Variables in Ansible

While automation exists to make it easier to make things repeatable, all systems are not exactly alike; some may require configuration that is slightly different from others. In some instances, the observed behavior or state of one system might influence how you configure other systems. For example, you might need to find out the IP address of a system and use it as a configuration value on another system.

Ansible uses *variables* to help deal with differences between systems.

To understand variables you’ll also want to read [Conditionals](https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html#playbooks-conditionals) and [Loops](https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#playbooks-loops). Useful things like the **group\_by** module and the whenconditional can also be used with variables, and to help manage differences between systems.

The [ansible-examples github repository](https://github.com/ansible/ansible-examples) contains many examples of how variables are used in Ansible.

[**Creating valid variable names**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id19)

Before you start using variables, it’s important to know what are 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.

YAML also supports dictionaries which map keys to values. For instance:

foo**:**

field1**:** one

field2**:** two

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

foo['field1']

foo.field1

These will both reference the same value (“one”). However, if you choose to use dot notation be aware that some keys can cause problems because they collide with attributes and methods of python dictionaries. You should use bracket notation instead of dot notation if you use keys which start and end with two underscores (Those are reserved for special meanings in python) or are any of the known public attributes:

add, append, as\_integer\_ratio, bit\_length, capitalize, center, clear, conjugate, copy, count, decode, denominator, difference, difference\_update, discard, encode, endswith, expandtabs, extend, find, format, fromhex, fromkeys, get, has\_key, hex, imag, index, insert, intersection, intersection\_update, isalnum, isalpha, isdecimal, isdigit, isdisjoint, is\_integer, islower, isnumeric, isspace, issubset, issuperset, istitle, isupper, items, iteritems, iterkeys, itervalues, join, keys, ljust, lower, lstrip, numerator, partition, pop, popitem, real, remove, replace, reverse, rfind, rindex, rjust, rpartition, rsplit, rstrip, setdefault, sort, split, splitlines, startswith, strip, swapcase, symmetric\_difference, symmetric\_difference\_update, title, translate, union, update, upper, values, viewitems, viewkeys, viewvalues, zfill.

[**Defining variables in inventory**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id20)

Often you’ll want to set variables for an individual host, or for a group of hosts in your inventory. For instance, machines in Boston may all use ‘boston.ntp.example.com’ as an NTP server. The [Working with Inventory](https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#intro-inventory) page has details on setting [Assigning a variable to one machine: host variables](https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#host-variables) and [Assigning a variable to many machines: group variables](https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#group-variables) in inventory.

[**Defining variables in a playbook**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id21)

You can define variables directly in a playbook:

**-** hosts**:** webservers

vars**:**

http\_port**:** 80

This can be nice as it’s right there when you are reading the playbook.

[**Defining variables in included files and roles**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id22)

As described in [Roles](https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html#playbooks-reuse-roles), variables can also be included in the playbook via include files, which may or may not be part of an Ansible Role. Usage of roles is preferred as it provides a nice organizational system.

[**Using variables with Jinja2**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id23)

Once you’ve defined variables, you can use them in your playbooks using the Jinja2 templating system. Here’s a simple Jinja2 template:

My amp goes to {{ **max\_amp\_value** }}

This expression provides the most basic form of variable substitution.

You can use the same syntax in playbooks. For example:

template**:** src=foo.cfg.j2 dest={{ **remote\_install\_path** }}/foo.cfg

Here the variable defines the location of a file, which can vary from one system to another.

Inside a template you automatically have access to all variables that are in scope for a host. Actually it’s more than that – you can also read variables about other hosts. We’ll show how to do that in a bit.

**Note**

ansible allows Jinja2 loops and conditionals in templates, but in playbooks, we do not use them. Ansible playbooks are pure machine-parseable YAML. This is a rather important feature as it means it is possible to code-generate pieces of files, or to have other ecosystem tools read Ansible files. Not everyone will need this but it can unlock possibilities.

**See also**

[**Templating (Jinja2)**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_templating.html#playbooks-templating)

More information about Jinja2 templating

[**Transforming variables with Jinja2 filters**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id24)

Jinja2 filters let you transform the value of a variable within a template expression. For example, the capitalize filter capitalizes any value passed to it; the to\_yaml and to\_json filters change the format of your variable values. Jinja2 includes many [built-in filters](http://jinja.pocoo.org/docs/templates/#builtin-filters) and Ansible supplies [many more filters](https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html#playbooks-filters).

[**Hey wait, a YAML gotcha**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id25)

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. This is covered on the [YAML Syntax](https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html#yaml-syntax) documentation.

This won’t work:

**-** hosts**:** app\_servers

vars**:**

app\_path**:** {{ **base\_path** }}/22

Do it like this and you’ll be fine:

**-** hosts**:** app\_servers

vars**:**

app\_path**:** "{{ **base\_path** }}/22"

[**Variables discovered from systems: Facts**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id26)

There are other places where variables can come from, but these are a type of variable that are discovered, not set by the user.

Facts are information derived from speaking with your remote systems. You can find a complete set under the ansible\_facts variable, most facts are also ‘injected’ as top level variables preserving the ansible\_ prefix, but some are dropped due to conflicts. This can be disabled via the [INJECT\_FACTS\_AS\_VARS](https://docs.ansible.com/ansible/latest/reference_appendices/config.html#inject-facts-as-vars) setting.

An example of this might be the IP address of the remote host, or what the operating system is.

