Creating a basic inventory file

- Inventory is a collection of hosts (nodes) with associated data and groupings that Ansible can connect and manage.
- Ansible uses an inventory file (basically, a list of servers) to communicate with the servers.
- It consists of:
  - Hosts (nodes)
  - Groups
  - Inventory-specific data (variables)
  - Static or dynamic sources

Cont.

- If we are not using port 22 for SSH on the server, we need to add it to the domain address, like www.example.com:2525

- Since, Ansible defaults to port 22

10.42.0.2

10.42.0.6

10.42.0.7

10.42.0.8

10.42.0.100

host.example.com

Static Inventory Example

```
[devserver]
10.42.0.2

[webservers]
10.42.0.[6:8]

[haproxy]
10.42.0.100

[allservers : children]
devserver
Webservers

[all:vars]
ansible_user=ec2-user
```

Client Management / Node Configuration

- To enable client machine authentication, you have to modify the below two properties in **/etc/ssh/sshd_config** of Client/Node machine.

PermitRootLogin yes

PasswordAuthentication yes

- Restart **sshd** service

systemctl restart sshd

Cont.

- On the Management server, we need to generate key using the below commands

  ssh-keygen

  It will ask you for the path - If the key is not generated before use the same path.

  Then, It will prompt for password – Give it Empty

  ssh-copy-id root@<ipaddress of a node>

- It will prompt for password of a node. The authentication key will be copied to the node once the authentication is successful.

Modules

Modules are bits of code transferred to the target system and executed to satisfy the task declaration.

- apt/yum
- copy
- file
- get_url
- git
- ping
- debug
- service
- synchronize
- template

Ansible modules can be written in any language and are only required to take JSON as input and produce JSON as output.

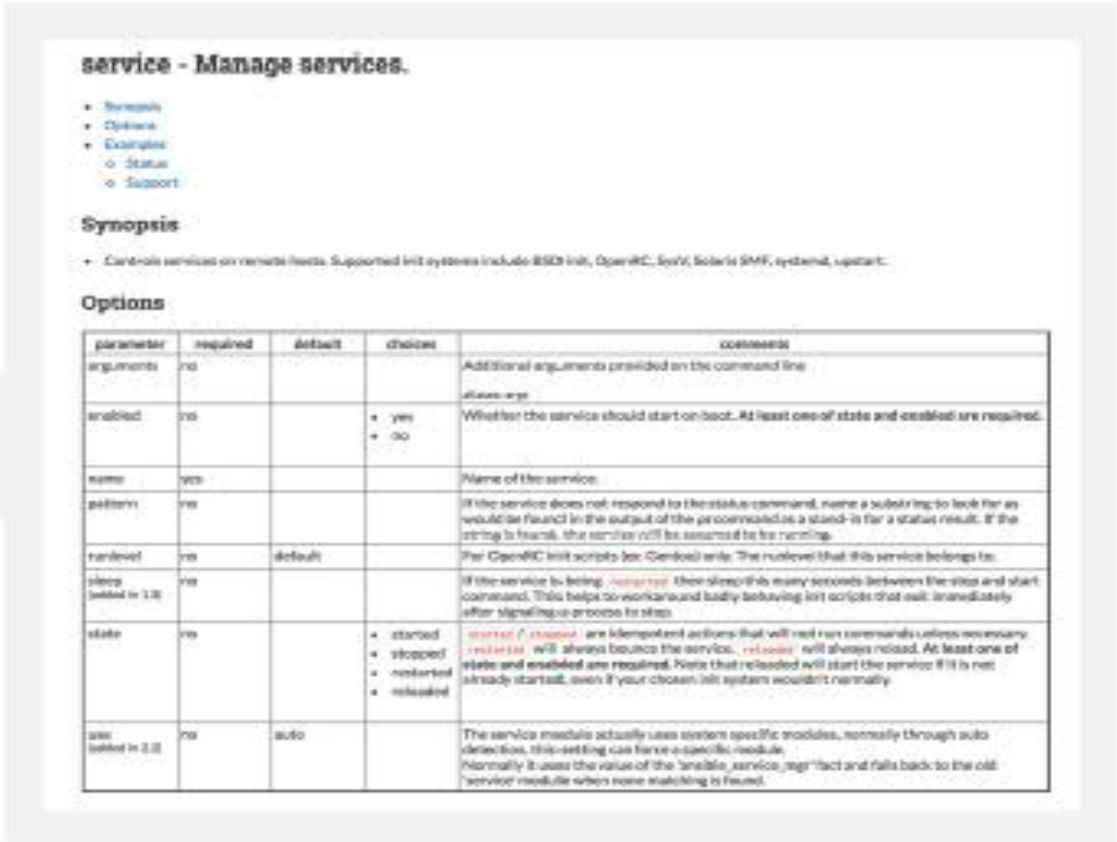
Modules Documentation

Docs » Module Index

Module Index

http://docs.ansible.com/

- All Modules
- Cloud Modules
- Clustering Modules
- Commands Modules
- Crypto Modules
- Database Modules
- Files Modules
- Identity Modules
- Inventory Modules
- Messaging Modules
- Monitoring Modules
- Network Modules
- Notification Modules
- Packaging Modules
- Remote Management Modules
- Source Control Modules
- Storage Modules
- System Modules
- Utilities Modules
- Web Infrastructure Modules
- Windows Modules



service - Manage services.

