Inventory for variables and roles
Playbook= actions, plays in sequences
Play= task

Plays map host to tasks, can have multiple tasks; playbook has multiple plays.

Inventory maps hosts
Config sets Ansible parameters Module define actions
Playbooks coordinate multiple tasks
Python to deliver execution
SSH to deliver
JSON to report back

Ansible config global, but can be specific  (separate one) for a playbook/ host(s)

Per play Ansible Control Server sends python package to host(s) JSON results are sent back

Variables
 - /defaults/main.yaml, defaults overriden by other files, in inventory, then vars, then CLI (-e "varname = X")
 - facts about host(s)
 - Dynamic variable for temp info (such as host state)

Vagrant

Vagrantfile config items: See /ansible-vbox-vagrant-kubernetes-master/Vagrantfile

Simple example ———————————————————————————————

Vagrant.configure("2") do |config|

  config.vm.define “acs” do |acs|
    acs.vm.box = “ubuntu/trusty64”
    acs.vm.hostname = “acs”
    acs.vm.network “private_network”, ip: "192.168.33.10”
  end

  config.vm.define “web” do |web|
    web.vm.box = “nrel/CentOS-6.5-x86_64”
    web.vm.hostname = “web”
    web.vm.network “private_network”, ip: "192.168.33.20”
    web.vm.network “forwarded_port”, guest: 80, host: 8080
  end

```
  config.vm.define “db” do |db|
    db.vm.box = “nrel/CentOS-6.5-x86_64”
    db.vm.hostname = “db”
    db.vm.network “private_network”, ip: "192.168.33.30”
  end

end
```

———————

Vagrant up
Vboxmanage list runningvms
Vagrant ssh acs
apt-get install ansible -y
Vagrant ssh web
yum install epel-release -y
yum install ansible -y

yum install gcc
yum install python-setuptools
easy_install pip
yum install python-devL
pip install ansible

---

- new folder, new file 'inventory', 2 lines 192.168.33.20 and .30
- ssh vagrant@192.168.33.20
- ansible 192.168.33.20 -i inventory -u vagrant -m ping -k
--  -k to prompt for password
-- adding a -vvv to the above lets you debug if ansible looks like it is screwing up

ansible all -i inventory -u vagrant -m command -a "/usr/sbin/yum update -y
  - '-m command' can be excluded. command module is default
Shell module vs command module- command module passes commands through python, shell gives access to ENVVARs and such.

---

Inventory features:
- behavioral params, groups, groups of groups, variable assignment, scale to multiple files
- static or dynamic (like parsed output of a script)

Start inventory with :
db1.company.com
db2.company.com

---

```
# specific params for host- username on one, specific py interpreter for the other
# db is group
[db]
db1.company.com ansible_ssh_user=dave ansible_ssh_pass=abc123
db2.company.com ansible_python_interpreter=/usr/bin/python

# children denotes DC-west is parent
[datacenter-west:children]
db

# vars for all in group
[datacenter-west:vars]
 ansible_ssh_user=ansib_user
 ansible_ssh_pass=%&%TYDFRYH
 ntp-server= 5.6.7.8
==========================================

# use PKI or Ansible Vault for passwords!
 web1.company.com ansible_ssh_host=192.168.33.20 ansible_ssh_user=vagrant
ansible_ssh_pass=vagrant
  web2.company.com ansible_ssh_host=192.168.33.30 ansible_ssh_user=vagrant
ansible_ssh_pass=vagrant

# Here and above web1 if not a full domain name will slap the local domain name on
# (still best practice to specify)

[webservers]
web1
web2
# "ansible webservers -i inventory -u vagrant -m ping -k" works
# Aside: systems can be in multiple groups

[dbservers]
db1
db2

[datacenter:children]
webservers
dbservers

# move the items from above into veriables
[datacenter:vars]
ansible_ssh_user=vagrant
ansible_ssh_pass=vagrant

===============
Inventory in yaml (this has been ini)
```

```
all:
  hosts:
    mail.mycorp.local:
    smtp.mycorp.local:
    auth.mycorp.local:
  children:
    itlappsrv:
      children:
        ldappsrv:
          hosts:
            app01.ldn.mycorp.local:
            app02.ldn.mycorp.local:
        txappsrv:
          hosts:
            app03.tx.mycorp.local:
            app04.tx.mycorp.local:
    dc:
      children:
        dc01.mycorp.local:
        dc02.mycorp.local:
```