To see what information is available, try the following in a play:

**-** debug**:** var=ansible\_facts

To see the ‘raw’ information as gathered:

ansible hostname -m setup

This will return a large amount of variable data, which may look like this on Ansible 2.7:

**{**

"ansible\_all\_ipv4\_addresses"**:** **[**

"REDACTED IP ADDRESS"

**],**

"ansible\_all\_ipv6\_addresses"**:** **[**

"REDACTED IPV6 ADDRESS"

**],**

"ansible\_apparmor"**:** **{**

"status"**:** "disabled"

**},**

"ansible\_architecture"**:** "x86\_64"**,**

"ansible\_bios\_date"**:** "11/28/2013"**,**

"ansible\_bios\_version"**:** "4.1.5"**,**

"ansible\_cmdline"**:** **{**

"BOOT\_IMAGE"**:** "/boot/vmlinuz-3.10.0-862.14.4.el7.x86\_64"**,**

"console"**:** "ttyS0,115200"**,**

"no\_timer\_check"**:** **true,**

"nofb"**:** **true,**

"nomodeset"**:** **true,**

"ro"**:** **true,**

"root"**:** "LABEL=cloudimg-rootfs"**,**

"vga"**:** "normal"

**},**

"ansible\_date\_time"**:** **{**

"date"**:** "2018-10-25"**,**

"day"**:** "25"**,**

"epoch"**:** "1540469324"**,**

"hour"**:** "12"**,**

"iso8601"**:** "2018-10-25T12:08:44Z"**,**

"iso8601\_basic"**:** "20181025T120844109754"**,**

"iso8601\_basic\_short"**:** "20181025T120844"**,**

"iso8601\_micro"**:** "2018-10-25T12:08:44.109968Z"**,**

"minute"**:** "08"**,**

"month"**:** "10"**,**

"second"**:** "44"**,**

"time"**:** "12:08:44"**,**

"tz"**:** "UTC"**,**

"tz\_offset"**:** "+0000"**,**

"weekday"**:** "Thursday"**,**

"weekday\_number"**:** "4"**,**

"weeknumber"**:** "43"**,**

"year"**:** "2018"

**},**

"ansible\_default\_ipv4"**:** **{**

"address"**:** "REDACTED"**,**

"alias"**:** "eth0"**,**

"broadcast"**:** "REDACTED"**,**

"gateway"**:** "REDACTED"**,**

"interface"**:** "eth0"**,**

"macaddress"**:** "REDACTED"**,**

"mtu"**:** 1500**,**

"netmask"**:** "255.255.255.0"**,**

"network"**:** "REDACTED"**,**

"type"**:** "ether"

**},**

"ansible\_default\_ipv6"**:** **{},**

"ansible\_device\_links"**:** **{**

"ids"**:** **{},**

"labels"**:** **{**

"xvda1"**:** **[**

"cloudimg-rootfs"

**],**

"xvdd"**:** **[**

"config-2"

**]**

**},**

"masters"**:** **{},**

"uuids"**:** **{**

"xvda1"**:** **[**

"cac81d61-d0f8-4b47-84aa-b48798239164"

**],**

"xvdd"**:** **[**

"2018-10-25-12-05-57-00"

**]**

**}**

**},**

"ansible\_devices"**:** **{**

"xvda"**:** **{**

"holders"**:** **[],**

"host"**:** ""**,**

"links"**:** **{**

"ids"**:** **[],**

"labels"**:** **[],**

"masters"**:** **[],**

"uuids"**:** **[]**

**},**

"model"**:** **null,**

"partitions"**:** **{**

"xvda1"**:** **{**

"holders"**:** **[],**

"links"**:** **{**

"ids"**:** **[],**

"labels"**:** **[**

"cloudimg-rootfs"

**],**

"masters"**:** **[],**

"uuids"**:** **[**

"cac81d61-d0f8-4b47-84aa-b48798239164"

**]**

**},**

"sectors"**:** "83883999"**,**

"sectorsize"**:** 512**,**

"size"**:** "40.00 GB"**,**

"start"**:** "2048"**,**

"uuid"**:** "cac81d61-d0f8-4b47-84aa-b48798239164"

**}**

**},**

"removable"**:** "0"**,**

"rotational"**:** "0"**,**

"sas\_address"**:** **null,**

"sas\_device\_handle"**:** **null,**

"scheduler\_mode"**:** "deadline"**,**

"sectors"**:** "83886080"**,**

"sectorsize"**:** "512"**,**

"size"**:** "40.00 GB"**,**

"support\_discard"**:** "0"**,**

"vendor"**:** **null,**

"virtual"**:** 1

**},**

"xvdd"**:** **{**

"holders"**:** **[],**

"host"**:** ""**,**

"links"**:** **{**

"ids"**:** **[],**

"labels"**:** **[**

"config-2"

**],**

"masters"**:** **[],**

"uuids"**:** **[**

"2018-10-25-12-05-57-00"

