testpool Documentation

Release 0.0.3

Mark Hamilton

Sep 03, 2018

Contents

Contents:

CHAPTER 1

Quick Start

Testpool maintains a pool of pristine VMs cloned from a template. Users can immediate acquire a VM, use it and then throw it away. Testpool then replaces discarded VMs with a fresh clone. Cloning VMs can take a considerable amount of time, but with a pool of VMs, acquiring a single VM is immediate. Testpool supports KVM and docker.

There are three demonstrations of Testpool. One uses fake resources to demostrate a large deployment. The second demo usss docker and is designed to work on a sinlge laptop for the sake of having an easy demo. The third uses KVM hypervisors.

1.1 Simulation Demonstration

1.2 Testpool Installation

We'll install Testpool from Debian.

1. Install several required packages:

```
sudo apt-get install -y apt-file libvirt0 virtinst pm-utils
sudo apt-get install -y libvirt-bin libvirt-dev qemu-system debhelper
sudo apt-get install -y python-yaml python-pip python-all enchant
sudo apt-get install -y fakeroot dh-python
sudo apt-file update
sudo pip install -q docker==3.4.1 docker-pycreds==0.3.0 requests>=2.19.1
sudo pip install -q pytz>=2018.5 Django==1.11.13
sudo pip install -q djangorestframework>=3.8.2
sudo pip install -q django-pure-pagination==0.3.0
sudo pip install -q django-split-settings==0.3.0
sudo pip install -q libvirt-python==4.0 ipaddr>=2.1.11 structlog>=16.1.0
sudo pip install -q pyyaml easydict pyenchant==2.0.0 pybuild==0.2.6
```

2. Download Testpool from github release page:

   Check for the latest release at:

https://github.com/testcraftsman/testpool/releases

below is an example:

https://github.com/testcraftsman/testpool/releases/download/v0.1.5/python-testpool_0.1.5-1_all.deb sudo dpkg -i python-testpool_0.1.5-1_all.deb

1. Check testpool services are running:

   systemctl status tpl-daemon systemctl status tpl-db

2. Run the Testpool demo that ships with the product

   tpl-demo -v

   The demo creates several fake pools, then periodically acquires a resource then releases it. After 60 seconds, all resources are released for 1 minute. The dashboard shows the status of the various resources:

   http://127.0.0.1:8000/testpool/view/dashboard

   Alternatively, *tpl-demo* can be run with *–product docker*. Don't run tpl-demo if going through the next section in the **Short Tour**.

1.3 A Short Tour

In order for Testpool to manage VMs, Hypervisor information is registered with Testpool along with a VM template.

1.4 KVM Hypervisor Demonstration

Normally Testpool is installed on a central server and configured to manage several hypervisors. Testpool supports KVM which is required for this demonstration.

To expedite this guide, Testpool content is installed on the KVM hypervisor. For final installation, Testpool can be installed either on the hypervisor or a separate system. The differences will be identified during the installation steps.

1.5 Testpool Stack

Testpool consists of several releated products. They are:

- Testpool-client - installed on each client, this package provides an API

  to acquire and release VMs. This is useful when writing tests and not wanting to use the REST interface directly.

- Testpool-beat - pushes testpool metrics to logstash. This is useful for monitoring VM pools.

Make sure to install the appropriate major and minor version that matches the testpool package. For example, if the version of Testpool is 0.1.0. Then install 0.1.Y of Testpool-client and Testpool-beat. Where Y can be any value.

1.6 KVM Installation

For this quick start guide, we'll need a single VM named test.template on the hypervisor which is off and ready to be cloned. When the VMs starts it must use DHCP to acquire its IP address. What the VM is running is not important and

there are good instructions on the internet for setting up a KVM hypervisor and creating a VM. For installing KVM on Ubuntu 18.04, refer to this site https://help.ubuntu.com/community/KVM/Installation. Once complete, you will need the following information:

- user and password that can install VMs. This is the user that is part of the libvirtd and kvm groups.
- IP Address of the KVM hypervisor if Testpool is not running on the hypervisor

For the rest of this guide, we'll assume the user tadmin with password 'password'. Since Testpool is installed on the hypervisor, the IP address used is localhost.

