Jujuna Documentation

Release 2.9.7

Matus Kosut

Aug 24, 2022

Contents

At HUNT Cloud, we run our scientific services based on OpenStack orchestrated by Juju. Such cloud deployments rely on a large set of collaborative softwares, and upgrades can sometimes cause considerable pain. We are therefore introducing Jujuna - a tool to simplify the validation of Juju-based OpenStack upgrades.

New to Juju? Juju is a cool controller and agent based tool from Canonical to easily deploy and manage applications (called Charms) on different clouds and environments (see how it works for more details).

Jujuna validates OpenStack upgrades from a specific Juju bundle to a new predefined set of charm revisions and software versions. First, Jujuna automates the deployment of a specific OpenStack Juju bundle into a testing stack. Next, it automates the upgrade process to a new set of specific software versions, including rolling upgrade of HA configurations. Then, it validates the infrastructure status during and after the deployment. Finally, it can clean up the deployment.

CHAPTER 1

Quickstart

1.1 Installation

To install Jujuna, open an interactive shell and run:

```
pip3 install jujuna
```

Note: It is **very** important to install Jujuna on the Python 3.5 (or higher), you need it to be installed at least on 3.5 because of the main features used in Jujuna and it's dependencies.

1.1.1 Using Jujuna

1.1.2 Try our examples

In the *examples* folder you can find a minimal OpenStack bundle (includes only Keystone and database) and a test suite.

Testing the bundle requires a working juju controller, in case you don't have one, you can try our vagrant configuration.

I have Juju Controller

First you deploy the Openstack bundle, with older version of keystone (Newton):

```
jujuna deploy minimal-openstack.bundle.yaml -w
```

When deploy is done, you can try upgrading Keystone to the next version (Ocata):

```
jujuna upgrade -o cloud:xenial-ocata -p -a keystone
```

After the upgrade you want to test our services with a test suite:

```
jujuna test minimal-openstack.test.yaml
```

If the tests were successful you can continue in the pipeline with upgrading to higher versions (Pike, Queens,...) or you can cleanup the model and remove all the applications:

```
jujuna clean -w
```

I dont't have Juju controller

If you don't have a working juju controller available. Deploying one locally on your device can be a choice for you when trying out *jujuna*:

```
cd examples && vagrant up
```

Connect to vagrant:

```
vagrant ssh
```

You can try to run *juju status* to make sure that the lxd controller is deployed properly.

When you are in vagrant, you can deploy our example Openstack bundle, with older version of keystone (Newton):

```
jujuna deploy /vagrant/minimal-openstack.bundle.yaml -w
```

When deploy is done, you can try upgrading Keystone to the next version (Ocata):

```
jujuna upgrade -o cloud:xenial-ocata -p -a keystone
```

After the upgrade you want to test our services with a test suite:

```
jujuna test /vagrant/minimal-openstack.test.yaml
```

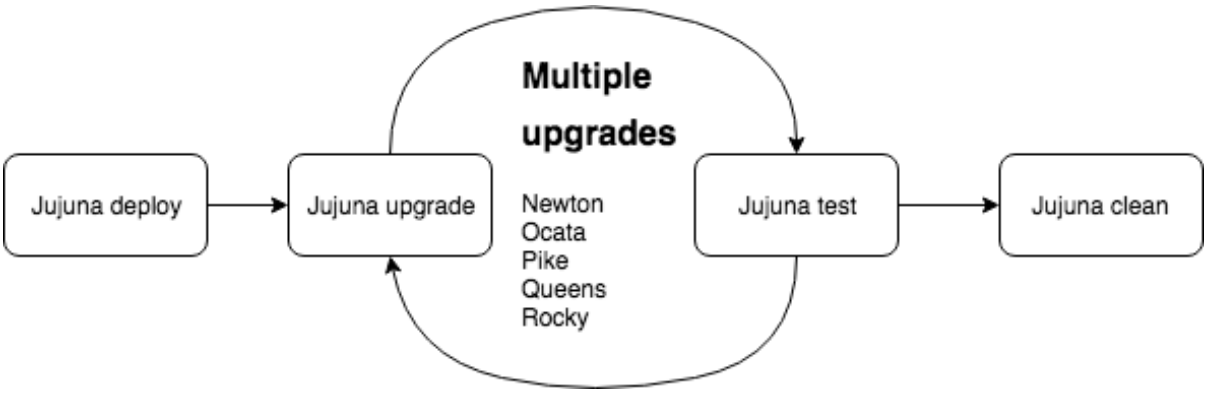
If the tests were successful you can continue in the pipeline with upgrading to higher versions (Pike, Queens,...) or you can cleanup the model and remove all the applications:

```
jujuna clean -w
```

When you are done with testing you can *exit* the vagrant.

1.1.3 Use cases

Jujuna provides four main functions that allow us to assemble various pipelines and test multiple scenarios: deploy, upgrade, test, and cleanup. These functions allow us to properly test software upgrades, from simple tests up to multistage upgrades.



