****

**云原生测试白皮书**

**（2021年）**

**云原生产业联盟**

**Cloud Native Industry Alliance，CNIA**

**2021年5月**

版权声明

**本白皮书版权属于云原生产业联盟，并受法律保护。转载、摘编或利用其它方式使用本白皮书文字或者观点的，应注明“来源：云原生产业联盟”。违反上述声明者，本联盟将追究其相关法律责任。**

**编写说明**

**牵头编写单位：**中国信息通信研究院

**参与编写单位：**阿里云计算有限公司 杭州端点网络科技有限公司 中移（杭州）信息技术有限公司 中移（苏州）软件技术有限公司

**编写组成员：**

中国信息通信研究院：

阿里云计算有限公司：

杭州端点网络科技有限公司：

中移（杭州）信息技术有限公司：

中移（苏州）软件技术有限公司：

**注：**编写单位按首字母顺序排列

## 前 言

软件测试从20世纪80年代发展至今，包含的内容越来越多、工种划分越来越细，现已覆盖功能复核验证、自动化测试执行、环境部署测试、性能测试、故障注入测试、兼容性测试、安全测试等多个领域。

本白皮书对云原生时代的软件测试进行了全面分析，重点介绍了云原生时代软件测试面临的变化、机遇与挑战。

**目 录**

[一、 软件测试发展历程 1](#_Toc71364689)

[(一) 传统时代的测试 1](#_Toc71364690)

[(二) 互联网时代的测试 1](#_Toc71364691)

[(三) 移动互联网时代的测试 2](#_Toc71364692)

[(四) 云原生时代的测试 3](#_Toc71364693)

[二、 云原生测试概述 4](#_Toc71364694)

[三、 云原生给测试领域带来的红利 8](#_Toc71364695)

[(一) 测试环境部署更加便捷 8](#_Toc71364696)

[1. 资源获取成本降低 8](#_Toc71364697)

[2. 环境即用即拉 8](#_Toc71364698)

[3. 测试服务快速弹性扩展 9](#_Toc71364699)

[4. 测试环境即弃即抛 9](#_Toc71364700)

[(二) 测试成本大幅降低 10](#_Toc71364701)

[1. 人力成本大幅下降 10](#_Toc71364702)

[2. 测试效率大幅提升 10](#_Toc71364703)

[3. 测试内容更加聚焦于应用本身 10](#_Toc71364704)

[四、 云原生给测试领域带来的挑战 11](#_Toc71364705)

[五、 云原生测试关键要素 12](#_Toc71364706)

[(一) 持续测试 12](#_Toc71364707)

[(二) 面向可恢复的测试 13](#_Toc71364708)

[(三) 应用运维的测试 17](#_Toc71364709)

[1. 可扩展性 17](#_Toc71364710)

[2. 弹性 18](#_Toc71364711)

[3. 可观测性 19](#_Toc71364712)

[4. 一致性 20](#_Toc71364713)

[5. 健壮性 20](#_Toc71364714)

[6. 可靠性 21](#_Toc71364715)

[7. 安全性 21](#_Toc71364716)

[8. 自动预案验证 22](#_Toc71364717)

[(四) 测试服务化 22](#_Toc71364718)

[1． E2E测试 23](#_Toc71364719)

[2． 接口测试 23](#_Toc71364720)

[3． 性能测试 23](#_Toc71364721)

[4． 移动端测试 24](#_Toc71364722)

[5． 测试数据管理 25](#_Toc71364723)

[(五) 数据驱动测试 26](#_Toc71364724)

[(六) 测试智能化 28](#_Toc71364725)

[六、 云原生测试发展趋势 34](#_Toc71364726)

[(一) 云原生时代测试角色的变化 34](#_Toc71364727)

[(二) 云原生测试对技术要求越来越强 35](#_Toc71364728)

[(三) 云原生测试平台持续演进 35](#_Toc71364729)

[附录 云原生测试最佳实践案例 36](#_Toc71364730)

[(一) \*\*\*\*\* 36](#_Toc71364731)

[(二) \*\*\*\*\* 36](#_Toc71364732)

[(三) \*\*\*\*\* 36](#_Toc71364733)

# 一、 软件测试发展历程

## (一) 传统时代的测试

测试行业从诞生之日起，就围绕质量管控制定了大量的方法、标准。比如ISO9000质量系列标准，CMMI软件能力成熟度模型集成，六西格玛（Six Sigma）管理法等。而测试的分类也细分出了自动化测试、安全测试、性能测试、兼容性测试等类型，此时的测试验证，更像是一个验收环节，所有的测试活动目的都是为产品的质量作保证，避免经过测试验收后仍然有漏洞或缺陷出现在客户现场。传统时代，研发测试比、测试遗漏率、产品迭代周期等指标，就是用来考察测试团队的质量和效率的。传统时代的软件研发注重流程保障，通常采用瀑布模型来进行软件研发。在瀑布模型的约束下，需求分析、产品设计、研发、测试分工明确，各司其职，每个环节都有相应的交付物作为下一环节的输入，最终通过测试验收，发版交付。整个过程中，测试验收的速度是缓慢的。测试人员除了要做浏览器的自动化，还要面对MFC、Delphi、Java Swing等图形界面，这些自动化测试工具的维护成本也非常高，很难用一个测试软件解决行业内大部分问题。在传统时代，整个软件行业束缚在瀑布模型中的测试环境，没有突破性进展。

## (二) 互联网时代的测试

随着互联网技术的兴起，中心化、服务化的软件形态渐渐成为主流，互联网公司的测试人员成为了第一批受益者，由于网站随时处于运维人员的管控中，因此发布的版本存在缺陷时，运维人员可以快速地通过在线回滚、在线隔离、甚至是在线修复来解决问题，在线模式大大提升了生产效率。一夜之间，敏捷研发模式颠覆了传统的瀑布模式，成为主流。通过去掉需求分析角色、弱化研发和测试角色、简化流程等操作，大大加快了产品的迭代速度。在这个时代，企业纷纷设立“测试开发工程师”，“测试也要具备研发能力”被更多公司所推崇，“TDD——测试驱动研发”一词也被广泛使用。基于敏捷研发的流程、持续集成（Continuous Integration, CI）逐渐成为一种标准，软件的版本管理不再成为一个强制条件，基于主干的持续发布能力成为了验证产品成熟度的标准。由于浏览器成为主流，这个时期的测试领域工作范围不再局限于瀑布模式的测试模块，并且测试人员有了很强的研发能力。测试领域进入了一个百花齐放的时代，Web自动化、A/B测试、敏捷测试、Web性能测试、浏览器兼容性测试、SOAP、硬件兼容性、持续集成等多个概念被提出，这些技术和工具的通用性很强，通过开源的方式让测试技术的门槛大大降低，更多的企业在测试效率方面大大提升，很多传统企业甚至是工厂都开始拥抱互联网，以降低信息获取的成本、提升生产效率。

