developer.skatelescope.org Documentation

Release 0.1.0-beta

Marco Bartolini

Aug 18, 2020

CONTENTS:

This project builds a generic Kubernetes cluster based on dynamic inventory generated by https://gitlab.com/ska-telescope/sdi/heat-cluster, which in turn relies on ansible collections from https://gitlab.com/ska-telescope/sdi/systems-common-roles.

CHAPTER

ONE

README

Deploy a Kubernetes cluster with Ansible.

1.1 Summary

This repo builds a generic Kubernetes cluster based on dynamic inventory generated by https://gitlab.com/ska-telescope/sdi/heat-cluster, which in turn relies on ansible collections from https://gitlab.com/ska-telescope/sdi/systems-common-roles .

Checkout cluster-k8s (this repo) and pull the dependent collections with:

```
git clone git@gitlab.com:ska-telescope/sdi/cluster-k8s.git
cd cluster-k8s
make install
```

This then needs an ansible vars file that describes the cluster to be built, and to know where to write out the dynamic inventory describing the nodes in the cluster.

1.1.1 Ansible vars for the cluster

In order to define the architecture of the cluster, one needs to describe, at a minimum, the *loadbalancer*, *master* and *worker* nodes of which the cluster will be comprised of.

In dev_cluster_vars.yml, it is described a cluster containing 1 *loadbalancer* (there is always just 1), 1 *master* and 1 *worker*. Masters can be scaled out in odd numbers. Workers can be any number. It is suggested to use this file as a starting point for defining and creating a new cluster.

Certificate Management

Although there is a predefined certificate defined in dev_cluster_vars.yml, it is suggested that you generate your own `k8s_certificate_key`, with `kubeadm alpha certs certificate-key`.

For more information on the subject read Certificate Management with kubeadm.

1.1.2 Ansible inventory file

Creating a new cluster by using this playbook, will add the necessary entries, generated by heat-cluster, specifying the new machines in inventory_dev_cluster.

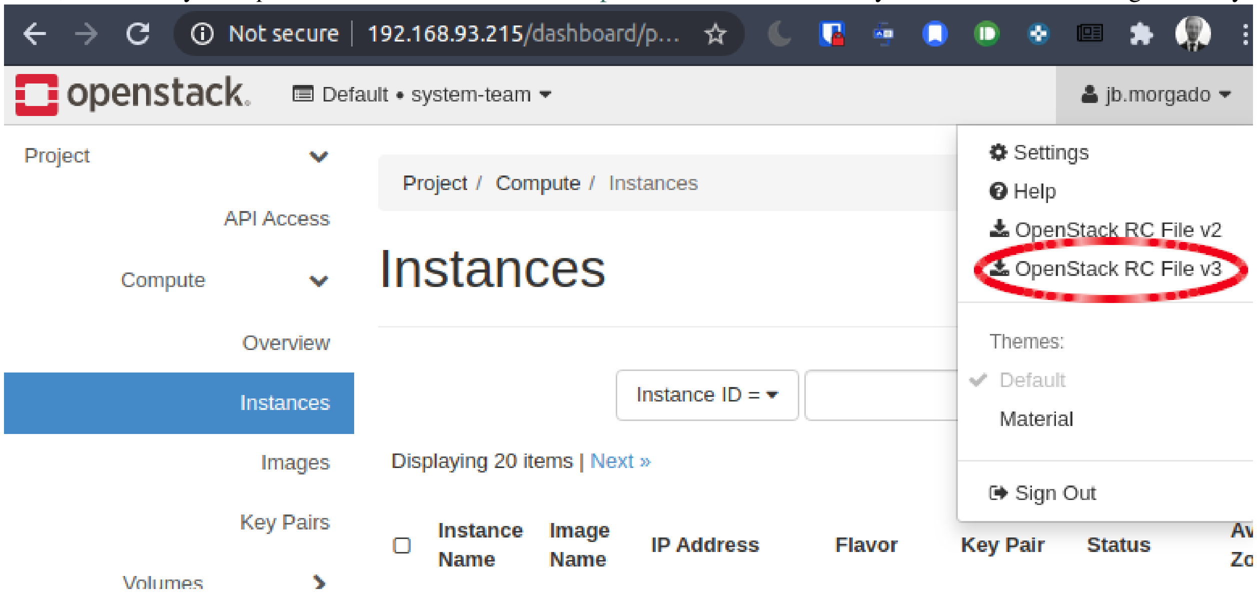
These entries can then be used to add the necessary ssh keys for access to the newly created cluster with the use of distribute-ssh-keys (more on that later).

