Harnessing your cluster with Ansible (hands-on)

Autor: Iñigo Aldazabal Mensa <inigo_aldazabal@ehu.es>

Date: 2015/02/04 - HPC Knowledge Meeting'15, Barcelona

Contents

1 Introduction

The way to manage the configuration of computing nodes in HPC clusters is normally through, first, the use of some kind of master image deployed to the nodes and, second, a "post configuration" stage in which the installed system is modified in order to adapt it to the changes made to this base image: modified SLURM configuration files, new filesystems to be mounted, updated packages, new monitoring tools to be installed, etc.

One way to deal with this post-configuration stage, and also with further changes which happen along the life of a computing node, is using a Configuration Management System - CMS. CMS's, such as CFEngine, Puppet, Chef, Salt, etc., are specifically designed to deal with system configuration changes and to mantain consistency in complex systems: they allow us to define nodes' service states, configuration files, packages installed, mount points, security policies and much more.

But this also comes with a price: a steep learning curve and the CMS system setup itself. Here we will present Ansible, a very easy to use CMS which, with its clientless (zero initial setup in the nodes) push model and the simple, human readable syntax of its YAML configuration files, perfectly fits the mindset of HPC cluster administrators.

We will walk through setting up a basic Ansible configuration for a simple computing cluster created with virtual machines. We will deploy this virtual cluster setup using a very simple Vagrant configuration for a head node and a few computing nodes.

We will trick ourselves and, as Vagrant allows us, use Ansible as the configuration "provider" for the virtual cluster. Thus we will take a look at the simple, one file, Ansible playbook used to initially configure mainly the Vagrant head node.

After instantiating the "cluster" we will examine the inventory file with nodes of different types, make some test runs and create a basic Ansible playbook for eg. install some packages, distribute some configuration files and configure some services both in the head and the computing nodes.

2 Required software

We will use VirtualBox based virtual machines deployed with Vagrant, so in order to follow the tutorial you must have both [VirtualBox]() and [Vagrant]() installed in your laptop.

- VirtualBox: install it from your distribution repositories or straigth from its webpage.
- Vagrant: Installing vagrant is trivial, just a matter of donwloading the installer/package and installing it. See the Vagrant install documentation.

After installing them get the VirtualBox machine template I prepared so that you have everything ready for the hands-on session. It's a Scientific Linux 6.5 minimal install Vagrant "box" (virtual machine template) I created myself. Just download it (almost 600MB) and add it to the Vagrant boxes:

```
wget https://dl.dropboxusercontent.com/u/49910137/scientific65_x86_64_minimal.box
vagrant box add --name SL-65-x86_64-minimal scientific65_x86_64_minimal.box
```

Clone the github repository so you have all the Vagrant and Ansible configuration files:

```
git clone https://github.com/iamc/HPCKP15-Ansible-hands-on.git
cd HPCKP15-Ansible-hands-on
```

3 Virtual Machines setup

Using Vagrant (checkout the `Vagrantfile` Vagrant configuration file) we will create a very simple virtual cluster with one head node and three computing nodes. Mimicking a real cluster, the head node will have both a "public" and "private" network interface while the computing nodes only belong to the "private" network. In this case we pretend that this is so for demostration purposes and they are all similar "host-only" VirtualBox interfaces.

Head node:	headnode
computing nodes:	node1, node2, node3

3.1 Vagrant "deploy"

In order to instantiate the virtual cluster we just invoke Vagrant:

> ***Note***
>
> Beware that you'll need at least 2GB of free memory for the four virtual machines we will create.

```
vagrant up
```

This takes some time (about 15 min. in a ThinkPad x220 with SSD) as we are deploying four virtual machines and also refreshing all of them repositorie's data, installing packages, etc. So be patient. If you plan to follow the hands-on session live think about doing it beforehand.

We are using Ansible itslef for the initial configuration of the virtual cluster, mainly for the head node as we want to manage the computing nodes later on from it. So while you wait for the initial deployment to finish its a good time to take a look at the main Ansible and Vagrant configuration files, `bootstrap/playbook.yml` and `Vagrantfile` respectively. Their syntax is so simple that they do not require further explanation. Take a look at them!.

> ***Note***
>
> Take your time and inspect the `bootstrap/playbook.yml` file. You'll see how the head node is configured by means of Vagrant using using Ansible as a provider, at bootstrap time. It will already give you an idea on how Ansible works.

