ra-tester

Resource agent testing framework

Damien Ciabrini
Principal Software Engineer

Red Hat

In a nutshell

A simple python framework for testing resource agents

A tool for pre-provisioning VM and clusters

A Command-line for running test cases

How it all started

Red Hat

The tale of a maintainer...

Maintainer of the Galera resource-agent, used in OpenStack

A rather new agent, updating it was always delicate

Only tool to test it back then was phd

- Automates resource deployments in a pacemaker cluster
- Difficult to test all the various paths or states of an agent

Idea #1: unit test the resource agent

Come up with a set of repeatable tests (features, edge cases, recoveries...)

Without mocking pacemaker or communication

Run tests in several initial conditions (IPv6, containers, pacemaker remotes...)

Bonus: make it applicable to other resource agents

first candidates: rabbitmq, redis, other agents used in OpenStack

Idea #2: Leverage existing stuff

Reimplementing an entire testing framework is no fun...

Luckily CTS exists 😊

- A complete testing framework for pacemaker
- Implements cluster tests: start, stop, rolling update, degraded network
- Comes with various runners: sequence, random, containerized
- Auditing: check for bad logs, monitor disk usage

CTS: Cluster Test Suite

A Python package to run cluster tests expressed as python classes

`CTSScenario`: runs a series of `CTSTest`, analyzes failures in logs, bookkeeping

`ScenarioComponent`: setup and teardown actions for the run

`CTSTest`: set up a test, execute it and tear it down

Each test can whitelist a set of warn/error log messages to pass

`Environment`: auto-detects system, store k-v pairs to configure the run

CTS: Cluster Test Suite - Internals

`Audit`: check system sanity throughout the run (SHM, disk, logs)

`RemoteFactory`: remote command execution, copy files to/from remote hosts

Implemented via qa`rsh` (passwordless rsh)

`Watcher`: remote log watching mechanism

Scans system's journal on all cluster nodes concurrently

Searches for a series of regex in remote logs until any/all match

⇒ used in `CTSTest`(s) to assert conditions during a test

How does it relate to resource agents?

- CTS is great for building a higher-level framework

  Can monitor the cluster's state

  Can parse remote logs for arbitrary strings

  Implements the basics: setup, execution, teardown, reporting

- One just need a few more things on top of it (quite a lot, in fact)

Introducing ra-tester

Red Hat

What we want to achieve

- A repeatable way of setting up a cluster!
- A programmatic way to define and implement tests
- Run same tests with different configurations
- Work no matter the host system (distribution, packages, containers...)

Out of scope: No long-running test, no chaos testing...

Setting up a VM-based test cluster

You need a cloud image, libvirt and a hypervisor

```
./ra-tester-build-vms --hypervisor root@hypervisor
                      --img /tmp/CentOS-7-x86_64-GenericCloud.qcow2
                      --name centos --nodes 3
                      --opt-cmd "yum install -y centos-release-openstack-train"
                      --opt-pkgs vim
```

Libvirt auto-detects the distribution, install packages, enable services

cloud-init configures the VM (network, NICs, IP, hostname...)

Setting up a VM-based test cluster

Ra-tester VMs have two networks:

- One DHCP network for accessing the node over SSH
- One static IP network for cluster and resources

Fencing device configuration

- supports xvm or virsh fence agents out of the box

Can be configured to enable/disable SELinux

Basic example: start galera

Once the VMs are configured:

```
./ra-tester --ssh ssh_config_centos --nodes 'centos1 centos2 centos3'
            --choose Galera:SimpleSetup:ClusterStart
            --set keep_resources=1
            --set verbose=1
```