**]**

**},**

"model"**:** **null,**

"partitions"**:** **{},**

"removable"**:** "0"**,**

"rotational"**:** "0"**,**

"sas\_address"**:** **null,**

"sas\_device\_handle"**:** **null,**

"scheduler\_mode"**:** "deadline"**,**

"sectors"**:** "131072"**,**

"sectorsize"**:** "512"**,**

"size"**:** "64.00 MB"**,**

"support\_discard"**:** "0"**,**

"vendor"**:** **null,**

"virtual"**:** 1

**},**

"xvde"**:** **{**

"holders"**:** **[],**

"host"**:** ""**,**

"links"**:** **{**

"ids"**:** **[],**

"labels"**:** **[],**

"masters"**:** **[],**

"uuids"**:** **[]**

**},**

"model"**:** **null,**

"partitions"**:** **{**

"xvde1"**:** **{**

"holders"**:** **[],**

"links"**:** **{**

"ids"**:** **[],**

"labels"**:** **[],**

"masters"**:** **[],**

"uuids"**:** **[]**

**},**

"sectors"**:** "167770112"**,**

"sectorsize"**:** 512**,**

"size"**:** "80.00 GB"**,**

"start"**:** "2048"**,**

"uuid"**:** **null**

**}**

**},**

"removable"**:** "0"**,**

"rotational"**:** "0"**,**

"sas\_address"**:** **null,**

"sas\_device\_handle"**:** **null,**

"scheduler\_mode"**:** "deadline"**,**

"sectors"**:** "167772160"**,**

"sectorsize"**:** "512"**,**

"size"**:** "80.00 GB"**,**

"support\_discard"**:** "0"**,**

"vendor"**:** **null,**

"virtual"**:** 1

**}**

**},**

"ansible\_distribution"**:** "CentOS"**,**

"ansible\_distribution\_file\_parsed"**:** **true,**

"ansible\_distribution\_file\_path"**:** "/etc/redhat-release"**,**

"ansible\_distribution\_file\_variety"**:** "RedHat"**,**

"ansible\_distribution\_major\_version"**:** "7"**,**

"ansible\_distribution\_release"**:** "Core"**,**

"ansible\_distribution\_version"**:** "7.5.1804"**,**

"ansible\_dns"**:** **{**

"nameservers"**:** **[**

"127.0.0.1"

**]**

**},**

"ansible\_domain"**:** ""**,**

"ansible\_effective\_group\_id"**:** 1000**,**

"ansible\_effective\_user\_id"**:** 1000**,**

"ansible\_env"**:** **{**

"HOME"**:** "/home/zuul"**,**

"LANG"**:** "en\_US.UTF-8"**,**

"LESSOPEN"**:** "||/usr/bin/lesspipe.sh %s"**,**

"LOGNAME"**:** "zuul"**,**

"MAIL"**:** "/var/mail/zuul"**,**

"PATH"**:** "/usr/local/bin:/usr/bin"**,**

"PWD"**:** "/home/zuul"**,**

"SELINUX\_LEVEL\_REQUESTED"**:** ""**,**

"SELINUX\_ROLE\_REQUESTED"**:** ""**,**

"SELINUX\_USE\_CURRENT\_RANGE"**:** ""**,**

"SHELL"**:** "/bin/bash"**,**

"SHLVL"**:** "2"**,**

"SSH\_CLIENT"**:** "REDACTED 55672 22"**,**

"SSH\_CONNECTION"**:** "REDACTED 55672 REDACTED 22"**,**

"USER"**:** "zuul"**,**

"XDG\_RUNTIME\_DIR"**:** "/run/user/1000"**,**

"XDG\_SESSION\_ID"**:** "1"**,**

"\_"**:** "/usr/bin/python2"

**},**

"ansible\_eth0"**:** **{**

"active"**:** **true,**

"device"**:** "eth0"**,**

"ipv4"**:** **{**

"address"**:** "REDACTED"**,**

"broadcast"**:** "REDACTED"**,**

"netmask"**:** "255.255.255.0"**,**

"network"**:** "REDACTED"

**},**

"ipv6"**:** **[**

**{**

"address"**:** "REDACTED"**,**

"prefix"**:** "64"**,**

"scope"**:** "link"

**}**

**],**

"macaddress"**:** "REDACTED"**,**

"module"**:** "xen\_netfront"**,**

"mtu"**:** 1500**,**

"pciid"**:** "vif-0"**,**

"promisc"**:** **false,**

"type"**:** "ether"

**},**

"ansible\_eth1"**:** **{**

"active"**:** **true,**

"device"**:** "eth1"**,**

"ipv4"**:** **{**

"address"**:** "REDACTED"**,**

"broadcast"**:** "REDACTED"**,**

"netmask"**:** "255.255.224.0"**,**

"network"**:** "REDACTED"

**},**

"ipv6"**:** **[**

**{**

"address"**:** "REDACTED"**,**

"prefix"**:** "64"**,**

"scope"**:** "link"

**}**

**],**

"macaddress"**:** "REDACTED"**,**

"module"**:** "xen\_netfront"**,**

"mtu"**:** 1500**,**

"pciid"**:** "vif-1"**,**

"promisc"**:** **false,**

"type"**:** "ether"

**},**

"ansible\_fips"**:** **false,**

"ansible\_form\_factor"**:** "Other"**,**

"ansible\_fqdn"**:** "centos-7-rax-dfw-0003427354"**,**

"ansible\_hostname"**:** "centos-7-rax-dfw-0003427354"**,**

"ansible\_interfaces"**:** **[**

"lo"**,**

"eth1"**,**

"eth0"

**],**

"ansible\_is\_chroot"**:** **false,**

"ansible\_kernel"**:** "3.10.0-862.14.4.el7.x86\_64"**,**

"ansible\_lo"**:** **{**

"active"**:** **true,**

"device"**:** "lo"**,**

"ipv4"**:** **{**

"address"**:** "127.0.0.1"**,**

"broadcast"**:** "host"**,**

"netmask"**:** "255.0.0.0"**,**

"network"**:** "127.0.0.0"

**},**

"ipv6"**:** **[**

**{**

"address"**:** "::1"**,**

"prefix"**:** "128"**,**

"scope"**:** "host"

**}**

**],**

"mtu"**:** 65536**,**

"promisc"**:** **false,**

"type"**:** "loopback"

**},**

"ansible\_local"**:** **{},**

"ansible\_lsb"**:** **{**

"codename"**:** "Core"**,**

"description"**:** "CentOS Linux release 7.5.1804 (Core)"**,**

"id"**:** "CentOS"**,**

"major\_release"**:** "7"**,**

"release"**:** "7.5.1804"

