Configuration Management Systems: Working with Red Hat Ansible

Nicole Hessner
Independent Project 01, CS504 – Software Engineering,
M.S. Computer Science, School of Computing and Technology,
City University of Seattle
hessnernicole@cityuniversity.edu

Abstract

Configuration Management Systems (CMS) are useful tools for ensuring version control and configuration consistency across various groups of users, operating systems, and functions within a project group or enterprise. With both open source and enterprise-level CMS software available, it can be difficult to choose a CMS program that adequately suits the needs of an organization. This paper uses Red Hat Ansible as an exploration of both Configuration Management Systems as a software package and the process of learning to make choices that fulfill the requirements of the project. Due to the heavy reliance of the command line interface in Ansible and the author's relative inexperience with scripting, this paper is also a reflection on the process of learning computing and technology concepts. There is a heavy emphasis on learning to use the tool that the author has chosen to accomplish a list of tasks, even when another tool may be better or more easily suited to one's needs. It is often the case in a large enterprise setting that the tools are chosen and contracted by someone other than the person who becomes the primary administrator. This project offers a chance to practice navigating that environment, because requirements can only be appropriately interpreted through the lens of experience, which the author gained rapidly over the course of this assignment.

Keywords: configuration management system, Ansible, Git, GitHub, continuous integration/continuous delivery, software decisions, command line interface

1. INTRODUCTION

Software Configuration Management Systems (CMS) are utilized in development projects to manage complex projects. There are many CMS systems on the market, with both Enterprise and Open Source options, and before any system can be explored, one must first choose a software package to work with. Given my lack of experience with the different software packages, I committed myself to learning whichever software I had chosen.

After choosing the software, the next task was to learn how to perform a set of standard version control operations using the chosen software. These tasks include standard Git tasks such as code check-out, check-in, and merging. It also included taking a code snapshot and managing simultaneous development on the same code by two users, code rollback, and managing users developing different versions of the same code. Finally, there were tasks related to the operation of the CMS environment itself, such as defining runtime variables within the CMS, deploying the CMS to users, and managing CMS configuration and Playbooks.

This paper includes some commentary about the process of learning how to use the software, including notes where some steps were simplified for the sake of successful execution, with the understanding that advanced tasks become simpler after mastering the basics.

2. CONFIGURATION MANAGEMENT SYSTEM (CMS) SELECTION

The Configuration Management System I have selected for this project is Ansible. I chose this software based on a shortlist of recommendations from my professor, a preference for Open Source over Enterprise solutions due to financial and philosophical considerations, and a review of the different options using G2, a public, peer-reviewed software evaluation website.

As of the writing of this paper, Ansible has 4.5 out of 5 stars, based on 108 reviews, the highest rating of any of the CMS software options reviewed. Despite listing three pricing tiers (Basic Tower - $5,000, Enterprise Tower - $10,000, and Premium Tower - $14,000), several reviews state that the software itself is free (Red Hat Ansible Automation Platform, n.d.). Also, the source code is freely available on GitHub (Ansible, 2022d). Based on the installation guide, Ansible is also a good choice for me, because the control node can be installed on any popular operating system besides Windows, and I am running macOS Monteray and Ubuntu systems.

While Ansible's feature set includes features for app deployment, provisioning, continuous delivery, security and compliance, and orchestration (Ansible [Linux0], 2018), the scope of this paper is limited to configuration management tasks only.

3. CODE CHECKOUT (OR GIT CLONE)

To access a private repository, Ansible uses a user's existing GitHub or other source code management (SCM) tool login (for GitHub: SSH, username and token), which can then be stored and encrypted in an Ansible vault. Ansible vaults are created using the `ansible-vault create {file-name}.yml` command (Ak, 2022b).

Accessing a public repository is simpler than accessing a private repository, since credentials are not required for cloning. For the scope of this project, I decided that I did not have the security background to feel comfortable using a vault and encrypted credential file I would eventually need to share. Instead, I focused on the basic Git functionality of Ansible, and created a sample public repository and a playbook for retrieving it on both of my hosts simultaneously. My sample repository is available here:

https://github.com/CityUhessnern/AnsibleCloneTest.

