Using Ansible to Manage your Pure Infrastructure

It has been some time since I wrote my last blog about integrating Pure Storage products into Ansible® and a lot has changed in that time, so I thought it was about time to impart some more information.

With the release of Ansible 2.8, there are now 14 FlashArray™ modules and 10 FlashBlade™ modules. Find more details of these in the Ansible documentation.

The modules for both FlashArray and FlashBlade can be broken down into two distinct sections: provisioning and configuration, and we will look at these separately.

Provisioning

FlashBlade

There is full support for FlashBlade provisioning whether that be for the file or object services.

For the file operation of the FlashBlade, there are modules for share creation, modification and deletion, including allowing access over different protocols for a share, such as HTTP, setting NFS access rules and configuring special directories. If you set the snapshot special directory, then you can use a module to manage these snapshots. If you are using the FlashBlade as an object store, there is a full suite of modules

to allow creation and management of S3 accounts, users, and bucket management.

FlashArray

The FlashArray modules do not, sadly, cover the full feature set of Purity; however, active development is ongoing to fill these gaps. The main missing piece is ActiveCluster™ integration. Expect to see this resolved with the release of Ansible 2.9. If we ignore ActiveCluster, then you can do anything using Ansible modules that you could do with the GUI or the Purity CLI. These modules include standard CRUD operations, snapshot management and recovery, protection group support (including asynchronous replication) and support for offload, also referred to as Snap to NFS and CloudSnap.

Configuration

The most important thing about the configuration of your Pure devices is understanding what they currently look like – so, there are facts modules for both the FlashBlade and FlashArray which populate a dictionary with the totality of a device current configuration.

These dictionaries can be leveraged to check your current configuration and make decisions on what your Ansible playbooks do using standard Ansible decision-making tools including Jinja2 templates.

FlashArray

There are a steadily growing number of Ansible modules that can be used to configure your FlashArray appliances, including the configuration of directory services, remote assist, DNS and NTP settings. More modules are being actively developed and expect to see these in Ansible 2.9, or in the devel branch of the Ansible GitHub repo should you wish to try out newer modules early.

FlashBlade

Current configuration modules cover the management of directory services, network and subnets. There will be more modules coming soon, so watch the Ansible devel GitHub repo.

Using the Ansible modules

Now that we know the modules we have available, how can you use them within the context of provisioning and configuration management?

One of the most critical questions I get asked about both the FlashArray and FlashBlade modules is how to manage multiple devices and also how to ensure that access information about the arrays, such as the API tokens, can be secured.

So let's have a look at what we can do with these modules, and also how we can secure information related to your Pure infrastructure so that sensitive configuration information does not fall into the wrong hands.

The first thing to note is that the Ansible nodes you run any Pure modules on require the appropriate Pure Python SDK installed on them, whether they are the localhost or remote nodes defined in an inventory. Specifically, you will need the `purestorage` SDK for FlashArray modules and the `purity_fb` SDK for FlashBlade modules.

Here is a short example of a pre-requisites playbook that will correctly install these SDKs on your Ansible

nodes:
[crayon-6424cdc4a8494696078701/]
Once you have ensured that your nodes are capable of running the Pure Storage Ansible modules, you can start to experiment with their functionality and capabilities.

It is important to note that each play in your playbooks that uses either FlashArray or FlashBlade modules needs to have information passed to them so that the play executes against the correct backend storage.

For FlashArray modules, the parameters `fa_url` and `api_token` are required, whereas for FlashBlade modules they are `fb_url` and `api_token`. These represent the IP address or FQDN of the FlashArray or FlashBlade management ports and an API token for a privileged account of the FlashArray or FlashBlade (both local accounts and those managed by a directory service configured on the storage array are supported).

These parameters can be passed to every module in the playbook individually as shown in this simple example playbook for creating a FlashArray volume and attaching it to a host:
[crayon-6424cdc4a849c232642726/]
You could create variables at the beginning of each playbook that can then be used in the plays as shown in this example:
[crayon-6424cdc4a849d923152867/]
If using Ansible roles then these variables could be defined in a variables YAML file.

