developer.skatelescope.org Documentation

Release 0.1.0-beta

Marco Bartolini

Jul 06, 2020

Home

Documentation Status

BIFROST Is For Running Operations of the System Team.

This project is supported by an Ansible playbook that creates a Management Server, which is hosted on an Openstack cluster. Once the Bifrost is active, any member of the System Team can *connect to the server* in order to do routine maintenance, setting up of new kubernetes clusters, etc.

CHAPTER 1

Requirements

The machine used for development of this repository needs to have Python 3 and `pip` installed. You should also install Ansible:

```
sudo apt-add-repository --yes --update ppa:ansible/ansible && sudo apt-get install␣
↪ansible -y
```

In case you need to recreate an instance of the BIFROST from scratch, you also require the Openstack RC file that can be retrieved from the admin page. **NB: Make sure you download the V3 and not the V2 file!**

CHAPTER 2

Environment setup

2.1 Repository and Python packages

To use and update the BIFROST repository, you need to clone it and install the required python packages first:

```
git clone git@gitlab.com:ska-telescope/sdi/bifrost.git
cd bifrost
make install # or make reinstall if role collections have been upgraded
virtualenv venv && . venv/bin/activate
pipenv install #assumes you have Pipenv, otherwise use pip install -r requirements.txt
```

2.2 Ansible Galaxy

The playbook also makes use of Ansible Galaxy for installing common roles from the Nexus-hosted Galaxy repository. For simplifying this process, the `make install` target is included. You may notice a `.gitignore`'d directory added by this target, called `collections`. You can also use `make reinstall` to uninstall and reinstall these common roles. Refer to the system-common-roles repository for how to publish Galaxy roles for re-use.

2.3 Gitlab Runner

For testing the execution of the Gitlab CI/CD pipeline stages locally, a `docker-executor` Gitlab runner can be installed locally. Since you will be downloading protected Git repositories as part of the pipeline execution, you also need to get the Personal Access Token from your Gitlab Profile. Set the value of `CI_JOB_TOKEN` in `PrivateRules.mak` to this token.

CHAPTER 3

Using this repository

3.1 Create a new BIFROST

To create a new server (for development & testing, or if you need to rebuild production), follow the above steps on *Requirements*, *Python packages* and *Ansible Galaxy*.

3.1.1 TL;DR

Once your environment is set up, run `make bridge_of_rainbows` to create a new BIFROST from scratch. Or, if you just want to update configuration, call `make bifrost`. **Make sure your `PrivateRules.mak` is populated correctly!** See *here* for details.

3.1.2 Openstack connection

You need to use your own Openstack keypair name (which is most likely also your username) and password, and your own RC file. Set the value of `CLUSTER_KEYPAIR` to this keypair name in your `PrivateRules.mak` file:

```
echo "CLUSTER_KEYPAIR=<your-own-keypair>" >> PrivateRules.mak
```

Refer to the section on *make targets and variables* below for the rest of the variables that may be useful during development.

Source your Openstack RC file and test your connection to the Openstack API:

```
. openrc.sh
make open
```

Now create a new VM:

```
make build_node
```

This role creates a VM on the Openstack cluster, which is managed as a heat stack. A new inventory file is created by this role and this inventory file will be used by all the other playbooks called via make targets. The filename of this inventory file is a `make` variable `INVENTORY_FILE`. The VM can be destroyed using the `make clean_nodes` target - be careful - this is irreversable. The "are you sure?" protection is only bypassed in the `make ragnarok` target - familiarise yourself with it's functionality.

3.1.3 Common roles and docker

Install common packages and docker:

```
make build_common
make build_docker
```

3.1.4 Configure BIFROST

This is the part that may be updated as more functionality is added to the BIFROST and should be tested regularly. The `setup_bifrost.yml` playbook is called with all the necessary parameters, by running

```
make bifrost
```

3.2 Updating the BIFROST

Running the complete BIFROST setup can take some time. If you only need to add a minor update to the configuration, or if you need to test or add only minor functionality, you may want to limit the amount of work done. Set the skip-tags variable like so:

```
make bifrost SKIP_TAGS=ansible,openstack,repos,ssh,k8s,kubeconfig
```

The above set of skipped tags will cause the whole playbook to be skipped at the moment - removing tags will switch on those roles. Tagging your newly added roles appropriately will make it simple to manage - easiest will be to set the variable in your *PrivateRules.mak* file.

If you want to check which roles are going to be executed by calling `make bifrost`, you can call `make plan` - Ansible will list all the tasks that will be executed.

CHAPTER 4

make targets and variables

The command `make` (or `make help`) will output all the available make targets and the set defaults. There are "nickname" targets also included, but because they are not commented, they will not be output when you run this target.

