ZStack 技术白皮书精选

自动化测试系统 2：系统测试

扫一扫二维码，获取更多技术干货吧

版权声明

摘要

大道至简·极速部署，ZStack 致力于产品化私有云和混合云。

ZStack 是新一代创新开源的云计算 IaaS 软件，由英特尔、微软、CloudStack 等世界上最早一批虚拟化工程师创建，拥有 KVM、Xen、Hyper-V 等成熟的技术背景。

ZStack 创新提出了云计算 4S 理念，即 Simple （简单）、Strong （健壮）、Smart （智能）、Scalable （弹性），通过全异步架构，无状态服务架构，无锁架构等核心技术，完美解决云计算执行效率低，系统不稳定，不能支撑高并发等问题，实现 HA 和轻量化管理。

ZStack 发起并维护着国内最大的自主开源 IaaS 社区——zstack.io，吸引了 6000 多名社区用户，对外公开的 API 超过 1000 个。基于这 1000 多个 API，用户可以自由组装出自己的私有云、混合云，甚至利用 ZStack 搭建公有云对外提供服务。

ZStack 拥有充足的知识产权储备，积极申报多项软著和专利，参与业内标准、白皮书的撰写，入选云计算行业方案目录，还通过了工信部云服务能力认证和信通院可信云认证。ZStack 面向企业用户提供基于 IaaS 的私有云和混合云，是业内唯一一家实现产品化，并领先业内首家推出同时打通数据面和控制面无缝混合云的云服务商。选择 ZStack，用户可以官网直接下载、1 台 PC 也可上云、30 分钟完成从裸机的安装部署。

目前已有 1000 多家企业用户选择了 ZStack 云平台。

ZSTACK——自动化测试系统 2：系统测试

ZStack 的系统测试系统在真实的硬件环境中运行测试用例；像集成测试一样，这个系统测试也是全自动的，而且覆盖的层面包括：功能性测试、压力测试、性能测试。

概述

虽然集成测试系统，如我们在 ZStack—自动化测试系统 1：集成测试中所介绍的，强大到可以暴露开发过程中大多数的缺陷，也是有着固有的弱点的。首先，由于测试用例使用模拟器，它们不能测试真实场景，比如在一个物理的 KVM 主机上创建一个 VM。第二，集成测试用例主要关注一个简单的场景，在一个简单的人造的环境中；举个例子，还是创建 VM 的这个用例，它可能只部署一个最小的环境，包括一个主机和一个 L3 网络，仅仅用于满足创建一个 VM 的需求。这些弱点，然而也是深思熟虑过的，因为我们想要开发人员能够在他们开发新特性时快速和容易地写测试用例，这是一个我们必须采取的权衡。