Basic example: start galera

```
Feb 02 16:39:40 >>>>>>>>>>>>>>>> Starting scenario Galera:SimpleSetup (1 tests)
Feb 02 16:39:46 Prepare log directories on all cluster nodes
Feb 02 16:39:48 Writing log with key: dae09a5a-2181-470d-bfed-46dc86c81288
Feb 02 16:39:52 [Installing/Updating dependencies]
Feb 02 16:39:53 > [centos1] yum install -y mariadb-server-galera
Feb 02 16:40:20 > [centos2] yum install -y mariadb-server-galera
Feb 02 16:40:43 > [centos3] yum install -y mariadb-server-galera
Feb 02 16:41:06 Setting up galera config files
Feb 02 16:41:08 recreating empty mysql database on node centos1
Feb 02 16:41:13 recreating empty mysql database on node centos2
Feb 02 16:41:19 recreating empty mysql database on node centos3
Feb 02 16:41:24 Creating cluster for nodes ['centos1', 'centos2', 'centos3']
Feb 02 16:41:24 > [centos1] pcs cluster auth -u hacluster -p ratester centos1 centos2 centos3
Feb 02 16:41:26 > [centos1] pcs cluster setup --force --name ratester centos1 centos2 centos3
Feb 02 16:41:44 > [centos1] systemctl enable pacemaker
Feb 02 16:41:44 > [centos2] systemctl enable pacemaker
Feb 02 16:41:44 > [centos3] systemctl enable pacemaker
Feb 02 16:41:44 > [centos1] pcs cluster start --all
Feb 02 16:41:47 > [centos1] pcs property set stonith-enabled=false
Feb 02 16:42:14 Running test ClusterStart           (centos1)     [  1]
Feb 02 16:42:14 > [centos1] pcs resource create galera ocf:heartbeat:galera
wsrep_cluster_address='gcomm://centos1,centos2,centos3'  log=/var/log/mysql/mysqld.log op promote timeout=60
on-fail=block meta master-max=3 --master --disabled
Feb 02 16:42:17 > [centos1] pcs resource refresh galera-master
Feb 02 16:42:22 > [centos1] pcs resource enable galera-master
Feb 02 16:42:52 Leaving cluster running on all nodes
Feb 02 16:42:56 ****************
Feb 02 16:42:56 Overall Results:{'success': 1, 'failure': 0, 'BadNews': 0, 'skipped': 0}
```

Class hierarchy in ra-tester

`RARunner` (`Scenario`): destroys existing cluster and runs tests in sequence

`RATesterScenarioComponent` (`ScenarioComponent`):

Sets up new pacemaker cluster

Installs packages, container, templated config files

Runs optional setup step (e.g. DB creation on disk)

`RAConfig`: defines the list of known settings for a resource agent test suite

Example: galera base scenario

```
class PrepareCluster(RATesterScenarioComponent):
    def __init__(self, environment):
        RATesterScenarioComponent.__init__(self, environment, scenario_module_name="galera")
        self.dependencies = ["mariadb-server-galera"]

    def setup_configs(self, cluster_nodes):
        gcomm = "gcomm://" + (",".join(cluster_nodes))
        for node in cluster_nodes:
            shortname = self.node_shortname(node)
            self.copy_to_node(node, [(galeracfg, "/etc/my.cnf.d/galera.cnf")], "root", "0444",
                              {"%GCOMM%": gcomm, "%HOSTNAME%": shortname})

    def setup_state(self, cluster_nodes):
        config=self.Env["config"]
        for node in cluster_nodes:
            if not bool(config["skip_install_db"]):
                self.rsh(node, "mkdir -p /var/lib/mysql /var/log/mysql")
                self.rsh(node, "sudo mysql_install_db")
```

Example: instantiated scenario

Simple scenario that sets up pacemaker and galera before running tests

```
class SimpleSetup(PrepareCluster):
    def setup_scenario(self, cluster):
        config = RAConfig(self.Env, self.module_name, {
            "ocf_name": "galera",
            "skip_install_db": False,
        })
        self.Env["config"] = config
        PrepareCluster.setup_scenario(self,cm)
```

Class hierarchy in ra-tester (2)

`RATest`: create a resource, initially disabled

Runs resource-specific test (e.g. pcs command, node reboot...)

Assert resource state from cluster state, journal logs

`RAAction`: actions that can be run by `RAScenario` or `RATest`

Get node IP/IPv6/FQDN, get/set cluster attribute...

Run remote command, wait until remote command is successful...