The code included in my gitClonePublic.yml playbook, shown in Figure 1 is a modified version of a playbook shared by the LHB Community (2021), altered to use my test repository and destination directory. While it is not a best practice to Git clone a repository onto every user's hard drive in a production environment, it was a very good way to test Ansible's playbook methodology on my two virtual machines.

```
---
- hosts: all
  tasks:
  - name: Clone a github repository
    git:
      repo: https://github.com/CityUhessnern/AnsibleCloneTest
      dest: ~/Desktop/CS504-Project-HessnerNicole/Submission1/AnsibleCloneTest
      clone: yes
      update: yes
...
```

Figure 1. Ansible Playbook for cloning a public GitHub repository.

4. CODE CHECK-IN (OR GIT COMMIT)

While Ansible does not have the same integrated functionality for code check-in as it does for code

check-out from GitHub, including the `shell` command in playbook tasks allows the CMS to automate the same Git commands that would be used in the Terminal (Ak, 2021). To simplify my efforts, this will commit be working exclusively with the local host.

The playbook for this command, located in Appendix A, uses code modified from Reddit user u/mobbeduo (2019), and is made up of four tasks: (1) checking for changes from the master repository, (2) performing a `git add` to add local changes, (3) committing the changes, and (4) pushing the changes to the repository. I would like to investigate the security implications of the way Ansible deals with passing through GitHub credentials in a playbook, going forward.

5. CODE PULL REQUEST

According to the GitHub (n.d.) documentation, "Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch."

A `git pull` request first runs the `git fetch` command, before calling either the `git rebase` or the `git merge` command. `git fetch` gets the changes from the server, `git rebase` reapplies commits over another branch, and `git merge` "joins two or more development histories together (git-scm, 2022b)." Typically, it is a best practice to perform a `git status` command and commit changes before performing before doing a `git pull`, to prevent local changes from being overwritten (GitHub, 2022). In Ansible, this is another Playbook, which is available for review in Appendix B. The pull request created from my Ansible playbook is available in Appendix C.

6. CODE MERGE

A git merge operation combines the commits on one branch of a repository with another branch. This is often done when bug fixes or a new version of the software are completed and ready to be merged with the master branch (git-scm, 2022a). In Ansible, this will be another example of using the shell command for a Git task. The Ansible Playbook for performing a merge first asks for a local branch name, a commit message, and a remote branch name. Next, it executes the following tasks: Git Checkout to create and/or move to the desired local branch, Git Add and Git Commit to commit local changes, then Git Pull from the remote branch to perform the combined Git Fetch and Git Merge operations.

I created notes in the README for the repository noting which branch the code was merged from when it was successfully pulled into the local README document. I also included a local note to ensure that existing commits were not being overwritten. Figure 2 and Figure 3 show that the code in command-line-pull-test-branch successfully merged into the local README. My gitMerge.yml playbook is available in Appendix D.

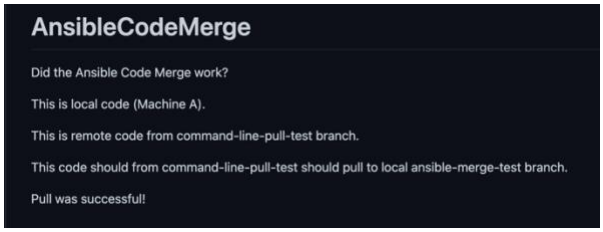


Figure 2. Screenshot of AnsibleCodeMerge section of README.md, located on GitHub.

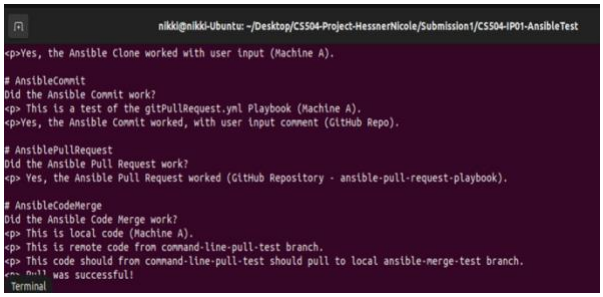


Figure 3. Screenshot of AnsibleCodeMerge section of README.md, located in local README.md.

