Khaleesi Documentation

Release 0.1

contributors

February 04, 2016

Contents:

CHAPTER 1

Using Khaleesi

Khaleesi is an ansible based deployment tool Red Hat Openstack CI is using for automation. In order to work, khaleesi need a configuration file which is provided by khaleesi-settings project. Khaleesi-settings provide the config file using ksgen tool, located in khaleesi project.

> https://github.com/redhat-openstack/khaleesi-settings or http://<redhat-internal-git-server>/git/khaleesi-settings.git

1.1 Prerequisites

Fedora21+ with Python 2.7. For running jobs, khaleesi requires a dedicated RHEL7 or F21 Jenkins slave. We do have an ansible playbook that sets up a slave, see *Creating a Jenkins slave*.

> **Warning:** Do not use the root user, as these instructions assumes that you are a normal user and uses venv. Being root may shadow some of the errors you may make like forgetting to source venv and pip install ansible.

Update your system, install git and reboot:

```
sudo yum -y update && sudo yum -y install git && sudo reboot
```

Install the 'Development Tools' Package group, python-devel and sshpass packages:

```
sudo yum group install -y 'Development Tools'
sudo yum -y install python-devel python-virtualenv sshpass
```

Install the OpenStack clients:

```
sudo yum install python-novaclient python-neutronclient python-glanceclient -y
```

1.2 Installation

Create or enter a folder where you want to check out the repos. We assume that both repo and your virtual environment are in the same directory. Clone the repos:

```
git clone https://github.com/redhat-openstack/khaleesi.git
or
git clone https://github.com/redhat-openstack/khaleesi-settings.git
```

read-only mirror:

```
git clone http://<redhat-internal-git-server>/git/khaleesi-settings.git
```

Gerrit:

```
https://review.gerrithub.io/#/q/project:redhat-openstack/khaleesi
```

Create the virtual envionment, install ansible, ksgen and kcli utils:

```
virtualenv venv
source venv/bin/activate
pip install ansible==1.9.2
cd khaleesi
cd tools/ksgen
python setup.py develop
cd ../kcli
python setup.py develop
cd ../..
```

Create the appropriate ansible.cfg for khaleesi:

```
cp ansible.cfg.example ansible.cfg
```

If you don't have a key you need to create it and upload it to your remote host or your tenant in blue if you are using the Openstack provisoner.

Copy your private key file that you will use to access instances to `khaleesi/`. We're going to use the common `example.key.pem` key.:

```
cp ../khaleesi-settings/settings/provisioner/openstack/site/qeos/tenant/keys/example.key.pem  <dir>/k
chmod 600 example.key.pem
```

1.3 Overview

By using Khaleesi you will need to choose which installer you want to use, on which provisioner.The provisioners corresponding to the remote machines which will host your environment. Khaleesi provide two installers: rdo-manager and packstack, and four provisioners: beaker, centosci, openstack and manual. For all of those, the settings are provided by khaleesi-settings through ksgen tool. You will find configuration variable under the folder "settings":

settings:

```
|-- provisioner
|   |-- beaker
|   |-- libvirt
|   |-- openstack
|   `-- rackspace
|-- installer
|   |-- foreman
|   |-- opm
|   |-- packstack
|   |-- rdo_manager
|   `-- staypuft
|-- tester
|   |-- integration
|   |-- pep8
|   |-- rally
|   |-- rhosqe
|   |-- tempest
```

```
|   `-- unittest
|-- product
|   |-- rdo
|   `-- rhos
|-- distro
```

The whole idea of the configuration repo is to break everything into small units. Let's use the installer folder as an example to describe how the configuration tree is built. When using ksgen with the following flags:

```
--installer=packstack \
--installer-topology=multi-node \
--installer-network=neutron \
--installer-network-variant=ml2-vxlan \
--installer-messaging=rabbitmq \
```

When the given –installer=packstack, ksgen is going to the folder called "installer" in khaleesi-settings and looking for a "packstack.yml" file.

after that, it goes down the tree to the folder "packstack/topology/multi-node.yml" (because of the flag –installer-topology=multi-node), "packstack/network/neutron.yml", etc (according to the additional flags) and list all yml files it finds under those folders.

Then ksgen starts merging all YAML files using the parent folders as a base, that means, that packstack.yml (which holds configuration that is common to packstack) will be used as base and be merged with "packstack/topology/multi-node.yml" and "packstack/network/neutron.yml" and so on.

1.4 Usage

After you have everything set up, let's see how you can create machines using rdo-manager or packstack installer. In both cases we're going to use ksgen (Khaleesi Settings Generator) for supplying Khaleesi's ansible playbooks with a correct configuration.

1.5 Installing rdo-manager with the manual provisioner

Here, we will deploy a RDO-Manager environment using the manual environment.