==========

Scaling out to multiple files when files get too big

```
|------------ group_vars
|         |------------ all
|         |------------ db
|------------ host_vars
|         |------------ web1
|------------ inventory_prod
|------------ inventory_test
```

# Here inventory_prod and test, group, host can share variables (same directory layer)

# Better way:

```
    /production
|------------ group_vars
|         |------------ all
|         |------------ db
|------------ host_vars
|         |------------ web1
|------------ inventory_prod
     /test
|------------ group_vars
```

|  |---------- all
|  |---------- db
|---------- host_vars
|  |---------- web1
|---------- inventory_test

Precedence
1 - group_vars all
2 - group_vars grp name - overrides above defined stuff
3 - host_vars hostname - overrides above defined stuff

name={{username}} -- insert variable

========== Ansible config

Precedence (in this order- finds the config it can use and stops there and looks no further)
1 -  $ANSIBLE_CONIG
2 - ./ansible.cfg (in current folder)
3 - ~/.ansible.cfg
4 - /etc/ansible/ansible.cfg (global config)
ENV VAR setting precedes all of these. $ANSIBLE_<configsetting> - export ANSIBLE_FORKS=10
For on-the-fly changes

Common items
Forks - default is 5, production recommendation +/-20 based on performance
host_key_checking- production true, dev environment false
log_path = default is null, set path, add users for wrrite permissions

==========
Current Ansible version is 4.6 - Prior to Ans 2.5, Ansible required Py2.x, not 3.x... install pkg mgr over existing 3.x, then do whereis to point anasible to 2.x version
Add in inventory for that host something like:
192.168.33.50 ansible_python_interpreter=/usr/bin/python2.7

==========

Modules: Core, extras (included 3rd party), and deprecated.
ansible-doc -t become -1
 -l for list, -s <name> for playbook examples/snippet
Frequent modules: Copy (push), Fetch (get), apt, yum,etc, service (like systemd, sysinit)

Setup module - gather facts.
# ansible web1 -i inventory -m setup
Lists facts that can be used as variables
# ansible web1 -i inventory -m setup -a "filter=ansible_eth*" -- list all eth interfaces on

host
# ansible web1 -i inventory -m setup -a "filter=ansible_mounts" -- list all drives on host
# ansible all -i inventory -m setup ---tree ./setup -- dump all info one file per host in setup dir

==========
2 plays in a playbook:

```
---
-hosts: webservers
  remote_user:root
  tasks:
  - name: Install Apache
    yum: name=httpd state=present
  - name: Start Apache
    service: name=httpd state=started

-hosts: dbservers
  remote_user:root
  tasks:
  - name: Install MySQL
    yum: name=mysql-server state=present
  - name: Start MySQL
    service: name=mysqld state=started

-------
-hosts: webservers
  remote_user:root
  tasks:
## a block like this is sortof a global play declaration

-hosts: webservers
  vars:
     git_repo: https:// github.com/.....
     http_port: 8080
     db_name: wordpress
## declare user to run task
  sudo: yes
  sudo_user: wordpress_user
  gather_facts: no
## gathering facts take a lot of resource - forget it unless needed
```

ansible-playbook playbook.yml

If a host fails it won't keep going- it is removed from the "pool"
to fix, use retry file and execute ansible-playbook with --limit @home/ping.retry (it gives the real path)

ansible-playbook playbook.yml --limit @home/web_db.yaml.retry

---
This in the local ansible config will make it unnecessary to -i to specify inventory file
[defaults]
hostfile = inventory