7. CODE SNAPSHOT (OR RELEASE POINT)

A code snapshot or release point is a copy of all source code, code data structures, data or test data, and test cases frozen simultaneously, creating coherence (Annubex, n.d.). In GitHub, releases are managed by adding a Git tag that identifies files as part of the same release (git-scm, 2022b). To manage these releases in Ansible, the GitHub CLI tool must be installed on all relevant machines. While this is a task that could be automated with Ansible, I chose to just install it manually on my two test machines to reduce the complexity of this project. As with previous actions that utilize GitHub repositories, the shell command will be utilized in the Ansible Playbook, as shown in Appendix . The command `gh release create` *`tag`* is used to manage Release Points in GitHub. The playbook must include the arguments `--title` and `--notes`, or else the playbook will hang as it tries to open GitHub CLI's

editor (GitHub, n.d.-c). My release playbook is available in Appendix E, and a screenshot of successful execution can be viewed in Appendix F.

8. CODE IN DEVELOPMENT BY TWO USERS

To simulate the process of two users editing code simultaneously, first, I had to make user accounts in Ansible. The scope of this project required two users, one of whom is an admin. While I did make users in Ansible, I could not see how to use the functionality to collaborate on a GitHub repository that was different than what GitHub offers, so I added my non-admin user to the GitHub repository to run this test. The admin account was on Machine A in the second user was on Machine B.

Since I was using Ansible to interface with GitHub, my users had to deal with non-fast-forward errors in GitHub by using `git pull` after performing a `git commit` (GitHub, n.d.-b). Since I have both the gitMerge.yml and gitCommit.yml playbooks, I ran them in sequence when working with my second user. Figure 4 shows the combined changes that were made between Machine A, the server, and Machine B. In the future, I could combine them into a single gitUpdate.yml playbook that calls the original two playbooks.

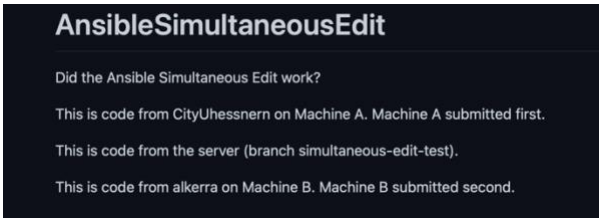


Figure 4. Results of two users editing the same branch of a repository simultaneously.

9. CODE ROLLBACK

To rollback code, use the git reset command with the commit partial checksum that needs to be rolled back (Burns, 2021). Because of the nature of Ansible when run in the command line, it is easiest to simply run the `git log --oneline` command to get the commit history, and then manually input the `git reset` command. To reset a commit that has already been pushed, such as one created using gitCommit.yml, use the force push option by adding + ahead of the desired branch name.

10. GIT FORK

A GitHub Fork is a copy of a repository that a local developer can edit, usually to propose changes to a repository to which they do not have access (GitHub, 2022b). Forks are a functionality of GitHub, and not standard Git, although GitHub utilizes Git tags to identify files for the release. Before giving my secondary account access for Section 8, I used it as a test bed for creating a Fork, which I successfully completed using the playbook in Appendix F. Evidence of that completion is in Appendix G.

11. RUNTIME VARIABLE DEFINITION

According to the Ansible (2022c) Documentation, variables can be passed into a playbook at runtime using the argument `--extra-vars` or `-e`. User input can be requested using a `vars_prompt` command in the playbook. To pass strings into the variable, use key=value format. Otherwise, JSON format should be used, or variables with a lot of special characters can be imported from either a JSON or YAML file. vars_prompt. Runtime variables are utilized throughout the Ansible playbooks scripted for this project, most often asking for the user to provide a commit message or Git branch. Figure 5 shows an example of a playbook requesting runtime variable prompts.

