ZStack 技术白皮书精选

高效开发测试打造产品化私有云

扫一扫二维码，获取更多技术干货吧

高效开发测试打造产品化私有云

摘要:做产品化私有云，ZStack 是认真的。ZStack 测试架构师带你一起领略产品化背后高效开发测试的秘密

作者：许佳珺

前言

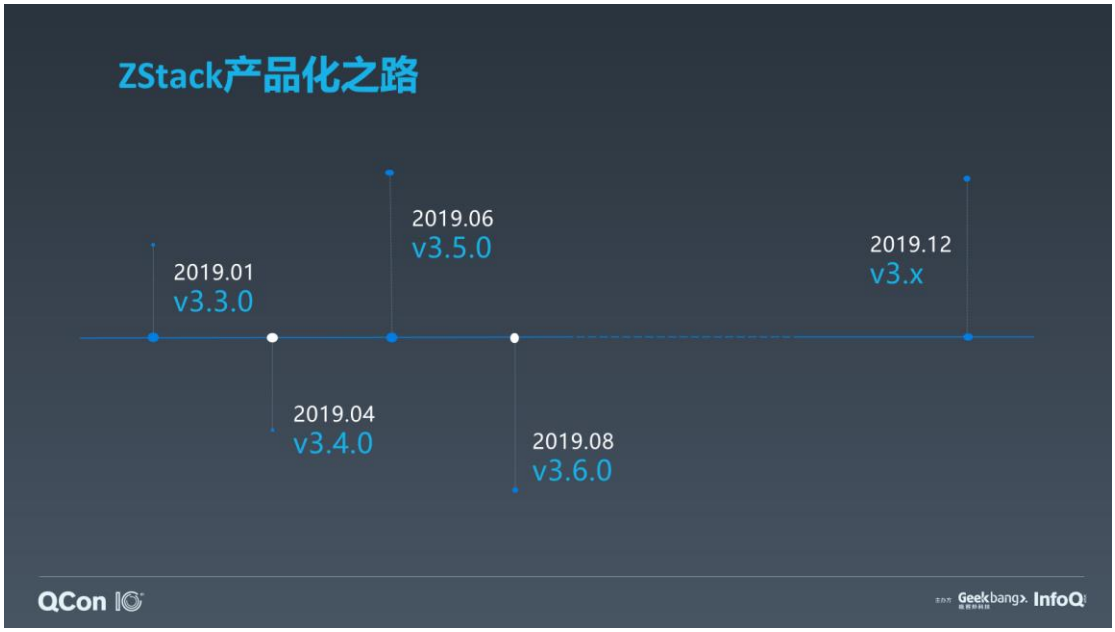
随着越来越多的企业将云计算产品应用到基础设施及其核心业务中，如何提高和保证软件交付质量、减少软件开发迭代周期、加速软件发布频率成为所有云厂商面临的关键问题。

根据 IDC 2018 年的预测，中国云计算市场在未来 5 年将持续高速发展的态势，主要表现为：中国传统的非云计算 IT 基础架构占整体 IT 基础架构的投入比例将从 2018 年的 50.3%下降到 2022 年的 40.7%；中国私有云平台建设的市场规模将以年均 24.8%的复合增长率快速增长；中国云计算 IT 基础架构支出占全球市场比将从 2018 年的 12%上升到 2022 年的 25%，届时中国私有云 IT 基础架构支出将超过美国，成为全球第一大市场。在这一轮新的迭代更新中，更多的企业和行业开始部署或者建立更大规模的私有云；而新应用（数据分析，AI，IoT，移动）和新场景（边缘计算，智慧/平安城市，行业云）也对云平台提出了更高的

需求。

ZStack 凭借创新的产品化理念，在业内率先提出云计算的 4S 标准 – 简单 Simple，健壮 Strong，弹性 Scalable，智能 Smart。同时，ZStack 企业版从第一版发布到最新的 3.5.0 版本，一直以每六周一次的周期迭代更新软件版本，快速提升和扩展产品功能，积极应对云计算市场对私有云产品不断增长的需求。而保证其私有云产品化的关键要素有以下三点：