==========

**Include Files to Extend Playbook**
- Breaks up long playbooks
- Use to add external variable files Reuse other playbooks

```
tasks:
    - include: wordpress.yml

    vars:
      sitename: My Awesome Site
    - include: loadbalancer.yml
    - include_vars: variables.yml
```

**----------**

**Grab output of task for another task**
- Useful to use tasks to feed data into other tasks
- Useful to create custom error trapping

```
tasks:
    - shell: /usr/bin/whoami
        register: username
    - file: path=/home/myfile.txt
        owner={{ username }}
```

**----------**

**Add debug to tasks**
- Useful to send output to screen during execution
- Helps find problems

```
tasks:
    - debug: msg="This host is {{ inventory_hostname }} during execution"

    - shell: /usr/bin/whoami register: username
    - debug: var=username
```

**----------**

**Prompt user during execution**
- Creates Dynamic Playbooks

```
- hosts: web1

vars_prompt:
   - name: "sitename"
        prompt: "What is new site name?"

tasks:
   - debug: msg="The name is {{ sitename }}"
```

**----------**

**Playbook Handlers**
- Tasks with asynchronous execution
- Only runs tasks when notified
- Tasks only notify when state=changed
- Does not run until all playbook tasks have executed
- Most common for restarting services to load changes (if changes are made)

**Notify handlers from your tasks**

```
tasks:
   - copy: src=files/httpd.conf dest=/etc/httpd/conf/
     notify:
       - Apache Restart

handlers:
   - name: Apache Restart
       service: name=httpd state=restarted
```

Use the clause “when” to choose if task should run.

**Conditional Clause - Choose when to execute tasks**
Uses YUM if OS is RedHat
Uses APT if OS is Debian

```
tasks:
   - yum: name=httpd state=present
        when: ansible_os_family == “RedHat”
   - apt: name=apache2 state=present
        when: ansible_os_family == “Debian”
```

**Conditional Clause Based on Output**
Track whether previous task ran
Searches JSON result for status
Status Options: success, failed, skipped

```
tasks:
   - command: ls /path/doesnt/exist
         register: result
         ignore_errors: yes
   - debug: msg="Failure!"
         when: resultlfailed
```

---

**Templates**
- Use Jinja2 Engine
- Insert variables into static files
- Creates and copies dynamic files
- Deploy custom configurations

Template Module - Modify Template and Copy
Takes a file with pre-defined variable names
Inserts variable values in file Copies file to destination

```
tasks:
   - template:
         src=templates/httpd.j2
         dest=/etc/httpd/conf/httpd.conf
         owner=httpd
```

--- > httpd.j2

```
<VirtualHost *:80>
ServerAdmin {{ server_admin }}
DocumentRoot {{ site_root }}
ServerName {{ inventory_hostname }}
</VirtualHost>
```

ROLE: Builder

```
    - tasks:
         - name: install gcc
         - name: install jdk
         - name: setup git
```

ROLE: Server-Common

```
    - tasks:
```

```
        - name: configure SNMP
        - name: configure SYSLOG
        - name: configure NTP

ROLE: Repo
   - tasks:
        - name: git
        - name: configure
        - name: pull latest
```

```
└──── roles
├──── builders
├──── server-common
|   ├──── defaults
|   ├──── files   <----------
|   ├──── handlers
|   ├──── meta
|   ├──── tasks    <----------
|   ├──── templates    <----------
|   └──── vars
└──── webservers
```

**main.yml - Primary file that Ansible looks for**

```
├──── handlers
|   └──── main.yml
├──── tasks
|   └──── main.yml
├──── templates
└──── vars
    └──── main.yml
```

tasks/main.yml  -- Includes to break-up long files

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

**--- ----- site.yml - Primary file to include entire infrastructure**

```
├──── site.yml
├──── handlers
|   └──── main.yml
├──── tasks
|   └──── main.yml
├──── templates
└──── vars
    └──── main.yml
```

site.yml - Use tags to define categories within your playbooks
- include: webservers.yml tags=web
- include: dbservers.yml tags=db

```
tasks:
  debug: msg="This will only run on tag 'debug'"
  tags:
        - debug

  debug: msg="You can also use multiple tags" tags:
        - debug
        - tag2
```