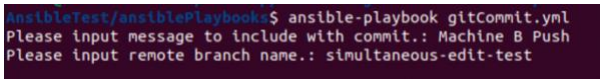


Figure 5. Example of runtime variable prompts.

12. SOFTWARE DEPLOYMENT

Despite intending to use my MacBook as the control node, I decided to work with Ansible using two Ubuntu virtual machines, after deciding that troubleshooting the connection and file location issues I was having was outside the scope of this project. Because the instructions in the Ansible documentation refer to default Linux file locations, and macOS installations follow the same conventions due to macOS's handling of root and user folders, the project tasks became much simpler after that decision point.

The core of Ansible's function is managing networks and projects, to ensure that similar groups of computers and users are working with the same versions of software and environment configurations. The first step of an Ansible deployment is installing Ansible on a computer that is to be a designated Control Node. As noted previously, any Linux or macOS machine can be a control node (Ansible, 2022c), but available

documentation is better suited to Linux implementations, especially for beginners.

The simplest method for Ansible installation is dependent on Python 3, and the Python Package Installer (PyPI/pip). Once PyPI is installed, the command to install Ansible is `$ python3 -m pip install --ansible` (Ansible, 2022c). After installation, I edited the default `/etc/ansible/hosts` file to include my second Ubuntu Virtual Machine, as shown in Figure 6. Since this is only a two-machine operation, I wanted to include my control node as a regular host, as well, and found the instructions on running playbooks locally by Ak (2022a) very helpful.

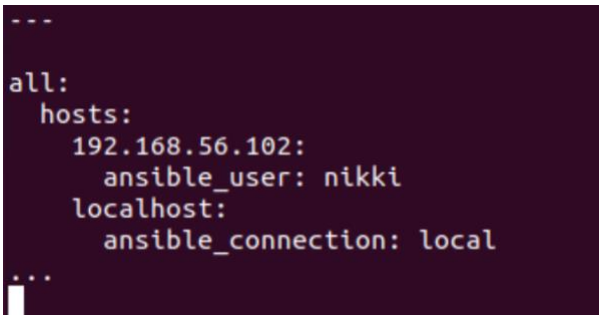


Figure 6. Ansible hosts file with localhost and IP address for second host in YAML format.

I tried pinging the hosts using `$ ansible all -m ping --ask-pass`, as indicated in the Ansible Installation Documentation (2022b), but had authentication issues. To troubleshoot, I generated an SSH key pair for the two machines using `ssh-keygen` and `ssh-copy-id` (SSH, n.d.), after which point authentication was possible without a password, and `--ask-pass` was no longer required.

13. CMS PLAYBOOK (CONFIGURATION FILE)

Playbooks

Ansible runs on "Playbooks," which are, "YAML files that describe the desired end-state of something (Ansible [Linux0], 2018)." A Playbook can be located anywhere on the system, but must be in the current working directory when it is called (Both, 2020a). Playbooks contain plays, tasks, and a handler, which is a special task that may be triggered by a task. An example of a handler is restarting an application when a task changes its configuration. Roles are special playbooks that are self-contained with tasks, variables, configurations templates, and supporting files (Ansible [Linux0], 2018).

Configuration

The default location for the configuration file for Ansible in Linux installations is `/etc/ansible/ansible.cfg`. Most of the configuration options are commented out, but can be uncommented and changed by a superuser. According to the Ansible Documentation, the default configuration is appropriate for most users.

14. CONCLUSION

Knowing what I have learned about the basic processes expected from a Configuration Management System for the purpose of software engineering, I probably will be more conscious of ensuring that my software choices are aligned with my and/or my customer's requirements and skill levels in the future. While choosing a piece of software and learning to interpret the requirements was a good experience, having gained that experience, I better understand what I should be looking for when choosing software to implement for this use case.