**},**

"ansible\_machine"**:** "x86\_64"**,**

"ansible\_machine\_id"**:** "2db133253c984c82aef2fafcce6f2bed"**,**

"ansible\_memfree\_mb"**:** 7709**,**

"ansible\_memory\_mb"**:** **{**

"nocache"**:** **{**

"free"**:** 7804**,**

"used"**:** 173

**},**

"real"**:** **{**

"free"**:** 7709**,**

"total"**:** 7977**,**

"used"**:** 268

**},**

"swap"**:** **{**

"cached"**:** 0**,**

"free"**:** 0**,**

"total"**:** 0**,**

"used"**:** 0

**}**

**},**

"ansible\_memtotal\_mb"**:** 7977**,**

"ansible\_mounts"**:** **[**

**{**

"block\_available"**:** 7220998**,**

"block\_size"**:** 4096**,**

"block\_total"**:** 9817227**,**

"block\_used"**:** 2596229**,**

"device"**:** "/dev/xvda1"**,**

"fstype"**:** "ext4"**,**

"inode\_available"**:** 10052341**,**

"inode\_total"**:** 10419200**,**

"inode\_used"**:** 366859**,**

"mount"**:** "/"**,**

"options"**:** "rw,seclabel,relatime,data=ordered"**,**

"size\_available"**:** 29577207808**,**

"size\_total"**:** 40211361792**,**

"uuid"**:** "cac81d61-d0f8-4b47-84aa-b48798239164"

**},**

**{**

"block\_available"**:** 0**,**

"block\_size"**:** 2048**,**

"block\_total"**:** 252**,**

"block\_used"**:** 252**,**

"device"**:** "/dev/xvdd"**,**

"fstype"**:** "iso9660"**,**

"inode\_available"**:** 0**,**

"inode\_total"**:** 0**,**

"inode\_used"**:** 0**,**

"mount"**:** "/mnt/config"**,**

"options"**:** "ro,relatime,mode=0700"**,**

"size\_available"**:** 0**,**

"size\_total"**:** 516096**,**

"uuid"**:** "2018-10-25-12-05-57-00"

**}**

**],**

"ansible\_nodename"**:** "centos-7-rax-dfw-0003427354"**,**

"ansible\_os\_family"**:** "RedHat"**,**

"ansible\_pkg\_mgr"**:** "yum"**,**

"ansible\_processor"**:** **[**

"0"**,**

"GenuineIntel"**,**

"Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz"**,**

"1"**,**

"GenuineIntel"**,**

"Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz"**,**

"2"**,**

"GenuineIntel"**,**

"Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz"**,**

"3"**,**

"GenuineIntel"**,**

"Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz"**,**

"4"**,**

"GenuineIntel"**,**

"Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz"**,**

"5"**,**

"GenuineIntel"**,**

"Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz"**,**

"6"**,**

"GenuineIntel"**,**

"Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz"**,**

"7"**,**

"GenuineIntel"**,**

"Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz"

**],**

"ansible\_processor\_cores"**:** 8**,**

"ansible\_processor\_count"**:** 8**,**

"ansible\_processor\_threads\_per\_core"**:** 1**,**

"ansible\_processor\_vcpus"**:** 8**,**

"ansible\_product\_name"**:** "HVM domU"**,**

"ansible\_product\_serial"**:** "REDACTED"**,**

"ansible\_product\_uuid"**:** "REDACTED"**,**

"ansible\_product\_version"**:** "4.1.5"**,**

"ansible\_python"**:** **{**

"executable"**:** "/usr/bin/python2"**,**

"has\_sslcontext"**:** **true,**

"type"**:** "CPython"**,**

"version"**:** **{**

"major"**:** 2**,**

"micro"**:** 5**,**

"minor"**:** 7**,**

"releaselevel"**:** "final"**,**

"serial"**:** 0

**},**

"version\_info"**:** **[**

2**,**

7**,**

5**,**

"final"**,**

0

**]**

**},**

"ansible\_python\_version"**:** "2.7.5"**,**

"ansible\_real\_group\_id"**:** 1000**,**

"ansible\_real\_user\_id"**:** 1000**,**

"ansible\_selinux"**:** **{**

"config\_mode"**:** "enforcing"**,**

"mode"**:** "enforcing"**,**

"policyvers"**:** 31**,**

"status"**:** "enabled"**,**

"type"**:** "targeted"

**},**

"ansible\_selinux\_python\_present"**:** **true,**

"ansible\_service\_mgr"**:** "systemd"**,**

"ansible\_ssh\_host\_key\_ecdsa\_public"**:** "REDACTED KEY VALUE"**,**

"ansible\_ssh\_host\_key\_ed25519\_public"**:** "REDACTED KEY VALUE"**,**

"ansible\_ssh\_host\_key\_rsa\_public"**:** "REDACTED KEY VALUE"**,**

"ansible\_swapfree\_mb"**:** 0**,**

"ansible\_swaptotal\_mb"**:** 0**,**

"ansible\_system"**:** "Linux"**,**

"ansible\_system\_capabilities"**:** **[**

""

**],**

"ansible\_system\_capabilities\_enforced"**:** "True"**,**

"ansible\_system\_vendor"**:** "Xen"**,**

"ansible\_uptime\_seconds"**:** 151**,**

"ansible\_user\_dir"**:** "/home/zuul"**,**

"ansible\_user\_gecos"**:** ""**,**

"ansible\_user\_gid"**:** 1000**,**

"ansible\_user\_id"**:** "zuul"**,**

"ansible\_user\_shell"**:** "/bin/bash"**,**

"ansible\_user\_uid"**:** 1000**,**

"ansible\_userspace\_architecture"**:** "x86\_64"**,**

"ansible\_userspace\_bits"**:** "64"**,**

"ansible\_virtualization\_role"**:** "guest"**,**

"ansible\_virtualization\_type"**:** "xen"**,**

"gather\_subset"**:** **[**

"all"

**],**

"module\_setup"**:** **true**

**}**

In the above the model of the first disk may be referenced in a template or playbook as:

{{ **ansible\_facts[**'devices'**][**'xvda'**][**'model'**]** }}

Similarly, the hostname as the system reports it is:

{{ **ansible\_facts[**'nodename'**]** }}

Facts are frequently used in conditionals (see [Conditionals](https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html#playbooks-conditionals)) and also in templates.

