/var/log/mind

I conducted a hands on tutorial for ansible on Oct 26, 2013 over Google Hangout. These are the session instructions I published just prior to the session.

What is ansible

A configuration management tool. Helps you automate your devops, system / operations administration tasks.

Some salient aspects

- Primarily uses a simple dsl and yaml data files
- Uses python as a default languages for modules / extensions
- Uses a push based model by default, though a pull based model also supported

About this tutorial

Source code

You can check out all the source code from bitbucket repository at https://bitbucket.org/dnene/floo .

Almost all the commits are tagged and checking out the appropriate commit can help you review the source at that particular point in time

Assumed operating systems

Both the host and remote / guest operating systems are assumed to be ubuntu 12.04. In most cases if you use a different operating system it should not matter. In case you use a non debian based linux operating system, you may need to read the `apt` related commands and reinterpret them for your operating system (eg. you may want to use `yum` if on fedora). Also the corresponding ansible tasks may need to be modified to use the appropriate module instead of those for `apt`. Similarly appropriate commands will need to be modified for other operating systems as well.

About task selection

There is no particular set of tasks I selected to use in this tutorial. In fact most were selected rather randomly based on functionalities I wanted to demonstrate. The two server installation is primarily to demonstrate how role based tasks can also be performed.

Install Ansible

First make sure you have python. I've used python 2.7. Ansible does not work with python 3.x. Now create a virtualenv for this exercise. We shall name it `floo`

```
$ sudo apt-get install python-virtualenv # for debian based systems
$ virtualenv-2.7 --no-site-packages ~/pyenv/floo
$ . ~/pyenv/floo/bin/activate
$ pip install ansible
```

```
$ pip install passlib
```

Create virtualboxes (preferably using Vagrantfile)

You can choose to perform either the vagrant setup (which is explained in detail) or the alternative setup described later in terms of what the setup must achieve.

Vagrant setup

Install vagrant

http://docs.vagrantup.com/v2/installation/index.html

(You might need to uninstall older version of vagrant if already installed, see the uninstallation subsection)

We shall be using the precise32 vagrant box.

Create a project directory for this project (I'm calling it `floo`)

```
$ mkdir floo
$ cd floo
$ vagrant init precise32 http://files.vagrantup.com/precise32.box
```

Replace the Vagrantfile that was generated with the following contents

```
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "precise32"
  config.vm.box_url = "http://files.vagrantup.com/precise32.box"
  config.vm.provision "shell", inline: "echo Hello"

  config.vm.define "web" do |web|
    web.vm.box = "precise32"
    web.vm.hostname = "web"
    web.vm.network "private_network", ip: "192.168.101.11"
  end

  config.vm.define "db" do |db|
    db.vm.box = "precise32"
    db.vm.hostname = "db"
    db.vm.network "private_network", ip: "192.168.101.12"
  end
end
```

Start vagrant servers as shown in the commands below. Preferably in separate windows / panes and keep the panes open.

Note: The first vagrant up command will download the precise32 (ubuntu 12.04 32 bit) box and will take a fair amount of time and bandwidth (> 300MB). Once done, it will reuse it for other boxes so the subsequent commands should go through quickly.

The first step will trigger a download of 300+MB if you haven't downloaded the vagrant box earlier.

Now create two different terminals (one for each of your vagrant boxes) and in each of them execute the following commands (Repeat: You'll have to run these twice, once in each terminal)

```
# Make sure the current directory is the floo project directory where
# the current directory contains Vagrantfile

$ vagrant up web
$ vagrant ssh web
```

Please update your package indices and update the packages for both the virtualboxes

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

To enable a passwordless sudo, on both the machines do,

```
$ sudo visudo
```

And add the following line near the end of the file

```
%vagrant ALL=(ALL:ALL) NOPASSWD: ALL
```

Alternative setup

Alternatively you may provision two virtual machines either on the cloud or on your desktop using a virtualisation solution like virtualbox or Xen etc.

The requirements of this setup are as follows :

- You should have two virtual boxes apart from your host. One called web and another called db for the rest of this tutorial.
- Your host should be able to communicate and SSH into both the virtual boxes. It is highly recommended that this be done using private key authentication, though that is not mandatory.
- The userid used (`vagrant` for the purposes of this tutorial) to SSH into both the VMs should have privileges to do passwordless sudo.