First, we create the appropriate configuration file with ksgen. Make sure that you are in your virtual environment that you previously created.

```
source venv/bin/activate
```

Export the ip or fqdn hostname of the test box you will use as the virtual host for osp-director:

```
export TEST_MACHINE=<ip address of baremetal virt host>
```

Generate the configuration with the following command:

```
ksgen --config-dir=../khaleesi-settings/settings generate \
    --provisioner=manual \
    --product=rdo \
    --product-version=liberty \
    --product-version-build=last_known_good \
    --product-version-repo=delorean_mgt \
    --distro=centos-7.0 \
    --installer=rdo_manager \
```

```
    --installer-env=virthost \
    --installer-images=build \
    --installer-network=neutron \
    --installer-network-isolation=none \
    --installer-network-variant=ml2-vxlan \
    --installer-topology=minimal \
    --installer-deploy=templates \
    --installer-post_action=none \
    --installer-tempest=disabled \
    --workarounds=enabled \
    --extra-vars @../khaleesi-settings/hardware_environments/virt_default/hw_settings.yml \
    ksgen_settings.yml
```

Note: The "base_dir" key is defined by either where you execute ksgen from or by the $WORKSPACE environment variable. The base_dir value should point to the directory where khaleesi and khaleesi-settings have been cloned.

The result is a YAML file collated from all the small YAML snippets from `khaleesi-settings/settings`. All the options are quite self-explanatory and changing them is simple as well. The rule file is currently only used for deciding the installer+product+topology configuration. Check out ksgen for detailed documentation.

The next step will run your intended deployment:

```
ansible-playbook -vv --extra-vars @ksgen_settings.yml -i local_hosts playbooks/full-job-no-test.yml
```

If any part fails, you can ask for help on freenode #rdo channel. Don't forget to save the relevant error lines on something like pastebin.

1.5.1 Using your new undercloud / overcloud

When your run is complete (or even while it's running), you can log in to your test machine:

```
ssh root@<test_machine>
su stack
```

If you want to log to your new undercloud machine

```
ssh -F ssh.config.ansible undercloud
```

Here you could play with your newly created Overcloud

1.6 Installing rdo-manager with centosci provisioner

Here the installation is quite similar with Beaker provisioner. Just notice the changes into the configuration for ksgen:

```
ksgen --config-dir=../khaleesi-settings/settings generate \
--provisioner=centosci \
--provisioner-site=default \
--provisioner-distro=centos \
--provisioner-distro-version=7 \
--provisioner-site-user=rdo \
--product=rdo \
--product-version=kilo \
--product-version-build=last_known_good \
--product-version-repo=delorean_mgt \
```

```
--distro=centos-7.0 \
--installer=rdo_manager \
--installer-env=virthost \
--installer-images=build \
--installer-network=neutron \
--installer-network-isolation=none \
--installer-network-variant=ml2-vxlan \
--installer-topology=minimal \
--installer-post_action=none \
--installer-tempest=disabled \
--installer-deploy=templates \
--workarounds=enabled \
--extra-vars @../khaleesi-settings/hardware_environments/virt_default/hw_settings.yml \
ksgen_settings.yml
```

If any part fails, you can ask for help on the internal #rdo-ci channel. Don't forget to save the relevant error lines on something like pastebin.

1.6.1 Using your new undercloud / overcloud

When your run is complete (or even while it's running), you can log in to your host

```
ssh root@$HOST
su stack
```

If you want to log to your new undercloud machine, just make on your host:

```
ssh -F ssh.config.ansible undercloud
```

Here you could play with your newly created Overcloud

1.7 Installing Openstack on Bare Metal via Packstack

All the steps are the same as the All-in-one case. The only difference is running the ksgen with differents paramters: Please change the below settings to match your environment:

```
ksgen --config-dir=/khaleesi_project/khaleesi-settings/settings generate \
--provisioner=foreman \
--provisioner-topology="all-in-one" \
--distro=rhel-7.1 \
--product=rhos \
--product-version=7.0 \
--product-version-repo=puddle \
--product-version-build=latest \
--extra-vars=provisioner.nodes.controller.hostname=puma06.scl.lab.tlv.redhat.com \
--extra-vars=provisioner.nodes.controller.network.interfaces.external.label=enp4s0f1 \
--extra-vars=provisioner.nodes.controller.network.interfaces.external.config_params.device=enp4s0f1 \
--extra-vars=provisioner.nodes.controller.network.interfaces.data.label="" \
--extra-vars=provisioner.nodes.controller.network.interfaces.data.config_params.device="" \
--extra-vars=provisioner.network.network_list.external.allocation_start=10.35.175.1 \
--extra-vars=provisioner.network.network_list.external.allocation_end=10.35.175.100 \
--extra-vars=provisioner.network.network_list.external.subnet_gateway=10.35.175.101 \
--extra-vars=provisioner.network.network_list.external.subnet_cidr=10.35.175.0/24 \
--extra-vars=provisioner.network.vlan.external.tag=190 \
--extra-vars=provisioner.remote_password=mypassword \
```

```
--extra-vars=provisioner.nodes.controller.rebuild=yes \
--extra-vars=provisioner.key_file=/home/itbrown/.ssh/id_rsa \
--installer=packstack \
--installer-network=neutron \
--installer-network-variant=ml2-vxlan \
--installer-messaging=rabbitmq \
ksgen_settings.yml
```