- Synopsis
- Options
- Examples
  - Status
  - Support

Synopsis

- Controls services on remote hosts. Supported init systems include BSD init, OpenRC, SysV, Solaris SMF, systemd, upstart.

Options

parameter	required	default	choices	comments
arguments	no			Additional arguments provided on the command line aliases: args
enabled	no		• yes • no	Whether the service should start on boot. **At least one of state and enabled are required.**
name	yes			Name of the service.
pattern	no			If the service does not respond to the status command, name a substring to look for as would be found in the output of the ps command as a stand-in for a status result. If the string is found, the service will be assumed to be running.
runlevel	no	default		For OpenRC init scripts (ex: Gentoo) only. The runlevel that this service belongs to.
sleep (added in 1.3)	no			If the service is being `restarted` then sleep this many seconds between the stop and start command. This helps to workaround badly behaving init scripts that exit immediately after signaling a process to stop.
state	no		• started • stopped • restarted • reloaded	`started` / `stopped` are idempotent actions that will not run commands unless necessary. `restarted` will always bounce the service. `reloaded` will always reload. **At least one of state and enabled are required.** Note that reloaded will start the service if it is not already started, even if your chosen init system wouldn't normally.
use (added in 2.2)	no	auto		The service module actually uses system specific modules, normally through auto detection, this setting can force a specific module. Normally it uses the value of the 'ansible_service_mgr' fact and falls back to the old 'service' module when none matching is found.

Cont.

- Command to List out all modules installed **ansible-doc –l**
- Read documentation for installed module **ansible-doc ping**

Note: We can get the full details like purpose, options, examples from the document.

*For Windows targets, use the [**win_ping**] module instead. For Network targets, use the [**net_ping**] module instead.*

- To run the module **ansible localhost -m ping**

Run Commands

- If Ansible doesn't have a module that suits our needs there are the "run command" modules:

command: Takes the command and executes it on the host.

The most secure and predictable.

shell: Executes through a shell like /bin/sh, so that we can use pipes etc. Be careful.

script: Runs a local script on a remote node after transferring it.

raw: Executes a command without going through the Ansible module subsystem.

Ad-Hoc Commands

- An ad-hoc command is a single Ansible task to perform quickly, but don't want to save for later.

Check all my inventory hosts are ready to be managed by Ansible

ansible localhost -m ping # Individual Node/System

ansible devservers -m ping # Single Group

ansible all -m ping # All Group

ansible all -m ping -u ec2-user # -u : the user used to log into the server

ansible devservers -m ping -u ec2-user –vvvv # To display verbose output

ansible 13.126.216.73 -m ping --ask-pass # Insist on using passwords

Cont.

#To get hostname of all servers **ansible allservers -a "hostname"**

- If Ansible reports no data or returns some other inventory-related error, try setting the ANSIBLE_HOSTS environment variable explicitly:

  export ANSIBLE_HOSTS=/etc/ansible/hosts.

- The command result was not run on each server in the order (sequence).

- By default, Ansible will run the commands in parallel, using multiple process forks, so the command will complete more quickly.

- If we are managing a few servers, this may not be much quicker than running the command serially, on one server after the other, but even managing 5-10 servers, we will notice a dramatic speedup if we use Ansible's parallelism (which is enabled by default).

Cont.

To perform Ansible to use only one fork (basically, to perform the command on each server in sequence):

ansible allservers -a "hostname" -f 1

Run the same command over and over again, and it will always return results in the same order.

We can put the target *after* the arguments also. Both are same.

ansible allservers -a "hostname“

ansible -a “hostname” allservers

Cont.

- We can do cool things like run a command remotely using **ansible <host> -m command -a "command_to_run"**.

  ansible localhost -m command -a uname

  ansible localhost -m command -a 'ls /etc/ansible'

  ansible localhost -m command -a 'ifconfig'

  ansible devservers -a "date"

Memory utilization

ansible all -a "free -m"

Disk utilization

ansible all -a "df -h"

Run the uptime command on all hosts in the devservers group

ansible devservers -m command -a "uptime"

Make changes using Ansible modules

- We want to install the NTP daemon on the server to keep the time in sync. Instead of running the command yum install -y ntp on each of the servers, we'll use ansible's yum module to do the same. [**yum install -y ntp**]

ansible all -m yum -a "name=ntp state=installed"

ansible multi -s -m yum -a "name=ntp state=installed“

-s => alias for --sudo

[DEPRECATION WARNING]: The sudo command line option has been deprecated in favor of the "become" command line arguments. This feature will be removed in version 2.6.

Deprecation warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.

Cont.

- If we are running commands against a server where the user account requires a **sudo** password

--ask-sudo-pass

Now, we'll make sure the NTP daemon is started and set to run on boot.
We could use two separate commands, **service ntpd start** and **chkconfig ntpd on**

ansible all -m service -a "name=ntpd state=started enabled=yes"

Result:

"changed": true,
"enabled": true,
"name": "ntpd",
"state": "started"