Linux Academy

Ansible Setup, Configuration, and Ad-Hoc Commands Deep Dive

Course Outline Companion

Stosh Oldham

stosh@linuxacademy.com

January 30, 2019

Contents

How to Install Ansible 1

Configure SSH users (with sudo) and keys 1

The Ansible Configuration File 2

Setting Up the Ansible inventory 3

The Ansible Command 4

Understanding Ansible Modules 4

The Shell and Command modules 5

Collecting System information 5

Working with the File and Copy Modules 6

Editing File Contents with the `lineinfile` Module 7

Downloading Files with the `get_url` Module 7

Working with File Archives 8

Creating System Users with the User Module 9

Working with the Group Module 9

Installing Software 9

Controlling Daemons with the Service Module 10

Managing Long-running Commands 10

Parallelism in Ansible Ad-hoc 11

How to Install Ansible

- In order to install Ansible, you must configure the EPEL repository on your system.
- Once the EPEL repository is configured, your package manager can install Ansible and manage dependencies.
    - `sudo yum install ansible`
- It is advisable to install some means of source control as well.
    - `sudo yum install git`
- After package installation, there are two further steps to be taken before Ansible may be used.
    - SSH access must be provided on host you wish to control with Ansible.
    - The Ansible inventory must be configured for each host you wish to control with Ansible.
    - These points are covered in the next sections.

Configure SSH users (with sudo) and keys

- Ansible is best implemented using a common user across all Ansible controlled systems.
- Creating a System User for Ansible
    - Only a basic system user with ssh access is needed for Ansible to connect to a host.
    - The following commands, executed as root, will create a new user on a system called `ansbile` and allow for the user password to be set:
    - `useradd ansible`
    - `passwd ansible`
    - This step should be performed on the Ansible control node and repeated on each other node you wish to have Ansible control.
- Pre-shared Key Authentication
    - While it is possible to connect to a remote host with Ansible using password authentication using `-k` (note lowercase), it is not a common practice as it can incur significant overhead in terms of manual intervention.
    - The `ssh-keygen` and `ssh-copy-id` command can facilitate creating a pre-shared key for user authentication.
        - As the `ansible` user, first run `ssh-keygen` without parameters and follow the prompts to create a key pair for the user.
        - It is recommended that you perform this step on your Ansible control node.
        - Once the key pair has been created, run `ssh-copy-id ansible@TARGET_HOST` to copy the `ansible` user's public key to TARGET_HOST.

    - (**Note**: The `ansible` user must already exist on `TARGET_HOST` and be able to log in with password authentication.)
    - After the public key is in place, you should be able to connect to `TARGET_HOST` as the `ansible` user from your original host without being prompted for a password.

- Escalating permissions

    - For effective system management, the `ansible` user will need a means of privilege escalation.
    - `/etc/sudoers` may be edited to allow your selected user to sudo any command without password for the most automated configuration using the line `ansible ALL=(ALL) NOPASSWD: ALL`.
    - It is also possible to prompt for a sudo password at runtime using `-K`(note uppercase) if desired (**Note**: This can become a challenge when executing against many systems).

The Ansible Configuration File

- Ansible has a number of settings that may be adjusted.
- The default Ansible configuration file is`/etc/ansible/ansible.cfg`
- Notable configurations include:

    - Default inventory configuration
    - Default remote user

- The configuration properties may be overridden using local files.

    - The first configuration file found is used and later files are ignored.
    - Configuration file which will be searched for is in the following order:
        - `ANSIBLE_CONFIG` (environment variable if set)
        - `ansible.cfg` (in the current directory)
        - `~/.ansible.cfg` (in the home directory)
        - `/etc/ansible/ansible.cfg`

- Starting at Ansible 2.4, the `ansible-config` utility ships with Ansible.

    - The utility allows users to see:
        - All the configuration settings available
        - Their defaults
        - How to set them
        - Where their current value comes from.

- The utility may use a few different sub-commands:

    - `view`: Displays the current config file.
    - `list`: Provides all current configs reading `lib/constants.py` and shows env and config file setting names.

    - `dump`: Shows the current settings, merges `ansible.cfg` if specified.
    - **Note**: the `dump` sub-command may take the flag `--only-changed` which will only show configurations not set to the default.
- The `-c` or `--config` switches may be used to specify a particular configuration file.
- If one is not provided, the highest precedent configuration file is used.
- See `man ansible-config` for more information.