## (三) 移动互联网时代的测试

在互联网时代，很多测试工作、发布工作是在凌晨进行的，因为此时用电脑浏览网站的用户较少，一旦发布失败造成了用户资损，也可以把损失控制在一个很小的规模。而这看似万无一失的操作，被移动互联网打破了。随着智能手机的普及，浏览器不再是网站的唯一入口，手机APP同样会成为流量的大户。手机的便携性导致流量不可预测，随时发生的一个热点事件都可以造成流量的洪峰。首先，移动互联网时代对于网站的发布即时性提出了更高要求；同时，多款智能手机造成的碎片化，为测试验证带来了非常大的挑战。尽管如此，这个时代测试人员可操作的范围增大了，很多基于流量、手机移动端、弹性的测试工具和服务应运而生，例如流量隔离、流量复制、移动端测试框架、移动端兼容性测试等技术。在持续集成的基础之上，CI/CD（Continuous Integration/Continuous Delivery, 持续集成持续交付）的概念在此时被提出。CI/CD目的是让产品每时每刻都能处在发布态，进一步提升迭代效率。

## (四) 云原生时代的测试

随着云厂商IaaS的大力发展，自建IDC、租用服务器已经不是公司的最佳选择。在云厂商收敛底层硬件的同时，PaaS层面的革新到来了，容器化、微服务、Service Mesh等技术进一步降低了企业业务研发的成本，也同时为测试人员带来了红利。在云原生时代以前，虽然测试人员可操作的范围增大、测试程序转向服务化，但测试领域的一些通用问题并没有得到很好的解决，比如环境隔离的复杂度、获取流量的成本、构建一个新环境的复杂度等问题。随着云原生的出现，这些问题都有了相应的解决方案。诸如测试环境的隔离轻量化、测试服务化、混沌工程、大数据驱动测试、人工智能测试、基于SLA的测试、基于范式的缺陷定位等新的测试技术被广泛应用，给测试领域带来了很大红利，使测试人员能够更加聚焦于被测应用自身，测试验证活动更加频繁、稳定。

每个时代都对应了许多测试技术，无论是旧技术还是新技术，测试活动都在朝着“高频、隔离、用完即抛”的目的在努力。通过更高频、更稳定的验证方式，为产品的高频迭代提供有力保障。

# 二、 云原生测试概述

测试行业发展至今，一直在围绕质量和效率这两个领域展开。如何把质量验收的工作尽可能地前置执行、随时执行、自动化执行，成为了测试人员的极值追求。在传统的瀑布模式中，测试被定义为一个验收与验证的环节，很难对上游或下游的活动产生影响，此时的测试自动化就像工业革命1.0的机械化一样，提升了测试验证的效率，但仅此而已。随着互联网技术的兴起，测试行业得到了很大的发展，测试角色由“用例验收”逐步向“加速交付”转型，测试工具的平台化、服务化使测试活动更为灵活，可以把测试活动“左移”至研发过程中来完成，测试活动贯穿整个软件生命周期，提升了产品迭代的效率。一时间，产品是否具备“测试左移”的能力，成为了衡量产品成熟度的指标之一。而随着互联网技术的进一步发展，测试活动也开始“右移”至发布后的生产环境，更多的测试工具在线上展开，灰度发布、流量管控、流量复制、端到端巡检等多项技术诞生，标志着测试活动正式覆盖了产品生命周期的全领域。

在云原生时代，产品技术架构被进一步整合，更多的技术红利被释放出来，诸如环境轻量级部署、环境隔离、流量管控等“高阶”技术的门槛会被降低，使混沌工程和人工智能测试（AI测试）成为可能。云原生技术让应用能够构建、运行在公有云、私有云及混合云等新型动态环境中，具备应用构建容错性好、易于管理、便于观察等特性。基于这些云原生基本特性，测试服务平台无需在测试服务的部署、维护和扩展等方面花费大量精力，以低成本轻松实现环境隔离、弹性伸缩等。依托云原生基础能力，以微服务的形式提供给开发和测试者直接使用，或通过声明式API给DevOps平台或者其他平台提供调用测试服务能力，由此使测试平台可以更专注于测试服务能力体系的建设。如图1所示，测试服务其他包含了基于测试服务框架和测试服务引擎提供各种测试服务能力的测试服务层，也包含了测试数据治理、分析的数据服务层，以及针对测试各维度的分析度量、监控告警和故障自愈的智能分析服务层，