**Adding Roles to Playbook**

```
- hosts: code-dev
  gather_facts: no

  tasks:
# Build your extra tasks here like
# creating users, or deploying a specific config

  roles:
        - server-common
        - builders
```

**Pre-tasks and Post-tasks**

pre_tasks: Executes plays BEFORE roles
Use-Cases:
- Setup of maintenance windows
- Removing servers from Load- balancers
- Silencing alarms

post_tasks: Executes plays AFTER roles
Use-Cases:
- Clearing of maintenance windows
- Adding servers to Load- balancers

- Enabling Alarms

```
---
- hosts: webservers
  pre_tasks:
        - # Remove from load-balancer
  roles:
        - server-common
        - jboss
  post_tasks:
        - # Add to load-balancer
  gather_facts: no
```

Tagged execution of roles, limited by another tag
$ ansible-playbook site.yml —tags “web” —limit atlanta

Getting Roles
•Create your own roles - Perfect for proprietary applications or workflows
• Find roles to download - Look for others that had the same requirement and shared their work

$ ansible-galaxy install username.role

$ sudo ansible-galaxy install geerlingguy.git

Step 1: Enter/create a roles folder in existing project, then execute this:
$ ansible-galaxy init apache -------- spits out an apache role

Step 2: make new playbook.yaml in roles next to apache directory

Step 3: tasks/ and handlers/main.yaml in Apache will be edited with our info in old playbook
        - Handlers gets block "Reload Apache"
        - Tasks/main.yaml gets this from pre-role playbook previously written:

```
---
- name: Update All Packages
  yum:
    name: '*'
    state: latest
- name: Install Apache
  yum:
    name: httpd
    state: installed
- name: Create The HTML File
  shell: echo "Hello From The Ansible Challenge" > /var/www/html/index.html
```

```
  args:
    executable: /bin/bash
  notify:
    - Reload Apache
- name: Public IP
  shell:
    cmd: curl http://169.254.169.254/latest/meta-data/public-ipv4
  register: curl
- debug: var=curl.stdout_lines
```

Step 4: in new Playbook.yaml, (in Role next to Apache directory) paste this, add Roles directive.

```
- hosts: web
  remote_user: root
  become: yes
  roles:
      - apache     # if playbook not in roles folder specify path
```

ansible-playbook -i aws.ini roles/playbook.yaml

---------- ROLE IMPORT via URL

In roles>requirements.yaml

```
roles:
   - name: geerlingguy.docker    # from Galaxy
   - src: (github link)
     name: apache-ch

collections:
     - name: geerlingguy.k8s
```

ansible-galaxy install --roles-path . -r requirements.yaml   ("." means install "here") dumps the directories into roles directory for us

ansible-galaxy collection install -p . -r requirements.yaml

----------VARIABLES

**Variables 3 basic ways: Each more portable**

```
- hosts: database
  vars:
          conf_name: myconf.sql
  tasks:
  - name: ensure SQL is at the latest version
    yum:
      name: mysql-server
      state: latest
  - name: write the apache config file
    copy:
      src: {{conf_name}}
      dest: /etc/{{conf_name}}
```

----------AND----------

```
  tasks:
  - name: Install packages
    yum:
      name: {{item}}
      state: present
      with_items:
          - python
          - python-pip
          - vim
```

{{item}} works with the reserved keyword "with_items"; the variable in {{}} declared first- the reverse of previous example.

