platform-ansible

Tom Gidden, Starberry Ltd
<gid@starberry.tv>
September 2024

The `platform-ansible` repo is a collection of Ansible roles and playbooks for provisioning and maintaining a Linux-based hosting platform familiar to Starberry.

It is an adaption of the earlier `ansible` repo, but with an emphasis on a slimmer and more modular platform with many services based around Docker.

There is no UI, but it can be easily integrated into a UI or used with an off-the-shelf tool.

Installation

Prerequisites

- Python 3.12+ recommended
- Git
- Access to the GitHub repository
- doctl and/or Azure CLI, or preconfigured keys.

First use

As the repo necessarily contains powerful credentials to manage our entire platform, **do not install it on a cloud server**; instead, install it on your private workstation or on a firewalled office server. **The computer with the Ansible repo you use should not be accessible on the public internet**.

On your local machine, get the repo and run `make setup`:

```
git clone git@github.com:starberry/platform-ansible
cd platform-ansible
make setup
```

or alternatively, if you prefer use *Poetry*, or another virtual environment manager, feel free. I've found that with Ansible, dependencies can get messed up if you use the system Python, so it's best to use some sort of isolation, and `venv` is the quickest to set up.

If you want to work with Azure resources, also do `make azure`

Subsequent use

Each time you use this repo, you'll need to run the following command to activate the Python environment:

```
. ./venv/bin/activate
```

Configuration

- Set up credentials in `group_vars/all.yml`
- Set up `DO_API_TOKEN` in your environment.

Dynamic or static inventory

You have a choice between using a dynamic inventory where the list of servers is generated on use, or a static inventory where you have a list of servers.

You can generate a static list of servers using `python ./digital_ocean.py -p > ./inventory/digital_ocean.json`, or the superior do-ansible-inventory.

Or you can use the dynamic inventory, but it can be slow. It can cache results, but you'll need to deal with that appropriately – eg. the server you just created won't appear in the cached list! The configuration for this is in `./inventory/digital_ocean.yml`. If you don't want to use it, delete that file.

You can clear the cache by adding `--flush-cache` into a playbook command.

Test it:

```
ansible mon1 -i inventory -m ping
```

- (TODO: Azure configuration, using Azure CLI)

Usage

Okay, this should be enough to get going with Digital Ocean.

The typical usage of a playbook to run on all servers is:

```
ansible-playbook -i invento 0 === 1 ry -vv ./playbook_WHATEVER.yml
```

and on a specific server or servers (assuming the playbook uses the `all` group):

```
ansible-playbook -l ggfx -i inventory -vv ./playbook_WHATEVER.yml
```

or on a specific IP or host, using an "ad-hoc" inventory (remember to use the comma!):

```
ansible-playbook -i mon1.dx.starberry.io, -vv ./playbook_WHATEVER.yml
```

You can use tags to group servers, as Droplet Tags are automatically converted into Ansible groups, so:

```
ansible-playbook -l sysops -i inventory -vv ./playbook_update_customer.yml
```

(Unfortunately the structured tags like `c:foobar` don't parse properly, so you just have to use `foobar` instead.)

Playbooks

A number of playbooks have been created for common tasks, as well as the individual roles needed to implement them.

A very common one is: `playbook_droplet_new_basic.yml`, which will set up a Digital Ocean Droplet (ie. cloud server, ie. VM) with prompts.

```
ansible-playbook -vv ./playbook_droplet_new_basic.yml
```

This will ask for:

- A name for the server, conforming to Starberry naming conventions, eg. `web-fineandcountry-1`; `db-fineandcountry-1`

- A DigitalOcean project to associate it with. We have previously just used a single project, but I recommend using customer-specific projects, eg. `fineandcountry`. *Do not add* `-dev` *or similar to the end.*

- The server spec, eg. `s-1vcpu-1gb` for a low-end server. Use command `doctl compute size list|sort` for available choices.

- The OS image to use, eg. `ubuntu-24-04-x64`.

  Use command `doctl compute image list-distribution | grep ubuntu` for available choices.

  It *is* possible to enter the slug of a backup or snapshot, but this is not recommended for this playbook, as it won't just restore the backup: it'll also try to reconfigure it as a new server, which will probably fail!

- The region. Use `lon1` unless you have a good reason otherwise.

It'll take some time to run, but you should get a fully-configured server at the end.

You can use the `playbook_droplet_delete.yml` playbook to delete a server, although be careful: there's no Undo.

Azure

There's a similar playbook for Azure: `playbook_azurevm_new_basic.yml`, but it hasn't been tested as much as the Digital Ocean playbook.

Azure account configuration is more complex – it's somewhere between Digital Ocean and AWS in complexity – so we haven't used Azure servers in production (at the time of writing) However, they do seem to work well and function very much like Digital Ocean servers.

Cleaning up afterwards is harder as Azure creates a lot of resources. As a result it's **far more important to use an appropriate resource group.** That way, you can use `playbook_azurevm_delete_rg.yml` to delete the whole installation in one go.

Note, the Azure playbooks/roles additionally use the `platform` resource group which is used to maintain the `zx.starberry.io` and `zi.starberry.io` domains. **Do not** delete these resource groups!

Azure setup note

Incidentally, for the first use of the Azure playbooks, you'll have to create the `platform` resource group and set up those domains, as the Azure account/tenant/whatever that I was using for testing has expired. This should be a one-time task.