ILG

Insight GNU/Linux Group

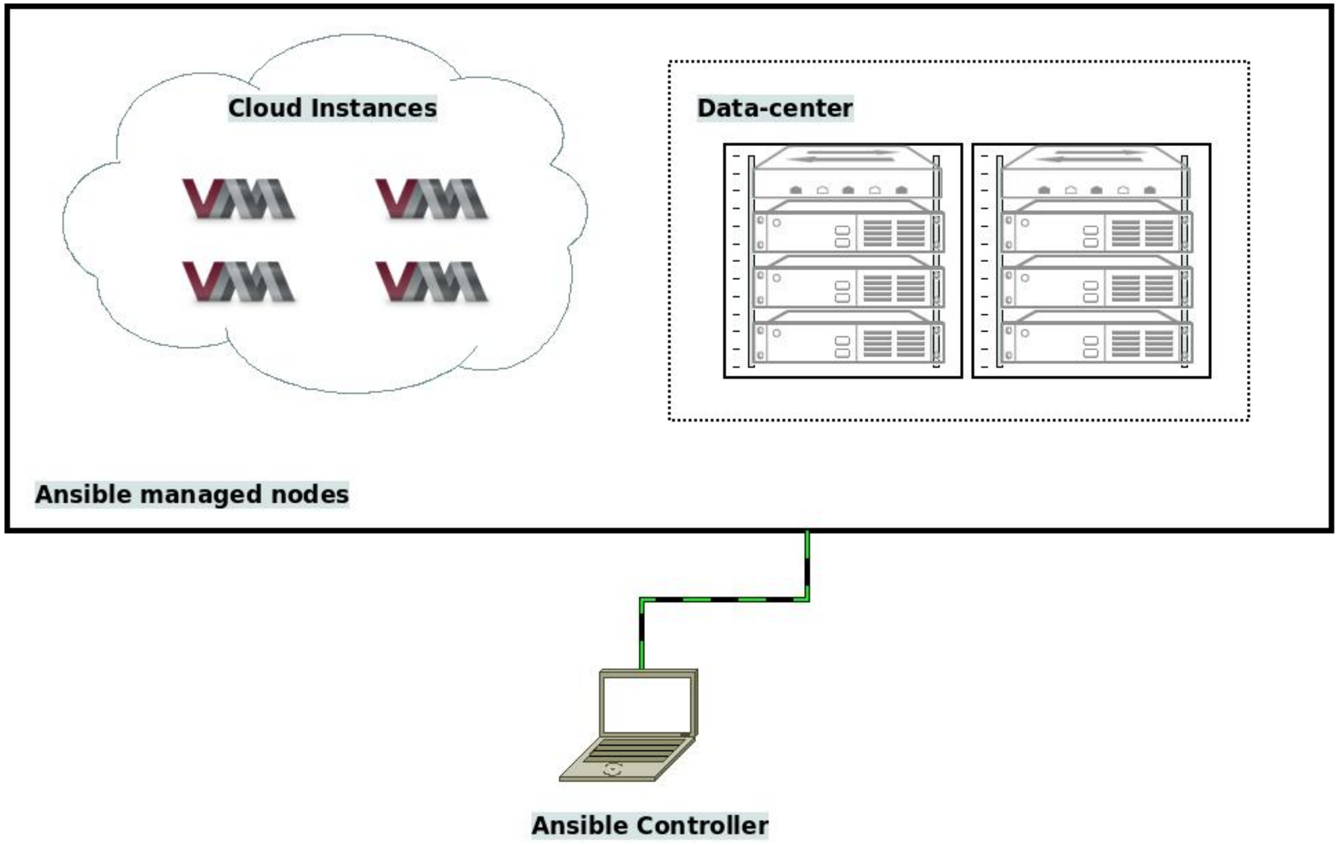
Reinventing the way you,
Think,
Learn,
Work

SIMPLE. POWERFUL. AGENTLESS.

ANSIBLE

- Configuration management.
- Deploy software.
- Continuous delivery.
- Cloud ready (AWS, Openstack, Vmware, Azure)
- Automate in a language that approches plain english.
- Uses SSH, no agents to install.
- Free Software (GNU GPL v3)

Ansible’s usage architecture



Ansible controller:
It is a system that has Ansible installed and from where you execute your playbooks.

