Yardstick in depth

Jing Lu
lvjing5@huawei.com
Content

1. Yardstick framework architecture
2. Add a test scenario & Write a test case
3. VSperf integration
Use-Case view
Deployment view

Yardstick

Jump Server

Network

Control Server

Control Server

Control Server

Compute Server

Compute Server
Concept

Scenario: Type of measurement/benchmark (e.g., Ping, Pktgen, Iperf, LmBench).

Context: The set of cloud resources used by a scenario.

Runner: Logic that determines how a test scenario is run and reported.
Logical view

Yardstick Overview, Release Draft (5b6dc00)

Test Scenario - Type/class of measurement for example Ping, Pkt gen, (Perf, LMBench,...)

Dispatcher - Choose user defined way to store test results.

Task Commands is the "yardstick task" subcommand's main entry. It takes YAML file (e.g. test.yml) as input, and uses Heat Context to convert the YAML file's context section to HTT.

After OpenStack heat stack is deployed by Heat Context with the converted HTT, Task Commands use Runner to run specified Test Scenario. During runner initialization, it will create output process. The output process uses Dispatcher to push test results. The Runner will also create a process to execute Test Scenario. And there is a multiprocessing queue between each runner process and output process, so the runner process can push the real-time test results to the storage media.

Test Scenario is commonly connected with VMs by using ssh. It sets up VMs and runs test measurement scripts through the ssh tunnel. After all Test Scenario is finished, Task Commands will undeploy the heat stack. Then the whole test is finished.
Add a scenario & Write a test case

1. Tool installation
2. Task configuration file
3. Scenario file
4. Unit test file

Yardstick

- Test.yaml
- Output.json
- Results API
- InfluxDB

Test Scenario

1. Input
2. Deploy
3. SSH
4. Test
5. Output
6. Undeploy

OpenStack

- Heat
- Nova
- Neutron
- Phy-eth
- Br-ex
- Br-int
- Host VM
- Target VM
## Netperf scenario

<table>
<thead>
<tr>
<th>General case</th>
<th>Netperf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Tool installation</td>
<td>1. Build VM image</td>
</tr>
<tr>
<td>2. Task configuration file</td>
<td>2. netperf.yaml</td>
</tr>
<tr>
<td>3. Scenario file</td>
<td>3. netperf.py</td>
</tr>
<tr>
<td>4. Unit test file</td>
<td>4. test_netperf.py</td>
</tr>
</tbody>
</table>
Netperf is required to be installed on VMs.

Yardstick has a tool for building a custom Ubuntu Cloud Server image containing all the required tools to run test cases.
Task configuration file - netperf.yaml

```yaml
...
schema: "yardstick:task:0.1"
scenarios:
  - type: Netperf
    options:
      testname: 'UDP_STREAM'
      send_msg_size: 1024
      duration: 20
    host: Chang'e.demo
target: Houyi.demo
runner:
  type: Iteration
  iterations: 1
  interval: 1
sla:
  mean_latency: 100
  action: monitor
```

- **type**: corresponding to “__scenario_type__” in scenario file
- **options**: contains parameters used by the netperf runtime usage:
  ```bash
  netperf -H ipaddr -l testlen -t testname -m send_msg_size
  ```
- **runner**: 4 pre-defined types (i.e. arithmetic, duration, sequence, iteration)
- **SLA**: ‘monitor’ or ‘assert’
Task configuration file - netperf.yaml

Context

- type: ‘heat’ or ‘Dummy’
- name: name of the heat stack
- image: vm image
- flavor: openstack virtual hardware template
- user: ssh log names

```yaml
context:
  type: heat
  name: demo
  image: yardstick-trusty-server
  flavor: yardstick-flavor
  user: ubuntu

  placement_groups:
    pgrp1:
      policy: "availability"

  servers:
    Chang'e:
      floating_ip: true
      placement: "pgrp1"
    Houyi:
      floating_ip: true
      placement: "pgrp1"

  networks:
    test:
      cidr: '10.0.1.0/24'
```
Scenario file

- Define a ‘setup’ function
- Define a ‘run’ function
- Define a ‘teardown’ function

- Test environment setup
- Execute the test
- Reset environment
netperf.py – def setup(

# log on the host and target VMs
LOG.info("user:%s, target:%s", target_user, target_ip)
self.server = ssh.SSH(target_user, target_ip,
key_filename=target_key_filename)
self.server.wait(timeout=600)