1.1.3 Setting your OpenStack environment

First you will need to install openstacksdk in order to be able to create the VMs using openstack (using heat-cluster).

```
sudo apt install python-openstackclient
```

Then, download your *OpenStack RC File v3* from the OpenStack Dashboard into your *cluster-k8s* working directory:



Change the name of the **-openrc.sh* file for easier usage:

```
mv *-openrc.sh to openrc.sh
```

Then source *openrc.sh* and test connectivity (it will ask you for a password, use your OpenStack access password):

```
source ./openrc.sh
openstack server list
```

The openstack environment variables (set by the `openrc.sh` file) default values are:

```
OS_AUTH_URL: "http://192.168.93.215:5000/v3/"
OS_PROJECT_ID: "988f3e60e7834335b3187512411d9072"
OS_PROJECT_NAME: "system-team"
OS_USER_DOMAIN_NAME: "Default"
OS_PROJECT_DOMAIN_ID: "default"
OS_REGION_NAME: "RegionOne"
OS_INTERFACE: "public"
OS_IDENTITY_API_VERSION: 3
```

The variables with no defaults are:

- **OS_USERNAME**: the openstack username of the user who will access the openstack api
- **OS_PASSWORD**: the openstack password of the user who will access the openstack api

Also, you need to make sure that the OpenStack account used has the `stack_owner`, as well as the `member` (sometimes `_member`) roles.

1.1.4 Referencing custom configuration

In order not to have to specify the `CLUSTER_KEYPAIR`, `PRIVATE_VARS` and `INVENTORY_FILE` variables every time a make command is issued, create the *PrivateRules.mak* file in the cluster-k8s root directory specifying these three variables (**these need to be changed accordingly to each specific purpose**):

```
CLUSTER_KEYPAIR=your-openstack-key-name
PRIVATE_VARS=./dev_cluster_vars.yml
INVENTORY_FILE=./inventory_dev_cluster
```

Note: `your-openstack-key-name` *should be the one defined under OpenStack Key Pairs.*

1.1.5 Running the build

The build process is broken into two phases:

- create the cluster of VMs, and do the common install steps
- build the HAProxy loadbalancer and the Kubernetes Cluster

This can be run simply with:

```
make build
```

Or, in case you didn't define the *PrivateRules.mak* with:

```
make build CLUSTER_KEYPAIR=your-openstack-key-name PRIVATE_VARS=./dev_cluster_vars.
↪yml INVENTORY_FILE=./inventory_dev_cluster
```

You can also break the `make build` command into the three steps comprising it:

```
# create the VMs and do the common install
make build_nodes
# build the loadbalancer
make build_haproxy
# build the Kubernetes cluster
make build_k8s
```

1.1.6 Deploy the SSH access keys on the newly created cluster

After creating the cluster, only the user issuing the build commands specified above, will have access to the cluster.

In order to have a group of users (usually your team) being able to login into the various VMs that were created, it is needed to distribute their respective ssh keys into those VMs.

To do this, one needs to use the functionality provided by the *distribute-ssh-keys* SKA repository.

It is beyond the scope of this README to explain all the functionality of the *distribute-ssh-keys* repo, but for this specific purpose one needs to accomplish the following steps:

1: Modify the *inventory-ssh-keys* in the *distribute-ssh-keys* repo

When issuing the build command(s) specified above, the file specified by the *INVENTORY_FILE* variable (*inventory_dev_cluster* by default) is automatically updated with the newly created VMs:

```
[cluster:children]
k8s_dev_cluster_loadbalancer
k8s_dev_cluster_master
k8s_dev_cluster_worker

[k8s_dev_cluster_loadbalancer]
k8s-dev-cluster-loadbalancer-0 ansible_host=192.168.93.137 docker_vol_diskid=
↪"5fca38ce-3edc-4fe8-b" data_vol_diskid="11c94e16-8482-47bc-a"  data2_vol_diskid=""

...

[k8s_dev_cluster_master]
k8s-dev-cluster-master-0 ansible_host=192.168.93.106 docker_vol_diskid="0b53dc74-5709-
↪46c8-b" data_vol_diskid="30e1928c-3bfd-43c4-b"  data2_vol_diskid=""
k8s-dev-cluster-master-1 ansible_host=192.168.93.130 docker_vol_diskid="e8c4b3c3-8166-
↪4cc6-b" data_vol_diskid="ce5116d7-3c87-40af-9"  data2_vol_diskid=""
k8s-dev-cluster-master-2 ansible_host=192.168.93.24 docker_vol_diskid="cd976a69-d37b-
↪4492-b" data_vol_diskid="b48fa69c-37e1-4a18-9"  data2_vol_diskid=""

...

[k8s_dev_cluster_worker]
k8s-dev-cluster-worker-0 ansible_host=192.168.93.125 docker_vol_diskid="e75f9f0f-ccaa-
↪46a9-a" data_vol_diskid="d98e0d88-8b57-47c7-9"  data2_vol_diskid=""
k8s-dev-cluster-worker-1 ansible_host=192.168.93.119 docker_vol_diskid="db5236ce-4769-
↪47bb-8" data_vol_diskid="76b5aba6-af41-4416-b"  data2_vol_diskid=""
k8s-dev-cluster-worker-2 ansible_host=192.168.93.85 docker_vol_diskid="92f1f155-2c6c-
↪494d-a" data_vol_diskid="3eeef438-f6cd-44a4-a"  data2_vol_diskid=""

...

# Specific roles for cluster deployment assignments
[cluster_nodes:children]
k8s_dev_cluster_loadbalancer
k8s_dev_cluster_master
k8s_dev_cluster_worker
```

