Automation with Ansible

The DevOps revolution has no end of brilliant projects and products that promise to get you closer to the “Infrastructure as Code” ideology. I’ve briefly introduced the rapid deployment of virtual machines using Vagrant here and now it’s time to introduce Ansible.

Ansible is a tool that should be available from your repositories already, just like the afore mentioned Vagrant. It’s a RedHat project, but is available across the Linux distribution landscape. So on Debian/Ubuntu/Mint, installation is as easy as…

sudo apt-get install ansible

It is agentless, which is great as it radically simplifies the process of getting up and running, but like many agentless tools *(the ones that don’t require the installation of a client daemon on all machines),* you will either need to be using a directory admin account that is already set up to have privileges on all other servers in the domain, or else copy SSH keys out to all the machines that you intend to use ansible against in order to bring automation and consistency to your linux network. The process of setting up passwordless authentication has already been covered here but it’s simple enough so I’ll summarise it here for convenience.

Lets say you have a machine *linux1* with a user *matt* that you want to use as your ansible “server” to run commands against servers *linux10, linux11* and *linux12*. The other servers also have a user *matt* but it’s a local user, not a user in a directory. In order for *matt* on *linux1* to be accepted as being synonymous with matt on the other servers *linux10, linux11 and linux12, matt*‘s SSH keys will need to be generated and the public key copied to the other machines.

In order for matt on linux1 to be accepted as being

> *synonymous with matt on the other servers linux10, linux11 and linux12, matt's SSH keys will need to be generated on linux1 and the public key copied to the other linux10, 11 and 12 machines.*

su – matt

ssh-keygen *(Note: do not use passphrase, leave blank or you'll be prompted every time you attempt a passwordless connection to a remote host and this will obstruct using your public key authentication as root on remote system.)*

cd .ssh

ssh-copy-id -i id_rsa.pub linux10 linux11 linux12 (*Note: On reflection, use the full path to the id_rsa file, e.g.* ***/home/root/.ssh/id_rsa.pub****. This is because there is the potential to su to root and land in the previous users .ssh folder, and subsequently copy that users keys instead of the root users. You'll be hours figuring that one out).*

Now that we've got that out of the way, we can get back to the subject in hand, namely ansible. Ansible is a way of doing away with having to ssh to every machine in order to execute something locally on that remote machine in order to make it consistent with the other machines in your enterprise environment.

> *Ansible is a way of doing away with having to ssh to every machine in order to execute something locally on that remote machine in order to make it consistent with the other machines in your enterprise environment.*

There are many modules available in ansible, documented here but in order to keep this introduction to ansible simple, we'll just demo the *command* module.

There is just one last thing to set up before that, and that is a hosts file that groups together your hosts in your network. hosts can belong to more than one group, but in this simple demo, we need to create a file called *hosts* and in it, create a group called *[group1]* with *linux10, linux11* and *linux12* hosts as members…

vi hosts

[group1]