And then simply run:

```
ansible-playbook -vv --extra-vars @ksgen_settings.yml -i local_hosts playbooks/full-job-no-test.yml
```

1.8 Cleanup

After you finished your work, you can simply remove the created instances by:

```
ansible-playbook -vv --extra-vars @ksgen_settings.yml -i hosts playbooks/cleanup.yml
```

CHAPTER 2

Community Guidelines:

2.1 Blueprints:

What is a blueprint?

1. Any new feature requires a blueprint[1].
2. A new feature is anything that changes API / structure of current code and requires a change that spans for more than one file.

   [1] - https://wiki.openstack.org/wiki/Blueprints#Blueprints_reference

Where should one submit blueprints?

1. Any new blueprint requires a discussion in the ML / weekly sync. (we encourage everyone who is involved with the project to join)
2. A code sample / review of a blueprint should use the *git-review -D* for publishing a draft on gerrit.
3. Since the review process is being done as a draft, it is possible to submit the draft prior to an actual ML e-mail.

2.2 Reviews:

1. https://review.gerrithub.io/Documentation/config-labels.html#label_Code-Review
2. A "-1"/"-2" from a core requires special attention and a patch should not be merged prior to having the same core remove the "-1"/"-2".
3. In case of a disagreement between two cores, the matter will be brought into discussion on the weekly sync / ML where each core will present his / her thoughts.
4. Self reviews are not allowed. You are required to have at least one more person +1 your code.
5. No review should be merged prior to all gates pass.
6. Bit Rot. To keep the review queue clean an auto-abandoning of dead or old reviews is implemented. A dead review is defined by there are no comments, votes, activity for some agreed upon length of time. Warning will be posted on the review for two weeks of no activity, after the third week the review will be abandoned.

2.3 Gates:

1. If a gate has failed, we should first fix that gate and rerun the job to get it passing.

2. When a gate fails due to an infrastructure problem (example: server timeout, failed cleanup, etc), two cores approval is required in order to remove a gate "-1" vote

2.4 Commits:

1. Each commit should be dedicated to a specific subject and not include several patches that are not related.
2. Each commit should have a detailed commit message that describes the "high level" of what this commit does and have reference to other commits in case there is a relationship.

2.5 Cores:

1. Need to have quality reviews.
2. Reviews are well formed, descriptive and constructive.
3. Reviews are well thought and do not result in a followed revert. (often)
4. Should be involved in the project on a daily basis.

CHAPTER 3

Contributing to Khaleesi development

3.1 Getting Started with Khaleesi.

see *Prerequisites*

3.2 Associated Settings Repository

https://github.com/redhat-openstack/khaleesi-settings

3.3 Help, I can't run this thing

Look under the khaleesi/tools/wrappers directory

3.4 Code Review (IMPORTANT)

Pull requests will not be looked at on khaleesi github. Code submissions should be done via gerrithub (https://review.gerrithub.io). Please sign up with https://www.gerrithub.io and your github credentials to make submissions. Additional permissions on the project will need to be done on a per-user basis.

When you set up your account on gerrithub.io, it is not necessary to import your existing khaleesi fork.:

yum install git-review

To set up your repo for gerrit:

Add a new remote to your working tree:

```
git remote add gerrit ssh://username@review.gerrithub.io:29418/redhat-openstack/khaleesi
```

Replace username with your gerrithub username.

Now run:

```
git review -s
scp -p -P 29418 username@review.gerrithub.io:hooks/commit-msg `git rev-parse --git-dir`/hooks/commit-
```

Again, replace username with your gerrithub username.

3.5 Required Ansible version

Ansible 1.8.2 is now required.

3.6 Std{out,err} callback plugin

To use the callback plugin that will log all stdout, stderr, and other data about most tasks, you must set the ANSIBLE_CALLBACK_PLUGINS envvar. You can also set the KHALEESI_LOG_PATH envvar. KHALEESI_LOG_PATH defaults to /tmp/stdstream_logs.:

export ANSIBLE_CALLBACK_PLUGINS=$WORKSPACE/khaleesi/plugins/callbacks export KHALEESI_LOG_PATH=$WORKSPACE/ansible_log

3.7 Khaleesi use cases

Check khaleesi *Usage*

CHAPTER 4

ksgen - Khaleesi Settings Generator

4.1 Setup

It's advised to use ksgen in a virtual Python environment.

```
$ virtualenv ansible # you skip this and use an existing one
$ source ansible/bin/activate
$ python setup.py develop # do this in the ksgen directory
```

4.2 Running ksgen

Assumes that ksgen is installed, else follow *Setup*.