1. 流程 – 快速敏捷
2. 运维 – 智能高效
3. 测试 – 严谨全面

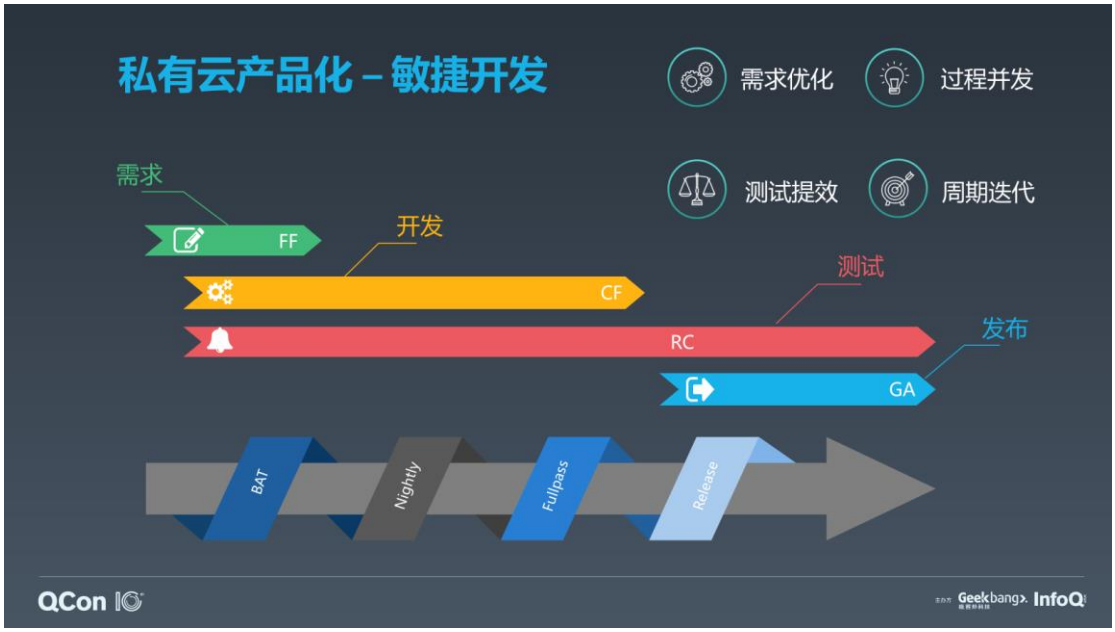


注：ZStack 坚持快速、简洁、高效的开发、运维、测试流程，确保新需求六周便可实现

1. 流程 – 快速敏捷

ZStack 开发流程依然定义了传统开发模式中的几个关键阶段 - FF、CF、RC 和 GA。同

时针对不同阶段的任务和目标，进行有的放矢地优化。在 Feature Freeze 阶段，主要以需求分析为主，要求产品经理将客户的需求分片化、分级化，需求描述本地化，更有效地将需求安排到不同发布版本周期中。开发和测试工程师则需要将 Code Freeze 和 Release Candidate 的任务提前到 Feature Freeze 阶段中，减少互相之间任务的依赖，提高各个阶段的并发度。而测试不仅需要渗透到开发的每个环节中，同时也要通过模型测试、路径测试、稳定性测试等方法，提高代码的覆盖度和测试效率。每个发布周期通过反复地从需求->开发->测试的快速迭代，保证了产品的新需求和问题始终能够被快速满足和解决。