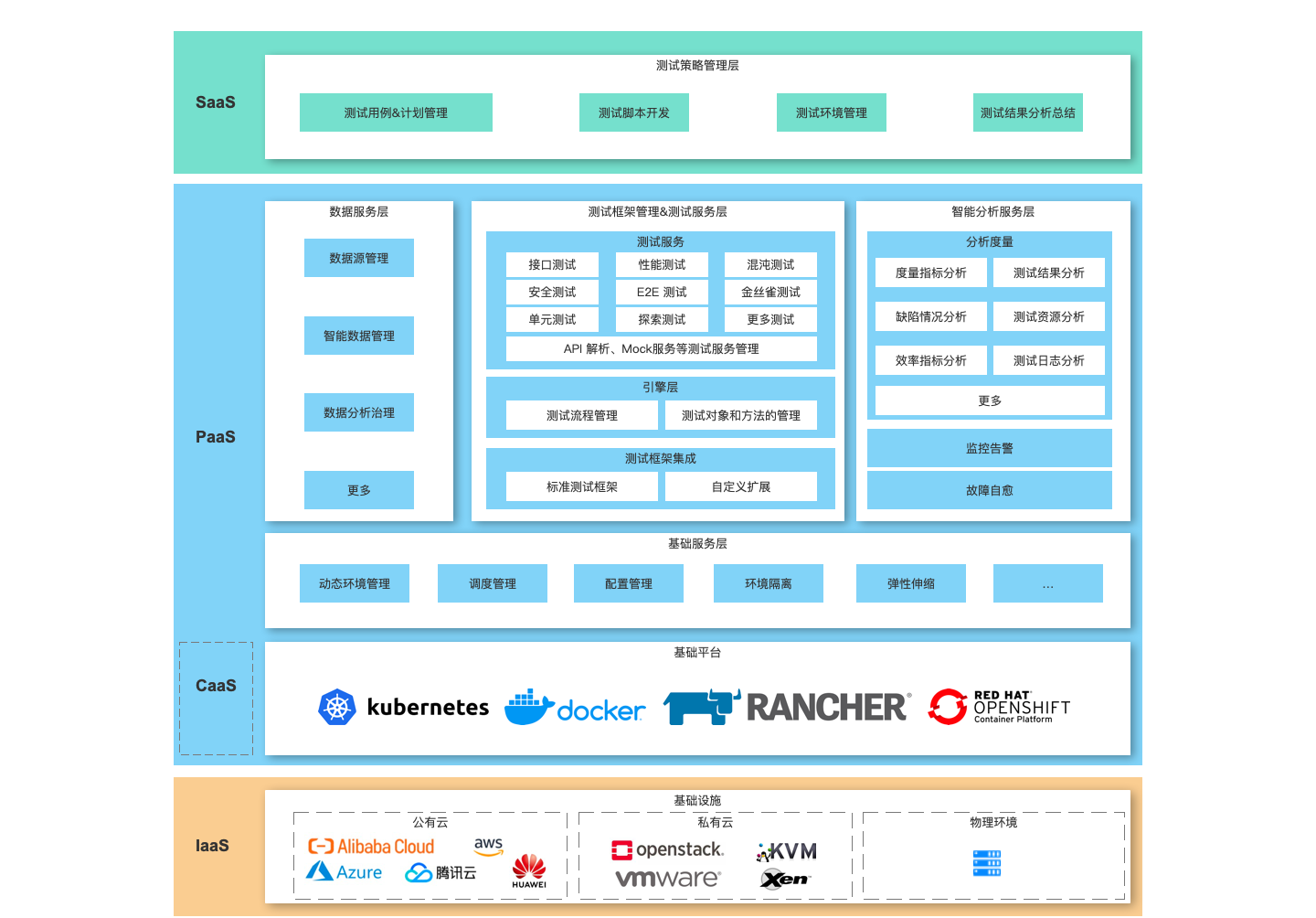


图1 云原生测试架构图

云原生是多种概念和思想的集合，包含了DevOps、持续交付、微服务和敏捷基础设施等。随着时间的推移和技术的发展，这个集合还将不断壮大。伴随着云原生的落地和发展，测试也发生了较大变化，在变化过程中，测试左移和测试右移的概念随之而起。结合DevOps、持续交付、微服务和敏捷基础设施这四大云原生领域来分析测试左移和右移的本质，如图2所示，测试左移需要尽早发现缺陷，以最小的成本解决问题，因此代码质量、单元测试、安全扫描等自动化测试框架能力应尽量左移到持续构建的过程中来让问题尽早暴露和解决，继而要求测试服务平台能够提供相关测试服务能力，以微服务调用的方式加入到应用的CI/CD过程中。在测试左移保证了应用质量、满足了业务需求的情况下，还需要通过测试右移来保障现实世界环境中的运行质量、性能和安全等。如何降低测试右移的成本和使用门槛，是测试领域的又一挑战。依托云原生微服务部署管理的能力，通过测试AI智能分析服务、数据治理服务、混沌工程等可为测试右移提供有力的技术支撑。不管是测试左移、测试右移还是其他自动化测试框架服务能力，都是基于云原生技术架构下的环境管理、测试服务能力调度部署、弹性伸缩和配置管理等基本特性，来实现测试服务能力本身的部署和调度管理。