You can get general usage information with the `--help` option. After you built a proper settings directory ("configuration tree") structure, you need to let ksgen know where it is. Invoke ksgen like this to show you all your possible options:

```
ksgen --config-dir <dir> help
```

If `--config-dir` is not provided, ksgen will look for the `KHALEESI_DIR` environment variable, so it is a good practice to define this in your `.bashrc` file:

```
export KHALEESI_DIR=<dir>
ksgen help
```

This displays options you can pass to ksgen to *generate* the all-in-one settings file.

4.3 Using ksgen

4.3.1 How ksgen works

ksgen is a simple utility to **merge** dictionaries (hashes, mappings), and lists (sequences, arrays). Any scalar value (string, int, floats) are overwritten while merging.

For e.g.: merging `first_file.yml` and `second_file.yml`

first_file.yml:

```
foo:
  bar: baz
  merge_scalar: a string from first dict
  merge_list: [1, 3, 5]
  nested:
    bar: baz
    merge_scalar: a string from first dict
    merge_list: [1, 3, 5]
```

and second_file:

```
foo:
  too: moo
  merge_scalar: a string from second dict
  merge_list: [6, 2, 4, 3]
  nested:
    bar: baz
    merge_scalar: a string from second dict
    merge_list: [6, 2, 4, 3]
```

produces the output below:

```
foo:
  bar: baz
  too: moo
  merge_scalar: a string from second dict
  merge_list: [1, 3, 5, 6, 2, 4, 3]
  nested:
    bar: baz
    merge_scalar: a string from second dict
    merge_list: [1, 3, 5, 6, 2, 4, 3]
```

4.3.2 Organizing settings files

ksgen requires a `--config-dir` option which points to the directory where the settings files are stored. ksgen traverses the *config-dir* to generate a list of options that sub-commands (`help`, `generate`) can accept.

First level directories inside the config-dir are used as options, and yml files inside them are used as option values. You can add suboptions if you add a directory with the same name as the value (without the extension). Inside that directory, the pattern repeats: you can specify options by creating directories, and inside them yml files.

If the directory name and a yml file don't match, it doesn't add a suboption (it gets ignored). You can use this to store YAMLs for includes.

Look at the following schematic example:

```
settings/
-- option1/
|   -- ignored/
|   |   -- useful.yml
|   -- value1.yml
|   -- value2.yml
-- option2/
    -- value3/
    |   -- suboption1/
    |       -- value5.yml
    |       -- value6.yml
```

```
    -- value3.yml
    -- value4.yml
```

The valid settings will be:

```
$ ksgen --config-dir settings/ help
[snip]
 Valid configs are:
    --option1=<val>            [value2, value1]
    --option2=<val>            [value4, value3]
    --option2-suboption1=<val> [value6, value5]
```

A more organic settings example:

```
settings/
-- installer/
|   -- foreman/
|   |   -- network/
|   |       -- neutron.yml
|   |       -- nova.yml
|   -- foreman.yml
|   -- packstack/
|   |   -- network/
|   |       -- neutron.yml
|   |       -- nova.yml
|   -- packstack.yml
-- provisioner/
    -- trystack/
    |   -- tenant/
    |   |   -- common/
    |   |   |   -- images.yml
    |   |   -- john-doe.yml
    |   |   -- john.yml
    |   |   -- smith.yml
    |   -- user/
    |       -- john.yml
    |       -- smith.yml
    -- trystack.yml
```

ksgen maps all directories to options and files in those directories to values that the option can accept. Given the above directory structure, the options that `generate` can accept are as follows

Options	Values
provisioner	trystack
provisioner-tenant	smith, john, john-doe
provisioner-user	john, smith
installer	packstack, foreman
installer-network	nova, neutron

Note: ksgen skips provisioner/trystack/tenant/common directory since there is no `common.yml` file under the `tenant` directory.

4.3.3 Default settings

Default settings allow the user to supply only the minimal required flags in order to generate a valid output file. Defaults settings will be loaded from the given 'top-level' parameters settings files if they are defined in them. Defaults settings

for any 'non top level' parameters that have been given will not been loaded.

Example of defaults section in settings files:: provisioner/openstack.yml: defaults:

site: openstack-site topology: all-in-one

provisioner/openstack/site/openstack-site.yml: defaults:

user: openstack-user

Usage example:: ksgen –config-dir=/settings/dir/path generate –provisioner=openstack settings.yml

4.3.4 generate: merges settings into a single file

The `generate` command merges multiple settings file into a single file. This file can then be passed to an ansible playbook. ksgen also allows merging, extending, overwriting (!overwrite_) and looking up (!lookup_) settings that ansible (at present) doesn't allow.