Now a single VM is required which represents the template that is managed and cloned by Testpool. Using virt-manager, these instructions will create an Ubuntu 16.04 server VM.

1. sudo apt-get install virt-manager
2. Run **virt-manager**
3. From File, choose **Add Connection**.
4. If applicable, choose **Connect to remote host**
5. Enter admin for **Username** and IP address for the **Hostname**. This may be either localhost or the IP address of the KVM hypervisor. The default ssh method will probably work.
6. Now connect and enter the user password.
7. Select Hypervisor in the virt-manager,
8. Choose **Create a new virtual manager**.
9. Choose **Network Install (HTTP, FTP or NFS)** then Forward.
10. For URL, enter **http://us.archive.ubuntu.com/ubuntu/dists/bionic/main/installer-amd64/** The URL changes periodically, check the Ubuntu site for the latest valid links.
11. Choose appropriate RAM and CPU. For a demo, select 512 and 1 CPU.
12. Create a disk with 5 GiB of space.
13. Then select Finish fro the network setup.

1.7 A Short Tour

In order for Testpool to manage VMs, Hypervisor information is registered with Testpool along with a VM template.

Create a VM on the KVM hypervisor called test.template and keep it shutdown. Now create a Testpool pool given the IP address and name of the VM template. Since we're running on the hypervisor, the IP address is localhost.

Where hypervisor-ip is replaced with the actual Hypervisor IP address. While running Testpool on the hypervisor, use the tpl CLI to create a test pool:

```
./bin/tpl pool add example kvm qemu:///system test.template 3
```

Confirm the pool is valid:

```
./bin/tpl pool detail example
```

The Testpool Daemon will clone 3 VMs from the test.template. This can take a while which is the point of this product. In that, Testpool generates new clean clones based on test.template. The VMs available line in the detail output shows the current number of available VMs. Use **virt-manager** to see the VMs being created.

From this point, Testpool daemon is cloning VMs. There are several examples to look through. The file examples/rest.py provides documentation and demonstrates how to use Testpool's REST interface. Simply refer to the file examples/rest.py.

Additionally, Testpool-client can be installed which provides a python API on top of the REST interface. To learn more, http://testpool-client.readthedocs.io/en/latest.

```
"""
Examples on how to call the REST interfaces. Read the quick start guide in
order to configure Testpool server and then come back to this script.

As discussed in the Testpool quickstart guide. This example uses a
profile named example. These examples work best when all VMs have been cloned
and have retrieved their IP address.  Make sure VMs are avaliable, run:

  ./bin/tpl profile list

To run this file type

  py.test -s examples/python_api.py

These examples illustrates the use of the testpool.client. The global variable
GLOBAL in conftest defines the Testpool profile. Once a VM is acquired, this
test can login and use the VM throughout the entire testsuite. This assumes
that the VM has negotiated an IP address usually throught DHCP.

As these examples are running, use virt-manager to see hypervisor changes.
"""
import time
import json
import datetime
import urllib
import unittest
import requests
import conftest


TEST_URL = "http://%(hostname)s:8000/testpool/api/v1/" % conftest.GLOBAL


def acquire_get(url):
    """ Wrap acquire with a delay in case none are available. """
    ##
    # previous tests may have acquired all VMs wait for a while to
    # acquire one
    for _ in range(10):
        resp = requests.get(url)
        if resp.status_code == 403:
            time.sleep(60)
        else:
            rsrc = json.loads(resp.text)
            return rsrc
    resp.raise_for_status()
    ##
    return None


class Testsuite(unittest.TestCase):
```

(continues on next page)

