opstools-ansible Documentation

Release 0.1.0

centos-opstools

Jun 07, 2019

Contents:

CHAPTER 1

State of this project

This was intended to install a demo deployment. It was not intended for production. This is neither supported nor being actively developed anymore.

1.1 Overview

The opstools-ansible project is a collection of Ansible roles and playbooks that will configure an environment that provides centralized logging and analysis, availability monitoring, and performance monitoring.

1.2 Using opstools-ansible

1.2.1 Before you begin

Before using the opstools-ansible playbook, you will need to deploy one or more servers on which you will install the opstools services. These servers will need to be running either CentOS 7 or RHEL 7 (or a compatible distribution).

These playbooks will install packages from a number of third-party repositories. The opstools-ansible developers are unable to address problems with the third party packaging (other than via working around problems in our playbooks).

1.2.2 Creating an inventory file

Create an Ansible inventory file `inventory/hosts` that defines your hosts and maps them to host groups declared in `inventory/structure`. For example:

```
server1 ansible_host=192.0.2.7 ansible_user=centos ansible_become=true
server2 ansible_host=192.0.2.15 ansible_user=centos ansible_become=true

[am_hosts]
server1
```

(continues on next page)

(continued from previous page)

```
[logging_hosts]
server2
```

There are two high-level groups that can be used to control service placement:

- `am_hosts`: The playbooks will install availability monitoring software (Sensu, Uchiwa, and support services) on these hosts.
- `logging_hosts`: The playbooks will install the centralized logging stack (Elasticsearch, Kibana, Fluentd) on these hosts.

While there are more groups defined in the `inventory/structure` file, use of those for more granular service placement is neither tested nor supported at this time.

You can also run post-install playbook after overcloud deployment to finish server side configuration dependent on the information about the overcloud. For that you will need to add undercloud host to the inventory. So for example, after deploying overcloud via tripleo-quickstart tool, you should add something like following to the inventory file before running the playbook:

undercloud_host ansible_user=stack ansible_host=undercloud ansible_ssh_extra_args='-F "/root/.quickstart/ssh.config.ansible"'

1.2.3 Create a configuration file

Put any necessary configuration into an Ansible variables file (which is just a YAML document). For example, if you wanted to enable logging via SSL, you would need a configuration file that looked like this:

```
fluentd_use_ssl: true
fluentd_shared_key: secret
fluentd_ca_cert: |
  -----BEGIN CERTIFICATE-----
  ...
  -----END CERTIFICATE-----
fluentd_private_key: |
  -----BEGIN RSA PRIVATE KEY-----
  ...
  -----END RSA PRIVATE KEY-----
```

You don't need a configuration file if you wish to use default options.

1.2.4 Run the playbook

Once you have your inventory and configuration in place, you can run the playbook like this:

```
ansible-playbook playbook.yml -e @config.yml
```

1.3 Roles

The following documentation is automatically extracted from the `roles` directory in this distribution.

1.4 Integration with TripleO

The TripleO installer for OpenStack includes support for Fluentd and Sensu clients. See *Integrating opstools-ansible with a TripleO deployment*.

1.5 Contributing

If you encounter problems with or have suggestions about opstools-ansible, open an issue on our Github issue tracker.

If you would like to contribute code, documentation, or other changes to the project, please read the *Contributing to opstools-ansible*.

1.6 License

1.7 Contributing to opstools-ansible

You can contribute bug fixes or enhancements to the opstools-ansible project by submitting a patch review via git review following the same process as described in submitting a patch. The required .gitreview file is already contained in the repository.

1.7.1 Conventions

When creating or modifying Ansible roles, please use YAML format for module arguments as in:

```
- debug:
    var: somevar
```

Instead of the legacy format:

```
- debug: var=somevar
```

We extract documentation automatically out of `defaults/main.yml` in each role. In order for this to work correctly, each variable in that file should be preceded by a comment with no intervening whitespace, and should be separated from other variables by one (or more) blank linkes, like this:

```
# This is some documentation.
some_variable: some value

# This is documentation for another_variable.
```

(continues on next page)

(continued from previous page)

```
another_variable:
  - this
  - is
  - a
  - test
```

1.7.2 Running tests

To perform some simple YAML validations, run:

```
tox
```

You should do this *before* submitting pull requests. Arranging to run `tox` via your local Git `pre-commit` hook will save you the inconvenience of submitting a pull request only to have it rejected by our CI testing.

1.7.3 Updating the documentation

The role documentation in `README.rst` is generated automatically from (a) comments in the `defaults/main.yml` file for each role and (b) a `README.rst` included in each role.

After making changes, you can regenerate the documentation by running `python setup.py build_sphinx` in the top level of the repository.

1.8 Integrating opstools-ansible with a TripleO deployment

1.8.1 Before you begin

This guide assumes that you are working with the **OpenStack Newton** release.

For the purpose of creating some concrete examples, this document assumes that you have deployed your opstools-ansible environment using the following inventory:

```
ops0 ansible_host=192.168.10.10
ops1 ansible_host=192.168.10.20

[am_hosts]
ops0

[logging_hosts]
ops1
```

And the following configuration:

```
fluentd_use_ssl: true
fluentd_shared_key: secret
fluentd_ca_cert: |
  -----BEGIN CERTIFICATE-----
  ...
  -----END CERTIFICATE-----
fluentd_private_key: |
  -----BEGIN RSA PRIVATE KEY-----
```

(continues on next page)

(continued from previous page)

```
  ...
  -----END RSA PRIVATE KEY-----
```

Configure TripleO

Once your opstools environment is running, create a configuration file for your TripleO environment that will point the Sensu and Fluentd agents at the opstools hosts. Look at the logging and monitoring environment files for a list of available parameters.

Given the example configuration presented earlier in this document, you might end up with something like this (in a file we'll call `params.yaml`):

```
parameter_defaults:

  LoggingServers:
    - host: 192.168.10.20
      port: 24284

  LoggingUsesSSL: true

  # This must match the fluentd_shared_key key setting you
  # used in your Ansible configuration.
  LoggingSharedKey: secret

  # This must match the certificate you used for the
  # fluentd_ca_cert setting in your Ansible configuration.
  LoggingSSLCertificate: |
    -----BEGIN CERTIFICATE-----
    -----END CERTIFICATE-----

  MonitoringRabbitHost: 192.168.10.10
  MonitoringRabbitUsername: sensu
  MonitoringRabbitPassword: sensu
```

Run overcloud deploy command

Deploy your TripleO environment. In addition to whatever command line options you normally use, you will need to include the `monitoring-environment.yaml` file (if you are configuring availability monitoring), and `logging-environment.yaml` file (if you arae configuring logging), and the `params.yaml` file described in the previous step. Your `overcloud deploy` command line should look something like:

```
openackstack overcloud deploy ... \
  -e /usr/share/openstack-tripleo-heat-templates/environments/monitoring-environment.
↪yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/logging-environment.
↪yaml \
  -e params.yaml
```

When the deployment completes, you should see logs appearing in Kibana on your opstools server (`https://192.168.10.20/kibana`) and you should see the results of Sensu checks in Uchiwa (`https://192.168.10.10/uchiwa`).

CHAPTER 2

Indices and tables

- genindex
- modindex
- search