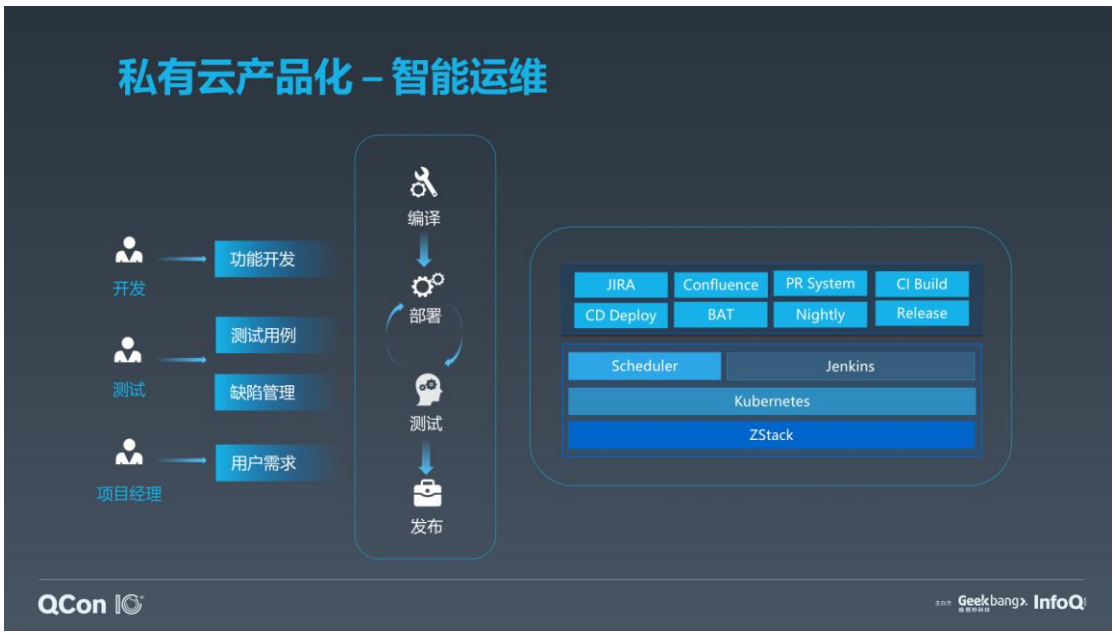


注：ZStack 产品开发流程高度并发，保证版本之间快速迭代

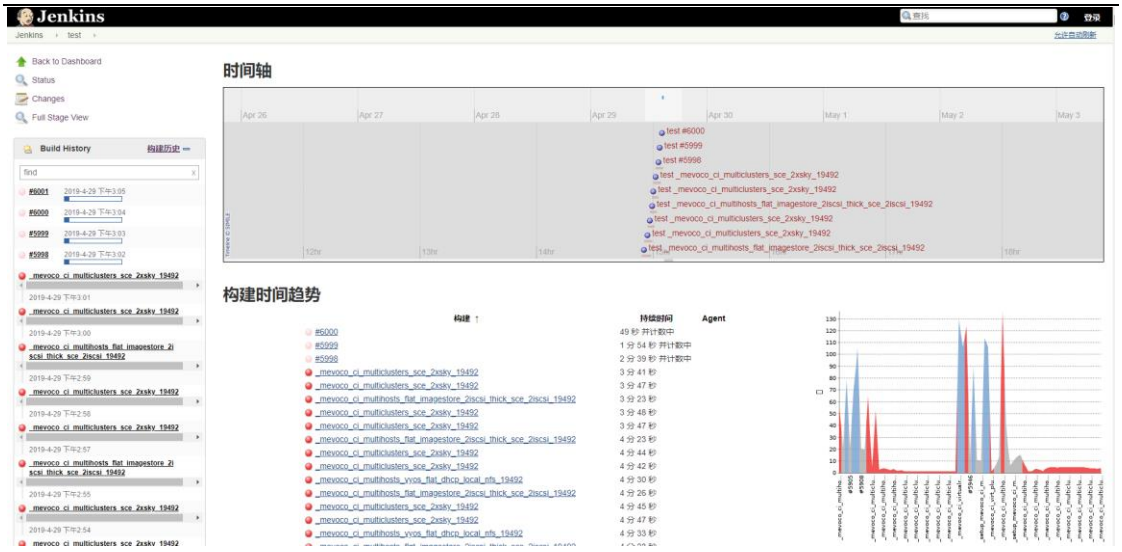
2. 运维 – 智能高效

作为私有云产品开发的基础保证，一套快速、稳定、高并发、可伸缩的运维系统是必要的。而传统运维提供的简单 CI 和 CD 功能是显然无法满足这样快速迭代的需求。ZStack 产品化过程中，搭建了一套以 ZStack + Kubernetes 为基础、面向公司各个部门的整体性服务