The following targets and variables are important for understanding how the playbooks fit together, testing and running it efficiently:

4.1 Local (Private) variables

Populate your `PrivateRules.mak` file with something like

```
# For creating a new VM
CLUSTER_KEYPAIR=your-openstack-key-name
PRIVATE_VARS=./dev_bifrost_vars.yml # Change this and the file only if you plan to␣
↪change specs for the development server
INVENTORY_FILE=./inventory_bifrost # This inventory file is created when running the␣
↪make build target

# optional during development / testing
BIFROST_IP=127.0.0.1 # In case you are building a BIFROST development VM, for␣
↪instance using VirtualBox
CI_JOB_TOKEN=RandOMl3tt3r5DOWnL_dedFroMgitlab_com # For CI Pipeline testing
EXTRA_VARS="mode=test" # This helps skipping some desctructive steps on the BIFROST
SKIP_TAGS=ansible,openstack,repos,ssh,k8s,kubeconfig # makes testing small bits of␣
↪the playbook a breeze - leave blank for running the complete suite
GIT_BASE=https://gitlab-ci-token:$(CI_JOB_TOKEN)@gitlab.com/ska-telescope # needed␣
↪for testing Pipeline execution - only include if you want to run the rtest targets␣
↪(pipeline stage tests)
MOLECULE_SCENARIO_NAME=default # change to the directory name of the scenario you␣
↪want to test.
```

(continues on next page)

(continued from previous page)

```
# Usage parameters
SSH_USER=<your-first-name> # a user account exists for each user, add your SSH␣
↪details to the distribute-ssh-keys repository
```

CHAPTER 5

Using the BIFROST

5.1 Connect

Connecting to the BIFROST is simplified in this repository, for System Team members. Your SSH credentials need to be on the BIFROST, and by the same mechanism it should be on all the other VM's in the Openstack cluster. You should connect with SSH using Agent Forwarding:

```
ssh -A <your-username>@<the-bifrost-IP-address>
```

If you are a creature of comfort, you might as well just use the make target:

```
make heimdall SSH_USER=ubuntu
```

This connects you to the BIFROST with your SSH agent forwarded, which means you now have the same access you normally would have, to all the machines. The success of your SSH Agent Forwarding is tested at login, and if it fails, you'll be told. ***NOTE: A user profile has been created for each member of the System Team, so it is unnecessary to connect as `ubuntu`. Set this SSH_USER parameter in `PrivateRules.mak`.***

5.2 Manage Openstack VM's

As mentioned before, once you're on the BIFROST, you should pay attention to the bash login output - some valuable information is given. Notably, the directories with configuration and repositories that were downloaded are listed.

You can use the common names of all the servers (as they are labelled on Grafana) to SSH into them, for instance:

```
ssh tmc-runner-1-gitlabrunnernode-msmhbqxbx3dh
```

Refer to ~/.ssh/config for all the common VM names.

5.3 Kubernetes management

You can manage the kubernetes cluster from here - refer to the output of the SSH welcome screen:

```
KUBECONFIG has been set to connect to syscore
using /srv/config/syscore_kubeconfig:
    server: https://192.168.93.102:6443
```

5.4 Managing Openstack machines

When logging in as `ubuntu` user, you are asked for Openstack credentials. This step is skipped when logging in as yourself, but a reminder is given:

```
To connect to Openstack, run:
$ source /srv/config/openrc.sh
and follow the prompts.
```

Once this has been done, the servers can be managed. Getting a list of Openstack servers available, for instance, is done by

```
openstack server list
```

5.5 Software Defined Infrastructure repositories

All the important SDI repositories are already cloned onto the BIFROST. If a new machine is created, the relevant inventory file should be updated accordingly.

#TODO: automate updating the central inventory.

CHAPTER 6

Testing the Ansible Playbook

The playbooks included in this repo are tested with pytest-molecule. Molecule is a project that helps with development and testing of Ansible roles. In this repository there are `make` targets for running these tests locally as well as bypassing the buildup and teardown steps:

```
make rtest # runs CI stages on gitlab runner - see below
make test # this will run pytests locally and is used by the test stage of the CI␣
↪Pipeline
make molecule
make verify
make destroy
```

6.1 Local testing

For testing locally (using molecule[docker]), it is perhaps necessary to understand how we deviate from the normal use of Molecule for developing and testing roles in Ansible. For the general case, a call to `molecule init role my-new-role` will create a directory called `my-new-role`, containing an extensive list of directories and files that can be modified for development and testing. The testing code for a role is all inside the `default` directory, and therefore contained inside the directory containing the role:

```
.
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── molecule
│   └── default
│       ├── INSTALL.rst
```

(continues on next page)