linux10

linux11

linux12

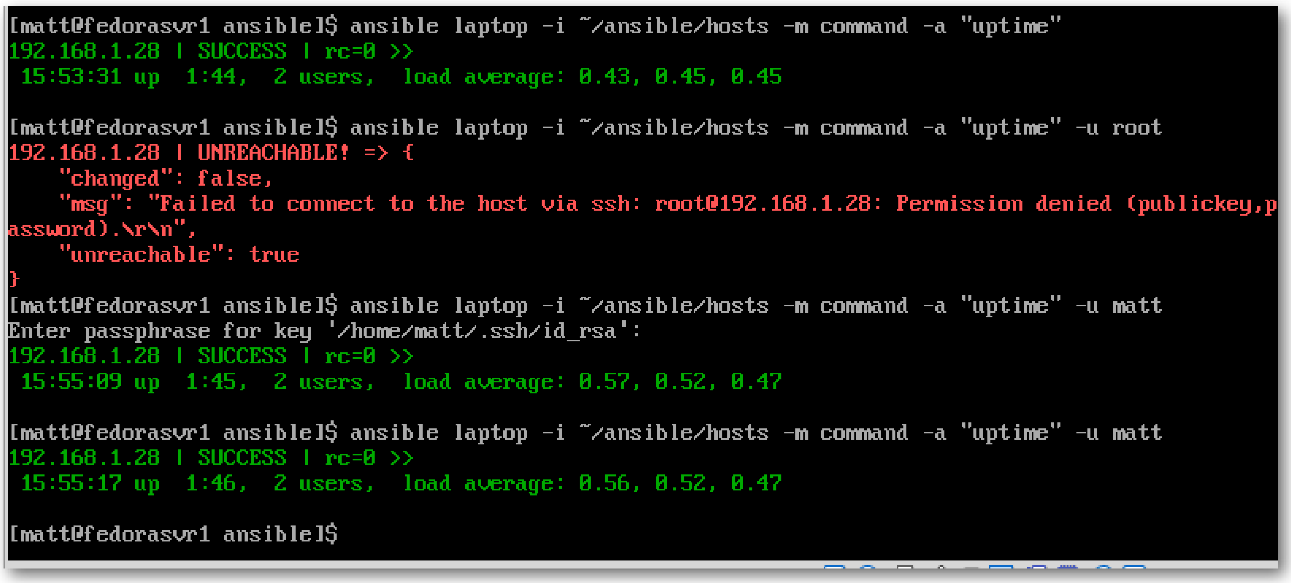
With this group created, we can now execute a command against each of the hosts in the group using ansible.

The syntax is **ansible**, followed by the **group** name, followed by **-i** (information), in our case the **hosts** file *(not to be confused with /etc/hosts)* , followed by **-m** *(module name, in our case* **command** *module),* followed by **-a** *(arguments to be passed to the module, in our case* **"uname -a"***).*

ansible group1 -i ./hosts -m command -a "uname -a"

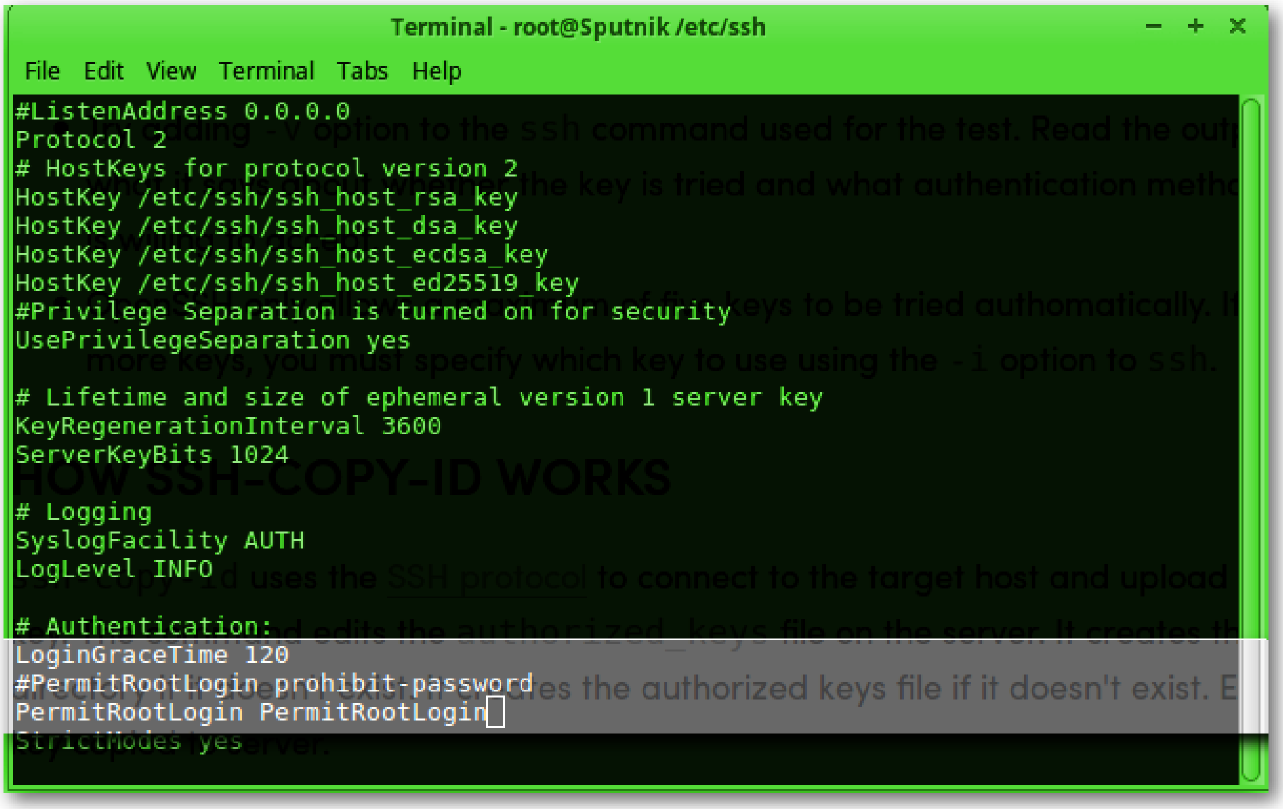
This will return the results of running uname -a on each of the servers listed in the group in our hosts file, to stdout just as if we had ssh'd to each of them in the same terminal and executed the command.