框架。这套框架中所包括的服务内容涵盖从开发&测试人员使用的测试环境、到整个项目的管理工具。框架的底层以 ZStack 作为 IaaS 提供给上层可靠的、可扩展的物理资源，同时结合 Kubernetes，将容器运行于云主机中，既保证了隔离性、又充分利用了 ZStack 和 Kubernetes 对云主机和 Docker 调度的优势，起到了对上层服务高可用、高并发及可伸缩的双重保障。



注：ZStack 作为 IaaS 层向上层服务提供可靠的物理资源，而更重要的是，内部的 ZStack 环境也会随着发布版本更新，真正做到了自己的产品自己先用起来



注：实际生产环境中，一次自动化测试至少在 Jenkins 上并发创建 500+个请求，每个请求包含 10~50 个测试用例，ZStack + Kubernetes 保证了这些请求几秒内可以被处理

3. 测试 – 严谨全面

打造一个产品化的私有云软件需要全面且严谨的测试，这不仅仅是单元测试和集成测试能保证的。ZStack 从以下四个方面入手强化测试：

3.1 测试高效化：整个产品流程中开发和测试要同步进行，这包括了对不同的开发分支需要有不同深度的测试代码保证其质量——例如，对于 Release 分支，必须有持续性的 Nightly 测试把控每天进入的代码质量；对于 Feature 分支，需要能快速检测出 patch 对代码核心功能影响的 BAT 测试。同时测试系统和 CI 系统要高度集成并且做到同步触发。

高效化的另一个重点就是要做到所有测试都能运行在云端，提高测试的并发度和资源利用率。ZStack 内部的测试都是跑在云端的，而云端环境也是基于 ZStack 自身搭建的，利用

其对底层硬件资源的抽象和管理，模拟出测试中需要的不同的硬件配置场景，包括网络、存储、虚拟化平台、甚至不同的 ZStack 高级功能配置，如企业管理、灾备服务等。同时，为了满足大规模资源需求的测试场景，例如 1 万台或 10 万台云主机的测试场景，ZStack 测试中还实现了 simulator 机制，即不真实分配硬件资源，而使用 mock 后端 API 的方式提供了对后端资源的调配，真正做到了有针对性的测试。