In this particular example, you need to add these newly created machines into the *inventory-ssh-keys* in the *distribute-ssh-keys* repo by creating a new section similar to the following:

```
# K8s dev cluster - k8s-dev-cluster
[k8s_dev_cluster_loadbalancer]
k8s-dev-cluster-loadbalancer-0 ansible_host=192.168.93.137

[k8s_dev_cluster_master]
k8s-dev-cluster-master-0 ansible_host=192.168.93.106
k8s-dev-cluster-master-1 ansible_host=192.168.93.130
k8s-dev-cluster-master-2 ansible_host=192.168.93.24

[k8s_dev_cluster_worker]
k8s-dev-cluster-worker-0 ansible_host=192.168.93.125
k8s-dev-cluster-worker-1 ansible_host=192.168.93.119
k8s-dev-cluster-worker-2 ansible_host=192.168.93.85
```

(continues on next page)

(continued from previous page)

```

[k8s_dev_cluster:children]
k8s_dev_cluster_loadbalancer
k8s_dev_cluster_master
k8s_dev_cluster_worker

[k8s_dev_cluster:vars]
ansible_python_interpreter=python3
ansible_user=ubuntu
```

2: Add your keys to the *ssh_key_vars.yml* in the *distribute-ssh-keys* repo

You then need to specify which keys will be distributed among the new nodes. In order to do so, you add a new entry to the *ssh_key_vars.yml* file, like such:

```
# Bruno
  - ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAADAQABAAACAQDI5IHBq3DUh97aWzSAlBFov5FaNtgut6oW9QvZ7NRFplhskKqze57xcWOkL0y22n1Rao
↪PB5blAUB8DJEJXz19py3pVb3BML2PBHYN+p/
↪wqCNoKu2n22grmYphVnY5rjjgW4K4Y8AkBa8vv4YzyFvXrPRp4GD3THelkM7YsgnlZsU/
↪QHw7rxOtWRTpeM4U4ZVdLmHWG45L1x/FjFvnsLSMGipRIaY0Oy/
↪bC+29MCGRFZYrojSmy8KOPxjEBmwyrRe8ooDRtMtkMLQDCD8baidLnIv2yM+BdZXHBXh22f4YHJMakoPwC3n57o5x/
↪NUAjn+0DRdgmjhimQFdh6Untd1kAnfQOd6kl/
↪IMKyiJBZd8HacVF5XdIn9kc6l98d05uqZM71noHusHlvLKv0dtbCCM7myNmIPotT4albEJGAv22+siN8awLxepOYOBFL5sOsEW
↪KmACaH7mKiBLtq0wwW8xxoibLWFCX8C3VIFS3GOWrQU/
↪Q49XSC3RNqqc7VgW1xZjYYb3BELT5tPMJbH3hKilfrg5NqWPj/
↪2TDFDS6Za1QhIPbyMSxvfkf8usAqLVRW7nF4Q+UcazVg2nQ== jb.morgado@gmail.com bruno
```

Warning: This must be done locally, only the System Team should update the distribute-ssh-keys *repo on GitLab.*

3. Distribute the new keys

Now the only thing left is to distribute the keys among the nodes you added to the *inventory_ssh_keys* file. In our case, since we defined the `k8s_syscore:children` structure with reference to our new machines, we can just issue:

```
make add NODES=k8s_syscore
```

You now should be able to login in any of the new nodes by doing (we will use the loadbalancer ip address on our example 192.168.93.137):

```
ssh 192.168.93.137
```

1.1.7 Destroying the nodes

Caution: The following command deletes everything that was installed and destroys the all cluster specified by the *PRIVATE_VARS* variable.