Merge order

Refering back to the *settings example* above, if you execute the command:

```
ksgen --config-dir sample generate \
  --provisioner trystack \
  --installer packstack \
  --provisioner-user john \
  --extra-vars foo.bar=baz \
  --provisioner-tenant smith \
  output-file.yml
```

generate command will create an `output-file.yml` that include all contents of

SL	File	Reason
1	provisioner/trystack.yml	The first command line option
2	merge provisioner/trystack/user/john.yml	The first child of the first command line option
3	merge provisioner/trystack/tenant/smith.yml	The next child of the first command line option
4	merge installer/packstack.yml	the next top-level option
5	add/merge foo.bar: baz. to output	extra-vars get processed at the end

Rules file

ksgen arguments can get quite long and tedious to maintain, the options passed to ksgen can be stored in a rules yaml file to simplify invocation. The command above can be simplified by storing the options in a yaml file.

rules_file.yml:

```
args:
  provisioner: trystack
  provisioner-user: john
  provisioner-tenant: smith
  installer: packstack
  extra-vars:
    - foo.bar=baz
```

ksgen generate using rules_file.yml:

```
ksgen --config-dir sample generate \
  --rules-file rules_file.yml \
  output-file.yml
```

Apart from the **args** key in the rules-files to supply default args to generate, validations can also be added by adding a 'validation.must_have' like below:

```
args:
  ...
    default args
  ...
validation:
  must_have:
      - topology
```

The generate commmand would validate that all options in must_have are supplied else it will fail with an appropriate message.

4.4 YAML tags

ksgen uses Configure python package to keep the yaml files DRY. It also adds a few yaml tags like !overwrite, !lookup, !join, !env to the collection.

4.4.1 overwrite

Use *overwrite* tag to overwrite value of a key. This is especially useful when to clear the contents of an array and add new one

For e.g.: merging

```
foo: bar
```

and

```
foo: [1, 2, 3]
```

will fail since there is no reasonable way to merge a string and an array. Use overwrite to set the contents of foo to [1, 2, 3] as below

```
foo: !overwrite [1, 2, 3]
```

4.4.2 lookup

Lookup helps keep the yaml files DRY by replacing looking up values for keys.

```
foo: bar
key_foo: !lookup foo
```

After ksgen process the yaml above the value of *key_foo* will be replaced by *bar* resulting in the output below.

```
foo: bar
key_foo: bar
```

This works for several consecutive !lookup as well such as

```
foo:
    barfoo: foobar
bar:
    foo: barfoo

key_foo: !lookup foo[ !lookup bar.foo ]
```

After ksgen process the yaml above the value of *key_foo* will be replaced by *foobar*

Warning: (Limitation) Lookup is done only after all yaml files are loaded and the values are merged so that the entire yaml tree can be searched. This prevents combining other yaml tags with *lookup* as most tags are processed when yaml is loaded and not when it is written. For example:
`home: /home/john` `bashrc: !join [ !lookup home, /bashrc ]`
This **will fail** to set bashrc to */home/john/bashrc* where as the snippet below will work as expected:
`bashrc: !join [ !env HOME, /bashrc ]`

4.4.3 join

Use join tag to join all items in an array into a string. This is quite useful when using yaml anchors or *env* tag.

```
unused:
  baseurl: &baseurl http://foobar.com/repo/

repo:
  epel7: !join[ *baseurl, epel7 ]

bashrc: !join [ !env HOME, /bashrc ]
```

4.4.4 env

Use env tag to lookup value of an environment variable. An optional default value can be passed to the tag. if no default values are passed and the lookup fails, then a runtime KeyError is generated. Second optional argument will reduce length of value by given value

```
user_home: !env HOME
user_shell !env [SHELL, zsh]  # default shell is zsh
job_name_parts:
   - !env [JOB_NAME, 'dev-job']
   - !env [BUILD_NUMBER, None ]
   - !env [USER, None, 5]

job_name: "{{ job_name_parts | reject(none) | join('-') }}"
```

The snippet above effectively uses *env* tag and default option to set the *job_name* variable to *JOB_NAME-BUILD_NUMBER-${USER:0:5}* if they are defined else to 'dev-job'.

4.4.5 limit_chars

This function will trim value of variable or string to given length.

debug: message: !limit_chars ['some really looong text' 10]

4.5 Debugging errors in settings

ksgen is heavily logged and by default the log-level is set to **warning**. Changing the debug level using the `--log-level` option to **info** or **debug** reveals more information about the inner workings of the tool and how values are loaded from files and merged.

4.6 Developing ksgen

4.6.1 Running ksgen unit-tests

```
pip install pytest
py.test tests/test_<filename>.py
# or
python tests/test_<filename>.py  <method_name>
```

CHAPTER 5

kcli - Khaleesi CLI tool

`kcli` is intended to reduce Khaleesi users' dependency on external CLI tools.