Facts can be also used to create dynamic groups of hosts that match particular criteria, see the [Importing Modules](https://docs.python.org/3/library/modules.html#modules) documentation on **group\_by** for details, as well as in generalized conditional statements as discussed in the [Conditionals](https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html#playbooks-conditionals) chapter.

[**Disabling facts**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id27)

If you know you don’t need any fact data about your hosts, and know everything about your systems centrally, you can turn off fact gathering. This has advantages in scaling Ansible in push mode with very large numbers of systems, mainly, or if you are using Ansible on experimental platforms. In any play, just do this:

**-** hosts**:** whatever

gather\_facts**:** no

[**Local facts (facts.d)**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id28)

*New in version 1.3.*

As discussed in the playbooks chapter, Ansible facts are a way of getting data about remote systems for use in playbook variables.

Usually these are discovered automatically by the setup module in Ansible. Users can also write custom facts modules, as described in the API guide. However, what if you want to have a simple way to provide system or user provided data for use in Ansible variables, without writing a fact module?

“Facts.d” is one mechanism for users to control some aspect of how their systems are managed.

**Note**

Perhaps “local facts” is a bit of a misnomer, it means “locally supplied user values” as opposed to “centrally supplied user values”, or what facts are – “locally dynamically determined values”.

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. An alternate directory can be specified using the fact\_pathplay keyword.

For example, assume /etc/ansible/facts.d/preferences.fact contains:

**[general]**

asdf=1

bar=2

This will produce a hash variable fact named general with asdf and bar as members. To validate this, run the following:

ansible <hostname> -m setup -a "filter=ansible\_local"

And you will see the following fact added:

"ansible\_local"**:** **{**

"preferences"**:** **{**

"general"**:** **{**

"asdf" **:** "1"**,**

"bar" **:** "2"

**}**

**}**

**}**

And this data can be accessed in a template/playbook as:

{{ **ansible\_local[**'preferences'**][**'general'**][**'asdf'**]** }}

The local namespace prevents any user supplied fact from overriding system facts or variables defined elsewhere in the playbook.

**Note**

The key part in the key=value pairs will be converted into lowercase inside the ansible\_local variable. Using the example above, if the ini file contained XYZ=3 in the [general] section, then you should expect to access it as: {{ ansible\_local['preferences']['general']['xyz'] }} and not {{ ansible\_local['preferences']['general']['XYZ'] }}. This is because Ansible uses Python’s [ConfigParser](https://docs.python.org/2/library/configparser.html) which passes all option names through the [optionxform](https://docs.python.org/2/library/configparser.html" \l "ConfigParser.RawConfigParser.optionxform) method and this method’s default implementation converts option names to lower case.

If you have a playbook that is copying over a custom fact and then running it, making an explicit call to re-run the setup module can allow that fact to be used during that particular play. Otherwise, it will be available in the next play that gathers fact information. Here is an example of what that might look like:

**-** hosts**:** webservers

tasks**:**

**-** name**:** create directory for ansible custom facts

file**:** state=directory recurse=yes path=/etc/ansible/facts.d

**-** name**:** install custom ipmi fact

copy**:** src=ipmi.fact dest=/etc/ansible/facts.d

**-** name**:** re-read facts after adding custom fact

setup**:** filter=ansible\_local

In this pattern however, you could also write a fact module as well, and may wish to consider this as an option.

[**Ansible version**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id29)

*New in version 1.8.*

To adapt playbook behavior to specific version of ansible, a variable ansible\_version is available, with the following structure:

"ansible\_version"**:** **{**

"full"**:** "2.0.0.2"**,**

"major"**:** **2,**

"minor"**:** **0,**

"revision"**:** **0,**

"string"**:** "2.0.0.2"

**}**

[**Caching Facts**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id30)

*New in version 1.8.*

As shown elsewhere in the docs, it is possible for one server to reference variables about another, like so:

{{ **hostvars[**'asdf.example.com'**][**'ansible\_facts'**][**'os\_family'**]** }}

With “Fact Caching” disabled, in order to do this, Ansible must have already talked to ‘asdf.example.com’ in the current play, or another play up higher in the playbook. This is the default configuration of ansible.

To avoid this, Ansible 1.8 allows the ability to save facts between playbook runs, but this feature must be manually enabled. Why might this be useful?

With a very large infrastructure with thousands of hosts, fact caching could be configured to run nightly. Configuration of a small set of servers could run ad-hoc or periodically throughout the day. With fact caching enabled, it would not be necessary to “hit” all servers to reference variables and information about them.

With fact caching enabled, it is possible for machine in one group to reference variables about machines in the other group, despite the fact that they have not been communicated with in the current execution of /usr/bin/ansible-playbook.

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.

Currently, Ansible ships with two persistent cache plugins: redis and jsonfile.

To configure fact caching using redis, enable it in ansible.cfg as follows:

**[defaults]**

gathering = smart

fact\_caching = redis

fact\_caching\_timeout = 86400

# seconds

To get redis up and running, perform the equivalent OS commands:

yum install redis

service redis start

pip install redis

Note that the Python redis library should be installed from pip, the version packaged in EPEL is too old for use by Ansible.

In current embodiments, this feature is in beta-level state and the Redis plugin does not support port or password configuration, this is expected to change in the near future.