In case you want to delete the all stack issue the `make clean` command.

1.1.8 Help

Run `make` to get the help:

```
make targets:
Makefile:build                 Build nodes, haproxy and k8s
Makefile:build_charts          Build charts
Makefile:build_common          apply the common roles
Makefile:build_docker          apply the docker roles
Makefile:build_haproxy         Build haproxy
Makefile:build_k8s             Build k8s
Makefile:build_nodes           Build nodes based on heat-cluster
Makefile:check_nodes           Check nodes based on heat-cluster
Makefile:clean_k8s             clean k8s cluster
Makefile:clean_nodes           destroy the nodes - CAUTION THIS DELETES EVERYTHING!!!
Makefile:help                  show this help.
Makefile:install               Install dependent ansible collections
Makefile:lint                  Lint check playbook
Makefile:reinstall             reinstall collections
Makefile:test                  Smoke test for new created cluster
Makefile:vars                  Variables

make vars (+defaults):
Makefile:ANSIBLE_USER          centos## ansible user for the playbooks
Makefile:CLUSTER_KEYPAIR       piers-engage## key pair available on openstack to be␣
↪put in the created VMs
Makefile:COLLECTIONS_PATHS     ./collections
Makefile:CONTAINERD            true
Makefile:DEBUG                 false
Makefile:EXTRA_VARS ?=
Makefile:IGNORE_NVIDIA_FAIL    false
Makefile:INVENTORY_FILE        ./inventory_k8s##inventory file to be generated
Makefile:NVIDIA                false
Makefile:PRIVATE_VARS          ./centos_vars.yml##template variable for heat-cluster
Makefile:TAGS ?=
```

Todo:

- Add X-ray reports

CHAPTER

TWO

BDD TESTING OF CLUSTER-K8S

The SKA follows a BDD pattern wherever it is applicable. This kubernetes cluster provisioning repository includes a few Gherkin tests that enable an interface between python code written by developers (using pytest) and natural language scenarios that can be written by Business Owners, Product Owners, Testers and the like, using pytest-bdd.

2.1 Testing-harness

The following files and directories are relevant at the time of writing (this may not be kept up to date):

```
1   ../test-harness/
2   ├── Makefile
3   ├── README.md
4   ├── cucumber.json
5   ├── features
6   │   ├── auth.feature
7   │   ├── ceph.feature
8   │   ├── cluster.feature
9   │   ├── dns.feature
10  │   ├── etcd.feature
11  │   ├── networking.feature
12  │   ├── resources.feature
13  │   ├── sandbox.feature
14  │   └── skampi.feature
15  ├── resources
16  ...
17  │   └── skampi_values.json
18  ├── setup.cfg
19  ├── test-requirements.txt
20  └── tests
21      ├── conftest.py
22      ├── test_auth.py
23      ├── test_ceph.py
24      ├── test_cluster.py
25      ├── test_dns.py
26      ├── test_etcd.py
27      ├── test_networking.py
28      ├── test_resources.py
29      └── test_skampi.py
```

Refer to the line numbers. Each feature, with name `<feature_name>`, has a corresponding `<feature_name>.feature` file under the `features/` directory, and a `test_<feature_name>.py` under the `/tests/` directory.

At the bottom of the python file where the test functions are developed, we include a line that links the feature file to the test file. As an example, the skampi feature is described here:

The feature is written in Gherkin language:

```
<features/skampi.feature>

Feature: SKAMPI
    Install skampi and smoke test it

Scenario: Install skampi and smoke test it
    Given a Kubernetes cluster with KUBECONFIG .kube/config
    When I install the chart at stable/etcd-operator with name etcd0 with values file␣
↪resources/etcd_values.json
    And I install the chart at https://gitlab.com/ska-telescope/skampi/-/raw/master/
↪repository/skampi-0.1.0.tgz with name test2 with values file resources/skampi_
↪values.json
    Then after 10 minutes all pods are running
```

We will not include the full file here, but under the `/tests/` directory there is a corresponding python file named `test_skampi.py`. The `test_` prefix ensures that pytest picks it up as part of the test suite.

The last line in this file is of importance:

```
scenarios('../features/skampi.feature')
```

This pulls the scenarios from the .feature file and links each test function to a corresponding line in the Scenario description.

Additional and helper code is added to the `resources/` directory.