(continued from previous page)

```
│       ├── converge.yml
│       ├── molecule.yml
│       └── verify.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

The `molecule.yml` file contains only some basic outlines for the testing and tests are driven from the '/tests/test/yml' file. For more information about how to develop roles using Ansible, a useful video exists.

In this repository, we have a molecule directory inside our `tests/` directory. There are subdirectories for different scenarios, and they are all picked up when running `pytest` from within this directory. However, when we run `molecule test` like we do in the `make molecule` target, only a specified scenario is picked up (if left unspecified, the default scenario will always run). When `pytest` discovers the scenarios, they are run in alphabetical order. This means that, by splitting out the test scenarios and making a destroy scenario run at the end, we can test and develop scenarios separately. It also means that, by not destroying the container against which we test, we can increase our speed of development dramatically.

6.1.1 Adding a test scenario

As an example, we will add a verification step to check if Helm was installed. We will use an already developed Ansible collection that does the installation of all kinds of k8s related tooling, called `ska-systems_k8s`.

When developing a new feature, it is most likely necessary to create a new scenario. In terms of the user story "Install kubernetes tooling", Helm installation is only one of the tasks that should fulfill this story, and testing this could ideally reside in the `ska-systems_k8s` repository, but we'll use it as a small example here. If no related test scenario exists yet, it is likely a good time to create a new one. Check the alphabetical names of the roles - at the time of writing we have t01XXX and t02XXX scenarios already defined, so we will name our k8s scenario t03k8s.

Creating a new scenario can be done with the `molecule init` command, which creates a directory and boilerplate files for this new scenario:

```
../bifrost/tests $ molecule init scenario t03k8s
--> Initializing new scenario t03k8s...
Initialized scenario in /usr/src/bifrost/tests/molecule/t03k8s successfully.
../bifrost/tests $ tree molecule/t03k8s
molecule/t03k8s/
├── INSTALL.rst
├── converge.yml
├── molecule.yml
└── verify.yml
```

Prerequisites

- You will notice a simplistic `molecule.yml` file - replace the contents with that of the previous scenario (such as t02repos).
- Immediately change the scenario name in `molecule.yml` after copying it:

```
scenario:
  name: t03k8
```

- Add missing details around the provisioner, such as environment variables required:

```
provisioner:
  name: ansible
  options:
    vvv: True
  config_options:
    # defaults:
    #   library: Library
    ssh_connection:
      scp_if_ssh: True
  env:
    ANSIBLE_ROLES_PATH: "../roles"
    ANSIBLE_COLLECTIONS_PATHS: "../../../collections"
    USER: ubuntu
    # GIT_SSH_COMMAND: "ssh -i /root/id_rsa_for_git -F /dev/null"
  inventory:
    host_vars:
      # setting for the platform instance named 'ubuntu1804_bifrost'
      ubuntu1804_bifrost:
        ansible_user: root
```

- Add collection that should be installed to the `ska.systems` namespace to `/molecule/default/collections.yml`:

```
---
collections:
...
- name: https://nexus.engageska-portugal.pt/repository/ansible-roles/systems-common-
↪roles/ska-systems_k8s-0.2.1.tar.gz
  version: 0.2.1
```

Converge

The converge step in molecule is an emulation of running our playbook against the container. Any special variables required to allow this step to work can be included here.

- Ensure that the converge step is uncommented in `/t03k8s/molecule.yml`:

```
scenario:
  name: t03k8s
  test_sequence:
    # - lint
    # - destroy
    # - dependency
    # - syntax
    # - create
    # - prepare
    - converge
    # - side_effect
    - verify
    # - destroy
```

- Add the new collection to the converge step in `/molecule/t03k8s/converge.yml`. This will install the collection when molecule executes the converge.yml playbook:

```
collections:
- ska.systems_k8s
```

- Add variables used for this role (that are also to be used by the playbook setup_bifrost.yml):

```
vars:
  # helm vars
  helm_version: v3.1.2
  helm_name: helm
  helm_mirror: https://get.helm.sh
  helm_stable_repo_url: https://kubernetes-charts.storage.googleapis.co
```

- Execute the Helm installation role:

```
tasks:
- name: "Include ska.systems_k8s.helm (Install Helm)"
  include_role:
    name: "ska.systems_k8s.helm"
```

Verification

Now we need to verify that our execution of the roles under test actually worked - in this case, was Helm installed, and does it work, etc. We put our assertions under `verify.yml`:

```
```

6.2 Test on Gitlab Runner

If you have your *Gitlab Runner* set up on your development machine, you can test any pipeline stage by calling

```
make rtest STAGE=test
```

`STAGE` is by default set to `test`, so you don't need to include that in your command.

A docker-executor gitlab runner executes the stage, which in turn creates a docker container on which the complete BIFROST will be created and verified.