Setting Up the Ansible inventory

- An inventory is a list of hosts that Ansible manages.
- The default Ansible Inventory File is `/etc/ansible/hosts`.
- Inventory location may be specified as follows:
    - Default: `/etc/ansible/hosts`
    - Specified by CLI: `ansible -i <filename>`
    - Can be set in `ansible.cfg`
- Example Inventory file:

```
mail.example.com ansible_port=5556 ansible_host=192.168.0.10

[webservers]
httpd1.example.com
httpd2.example.com

[labservers]
lab[01:99]
```

    - The first line of the example file defines a host, `mail.example.com`.
    - Two variables are affiliated with the host, `ansible_port` and `ansible_host`.
    - The groups `webservers` and `labservers` are defined in this example.
    - Note the `labservers` group has 99 hosts in it that are defined via a pattern
    - (The expression `lab[01:03]` is the same as specifying `lab01`, `lab02`, `lab03`.)
    - Note that inventory files may be provided in YAML format but that is uncommon.
    - It is recommended that you keep separate inventories for staging and production.

The Ansible Command

- Ansible ad-hoc commands analogous to bash commands.
- Playbook analogous to bash script (but, that is another course).
- Syntax: `ansible <HOST> -b -m <MODULE> -a "<ARG1 ARG2 ARGN>" -f <NUM_FORKS>`
  - `HOST` is a host or host group defined in the Ansible inventory file.
  - `-b` is for 'become' (This replaces the depreciated `-s` flag as in "sudo").
    - Ansible escalates permission to `--become-user` using the method defined by `--become-method`.
    - Default become-user is `root`.
    - Default become-method is `sudo`.
  - `-m` is for module or command to use (We will look at modules in the next section).
  - `-a` is for parameters to pass. If used without module, it is like running a shell command on the target system(s).
  - `-f` is used to set forks for parallelism, which is how you can have Ansible execute plays simultaneously on many hosts.

Understanding Ansible Modules

- Modules are discrete units of code that can be used from the command line or in a playbook task.
  - Modules may take arguments.
  - Most modules take `key=value` arguments, delimited by whitespace.
  - There are a couple exceptions found in the command and shell modules which we will get to later.
- Common modules (more detail on each and more modules later):
  - Module: `ping`
    Purpose: Verify Ansible connectivity between hosts.
    Arguments: None
  - Module: `setup`
    Purpose: Collect Ansible facts.
    Arguments: None
  - Module: `yum`
    Purpose: Install software with yum.
    Arguments: `name=<PACKAGE_NAME> state=<STATE>`
  - Module: `service`
    Purpose: Control service daemons.
    Arguments: `name=<SERVICE_NAME> state=<STATE>`
  - Module:`copy`
    Purpose: Copy a file to a particular location on a target host.
    Arguments: `src=<SOURCE_PATH> dest=<ABSOLUTE_DESTINATION_PATH>`

- When you need help
    - There are a very large number of modules that perform a variety of tasks.
    - It is nearly impossible to know every module.
    - That said, there is good documentation provided on *docs.ansible.com*.
    - The module index is provided at *docs.ansible.com* and it provides detailed information on each module.
    - Ansible ships with the `ansible-doc` command as well.
        - Specifying a module name as a parameter will provide module specific documentation.
        - The `-l` flag will list installed modules with a brief description.

The Shell and Command modules

- The `shell` and `command` modules are both effective ways to run 'raw' commands on a target host.
- The `command` module
    - It is the default module used with Ansible ad-hoc when no module is specified.
    - The arguments provided are 'free form' and ran as a command on each specified remote host.
    - The module itself may take a few parameters:
        - `creates`: Execution of the module results in the creation of the provided file.
        - `removes`: Execution of the module results in the removal of the provided file.
        - `chdir`: Change to the provided directory before executing the command.
    - The `shell` modules
        - Works almost exactly like the command module except commands are ran in a shell environment (using `/bin/sh`).
        - The primary reason the `shell` module might be used over the `command` module is for use of input / output redirection, environment variables, or other such shell features (which are not supported with the `command` module).
        - Per the Ansible documentation, the best practice is to use the command module unless you require shell features.
        - **Pro tip**: If running multiple shell commands is necessary, there is a `script` module that will copy a shell script to a remote host and execute it, featuring the same arguments that are supported by the `shell` command. (Alternatively, you can create your own Ansible module!)

Collecting System information

- Ansible facts are simply various properties regarding a given remote system.
- The `setup` module can retrieve facts.
    - The filter parameter takes regex to allow you to prune fact output.

