Using the local git repository for configuration

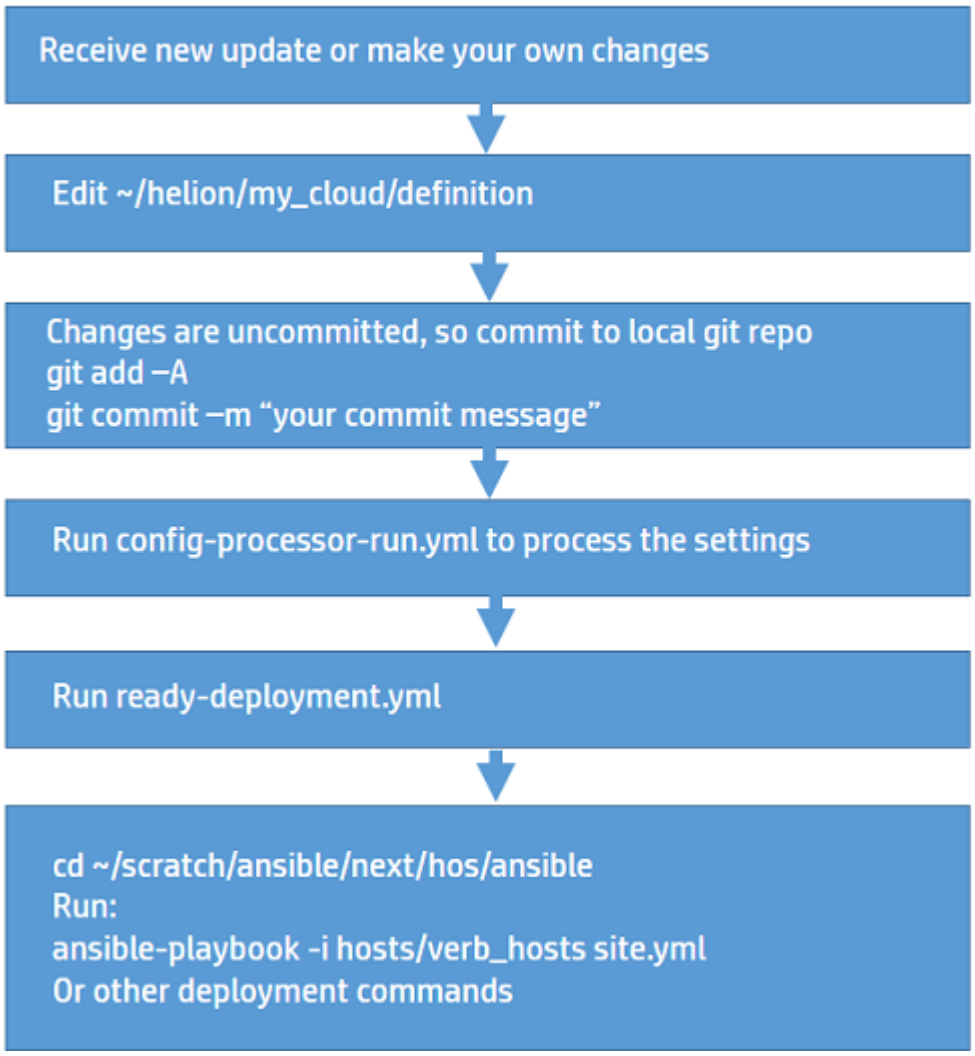
Contents

Using the local git repository for configuration

In HP Helion OpenStack® 2.0 versions, inclusding Beta1, a local git repository is used to track configuration changes and for the config processor to look to for configuration changes. The introduction of a git workflow also means that your configuration history is maintained, making rollbacks easier as well as keeping a record of previous configuration settings.

The git repository, which is installed by the deployer, is local to your cloud servers, but in other ways functions like a remote git repository.

Below is a representation of the typical git workflow in HP Helion OpenStack 2.0.



The git repository provides a way for you to merge changes that you pull down as "upstream" updates, and allows you to manage your own configuration changes.

Initialization on a new deployment

On a system new to HP Helion OpenStack 2.0, the deployer will prepare a git repository under ~/helion. The deployer provisioning runs the hlm-init-deployer script automatically; this calls ansible-playbook -i hosts/localhost git-00-initialise.yml.

As a result, the ~/helion directory is initialized as a git repo (if it's empty). It is initialized with four empty branches:

hos

This holds the upstream (ie, from HP) source code - that is, what you'll see under the ~/helion directory on a pristine fresh installation. Every source code release that's downloaded from us is applied as a fresh commit to this branch. This branch contains no customisation by the end user.

site

This branch begins life as a copy of the first 'hos' drop. It is onto this branch that you commit your configuration changes. It's the branch most visible to the end user.

ansible

This branch contains the variable definitions that the CP outputs that our main ansible playbooks need: the verb_hosts file (which describes to ansible what servers are playing what roles), amongst others. The ready-deployment playbook takes this output and assembles a 'scratch' directory containing the ansible playbooks together with the variable definitions in this branch; the result is a working ansible directory from which the main deployment playbooks may be successfully run.

cp-persistent

This branch contains the persistent state that the CP needs to keep hold of. That state is mostly the assignment of IP addresses and roles to particular servers. Some operational procedures may involve editing the contents of this brach: for example, retiring a machine from service or repurposing it.

Two temporary branches are created and populated at run time:

staging-ansible

This branch hosts the most recent commit that will be appended to the ansible branch.

staging-cp-persistent

This branch hosts the most recent commit that will be appended to the cp-persistent branch.

Note: The information above provides insight into the workings of the configuration processor and the git repository. However, in practice you can simply follow the steps below to make configuration changes.

Updating any configuration, including the default configuration

When you need to make updates to a configuration you must

1. Check out the **site** branch. You may aready be on that branch. If so, git will tell you that and the command will leave you there.

```
git checkout site
```

2. Edit the YAML file or files that contain the configuration you want to change.
3. Commit the changes to the **site** branch.

```
git add -A
git commit -m "your commit message goes here in quotes"
```

4. Run the configuration processor:

```
cd ~/helion/hos/ansible
ansible-playbook -i hosts/localhost config-processor-run.yml
```

   to produce the required configuration processor output from those changes. Review the output files manually if required.
5. Ready the deployment area

```
ansible-playbook -i hosts/localhost ready-deployment.yml
```

6. Run the deployment playbooks from the resulting scratch directory.

```
cd ~/scratch/ansible/next/hos/ansible
```

```
ansible-playbook -i hosts/verb_hosts site.yml
```