Ansible Managed nodes:
These are systems which have SSH server installed and are included in your controller’s inventory file.
These systems could be instances on any **Cloud infrastructure** or **Remote Data-center servers.**

Conventions

- Any command that has a # at the beginning must be run as root user. For example:

-
  ```
  # passwd
  ```
- And command that has $ at the beginning must be run as standard user.

-
  ```
  $ ls
  ```

-
  ```
  $ sudo ls
  ```
- But do not include # or $ in commands, they are symbols you would see on command prompt.

- We will be using two systems: ***controller*** and ***node1***
  - ***controller:*** is our system on which we will have ansible installed
  - ***node1:*** is our system on which we will manage remotely and run our automation playbooks

Configuration: On Managed Nodes

- Creating the Sudo user for ansible
  - Create a new user and set it's password with the commands
    - `# useradd ansible`
    - `# passwd ansible`
  - Install the package '*sudo*' if it is not already installed.
  - Create the file "*/etc/sudoers.d/ansible*"
    - `# vim /etc/sudoers.d/ansible`
  - add the following line to it:
    - `ansible   ALL=(ALL:ALL) NOPASSWD:ALL`

Configuration: On controller node

- Create a new user ‘ansible’ on your controller node and set its password.
  -
    ```
    # useradd ansible
    # passwd ansible
    ```
- Login or switch to user ‘ansible’ on your controller node.
- Generate ssh key for the user ‘ansible’
  -
    ```
    $ ssh-keygen
    ```
- Add host entries in your controller node for your nodes/server
  - Edit */etc/hosts*

    ```
    # vim /etc/hosts
    ```
  - Add the following host entries

    ```
    192.168.1.210   controller
    192.168.1.211   node1
    ```
- Copy the key to managed node for passwordless ssh
  -
    ```
    $ ssh-copy-id -i “path/to/key/” ansible@node1
    ```

Installation

- Ansible only needs to be installed on the controller node.
- There are two methods to install Ansible:
  - It can be installed via the native package manager of the distribution
  - Python Package Index’s pip package manager.
- Pre-requisites for Managed node machines
  - The following packages are required for ansible to manage nodes remotely
  - python2 – Python version 2.X
  - python-simplejson – Python module
  - libselinux-python – For selinux support

Installation Method 1: Package Manager

- On Debian and Debian-based distros run the following command as root user

```
# apt-get install ansible -y
```

- On CentOS and RHEL,
  - enable the epel repository and then install ansible as root user

```
# yum install epel-release -y && yum install ansible -y
```

- On OpenSuse and Suse as root user

```
# zypper install ansible
```

Installation Method 2: pip package manager

- Using the distro's native package manager install the following packages
  - python-pip or python2-pip
- Install ansible using pip
  - Run the following command as standard user

  -
    ```
    $ pip install ansible
    ```
  - This installs pip only for the current user.
  - Edit the file *<u>/home/ansible/.bashrc</u>*

    ```
    $ vim /home/ansible/.bashrc
    ```
  - and the below configuration to it.

    ```
       export PATH="${HOME}/.local/bin:${PATH}"
    ```
  - Source the *<u>.bashrc</u>* file

    ```
    $ source /home/ansible/.bashrc
    ```

Working directory on controller node

- User *ansible*'s home directory (/home/ansible) is your working directory.
- This directory is where you will create configuration file, inventory, playbook and roles.
- Create the configuration file "/home/ansible/*.ansible.cfg*"
  - Add the following lines to it

```
$ vim /home/ansible/.ansible.cfg
```

This sets the location of your default inventory file to */home/ansible/hosts*

```
[defaults]
inventory       = $HOME/hosts
```

Inventory

- Create an inventory file named “*hosts*” in the home directory.

  -
    ```
    $ vim /home/ansible/hosts
    ```

  -
    ```
    # These are your nodes/servers
    controller
    node1

    # and now with node1 as a member of a group inventory
    [wordpress-server]
    node1
    ```
- Check your host is alive.

  -
    ```
    ansible all -m ping
    ```

Inventory: other examples

- If you have hosts that run on non-standard SSH ports you can put the port number after the hostname with a colon