5.1 Setup

Note: Khaleesi is based on ansible so for setup to work, `kcli` requires ansible installed:

```
$ pip install ansible
```

from khaleesi directory.

```
$ cd tools/kcli
$ python setup.py install # do this in the ``kcli`` directory
```

5.2 Running kcli

Assumes that `kcli` is installed, else follow *Setup*.

You can get general usage information with the `--help` option:

```
kcli --help
```

This displays options you can pass to `kcli`.

5.3 KCLI execute

Note: This is a wrapper for the `ansible-playbook` command. In verbose mode, the equivalent anisble command will be printed.

Executes pre-configured ansible-playbooks, with given settings YAML file generated by ksgen. if no settings file is defined, will look for the default name `ksgen_settings.yml`:

```
kcli [-vvvv] [--settings SETTINGS] execute [-i INVENTORY] [--provision] [--install] [--test] [--colle
```

CHAPTER 6

Handling the Jenkins job definitions

This section deals with the issue of adding, removing and changing of job definitions through the JJB files.

A general documentation about JJB can be found on its website. When in doubt about what an option means in the job description, search in this manual.

6.1 Location and structure

The job definitions reside in `khaleesi-settings/jobs` and they are in YAML format. The changes are applied on the official Jenkins server by submitting a change to the files in this repository, and running the jenkins_job_builder job.

If you removed some job, please make sure to disable or delete the job that is no longer used. The job has a diff output at the end of the run that compares the jobs that exist on the server but are not part of the job definitions.

The `defaults.yaml` file containts the default values that all jobs get by default. It also contains some macros that can be referenced later. You probably don't need to modify this file.

At the moment the `main.yaml` file contains the definitions for all RDO CI jobs. On the top of the file you find a **job-template** that is the base of our jobs. You can see that its name contains a lot of variables in curly brackets "{}", they are replaced by the actual job definitions, and we give them values by the **project** definitions lower in the file.

The project definitions are creating a matrix of variables, from which all the possible combinations get created on the Jenkins server.

6.2 Adding new jobs

The need for a new job could arise when we want to extend our testing. There's a significant difference between two cases:

- adding a new value to an existing ksgen option (can be thought of as extending the testing matrix in an existing dimension)
- adding a new option to ksgen (adding a new dimension to the testing matrix)

The first case is significantly easier to deal with, so let's discuss that first.

Let's say you added the new variable *foo* for the **distro** setting. If you want to create a whole new set of jobs, then you might want to create a new **project** definition. In most cases, it's enough if you extend an existing definition. In that case, just add the relevant option to the proper place. Here's an example:

```
- project:
    name: rhos7-jobs
    product:
            - rhos
        product-version:
            - 7_director
        product-version-repo:
            - poodle
            - puddle
        distro:
            - rhel-7.1
            - foo
        messaging:
            - rabbitmq
[the rest of the definition is omitted]
```

If you want to extend the matrix, the changes are more numerous.

- the **job-template** has to be changed, and the new option added to the name
- the *ksgen-builder* macro needs alteration, both in the calling in the job template, and in the shell script part (add it to the ksgen command).
- add the option to all the project definitions that are using the template (currently all of them), modifying the template name in them too.

This will also result in a replacement of all the Jenkins jobs that use the template, as the naming changes.

CHAPTER 7

Creating a Jenkins server with Khaleesi jobs

7.1 Getting a Jenkins

Deploying the jobs require a properly configured Jenkins server. We have a couple of them already, but if you want to experiment without any fear of messing with other jobs, the best is to get yourself a server. It's recommended to use the Long Term Support (LTS) version.

You can create a VM on any of our OpenStack instances (don't forget to use your public key for it), attach a floating IP and then install Jenkins. This should work both on Fedora and RHEL:

```
sudo wget -O /etc/yum.repos.d/jenkins.repo \
http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo
sudo rpm --import \
http://pkg.jenkins-ci.org/redhat-stable/jenkins-ci.org.key
yum install jenkins
service jenkins start
chkconfig jenkins on
```

7.2 Installing plugins

Our jobs require quite a few plugins. So when your Jenkins is up and running, navigate to `http://$JENKINS_IP:8080/cli` and download jenkins-cli.jar.