Based on the information contained in search results throughout my research, it seems that Ansible's dominant use case is for administering networks and pushing updates to various machines. While that is the dominant use case, Ansible's command line and Playbook structure is infinitely flexible, and can be utilized well for any repetitive task. This is especially true of tasks that need to be executed against several users or machines, such as installing or updating dependencies, or tasks that are usually completed in succession, such as git-status, git-add, git-commit, and git-push/pull. Due to its lack of native GUI, however, it is not for the casual user who is not familiar with the command line, in general. For those who are familiar with Git and GitHub commands, it is relatively simple to convert those commands into YAML format and create functioning playbooks.

Due to my relative inexperience, most of the playbooks created over the course of this project were rudimentary and do not represent the full power of Ansible. Going forward, I would like to continue to develop my skills utilizing this platform and implement more complex playbooks that utilize more user input variables, where appropriate.

16. REFERENCES

Ak, S. (2021, January 12). Ansible Shell Module Examples. Middleware Inventory. Retrieved April 30, 2022, from https://www.middlewareinventory.com/blog/ansible-shell-examples/#Example_1_Execute_a_Single_Command_with_Ansible_Shell

Ak, S. (2022a, January 1). How to Run Ansible Playbook Locally. Middleware Inventory. Retrieved April 24, 2022, from https://www.middlewareinventory.com/blog/run-ansible-playbook-locally/#Method3_Add_an_entry_in_your_Inventory

Ak, S. (2022b, January 19). Ansible Git Example – Checkout code from Git Repo Securely. Middleware Inventory. Retrieved April 23, 2022, from https://www.middlewareinventory.com/blog/ansible-git-example/

Ansible. (2022a, April 20). Connection methods and details — Ansible Documentation. Retrieved April 24, 2022, from https://docs.ansible.com/ansible/latest/user_guide/connection_details.html#connections

Ansible. (2022b, April 20). Installing Ansible — Ansible Documentation. Retrieved April 24, 2022, from https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html

Ansible. (2022c, April 20). Using Variables — Ansible Documentation. Retrieved April 23, 2022, from https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#defining-variables-at-runtime

Ansible. (2022d, April 22). GitHub - ansible/ansible. GitHub. Retrieved April 23, 2022, from https://github.com/ansible/ansibleAnsible [Linux0]. (2018, September 8). What is Ansible? Ansible Quick Start Video [Video]. YouTube. https://www.youtube.com/watch?v=9p8W7j2l7Mg&t=9s

Anubex. (n.d.). What is a Code Snapshot? Retrieved April 24, 2022, from https://www.anubex.com/glossary/what-is-a-code-snapshot

Both, D. (2020b, November 12). How to create an Ansible Playbook. Red Hat. Retrieved April 23, 2022, from https://www.redhat.com/sysadmin/ansible-updates1

Burns, S. (2021, March 25). How to Roll Back Git Code to a Previous Commit. TechTarget. Retrieved May 1, 2022, from https://www.techtarget.com/searchitoperations/answer/How-to-roll-back-Git-code-to-a-previous-commit

git-scm. (2022a, April 26). Git - git-merge Documentation. Git. Retrieved May 1, 2022, from https://git-scm.com/docs/git-merge

git-scm. (2022b, April 26). Git - Tagging. Git. Retrieved May 1, 2022, from https://git-scm.com/book/en/v2/Git-Basics-Tagging

GitHub. (n.d.-a). CLI. Retrieved May 1, 2022, from https://cli.github.com/

GitHub. (n.d.-b). Dealing with Non-Fast-Forward Errors. Retrieved April 30, 2022, from https://docs.github.com/en/get-started/using-git/dealing-with-non-fast-forward-errors

GitHub. (n.d.-c). Releasing Projects on GitHub - Managing Releases in a Repository. Retrieved May 1, 2022, from https://docs.github.com/en/repositories/releasing-projects-on-github/managing-releases-in-a-repository

GitHub. (2022a). About Pull Requests. Retrieved May 1, 2022, from https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests

GitHub. (2022b). Fork a Repo. Retrieved May 1, 2022, from https://docs.github.com/en/get-started/quickstart/fork-a-repo

GitHub. (2022c). Git Pull. GitHub Cheatsheets. Retrieved May 1, 2022, from https://training.github.com/git-guides/git-pull/

