## 1. Directory Layout

This is the directory layout of this repository with explanation.

production.ini # inventory file for production stage

development.ini # inventory file for development stage

test.ini # inventory file for test stage

vpass # ansible-vault password file

# This file should not be committed into the repository

# therefore file is in ignored by git

group\_vars/

all/ # variables under this directory belongs all the groups

apt.yml # ansible-apt role variable file for all groups

webservers/ # here we assign variables to webservers groups

apt.yml # Each file will correspond to a role i.e. apt.yml

nginx.yml # ""

postgresql/ # here we assign variables to postgresql groups

postgresql.yml # Each file will correspond to a role i.e. postgresql

postgresql-password.yml # Encrypted password file

plays/

ansible.cfg # Ansible.cfg file that holds all ansible config

webservers.yml # playbook for webserver tier

postgresql.yml # playbook for postgresql tier

roles/

roles\_requirements.yml# All the information about the roles

external # All the roles that are in git or ansible galaxy

# Roles that are in roles\_requirements.yml file will be downloaded into this directory

internal # All the roles that are not public

extension/

setup # All the setup files for updating roles and ansible dependencies

## 2. How to Manage Roles

It is a bad habit to manage the roles that are developed by other developers, in your git repository manually. It is also important to separate them so that you can distinguish those that are external and can be updated vs those that are internal. Therefore, you can use ansible-galaxy for installing the roles you need, at the location you need, by simply defining them in the roles\_requirements.yml:

---

- src: ANXS.build-essential

version: "v1.0.1"

Roles can be downloaded/updated with this command:

./extensions/setup/role\_update.sh

This command will delete all external roles and download everything from scratch. It is a good practice, as this will not allow you to make changes in the roles.

## 3. Keep your plays simple

If you want to take the advantage of the roles, you have to keep your plays simple. Therefore do not add any tasks in your main play. Your play should only consist of the list of roles that it depends on. Here is an example:

---

- name: postgresql.yml | All roles

hosts: postgresql

sudo: True

roles:

- { role: common, tags: ["common"] }

- { role: ANXS.postgresql, tags: ["postgresql"] }

As you can see there are also no variables in this play, you can use variables in many different ways in ansible, and to keep it simple and easier to maintain do not use variables in plays. Furthermore, use tags, they give wonderful control over role execution.

## 4. Stages

Most likely you will need different stages (e.g. test, development, production) for the product you are either developing or helping to develop. A good way to manage different stages is to have multiple inventory files. As you can see in this repository, there are three inventory files. Each stage you have must be identical as possible, that also means, you should try to use few as possible host variables. It is best to not use at all.

## 5. Variables

Variables are wonderful, that allows you to use all this existing code by just setting some values. Ansible offers many different ways to use variables. However, soon as your project starts to get bigger, and more you spread variables here and there, more problems you will encounter. Therefore it is good practice to keep all your variables in one place, and this place happen to be group\_vars. They are not host dependent, so it will help you to have a better staging environment as well. Furthermore, if you have internal roles that you have developed, keep the variables out of them as well, so you can reuse them easily.

## 6. Name consistency

If you want to maintain your code, keep the name consistency between your plays, inventories, roles and group variables. Use the name of the roles to separate different variables in each group. For instance, if you are using the role nginx under webservers play, variables that belong to nginx should be located under group\_vars/webservers/nginx.yml. What this effectively means is that group\_vars supports directory and every file inside the group will be loaded. You can, of course, put all of them in a single file as well, but this is messy, therefore don't do it.

## 7. Encrypting Passwords and Certificates

It is most likely that you will have a password or certificates in your repository. It is not a good practise to put them in a repository as plain text. You can use [ansible-vault](http://docs.ansible.com/playbooks_vault.html) to encrypt sensitive data. You can refer to [postgresql-password.yml](https://github.com/enginyoyen/ansible-best-practises/blob/master/group_vars/postgresql/postgresql-password.yml) in group variables to see the encrypted file and [postgresql-password-plain.yml](https://github.com/enginyoyen/ansible-best-practises/blob/master/group_vars/postgresql/postgresql-password-plain.yml) to see the plain text file, commented out. To decrypt the file, you need the vault password, which you can place in your root directory but it MUST NOT be committed to your git repository. You should share the password with you coworkers with some other method than committing to git a repo.

There is also [git-crypt](https://github.com/AGWA/git-crypt) that allow you to work with a key or GPG. Its more transparent on daily work than ansible-vault

## 8. Project Setup

As it should be very easy to set-up the work environment, all required packages that ansible needs, as well as ansible should be installed very easily. This will allow newcomers or developers to start using ansible project very fast and easy. Therefore, python\_requirements.txt file is located at:

extensions/setup/python\_requirements.txt

This structure will help you to keep your dependencies in a single place, as well as making it easier to install everything including ansible. All you have to do is to execute the setup file:

./extensions/setup/setup.sh

# Running the Code

Code in this repo is functional and test it. To run it, you need to install ansible and all the dependencies. You can do this simply by executing:

./extensions/setup/setup.sh

* If you already have ansible, and you do not want to go through the installation simply create a vpass text file in the root directory and add the secret code (123456)
* To install roles execure the role\_update.sh which will download all the roles

./extensions/setup/role\_update.sh

* Go to plays directory and the execute and do not forget to change the host address in the development.ini

ansible-playbook -i ../development.ini webservers.yml

Encryption of password file:

Create directory under /fs01/ansible/inv/stC01/group\_vars/all/secrets/secrets.yml

Add admin\_password: devops123

Now encrypt the this file with a vault password

My vaultpassword is : Perf0rce!2 write this is in a file under my home directory as /home/devops/.vpass

And run command

ansible-vault encrypt /fs01/ansible/inv/stC01/group\_vars/all/secrets/secrets.yml –vault-password-file ~/.vpass

This was still asking for password, so I added the defaults to ansible.cfg as

[defaults]

vault\_password\_file = /home/devops/.vpass

now encrypt worked perfect, now check and see the contents in secrets.yml is changed

now in inventory : inv/stC01/group\_vars/all/config/wls\_props.yml change admin\_password={{ admin\_password }} - so nobody know the password and chekin to GIT, but not the vault password.

now run

ansible-playbook –I inv/stC01 playboos/config-ics.yml ,this works fine now