Also please update your package indices and update the packages for both the virtualboxes

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Git setup

Make sure you have git installed on your host.

```
$ sudo apt-get install git
```

Now checkout the git source using the following command that you can use as a reference.

```
git clone https://dnene@bitbucket.org/dnene/floo.git floo-ref
```

Create IP address mappings in /etc/hosts

Please map the ip addresses of the VMs in your /etc/hosts file (unless these already have a public DNS

entry). The exact IP addresses may be different depending upon how you configured your VMs. The ones below assume you used the sample Vagrantfile

```
192.168.101.11 web
192.168.101.12 db
```

Create inventory file

Note: I have added the file `insecure_private_key` distributed with vagrant to the root directory of the project. Please ensure that you change the mode to `600` ie. readable by owner only.

```
$ cp ../floo-ref/insecure_private_key .
$ chmod 600 insecure_private_key
```

The inventory file is called `hosts` in this tutorial

```
[vms]
web hostname=192.168.101.11 ansible_ssh_user=vagrant
ansible_ssh_private_key_file=insecure_private_key
db hostname=192.168.101.12 ansible_ssh_user=vagrant
ansible_ssh_private_key_file=insecure_private_key

[webservers]
web

[dbservers]
db

[_vms]
web
db
```

Also create the following file `.gitignore` to ensure some of the files do not get committed to the git repository where you may choose to store your ansible project

```
.*.swp
.vagrant
group_vars/_*
```

Now run the following commands to test basic connectivity

```
ansible -i hosts -a hostname web
ansible -i hosts -a hostname db

ansible -i hosts -a "sudo ifconfig" web
ansible -i hosts -a "sudo ifconfig" db
```

If they print the hostname and the output of the ifconfig, then basic connectivity between ansible and the machines has been setup.

A minimal ansible playbook. Create a group

Playbooks are the way ansible organises the scripts. We shall create a minimal ansible playbook. Over the next few steps we shall want to create a new user and a group and give the user appropriate privileges and

ability to login. To that end we shall create a variables file for the group `common`

In this case I shall be creating the user `newuser` with the group `newgroup`. This activity shall be done for both the nodes `web` and `db`.

The user, group will need to be created across all the machines. So we shall configure it under the role `common`. Create a file `group_vars/vms` with the following content.

```
group: newuser
user: newuser
user_name: "New User"
```

Create a file `site.yml` in the project directory as follows

```
---
- include: webservers.yml
- include: dbservers.yml
```

Now a file `webservers.yml` in the project directory with the following contents. Also create another file `dbservers.yml` with similar content. Only the value for the `hosts` tag should be set to `dbservers`

```
- hosts: webservers
  sudo: yes
  roles:
  - common
```

Create a file `roles/common/tasks/main.yml` as follows.

The script creates a group as defined by the var `group`.

```
- name: Create {{ group }} group
  group: name={{ group }} state=present
```

Note the following :

- group level variables can be declared in group specific files ie. `groupvars/`
- A playbook is essentially a set of tasks
- A site wide policy file (`site.yml`, also referred to as a master playbook) can include other specific playbooks
- A playbook declares the hosts and roles it is applicable for
- Role specific tasks can be configured in `roles//tasks/main.yml`
- Each task consists of a name and an action. Each name is just a text message for echoing / debugging and has no other value. The action typically uses a module - in this case `group`
- Each action specification is accompanied by additional name value pairs.
- Some of these will be used to implement idempotence (eg. in this case, "state=present"). Thus the group will not get created if it is already present. This allows the playbooks to be run repeatedly.

Now run the ansible playbook as follows

```
$ ansible-playbook -i hosts site.yml
```

Remember this command. We shall be running this command repeatedly every time we add a small bit of

functionality.

If you now look at the `web` and `db` servers, you will find that a unix group `newuser` has been created.

Create user

We shall now create a new user. However when creating a new user we need to specify the password. It is preferred to keep the password encrypted for appropriate security. To generate the encrypted password please create a small python program `pyhelpers/crypt.py` as follows

```
from passlib.hash import sha512_crypt
import sys
hash = sha512_crypt.encrypt(sys.argv[1])
print(hash)
```

Now run the python program giving it the password as an argument. eg.

```
$ python pyhelpers/crypt.py secret
```

Copy paste the output of the program into `group_vars/vms` as follows

```
password: $6$rounds=60000$NW8pUrwx.......
```

Also add the following lines to `roles/common/tasks/main.yml`

```
- name: Create {{ user }} account
  user:
    name={{ user }}
    comment="{{ user_name }}"
    group={{ user }}
    createhome=yes
    home=/home/{{ user }}
    shell=/bin/bash
    password={{ password }}
    state=present
```