系统测试，目标在于测试整个软件，在一个真实的、复杂的环境中，很自然地补充集成测试。ZStack 的系统测试系统被设计用于以下两个目标：

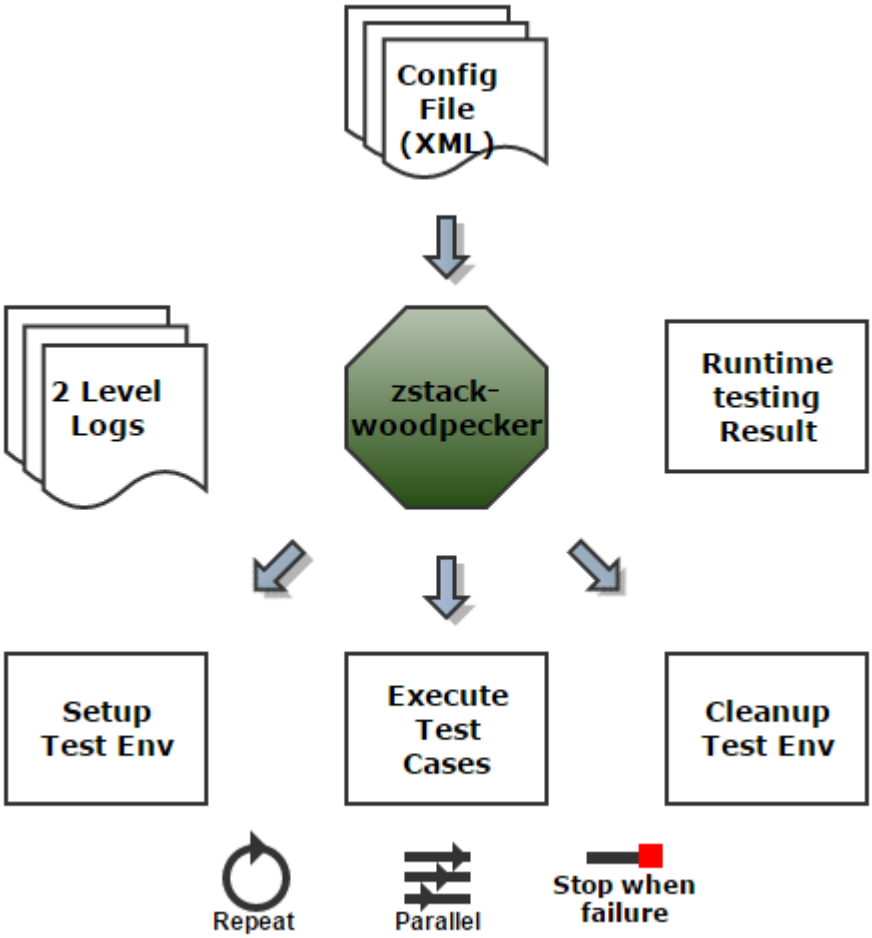
1. 复杂的场景：这些场景应该比真实世界的使用场景更复杂，以测试软件的极限。举个例子，挂载和卸载磁盘的测试用例应该持续地、重复地对虚拟机执行，以一种非常快，人类无法手动做到的方式。

2. 易于编写和维护测试用例：就像集成测试系统，系统测试系统接管了大多数无聊重复的任务，让测试人员有效率地写测试用例。

这个系统测试系统是一个 Python 项目，命名为 zstack-woodpecker，由以下三个部分组成：

1. 测试框架：一个测试框架，管理所有的测试用例，以及提供必须的库和工具。
2. 环境部署工具：一个工具，用于从 XML 配置文件部署一个环境；它非常类似于集成测试系统的部署器。
3. 模块化测试用例：测试用例是高度模块化的，而且覆盖了：功能测试、性能测试和压力测试。

系统测试



zstack-woodpecker 完全由我们自己创建；在决定重新造这个轮子之前，我们试过了流行的 Python 测试框架，像 nose，然后最终选择了创造一个新的工具，用以最大化地满足我们的目标。

套件配置

类似所有的其他测试框架，一个 zstack-woodpecker 中的测试套件是以 suite setup 开始，以 suite teardown 结束，在其中有一些

测试用例。这里的 suite setup 和 suite teardown 是两个特殊的测试用例，suite setup 负责准备后续的测试用例所需的环境，suite teardown 负责在所有测试用例结束之后清理这个环境。一个典型的测试套件配置文件看起来像：

```
<integrationTest>
    <suite name="basic test" setupCase="suite_setup.py" teardownCase="suite_teardown.py" parallel="8">
        <case timeout="120" repeat="10">test_create_vm.py</case>
        <case timeout="220">test_reboot_vm.py</case>
        <case timeout="200">test_add_volume.py</case>
        <case timeout="200">test_add_volume_reboot_vm.py</case>
        <case timeout="400">test_add_multi_volumes.py</case>
        <case timeout='600' repeat='2' noparallel='True'>resource/test_delete_l2.py</case>
    </suite>
</integrationTest>
```