  -
    ```
    nodeweb:5309
    ```
- If you have a lot of hosts following similar patterns you can do this rather than listing each hostname:

  -
    ```
    [webservers]
    www[01:50].example.com
    ```
- You can also define alphabetic ranges:

  -
    ```
    [databases]
    db-[a:f].example.com
    ```
- You can also select the connection user on a per host basis:

  -
    ```
    [databases]
    other1.example.com     ansible_user=mpdehaan
    other2.example.com     ansible_user=mdehaan
    ```

Ad-hoc commands

- On any given day a system administrator performs the following tasks:
  - Apply patches and updates via yum, apt, and other package managers.
  - Check resource usage (disk space, memory, CPU, swap space, network).
  - Manage system users and groups.
  - Manage DNS settings, hosts files, etc.
  - Copy files to and from servers.
  - Deploy applications or run application maintenance.
  - Reboot servers.
  - Manage cron jobs.
- It would be hectic to login to each of the servers and performs these tasks, so sysadmins would usually have an automated system of scripts to get these done.
- And we are going to use Ansible to perform some of these tasks.

Parallelism in ansible

- Let's us see if our nodes have internet access,

  -
    ```
    ansible all -a "ping -c 2 google.com"
    ```

```
node2 | UNREACHABLE! => {
"changed": false,
"msg": "Failed to connect to the host via ssh.",
"unreachable": true
}
controller | SUCCESS | rc=0 >>
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 1.882/1.921/1.960/0.039 ms

node1 | SUCCESS | rc=0 >>
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 2.429/2.440/2.452/0.050 ms
```

Parallelism

- Ansible will run command on all nodes, and return the results (if Ansible can't reach one a server, it will show an error for that server, but continue running the command on the others).
- Ansible runs commands in parallel using multiple forks to complete commands more quickly.
- This may not be much quicker when managing a few servers, but you will notice this speed while managing 5-10 servers
- Adding the argument –f 1 to the command would run sequencially on each node.

```
ansible all -a "ping -c 3 google.com" -f1
```

Managing user and groups

- creating a group named *“appadmin”*

```
ansible node1 --become --become-method=sudo -m group -a "name=appadmin state=present"
```

```
node1 | SUCCESS => {
    "changed": true,
    "gid": 504,
    "name": "appadmin",
    "state": "present",
    "system": false
}
```

Managing user and groups

- Creating an user *“appuser”* and making it a member of the group *“appadmin”*

```
ansible node1 --become –become-method=sudo \
-m user -a "name=appuser group=appadmin createhome=yes password=\
$6\$9N9wtiyi6JH$7tD5G6fx.yBKVaafeVP5IbOB9iDnTLWW1yrq4YqHQ8zZhYOTKFq0WbRHQ91A2s5RAgAeT94.T5Lkq11
1dWTzq1"
```

The password field is in hash which can be generated using *“mkpasswd –m sha-512”*

or use ***‘openssl passwd -6’***

```
node1 | SUCCESS => {
    "changed": true,
    "createhome": true,
    "group": 504,
    "home": "/home/appuser",
    "name": "appuser",
    "password": "NOT_LOGGING_PASSWORD",
    "shell": "/bin/bash",
    "state": "present",
    "system": false,
    "uid": 510
}
```

Managing user and groups

– Removing a user

```
ansible node1 --become --become-method=sudo -m user -a "name=appuser state=absent remove=yes"
```

– Removing a group

```
ansible node1 --become --become-method=sudo -m group -a "name=appadmin state=absent"
```

Copying, fetching files and directories

- You would probably use scp or rsync for copying files to and from remote server.
- Most file copy operations can completed using Ansible's copy module.

```
ansible node1  -m copy -a "src=/etc/skel dest=/tmp/skel"
```

```
node1 | SUCCESS => {
    "changed": true,
    "dest": "/tmp/skel",
    "src": "/etc/skel"
    }
```

Copying, fetching files and directories

- The earlier command while copying created *"/tmp/skel/skel"* in the remote node and copied the contents from *"/etc/skel"* to *"/tmp/skel/skel"*
- But if we wanted the directory contents to be copied to *"/tmp/skel"* we should add a trailing slash (/) to the src directory *"/etc/skel/"* so our command would become this.

```
ansible node1  -m copy -a "src=/etc/skel/ dest=/tmp/skel"
```