The example below shows the results of executing **uptime** against my laptop from a centos vm running on virtualbox, as user matt, where the ssh keys have been prior copied to my laptop, then again as the root user where the ssh keys have not. Note also that once the passphrase has been entered once for the user, that's it from that point on and the ansible host is effectively trusted to execute commands on remote hosts. Powerful and Convenient stuff.



```
[matt@fedorasvr1 ansible]$ ansible laptop -i ~/ansible/hosts -m command -a "uptime"
192.168.1.28 | SUCCESS | rc=0 >>
 15:53:31 up  1:44,  2 users,  load average: 0.43, 0.45, 0.45

[matt@fedorasvr1 ansible]$ ansible laptop -i ~/ansible/hosts -m command -a "uptime" -u root
192.168.1.28 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: root@192.168.1.28: Permission denied (publickey,p
assword).\r\n",
    "unreachable": true
}
[matt@fedorasvr1 ansible]$ ansible laptop -i ~/ansible/hosts -m command -a "uptime" -u matt
Enter passphrase for key '/home/matt/.ssh/id_rsa':
192.168.1.28 | SUCCESS | rc=0 >>
 15:55:09 up  1:45,  2 users,  load average: 0.57, 0.52, 0.47

[matt@fedorasvr1 ansible]$ ansible laptop -i ~/ansible/hosts -m command -a "uptime" -u matt
192.168.1.28 | SUCCESS | rc=0 >>
 15:55:17 up  1:46,  2 users,  load average: 0.56, 0.52, 0.47

[matt@fedorasvr1 ansible]$
```

If you want to be able to use ansible as root to execute commands remotely *(using ansibles -b option, i.e. become)* then you'll need to copy the root users ssh keys over to the remote hosts too. You can do this the exact same way as you copy over any other users ssh keys, only this one comes with an added obstacle – ssh as root is not permitted by default in most modern linux distributions as a way of hardening against a brute force attack as root. Sensible stuff, and not that difficult to overcome. You just need to edit the **/etc/ssh/sshd-config** file on the remote host to **permit root login** while you copy the keys across.