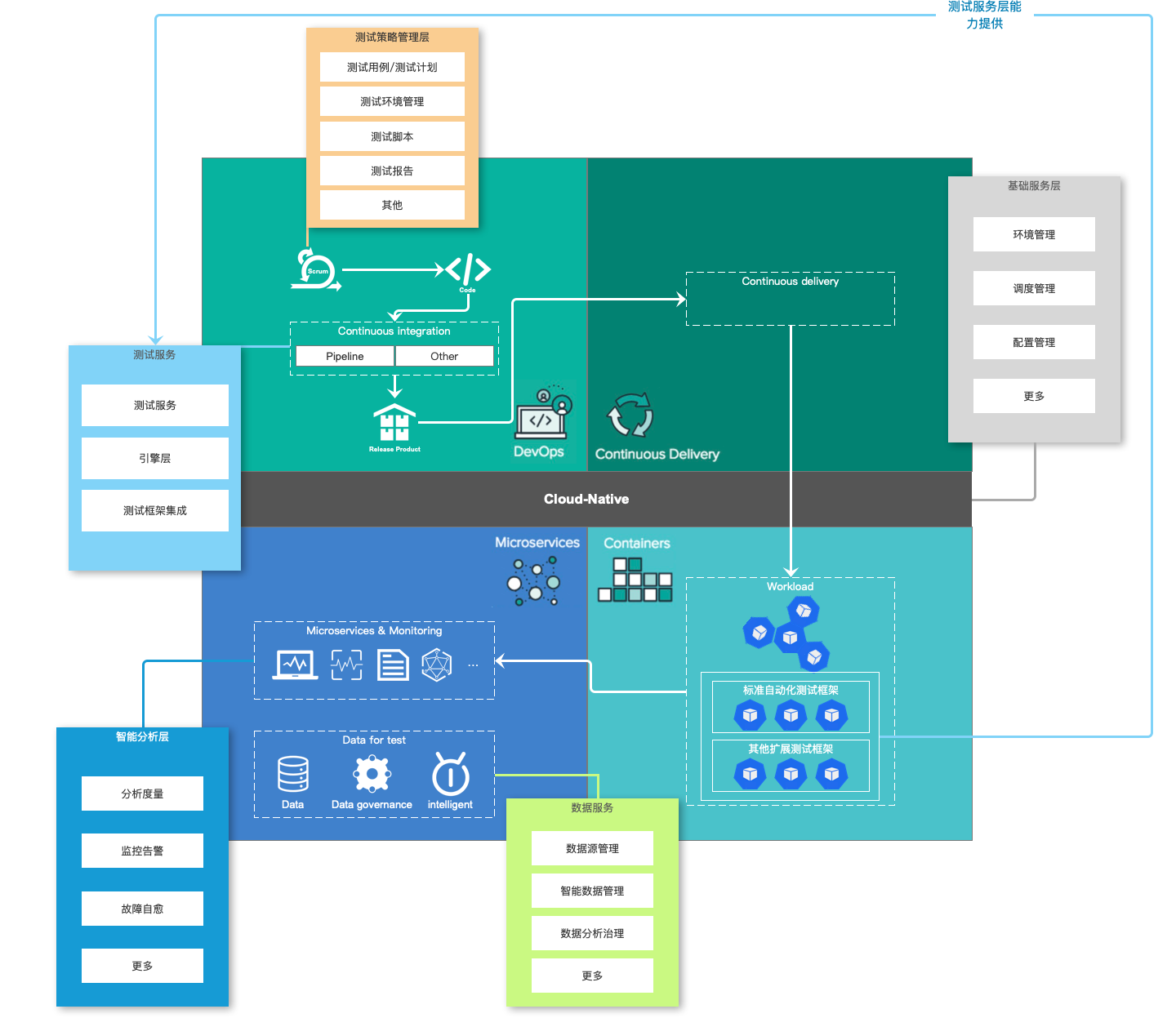


图2 基于云原生四大领域的分布架构图

# 三、 云原生给测试领域带来的红利

## (一) 测试环境部署更加便捷

云原生的容器技术应用为测试环境构建带来了极其轻量、秒级部署、易于移植、弹性伸缩的技术红利。

测试环境的搭建一般需要如下几个步骤：准备和维护管理测试资源->测试服务部署->测试环境扩展->测试环境销毁，这是一般测试环境搭建的主要步骤。当前云原生时代，应用的云原生化为测试环境的生命周期带来了变革。

### 1. 资源获取成本降低

在云原生时代，线下自建服务资源变成了线上按需购买，优势是显而易见的，用户不需要关心机房机器如何运输、如何布线、如何划分局域网、如何放置路由器等一系列硬件管理，也不需要关心机房停电问题，只需按需购买资源即可。这极大地降低了测试环境资源建设的门槛和费用。

### 2. 环境即用即拉

云原生时代，以容器化为代表，部署一个服务一般需要如下步骤:

（1）部署服务需要的操作系统版本；

（2）安装服务需要的各种运行环境依赖包和容器，例如Jdk、Tomcat等；

（3）启动服务/停止服务。

云原生架构下，服务以镜像方式存在，只需要提供一个镜像，一键拉起服务。

### 3. 测试服务快速弹性扩展

云原生到来之前，当测试服务负载能力不足、需要扩展服务时，要重新准备测试资源，然后进行一系列的测试服务安装部署才能够使用。而云原生只需要按需弹出一个服务，甚至智能化的云原生管理系统可根据负载要求自动弹出一个服务来适应服务压力变化。这给测试服务扩展带来了极致的体验。

### 4. 测试环境即弃即抛

云原生的应用让我们能够以零人力成本非常快速地从无到有建立一套“开箱即用”的测试环境，能够构造出测试需要的所有数据，同时也能快速回收，做到测试环境的用完即抛。

用完即抛的好处是：

* + 解决环境腐化问题，减少脏数据
  + 提高重复性，确保每次测试运行的环境都是一致的
  + 反向推进各种优化和自动化能力的建设（测试环境的准备、造数据等）
  + 提高资源使用的流动性。实际的物理资源不变的前提下，增加流动性就能增加实际容量。

## (二) 测试成本大幅降低

### 1. 人力成本大幅下降

云原生时代的到来会给测试带来大幅度的效率提升和成本降低。以往，测试环境的搭建和维护是十分复杂的。有时甚至需要开发人员花费几个人·日搭建。但随着云原生应用的普及，应用容器化、轻量化，通过云上便捷的弹性资源购买、应用镜像部署，测试人员完全可以自助搭建起云原生应用的测试环境。

### 2. 测试效率大幅提升

云原生时代是服务的时代、是标准化API的时代，测试能力也会越来越多的以云服务的形式展现。通过广泛使用开源技术，测试工具平台也得到了统一。通过基于云原生的测试服务，可以大幅度地提升测试效率。云原生时代，测试人员只需更关注应用本身的业务逻辑，通过使用标准化、统一的云原生测试服务进行测试，从而大幅提升整体的测试效率。

### 3. 测试内容更加聚焦于应用本身

Service Mesh是下一代微服务技术的代名词，同时也是云原生时代的代表技术之一。Service Mesh将以往隐藏在微服务框架中的服务间通信、鉴权、流量控制、监控，整体下沉到服务网格中实现。将与应用业务本身无关的通信层事情单独处理，使得应用可以更加聚焦在自身业务本身，同时服务间通信的变更也与业务彻底解耦。与此同时，测试也可以更加专注在应用本身的业务逻辑上的验证，而无需关注服务发现、鉴权、监控基础设施的验证，大大缩小了测试的验证范围，同时微服务框架升级带来的全量回归验证也随之减少。由于很多高可用的设计下沉到服务网格中实现，所以基于Service Mesh的应用比以往微服务框架搭建的应用也更加稳定可靠。

# 四、 云原生给测试领域带来的挑战

随着云原生化的推进，业务的迭代速度会越来越快，如何更快更好地保障质量是测试面临的挑战。挑战来自多个方面。

一是测试效能需要进一步提升。云原生化后版本发布效率加快，使得测试工作变得更加高频，需要加快测试工具向面向云原生转型，根据云原生的测试需求选取开源工具或自主研发方式，来提升测试效能。