敏锐的读者可能会注意到一些参数是在其他的测试框架中看不到的。

第一个是 timeout；每一个测试用例可以定义自己的超时时间，如果在这段时间内不能完成，它将被在最终的结果里被标记成超时。

第二个是 repeat，允许你在测试套件中指定这个用例应该被执行多少次。

第三个，也是杀手级的参数是 parallel，允许测试人员设定这个套件的并行级别；这是一个使得 zstack-woodpecker 运行测试用例非常快的关键特性；在上面这个例子中，parallel 被设置成 8，这意味着将有至多 8 个用例在同时运行；这不只是加速运行测试用例，也创造了一个复杂的场景，模拟许多用户在共享同一个环境时执行不同的任务。然而，不是所有的用例都可以被同时执行；在我们的例子中，用例 test_delete_l2.py 将会删除被其他用例依赖的

L2 网络，所以在其他用例执行时，它不能被执行；这就是第四个参数 noparallel 发挥作用的地方；一旦它被设置成 true，这个用例将会单独被执行，不会有其他用例可以同时运行。

命令行工具

zstest.py 是一个命令行工具，用于帮助测试人员控制测试框架，执行任务，像启动测试套件，列出测试用例，等等。zstest.py 提供了丰富的选项帮助测试人员简化他们的工作。这些选项中的一些，用于在我们的日常测试中，特别有用，列在了下面。

测试人员可以通过选项-l 获取可用的测试用例，例如：./zstest.py -l

它将会展示如下的结果：

```
[root@centos61 dailytest]# ./zstest.py -l -s basic

 - Available ZStack Integration Test Cases:

================================================================================
  No.   |       Test Case
--------------------------------------------------------------------------------
  [1]   | basic/suite_setup.py
  [2]   | basic/suite_teardown.py
  [3]   | basic/test_add_multi_volumes.py
  [4]   | basic/test_add_vol_to_stopvm.py
  [5]   | basic/test_add_volume_negative.py
--------------------------------------------------------------------------------
  [6]   | basic/test_add_volume.py
  [7]   | basic/test_add_volume_reboot_vm.py
  [8]   | basic/test_create_vm.py
  [9]   | basic/test_crt_temp_from_volume.py
  [10]  | basic/test_crt_vm_with_volume.py
--------------------------------------------------------------------------------
  [11]  | basic/test_reboot_vm.py
  [12]  | basic/test_start_vm.py
  [13]  | basic/test_stop_vm.py
  [14]  | basic/test_vm_securitygroup.py
  [15]  | basic/zstack_restart.py
================================================================================

 - List ZStack Integration Test Case Completed
```

测试套件名，是测试用例的第一级文件夹的名称；例如，在上图中你看到了大量的用例以 basic 开头（例如：basic/test_reboot_vm.py），是的，basic 就是这个测试套件的名字。测试人员可以通过选项-s 启动一个套件，使用套件名的全称或者部分都行，只要它是独一无二的，例如：./zstest.py -s basic 或 ./zstest.py -s ba

```
[root@centos61 dailytest]# ./zstest.py -s basic
export woodpecker_http_proxy=$http_proxy; export woodpecker_https_proxy=$https_p
roxy; unset http_proxy; unset https_proxy; zstack-woodpecker -f /root/zstack-woo
dpecker/dailytest/config_xml/test_suite.xml
Woodpecker is correctly installed. Test begins ...

[Begin suite parsing]

discovering test cases in /root/zstack-woodpecker/dailytest/config_xml/test_suit
e.xml ...
discovering test cases in /root/zstack-woodpecker/dailytest/config_xml/basic_int
egration.xml ...
        Suite [basic_test] will run [1] times:
                Run [1] times for Case: [basic/test_create_vm]
                Run [1] times for Case: [basic/test_stop_vm]
                Run [1] times for Case: [basic/test_start_vm]
                Run [1] times for Case: [basic/test_reboot_vm]
                Run [1] times for Case: [basic/test_add_volume]
                Run [1] times for Case: [basic/test_add_vol_to_stopvm]
                Run [1] times for Case: [basic/test_add_volume_reboot_vm]
                Run [1] times for Case: [basic/test_add_multi_volumes]
                Run [1] times for Case: [basic/test_add_volume_negative]
                Run [1] times for Case: [basic/test_crt_temp_from_volume]
                Run [1] times for Case: [basic/test_crt_vm_with_volume]
                Run [1] times for Case: [basic/test_vm_securitygroup]

[Suite parsing finished]: 1 test suites discovered, total 14 cases

[Test Begin]

Test suite: [basic_test], 14 cases, 2 execution threads, repeat 1 times:
suite_setup                    ..........................  [ success 0:03:19 ]
basic/test_create_vm           ..........................  [ success 0:00:13 ]
basic/test_stop_vm             ..........................  [ success 0:00:18 ]
basic/test_reboot_vm           .....                          [ 0:00:04 ]
```