LHB Community. (2021, June 28). How to Clone a Git Repository with Ansible. Linux Handbook. Retrieved April 24, 2022, from https://linuxhandbook.com/clone-git-ansible/

Red Hat Ansible Automation Platform. (n.d.). G2. Retrieved April 23, 2022, from https://www.g2.com/products/red-hat-ansible-automation-platform/reviews

Sagar (2022, January 8). How to Use Ansible Create User Functionality in Linux. ATA Learning. https://adamtheautomator.com/ansible-create-user/

SSH. (n.d.). ssh-copy-id for copying SSH keys to servers. SSH. Retrieved April 24, 2022, from https://www.ssh.com/academy/ssh/copy-id

u/mobbeduo (2019, February 21). Easy way to push a local file to a Git repo? [Reddit]. https://www.reddit.com/r/ansible/comments/asdxlo/comment/egy4vea/?utm_source=share&utm_medium=web2x&context=3

APPENDIX A

gitCommit.yml Code

```
---
- name: Git Add + Git Commit
  hosts: localhost
  vars:
  - destdir: ~/Desktop/CS504-Project-HessnerNicole/Submission1/CS504-IP01-AnsibleTest
  vars_prompt:
  - name: "commitMessage"
    prompt: Please input message to include with commit.
    private: no

  - name: "branchName"
    prompt: Please input remote branch name.
    private: no

  tasks:

  - name: Git update
    git:
      repo: 'https://{{gituser}}:{{gitpass}}@github.com/CityUhessnern/AnsibleCloneTest'
      dest: "{{destdir}}"
      update: yes
      version: master
    register: git
    ignore_errors: True
  - debug:
      var: git

  - name: Git - Adding
    shell:
      "git add --all"
    args:
      chdir: "{{destdir}}"
    register: gitadd
    when: git.msg is defined
  - debug:
      var: gitadd.cmd

  - name: Git - Commiting
    shell:
      git commit -m '{{commitMessage}}'
    args:
      chdir: "{{destdir}}"
    register: gitcommit
    when: git.msg is defined
  - debug:
      var: gitcommit.stdout

  - name: Git - Push
    shell: git push --set-upstream origin '{{branchName}}'
    args:
      chdir: "{{destdir}}"
    register: gitpush
    when: git.msg is defined
  - debug:
      var: gitpush
...
```

