testpool Documentation

Release 0.0.3

Mark Hamilton

Mar 21, 2017

Contents

Contents:

CHAPTER 1

Quick Start

Testpool clones VMs from a template VM. Testpool monitors each VM waiting for their management interface to be assgiend an IP address, usually through DHCP. Once assigned, Testpool makes the VM available for users. Users acquire a VM and then release it when done. Where Testpool replaces the VM with a fresh clone. Cloning VMs can take a considerable amount of time. Testpool manages the VMs so that acquiring VMs is immediate.

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

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

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. 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. This section will provide references to these sites.

For installing KVM on Ubuntu 16.04, 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 as 'password'. Since testpool will be installed on the hypervisor, so 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/xenial/main/installer-amd64/** The URL changes periodically, check the ubuntu site for the latest valid links.

Testpool Installation

We'll install Testpool from source.

1. Download testpool from github release area:

```
wget https://github.com/testcraftsman/testpool/archive/v0.0.7.tar.gz
tar -xf testpool-0.0.7.tar.gz
```

2. Install several required packages:

```
cd testpool
cat requirements.system | sudo xargs apt-get install
sudo apt-file update
sudo pip install -qr requirements.txt
sudo pip install easydict
sudo pip install django-pure-pagination==0.2.1
sudo pip install django-split-settings==0.1.3
sudo apt-get -f install
```

3. Create debian packages,in a shell run:

```
make deb.build
```

4. Install Testpool server, Ina shell run:
5. Run Testpool database. In a shell run:

```
cd ../..
./bin/tpl-db runserver -v 3
```

6. In a second shell, run the Testpool daemon:

```
cd testpool
./bin/tpl-daemon -v
```

A Short Tour

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

Create a VM on the KVM hypervisor called test.template and keep it shutdown. Now create a testpool profile 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 profile:

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

Confirm the profile is valid:

```
./bin/tpl profile 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 is cloning VMs for use, the examples folder relies on this configuration to run. Refer to the example below to see how to use Testpool.

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

As discussed in the quickstart guide. This example uses a Testpool 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.

This example checks for a hypervisor profile named example. If
one does not exist, a fake profile is created.  A fake.profile is used
to show examples without having to take the time to configure an actual
hypervisor.

As these examples are running, use virt-manager to see the hypervisor change.
"""
import time
import unittest
from testpoolclient import tplvm
import conftest


class Testsuite(unittest.TestCase):
    """ Demonstrate testpool.client API. """
```

```
    def test_vm_acquire(self):
        """ test_vm_acquire.

        Acquire a single VM. Demonstrate how to determine the VMs IP address.
        """
        hndl = tplvm.Hndl(conftest.GLOBAL["hostname"],
                          conftest.GLOBAL["profile"], 10, True)
        current_vms = hndl.detail_get()["vm_available"]
        hndl.acquire()
        ##
        # The ip attribute provides the IP address of the VM.
        self.assertTrue(hndl.ip_addr is not None)
        ##

        ##
        # Assert that one VM was acquires. The number of avaliable VMs
        # will now be less max.
        details = hndl.detail_get()
        self.assertTrue(details["vm_available"] < current_vms)
        hndl.release()
        for _ in range(40 * 6):
            time.sleep(5)
            details = hndl.detail_get()
            self.assertTrue(details)
            if details["vm_available"] == current_vms:
                return
        details = hndl.detail_get()
        self.assertTrue(details)
        self.assertEqual(details["vm_available"], current_vms)

    def test_vm_context_manager(self):
        """ show using client context manager. """

        ##
        # Shows an example of the context manager.
        with tplvm.Hndl(conftest.GLOBAL["hostname"],
                        conftest.GLOBAL["profile"], 10) as hndl:
            ##
            # This assert is to show that a different VM was picked.
            self.assertTrue(hndl.vm.id is not None)
            self.assertTrue(hndl.vm.ip_addr is not None)
            ##
        ##

    def test_detail_get(self):
        """ Show Hypervisor details. """

        ##
        # Shows an example of the context manager.
        hndl = tplvm.Hndl(conftest.GLOBAL["hostname"],
                          conftest.GLOBAL["profile"], 10)
        details = hndl.detail_get()
        self.assertTrue(details)
        self.assertEqual(details["vm_max"], 3)
        ##

    def test_blocking(self):
```

```
        """ test_blocking. show waiting for VM to be available.

        There are at most 3 VMs available so take 4. With blocking
        there should never be an exception thrown.
        """

        ip_addresses = set()
        ##
        # Shows an example of the context manager.
        for count in range(3):
            with tplvm.Hndl(conftest.GLOBAL["hostname"],
                            conftest.GLOBAL["profile"], 10, True) as hndl:
                ##
                # This assert is to show that a different VM was picked.
                self.assertTrue(hndl.vm)
                ip_addresses.add(hndl.vm.ip_addr)
                ##
        ##

        hndl = tplvm.Hndl(conftest.GLOBAL["hostname"],
                          conftest.GLOBAL["profile"], 10, True)
        hndl.acquire(True)
        self.assertTrue(hndl.vm.ip_addr not in ip_addresses)
        hndl.release()

        count = 0
        for _ in range(100):
            details = hndl.detail_get()
            count = details["vm_available"]
            if count == 3:
                return
            time.sleep(20)
        raise ValueError("never recovered all three VMs.")
```