测试人员也可以选择性地执行测试用例，通过使用它们的名字或者 ID，已经选项-c；例如：./zstest.py -c 1,6 或 ./zstest.py -c suite_setup,test_add_volume.py

记住，你需要运行 suite setup 的用例：suite_setup.py 作为第一个用例，除非你已经这么做了。

由于一个测试套件将会执行所有的测试用例，清理环境，发出一个结果报告，测试人员有时可能想要停止测试套件，并在一个用例失败时保持环境，这样他们就可以深入查看失败结果并调试；选项-n 和-S 就是为此准备的；-n 指示测试框架不要清理环境，-S 要求跳过没有被执行的用例；例如：./zstest.py -s virtualrouter -n -S

另外，选项-b 可以拉取最新的源代码并构建一个全新的 zstack.war，这在 Nightly 测试中特别有用，这种测试被假定为测试最新的代码：./zstest.py -s virtualrouter -b

一旦所有的测试用例完成，一个报告将会被生成并被打印到屏幕上：

```
Test Summary:
=============================================================================
 Test Case                                      Pass    Fail    TMO     Skip
-----------------------------------------------------------------------------
 basic_test:
    suite_setup                                 1       0       0       0
    basic/test_create_vm                        1       0       0       0
    basic/test_stop_vm                          1       0       0       0
    basic/test_start_vm                         1       0       0       0
    basic/test_reboot_vm                        1       0       0       0
    basic/test_add_volume                       1       0       0       0
    basic/test_add_vol_to_stopvm                1       0       0       0
    basic/test_add_volume_reboot_vm             1       0       0       0
    basic/test_add_multi_volumes                1       0       0       0
    basic/test_add_volume_negative              1       0       0       0
    basic/test_crt_temp_from_volume             1       0       0       0
    basic/test_crt_vm_with_volume               1       0       0       0
    basic/test_vm_securitygroup                 1       0       0       0
    suite_teardown                              1       0       0       0
-----------------------------------------------------------------------------
 Total: 14                                      14      0       0       0
=============================================================================

The detailed test results are in /root/zstack-woodpecker/dailytest/config_xml/te
st-result/151231-064358

Total test time: 00:06:06.849 (366.84996295)


 - ZStack Auto Test Completed
```