Creating files and directories

We can use the file module to create file and directory, manage file permissions and ownership, create symbolic links,

- Creating file
- *path* argument has aliases *dest* and *name*

```
ansible node1  -m file -a "path=/tmp/test-file  state=touch"
```

Creating files and directories

- creating directory with alias dest

```
ansible node1  -m file -a "dest=/tmp/test state=directory"
```

- Create symbolic link

```
ansible node1  -m file -a "src=/tmp/test-file dest=/tmp/file-symlink state=link"
```

- Deleting file or directory

```
ansible node1  -m file -a "dest=/tmp/file-symlink state=absent"
```

File permission and ownership

- Changing ownership of file

```
ansible node1 --become --become-method=sudo -m file -a "path=/tmp/test-file owner=appuser
group=appadmin"
```

- Changing ownership of directory

```
ansible node1 --become --become-method=sudo -m file -a "path=/tmp/test-dir state=directory
owner=appuser group=appadmin recurse=yes"
```

- *recurse=yes* applies permission to all files and subdirectories in the directory

File permission and ownership

- Changing permission of a directories,
  - permission mode can be specified in octal mode or symbolic form.

```
ansible node1 --become -m file  -a "path=/tmp/test-file mode=u+rwx,g=,o="

ansible node1 --become -m file  -a "path=/tmp/test-dir mode=750 recurse=yes"
```

Adhoc commands (Cont.)

Let us run a few more adhoc commands and deploy a webserver with a little content.

- Installing Webserver

```
ansible node1 -m yum -a "name=httpd state=installed" --become --become-method=sudo
```

```
node1 | SUCCESS => {
    "changed": false,
    "msg": "",
    "rc": 0,
    "results": [
        "httpd-2.2.15-54.el6.centos.x86_64 providing httpd is already installed"
    ]
}
```

Adhoc commands (Cont.)

- Copying website content

```
ansible node1 -m copy -a "src=testpages/website.html dest=/var/www/html/index.html" --become --become-method=sudo
```

```
node1 | SUCCESS => {
    "changed": false,
    "checksum": "ae44c75f537a2141b0d25160c2d1c6e5a4ea2ddb",
    "dest": "/var/www/html/index.html",
    "gid": 0,
    "group": "root",
    "mode": "0644",
    "owner": "root",
    "path": "/var/www/html/index.html",
    "size": 151,
    "state": "file",
    "uid": 0
}
```

Now you can check the webserver by accessing it via IP address on your web browser.

Adhoc commands (Cont.)

- A few more adhoc commands on Installing and creating MySQL databases

1. Installing MySQL server and ansible support.

```
ansible node1 -m yum -a "name=mysql-server state=installed" --become –become-method=sudo

ansible node1 -m yum -a "name=MySQL-python state=installed" --become –become-method=sudo
```

2. Creating database

```
ansible node1 -m mysql_db -a "name=db-test1 state=present" --become --become-method=sudo
```

3. Creating user for database

```
ansible node1 -m mysql_user -a "name=ilg password=ilg007 priv=db-test1.*:ALL host='%'
state=present" --become --become-method=sudo
```

Playbooks

- Playbooks are Ansible's configuration, deployment, and orchestration language.
- They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process.
- Playbooks are designed to be human-readable and are developed in a basic text language.

Playbooks

- Playbooks are written in YAML format
- At a basic level, playbooks can be used to manage configurations of and deployments to remote machines.
- At a more advanced level, they can sequence multi-tier rollouts involving rolling updates, and can delegate actions to other hosts, interacting with monitoring servers and load balancers along the way.

Playbooks: Tasks

- Each play contains a list of tasks.
- Tasks are executed in order, one at a time, against all machines matched by the host pattern, before moving on to the next task.
- Within a play, all hosts are going to get the same task directives.
- It is the purpose of a play to map a selection of hosts to tasks.

Playbooks: Handlers

- Handlers are tasks with globally unique name and are notified by notifiers.
- If nothing notifies a handler, it will not run.
- Regardless of how many tasks notify a handler, it will run only once, after all of the tasks complete in a particular play.
- The things listed in the notify section of a task are called handlers. "*notify: restart mysql*", "*restart mysql*" is the name of the handler.