CHAPTER 2

Installation

Getting Testpool

If you want the latest code you'll need a GitHub account. This is also where we track issues and feature request. This code is committed into <http://guthub.com/testcraftsman/testpool>.

What is Installed

Testpool consists of:

1. a KVM hypervisor
2. A test pool database installed on an Ubuntu 16.04 system, which can be on the KVM hypervisor
3. testpool client software installed on every client

Actually the last item is optional in that the client API is a thin wrapper around the server's REST interface. One could simply use the REST interface on each client. For evaluation purposes a single system can be used to install both the server and client packages. Actual deployments would install the server on a single system and the client packages on each client system.

Testpool Server Installation on Ubuntu 16.04

A single testpool server is required for store VM pool status. Here are the steps for installing testpool's server:

1. Download testpool from github release area:

   wget https://github.com/testcraftsman/testpool/archive/v0.0.7.tar.gz tar -xf testpool-0.0.7.tar.gz

1. Install several required packages:

   cd testpool cat requirements.system | sudo xargs apt-get install sudo pip install -qr requirements.txt sudo pip install easydict sudo apt-file update

1. Install latest testbed which can be found at **https://github.com/testbed/testbed/releases**. For example:

   sudo pip install https://github.com/testbed/testbed/archive/v0.0.7.tar.gz

1. Add testbed configuration
2. Copy example testbed configuration

   cd /usr/local/testbed cp examples/etc/mysql.cnf etc/mysql.cnf

1. Edit testbed configuration **/usr/local/testbed/etc/mysql.cnf** and change the password which was set in step 7.
2. Populate testbed database.

   /usr/local/bin/tbd-manage migrate

3. Create admin account for testbed database not to be confused with the mysql admin account. This is a user that had full edit access in the testbed database. Run the following command and answer the prompts

   /usr/local/bin/tbd-manage migrate

4. Validate proper configuration **tbd db check** to confirm all checks pass.

Client Installation on Ubuntu 16.04

Here are the steps to setup testbed on a client running Ubuntu 14.04. Versions are currently available through github.com on https://github.com/testbed/testbed/releases. Please look through the release site to find the latest version. The example below uses an older version:

1. Install several packages:

   sudo apt-get install python-pip python-yaml libmysqlclient-dev python-dev

1. Install testbed from the github release area:

   sudo pip install https://github.com/testbed/testbed/archive/v0.1-alpha.8.tar.gz

      1. Edit the file testbed configuration file:

   /usr/local/testbed/etc/mysql.cnf

   Set host to the IP address of the testbed server. The user and password properties should also be changed appropriately.

1. Validate proper configuration. confirm all checks pass.

   tbd db check

CHAPTER 3

Development

Web Development

To simplify web development, developers can avoid using an actual hypervisor which can be involved. Instead developers can create profiles using a fake hypervisor.

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 profiles. The following creates two profiles. The first profile uses template0 to generate two fake VMs. The second profile creates three fake VMs from template1.:

```
./bin/tpl profile add localhost fake profile0 template0 2
./bin/tpl profile add localhost fake profile1 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/profile
```

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

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

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

Indices and tables

- genindex
- modindex
- search