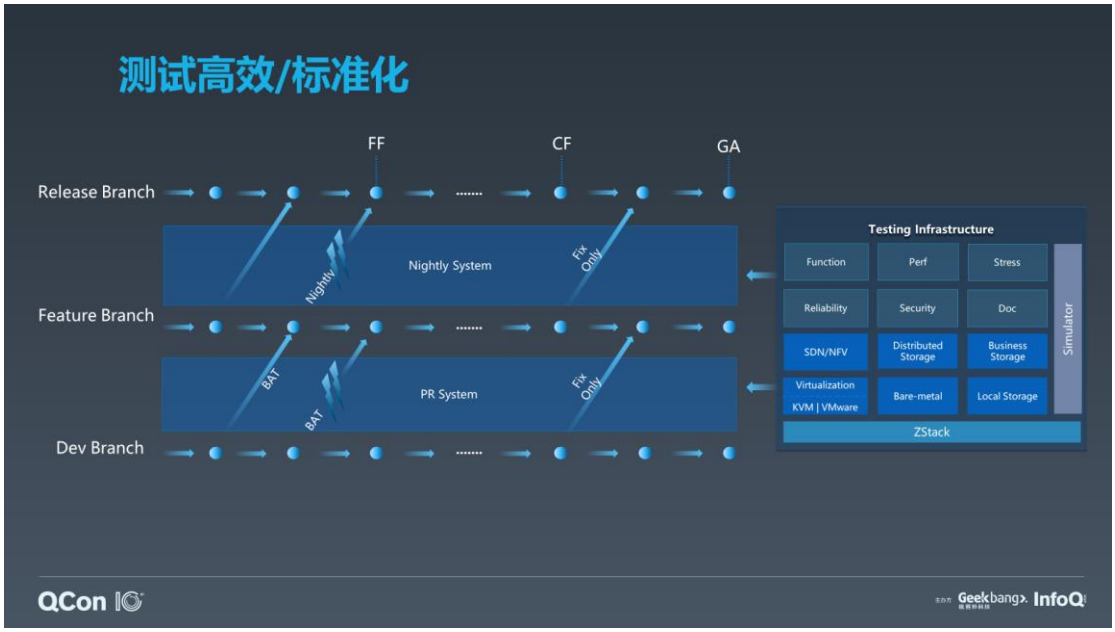
```
<zones>
  <zone name="$zoneName" description="Test">
    <backupStorageRef>$cephBackupStorageName</backupStorageRef>
    <backupStorageRef>$cephBackupStorageName2</backupStorageRef>
    <primaryStorages>
      <cephPrimaryMultipoolsStorage name="$cephPrimaryStorageName"
        description="Test Ceph PS"
        monUrls="$cephPrimaryStorageMonUrls"
        poolPrefix="test-pool"
        poolNum="2"
      />
      <cephPrimaryMultipoolsStorage name="$cephPrimaryStorageName2"
        description="Test Ceph PS2"
        monUrls="$cephPrimaryStorageMonUrls2"
        poolPrefix="test-pool"
        poolNum="2"
      />
    </primaryStorages>

    <clusters>
      <cluster name="$clusterName" description="Test"
        hypervisorType="$clusterHypervisorType">
        <hosts>
          <host name="$hostName" description="Test" managementIp="$hostIp"
            username="$hostUsername" password="$hostPassword" />
          <host name="$hostName2" description="Test2" managementIp="$hostIp2"
            username="$hostUsername2" password="$hostPassword2" port="$hostPort2"/>
        </hosts>
        <primaryStorageRef>$cephPrimaryStorageName</primaryStorageRef>
        <l2NetworkRef>$l2PublicNetworkName</l2NetworkRef>
        <l2NetworkRef>$l2NoVlanNetworkName1</l2NetworkRef>
        <l2NetworkRef>$l2NoVlanNetworkName2</l2NetworkRef>
        <l2NetworkRef>$l2VlanNetworkName1</l2NetworkRef>
        <l2NetworkRef>$l2VlanNetworkName2</l2NetworkRef>
```

注：ZStack 云端测试的环境构建是通过 XML 配置文件实现的，测试工程师可以非常简单地用几分钟配置出一台自动化环境

3.2 测试标准化：ZStack 所涵盖的测试内容不仅包括功能性测试，还包括一套完整测试体系所需要的各种测试，如开发工程师需要做的集成/单元测试，测试工程师需要做的系统测试中的压力、性能、可靠性测试、以及针对不同版本定制的发布测试。