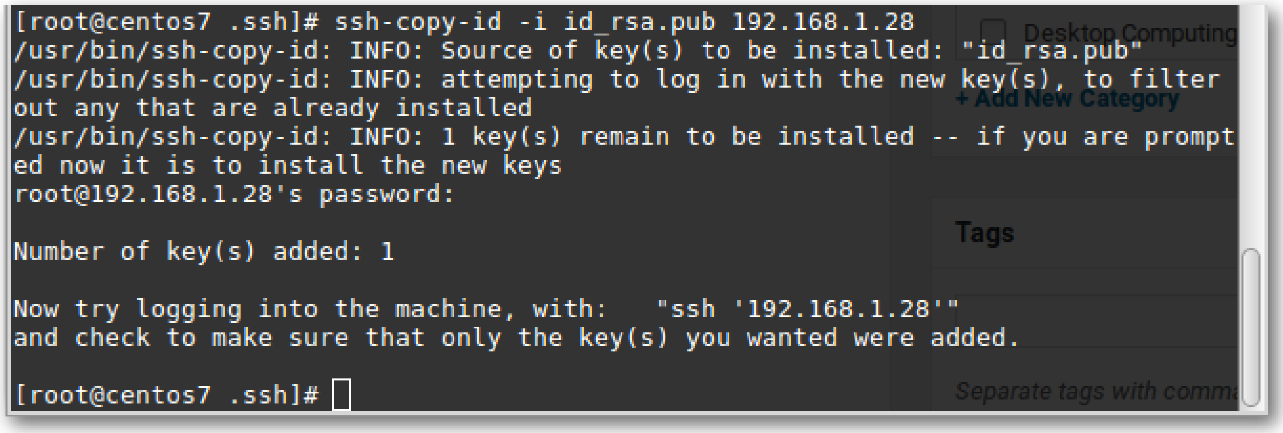

Terminal - root@Sputnik /etc/ssh
File Edit View Terminal Tabs Help

```
#ListenAddress 0.0.0.0
Protocol 2
# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
HostKey /etc/ssh/ssh_host_ed25519_key
#Privilege Separation is turned on for security
UsePrivilegeSeparation yes

# Lifetime and size of ephemeral version 1 server key
KeyRegenerationInterval 3600
ServerKeyBits 1024

# Logging
SyslogFacility AUTH
LogLevel INFO

# Authentication:
LoginGraceTime 120
#PermitRootLogin prohibit-password
PermitRootLogin PermitRootLogin
StrictModes yes
```

Just comment out the existing **PermitRootLogin prohibit-password** line and replace it with **PermitRootLogin yes.** Note: not **PermitRootLogin PermitRootLogin** as in the example above – I couldn't restart sshd.

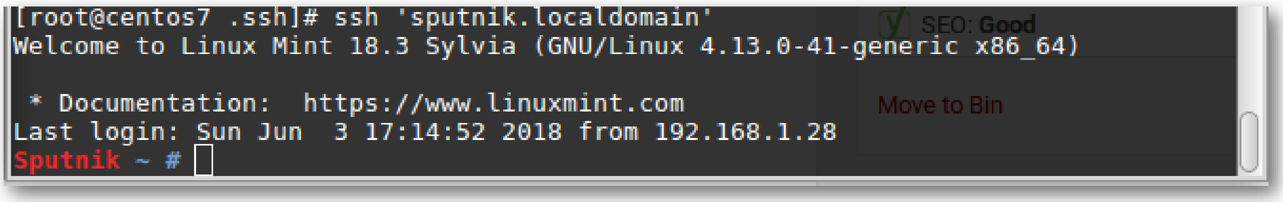
service sshd restart

And voila, the root users ssh keys copy across fine.



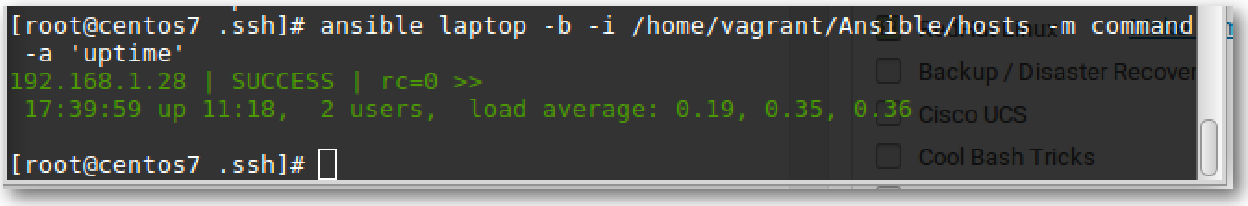
```
[root@centos7 .ssh]# ssh-copy-id -i id_rsa.pub 192.168.1.28
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompt
ed now it is to install the new keys
root@192.168.1.28's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh '192.168.1.28'"
and check to make sure that only the key(s) you wanted were added.

[root@centos7 .ssh]# 
```