Once the deployment ends the virtual cluster is already created. For convenience all the cluster machines have a `vagrant` user (password `vagrant`), which is also a passwordless sudoer.

We can check that the machines are created and running ok with:

```
vagrant status
```

To ssh into the machines Vagrant provides the command:

```
vagrant ssh <machine>
```

that logs us into the machine under the `vagrant` user. We can also use regular `ssh vagrant@...` connection if wanted.

We already deployed the ssh public keys and a known_hosts file using the bootstrap ansible playbook, so now let's just do an Ansible ping test using the ping module (`-m ping`). The hosts `inventory` file (`-i`, we'll see this later) is `ansible/hosts`, and `all` is recognized by default by ansible:

```
vagrant ssh headnode
ansible all -i ansible/hosts -m ping
```

All nodes should "pong" and after this we should be ready to play with ansible inside the "cluster".

4 Ansible

Let's login into the head node and work from there as we would do in a real cluster. We'll assume this from now on:

```
vagrant ssh headnode
```

and as we previuosly did, recheck that we can access all nodes:

```
ansible all -i ansible/hosts -m ping
```

4.1 Configuration files

We keep all Ansible configuration files under the `ansible` directory, which lives in the repository root directory and is exported by Vagrant into the hosts as /home/vagrant/ansible. This means that we can edit this directory both from the virtual machines and from our host computer. Thus, once you clone the github repository and bootstrap everything, the ansible configuration will already be accesible from the head (and the computing) node(s).

As one of the Ansible strengths is how easy is to read its configuration files we will just comment on how they are structured and which its funtion is and refer to them as they are self-explanatory.

Let's go into the ansible director and inspect it:

```
[vagrant@headnode ansible]$ ls -1 ansible/
check_mk-agent-1.2.4p5-1.noarch.rpm
cluster_production
cluster_stage
cluster.yml
computing.yml
head.yml
roles

[vagrant@headnode ansible]$ ls -1 ansible/roles/
check_mk
common
head
```

`cluster_production` and `cluster_stage` are the inventory files for the whole cluster and for the testing, staging nodes. The testing inventory only shows one node, the one we'll use for testing, namely `node1`.

The main playbook for deploying all the cluster is `cluster.yml` which itself just includes `head.yml` for the head node playbook and `computing.yml` for the computing ones.

The roles directory contain the Ansible "roles" which will be used for Ansible as defined in the corresponding main playbooks just mentioned. Take a look at it. In every role directory, `main.yml` is processed by ansible and all the variables/files whithin the role subfolders are directily accesible for any of its playbook files.

We will indicate whether we are deploying for testing or for production by means of the different hosts file.

4.2 Playbook deployment

The way to proceed is to modify the desired ansible playbook(s) and the test that everything works as expected using the testing computing node defined in `computing stage`:

```
ansible-playbook -i cluster_stage computing.yml
```

Once we are happy with this we can deploy it to all computing nodes just using the production inventory file:

```
ansible-playbook -i cluster_production computing.yml
```

We should, although did not do it here, set up also a head node testing system. Try it and see that nothing happens as we have not defined one:

```
ansible-playbook -i cluster_stage head.yml
```

And the same goes here; once happy with the changes just deploy:

```
ansible-playbook -i cluster_production head.yml
```

Now you can check that rerunning your paybooks cluster wide (head node + computing nodes) everything goes ok:

```
ansible-playbook -i cluster_production cluster.yml
```

Of course the Way To Go (TM) involves also using git, hg or so in order to keep track of the chages to the playbooks, inventory files, etc.

Now go around the playbooks and enjoy! Eg. modify the slurm file, and "deploy" it:

```
vim roles/common/files/slurm.conf
ansible-playbook -i cluster_stage computing.yml
```

Once we are happy with the new slurm.conf file we just deploy cluster wide:

```
ansible-playbook -i cluster_production cluster.yml
```

Now take a look at the files, go for the Ansible documentation and peek around and enjoy the play!

5 References

OMD documentation web page itself is very good and has modules documentation examples for various cases which many times you can just copy-paste to your playbooks with minimal adaption.

Also check out the github ansible tutorial colaboratively developed and that gives a very smooth entry path to Ansible, also using Vagrant for creating virtual machines, although you can use this tutorial virtual cluster if desired. It's structured in chapters with increasing difficulty.