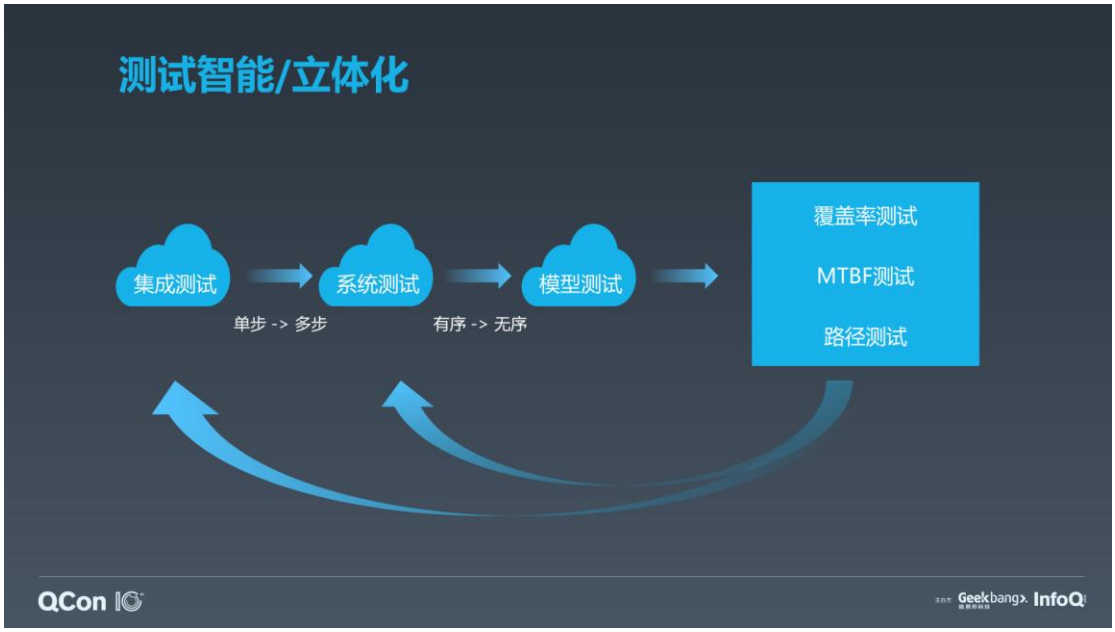
例如 ZStack 的可靠性测试就包括了两类测试 – MTBF 和 DPMO 测试，MTBF 会对 ZStack 平台进行 15,000 小时长时间的真实用户操作模拟；DPMO 测试则会对 ZStack 平台进行高达 10,000 次的断/上电、重启等测试。

标准化的另一方面体现在对关键节点的标准把控上，对 FF、CF、RC 和 GA 各个阶段都会有相应的代码准入和验收标准，例如 CF 阶段后功能开发代码禁止进入发布分支而只能进入下一个发布版本的周期；又例如各个阶段验收时要求的 bug 数量限制，CF 阶段要求小于 5 个 P0，GA 阶段要求没有 P0 的 bug。



3.3 测试覆盖智能化：软件测试没法达到 100%的覆盖率，所以我们要做的是在资源有限的情况下，以尽量少的代价做到尽可能高的覆盖率。要提高覆盖率，需从两方面入手，一方面是对代码进行覆盖率检查，我们在日常 CI 的包中插入了代码不同模块的覆盖率，不管是手动还是自动测试，或是日常 bug 的验证，都会为覆盖率提供数据。

另一方面我们增加了模型测试，它可以产生由随机 API 组合构成的场景，会持续运行直到遇到预定义的退出条件或者找到一个缺陷。这种模型测试很好地弥补了人为定义用例的不足，提高了测试场景和路径的覆盖率。由这种测试模型，也衍生出了三种不同场景的覆盖率提高测试：