----------AND finally----------

```
- hosts: database
  vars:
        packages:
          - python
          - python-pip
          - vim
  tasks:
  - name: install packages
    yum:
      name={{item}}
      state: present
      with_items: packages
```

---
```
- hosts: database
```

```
  vars:
       packages:
         - python
         - python-pip
         - vim
  tasks:
  - name: Display Value
          debug:
          msg: "{{ courses[0] }}".    <--- python
```

---

**Ultimately the better way of doing variables, etc:**

This puts things "in scope" more efficiently and they won't be cluttering the main playbook but isolated into their own units:

---

```
   vars.yaml:
http_port: 80
server_name: prod_dc01
```

---

```
   var_file_demo.yaml
- hosts: localhost
  vars_files:           ## add this var_files block to
    - vars.yaml       ## make it declaratively accessible
  tasks:
  - name: Display 1st Value
    debug:
      msg: "{{ http_port }}"
  - name: Display 2nd Value
    debug:
      msg: "{{ server_name }}"
```

ansible-playbook -e "@vars.yaml" var_file_demo.yaml
(manually passing a var file rather than declarative)

---

LOOP THROUGH

----- multi.yaml

```
- hosts: web
  remote_user: root
  become: yes
  vars:
```

```
    packages:
      - httpd
      - nano
      - mysql
  tasks:
  - name: Install Software
    yum:
      name: "{{ item }}"
      state: installed
    loop: "{{ packages }}"
  - name: Remove Software
    yum:
      name:
        - httpd
        - nano
        - mysql
      state: removed
```

ansible-playbook -i aws.ini variables/multi.yaml

aws.ini:
web ansible_host=ec2-18-130-249-7.eu-west-2.compute.amazonaws.com ansible_port=22 ansible_user=ec2-user ansible_ssh_private_key_file=/Users/Keys/ansible_lnd_key.pem

---

Override a var at the CLI for testing

cli.yaml:

```
- hosts: localhost
  tasks:
  - name: Display Value
    debug:
      msg: "{{ basic_var }}"
```

ansible-playbook -e "basic_var=CLI" cli.yaml

Role dependencies
 - Inside roles, meta directory playbook would specify other roles to incorporate into this one:
(these run first)

```
          dependencies:
```

```
        - common
        - uwsgi
        - nginx
```

---------- **Installing 2 roles from scratch**

1. New folder, in it a new playbook.yaml
2. Add

```
 - hosts: web
        remote_user: root
        become: yes
 - roles
        - web
        - db
```

3. Create roles folders:

```
        # ansible-galaxy init web
        # ansible-galaxy init db
```

4a. In /web/tasks/main.yml

```
 - name: Install Apache
    yum:
        name: httpd
        state: installed
    notify: Start Web
```

4b. In /web/handlers/main.yml

```
- name: Start Web
     service:
        name: httpd
        state: started
```

5a. In /db/tasks/main.yml

```
 - name: Install DB
    yum:
        name: "{{ db_server }}"
        state: installed
    notify: Start DB
 - name: Display DB installed
    debug:
        msg: "{{ db_server }}"
```

5b. In /db/defaults/main.yml

```
        db_server: mariadb    ## default
```

5c. In /db/vars/main.yml

```
        db_server: mysql       ## override the default
```

5d.  In /db/handlers/main.yml

```
- name: Start DB
     service:
        name: "{{ db_server }}"
        state: started
```

6. DB server needs to be started before web server
        Use order in playbook, or...
In /web/meta/main.yml

```
     dependencies:
        -db
```

ansible-playbook -i aws.ini webdb/playbook.yaml

Where aws.ini inventory is:
web ansible_host=ec2-18-130-249-7.eu-west-2.compute.amazonaws.com ansible_port=22 ansible_user=ec2-user ansible_ssh_private_key_file=/Users/Keys/ansible_lnd_key.pem