LOG.info("user:%s, host:%s", host_user, host_ip)
self.client = ssh.SSH(host_user, host_ip,
key_filename=host_key_filename)
self.client.wait(timeout=600)

# copy script to host
self.client.run("cat > ~/netperf.sh",
stdin=open(self.target_script, "rb"))

Use ssh protocol to log into VM

Copy test script to VM
netperf.py – def run()

```python
cmd = 'sudo bash netperf.sh %s' % (cmd_args)
LOG.debug("Executing command: %s", cmd)
status, stdout, stderr = self.client.execute(cmd)

if status:
    raise RuntimeError(stderr)

result.update(json.loads(stdout))

if result['mean_latency'] == '':
    raise RuntimeError(stdout)

# sla check
mean_latency = float(result['mean_latency'])
if "sla" in self.scenario_cfg:
    sla_max_mean_latency = int(
        self.scenario_cfg['sla']['mean_latency'])

assert mean_latency <= sla_max_mean_latency,
    "mean_latency %f > sla_max_mean_latency(%f); 
    (mean_latency, sla_max_mean_latency)"
```

Compose Netperf command arguments and entire command line

- using parameters from the scenarios section

Execute the test

- Via shell script

Get test result

SLA check
Reset environment configurations, e.g. enable security_group

In netperf scenario, teardown() does nothing.
  As the heat stack will be undeployed after the netperf test is finished.
## VSperf integration

<table>
<thead>
<tr>
<th>Generic case</th>
<th>Netperf</th>
<th>VSperf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Tool installation</td>
<td>1. Build VM image</td>
<td>1. Vsperf installation</td>
</tr>
<tr>
<td>2. Task configuration file</td>
<td>2. netperf.yaml</td>
<td>2. vsperf.yaml</td>
</tr>
<tr>
<td>3. Scenario file</td>
<td>3. netperf.py</td>
<td>3. vsperf.py</td>
</tr>
<tr>
<td>4. Unit test file</td>
<td>4. test_netperf.py</td>
<td>4. test_vsperf.py</td>
</tr>
</tbody>
</table>
CLI install VSperf

Yardstick plugin Commands

Example Command Lines:

$ yardstick plugin install <vsperf_conf_file>

$ yardstick plugin remove <vsperf_conf_file>
vsperf.yaml - scenarios

Netperf

```
schema: "yardstick:task:0.1"

scenarios:
  -
    type: Netperf
    options:
      testname: 'UDP_STREAM'
      send_msg_size: 1024
      duration: 20

host: Chang'e.demo
target: Houyi.demo

runner:
  type: Iteration
  iterations: 1
  interval: 1
sla:
  mean_latency: 100
  action: monitor
```

VSperf

```
schema: "yardstick:task:0.1"

scenarios:
  -
    type: VSperf
    options:
      tests: 'RCF2544Tput'
      conf_file: 'my_settings.py'
      test_params: 'duration=10'

runner:
  type: Iteration
  iterations: 1
  interval: 1
sla:
  key: value
  action: monitor"
vsperf.yaml - context

Netperf

```yaml
context:
  name: demo
  image: yardstick-trusty-server
  flavor: yardstick-flavor
  user: ubuntu

placement_groups:
  pgrp1:
    policy: "availability"

servers:
  Chang'e:
    floating_ip: true
    placement: "pgrp1"
  Houyi:
    floating_ip: true
    placement: "pgrp1"

networks:
  test:
    cidr: '10.0.1.0/24'
```

VSperf

```yaml
context:
  type: Dummy
```
vsperf.py - `def setup()`
vsperf.py – def run( )

Compose vsperf command arguments and entire command line
- using parameters from the scenarios section

Execute the test
- Via shell script

Get test result

SLA check

```python
# cmd = "sudo bash netperf.sh %s" % (cmd_args)
LOG.debug("Executing command: %s", cmd)
status, stdout, stderr = self.client.execute(cmd)

if status:
    raise RuntimeError(stderr)

result.update(json.loads(stdout))

if result['mean_latency'] == '':
    raise RuntimeError(stdout)

# sla check
mean_latency = float(result['mean_latency'])
if "sla" in self.scenario_cfg:
    sla_max_mean_latency = int(
        self.scenario_cfg['sla']['mean_latency'])

assert mean_latency <= sla_max_mean_latency,
    "mean_latency %f > sla_max_mean_latency(%f); " %
    (mean_latency, sla_max_mean_latency)
```
Demo
Thank You!