Creating Playbooks

Ad-Hoc Commands

First, we should be sure that ansible is configured correctly, to run commands on a server or a group of servers within the `/etc/ansible/hosts` file, run any of the below commands

```
ansible -m ping hostname
ansible -m ping 134.23.4.5

ansible -a "sudo ls /" hostname
ansible -a "sudo ls /" 134.23.4.5

ansible -a "free -h" hostname
ansible -a "free -h" 134.23.4.5
```

While the above is an example of running bash commands on remote hosts ad-hoc via the commandline, you can also run ansible modules from the commandline in a similar way -

```
ansible remotehostname -m fetch -a "src=/home/remoteuser/path/file.txt dest=/home/localuser/
flat=yes"
```

Here, we grab `file.txt` from a remote host and copy it to our local home directory. Where `-m` is selecting which module to use and `-a` is providing the options that you would specify within a normal playbook via a command. Be sure to enclose any module options after `-a` with double quotes or the command will fail. We use `flat=yes` to tell Ansible that we just want the file, and not to rebuild the directory from the remote host. `flat` defaults to `no`, which would result in this command building out the full directory `/home/localuser/www.remotedomain.com/home/remoteuser/path/file.txt` on our local host. See the Ansible documentation for each module for more information on their arguments. Here's a link to the documentation for the fetch module

For the above command, we used the `fetch` module, which may not work for directories and may consume a lot of memory if you are transferring a large file. For example, I experienced issues with this when transferring large database backup files between hosts. If this is your use case, I would recommend checking out the synchronize module documentation.

As an example of an ad-hoc `synchronize` command, I have used this in the past to retrieve fail2ban configurations on a remote host. Note the `mode=pull` parameter that tells ansible that we want to get the files from the remote host and place them at the local destination. By default, `mode` is set

to `push`, which would attempt to copy files from our local host and send them to a directory on the remote host.

```
ansible -m synchronize remotehostname -b -a "src=/etc/fail2ban/filter.d/
dest=/some/local/directory/fail2ban/ mode=pull"
```

Creating Playbooks

Ansible can be configured to carry out tedious or otherwise common tasks on any number of hosts, as we see below in the example playbook where Ansible is being used to backup an instance of Bookstack.

```
---
- hosts: bookstack
  become: yes
  tasks:
    - name: Backup Bookstack container files
      command: tar -cvzf bookstack-backup.tar.gz /home/admin/bookstack
    - name: Fetch backup files from remote host
      command: scp -P 2222 -i /home/username/.ssh/id_rsa /home/admin/bookstack-backup.tar.gz
admin@sub.domain.com:/home/admin/backups/bookstack/
```

Here, we use `scp` instead of Ansible's Fetch module to save memory on the small host that runs the BookStack you are viewing. When fetching large files, memory errors can be encountered so here we have worked around the module using an alternative method for transferring our files.

Here is another example, using ansible to synchronize the fail2ban configurations used between multiple hosts. This allows us to configure one host, which is the local host that runs the playbook, and once we have configured this host correctly we can just run the play and push our changes to a group of hosts. Note that `hostgroup` should be specified in the local `/etc/ansible/hosts` file, or ansible will not be able to run the play. Also notice that the `src` directories in this playbook are relative to the path of the playbook itself. This allows me to store custom fail2ban configurations for different groups of host alongside this playbook to avoid configuring fail2ban to monitor services that don't exist on the system. If you try to monitor a service that does not exist, fail2ban will fail to reload.

```
---
- hosts: hostgroup
  become: yes
  tasks:
    - name: Copy custom fail2ban filters
      synchronize:
        mode: push
```

```
      src: fail2ban/filter.d/
      dest: /etc/fail2ban/filter.d/
  - name: Copy custom fail2ban jail.local
    synchronize:
      mode: push
      src: fail2ban/jail.local
      dest: /etc/fail2ban/
  - name: Reload fail2ban service
    ansible.builtin.service:
      name: fail2ban
      state: reloaded
  - name: Checking status of fail2ban service after restart
    command: systemctl status fail2ban
    register: result
  - name: Showing fail2ban status report
    debug:
      var: result
```

> Be careful when synchronizing configurations in this way, hosts can be configured with different services which could result in the fail2ban service failing to reload when it is unable to find the related log files. For this reason, I use a seperate directory to configure fail2ban for hosts with similar filters. In my case, my host that runs the ansible playbooks does not have nginx installed, so copying over configurations for nginx jails will result in fail2ban failing to reload.

Revision #7
Created 2019-08-30 01:51:03 UTC by Shaun Reed
Updated 2021-06-11 06:28:09 UTC by Shaun Reed