Afterwards. just execute these commands:

```
java -jar jenkins-cli.jar -s http://$JENKINS_IP:8080/ install-plugin git \
xunit ansicolor multiple-scms rebuild ws-cleanup gerrit-trigger \
parameterized-trigger envinject email-ext sonar copyartifact timestamper \
build-timeout jobConfigHistory test-stability jenkins-multijob-plugin \
dynamicparameter swarm shiningpanda scm-api ownership mask-passwords \
jobConfigHistory buildresult-trigger test-stability dynamicparameter \
scm-api token-macro swarm scripttrigger groovy-postbuild shiningpanda \
jenkins-multijob-plugin ownership
```

7.3 Deploying the jobs

You can do this from any machine. Install JJB:

```
pip install jenkins-job-builder
```

Create a config file for it:

```
cat > my_jenkins << EOF
[jenkins]
user=my_username
password=my_password
url=http://$JENKINS_IP:8080/
EOF
```

Optional: I recommend turning off the timed runs (deleting - timed lines from the job template), otherwise they would run periodically on your test server:

```
sed '/- timed:/d' khaleesi-settings/jobs/main.yaml
```

Then just run the job creation (the last argument is the job directory of the khaleesi-settings repo, which I assume you cloned previously):

```
jenkins-jobs --conf my_jenkins update khaleesi-settings/jobs/
```

7.4 Bonus: Test your job changes

If you want to experiment with your own job changes:

- open `khaleesi-settings/jobs/defaults.yaml`
- **change the khaleesi and/or khalessi-settings repo URL to your own** and your own branch
- execute the job building step above

Now your test server will use your own version of the repos.

Tip: you can `git stash save testing` these changes, and later recall them with `git stash pop` to make this testing step easy along the code review submission.

7.5 Creating a Jenkins slave

Now you need to either set up the machine itself as a slave, or attach/create a slave to run the jobs. The slave needs to have the 'khaleesi' label on it to run the JJB jobs.

You can set up a slave with the help of the khaleesi-slave repo.

```
git clone git@github.com:redhat-openstack/khaleesi-settings.git
cd khaleesi-settings/jenkins/slaves
cat << EOF > hosts
$SLAVE_IP

[slave]
$SLAVE_IP
EOF
```

Check the settings in ansible.cfg.sample. If you run into weird ansible errors about modules you probably don't have them set up correctly. This should be enough:

```
[defaults]
host_key_checking = False
roles_path = ./roles
```

Execute the playbook, assuming that your instance uses the "fedora" user and you can access it by the "rhos-jenkins.pem" private key file. If you used a proper cloud image, it will fail.

```
ansible-playbook -i hosts -u fedora playbooks/basic_internal_slave.yml --private-key=rhos-jenkins.per
```

Login to the machine, become root and delete the characters from /root/.ssh/authorized_keys before the "ssh-rsa" word. Log out and rerun the ansible command. It should now finish successfully.

Add the slave to Jenkins. If you used the same machine, specify `localhost` and add the relevant public key for the rhos-ci user. use the `/home/rhos-ci/jenkins` directory, add the `khaleesi` label, only run tied jobs. You're done.

7.6 Jenkins RDO-Manager:

For using khaleesi with Jenkins, first of all see the steps *Getting a Jenkins* part for setting up a Jenkins slave and for use jjb.

If you want to setup a manual job on Jenkins you have to follow those steps:

7.6.1 Setup a slave (General):

Check the option:

```
Restrict where this project can be run
```

And put the name of your slave.

7.6.2 Clone the repositories (Source Code Management):

Select the choice:

```
Multiple SCMs
```

And put the urls of the khaleesi / khaleesi-settings repositories. You need to specify to jenkins to checkout the repositories in a sub-directory:

```
Check out to a sub-directory
```

And specify for each:

```
khaleesi
khaleesi-settings
```

7.6.3 Build Environment:

Check the option:

Delete workspace before build starts

7.6.4 Build:

Add a step:

```
Virtualenv Builder
```

And select:

```
Python version: System-CPython-2.7
Nature: Shell
```

And put the above informations into the shell command:

```
pip install -U ansible==1.9.2 > ansible_build; ansible --version
source khaleesi-settings/jenkins/ansible_rdo_mang_settings.sh

# install ksgen
pushd khaleesi/tools/ksgen
python setup.py develop
popd

pushd khaleesi
# generate config
ksgen --config-dir=../khaleesi-settings/settings generate \
    --provisioner=your_provisioner (see cookbook)

# get nodes and run test
set +e
anscmd="stdbuf -oL -eL ansible-playbook -vv --extra-vars @ksgen_settings.yml"

$anscmd -i local_hosts playbooks/full-job-no-test.yml
result=$?

infra_result=0
$anscmd -i hosts playbooks/collect_logs.yml &> collect_logs.txt || infra_result=1
$anscmd -i local_hosts playbooks/cleanup.yml &> cleanup.txt || infra_result=2

if [[ "$infra_result" != "0" && "$result" = "0" ]]; then
    # if the job/test was ok, but collect_logs/cleanup failed,
    # print out why the job is going to be marked as failed
    result=$infra_result
    cat collect_logs.txt
    cat cleanup.txt
fi

exit $result
```