Playbooks: Modules

- Ansible ships with a number of modules (called the 'module library') that can be executed directly on remote hosts or through Playbooks.
- Arguments used by modules can be viewed using

```
ansible-doc -l
ansible-doc file
```

```
## lists all modules
## will display arguments supported by the module "file"
```

- These Modules are idempotent, that is, running a module multiple times in a sequence should have the same effect as running it just once.

Playbooks: Idempotency

- Idempotency is to have a module check whether its desired final state has already been achieved, and if that state has been achieved, to exit without performing any actions.
  - For example, the *copy* module checks this by confirming whether the source and destination files have same checksum.
- If all the modules a playbook uses are idempotent, then the playbook itself is likely to be idempotent, so re-running the playbook should be safe.

Playbooks: Variables

- You can have some templates for configuration files that are mostly the same, but require slightly different entries for different servers based on variables like database name, database user, IP Addressess, etc.
- Valid variable names
  - Variable names should be letters, numbers, and underscores. Variables should always start with a letter.
  - foo_port is a great variable. foo5 is fine too.
  - foo-port, foo port, foo.port and 12 are not valid variable names.

Playbooks: Variables

- Playbooks (YAML) supports dictionaries which map keys to values. For instance:

```
mysql:
     user: ilg
     pass: ilg007
```

  - You can then reference a specific field in the dictionary using either bracket notation or dot notation:

```
mysql['user']
mysql.pass
```

Playbooks: Variable file

- Using one or more included variable files cleans up your main playbook file, and lets you organize all your configurable variables in one place
- You can do this by using an external variables file, or files, in your playbook just like this:

```
---
- host anode53
    vars_files:
    - vars.yml
```

- The contents of each variables file is a simple YAML dictionary, like this:

```
---
somevar: somevalue
password: magic
```

Playbook: Templates

Templates are files that you copy to a remote node, but they include variables which would be replaced with their specified values according to each node as written in a playbook.

Deploying a mysql database and a website executing playbook

- So we are going to configure a simple web page with a database connection.
- Create a file *playbook.yml*

```
---
- hosts: anode53
  remote_user: ansible
  become: true
  become_method: sudo
  vars:
  - mysql:
      user: ilg
      pass: ilg007
```

Executing Playbook (playbook.yml)

```
  tasks:
  - name: Install MySQL package
    yum: name=mysql-server state=installed
  - name: mysql ansible support
    yum: name=MySQL-python state=installed
  - name: Start MariaDB
    service: name=mysqld state=started enabled=yes
  - name: Create Database
    mysql_db: name=classics state=present
  - name: Create DB User
  mysql_user: name={{ mysql.user }} password={{ mysql[‘pass’] }} priv=classics.*:ALL host='%'
state=present
  handlers:
   - name: restart mysql
     service: name=mysqld state=restarted

- hosts: anode53
  remote_user: ansible
  become: true
  become_method: sudo
  vars:
  - mysql:
     user: ilg
     pass: ilg007
```

Executing Playbook (playbook.yml)

```
tasks:
- name: Install httpd package
  yum: name=httpd state=present
- name: enable httpd service
  service: name=httpd state=started enabled=yes
- name: replace variable the source file from controller
  template: src=testpages/dbconn.php.j2 dest=/var/www/html/dbconn.php
```

- Each playbook is composed of one or more ‘plays’ in a list.
- The goal of a play is to map a group of hosts to some well defined roles, represented by things ansible calls tasks.
- A task is nothing more than a call to an ansible module
- The above playbook is composed of two plays; to install apache webserver and mysql database.
- Run playbook using *“ansible-playbook playbook.yml”*

Playbook Explanation (playbook.yml)

1. The first line, ---, is how we mark this document as using YAML syntax (like using <html> at top of html document).
2. The second line, *remote_user: ansible,* is the user which would run playbook on remote node
3. The third line, *become: true,* allows to change to user *ansible* on remote node.
4. The fourth line, *become_method: sudo,* allows privilege escalation
5. The line, *vars:* , sets variables to be used in the play.
6. The line, *tasks:* , follows list of tasks which would be executed on remote node.
7. The line, *handlers:* , follows list of handlers which are to be notified by tasks, and finishes the first play.