二是链路测试能力需要进一步加强。微服务化的推进使得后端系统架构变得复杂，相互依赖增多，业务链路增长，测试的粒度需要更精细。对微服务开展接口测试、Mock测试、性能测试以及故障模拟测试，都是新的挑战。需要采用基于流量录制回放的测试方法，以及基于APM原理的全链路检测分析技术，来提升微服务架构平台的性能测试和故障检测能力。

三是集成测试的成本和复杂性提高。依据测试分层的金字塔理论进行云原生底层测试时，如单元测试、模块间测试、接口测试等需要进行更多的Mock测试，来减少外部依赖问题，使得过早进行集成测试的测试成本较高。

四是云原生测试体系和方案的标准化和规范化还不够。由于云基础设施的复杂性和不确定性，在云环境中如何进行管控面和数据面的功能测试、故障自愈测试、纵向单级和多级的混沌故障测试、全链路性能压测、云安全测试、运维操作自动化测试等，来保障各个服务之间的API可用性和健壮性等，云原生测试过程中面临业务链路问题多、跨层面问题定位难（IaaS、PaaS、SaaS），需要系统化的测试方案来保障。

五是云服务测试难度大。云服务的系统架构复杂，大量的计算或存储节点，配置参数的组合能够达到千万以上的巨量，同时对于云服务自身安全的测试，分布式云环境的多租户模式可能造成业务信息泄露等潜在风险，如数据隔离、隐私信息泄露、服务丢失、恶意攻击等，以上测试范围较难完全覆盖，云服务平台的容错性和可靠性测试极具挑战。

# 五、 云原生测试关键要素

## (一) 持续测试

DevOps是实施云原生的重要因素之一，持续测试则是实现DevOps的重要环节。在维基百科中，持续测试的概念是在软件交付流水线中执行自动化测试的过程，目的是为了获得关于预发布软件业务风险的即时反馈。

云原生发展至今，快速部署和迭代成为软件发布的常态，持续测试也成为不可忽视的一环。实施持续测试的关键环节如下：

* 测试要插入在软件生命周期的各个环节

测试左移、测试右移

* 快速测试

自动化测试（环境的自动化搭建、数据的自动化准备、脚本的自动化执行、结果的自动化分析和报告）结合手工测试

优化测试效率

确定正确的测试范围

* 及时反馈

持续测试成熟度的衡量指标：

## (二) 面向可恢复的测试

根据AWS等云计算厂商数据研究显示，在4年期间，20%的SSD会发生无法访问错误，30%~80%会出现block损坏，HDD等存储设备、物理服务器和周边其他硬件设备也有不同程度的损坏概率。当服务规模大于10000台时，对于单个设备来说为小概率的硬件故障在大规模集群中可以说每时每刻都在发生。

除此之外，现代分布式软件服务经常面临越来越复杂的网络环境，并且局域数据、服务上下有依赖关系。即使业务服务本身代码逻辑没有任何错误，由于基础设施IaaS和PaaS出现问题而导致整体业务失去服务能力的时间依然有相当的概率发生。

此时，如果每次异常都需要人为干预，系统就无法可靠的伸缩，很有可能导致经济损失。为此，每一层系统都会面向失败做设计，对下游组件零信任，确保在故障发生时可以快速发现和处理。

以Kubernetes、Service Mesh等为代表的云原生技术为产品服务的failover提供了重要支撑，从方法论和工具设计层面降低了灾难恢复的成本，但是这些支撑并不是完全免费的。

传统的单体架构应用通常需要经过改造适配并在合理的配置下，才能充分发挥云原生的稳定性、高可用能力。

这些改造适配、配置需要进行合理、充分、经常性的测试验证来保障合理性，才能确保迁移到云原生后的业务服务具有预期的高可用能力。

（1）面向灾备能力的测试场景分析方法

灾备是将容灾和备份结合的保障企业数据、业务高可用的方案之一，是一个产品服务必不可少的环节。传统灾备大多基于物理设备实现，需要在多点部署相应的物理资源，根据不同系统可靠性的要求，备份的地点和规模一般差异很大，并且需要大量的服务器及网络物理设备，投入规模较大，维护成本高。随着云计算的稳健迅速发展，基于云计算的云灾备方案应运而生，越来越多的企业转向云灾备，如将本地数据和应用灾备到云端，或从云端灾备到云端，这样可以借助云端的优势，实现异地灾备的目标，同时还可以充分利用云平台廉价的计算、存储和网络资源，云灾备方案建设成本低，且系统恢复更快。

灾备测试的核心原则是基于业务的影响分析，全面提升产品系统的高可用性，需要关注的两个关键技术指标RTO（恢复时间目标）和RPO（恢复点目标），而面向灾备能力的测试应遵循几个标准：

* 模拟极端错误发生，测试业务恢复功能和业务持续性流程；
* 发现业务系统潜在的隐患，确保出现突发情况时业务能够正常运行；
* 在极端流量冲击下，系统具备分流、服务降级的能力，保障整体系统的稳定以及主要功能的正常运行。

（2）云原生时代如何实施灾备演练

依托云的弹性、云的全球化部署能力，云灾备大大降低了灾备的门槛，已经是灾备行业的发展趋势。云容灾服务免去了灾备中心建设、硬件系统采购、运维等复杂工作，加上资源可弹性扩展、按量付费的特性，已大大降低了灾备系统建设的难度。通常，灾备系统建设完成后，面临的灾难不外乎数据级别、应用级别和灾备中心级别，因此所有的演练都是基于这三种级别中某一特定的场景而进行的。

Netflix公司提出了Chaos Engineering混沌工程的概念，通过随机关闭系统内虚拟机或容器来模拟系统故障，来测试系统的稳定性、可用性。随着云原生技术的发展，衍生出了遵循混沌模型的混沌实验执行工具ChaosBlade、Chaos Mesh等，可以在虚拟机、Kubernetes环境中注入故障，以达到灾备演练的目的。

* 准备工作和前提条件
* 1）灾备中心部署，了解关键应用的部署、环境的依赖以及应用的客户端连接等情况。
* 2）需求分析，明确灾备方案，根据应用服务器数据、数据量，制定RPO/RTO标准。
* 注入工具
* 备灾演练可通过故障注入工具来模拟各场景异常时系统的处理能力，注入工具可通过模拟调用延迟、服务不可用、机器资源满载等，来衡量微服务的容错能力，通过模拟杀服务pod、杀节点、增加pod资源负载，观察系统服务可用性，来验证容器编排配置是否合理，或者通过模拟上层资源负载、依赖的分布式存储不可用、调度节点不可用、主备节点故障等，来测试PaaS层是否健壮等等。
* 验证工具
* 风险控制