Note the following

- In this case we have provided the password as an encrypted value
- In this case we have provided the name value pairs to user over multiple lines

Run the playbook once again.

```
$ ansible-playbook -i hosts site.yml
```

User ssh initialisation

We shall perform some more user initialisation tasks. Add the following code snippets to `roles/common/tasks/main.yml`.

```
- name: ensure ssh directory exists for {{ user }}
  file: dest=/home/{{ user }}/.ssh owner={{ user }} group={{ group }}
state=directory
```

This will create a .ssh directory in the new user's home directory if one does not exist and set the appropriate

permissions on it.

```
- name: ensure public key is in authorized_keys
  lineinfile:
    dest=/home/{{ user }}/.ssh/authorized_keys
    state=present
    line="{{ lookup('file','~/.ssh/id_dsa.pub') }}"
    insertafter=EOF
    create=yes
    owner={{ user }}
    group={{ group }}
    mode=600
    regexp=^WILL_NOT_MATCH
```

This will copy the `id_dsa.pub` for the current logged in user on the host as the authorised_key. Note change the file name from id_dsa.pub if your public key file has a different name. Note the usage of the `lineinfile` module. This is frequently used for many tasks.

```
- name: update ssh parameters
  lineinfile:
    dest=/etc/ssh/sshd_config
    state=present
    regexp=^{{ item.key }}
    line="{{ item.key }} {{ item.value }}"
    insertafter=EOF
  with_items:
    - { key: 'PermitRootLogin', value: 'no' }
    - { key: 'LoginGraceTime', value: '20' }
    - { key: 'X11Forwarding', value: 'no' }
    - { key: 'ClientAliveInterval', value: '30' }
    - { key: 'ClientAliveCountMax', value: '1000' }
  notify:
    - restart ssh
```

This will change the various ssh parameters. Note here we are using the `lineinfile` module with a sequence of key value pairs simultaneously. Note the `notify` section. This informs ansible that ssh will eventually need to be restarted. For that we need to install a handler. Add the following section to the end of both `webservers.yml` and `dbservers.yml`

```
handlers:
- name: restart ssh
  service: name=ssh state=restarted
```

Note that the `handlers` tag is at the same level and in continuation to the `hosts`, `sudo` and `roles` tags.

Also note service action. You specify the name of the service and for purposes of idempotency the desired state. Ansible figures out the command to be run.

Housekeeping tasks

We shall now perform some basic housekeeping. Add the following to `roles/common/tasks/main.yml`

```
- name: Set timezone to UTC
```

```
  action: shell echo Etc/UTC > /etc/timezone

- name: Set localtime to UTC
  file: src=/usr/share/zoneinfo/Etc/UTC dest=/etc/localtime

- name: Reconfigure tzdata
  action: command dpkg-reconfigure -f noninteractive tzdata
```

Note that the `action` module can be used to run simple shell commands.

Package Management

Again add the following to `roles/common/tasks/main.yml`. These commands will update the apt keys (if any have expired), update the apt-cache, upgrade the packages and finally install two packages `heirloom-mailx` and `libpam-cracklib`

```
- name: Update apt keys
  command: apt-key update

- name: Update apt cache
  apt: update_cache=yes

- name: Upgrade apt cache
  apt: upgrade=yes

- name: Install additional packages
  apt: package=$item state=present
  with_items:
    - heirloom-mailx
    - libpam-cracklib
```

Additional password rules

We shall now use libpam-cracklib to configure additional password strength rules

Note that the current contents of the file `/etc/pam.d/common-password` has a line as follows

```
password        requisite                       pam_deny.so
```

We shall modify it to strengthen the password strength settings.

```
- name: Increase password strength
  lineinfile:
    dest=/etc/pam.d/common-password
    state=present
    line='password        requisite                       pam_cracklib.so
retry=3 minlen=8 difok=3 minclass=4'
    regexp="^password.*requisite.*pam_cracklib.so"
```

At the end of this exercise try logging in (`su`) as `newuser` and change your password. You will see that you are now required to enter stronger passwords than earlier

SMTP configuration using templates

We shall configure the mailx package using a template.

First create a template `roles/common/templates/os/etc/nail.rc` with the following content

```
{% if use_starttls == "true" %}
set smtp-use-starttls
{% endif %}
set from={{ from_address }}
set smtp={{ smtp_server }}
set smtp-auth-user={{ smtp_auth_user }}
set auth-login={{ auth_login }}
set smtp-auth-password={{ smtp_auth_password }}
```

This template is also written using `jinja2`. Here you can see the use of conditionals and variable substitution.