Includes

- Includes is a way of organising playbooks by writing some sections to a external file.
- Includes can be used for tasks, handler and complete playbooks.
- Let us use the previous playbook and write includes into it.

Includes

- Including tasks from an external file

```
---
- hosts: anode53
  remote_user: sao-pae
  become: true
  become_method: sudo
  vars:
  - mysql:
     user: ilg
     pass: ilg007
  tasks:
  - include: external-tasks.yml
```

Includes

- And our external-tasks.yml would look like

```
---
 - name: Install MySQL package
    yum: name=mysql-server state=installed
  - name: mysql ansible support
    yum: name=MySQL-python state=installed
  - name: Start MariaDB
    service: name=mysqld state=started enabled=yes
  - name: Create Database
    mysql_db: name=classics state=present
  - name: Create DB User
    mysql_user: name=ilg password=ilg007 priv=classics.*:ALL host='%' state=present
```

Includes

- Similarly you can include Handlers from external file by writing – *include: external-handlers.yml* under *handlers* in your playbooks.
- Include playbooks, you can create playbooks to configure your nodes and then include them in a master playbook

```
- hosts: all
   remote_user: root
   tasks:
    ...
- include: web.yml
- include: db.yml
```

Includes

Complete include example, you can separate each tasks, handlers to a separate file from the previous playbook and create this master playbook with includes

```
---
- hosts: anode53
  remote_user: sao-pae
  become: true
  become_method: sudo

  vars_files:
  - vars.yml

  handlers:
  - include: external-handlers.yml

  tasks:
  - include: tasks/install-mysql.yml
  - include: tasks/createdb-user.yml
  - include: tasks/install-httpd.yml
  - include: tasks/copy-webapp.yml

- include: playbook/db2.yml
- include: playbook/web2.yml
```

Roles

- Roles are the best way to organise your playbooks. Grouping content by roles allows easy sharing of roles with other users.
- Roles are ways of automatically loading certain vars_files, tasks, and handlers based on a known file structure.
- Roles are automation around **'include'** directives.
- Roles contain some improvements to search path handling for referenced files.
- Your roles can live in the default global Ansible roles_path (configurable in ansible.cfg), or a roles folder in the same directory as your main playbook file.

Roles: Example Project structure

```
master-playbook.yml
webservers.yml
fooservers.yml
roles/
   common/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
   webservers/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
```

Roles

In a playbook, it would look like this:

```
---
- hosts: webservers
  roles:
      - common
      - webservers
```

- This designates the following behaviors, for each role 'X':
  - If roles/X/tasks/main.yml exists, tasks listed therein will be added to the play
  - If roles/X/handlers/main.yml exists, handlers listed therein will be added to the play
  - If roles/X/vars/main.yml exists, variables listed therein will be added to the play
  - If roles/X/defaults/main.yml exists, variables listed therein will be added to the play
  - If roles/X/meta/main.yml exists, any role dependencies listed therein will be added to the list of roles
  - Any copy, script, template or include tasks (in the role) can reference files in roles/x/{files,templates,tasks}/ (dir depends on task) without having to path them relatively or absolutely

Roles

- If any files are not present, they are just ignored. So it's ok to not have a 'vars/' subdirectory for the role, for instance.
- You are still allowed to list tasks, vars_files, and handlers "loose" in playbooks without using roles

Deploying Wordpress

- Let us start by making an inventory and a master playbook (master-playbook.yml),
- Here's our inventory file: */home/ansible/hosts*

```
[wordpress-server]
node1
```

- *wordpress-server* is the group name of hosts,
- *node1* is a member of that group in our inventory.

Roles: our master playbook (master-playbook.yml)

```
---
- name: Install WordPress, MySQL, Nginx, and PHP-FPM
  hosts: wordpress-server
  remote_user: ansible
  become: true
  become_method: sudo

  roles:
  - common
  - mysql
  - nginx
  - php-fpm
  - wordpress
```

- We will be executing this playbook on a group in inventory *wordpress-server* since it includes only server now there will only be one server with wordpress else we would have had the same configuration for any server in the hostgroup *wordpress-server.*
- *remote_user* is the user name of the user which would execute the playbook on the remote server.
- *become* allows ansible to switch user in the remote to the *remote_user* and *become_method* specifies the method to switch to the user.