- Facts are gathered by default in Ansible playbook execution.
    - The keyword `gather_facts` may be set in a playbook to change fact gathering behavior.
- It is possible to print Ansible facts in files using variables.
- Facts may be filtered using the `setup` module ad-hoc by passing a value for the `filter` parameter.
- Ansible command output may be directed to a file using the `--tree outputfile` flag, which may be helpful when working with facts.
- It is possible to use `{{ ansible_facts }}` for conditional plays based on facts.

Working with the File and Copy Modules

- Working with files is an essential operation for systems management and configuration.
- Ansible has a number of core modules that provide for file manipulation that will be covered throughout this course.
- The `file` module
- The `file` module can be used to create, delete, and modify file properties.
- The `copy` module
- This module is used for copying files.
- Files may be copied from a number of sources.
    - Files may be copied from the control node to target nodes.
    - Files may be copied from file systems on the target node.
    - Files may be created during execution of the copy module and place on target nodes.
- The following are notable arguments used by the `copy` module:
    - `remote_src`
        - If no, it will search for `src` on the control node.
        - If yes it will go to the target node for the `src`.
        - Default is no.
        - Currently `remote_src` does not support recursive copying.
        - `remote_src` only works with mode=preserve as of version 2.6.
    - `content`: When used instead of `src`, sets the contents of a file directly to the specified value.
    - `dest`: Remote absolute path where the file should be copied to.
    - `mode` - Mode (in octal or symbolic notation) the file or directory should be.
    - `src`
        - Local path to a file to copy to the remote server; can be absolute or relative.

    - If path is a directory, it is copied recursively.
        - In this case, if path ends with "/", only contents inside of that directory are copied to destination.
        - Otherwise, if it does not end with "/", the directory itself with all contents is copied.
    - This behavior is similar to `rsync`.

Editing File Contents with the `lineinfile` Module

- The `lineinfile` module is a key way of interacting with a file's contents.
- The key feature of the module is inserting a line of provided text into a given file.
- The module will place the line of text at the end of a file by default, however it can place the line after or before a line of matched text.
- If the module detects the provided text already in place, it will not add it again.
- It may also remove lines of text.
- Key arguments for the `lineinfile` module:
    - `backup`: Create a backup file, including the timestamp information, so you can get the original file back if you somehow clobbered it incorrectly.
    - `create`: Create the file if it does not exist. (False by default.)
    - `state`: Whether or not the line should be there.
    - `insertafter`: Takes a regular expression and inserts `line` *after* the first matched line. (Only honored if `regexp` is undefined or unmatched.)
    - `insertbefore`: Takes a regular expression and inserts `line` *before* the first matched line. (Only honored if `regexp` is undefined or unmatched.)
    - `regexp`: Takes a regular expression and replaces the last line matched with **line** or removes the matched line if `state` is set to *absent*.
    - `line`: The text to insert/replace.
    - `path`: The file to modify.
- **Note**: There is a `replace` module that can make more granular file changes using regular expressions.
    - It uses a `replace` argument instead of the `line` argument.
    - The module also supports `before`, `after`, `path`, and `regexp` arguments.

Downloading Files with the `get_url` Module

- It is frequently necessary to download files from HTTP hosts on managed hosts.
- The `get_url` module provides functionality for downloaded files over HTTP with a variety of options.
    - The HTTP, HTTPS, and FTP protocols are supported

    - Basic and SSL Authentication
    - Checksum verification
    - Proxy network access

- Notable Arguments:

    - `dest`: Where the file should be placed on the *remote host*.
    - `checksum` - The algorithm and result value used to verify the file's checksum.
        - **Note**: The result value may be provided as a text file containing the value or the plan text value.
    - `force_basic_auth`: This option forces the sending of the Basic authentication header upon initial request.
    - `mode`: The mode of the final file.
    - `url`: The URL of the file to download.

Working with File Archives

- The `archive` and `unarchive` modules may be used to to work with various types of file archives.
- These modules work with common archives such as `tar`, `gunzip` (default), and `zip`.
- These modules allow for compression (or expansion) of files and directories.
- `archive` Notable Arguments:
- `dest`: The file name of the archive file.
- `format`: Select compression format.
- `path`: Remote absolute path, glob, or list of paths or globs for the files to archive.
- `remove`: Controls if the source files will be deleted after the archive is created.
- `unarchive` notable arguments:
- `dest`: Where archive files should be unpacked.
- `remote_src`: If set to 'yes', file is already on target system.
- `remove`: Controls if the source archive file will be deleted after execution.
- `src`: Location of source archive.
    - If `remote_src` is set to 'no' (the default) `src` is local to control server.
    - If `remote_src` is set to 'yes', `src` is on target server.
    - If `remote_src` is set to 'yes' and `src` contains `://`, the target server will download the file from specified URL (similar to `get_url` module).