(continued from previous page)

```
    """ Demonstrate each REST interface. """

    def test_profile_list(self):
        """ test_profile_list Show how to list profile content."""

        url = TEST_URL + "profile/list"
        resp = requests.get(url)
        resp.raise_for_status()
        profiles = json.loads(resp.text)

        self.assertEqual(len(profiles), 1)
        self.assertEqual(profiles[0]["name"], "example")

        self.assertTrue("resource_max" in profiles[0])
        self.assertTrue("resource_ready" in profiles[0])

    def test_profile_acquire(self):
        """ test_profile_acquire acquire a VM. """

        url = TEST_URL + "profile/acquire/example"
        requests.get(url)
        rsrc = acquire_get(url)

        ##
        # Cloned VMs begin with the name of the template.
        self.assertTrue(rsrc["name"].startswith("test.template"))
        self.assertTrue(len(rsrc["name"]) > len("test.template"))
        ##

        rsrc2 = acquire_get(url)

        self.assertTrue(rsrc2["name"].startswith("test.template"))
        self.assertTrue(len(rsrc2["name"]) > len("test.template"))

        url = TEST_URL + "profile/release/%d" % rsrc["id"]
        acquire_get(url)

        url = TEST_URL + "profile/release/%d" % rsrc2["id"]
        acquire_get(url)

    def test_acquire_too_many(self):
        """ test_acquire_too_many attempt to acquire too many VMs."""

        prev_rsrcs = set()
        url = TEST_URL + "profile/acquire/example"

        ##
        # Take all of the VMs
        for _ in range(conftest.GLOBAL["count"]):
            rsrc = acquire_get(url)

            self.assertTrue(rsrc["name"].startswith("test.template"))
            self.assertFalse(rsrc["name"] in prev_rsrcs)

            prev_rsrcs.add(rsrc["id"])
        ##
```

(continues on next page)

(continued from previous page)

```
        resp = requests.get(url)
        self.assertEqual(resp.status_code, 403)

        for rsrc_id in prev_rsrcs:
            url = TEST_URL + "profile/release/%d" % rsrc_id
            acquire_get(url)

    def test_acquire_renew(self):
        """ test_acquire_renew renew an acquired VM. """

        url = TEST_URL + "profile/acquire/example"

        rsrc = acquire_get(url)
        rsrc_id = rsrc["id"]

        url = TEST_URL + "resource/renew/%(id)s" % rsrc
        resp = requests.get(url)
        resp.raise_for_status()
        rsrc = json.loads(resp.text)
        self.assertEqual(rsrc["id"], rsrc_id)

        params = {"expiration": 100}
        resp = requests.get(url, urllib.urlencode(params))
        resp.raise_for_status()
        rsrc = json.loads(resp.text)
        self.assertEqual(rsrc["id"], rsrc_id)

        ##
        # Check to see if the expiration is roughly 100 seconds.
        timestamp = datetime.datetime.strptime(rsrc["action_time"],
                                               "%Y-%m-%dT%H:%M:%S.%f")
        expiration_time = timestamp - datetime.datetime.now()
        self.assertTrue(expiration_time.seconds <= 100)
        self.assertTrue(expiration_time.seconds >= 90)
        ##

        url = TEST_URL + "profile/release/%d" % rsrc_id
        resp = requests.get(url)
        resp.raise_for_status()
```

CHAPTER 2

Installation

2.1 Getting Testpool

Testpool is installed from source, download the latest from GitHub. This is also where we track issues and feature request.

2.2 What is Installed

Testpool consists of:

1. A database installed on an Ubuntu 16.04 system, which can also be a KVM hypervisor
2. testpool-client, another repo, is installed on every client

Actually the last item is optional in that the testpool-client provides an API above the server's REST API. One could simply use the REST interface directly.