Roles: our master playbook

- You would be using the following roles common, mysql, nginx, php-fpm, wordpress
  - common: This is the role which you would want on all servers you manage, like a basic setup necessary on all your servers
  - mysql: This would be a role for how you configure your mysql database server
  - php-fpm: A role for installing, configuring php support for your web application
  - wordpress: The final role for installing the wordpress application

Creating the roles

- The roles would be created under the directory *roles* so there would a subdirectory for your each role under *roles*.

  E.g:- role *common* would be under the directory roles and include subdirectories for tasks, files and handlers.

Creating the role: common

- Create directory common under the directory *roles.*
- Create subdirectories *files*, *handlers*, *tasks* under the directory *common*
- Create a playbook for tasks

  roles/common/tasks/main.yml

Creating the role: common

```
---
- name: Install libselinux-python
  yum: name=libselinux-python state=present

- name: Copy the EPEL repository definition
  copy: src=epel.repo dest=/etc/yum.repos.d/epel.repo

- name: Create the GPG key for EPEL
  copy: src=RPM-GPG-KEY-EPEL-6 dest=/etc/pki/rpm-gpg

- name: Set up iptables rules
  copy: src=iptables-save dest=/etc/sysconfig/iptables
  notify: restart iptables
```

- Create a playbook for handlers *roles/common/handlers/main.yml*

```
---
- name: restart iptables
  service: name=iptables state=restarted
```

.

Creating the role: common

- And now let us check or create the files which you are copying to the remote server. To prevent typing mistakes these files will be provided
- *roles/common/files/epel.repo*
- *roles/common/files/RPM-GPG-KEY-EPEL-6*
- *roles/common/files/iptables-save*

Creating the role: mysql

- Create directory mysql under the directory *roles*.
- Create subdirectories *templates*, *handlers*, *tasks* under the directory *mysql*.
- Create a playbook for tasks *roles/mysql/tasks/main.yml*.

Creating the role: mysql

```
---
- name: Install Mysql package
  yum: name={{ item }} state=present
  with_items:
   - mysql-server
   - MySQL-python
   - libselinux-python
   - libsemanage-python

- name: Configure SELinux to start mysql on any port
  seboolean: name=mysql_connect_any state=true persistent=yes
  when: ansible_selinux.status == "enabled"

- name: Create Mysql configuration file
  template: src=my.cnf.j2 dest=/etc/my.cnf
  notify:
  - restart mysql

- name: Start Mysql Service
  service: name=mysqld state=started enabled=yes
```

Creating the role: mysql

Create a playbook for handlers *roles/mysql/handlers/main.yml*

```
---
- name: restart mysql
  service: name=mysqld state=restarted
```

And now the template file used in the tasks *roles/mysql/templates/my.cnf.j2*

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended to prevent assorted
security risks
symbolic-links=0
port={{ mysql_port }}

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

Creating the role: nginx

- Create directory nginx under the directory roles
- Create subdirectories templates, handlers, tasks under the directory nginx
- Create a playbook for tasks roles/nginx/tasks/main.yml

```
---
- name: Install nginx
  yum: name=nginx state=present

- name: Copy nginx configuration for wordpress
  template: src=default.conf dest=/etc/nginx/conf.d/default.conf
  notify: restart nginx
```

Create a playbook for handlers *roles/nginx/handlers/main.yml*

```
---
- name: restart nginx
  service: name=nginx state=restarted enabled=yes
```

And now the template file used in the tasks

roles/nginx/templates/default.conf

Create Role php-fpm

- Create directory php-fpm under the directory *roles*.
- Create subdirectories *templates*, *handlers*, *tasks* under the directory *php-fpm*.
- Create a playbook for tasks *roles/php-fpm/tasks/main.yml*.