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

fact\_caching\_connection is a local filesystem path to a writeable directory (ansible will attempt to create the directory if one does not exist).

fact\_caching\_timeout is the number of seconds to cache the recorded facts.

[**Registering variables**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id31)

Another major use of variables is running a command and registering the result of that command as a variable. When you execute a task and save the return value in a variable for use in later tasks, you create a registered variable. There are more examples of this in the [Conditionals](https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html#playbooks-conditionals) chapter.

For example:

**-** hosts**:** web\_servers

tasks**:**

**-** shell**:** /usr/bin/foo

register**:** foo\_result

ignore\_errors**:** True

**-** shell**:** /usr/bin/bar

when**:** foo\_result.rc == 5

Results will vary from module to module. Each module’s documentation includes a RETURN section describing that module’s return values. To see the values for a particular task, run your playbook with -v.

Registered variables are similar to facts, with a few key differences. Like facts, registered variables are host-level variables. However, registered variables are only stored in memory. (Ansible facts are backed by whatever cache plugin you have configured.) Registered variables are only valid on the host for the rest of the current playbook run. Finally, registered variables and facts have different [precedence levels](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#ansible-variable-precedence).

When you register a variable in a task with a loop, the registered variable contains a value for each item in the loop. The data structure placed in the variable during the loop will contain a results attribute, that is a list of all responses from the module. For a more in-depth example of how this works, see the [Loops](https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#playbooks-loops) section on using register with a loop.

**Note**

If a task fails or is skipped, the variable still is registered with a failure or skipped status, the only way to avoid registering a variable is using tags.

[**Accessing complex variable data**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id32)

We already described facts a little higher up in the documentation.

Some provided facts, like networking information, are made available as nested data structures. To access them a simple {{ foo }} is not sufficient, but it is still easy to do. Here’s how we get an IP address:

{{ **ansible\_facts[**"eth0"**][**"ipv4"**][**"address"**]** }}

OR alternatively:

{{ **ansible\_facts.eth0.ipv4.address** }}

Similarly, this is how we access the first element of an array:

{{ **foo[**0**]** }}

[**Accessing information about other hosts with magic variables**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id33)

Whether or not you define any variables, you can access information about your hosts with the [Special Variables](https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html#special-variables) Ansible provides, including “magic” variables, facts, and connection variables. Magic variable names are reserved - do not set variables with these names. The variable environment is also reserved.

The most commonly used magic variables are hostvars, groups, group\_names, and inventory\_hostname.

hostvars lets you access variables for another host, including facts that have been gathered about that host. You can access host variables at any point in a playbook. Even if you haven’t connected 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.

If your database server wants to use the value of a ‘fact’ from another node, or an inventory variable assigned to another node, it’s easy to do so within a template or even an action line:

{{ **hostvars[**'test.example.com'**][**'ansible\_facts'**][**'distribution'**]** }}

groups is a list of all the groups (and hosts) in the inventory. This can be used to enumerate all hosts within a group. For example:

{% **for** **host** **in** **groups[**'app\_servers'**]** %}

# something that applies to all app servers.

{% **endfor** %}

A frequently used idiom is walking a group to find all IP addresses in that group.

{% **for** **host** **in** **groups[**'app\_servers'**]** %}

{{ **hostvars[host][**'ansible\_facts'**][**'eth0'**][**'ipv4'**][**'address'**]** }}

{% **endfor** %}

You can use this idiom to point a frontend proxy server to all of the app servers, to set up the correct firewall rules between servers, etc. You need to make sure that the facts of those hosts have been populated before though, for example by running a play against them if the facts have not been cached recently (fact caching was added in Ansible 1.8).

group\_names is a list (array) of all the groups the current host is in. This can be used in templates using Jinja2 syntax to make template source files that vary based on the group membership (or role) of the host:

{% **if** 'webserver' **in** **group\_names** %}

# some part of a configuration file that only applies to webservers

{% **endif** %}

inventory\_hostname is the name of the hostname as configured in Ansible’s inventory host file. This can be useful when you’ve disabled fact-gathering, or you don’t want to rely on the discovered hostname ansible\_hostname. If you have a long FQDN, you can use inventory\_hostname\_short, which contains the part up to the first period, without the rest of the domain.

Other useful magic variables refer to the current play or playbook, including:

*New in version 2.2.*

ansible\_play\_hosts is the full list of all hosts still active in the current play.

*New in version 2.2.*

ansible\_play\_batch is available as a list of hostnames that are in scope for the current ‘batch’ of the play. The batch size is defined by serial, when not set it is equivalent to the whole play (making it the same as ansible\_play\_hosts).

*New in version 2.3.*

ansible\_playbook\_python is the path to the python executable used to invoke the Ansible command line tool.

These vars may be useful for filling out templates with multiple hostnames or for injecting the list into the rules for a load balancer.

Also available, 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.

We then have role\_path which will return the current role’s pathname (since 1.8). This will only work inside a role.

And finally, ansible\_check\_mode (added in version 2.1), a boolean magic variable which will be set to True if you run Ansible with --check.

[**Defining variables in files**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id34)

It’s a great idea to keep your playbooks under source control, but you may wish to make the playbook source public while keeping certain important variables private. Similarly, sometimes you may just want to keep certain information in different files, away from the main playbook.

You can do this by using an external variables file, or files, just like this:

**---**

**-** hosts**:** all

remote\_user**:** root

vars**:**

favcolor**:** blue

vars\_files**:**

**-** /vars/external\_vars.yml

tasks**:**

**-** name**:** this is just a placeholder

command**:** /bin/echo foo

This removes the risk of sharing sensitive data with others when sharing your playbook source with them.

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

**---**

*# in the above example, this would be vars/external\_vars.yml*

somevar**:** somevalue

password**:** magic

**Note**

It’s also possible to keep per-host and per-group variables in very similar files, this is covered in [Organizing host and group variables](https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#splitting-out-vars).