Creating System Users with the User Module

- Interacting with system users is a common operational task.
- Ansible allows this functionality using the `user` module.
    - The `user` module may interact with users in many standard ways.
        - Create users
        - Remove users
        - Change user properties
- Common Arguments:
    - `group`: Specify a primary group for the named user.
    - `groups`: Specify secondary groups for the named user.
    - `name`: Name of the user with which to interact.
    - `generate_ssh_key`: Whether to generate a SSH key for the named user.
    - `remove`: When used with `state=absent`, the named user is deleted.
    - `state`: Creates the named user if it doesn't exist when set to 'present'.
    - `uid`: Optionally set the UID of the named user.

Working with the Group Module

- Just as with system users, Ansible support working with system groups.
- The `group` module serves this function.
    - The `group` module works with groups in the same way the `user` module interacts with users.
- Common Arguments:
    - `gid`: Optionally set the GID of the named group.
    - `name`: Name of the group with which to interact.
    - `state`: Creates the named group if it doesn't exist when set to 'present'.
    - `system`: Indicate if a group is a system group.

Installing Software

- One of the key features necessary for bootstrapping and managing a host is the ability to install and manage software packages.
- `package`, `yum`, and `apt` are used for this purpose depending on distribution.
    - `yum` and `apt` are used for distribution specific software management.

    - The `package` module will automatically detect the target host distribution and use the appropriate method of installing software.
    - In a mixed distribution environment, the `package` module is best practice.

- Common Arguments:

    - `name`: The name of the software package to be installed.
    - `state`: If the named package should be 'present' or 'absent'.
    - `user`: Which package manage to use. (Defaults to 'auto'.)

Controlling Daemons with the Service Module

- Another common administration task is to control system daemons.
- The `service` module serves this purpose.

    - It is compatible with both systemd and the older BSD init style of daemon management.

- Common Arguments:

    - `enabled`: Configures whether or not the service should start on boot.
    - `name`: The name of the service to manage.
    - `state`: The state of the service after the module executes. May be one of:
        - *started*: Service will be started (will not restart if already started).
        - *stopped*: Service will be stopped.
        - *restarted*: Service will be restarted even if it is already started.
        - *reloaded*: Service configuration will be reloaded if supported by the service.

Managing Long-running Commands

- Some operations may require a significant amount of time to execute.
- Ansible provides a feature to allow an operation to run asynchronously such that the status may be checked.
- There are two options related to running asynchronous commands:
- `-B` is used to provide a timeout value and initiate the operation.

    - Unit: seconds
    - Default value: N/A
    - Job is terminated when timeout value expires.

- `-P` may be provided to engage polling of the operation on a set interval.

    - Unit: seconds

    - Default value: 15
    - Polling will leave the Ansible command in the foreground until execution on all hosts completes.
    - Set to 0 to disable polling.

- Example: `ansible all -B 1200 -P 60 -a "/usr/bin/long_running_operation"`
- This example provides a 20 minute timeout and polls the operation's status every 60 seconds.
- The status may be checked using the module `async_status`.
    - Example: `ansible httpd1 - m async_status -a "jid=12334568901234.1234"`
    - Note: JID is only realistically obtained during playbook operation.

Parallelism in Ansible Ad-hoc

- Ansible uses what are know as forks to execute tasks in parallel.
- Each fork is able to run a task against a target host.
- By default, Ansible executes 5 forks.
- Assuming one runs a module against 5 hosts, the module should execute near simultaneously on all hosts.
- If more than 5 hosts are targeted, hosts after the first 5 must wait for a fork to become free before being able to execute.
- The number of forks used by Ansible may be configured in `ansible.cfg` or on an as-needed basis using the `-f` or `--forks` flag.
- **Note**: The more forks allowed, the more system resources will be required on the Ansible control node.
- Example: `ansible all -f 10 -m setup`
    - This example will use 10 forks to execute the `setup` module against all hosts in the inventory.
    - **Note**: Ansible will only fork as much as required.
    - Presuming Ansible is configured to run 10 forks but is only tasked with executing on 6 hosts, only 6 forks will be created.