Create Role php-fpm

- Create a playbook for handlers *roles/php-fpm/handlers/main.yml*

```
---
-  name: restart php-fpm
   service: name=php-fpm state=restarted
```

Create Role php-fpm

Create a playbook under **roles/php-fpm/tasks/main.yml**

```
---
- name: Install php-fpm and deps
  yum: name={{ item }} state=present
  with_items:
    - php
    - php-fpm
    - php-enchant
    - php-IDNA_Convert
    - php-mbstring
    - php-mysql
    - php-PHPMailer
    - php-process
    - php-simplepie
    - php-xml


- name: Disable default pool
  command: mv /etc/php-fpm.d/www.conf /etc/php-fpm.d/www.disabled creates=/etc/php-fpm.d/www.disabled
  notify: restart php-fpm


- name: Copy php-fpm configuration
  template: src=wordpress.conf dest=/etc/php-fpm.d/
  notify: restart php-fpm
```

Create Role php-fpm

- Create a playbook for handlers *roles/php-fpm/templates/wordpress.conf*

```
[wordpress]
listen = /var/run/php-fpm/wordpress.sock
listen.owner = nginx
listen.group = nginx
listen.mode = 0660
user = wordpress
group = wordpress
pm = dynamic
pm.max_children = 10
pm.start_servers = 1
pm.min_spare_servers = 1
pm.max_spare_servers = 3
pm.max_requests = 500
chdir = /srv/wordpress/
php_admin_value[open_basedir] = /srv/wordpress/:/tmp
```

Creating the role: wordpress

- Create directory *wordpress* under the directory *roles*.
- Create subdirectories *templates*, *tasks* under the directory *wordpress*.
- Create a playbook for tasks *roles/wordpress/tasks/main.yml*

```
---
- name: Download WordPress
  get_url: url=http://wordpress.org/wordpress-{{ wp_version }}.tar.gz dest=/srv/wordpress-
{{ wp_version }}.tar.gz
           sha256sum="{{ wp_sha256sum }}"

- name: Extract archive
  command: chdir=/srv/ /bin/tar xvf wordpress-{{ wp_version }}.tar.gz creates=/srv/wordpress

- name: Add group "wordpress"
  group: name=wordpress
```

Creating the role: wordpress

```
- name: Add user "wordpress"
  user: name=wordpress group=wordpress home=/srv/wordpress/

- name: Fetch random salts for WordPress config
  local_action: command curl https://api.wordpress.org/secret-key/1.1/salt/
  register: "wp_salt"
  become: no
  become_method: sudo

- name: Create WordPress database
  mysql_db: name={{ wp_db_name }} state=present

- name: Create WordPress database user
  mysql_user: name={{ wp_db_user }} password={{ wp_db_password }} priv={{ wp_db_name }}.*:ALL
host='localhost' state=present

- name: Copy WordPress config file
  template: src=wp-config.php dest=/srv/wordpress/

- name: Change ownership of WordPress installation
  file: path=/srv/wordpress/ owner=wordpress group=wordpress state=directory recurse=yes

- name: Start php-fpm Service
  service: name=php-fpm state=started enabled=yes
```

Creating the role: wordpress

- And now the template file *roles/wordpress/templates/wp-config.php*

```
<?php
define('DB_NAME', '{{ wp_db_name }}');
define('DB_USER', '{{ wp_db_user }}');
define('DB_PASSWORD', '{{ wp_db_password }}');
define('DB_HOST', 'localhost');
define('DB_CHARSET', 'utf8');
define('WPLANG', '');
define('WP_DEBUG', false);
define( 'AUTOMATIC_UPDATER_DISABLED', {{auto_up_disable}} );
define( 'WP_AUTO_UPDATE_CORE', {{core_update_level}} );
if ( !defined('ABSPATH') )
        define('ABSPATH', dirname(__FILE__) . '/');
require_once(ABSPATH . 'wp-settings.php');
```

Creating the role: Variables

- Now that you have completed the roles, you would need to set values for variables according to your configuration
- Create *group_vars* directory and a file named *all* under it, which we will use to replace variables in your roles with your custom values

```
---
wp_version: 4.2.4
wp_sha256sum: 42ca594afc709cbef8528a6096f5a1efe96dcf3164e7ce321e87d57ae015cc82
wp_db_name: wordpress
wp_db_user: wordpress
wp_db_password: secret
mysql_port: 3306
server_hostname: www.example.com
automatic updates
auto_up_disable: false
core_update_level: true
```

Running the master playbook

- Run the command *ansible-command* and provide the master playbook as a argument to it

```
ansible-playbook master-playbook.yml
```