7.6.5 Post-build actions:

Add a post build action for collecting logs and required files for debuging and archived them:

```
Archive the artifacts: **/collected_files/*.tar.gz, **/nosetests.xml, **/ksgen_settings.yml
```

If you run tempest during the deployment add the following step for collecting the tests result:

```
Publish JUnit test result report
Test Report XMLs : **/nosetests.xml
Check : Test stability history
```

CHAPTER 8

Khaleesi - Cookbook

By following these steps, you will be able to deploy rdo-manager using khaleesi on a CentOS machine with a basic configuration

8.1 Requirements

For deploying rdo-manager you will need at least a baremetal machine which must has the following minimum system requirements:

```
CentOS-7
Virtualization hardware extenstions enabled (nested KVM is not supported)
1 quad core CPU
12 GB free memory
120 GB disk space
```

Khaleesi driven RDO-Manager deployments only support the following operating systems:

```
CentOS 7 x86_64
RHEL 7.1 x86_64 ( Red Hat internal deployments only )
```

See the following documentation for system requirements:

```
http://docs.openstack.org/developer/tripleo-docs/environments/virtual.html
```

Note: There is an internal khaleesi-settings git repository that contains the settings and configuration for RHEL deployments. Do not attempt to use a RHEL bare metal host or RHEL options in ksgen using these instructions

8.2 Deploy rdo-manager

8.2.1 Installation:

Get the code :

khaleesi on Github:

```
git clone git@github.com:redhat-openstack/khaleesi.git
```

khaleesi-settings on Github:

```
git clone git@github.com:redhat-openstack/khaleesi-settings.git
```

Install tools and system packages:

```
sudo yum install -y python-virtualenv gcc
```

or on Fedora 22:

```
sudo dnf install -y python-virtualenv gcc
```

Create the virtual envionment, install ansible, ksgen and kcli utils:

```
virtualenv venv
source venv/bin/activate
pip install ansible==1.9.2
cd khaleesi/tools/ksgen
python setup.py develop
cd ../kcli
python setup.py develop
cd ../..
```

Note: If you get a errors with kcli installation make sure you have all system development tools intalled on your local machine: python2-devel for Fedora CentOS

8.2.2 Configuration:

Create the appropriate ansible.cfg for khaleesi:

```
cp ansible.cfg.example ansible.cfg
touch ssh.config.ansible
echo "" >> ansible.cfg
echo "[ssh_connection]" >> ansible.cfg
echo "ssh_args = -F ssh.config.ansible" >> ansible.cfg
```

8.2.3 SSH Keys:

Note: We assume that you will named the key : ~/id_rsa and ~/id_rsa.pub

Ensure that your ~/.ssh/id_rsa.pub file is in /root/.ssh/authorized_keys file on the baremetal virt host:

```
ssh-copy-id root@<ip address of baremetal virt host>
```

8.2.4 Deployment Configuration:

Export the ip or fqdn hostname of the test box you will use as the virtual host for osp-director:

```
export TEST_MACHINE=<ip address of baremetal virt host>
```

Create a ksgen-settings file for Khaleesi to be able to get options and settings:

```
ksgen --config-dir=../khaleesi-settings/settings generate \
    --provisioner=manual \
    --product=rdo \
    --product-version=liberty \
    --product-version-build=last_known_good \
    --product-version-repo=delorean_mgt \
    --distro=centos-7.0 \
    --installer=rdo_manager \
    --installer-deploy=templates \
    --installer-env=virthost \
    --installer-images=build \
    --installer-network=neutron \
    --installer-network-isolation=none \
    --installer-network-variant=ml2-vxlan \
    --installer-post_action=none \
    --installer-topology=minimal \
    --installer-tempest=disabled \
    --workarounds=enabled \
    --extra-vars @../khaleesi-settings/hardware_environments/virt/network_configs/none/hw_settings.yml
    ksgen_settings.yml
```

Note: The "base_dir" key is defined by either where you execute ksgen from or by the $WORKSPACE environment variable. The base_dir value should point to the directory where khaleesi and khaleesi-settings have been cloned.

If you want to have more informations about the options used by ksgen launch:

```
ksgen --config-dir=../khaleesi-settings/settings help
```

Note: This output will give you all options available in ksgen tools, You can also check into *Usage* for more examples.

Once all theses steps is done, you have a ksgen-settings file which contains all settings for your deployment. Khaleesi will load all the variables from this YAML file.

Review the ksgen_settings.yml file

8.2.5 Deployment Execution:

And then simply run:

```
ansible-playbook -vv --extra-vars @ksgen_settings.yml -i local_hosts playbooks/full-job-no-test.yml
```

8.3 Cleanup

After you finished your work, you can simply remove the created instances by:

```
ansible-playbook -vv --extra-vars @ksgen_settings.yml -i hosts playbooks/cleanup.yml
```

CHAPTER 9

Indices and tables

- genindex
- modindex
- search