APPENDIX B

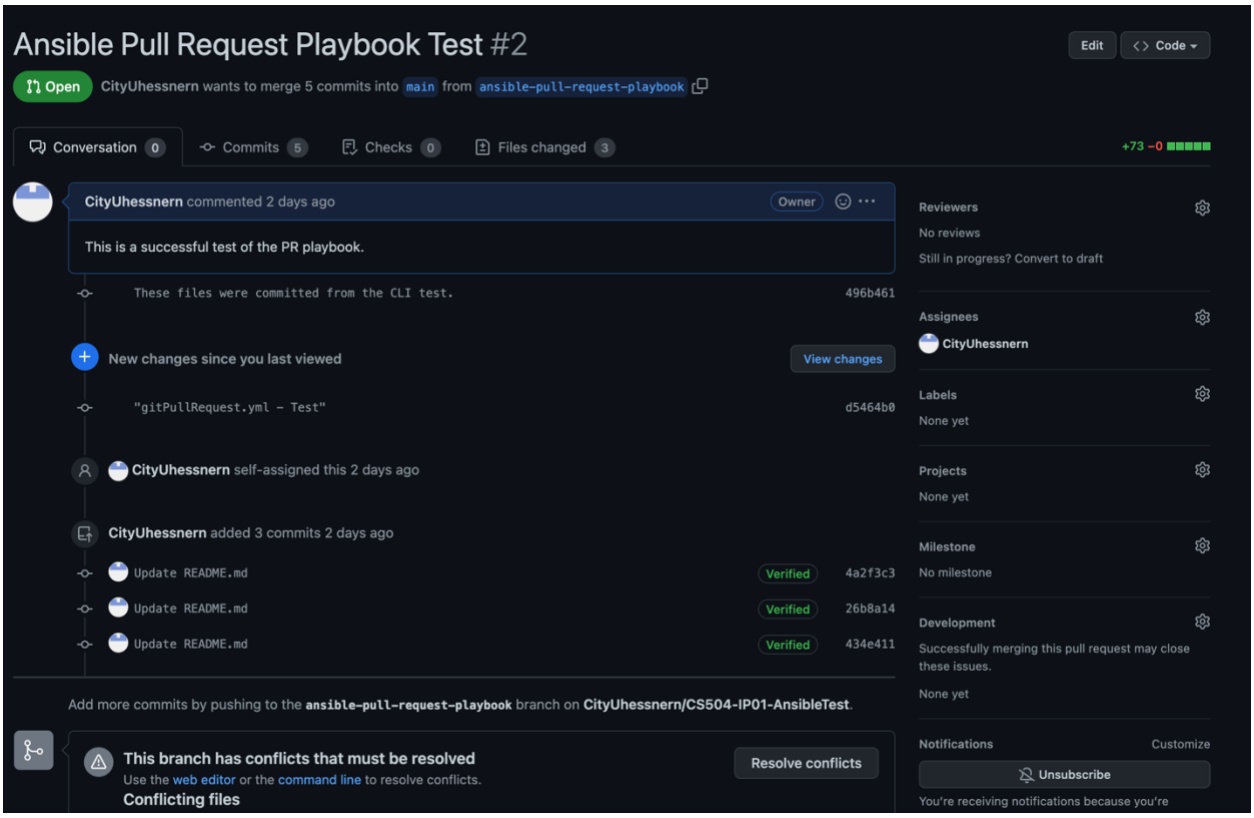
gitPullRequest.yml Code

```
---
        - name: GitHub Pull Request
          hosts: localhost
          vars:
          - destdir: ~/Desktop/CS504-Project-HessnerNicole/Submission1/CS504-IP01-
        AnsibleTest
          vars_prompt:
          - name: "prTitle"
            prompt: Please provide a title for the pull request.
            private: no

          - name: "commitBranch"
            prompt: Please enter the name of the branch you would like to use.
            private: no

          - name: "commitMessage"
            prompt: Please input message to include with commit.
            private: no

          - name: "prBody"
            prompt: Please provide more information about your pull requestion in the body.
            private: no

          tasks:

            - name: switchbranch
              shell:
                git checkout -b '{{commitBranch}}'
              args:
                chdir: '{{destdir}}'
              ignore_errors: yes
              tags:
                -always

            - name: Git - Adding
              shell:
                git add --all
              args:
                chdir: "{{destdir}}"
              ignore_errors: yes
```

```
    - name: Git - Commiting
      shell:
        git commit -m '{{commitMessage}}'
      args:
        chdir: "{{destdir}}"
      ignore_errors: yes
    - name: Git - Push
      shell: git push --set-upstream origin '{{commitBranch}}'
      args:
        chdir: "{{destdir}}"
      ignore_errors: yes

    - name: createPullRequest
      shell:
        gh pr create -a @me --title '{{prTitle}}' --body '{{prBody}}'
      args:
        chdir: '{{destdir}}'
...
```

APPENDIX C

Screenshot of Successful Creation of GitHub Pull Request with Ansible



APPENDIX D

gitMerge.yml Playbook

```
---
        - name: GitHub Pull Request
          hosts: all
          vars:
          - destdir: ~/Desktop/CS504-Project-HessnerNicole/Submission1/CS504-IP01-
        AnsibleTest
          - repo: https://github.com/CityUhessnern/CS504-IP01-AnsibleTest
          vars_prompt:

          - name: "commitBranch"
            prompt: Please enter the name of the local branch you would like to update.
            private: no

          - name: "commitMessage"
            prompt: Please input message to include with commit.
            private: no

          - name: "remoteBranch"
            prompt: Please input the name of the remote branch you would like to get
        updates from.
            private: no

          tasks:

            - name: switchbranch
              shell:
                git checkout -b '{{commitBranch}}'
              args:
                chdir: '{{destdir}}'
              ignore_errors: yes
              tags:
                -always

            - name: Git - Adding
              shell:
                git add --all
              args:
                chdir: "{{destdir}}"
              ignore_errors: yes

            - name: Git - Commiting
```

```
      shell:
        git commit -m '{{commitMessage}}'
      args:
        chdir: "{{destdir}}"
      ignore_errors: yes

    - name: Git - Pull
      shell: git pull origin '{{remoteBranch}}'
      args:
        chdir: "{{destdir}}"
      ignore_errors: yes
...
```