[**Passing variables on the command line**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id35)

In addition to vars\_prompt and vars\_files, it is possible to set variables at the command line using the --extra-vars (or -e) argument. Variables can be defined using a single quoted string (containing one or more variables) using one of the formats below

key=value format:

ansible-playbook release.yml --extra-vars "version=1.23.45 other\_variable=foo"

**Note**

Values passed in using the key=value syntax are interpreted as strings. Use the JSON format if you need to pass in anything that shouldn’t be a string (Booleans, integers, floats, lists etc).

JSON string format:

ansible-playbook release.yml --extra-vars '{"version":"1.23.45","other\_variable":"foo"}'

ansible-playbook arcade.yml --extra-vars '{"pacman":"mrs","ghosts":["inky","pinky","clyde","sue"]}'

vars from a JSON or YAML file:

ansible-playbook release.yml --extra-vars "@some\_file.json"

This is useful for, among other things, setting the hosts group or the user for the playbook.

Escaping quotes and other special characters:

Ensure you’re escaping quotes appropriately for both your markup (e.g. JSON), and for the shell you’re operating in.:

ansible-playbook arcade.yml --extra-vars "{\"name\":\"Conan O\'Brien\"}"

ansible-playbook arcade.yml --extra-vars '{"name":"Conan O'\\\''Brien"}'

ansible-playbook script.yml --extra-vars "{\"dialog\":\"He said \\\"I just can\'t get enough of those single and double-quotes"\!"\\\"\"}"

In these cases, it’s probably best to use a JSON or YAML file containing the variable definitions.

[**Variable precedence: Where should I put a variable?**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id36)

A lot of folks may ask about how variables override another. Ultimately it’s Ansible’s philosophy that it’s better you know where to put a variable, and then you have to think about it a lot less.

Avoid defining the variable “x” in 47 places and then ask the question “which x gets used”. Why? Because that’s not Ansible’s Zen philosophy of doing things.

There is only one Empire State Building. One Mona Lisa, etc. Figure out where to define a variable, and don’t make it complicated.

However, let’s go ahead and get precedence out of the way! It exists. It’s a real thing, and you might have a use for it.

If multiple variables of the same name are defined in different places, they get overwritten in a certain order.

Here is the order of precedence from least to greatest (the last listed variables winning prioritization):

1. command line values (eg “-u user”)
2. role defaults [[1]](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id15)
3. inventory file or script group vars [[2]](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id16)
4. inventory group\_vars/all [[3]](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id17)
5. playbook group\_vars/all [[3]](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id17)
6. inventory group\_vars/\* [[3]](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id17)
7. playbook group\_vars/\* [[3]](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id17)
8. inventory file or script host vars [[2]](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id16)
9. inventory host\_vars/\* [[3]](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id17)
10. playbook host\_vars/\* [[3]](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id17)
11. host facts / cached set\_facts [[4]](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id18)
12. play vars
13. play vars\_prompt
14. play vars\_files
15. role vars (defined in role/vars/main.yml)
16. block vars (only for tasks in block)
17. task vars (only for the task)
18. include\_vars
19. set\_facts / registered vars
20. role (and include\_role) params
21. include params
22. extra vars (always win precedence)

Basically, anything that goes into “role defaults” (the defaults folder inside the role) is the most malleable and easily overridden. Anything in the vars directory of the role overrides previous versions of that variable in namespace. The idea here to follow is that the more explicit you get in scope, the more precedence it takes with command line -e extra vars always winning. Host and/or inventory variables can win over role defaults, but not explicit includes like the vars directory or an include\_vars task.

**Footnotes**

|  |  |
| --- | --- |
| [[1]](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id5) | Tasks in each role will see their own role’s defaults. Tasks defined outside of a role will see the last role’s defaults. |
| [2] | *(*[*1*](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id6)*,*[*2*](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id11)*)* Variables defined in inventory file or provided by dynamic inventory. |

|  |  |
| --- | --- |
| [3] | *(*[*1*](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id7)*,*[*2*](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id8)*,*[*3*](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id9)*,*[*4*](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id10)*,*[*5*](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id12)*,*[*6*](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id13)*)* Includes vars added by ‘vars plugins’ as well as host\_vars and group\_vars which are added by the default vars plugin shipped with Ansible. |
| [[4]](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id14) | When created with set\_facts’s cacheable option, variables will have the high precedence in the play, but will be the same as a host facts precedence when they come from the cache. |

**Note**

Within any section, redefining a var will overwrite the previous instance. If multiple groups have the same variable, the last one loaded wins. If you define a variable twice in a play’s vars: section, the second one wins.

**Note**

The previous describes the default config hash\_behaviour=replace, switch to merge to only partially overwrite.

**Note**

Group loading follows parent/child relationships. Groups of the same ‘parent/child’ level are then merged following alphabetical order. This last one can be superceeded by the user via ansible\_group\_priority, which defaults to 1 for all groups. This variable, ansible\_group\_priority, can only be set in the inventory source and not in group\_vars/ as the variable is used in the loading of group\_vars/.

Another important thing to consider (for all versions) is that connection variables override config, command line and play/role/task specific options and keywords. See [Controlling how Ansible behaves: precedence rules](https://docs.ansible.com/ansible/latest/reference_appendices/general_precedence.html#general-precedence-rules) for more details. For example, if your inventory specifies ansible\_user: ramon and you run:

ansible -u lola myhost

This will still connect as ramon because the value from the variable takes priority (in this case, the variable came from the inventory, but the same would be true no matter where the variable was defined).