（3）业界最佳实践

## (三) 应用运维的测试

随着DevOps和AIOps概念的提出和实践，传统的运维已经由脚本、人工逐渐转变为白屏化、自动化、智能化。使用这些管控和运维工具同样面临着出错的风险。在应用发布、日常维护、故障恢复过程中，都伴随着大量的运维操作，一旦出错有可能会导致线上的重大故障。

从稳定性角度考虑，对运维工具进行验证非常重要。需要保证所有的关键运维链路准确无误，所有自动化和智能化运维决策都能达成预期效果。

### 1. 可扩展性

随着智能设备的普及和边缘端数量的增长，数据服务必须具备较强的可扩展性，在数据量变化时，灵活应对。为帮助创建安全的高性能部署，DevOps可通过自动数据备份和自动化一切，合作与组织，启用低风险释放，精益流程和易用性，正确部署和监控这5种方式使IT基础架构更具可扩展性，当业务需要增大流量时是否需要扩容？如果扩容是扩行还是增大单个容器CPU规格？当业务需要增大数据量时是拆列合适还是扩大单个容器的内存大小合适？这些都需要通过测试的手段根据自身应用的架构、资源规模、业务模型等因素制定符合要求的扩展策略，通过很少的改动实现整个系统处理能力的线性增长，实现高吞吐量和低延迟高性能。

### 2. 弹性

容器化带来极致的弹性，使扩缩容成本大幅下降，同时可以根据流量自动伸缩。这些特性对于应用运维是革命性的变化，但这些优秀的特性同样伴随着大量的潜在风险。

（1）频率

（2）扩缩容规模

（3）其他基础组件的容量瓶颈

（4）弹性伸缩策略

（5）容器收缩时业务的稳定性

（6）扩缩容临界值，即何时开始扩缩容，例如并发数，响应时间，吞吐量

需要根据自身应用的架构、资源规模、业务模型等因素制定符合要求的扩缩容模型，并在测试环境中进行演练和验证，以保证模型能够达到预期效果，在服务实例伸展时，可以根据实际情况设置不同的权重值，提供服务实例权重值实时更改能力，也可以有监控工具对系统扩缩容之后的业务指标进行监控，提供扩缩容之后系统运行的各项指标数据。

### 3. 可观测性

应用的可观测性通常包含3个部分：

* 系统指标
* 应用指标
* 业务指标

大量的线上问题处理依赖这些指标的监控和告警，如果在关键时刻告警失效，将给业务带来重大损失。

可观测性是自动化智能化运维、故障自恢复的基础。但随着应用的发展，可观测系统往往会面临非常多的问题：

* 监控失效/过度监控
* 无效告警
* 误告警

针对以上三种情况，我们需要制定相应的测试方案，保障可观测系统的稳定可靠。从测试方案来讲，需要包含以下几个方面：

* 业务梳理
* 技术/工具，场景模拟工具，效果自动检测
* 防腐机制，持续迭代，防止可观测系统趋于混乱

同时，需要对可观测性的提升制定相应的策略，通过SLO的制定和持续观察，形成产品可观测性和SLO的提升闭环，达成相互促进。

对于可观测性而言，需要有相应的量化指标。

### 4. 一致性

DevOps表达的一致性路径包含理念、技术、环境三个维度上的一致性，应用运维的测试偏向于环境一致性，云原生应用部署、运行在不同的平台和环境上需要保持运行状态和结果的一致性，这样当底层架构变化或者平台升级时，业务应用可以保证正常运行，或者可以快速迁移到新环境中。我们需要制定相应的测试方案，保障生产环境与测试环境的一致性，多套环境的一致性。从测试指标来讲，需要包含以下几个方面：

* 配置文件的一致性
* 系统版本的一致性
* 程序版本的一致性
* 内核参数、系统配置的一致性

### 5. 健壮性

最大化的提供容错能力、优雅终止的能力，可保障整体业务运行的健壮性和易处理能力，除了系统性的混沌工程演练之外，每一个应用模块、进程均需考虑在分布式环境中，任意进程异常死亡后的应对能力。可结合故障注入操作指南，使用chaoseblade故障注入测试工具，进行基础的操作系统、云服务等故障注入，如CPU、内存、进程、文件、内核，以及容器服务故障注入，如杀Pod、删容器、驱逐节点等，在产生故障时测试关注点，如服务质量、监控告警、流量调度、弹性伸缩、服务稳定性、程序的自愈能力、系统的自我容错等能力，系统是否能够自动恢复或者忽略故障继续持续的运行，恢复时间等等。

### 6. 可靠性

确保应用程序跟新和基础设施变更之后的系统是否安全可靠，以便在保持最终用户优质体验的同时，更加快速可靠地进行交付。使用持续集成和持续交付等实践经验来测试每次变更是否能够正常运行，实时对当前系统的功能以及性能进行监控，判断系统是否持续可靠的在运行。

### 7. 安全性

DevOps的落地使得软件交付的频率大幅度提升，这就进一步使得DevOps安全问题变得更加需要重视。那么，有哪些层面的安全问题需要考虑呢？

（1）自研代码的安全

内部研发的代码，建议通过单元测试、静态扫描等基本手段进行安全合规性检查。自研代码发现问题可以给研发人员指定相应的任务去解决，因为这些代码都是自主可控，而开源代码缺陷基本不可能去修复，这是应对自研代码安全威胁的优势。自研代码经过扫描之后，可以知道目前的技术债务(Technical Debt)情况，防止技术债务不停增长。

（2）开源的安全问题

传统开源安全威胁解决手段通常是成立开源技术专家组(委员会)，列出目前已知的、企业必须重视的安全威胁列表，在上线前由安全组对交付件进行扫描，发现问题立即进行整改，然后再次反复此流程直到符合上线安全规范。从目前市场调查的情况来看，开源问题所带来的威胁遍布各行各业，且比例呈上升趋势。