---------- **Dynamic Inventory**
Run ./ec2.py   - basic polling info on running EC2 instances w/ ip addresses
Demo goes into AWS Dashboard and spins up 9 instances
Created ec2.yaml using playbook contents as base, inserting info from aws.yaml as vars

```
- hosts: ec2
  remote_user: root
  #become: yes
  strategy: free
  serial:
    - "50%"     # stagger concurrent processes. See below.
  vars:
    ansible_port: 22
    ansible_user: ec2-user
    ansible_ssh_private_key_file: /Users/Keys/ansible_lnd_key.pem
## task from old playbook
  tasks:
  - name: Install Apache
    yum:
      name: httpd
      state: installed
  - name: Remove Apache
    yum:
      name: httpd
```

state: removed

Testing, set ENVVAR----> # ANSIBLE_HOST_KEY_CHECKING=False
ansible-playbook -i ec2.py ec2.yaml  ## Note use of python output as an inventory file

About serial, and timing
serial: 2 is running on two instances simultaneously at a time

serial:
- 1
- 2
- 4

First one at a time, if successful, then two, then three
% as used in example is 50% of instances in inventory
Similarly you can do this:
serial:
- "25%"
- "50%"
....

---------- **Working with Windows EC2 Instances**
Security Group needs RDP, WinRM-HTTPS, All ICMP-IPv4 w/ source "My IP"
When it spins up, copy public DNS name from AWS dashboard

----------  wins_server.yaml:

```
all:
  hosts:
    windows:
      ansible_host: ec2-18-132-47-61.eu-west-2.compute.amazonaws.com
      ansible_user: administrator
      ansible_password: PASSWORD  #see below
      ansible_port: 5986       ## WinRM
      ansible_connection: winrm
      ansible_winrm_scheme: https
      ansible_winrm_server_cert_validation: ignore   # we aren't doing cert validation
```

Password: in AWS Instance list, select instance, go to Actions> Get Windows Password
It will ask for where the key pair is locally stored (Key Pair Path)
Click Decrypt Password and remote login including pass is given.
Use Ansible Vault!!

Mac: MS Remote Desktop app
Ansible docs: Setting up a Windows host has PS to run in the Win Instance running environment
(See WinRM Setup) copy and paste (notepad eg.) name winrm.ps1, right click and "run with PS"

ansible windows -i win_server.yaml -m win_whoami

---------- **Ansible Vault**
ansible-vault encrypt file_with_passwd.yaml
Asks for new pass and confirmation, opening file now shows encryption
ansible-vault view file_with_passwd.yaml
ansible-playbook -i file_with_passwd.yaml --ask-vault-pass playbook.yaml
ansible-vault rekey file_with_passwd.yaml  #### change password
ansible-vault decrypt file_with_passwd.yaml  ### remove encryption

This vault password is for the vault, not just the one file.  Need better solution:
Hashicorp Vault

---------- **HashiCorp Vault**
Mac need to install Ansible using pip, not brew (paths are different)
pip install hvac  ## needed py library

Link to install Vault: https://learn.hashicorp.com/tutorials/vault/getting-started-install
Link to the documentation for the vault module:
https://docs.ansible.com/ansible/latest/plugins/lookup/hashi_vault.html
If you run into: in progress in another thread when fork() was called
Run: export OBJC_DISABLE_INITIALIZE_FORK_SAFETY=YES
Then re-run the command

{sidenote: Chocolatey is like Homebrew for Windows)

---------vault_demo.yaml

```
- hosts: localhost
  vars:
    vault_token: 'TOKEN'
    vault_url: 'http://127.0.0.1:8200'
    db_password: "{{ lookup('hashi_vault', 'secret=secret/passwords/db:data
token={{ vault_token }} url={{ vault_url }}') }}"
    system_password: "{{ lookup('hashi_vault', 'secret=secret/passwords/system:data
token={{ vault_token }} url={{ vault_url }}') }}"
  tasks:
  - name: Value For db_password
    debug:
      msg: "db_password: {{ db_password }}"
  - name: Value For system_password
    debug:
      msg: "db_password: {{ system_password }}"
```

vault server -dev  # startt Vault server- dev, NOT production environment!
 You are given a Unseal Key and a Root Token