2.2.1 Testpool Server Installation on Ubuntu 16.04

A single testpool server is required. It maintains VM pool requirements for each hypervisor. Here are the steps to install a testpool's server:

1. Download testpool from github release area, for example v0.1.0:

```
wget https://github.com/testcraftsman/testpool/archive/v0.1.0.tar.gz
tar -xf testpool-0.1.0.tar.gz
```

2. Skip this step if you are installing Testpool on the KVM hypervisor, most likely these packages are already installed.

   sudo -H apt-get install -y libvirt-dev libxen-dev virtinst

3. Install several required packages:

```
cd testpool-0.1.0
cat requirements.system | sudo xargs apt-get -y install
sudo -H apt-file update
sudo -H pip install --upgrade pip
sudo -H pip install -qr requirements.txt
```

4. Create debian packages,in a shell run:

```
make deb.build
```

5. Install:

```
sudo -H make install
```

CHAPTER 3

Development

Prior to developing in testpool:

sudo -H make setup ./bin/tpl-db migrate

Additionally docker must be installed, tests and development rely on it.

3.1 Web Development

To simplify web development, developers can avoid using an actual hypervisor. Instead developers can create a fake pool.

Since testpool uses django, once the web server is running it will automatically restart when content changes. In one shell, run the testpool web server:

```
./bin/tpl-db runserver
```

In a new shell start the testpool daemon.:

```
./bin/tpl-daemon -vv
```

In a new shell, create several pools. The following creates two pool . The first pool uses template0 to generate two fake VMs. The second pool creates three fake VMs from template1.:

```
./bin/tpl pool add localhost fake pool0 template0 2
./bin/tpl pool add localhost fake pool1 template1 3
```

The tpl-daemon will over time generate 5 VMs in the ready state. In other words, fake VMs are transitioned from pending to reserved over a short period of time. Testpool web content showing overall VM pool statistics can be found:

```
http://127.0.0.1:8000/testpool/pool
```

To manipulate VM content, meaning reserve and release VMs, review the vm command help:

```
./bin/tpl vm --help
```

3.2 Debian Packaging

Before debian packages can be created apt-file must be installed and updated so that the python requirements.txt file can be mapped to equivalent debian package dependencies.:

```
sudo apt-get install apt-file
sudo apt-file update
pip install pep8>=1.7.0 pylint>=1.5.4 pytest>=2.8.3
```

Make sure to set EMAIL before using dch Also note that versions are incremented in the change log:

```
dch -U
```

Build Testpool debian package and install:

```
make deb.build
sudo make install
```

CHAPTER 4

Log Stash Support

Testpool provides a structured log of pool status that includes the number of available VMs for each pool. This information can be pushed to logstash and visualized with Kibana or Graphana.

The following instructions explain how to enable structured logging and push them to Logstash using Filebeat.

4.1 ELK Installation

ELK stack 5.5 is required which natively supports JSON FileBeat output. There are numerous sites to explain ELK installation e.g. http://www.itzgeek.com/how-tos/linux/ubuntu-how-tos/setup-elk-stack-ubuntu-16-04.html was used to test the following content.

4.2 Testpool Configuration

Configure testpool to save pool status. Edit the YAML file:

```
/etc/testpool/testpool.yml
```

Validate changes:

```
tplcfgcheck /etc/testpool/testpool.yml
```

Uncomment tpldaemon.pool.log. The default value is **/var/log/testpool/pool.log** and restart testpool daemon:

```
sudo systemctl restart tpl-daemon
```

Configure Logstash to receive JSON structured content. An example configuration file at **/etc/testpool/etc/logstash/conf.d/02-testpool-beat-input.conf**.:

```
sudo cp /etc/testpool/etc/logstash/conf.d/02-testpool-beat-input.conf /etc/logstash/
↪conf.d/02-testpool-beat-input.conf
sudo systemctl restart logstash
```