Example: galera cluster start

```
class GaleraCommonTest(RATest):
    def resource_command(self, cluster_nodes, config):
        name = config["ocf_name"]
        nodes = ",".join(cluster_nodes)
        return "pcs resource create %s ocf:heartbeat:galera wsrep_cluster_address='gcomm://%s'" \
            " log=/var/log/mysql/mysqld.log op promote timeout=60 on-fail=block" % \
            (name, nodes)

    def setup_test(self, node):
        self.setup_inactive_resource(self.Env["nodes"])

    def errorstoignore(self):
        return RATest.errorstoignore(self) + [
            r"ERROR:\s*MySQL is not running",
            r"rsyncd.*:\s*rsync error: received SIGINT, SIGTERM, or SIGHUP"
        ]
```

Example: galera cluster start

```
class ClusterStart(GaleraCommonTest):
    def __init__(self, cm):
        GaleraCommonTest.__init__(self,cm)
        self.name = "ClusterStart"

    def test(self, target):
        name = config["ocf_name"]
        target_nodes = self.resource_target_nodes(config, self.Env["nodes"])
        patterns = [self.ratemplates.build("Pat:RscRemoteOp", "promote", name, n, 'ok') \
                    for n in target_nodes]
        watch = self.make_watch(patterns)
        self.rsh_check(self.Env["nodes"][0], "pcs resource enable %s"%config["name"])
        watch.lookforall()
        assert not watch.unmatched, watch.unmatched
```

Handy mechanisms

Same test, multiple configs

Thanks to `RAConfig`, the same tests can be run with different initial conditions

```
config = RAConfig(self.Env, self.module_name, {
            "name": cluster.meta_promotable_resource_name("galera"),
            "ocf_name": "galera",
            "alt_node_names": {},
            "meta": cluster.meta_promotable_config(len(self.Env["nodes"])),
            "user": "mysql",
            "bundle": None,
            "skip_install_db": False,
            "tls": False,
            "ipv6": False,
        })
```

Config can be overridden with `./ra-tester --set key=value`

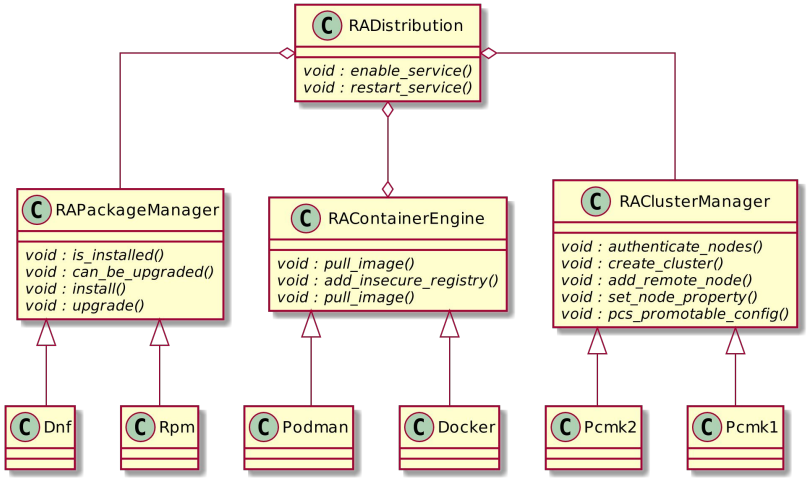
Same tests, different host system

The resource agent is used in many different ways

container vs bare metal

pacemaker 1 vs pacemaker 2

docker vs podman



`RADistribution` hides those details with APIs

Detects the running distribution with `lsb_release`

What's next?

Unsorted list of ideas

More resource agents, more tests...

Code coverage with a dedicated `ScenarioComponent`

Export CTS reports to JUnit/XML reports

Other host setup method (provision cluster with OpenStack)

CI jobs for non-regression tests

<your-request-here>

Thank you

http://github.com/dciabrin/ra-tester

dciabrin at #clusterlabs on freenode


linkedin.com/company/red-hat


youtube.com/user/RedHatVideos


facebook.com/redhatinc


twitter.com/RedHat

Red Hat