目前通过自研代码检查和第三方开源安全检查来发现系统存在的安全漏洞，通过自动化的统一流程结合全面深入的安全扫描能够实现对任何一个安全细节都兼顾到，并且可以快速了解安全问题的影响范围。面对这种情况，需要有一套安全测试的工具，可以与自动化扫描相结合，或者模拟一些安全事件来验证自动化安全扫描是否扫描出全部的安全问题。

### 8. 自动预案验证

大量的自动化预案是自动化和智能化运维的基石，由于预案数量庞大，有效性往往会随着时间逐渐劣化。甚至有的预案从设置开始，就没有被验证过。一旦出现问题，预期的运维动作将不会生效。面对这种情况，需要有一套预案验证的工具。可以与混沌工程相结合，在测试或演练环境中注入能够触发预案执行的动作，然后观察自动化预案是否被触发、效果是否达到预期。这样的机制需要定期执行，甚至是高频执行，从而保证自动化预案的有效性。

## (四) 测试服务化

测试服务化是服务能力一种拓展形式，测试服务化最大的好处是可以将提供的服务范围扩大化，甚至可以成为重要的测试基础服务，加速测试架构的演进，避免在重复造轮子的循环中浪费时间。

测试服务化领域众多，包括接口测试领域、功能回归框架平台领域、E2E测试领域、灰度测试领域、故障演练测试领域、性能兼容性测试领域等，所有领域的测试服务化构建出了测试的基础实施。

在云原生时代，将测试基础设施与DevOps、持续交付等概念有机结合，将服务化的测试工具平台集成到持续交付流水线上的工具链来实现更大的价值输出，是一个很好的尝试。

### 1． E2E测试

### 2． 接口测试

云原生软件系统在接口层面有两大特性：

* 在微服务场景下，接口数量大大提升
* 接口的相互依赖更加复杂，链路更长

在这两个前提下，接口测试成本显著提升，传统接口测试服务已经无法适应。

云原生基础设施也为接口测试带来了非常多的利好，应用这些云原生特性可以将接口测试的成本大幅降低。

### 3． 性能测试

性能测试服务化之后与之前的区别有以下三方面：

1. 使用场景

* 接口性能测试：常态化，基线管理，CI/CD结合
* 链路性能测试：基于mesh的压测管理和路由，无侵入业务代码

1. 成本

* 压测参数准备
* 压测环境准备

1. 最终目标

提升业务稳定性

* 排查能力
* 性能优化建议
* 容量评估能力

### 4． 移动端测试

在移动互联网时代，移动端的测试需要购置大量的移动端硬件设备，特别是安卓平台，碎片化问题严重，需要采购大量的移动设备来做兼容性测试。但是往往只有大公司才有预算采购众多的移动端设备来进行开发测试，而中小公司只能测试主流的几款机型。大量的终端适配问题、性能问题、体验问题只能依赖于众测模式或者种子用户反馈。但是，无论众测模式还是种子用户反馈，对于处于保密阶段研发的产品，信息安全都很难管控。创意、功能很有可能被泄漏，同时对收到的反馈问题进行研发调试也非常困难。而且，即便是大公司，测试设备的采集预算也是有限的，同时不同项目之间的设备不能共享、移动设备利用率低下也造成了极大的浪费。所以，在腾讯、阿里、百度等企业出现了云真机项目，在企业内部实现了移动硬件资源共享，大大提升了资源使用率。

在云原生时代，这一模式得到了进一步放大。以往只是企业内部共享的模式，通过云服务共享开发给公众用户。mPaaS在云原生时代发展迅速，Google的Firebase、Facebook的Parse、Apple的Cloudkit就是其中的佼佼者。

### 5． 测试数据管理

在数据驱动测试的前提下，测试数据自动录制和智能化生成会使测试数据数量暴涨，需要考虑专门的测试数据管理服务。这些测试数据将成为测试活动不断迭代和完善的核心资产。

数据管理服务并不只是一个数据仓库，数据服务是智能化测试的关键能力，在云原生时代的数据服务应具有以下几个特征:

（1）智能化

在基础测试服务相对完善的情况下，测试行为应以数据作为驱动，这就需要能够探索和挖掘数据之间的内部联系。测试数据往往是大量的异构数据，可以使用数据湖的方式。用AI和数据挖掘技术，建立数据与业务之间的模型，使得测试行为更智能，漏测率更低，同时成本大幅下降。

（2）可演进

业务的复杂和多变性使得生成测试数据的格式、内容都会发生变化，数据测试服务应该能够感知业务变化自动迭代，使得数据时刻保持最新，以适应当前多变的业务背景。

（3）安全性

数据的安全性是不可忽视的一环，从线上生产环境录制的数据中会带有大量的敏感信息。灵活可配的数据脱敏方式也是数据服务必须具备的能力。

## (五) 数据驱动测试

数据驱动测试，分为广义数据驱动测试和狭义的数据驱动测试，之前测试领域提及较多的数据驱动测试属于狭义驱动测试；而云原生时代，数据驱动测试应该兼具狭义驱动测试和广义数据驱动测试。

狭义的数据驱动测试：数据的改变驱动自动化测试的执行，最终引起测试结果的改变。例如在Excel中填入一个函数的各种参数值，用这个数据文件驱动函数运行，进行测试。

广义的数据驱动测试：通过大数据分析，找到具体风险点，通过数据分析驱动测试精准。例如通过对服务生产接口调用和用户使用场景的大数据分析，找到用户使用最高频的接口、函数等，进行针对性测试。

狭义的数据驱动测试是一种在软件测试过程中使用的方法，用于描述直接测试的输入、可验证输出的条件表，以及测试环境的设置还有控制编码的过程。这种方式中测试脚本与测试数据分离，让测试数据从测试脚本中分离出来单独存在，解除脚本与数据之间的强耦合。测试脚本中不再掺杂数据，测试数据在数据驱动测试中会以文件或者数据库等作为提供者的形式存在。脚本每次执行会机械地从数据文件或者数据库中读入测试数据，根据测试数据的不同走进不同的测试路径。在整个测试中，测试脚本基本是不变的，测试数据是丰富可变的，可通过不同的数据控制测试脚本中代码的走向。