We use Jujuna for three main purposes at HUNT Cloud, all to test desired deployments and service upgrades of OpenStack. We utilize a dedicated stack of test hardware, with very similar configuration to our production site. We deploy the OpenStack Juju bundle with all the applications that we have in production, although at a smaller scale.

Case 1: Continuous integration

Test of configuration changed as a part of bundle repository CI. Everytime the Juju bundle is changed it is automatically deployed and tested. All the results are pushed back to our CI. Passing result from pipeline approves the change.

Case 2: Revision upgrades

New charm revisions are released more often than the services. Release time also depends on channels that charm developers use. You can regularly run Jujuna to test new or nightly releases from edge channel of charm revisions.

Case 3: Service upgrades

Test before upgrade. Whenever there is need to upgrade production services, you can easily deploy your test stack, upgrade required services, and run your testing suite. We find both upgrade processes and testing useful to identify potential issues.

1.1.4 Writing tests

Examples and guides on how to write test suites for jujuna.

Quickstart

Format: *yaml*

Example 1 - Bundle of glance and openstack:

```
glance:
  service:
    glance-api:
      status: 'running'
    glance-registry:
      status: 'running'
  process:
    glance-api: True
```

(continues on next page)

(continued from previous page)

```
  network:
    port:
      '9292': True
mysql-db:
  service:
    mysql:
      status: 'running'
```

Module index

File module

Notation

```
file:
  'path1':
    param1: value1
    param2: value2
    param3: value3
  'path2':
    param1: value1
    param2: value2
```

Examples

File */etc/passwd* exists and is owned by root:

```
file:
  '/etc/passwd':
    st_uid: 0
    st_gid: 0
    is_reg: True
```

Parameters

Parameter	Type	Comments
st_mode		File type and mode
st_ino		
st_dev		
st_nlink		
st_uid	int	Owners uid
st_gid	int	Owners gid
st_size	int	File size
st_atime		
st_mtime		
st_ctime		
is_dir	Boolean	Is path a dir
is_chrv		
is_blk		
is_reg	Boolean	Is path a file
is_fifo	Boolean	Is path a fifo
is_lnk	Boolean	Is path a link
is_sock	Boolean	Is path a socket
imode		
ifmt		

Mount module

Notation

```
mount:
  regex:
    - 'path/sda1-[a-z0-9]+-[0-9]+'
    - 'path/sda2-[a-z0-9]+-[0-9]+'
    - 'path/sda3-[a-z0-9]+-[0-9]+'
```

Examples

Check if *lxd/containers/juju-2g34g34-1* is mounted:

```
mount:
  regex:
  - 'lxd/containers/juju-[a-z0-9]+-[0-9]+'
```

Parameters

Parameter	Type	Comments
regex	str	Match regex string in mounts

Network module

Network exporter is sourcing */proc/net/tcp* for information about interfaces and ports attached.

Notation

```
network:
  port:
    - port_num1
    - port_num2
    - port_num3
```

Examples

Check if:

```
network:
  port:
    - 6789
    - 22
```

Parameters

Parameter	Type	Comments
port	list	Check list of port numbers (int) whether attached

Package module

Notation

```
package:
  - 'pkg_name1'
  - 'pkg_name2'
  - 'pkg_name3'
```

Examples

Check if:

```
package:
  - 'ceph'
  - 'ceph-common'
  - 'lxd'
  - 'lxd-client'
```

Parameters

Parameter	Type	Comments
pkg_name	str	Check package name if installed

Process module

Listing */proc* for running processes.

Notation

```
process:
- '/usr/bin/service'
```

Examples

Check if:

```
process:
- '/usr/bin/ceph-mon'
```

Parameters

Parameter	Type	Comments
service	str	Check process name if running

Service module

Systemd services. Works with *dbus* python module.

Notation

```
service:
  service-name:
    status: 'running'
```

Examples

Check if:

```
service:
  ceph-mon:
    status: 'running'
```

Parameters

Parameter	Type	Comments
name	str	Match service status

User module

Notation

```
user:
  user1:
    group: 'user1'
    dir: '/home/user1'
```

Examples

Check if:

```
user:
  ceph:
    group: 'ceph'
    dir: '/var/lib/ceph'
```

Parameters

Parameter	Type	Comments
user	str	User existing in pwd file
uid	int	User's uid
gid	int	User's gid
group	str	User's group name
dir	str	Path to user's homedir
gecos	str	A general information about the account
shell	str	User's shell

1.1.5 Deploy

1.1.6 Upgrade

1.1.7 Tests

Jujuna tests are designed to validate configuration of infrastructure in a fast way. It is able to discover many common issues, that do not appear in Juju status or during upgrade procedure.

Test suite is a declarative config of infrastructure. Status is declared by referencing brokers and their variables.

Brokers are modules that are using exporters to extract specific information from units. They represent important system values. Exporters are modules that read and export information from units to brokers. There the information is evaluated.

Test brokers/exporters (named respectively):

- api
- file
- mount
- network
- package
- process
- service
- user

1.1.8 Clean

CHAPTER 2

Indices and tables

- genindex
- modindex
- search