Alternatively, you can define environment variables in the shell running the playbook that contains this information, in which case you do not have to add the URL or API parameters to the play. The environment variables that the modules look for if no parameters are provided in the play are `PUREFA_API` and `PUREFA_URL` in the case of a FlashArray module or `PUREFB_API` and `PUREFB_URL` for a FlashBlade module.

Some people don't like exposing specific parameters, especially those that may be considered secure. The API token provided to Ansible has to be associated with a FlashArray or FlashBlade user account that has Storage Admin privileges to perform all of the functions available in the Pure Ansible modules, so this is potentially a security risk. To ensure the security of this token, use Ansible Vault to encrypt it so that only Ansible administrators with the decryption password can implement the modules successfully.

Ansible Vault

I do not intend to describe how to configure Ansible Vault in any depth as there are lots of great blogs out there that do the same, and of course, there is the Ansible documentation as well.

In this example, I am going to use Vault to encrypt the API tokens for two different FlashArrays using an Ansible role I created called **array,** and then execute a simple playbook against both FlashArrays. The roles directory structure is:
[crayon-6424cdc4a849f747996286/]
Firstly we will look at the variable file, var/main.yaml:
[crayon-6424cdc4a84a2928957449/]
Here we see both encrypted and unencrypted information. The encrypted data is the sensitive API token for each of the two arrays. There are also a few other variables that we will be using in the playbook.

It is pretty simple to get those encrypted strings.

First I need to create a password that Vault will use to encrypt and decrypt variables, but this needs to be managed by your Ansible administrator. In this example, I have created a simple text file called `vault-password.yml` with the password as the only entry in it. Then I use the `encrypt_string` command provided by Vault to encrypt the information I want to secure, in this case, the API tokens for the arrays:

```
# ansible-vault encrypt_string --vault-id vault-password.yml --stdin-name api
```

The command will prompt for the string I want to encrypt and then produce an output similar to this
[crayon-6424cdc4a84a3727381926/]
which is then just cut and paste into the variables file. I did this for each API token.

Moving on to the primary role playbook; tasks/main.yaml which contains:
[crayon-6424cdc4a84a4216804840/]
This playbook is telling the Ansible to use the tasks file`all.yaml` and pass the variables `url` and `api` to this, but these are looping through the items called arrays which we saw defined above as the two arrays and their associated URL and encrypted API information.

Finally, we have `tasks/all.yaml` which is doing the actual work of running modules against the FlashArrays:
[crayon-6424cdc4a84a5658764266/]
Although this looks a little complex, it is just performing the following tasks against each array:

- Get the current configuration of the FlashArray;
- Set a fact for the actual name of the array (purely for informational purposes);
- Check the current DNS settings of the array and compare to the value defined in the vars/main.yaml. If they don't match then set the DNS settings on the array to be correct;
- Check the current NTP settings of the array and compare to the value defined in the vars/main.yaml. If they don't match then set the NTP settings on the array to be correct.

To execute these modules is a simple matter of creating a playbook (`infra.yaml`) to call the role array:
[crayon-6424cdc4a84a7435730721/]
And then run the following command:

```
# ansible-playbook infra.yaml --vault-id vault-password.yml
```

In this run, both of the arrays have the correct DNS and NTP settings, so nothing changes:
[crayon-6424cdc4a84a8754605710/]
One final thought for this. You could schedule runs using Ansible Galaxy to ensure you don't get configuration drift on your arrays, but also as a quick way to configure the infrastructure setting of a new array that you have added into your environment.

Remember development of more infrastructure modules for Pure is ongoing in the devel branch of the Ansible repo on GitHub, so keep a look out there, but if you feel a specific module is missing or a current module doesn't do what you expect, please feel free to comment here or raise an Issue on the Ansible repo. Pure also has a public Slack community called Pure/Code() with a specific Ansible channel where messages and comments can also be left.