We shall of course need to provide the values for these variables. Append the following to the end of `group_vars/vms`

```
use_starttls: true
from_address: sender@from.com
smtp_server: mail.from.com
smtp_auth_user: sender
auth_login: userid
```

Note that the template also expects a variable to be set ie. `smtp_auth_password`. But setting it in plain text in such data files could be insecure, especially since you are likely to be pushing these to common source code repositories. While there is no clean support for it yet, I suggest the following approach.

Create a file `group_vars/_vms` with the following content.

```
smtp_auth_password: __secret__
```

Note, that we've already declared both web and db to belong to the `vms` and the `__vms` group. Also, the `.gitignore` file specifies that files matching the pattern `group_vars/_*` should not get added to the git repository. Thus any values you set in `group_vars/_vms` will not get pushed out to your git repo.

The values are dummy values. At this stage we are not actually going to use the configuration to send email. So don't worry about what values you set.

Finally add the following tasks, again to `roles/common/tasks/main.yml`.

```
- name: Write mailx template
  template: src=os/etc/nail.rc dest=/etc/nail.rc
```

Role specific configuration (Database Server)

We shall now configure a role specific task. This task should be applied only to the database server `db`. So far we have described only one role, ie. `common`. We shall now define two more. To explicitly disambiguate the roles from the groups `webservers`/`dbservers`, we shall call the roles `webtier` and `dbtier`.

We shall install the mysql-server package only for the dbtier.

Create the file `roles/dbtier/tasks/main.yml` and add the following task to it.

First let us add the role `dbtier` to `dbservers.yml`

The updated file `dbservers.yml` should now look as follows.

```
- hosts: dbservers
  sudo: yes
  roles:
  - common
  - dbtier
  handlers:
  - name: restart ssh
    service: name=ssh state=restarted

- name: Install mysql server packages
  apt: package=$item state=present
  with_items:
    - mysql-server
```

When you run the playbook this time, note that only the `db` server will have the mysql-server package installed

Apache installation and configuration

Let us now do apache installation and configuration on the web server.

First modify `webservers.yml` to the following state.

```
- hosts: webservers
  sudo: yes
  roles:
  - common
  - webtier
  handlers:
  - name: restart ssh
    service: name=ssh state=restarted
  - name: restart apache
    service: name=apache2 state=restarted
```

Note that we have added the webtier role and also added a handler to restart apache service eventually.

We shall be creating an additional virtual host on the apache server that we shall be installing. This requires a virtual host file to be installed. Create the file `roles/webtier/templates/os/etc/apache2/sites-available/support` with the follwing content.

```
<VirtualHost support:80>
    ServerAdmin webmaster@localhost
    ServerName support

    DocumentRoot /var/www/support
    <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>
```

```
    <Directory /var/www/support/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/support_error.log

    # Possible values include: debug, info, notice, warn, error, crit,
    # alert, emerg.
    LogLevel warn

    CustomLog ${APACHE_LOG_DIR}/support_access.log combined
</VirtualHost>
```

We shall also want to deploy a website to the new virtual host that we shall be creating (for now just a single file ie. index.html). Hence create the index.file template at `roles/webtier/templates/os/var/www/support/index.html`

```
<html>
    <head>
        <title>Support site</title>
    </head>
    <body>
        <h1>Support site</h1>
        This is the support site
    </body>
</html>
```

Although we did not take advantage of templating, note that these are indeed templates and thus the power of jinja2 can be used to do appropriate parameterisation and template processing as necessary.

Finally create the tasks file for webtier as follows ie. file `roles/webtier/tasks/main.yml` with the following content.

```
- name: Install apache packages
  apt: package=$item state=present
  with_items:
    - apache2
    - python
    - libapache2-mod-wsgi
    - libmysqlclient18
    - libmysqlclient-dev
    - mysql-client


- name: Disable unused apache modules
  command: a2dismod autoindex cgi env

- name: copy apache vhost file
  template: src=os/etc/apache2/sites-available/support
```

```
            dest=/etc/apache2/sites-available/support
            owner=root
            group=root
            mode=0644

- name: make support directory
  file: dest=/var/www/support owner=www-data group=www-data state=directory

- name: copy support website
  template: src=os/var/www/support/index.html
            dest=/var/www/support/index.html
            owner=www-data
            group=www-data

- name: enable apache2 site
  command: /usr/sbin/a2ensite support
  notify:
    - restart apache
```

You may want to add an entry to `/etc/hosts` to point the web ip address to another name called "support" and then just goto *http://support* in your browser. You should be able to see your new site