-Root token keep secret, hide.  This goes above in vault_token (later into the variable)

---------- **Basics:**

**Ad hoc commands:** when you operate on an inventory with a single command like this:

```
ansible database -i inventory -m ping
ansible database -i inventory -m apt -a "name=sql-serve state=present"
```

Here working with the ping and apt modules, inventory being the path to an inventory

---

**Here is the example of using the template module in a playbook:**

```
- hosts: all
  tasks:
  - name: Update apt cache only at particular interval
    apt: update_cache=yes cache_valid_time=36000

- hosts: webservers
  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
```

Template module uses Jinja2 templating by default (var substitutions option)

**More template examples:**

```
- template:
    src: /mytemplates/foo.j2
    dest: /etc/file.conf
    owner: bin
    group: wheel
    mode: 0644

# The same example, but using symbolic modes equivalent to 0644
- template:
    src: /mytemplates/foo.j2
    dest: /etc/file.conf
    owner: bin
```

```
    group: wheel
    mode: "u=rw,g=r,o=r"

# Create a DOS-style text file from a template
- template:
    src: config.ini.j2
    dest: /share/windows/config.ini
    newline_sequence: '\r\n'

# Copy a new "sudoers" file into place, after passing validation with visudo
- template:
    src: /mine/sudoers
    dest: /etc/sudoers
    validate: '/usr/sbin/visudo -cf %s'

# Update sshd configuration safely, avoid locking yourself out
- template:
    src: etc/ssh/sshd_config.j2
    dest: /etc/ssh/sshd_config
    owner: root
    group: root
    mode: '0600'
    validate: /usr/sbin/sshd -t -f %s
    backup: yes
```

---

A directory structure of:

```
/
inventory
deploy.yml
group_vars/
        all
        database
```

host_vars/
	server1.blahcom.com

---

 - running "ansible-playbook -i inventory deploy.yml"
 - group_vars and host_vars are standard names Ansible looks for
 - *inventory can also be an executable script to go find eligible devices and output proper input*
 - there are standard scripts in the public repo for this

**Another way:**
/
deploy.yml

staging/
	hosts
	group_vars/
	host_vars/

live/
	hosts
	group_vars/
	host_vars/

 - running "ansible-playbook -i staging deploy.yml"

**Set up roles in site.yml to modularize**

- hosts: database
	roles:
	 - common
	 - webserver

- hosts: webservers
	roles:
	 - common
	 - sql

Using this directory structure:
inventory/
roles/
	common/
	database/
	webserver/
	uwsgi/
	nginx/

site.yml

- running "ansible-playbook -i inventory site.yml"

roles/
    common/
        defaults/
            main.yml    <----for default var values

        tasks/
            main.yml    <----for this role's tasks

        files/
            myscript.py    <---- copy for this role

        templates/
            config.py.j2    <---- copy for this role

        meta/
            main.yml    <---- copy for this role

- Default var values in roles can be overridden by inventory
- Ansible hub/ Galaxy has roles to download

This allows our main playbook to only require
--hosts: webservers
    roles:
    - webserver
Rather than a huge list of roles

site.yml
roles/
  common/
    files/
    templates/
    tasks/
    handlers/
    vars/
    defaults/
    meta/
  apache/
    files/
    templates/
    tasks/

```
    handlers/
    vars/
    defaults/
    meta/


# site.yml
---
- hosts: web
  roles:
    - common
    - apache
```

https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html
https://www.youtube.com/watch?v=5BhAJ4mEfZ8

https://www.infoq.com/articles/rest-anti-patterns

https://pythontips.com/2013/07/30/20-python-libraries-you-cant-live-without/

```
site.yml
roles/
        common/
                files/
                templates/
                tasks/
                handlers/
                vars/
                defaults/
                meta/
        apache/
                files/
                templates/
                tasks/
        ...etc...
```

The uWSGI project
https://uwsgi-docs.readthedocs.io/en/latest/