测试框架将会保存所有的日志，并直接输出每一个失败日志的绝对路径，如果存在的话。为了在一般的日志中记录更多的细节，有一种特殊的日志 action log，用于记录每一个 API 调用；因为这是一个完全纯粹关于 API 的日志，我们可以容易地找到一个失败的根本来源，而不用被测试框架的日志分散注意力。另外，它是一种重要的工具，可以自动地生成一个新的用例用于重现失败，这是一个我们所使用的魔法武器，用于在基于模型的测试（每个用例都随机地执行各种 API）中调试失败。你可以在 ZStack--自动化测试系统 3：基于模型的测试中找到细节。Action log 的片段如下：

```
   <<Action>> Create VM: test_vm_default_name with [image:] 07f7c252cdc14c4da5d3590f293b5852 and [l3_network:] ['6c7af862b03
44e32840e998aa5b49d52'] [14:56:28]

      <Log> [vm:] 2f5b5956418b46b2a017fcf5e7d31823 is created. [14:56:29]
      <Log> Add checker for [ZstackTestVm:] 2f5b5956418b46b2a017fcf5e7d31823. Checkers are: CheckerChain: [zstack_kvm_vm_set
_host_vlan_ip] [zstack_vm_db_checker] [zstack_kvm_vm_running_checker] [14:56:29]
      <Log> L2: 89e46a02c6c74570a32304a7a794edc2 did not have vlan.  [14:56:30]
      <Log> eth2 might be manually created vlan dev. [14:56:30]

     !!WARN!! L3 name: public l3 network uuid: 6c7af862b0344e32840e998aa5b49d52 network [mask:] 255.255.255.0 is not 255.255
.0.0 . Will not assign IP to host. Please change test configuration to make sure L3 network mask is 255.255.0.0. [14:56:30]
      <Log> Checker: [zstack_kvm_vm_set_host_vlan_ip] PASS. Expected result: True. Test result: True. [14:56:30]
      <Log> Checker: [zstack_vm_db_checker] begins. [14:56:30]
      <Log> Checker: [zstack_vm_db_checker] PASS. Expected result: True. Test result: True. [14:56:30]
      <Log> Checker: [zstack_kvm_vm_running_checker] begins. [14:56:30]
      <Log> Testagent is running on Host: 192.168.0.202 . Skip testagent installation. [14:56:30]
      <Log> L2: 89e46a02c6c74570a32304a7a794edc2 did not have vlan.  [14:56:30]
      <Log> eth2 might be manually created vlan dev. [14:56:30]

     !!WARN!! L3 name: public l3 network uuid: 6c7af862b0344e32840e998aa5b49d52 network [mask:] 255.255.255.0 is not 255.255
.0.0 . Will not assign IP to host. Please change test configuration to make sure L3 network mask is 255.255.0.0. [14:56:30]
      <Log> Check [vm:] 2f5b5956418b46b2a017fcf5e7d31823 running status on host [name:] 192.168.0.202 [uuid:] 546cf42092e249
428f0ebe65dbe0c625. [14:56:30]
      <Log> Check result: [vm:] 2f5b5956418b46b2a017fcf5e7d31823 is RUNNING on [host:] 192.168.0.202 . [14:56:30]
      <Log> Checker: [zstack_kvm_vm_running_checker] PASS. Expected result: True. Test result: True. [14:56:30]

   <<Action>> Create [Volume:] test_volume with [disk offering:] dbb8f3065efc463eaeac2bc0716d9234  [14:56:31]

      <Log> [volume:] 7db8d70d2eea44a8aec0235933cfedad is created. [14:56:31]

   <<Action>> Create [Volume:] test_volume with [disk offering:] dbb8f3065efc463eaeac2bc0716d9234  [14:56:31]

      <Log> [volume:] ada7968f3c7a4c0c90be47b23d0487d5 is created. [14:56:32]

   <<Action>> Create [Volume:] test_volume with [disk offering:] dbb8f3065efc463eaeac2bc0716d9234  [14:56:32]

      <Log> [volume:] 59e3d0e236ba49ce8d8ed0950d8449a7 is created. [14:56:32]

   <<Action>> Create [Volume:] test_volume with [disk offering:] dbb8f3065efc463eaeac2bc0716d9234  [14:56:33]

      <Log> [volume:] 11d2f2d9bf8c49c7ba5c744d79887b51 is created. [14:56:33]
```