Now you need to change the ssh-config file back to **PermitRootLogin prohibit-password** and restart sshd again to put the system back to it’s secure default state whereby the root user is allowed to attempt a connection, it’s just not allowed to send a password. If ssh keys are in place of course, passwords don’t need to be sent – that’s the whole point of ssh keys, after all!



```
[root@centos7 .ssh]# ssh 'sputnik.localdomain'
Welcome to Linux Mint 18.3 Sylvia (GNU/Linux 4.13.0-41-generic x86_64)

 * Documentation:  https://www.linuxmint.com
Last login: Sun Jun  3 17:14:52 2018 from 192.168.1.28
Sputnik ~ #
```

Voila, I can now ssh to the remote system as root, even thought the ssh daemon on the remote system is configured to not permit password authentication for inbound connections by the user *root*. If that’s the case, then you will now be able to use the **ansible -b option *(become)*** to execute commands or playbooks to configure remote systems as root.

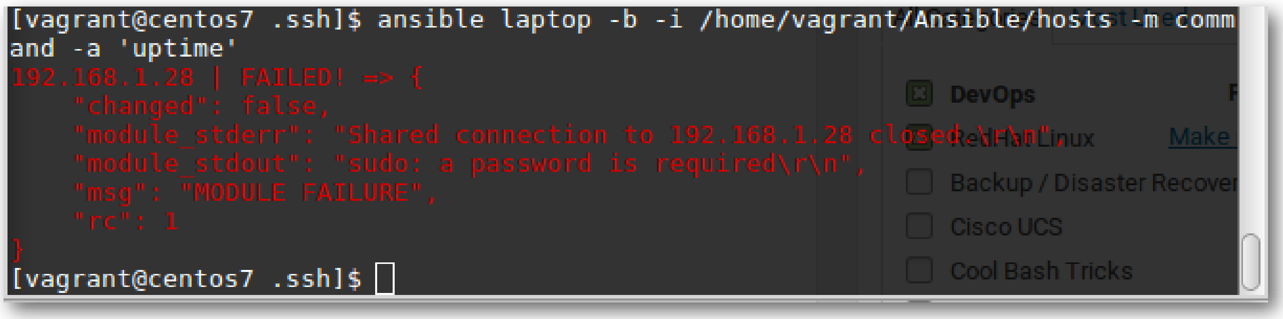


```
[root@centos7 .ssh]# ansible laptop -b -i /home/vagrant/Ansible/hosts -m command
 -a 'uptime'
192.168.1.28 | SUCCESS | rc=0 >>
 17:39:59 up 11:18,  2 users,  load average: 0.19, 0.35, 0.36

[root@centos7 .ssh]#
```

At this point, you may find yourself saying “not on my system it doesn’t!”

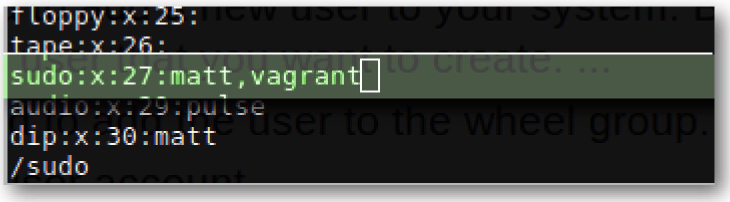
If that’s the case, please go to the end of the post and read the Troubleshooting SSH connections section for tips on what to do.

Although ansible now works as root on remote systems, you’ll find that sudoers throws you an error when attempting to use the -b (become root) option when running the ansible command as a user other than root on the ansible server.



