2. All in one (nginx server with app in one container). But we can deploy two env containers (dev and prod).  
   [Quick Preview - Setting up web servers with Nginx, configure enviroments, and deploy an App](http://www.bogotobogo.com/DevOps/Ansible/Ansible_SettingUp_Webservers_Nginx_Install_Env_Configure_Deploy_App.php)

However, with the current approach, I ended up running ssh using "docker exec" against a running container.

Anyway, I hope this experimental (playful) page has something we can learn from.

Install ssh to Docker container

There have been quite a few debates on docker ssh whether it's a right approach or not. However, this tutorial is not about ssh but about Ansible. So, I stick to sshing to a container.

# apt-get install openssh-server

Create a user other than root:

# adduser k

Then, copy public key to /home/k/.ssh/authorized\_keys.

Later the newly create user should be able to install packages, so we need to add the user 'k' to 'sudoers':

# adduser k sudo

Also, we prefer not to use password when we try to install something as sudo. So, let's edit sudoers using "visudo":

# %sudo ALL=(ALL:ALL) ALL =>

%sudo ALL=(ALL) NOPASSWD: ALL

As will be discussed later, we want to set the ssh to auto-start at boot otherwise it won't running when we start the docker run. So, we cannot ssh to the container. Let's check it if it's set that way:

# ls -l /etc/rc?.d/\*ssh

ls: cannot access /etc/rc?.d/\*ssh: No such file or directory

Indeed. It's not set. So, let's do it now:

# sudo update-rc.d ssh defaults

update-rc.d: warning: default stop runlevel arguments (0 1 6) do not match ssh Default-Stop values (none)

Adding system startup for /etc/init.d/ssh ...

/etc/rc0.d/K20ssh -> ../init.d/ssh

/etc/rc1.d/K20ssh -> ../init.d/ssh

/etc/rc6.d/K20ssh -> ../init.d/ssh

/etc/rc2.d/S20ssh -> ../init.d/ssh

/etc/rc3.d/S20ssh -> ../init.d/ssh

/etc/rc4.d/S20ssh -> ../init.d/ssh

/etc/rc5.d/S20ssh -> ../init.d/ssh

Autorun at boot time is set. Though it supposed to start automatically when we start to run this container, it's not.

So, at this point, we may want to issue "docker exec" against the running container:

$ docker exec -it <container-id> bash -c "service ssh restart"

Anyway, we can still move on because it does not affect what we're try to do.

Get Docker ip

$ docker inspect --format '{{ .NetworkSettings.IPAddress }}' $(docker ps -q)

172.17.0.5

SSH to docker container

Now that we know the ip of the docker container, we can ssh to it:

$ ssh k@172.17.0.5

Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-40-generic x86\_64)

...

k@b9b2ee7004fe:~$

Ansible ping test

This time, we can ping the container using Ansible.

First, we need to create a customized inventory: **~/ansible\_hosts**:

[mydocker]

172.17.0.5

Then, let Ansible know where it can find hosts:

$ export ANSIBLE\_INVENTORY=~/ansible\_hosts

$ echo $ANSIBLE\_INVENTORY

/home/k/ansible\_hosts

Let's ping it via Ansible:

$ ansible mydocker -m ping

172.17.0.5 | SUCCESS => {

"changed": false,

"ping": "pong"

}

Yes, we can connect to the container via Ansible.

Docker image save

At this point, we may want to save the works we've onto Ubuntu image. So, exit from docker.

We're now in a regular linux shell. Let's see what's in the docker process list:

$ docker ps -l

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

b9b2ee7004fe ubuntu:14.04 "/bin/bash" 2 hours ago Exited (0) 3 minutes ago fervent\_galileo

To save a new image, we should commit it:

$ docker commit b9b2ee7004fe ubuntu-ssh-k

where the 'b9b2ee7004fe' is the container-id and 'ubuntu-ssh-k' is the new name of the image.

Let's rerun it:

$ docker run -it ubuntu-ssh-k /bin/bash

root@517d0f895d8e:/# groups k

k : k sudo

We are back to where we left off!

But there is a catch. IP address of our container has been changed:

$ docker inspect --format '{{ .NetworkSettings.IPAddress }}' $(docker ps -q)

172.17.0.7

So, we need to update Ansible hosts in **/home/k/ansible\_hosts**.

Also, we need to more on the container: restart ssh:

root@e2481c1bad5d:/# service ssh restart

If the sss service won't start, we may have to do one more thing before we restart ssh servcie.

# /usr/bin/ssh-keygen -A

Nginx install on Docker

To install Ngix on Ubuntu servers, we'll use **apt** module.

k@laptop:~$ ansible mydocker -s -m apt -a 'pkg=nginx state=installed update\_cache=true'

172.17.0.7 | SUCCESS => {

"cache\_update\_time": 1465365329,

"cache\_updated": true,

"changed": true,

"stderr": "invoke-rc.d: policy-rc.d denied execution of start.\n",

"stdout": "Reading package lists...\nBuilding dependency tree...\nReading state information...\nThe following

...

"stdout\_lines": [

"Reading package lists...",

...

"Processing triggers for libc-bin (2.19-0ubuntu6.6) ..."

]

}

k@laptop:~$

This will use the **apt** module to update the repository cache and install Nginx. If we run it again:

k@laptop:~$ ansible mydocker -s -m apt -a 'pkg=nginx state=installed update\_cache=true'

172.17.0.7 | SUCCESS => {

"cache\_update\_time": 1465365644,

"cache\_updated": true,

"changed": false

}

k@laptop:~$

The result of running the Task was **"changed": false**. This shows that there were no changes; We had already installed Nginx. We can run this command repeatedly without worrying about it affecting the desired result (idempotence).

The **-a** flag is used to pass any arguments to the module. I use **-s** to run this command using **sudo**.

We can run all of our tasks using modules in this ad-hoc way, however, we want to perform these task in more managable way using a **Playbook**, which can run and coordinate multiple tasks.

Running Nginx on Docker

On our container, let's run nginx:

root@e2481c1bad5d:/# service nginx restart

\* Restarting nginx nginx [ OK ]

root@e2481c1bad5d:/#