环境部署工具

类似于集成测试，对每一个测试用例来说，准备环境是频繁且重复的任务；例如，用于测试创建虚拟机的用例需要去配置独立的资源，像 zone，cluster，host 等等。Zstack-woodpecker 调用 zstack-cli，这个 ZStack 的命令行工具去从一个 XML 配置文件部署测试环境。例如：zstack-cli -d zstack-env.xml

这里的 XML 配置文件的格式类似于集成测试所用的，一个片段看起来像这样：

```
...
  <zones>
    <zone name="$zoneName" description="Test">
      <clusters>
        <cluster name="$clusterName" description="Test"
          hypervisorType="$clusterHypervisorType">
          <hosts>
            <host name="$hostName" description="Test" managementIp="$hostIp"
              username="$hostUsername" password="$hostPassword" />
          </hosts>
          <primaryStorageRef>$nfsPrimaryStorageName</primaryStorageRef>
          <l2NetworkRef>$l2PublicNetworkName</l2NetworkRef>
          <l2NetworkRef>$l2ManagementNetworkName</l2NetworkRef>
          <l2NetworkRef>$l2NoVlanNetworkName1</l2NetworkRef>
          <l2NetworkRef>$l2NoVlanNetworkName2</l2NetworkRef>
          <l2NetworkRef>$l2VlanNetworkName1</l2NetworkRef>
          <l2NetworkRef>$l2VlanNetworkName2</l2NetworkRef>
        </cluster>
      </clusters>
...
      <l2Networks>
        <l2VlanNetwork name="$l2VlanNetworkName1" description="guest l2 vlan network"
          physicalInterface="$l2NetworkPhysicalInterface" vlan="$l2Vlan1">
          <l3Networks>
            <l3BasicNetwork name="$l3VlanNetworkName1" description = "guest test vlan network with DHCP DNS SNAT PortForwarding
 EIP and SecurityGroup" domain_name="$L3VlanNetworkDomainName1">
              <ipRange name="$vlanIpRangeName1" startIp="$vlanIpRangeStart1" endIp="$vlanIpRangeEnd1"
               gateway="$vlanIpRangeGateway1" netmask="$vlanIpRangeNetmask1"/>
              <dns>$DNSServer</dns>
              <networkService provider="VirtualRouter">
                <serviceType>DHCP</serviceType>
                <serviceType>DNS</serviceType>
                <serviceType>SNAT</serviceType>
                <serviceType>PortForwarding</serviceType>
                <serviceType>Eip</serviceType>
              </networkService>
              <networkService provider="SecurityGroup">
                <serviceType>SecurityGroup</serviceType>
              </networkService>
            </l3BasicNetwork>
          </l3Networks>
        </l2VlanNetwork>
...
```

部署工具通常在运行任何用例前被 suite setup 调用，测试人员可以在 XML 配置文件中通过以$符号开头来定义变量，然后在一个独立的配置文件中解析。通过这种方式，这个 XML 配置文件像模板一个工作，可以产生不同的环境。配置文件的例子如下：

```
TEST_ROOT=/usr/local/zstack/root/
zstackPath = $TEST_ROOT/sanitytest/zstack.war
apachePath = $TEST_ROOT/apache-tomcat
zstackPropertiesPath = $TEST_ROOT/sanitytest/conf/zstack.properties
zstackTestAgentPkgPath = $TEST_ROOT/sanitytest/zstacktestagent.tar.gz
masterName = 192.168.0.201
DBUserName = root

node2Name = centos5
node2Ip = 192.168.0.209
node2UserName = root
node2Password = password

node1Name = 192.168.0.201
node1Ip = 192.168.0.201
node1UserName = root
node1Password = password


instanceOfferingName_s = small-vm
instanceOfferingMemory_s = 128M
instanceOfferingCpuNum_s = 1
instanceOfferingCpuSpeed_s = 512

virtualRouterOfferingName_s = virtual-router-vm
virtualRouterOfferingMemory_s = 512M
virtualRouterOfferingCpuNum_s = 2
virtualRouterOfferingCpuSpeed_s = 512

sftpBackupStorageName = sftp
sftpBackupStorageUrl = /export/backupStorage/sftp/
sftpBackupStorageUsername = root
sftpBackupStoragePassword = password
sftpBackupStorageHostname = 192.168.0.220
```