For plays/tasks this is also true for remote\_user. Assuming the same inventory config, the following play:

**-** hosts**:** myhost

tasks**:**

**-** command**:** I'll connect as ramon still

remote\_user**:** lola

will have the value of remote\_user overwritten by ansible\_user in the inventory.

This is done so host-specific settings can override the general settings. These variables are normally defined per host or group in inventory, but they behave like other variables.

If you want to override the remote user globally (even over inventory) you can use extra vars. For instance, if you run:

ansible... -e "ansible\_user=maria" -u lola

the lola value is still ignored, but ansible\_user=maria takes precedence over all other places where ansible\_user (or remote\_user) might be set.

A connection-specific version of a variable takes precedence over more generic versions. For example, ansible\_ssh\_user specified as a group\_var would have a higher precedence than ansible\_user specified as a host\_var.

You can also override as a normal variable in a play:

**-** hosts**:** all

vars**:**

ansible\_user**:** lola

tasks**:**

**-** command**:** I'll connect as lola!

[**Scoping variables**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id37)

You can decide where to set a variable based on the scope you want that value to have. Ansible has three main scopes:

* Global: this is set by config, environment variables and the command line
* Play: each play and contained structures, vars entries (vars; vars\_files; vars\_prompt), role defaults and vars.
* Host: variables directly associated to a host, like inventory, include\_vars, facts or registered task outputs

[**Examples of where to set a variable**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#id38)

Let’s show some examples and where you would choose to put what based on the kind of control you might want over values.

First off, group variables are powerful.

Site-wide defaults should be defined as a group\_vars/all setting. Group variables are generally placed alongside your inventory file. They can also be returned by a dynamic inventory script (see [Working With Dynamic Inventory](https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html#intro-dynamic-inventory)) or defined in things like [Red Hat Ansible Tower](https://docs.ansible.com/ansible/latest/reference_appendices/tower.html#ansible-tower) from the UI or API:

**---**

*# file: /etc/ansible/group\_vars/all*

*# this is the site wide default*

ntp\_server**:** default-time.example.com

Regional information might be defined in a group\_vars/region variable. If this group is a child of the all group (which it is, because all groups are), it will override the group that is higher up and more general:

**---**

*# file: /etc/ansible/group\_vars/boston*

ntp\_server**:** boston-time.example.com

If for some crazy reason we wanted to tell just a specific host to use a specific NTP server, it would then override the group variable!:

**---**

*# file: /etc/ansible/host\_vars/xyz.boston.example.com*

ntp\_server**:** override.example.com

So that covers inventory and what you would normally set there. It’s a great place for things that deal with geography or behavior. Since groups are frequently the entity that maps roles onto hosts, it is sometimes a shortcut to set variables on the group instead of defining them on a role. You could go either way.

Remember: Child groups override parent groups, and hosts always override their groups.

Next up: learning about role variable precedence.

We’ll pretty much assume you are using roles at this point. You should be using roles for sure. Roles are great. You are using roles aren’t you? Hint hint.

If you are writing a redistributable role with reasonable defaults, put those in the roles/x/defaults/main.yml file. This means the role will bring along a default value but ANYTHING in Ansible will override it. See [Roles](https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html#playbooks-reuse-roles) for more info about this:

**---**

*# file: roles/x/defaults/main.yml*

*# if not overridden in inventory or as a parameter, this is the value that will be used*

http\_port**:** 80

If you are writing a role and want to ensure the value in the role is absolutely used in that role, and is not going to be overridden by inventory, you should put it in roles/x/vars/main.yml like so, and inventory values cannot override it. -e however, still will:

**---**

*# file: roles/x/vars/main.yml*

*# this will absolutely be used in this role*

http\_port**:** 80

This is one way to plug in constants about the role that are always true. If you are not sharing your role with others, app specific behaviors like ports is fine to put in here. But if you are sharing roles with others, putting variables in here might be bad. Nobody will be able to override them with inventory, but they still can by passing a parameter to the role.

Parameterized roles are useful.

If you are using a role and want to override a default, pass it as a parameter to the role like so:

roles**:**

**-** role**:** apache

vars**:**

http\_port**:** 8080

This makes it clear to the playbook reader that you’ve made a conscious choice to override some default in the role, or pass in some configuration that the role can’t assume by itself. It also allows you to pass something site-specific that isn’t really part of the role you are sharing with others.

This can often be used for things that might apply to some hosts multiple times. For example:

roles**:**

**-** role**:** app\_user

vars**:**

myname**:** Ian

**-** role**:** app\_user

vars**:**

myname**:** Terry

**-** role**:** app\_user

vars**:**

myname**:** Graham

**-** role**:** app\_user

vars**:**

myname**:** John

In this example, the same role was invoked multiple times. It’s quite likely there was no default for name supplied at all. Ansible can warn you when variables aren’t defined – it’s the default behavior in fact.

There are a few other things that go on with roles.

Generally speaking, variables set in one role are available to others. This means if you have a roles/common/vars/main.yml you can set variables in there and make use of them in other roles and elsewhere in your playbook:

roles**:**

**-** role**:** common\_settings

**-** role**:** something

vars**:**

foo**:** 12

**-** role**:** something\_else

**Note**

There are some protections in place to avoid the need to namespace variables. In the above, variables defined in common\_settings are most definitely available to ‘something’ and ‘something\_else’ tasks, but if “something’s” guaranteed to have foo set at 12, even if somewhere deep in common settings it set foo to 20.

So, that’s precedence, explained in a more direct way. Don’t worry about precedence, just think about if your role is defining a variable that is a default, or a “live” variable you definitely want to use. Inventory lies in precedence right in the middle, and if you want to forcibly override something, use -e.

If you found that a little hard to understand, take a look at the [ansible-examples](https://github.com/ansible/ansible-examples) repo on GitHub for a bit more about how all of these things can work together.