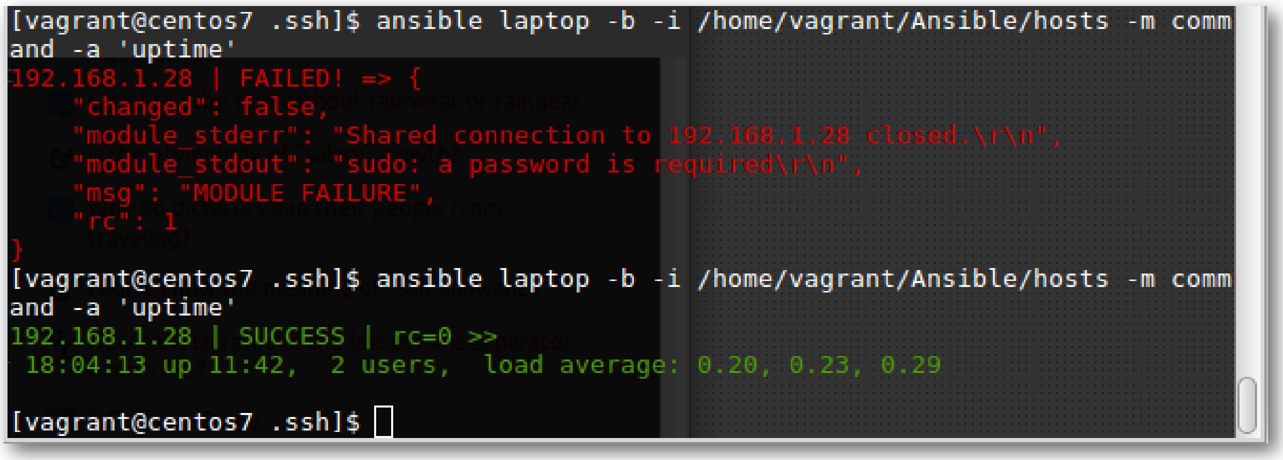
```
[vagrant@centos7 .ssh]$ ansible laptop -b -i /home/vagrant/Ansible/hosts -m comm
and -a 'uptime'
192.168.1.28 | FAILED! => {
    "changed": false,
    "module_stderr": "Shared connection to 192.168.1.28 closed.\r\n",
    "module_stdout": "sudo: a password is required\r\n",
    "msg": "MODULE FAILURE",
    "rc": 1
}
[vagrant@centos7 .ssh]$ 
```

Adding the user to the sudo group on the remote host doesn't fix this either, since the sudoers mechanism will still (by default) ask for the users password in order to run a command as root.



```
floppy:x:25:
tape:x:26:
sudo:x:27:matt,vagrant
audio:x:29:pulse
dip:x:30:matt
/sudo
```

Once this edit has been made to sudoers *using **visudo*** then you can see below, that re-running the same ansible -b command as the vagrant user, successfully executes the uptime command as root on the remote system.



```
[vagrant@centos7 .ssh]$ ansible laptop -b -i /home/vagrant/Ansible/hosts -m comm
and -a 'uptime'
192.168.1.28 | FAILED! => {
    "changed": false,
    "module_stderr": "Shared connection to 192.168.1.28 closed.\r\n",
    "module_stdout": "sudo: a password is required\r\n",
    "msg": "MODULE FAILURE",
    "rc": 1
}
[vagrant@centos7 .ssh]$ ansible laptop -b -i /home/vagrant/Ansible/hosts -m comm
and -a 'uptime'
192.168.1.28 | SUCCESS | rc=0 >>
 18:04:13 up 11:42,  2 users,  load average: 0.20, 0.23, 0.29

[vagrant@centos7 .ssh]$ 
```

And therein ends my initial introduction to *ansible* and hopefully some tips on getting it working the way you want. *Playbooks* will be covered in a separate post.

Troubleshooting SSH connections

You may find yourself having issues with connecting as root or any other user for that matter. Despite having created and copied you public keys to the remote systems, you’re still being prompted for passphrases or passwords for the user, defeating the whole point of setting up passwordless authentication.

Here’s a quick checklist of things to look out for and ways to troubleshoot the connection.

service stop sshd && /usr/sbin/sshd -d *(restart sshd in debug mode on the remote machine)*

ssh -vv <*remote-host*> *(connect to the remote host using ssh in verbose mode)*

Before Googling the errors, make sure you can confirm the following:

When you generated the public keys using **ssh-keygen** you left the passphrase blank.

When you copied the keys over to the remote machine using **ssh-copy-id** you used the **full path to the id_rsa.pub** file. If you’re root, it’s quite probable you copied another users ssh keys over instead of your own!

The .ssh directory in the users home directory has 700 permissions and the authorized-keys file has 600 permissions.