注意:正如你可能会猜测的，这个工具可以被管理员用于从一个 XML 配置文件部署一个云环境；更进一步，管理员们做相反的事情，将一个云环境写入到一个 XML 配置文件，通过 zstack-cli -D xml-file-name.

对于性能和压力测试，环境通常需要大量的资源，例如 100 个 zone，1000 个 cluster。为了避免手动在配置文件中重复 1000 行，我们引入了一个属性 duplication，用于帮助创建重复的资源。例如：

```
...
    <zones>
      <zone name="$zoneName" description="10 same zones" duplication="100">
        <clusters>
          <cluster name="$clusterName_sim" description="10 same Simulator Clusters" duplication="10"
            hypervisorType="$clusterSimHypervisorType">
            <hosts>
              <host name="$hostName_sim" description="100 same simulator Test Host"
                managementIp="$hostIp_sim"
                cpuCapacity="$cpuCapacity" memoryCapacity="$memoryCapacity"
                duplication="100"/>
            </hosts>
            <primaryStorageRef>$simulatorPrimaryStorageName</primaryStorageRef>
            <l2NetworkRef>$l2PublicNetworkName</l2NetworkRef>
            <l2NetworkRef>$l2ManagementNetworkName</l2NetworkRef>
            <l2NetworkRef>$l2VlanNetworkName1</l2NetworkRef>
          </cluster>
...
```

备注：这段不翻译了。

模块化的测试用例

在系统测试中测试用例可以被高度模块化。每一个用例本质上执行以下三步：

1. 创建要被测试的资源

2. 验证结果

3. 清理环境

Zstack-woodpecker 本身提供一个完整的库用于帮助测试人员调度这些活动。API 也很好地被封装在一个，从 zstack 源代码自动生成的库中。测试人员不需要去写任何的原生的 API 调用。检查器，用于验证测试结果，也已为每一个资源创建；例如，VM 检查器，云盘检查器。测试人员可以很容易地调用这些检查器去验证他们创建的资源，而不需写成顿成吨的代码。如果当前检查器不能满足某些场景，测试人员也能创建自己的检查器，并作为插件放入测试框架。一段测试用例看起来像：

```
def test():
    test_util.test_dsc('Create test vm and check')
    vm = test_stub.create_vlan_vm()
    test_util.test_dsc('Create volume and check')
    disk_offering = test_lib.lib_get_disk_offering_by_name(os.environ.get('rootDiskOfferingName'))
    volume_creation_option = test_util.VolumeOption()
    volume_creation_option.set_disk_offering_uuid(disk_offering.uuid)
    volume = test_stub.create_volume(volume_creation_option)
    volume.check()
    vm.check()
    test_util.test_dsc('Attach volume and check')
    volume.attach(vm)
    volume.check()
    test_util.test_dsc('Detach volume and check')
    volume.detach()
    volume.check()
    test_util.test_dsc('Delete volume and check')
    volume.delete()
    volume.check()
    vm.destroy()
    vm.check()
    test_util.test_pass('Create Data Volume for VM Test Success')
```

像集成测试一样，测试人员可以仅以十几行便写出一个测试用例。模块化不只是帮助简化测试用例的编写，也为基于模型的测试构建了一个坚实的基础，下篇文章我们会详细讨论。

总结

在这篇文章中，我们引入了我们的系统测试系统。通过执行比现实世界的用例更复杂的测试，系统测试可以给我们更多的自信，关于 ZStack 在真实的硬件环境中的表现。使得我们可以快速进化成一个成熟的产品。