How to use Ansible's lineinfile module in a bulletproof way

Ansible's lineinfile module is a double-edged sword so it pays to know a sure-fire approach to edit configuration files.

Michael Trojanek Dec 20, 2016 · 5 min read

Ansible's lineinfile module is used to add, change or remove a single line in a file.

Before putting it to use, you should make sure that the task you want to accomplish cannot be done more elegantly with Ansible's template or copy module:

Both of these allow you to prepare an entire file upfront (either with or without variable expansion) which gives you complete control over the

file's contents. So wherever possible, prefer the copy and template modules over manipulating only parts of a file with the lineinfile module.

Unfortunately, there are cases where this is not possible:

- **Different roles modify the same file**
  One role may need to edit the `ListenAddress` directive in the *sshd_config* file when adding a network interface while another role may need to adjust the `AddressFamily` directive to disable IPv6.
- **You are dealing with legacy systems**
  You may need to add configuration options to a legacy postfix configuration file which is so complicated that nobody knows what it is actually doing.
- **You are dealing with target systems which use different versions of the file**
  Your webservers may allow logins from different users than your backend servers in their respective *sshd_config* files.

Idempotency at all times

If you know my Efficient Rails DevOps products or are following my articles, it will not have escaped your notice that I am preaching

idempotency at all times when building Ansible tasks.

This means that an Ansible task which introduces a change to your system must not report the same change again when run a second time.

This becomes more and more important once your playbooks start to grow. It is crucial for any operator to be properly informed about things that have changed and things that have not.

Let's reuse the example from above and make our SSH daemon listen on the IP address `1.2.3.4`.

The first approach

The easiest way to do so is by using the following task:

```
- name: Listen on 1.2.3.4
  lineinfile: dest=/etc/ssh/sshd_config
              line="ListenAddress 1.2.3.4"
              state=present
```

The line `ListenAddress 1.2.3.4` will be added at the end of the *sshd_config* file because we did not specify where to put it. Ansible is smart enough not to add this line again as long as the last line of the *sshd_config* file is not changed. So while a bit wonky, out task is idempotent and does what we want.

However, this approach is dangerous: Our server is now listening on the address `1.2.3.4` but it may also listen on other addresses we do not know about.

Next iteration

The goal of our task is to make our SSH daemon listen on the one (and only the one) IP address we specify.

When we take a step back and examine the necessary steps, we will see that we actually need two tasks to reach our goal:

- First, we have to remove all occurrences of `ListenAddress` that do not read `ListenAddress 1.2.3.4`.
- Then we have to add the line `ListenAddress 1.2.3.4` at a specific place in the file.

Removing unwanted lines can be done with this task:

```
- name: Remove lines with unwanted occurrences of ListenAddress
       lineinfile: dest=/etc/ssh/sshd_config
                   regexp="^ListenAddress (?!1.2.3.4)"
                   state=absent
```

What's not so obvious here is the `(?!1.2.3.4)` : This is a regular expression construct called negative lookahead:

The regular expression `^ListenAddress (?!1.2.3.4)` matches lines starting with `ListenAddress` , followed by a space, then *not followed* by `1.2.3.4` . For our task, this means that all `ListenAddress` directives are removed except those which specify the IP address `1.2.3.4` .

Now that our file is clean, we can add the correct directive. Specifying a place where to add the line enables Ansible to judge whether this task needs to be executed or not on future runs:

```
- name: Listen on 1.2.3.4
  lineinfile: dest=/etc/ssh/sshd_config
              line="ListenAddress 1.2.3.4"
              insertafter="^#?AddressFamily"
```

This task will add the line `ListenAddress 1.2.3.4` directly under the line starting with `AddressFamily` (comment or not).

A real life example

In real life, you will often need to add more than one line to a file. This can easily be achieved with Ansible's `with_items` syntax and a little regular expression trickery (if you find yourself in need to add blocks to a file often, take a look at Ansible's blockinfile module).

Let's edit our example *sshd_config* file to make the SSH daemon listen on multiple IP addresses now. If you are using Digital Ocean's floating IPs, you may want to make your server accept SSH connections on its public and private IP — in this example, the public IP is `46.101.70.239` and the private IP is `10.19.0.6`.

First, we have to update our regular expression to preserve lines containing `ListenAddress` for both IPs:

```
- name: Remove lines with unwanted occurrences of ListenAddress
  lineinfile: dest=/etc/ssh/sshd_config
              regexp="^ListenAddress (?!46.101.70.239|10.19.0.6)"
              state=absent
```

Then we can use Ansible's `with_items` syntax to loop over both IPs and add the appropriate `ListenAddress` directives to the file:

```
- name: Listen on 1.2.3.4
  lineinfile: dest=/etc/ssh/sshd_config
              line={{ item.line }}
              insertafter={{ item.insertafter }}
  with_items:
    — { line: "ListenAddress 46.101.70.239", insertafter: "^#?
AddressFamily" }
    - { line: "ListenAddress 10.19.0.6", insertafter: "ListenAddress
46.101.70.239" }
```

Note that the line containing the public IP gets added under the line specifying the `AddressFamily` whereas the `ListenAddress` call for the private IP gets added under the one for the public IP.

The definitive version

Strictly speaking, we have to escape the IPs' dots in regular expressions (and write them like `46\.101\.70\.239`)—otherwise the `.` matches not an actual dot but *any character* (not much of a problem in this case but it's better to be on the safe side).

In your actual playbook the IP addresses will probably be stored in variables which leads us to the final version of our tasks (using Jinja2 filters to escape the dots in our IP addresses where appropriate):

```
- name: Remove lines with unwanted occurrences of ListenAddress
  lineinfile: dest=/etc/ssh/sshd_config
              regexp="^ListenAddress (?!{{ public_ip|replace('.',
'\.') }}|{{ private_ip|replace('.', '\.') }})"
              state=absent


- name: Listen on public and private IP
  lineinfile: dest=/etc/ssh/sshd_config
              line={{ item.line }}
              insertafter={{ item.insertafter }}
  with_items:
    - { line: "ListenAddress {{ public_ip }}", insertafter: "^#?
AddressFamily" }
    - { line: "ListenAddress {{ private_ip }}", insertafter:
'ListenAddress {{ public_ip|replace(".", "\.") }}' }
```

Using this bulletproof approach to edit lines in configuration files will make your Ansible tasks a lot more robust.

This article has been originally published on my website (you can find it here). If you think it's useful, you should join my email list — there's a lot more where this one came from!

Ansible DevOps