APPENDIX E

ghRelease.yml

```
---
        - name: GitHub Release and Git Tagging
          hosts: localhost
          vars:
          - destdir: ~/Desktop/CS504-Project-HessnerNicole/Submission1/CS504-IP01-
        AnsibleTest

          vars_prompt:
          - name: "commitMessage"
            prompt: Please input message to include with commit.
            private: no

          - name: "workingBranch"
            prompt: Please provide local working branch name.
            private: no

          - name: "pushBranch"
            prompt: Please provide branch name to push to.
            private: no

          - name: "version"
            prompt: Please provide a version tag.
            private: no

          - name: "tagMessage"
            prompt: Please provide a message for your tag.
            private: no

          - name: "releaseTitle"
            prompt: "Please provide a title for your release."
            private: no

          - name: "releaseNotes"
            prompt: "Please provide notes for your release."
            private: no

          tasks:
          - name: Git - Branch
            shell: git checkout -b "{{workingBranch}}"
            args:
              chdir: "{{destdir}}"
```

```
    register: gitcheckout
    ignore_errors: yes
#    when: git.msg is defined
#  - debug:
#      var: gitcheckout.stdout

  - name: Git - Tag
    shell:
      git tag -a "{{version}}" -m "'{{tagMessage}}'"
    args:
      chdir: "{{destdir}}/{{version}}"
    register: gittag
    ignore_errors: yes
#    when: git.msg is defined
# - debug:
#      var: gittag.cmd

  - name: Git - Adding
    shell:
      "git add --all"
    args:
      chdir: "{{destdir}}"
    register: gitadd
    ignore_errors: yes
#    when: git.msg is defined
# - debug:
#     var: gitadd.cmd

  - name: Git - Commiting
    shell:
      git commit -m '{{commitMessage}}'
    args:
      chdir: "{{destdir}}"
    register: gitcommit
    ignore_errors: true
#    when: git.msg is defined
#  - debug:
#      var: gitcommit.stdout

  - name: Git - Push Version
    shell: git push origin '{{version}}'
    args:
      chdir: "{{destdir}}"
```

```
    register: gitpushver
    ignore_errors: yes
#    when: git.msg is defined
#  - debug:
#      var: gitpushver.stdout

  - name: Git - Push
    shell:
      git push origin '{{pushBranch}}'
    args:
      chdir: "{{destdir}}"
    register: gitpush
    ignore_errors: yes
#    when: git.msg is defined
#  - debug:
#      var: gitpush.stdout

  - name: GitHub - Release
    shell:
      gh release create '{{version}}' -t "'{{releaseTitle}}'" --notes
"'{{releaseNotes}}'"
    args:
      chdir: "{{destdir}}"
    register: ghrelease

...
```

APPENDIX F

Successful Release Screenshot

APPENDIX G

ghFork.ynml Code

```
---
        - name: Git Add + Git Commit
          hosts: all
          vars:
          - destdir: ~/Desktop/CS504-Project-HessnerNicole/Submission1/AnsibleForkTest
          vars_prompt:
          - name: "forkName"
            prompt: Please provide a name for the fork.
            private: no

          tasks:

          - name: GitHub - Fork
            shell:
              gh repo fork --clone https://github.com/CityUhessnern/CS504-IP01-AnsibleTest
        --fork-name "'{{forkName}}'"
            ignore_errors: yes

        ...
```

APPENDIX H

Successful Fork Creation on Seconary Account