* Boolean Comparision
  + Ansible 2+:
    - **ensure no space before of after True or False in jinja filter else bolleans do not work!**
    - True represented internally as string 'True'
    - False represented internally as string 'False'
    - Is variable = tests correctly if true or false
    - Is Not Variable: does not work
      * Must use: variable != 'True'
    - Combine 2 boolean checks – use full form
      * is\_cluster == 'True' and is\_part\_of\_cluster == 'True'
* Playbooks
  + Keep top level playbook short and include other plabooks
    - - include: webservers.yml
    - - include: dbservers.yml
* --list-tasks
  + just lists tasks that would run
* --list-hosts
  + Just lists hosts that would be included in run
* Inventory
  + Ansible works against multiple systems in your infrastructure at the same time. It does this by selecting portions of systems listed in Ansible’s inventory file, which defaults to being saved in the location /etc/ansible/hosts.
  + –i specifies inventory file to use
  + -–limit limits run to matching hosts/groups
  + The format for /etc/ansible/hosts is an INI-like format and looks like this:
    - mail.example.com
    - [webservers]
    - foo.example.com
    - bar.example.com
    - [dbservers]
    - one.example.com
    - two.example.com
    - three.example.com
  + It is ok to put systems in more than one group. If you do, note that variables will come from all of the groups they are a member of, and variable precedence is detailed in a later chapter.
  + If you have hosts that run on non-standard SSH ports you can put the port number after the hostname with a colon
    - badwolf.example.com:5309
  + Suppose you have just static IPs and want to set up some aliases that live in your host file, or you are connecting through tunnels. You can also describe hosts like this:
    - jumper ansible\_port=5555 ansible\_host=192.168.1.50
  + Adding a lot of hosts? 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
    - db-[a:f].example.com
  + You can also select the connection type and user on a per host basis:
    - [targets]
    - localhost ansible\_connection=local
    - other1.example.com ansible\_connection=ssh ansible\_user=mpdehaan
    - other2.example.com ansible\_connection=ssh ansible\_user=mdehaan
  + As alluded to above, it is easy to assign variables to hosts that will be used later in playbooks:
    - [atlanta]
    - host1 http\_port=80 maxRequestsPerChild=808
    - host2 http\_port=303 maxRequestsPerChild=909
  + Variables can also be applied to an entire group at once:
    - [atlanta]
    - host1
    - host2
    - [atlanta:vars]
    - ntp\_server=ntp.atlanta.example.com
    - proxy=proxy.atlanta.example.com
  + It is also possible to make **groups of groups using the :children** suffix.
    - [southeast:children]
    - atlanta
    - Raleigh
* **Host Patterns**
  + The following patterns are equivalent and target all hosts in the inventory:
    - all
    - \*
  + It is also possible to address a specific host or set of hosts by name:
    - one.example.com
    - one.example.com:two.example.com
    - 192.168.1.50
    - 192.168.1.\*
  + Groups separated by a colon indicate an “OR” configuration. This means the host may be in either one group or the other:
    - webservers
    - webservers:dbservers
  + You can exclude groups as well, for instance, all machines must be in the group webservers but not in the group phoenix:
    - webservers:!phoenix
  + You can also specify the intersection of two groups. This would mean the hosts must be in the group webservers and the host must also be in the group staging:
    - webservers:&staging
  + You can do combinations:
    - webservers:dbservers:&staging:!phoenix
  + You can also use variables if you want to pass some group specifiers via the “-e” argument to ansible-playbook, but this is uncommonly used:
    - webservers:!{{excluded}}:&{{required}}
  + You also don’t have to manage by strictly defined groups. Individual host names, IPs and groups, can also be referenced using wildcards:
    - \*.example.com
    - \*.com
* **Splitting Out Host and Group Specific Data**
  + Assuming the inventory file path is:
    - /etc/ansible/hosts
  + If the host is named ‘foosball’, and in groups ‘raleigh’ and ‘webservers’, variables in YAML files at the following locations will be made available to the host: (# can optionally end in '.yml', '.yaml', or '.json')
    - /etc/ansible/group\_vars/raleigh
    - /etc/ansible/group\_vars/webservers
    - /etc/ansible/host\_vars/foosball
  + For instance, suppose you have hosts grouped by datacenter, and each datacenter uses some different servers. The data in the groupfile ‘/etc/ansible/group\_vars/raleigh’ for the ‘raleigh’ group might look like:
    - ntp\_server: acme.example.org
    - database\_server: storage.example.org
  + As an advanced use-case, you can create directories named after your groups or hosts, and Ansible will read all the files in these directories. An example with the ‘raleigh’ group:
    - /etc/ansible/group\_vars/raleigh/db\_settings
    - /etc/ansible/group\_vars/raleigh/cluster\_settings

# [Variables](http://docs.ansible.com/ansible/playbooks_variables.html#id13)

* <http://docs.ansible.com/ansible/playbooks_variables.html#passing-variables-on-the-command-line>
* You can then reference a specific field in the dictionary using either bracket notation or dot notation:
  + foo['field1']
  + foo.field1
* Variables Defined in a Playbook
  + vars:
  + http\_port: 80
* YAML syntax requires that if you start a value with {{ foo }} you quote the whole line, since it wants to be sure you aren’t trying to start a YAML dictionary.