3.3.1 覆盖率测试：除常规有序的测试步骤外，运用模型测试，收集无序测试步骤下的测试覆盖率。

3.3.2 MTBF 测试： 从有序和无序两种测试维度，对系统稳定性及可靠性进行测试。

3.3.3 路径测试： 通常一个系统测试用例最多 5~6 个操作步骤，而最终客户的问题场景是极其复杂的，通常需要 10~20 个以上的步骤才能重现，运用模型测试的方法，可以有效减少构建测试用例的代码量。

```
import zstackwoodpecker.test_state as ts_header
import os
TestAction = ts_header.TestAction
def path():

    return dict(initial_formation="template5", path_list=[
        [TestAction.create_mini_vm, "vm1", 'data_volume=false', 'cpu=2', 'memory=2', 'provisiong=thin'],
        [TestAction.create_volume, "volume1", "=scsi,thin"],
        [TestAction.attach_volume, "vm1", "volume1"],
        [TestAction.create_volume_backup, "volume1", "backup1"],
        [TestAction.create_mini_vm, "vm2", 'data_volume=false', 'cpu=2', 'memory=random', 'provisiong=thin'],
        [TestAction.detach_volume, "volume1"],
        [TestAction.create_mini_vm, "vm3", 'data_volume=false', 'cpu=2', 'memory=2', 'network=random', 'provisiong=thin'],
        [TestAction.delete_volume, "volume1"],
        [TestAction.add_image, "image1", 'root', "http://172.20.1.28/mirror/diskimages/centos_vdbench.qcow2"],
        [TestAction.delete_volume_backup, "backup1"],
        [TestAction.delete_image, "image1"],
        [TestAction.expunge_image, "image1"],
        [TestAction.create_volume, "volume2", "=scsi,thin"],
        [TestAction.attach_volume, "vm2", "volume2"],
        [TestAction.create_volume_backup, "volume2", "backup2"],
        [TestAction.reboot_vm, "vm1"],
        [TestAction.reboot_vm, "vm2"],
        [TestAction.reboot_vm, "vm3"],
        [TestAction.create_volume, "volume3", "=scsi,thin"],
        [TestAction.delete_volume, "volume3"],
        [TestAction.delete_vm, "vm3"],
        [TestAction.expunge_vm, "vm3"],
        [TestAction.add_image, "image2", 'root', "http://172.20.1.28/mirror/diskimages/centos_vdbench.qcow2"],
        [TestAction.expunge_volume, "volume1"],
        [TestAction.delete_image, "image2"],
        [TestAction.recover_image, "image2"],
        [TestAction.delete_volume_backup, "backup2"],
        [TestAction.add_image, "image3", 'root', "http://172.20.1.28/mirror/diskimages/centos_vdbench.qcow2"],
        [TestAction.recover_volume, "volume3"],
        [TestAction.change_vm_ha, 'vm1'],
        [TestAction.create_volume, "volume4", "=scsi,thin"],
        [TestAction.attach_volume, "vm2", "volume4"]])
```

注：一个典型路径测试，只需要将测试对象和操作步骤写到测试用例中即可完成

3.4 报告立体化： 主要从两方面实现，一是测试报告的结果自动化、可读化，是通过对测试用例中插入 DITA 描述实现的。另一方面是结果的可追溯和可回放，这是通过记录测试过程中 API 的调用顺序和参数实现的。

```
[root@172-24-252-242 zstack-woodpecker]# cat robot_action_log
Robot Action: create_vm
Robot Action Result: create_vm; new VM: fc2c0221be72423ea303a522fd6570e9
Rotob Action: stop_vm: on VM: fc2c0221be72423ea303a522fd6570e9
Robot Action: create_volume_snapshot; on Root Volume: fe839dcb305f471a852a1f5e21d4feda; on VM: fc2c0221be72423ea303a522fd6570e9
Robot Action Result: create_volume_snapshot; new SP: 497ac6abaf984f5a825ae4fb2c585a88
Robot Action: create_data_volume_template_from_volume; on Volume: fe839dcb305f471a852a1f5e21d4feda; on VM: fc2c0221be72423ea303a522fd6570e9
Robot Action Result: create_data_volume_template_from_volume; new DataVolume Image: fb23cdfce4b54072847a3cfe8ae45d35
Robot Action: destroy_vm; on VM: fc2c0221be72423ea303a522fd6570e9
Robot Action: create_data_volume_from_image; on Image: fb23cdfce4b54072847a3cfe8ae45d35
Robot Action Result: create_data_volume_from_image; new Volume: 20dee895d68b428a88e5ec3d3ef634d8
Robot Action: create_volume_snapshot; on Volume; 20dee895d68b428a88e5ec3d3ef634d8
[root@172-24-252-242 zstack-woodpecker]# ./zstackwoodpecker/zstackwoodpecker/robot_replay.py robot_action_log
```

注：一个测试结果的操作记录及回放方法，能够有效帮助开发测试工程师重现 bug

总结

作为产品化的云计算公司，ZStack 一直致力于打造自研的 ZStack 私有云、ZStack 混合云、ZStack Mini 超融合一体机、ZStack CMP 多云管理平台、ZStack 企业级分布式存储等产品和方案。本文从开发流程、基础运维以及测试能效等角度，介绍了 ZStack 团队如何高效打造一个产品化的私有云。