Make sure elastic search and logstash start on boot:

sudo systemctl enable logstash sudo systemctl enable elasticsearch

The Logstash **02-testpool-beat-input.conf** content.

```
##
# testpool stores profile status in a structured log located at
# /var/log/testpool/profile.log. As of elastic stack 5.0,
# structured log is supported natively.
input {
  beats {
    port => 5045
    codec => "json"
#    ssl => true
#    ssl_certificate => "/etc/ssl/logstash-forwarder.crt"
#    ssl_key => "/etc/ssl/logstash-forwarder.key"
  }
}

output {
 elasticsearch {
  hosts => "127.0.0.1"
    index => "%{[@metadata][beat]}-%{+YYYY.MM.dd}"
  }
  stdout {
    codec => rubydebug
  }
}
```

Configure Filebeat to push JSON content. An example is available at **/etc/testpool/filebeat/filebeat.yml** and can be copied verbatim.:

```
sudo cp /etc/testpool/filebeat/filebeat.yml /etc/filebeat/filebeat.yml
sudo systemctl restart filebeat
```

The content below shows a sample Filebeat configuration.

```
############## Testpool Filebeat Configuration Example ##############
#
# This file is an example configuration file highlighting only the
# most common options. The filebeat.full.yml file from the same
# directory contains all the supported options with more comments.
# You can use it as a reference.
#
# You can find the full configuration reference here:
# https://www.elastic.co/guide/en/beats/filebeat/index.html
#
#============= Testpool Filebeat prospectors ======================

filebeat.prospectors:

- input_type: log
```

(continues on next page)

(continued from previous page)

```
  ##
  # By default testpool stores profiel status at the path below.
  # Change this if you change the location in
  # /etc/testpool/testpool.yml
  paths:
    - /var/log/testpool/profile.log
  json:
    message_key: event
    keys_under_root: true
  ##
  # Change server to anything you want
  fields:
    server: localhost
  tags: [
    "testpool"
  ]

#========================== General =================================
#
# The name of the shipper that publishes the network data. It can be
# used to group all the transactions sent by a single shipper in the
# web interface.
#name: testpool

# The tags of the shipper are included in their own field with each
# transaction published.
#tags: ["service-X", "web-tier"]

#========================== Outputs =================================
#
# Configure what outputs to use when sending the data collected by
# the beat.  Multiple outputs may be used. The configuration below
#  assumes logstash

#---------------------- Elasticsearch output ------------------------
#output.elasticsearch:
  # Array of hosts to connect to.
  #hosts: ["localhost:9200"]
  #template.name: filebeat
  #template.path: filebeat.template.json

  # Optional protocol and basic auth credentials.
  #protocol: "https"
  #username: "elastic"
  #password: "changeme"

#----------------------- Logstash output ----------------------------
##
# uncomment the appropriate authentication
output.logstash:
  # The Logstash hosts
  hosts: ["127.0.0.1:5045"]
  template.name: filebeat
  template.path: filebeat.template.json

  # Optional SSL. By default is off.
  # List of root certificates for HTTPS server verifications
```

(continues on next page)

(continued from previous page)

```
  #ssl.certificate_authorities: ["/etc/pki/root/ca.pem"]

  # Certificate for SSL client authentication
  #ssl.certificate: "/etc/pki/client/cert.pem"

  # Client Certificate Key
  #ssl.key: "/etc/pki/client/cert.key"
  #ssl.key: "/etc/ssl/logstash-forwarder.key"
  #tls:
  #  certificate_authorities ["/etc/ssl/logstash-forwarder.crt"]


#============================ Logging =================================

# Sets log level. The default log level is info.
# Available log levels are: critical, error, warning, info, debug
#logging.level: debug

# At debug level, you can selectively enable logging only for some
# components. To enable all selectors use ["*"]. Examples of other
#  selectors are "beat", "publish", "service".
#logging.selectors: ["*"]
```

CHAPTER 5

Kibana 5.3

A sample Kibana dashboard with supporting visualization can be imported into Kibana. Content is available at: **/etc/testpool/kibana/testpool.json**. **/etc/testpool/kibana/testpool-dashboard.json**.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search