* Turning Off Facts
  + - hosts: whatever
  + gather\_facts: no
* Local Facts
  + If a remotely managed system has an /etc/ansible/facts.d directory, any files in this directory ending in .fact, can be JSON, INI, or executable files returning JSON, and these can supply local facts in Ansible.
  + And this data can be accessed in a template/playbook as:
    - {{ ansible\_local.preferences.general.asdf }}
* Fact Caching
  + Ansible 1.8 allows the ability to save facts between playbook runs, but this feature must be manually enabled.
  + To benefit from cached facts, you will want to change the gathering setting to smart or explicit or set gather\_facts to False in most plays.
    - To configure fact caching using jsonfile, enable it in ansible.cfg as follows:
      * [defaults]
      * gathering = smart
      * fact\_caching = jsonfile
      * fact\_caching\_connection = /path/to/cachedir
      * fact\_caching\_timeout = 86400
      * # seconds
* Registered Variables
  + register: foo\_result
  + Registered variables are valid on the host the remainder of the playbook run, which is the same as the lifetime of “facts” in Ansible. Effectively registered variables are just like facts.
* Magic Variables
  + hostvars
    - lets you ask about the variables of another host, including facts that have been gathered about that host. If, at this point, you haven’t talked to that host yet in any play in the playbook or set of playbooks, you can still get the variables, but you will not be able to see the facts.
  + group\_names
    - is a list (array) of all the groups the current host is in
  + groups
    - is a list of all the groups (and hosts) in the inventory.
  + inventory\_hostname
    - is the name of the hostname as configured in Ansible’s inventory host file
  + inventory\_hostname\_short
    - also contains the part up to the first period, without the rest of the domain.
  + play\_hosts
    - is available as a list of hostnames that are in scope for the current play.
  + inventory\_dir
    - is the pathname of the directory holding Ansible’s inventory host file,
  + inventory\_file
    - is the pathname and the filename pointing to the Ansible’s inventory host file.
  + playbook\_dir
    - contains the playbook base directory.
  + role\_path
    - which will return the current role’s pathname (since 1.8). This will only work inside a role.
  + ansible\_check\_mode
    - (added in version 2.1), a boolean magic variable which will be set to True if you run Ansible with --check.
* Variable File Separation
  + vars\_files:
* - /vars/external\_vars.yml
* Passing Variables On The Command Line
  + --extra-vars "version=1.23.45 other\_variable=foo"
  + --extra-vars '{"pacman":"mrs","ghosts":["inky","pinky","clyde","sue"]}'
  + --extra-vars "@some\_file.json"
  + --extra-vars "@some\_file.yaml"
* Variable Precedence
  + the last listed variables winning prioritization):
* role defaults [[1]](http://docs.ansible.com/ansible/playbooks_variables.html#id9)
* inventory vars [[2]](http://docs.ansible.com/ansible/playbooks_variables.html#id10)
* inventory group\_vars
* inventory host\_vars
* playbook group\_vars
* playbook host\_vars
* host facts
* play vars
* play vars\_prompt
* play vars\_files
* registered vars
* set\_facts
* role and include vars
* block vars (only for tasks in block)
* task vars (only for the task)
* extra vars (always win precedence)
  + Operating System and Distribution Variance
    - playbook
      * ---
      * # talk to all hosts just so we can learn about them
      * - hosts: all
      * tasks:
      * - group\_by: key=os\_{{ ansible\_distribution }}
      * # now just on the CentOS hosts...
      * - hosts: os\_CentOS
      * gather\_facts: False
      * tasks:
      * - # tasks that only happen on CentOS go here
    - If group-specific settings are needed, this can also be done. For example:
      * ---
      * # file: group\_vars/all
      * asdf: 10
      * ---
      * # file: group\_vars/os\_CentOS
      * asdf: 42
    - Alternatively, if only variables are needed:
      * - hosts: all
      * tasks:
      * - include\_vars: "os\_{{ ansible\_distribution }}.yml"
      * - debug: var=asdf
  + Best Practices
    - <http://docs.ansible.com/ansible/playbooks_best_practices.html#staging-vs-production>