广义的数据驱动测试核心是通过系统风险类的数据分析得出最需要做测试的点和相关测试方法进行测试，更加精准有效。随着系统的不断成长，会沉淀一些冗余代码，系统也会越来越复杂，云原生时代微服务架构让系统的复杂性更高。很少有人能够全面了解整个系统的逻辑，很难了解把控系统的风险点，在这种情况下测试，一般只能进行全面测试，对于已经无用的函数或功能、用户很少使用的功能，仍然要投入资源进行测试，导致资源的大量浪费。如果使用大数据分析驱动测试，可以通过对用户使用场景和对应代码路径覆盖记录进行大数据分析和处理，得出使用场景和对应的代码路径拓扑。有针对性地测试用户最优先使用的功能以及风险最高的函数，让测试更加精准有效。

云原生时代，海量的数据存储在云端，为测试数据挖掘提供了便利，可以方便地构建数据仓库和数据清洗逻辑。海量测试数据挖掘可以为应用的测试提供广阔的想象空间，例如通过功能测试数据挖掘，可以找到功能测试低效的具体点并提供解决思路，持续提升测试效能；通过对性能数据挖掘，可以预判系统性能瓶颈，实时根据生产流量变化自动化扩缩服务容量。

## (六) 测试智能化

当前云原生时代，测试应向智能化发展。测试的发展最早以手工测试为主，是劳动密集型时代。后来发展出自动化测试，提升测试人员的技术能力、写测试脚本进行测试，通过可重复使用的测试脚本来减少回归测试的人员投入，以自动替代手工，降低测试手工投入成本，是机械自动化时代。测试智能化是通过机器学习、流量录制回放等智能化手段使编写测试脚本的成本再大幅降低。

在云原生时代，测试走向智能化可带来以下变化：

1. 准确性增强
2. 效率更高——人工智能程序无需手动处理成千上万行代码，可在几秒钟内扫描代码并在更短的时间内检测到错误。通过将人工智能纳入重复测试中，质量检查工程师可以专注于测试新功能或关注软件的重要部分
3. 成本更低，更加自动化——人工智能程序可以随着代码的更改而发展，适应并学会识别新功能
4. 正确理解客户需求——人工智能程序可以检测类似的网站和应用程序，以确定哪些因素能帮助赢得目标受众，也可以帮助研究大量竞争产品以确定其优势。通过正确理解客户的需求，可以创建测试用例，以确保产品在实现这些特定目标时不会损坏

智能化测试经过实践，当前已有一些应用，主要分为如下几类：

* 差异测试——比较应用程序版本，对差异进行分类并从分类的反馈中学习
* 视觉测试——利用基于图像的学习和屏幕比较来测试应用程序的外观
* 声明式测试——以自然语言或特定领域的语言明确测试的意图，并让系统确定如何执行该测试
* 自我修复自动化——通过自我学习、大数据的训练来更加智能化

智能化测试可以定义为如下5个级别：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 级别及描述 | | 测试左移 | 测试主体 | | | | | | 测试右移 |
| 智能测试级别 | 名称 | 代码分析 | 测试场景构建 | 测试操作执行 | 结果解析理解 | 结果验证（认知/决策） | 缺陷定位 | 缺陷自愈 | 线上异常发现 |
| （认知/规划） | （感知/理解） | （感知） | (监控/巡检) |
|  |  |  |  |
| L0 | 人工测试 | 0 人工分析 | 0 人工构建测试场景，输出手工用例 | 0 人工执行 | 0 人工识别 | 0 人工验证 | 0 人工定位 | 0 人工修复 | 0 人工监控 |
| L1 | 工具辅助测试 | 1 人工分析，有工具辅助提供参考信息 | 1 人工构建测试场景，输出脚本用例 | 1 人工驱动机器执行，机器具备单步执行能力 | 1 人工编写代码驱动机器识别 | 1 人提供预期，编写代码驱动机器比对，可做简单逻辑比对，如是否相等 | 1 人工定位，机器提供辅助信息 | 1 人工修复，机器提供辅助修复信息 | 1 人工埋点，自动上报 |
| L2 | 自动化测试 | 2 机器基于人工编写的规则，工具实现分析、报告 | 2 人工构建测试场景，批量输出脚本用例 | 2 机器一键批量执行，即可连续多步、完整执行单个用例，可批量执行多个用例，有一定调度能力。 | 2 人工编写代码驱动机器识别，机器辅助比例较高 | 2 人提供判定逻辑，机器基于人提供的逻辑进行结果判定，可进行复杂逻辑判定，如图像识别，状态机等，新功能需要人介入提供新逻辑 | 2 人基于机器提供的信息和分析能力进行定位 | 2 人工编写修复脚本，人工触发脚本进行修复 | 2 自动埋点，自动上报，生成监控大盘，并提供报警规则 |
| L3 | 初级智能测试 | 3 机器基于机器学习训练出的模型，进行分析，报告，新类型的分析需要人协助训练新模型；bug准确率>=50%，召回率>=10% | 3 机器基于人给定的场景规则或模型，机器筛选场景，生成用例，可供下一环节执行，新功能场景需要人提供规则或模型；业务覆盖率>=70% | 3 机器一键批量执行，即可连续多步、完整执行单个用例，可批量执行多个用例，有一定调度能力。一键执行用例比例>50% | 3 机器基于人工编写的规则解析结果，结果变化时需要人工介入提供新规则 | 3 机器基于人提供的规则进行结果判定，可进行复杂逻辑判定；发现业务逻辑bug占比>=30% | 3 机器基于人提供的规则进行定位，可定位到功能、接口，新类型问题需要人介入提供新规则；定位的准确率>=50%，自动定位占比>=50% | 3 机器基于人提供的规则提供修复建议，人工驱动机器进行匹配修复；问题修复比例>=10% | 3 机器基于人提供的规则识别线上异常，发现率>=90%，误报率<30% |
| L4 | 高度智能测试 | 4 机器基于机器学习训练出的模型，进行分析，报告，新类型的分析需要人协助训练新模型；bug准确率>=70%，召回率>=20% | 4 机器基于人给定的场景规则或模型，机器筛选场景，生成用例，可供下一环节执行，新功能场景不需要人提供新规则或模型；业务覆盖率>=90% | 4 机器一键批量执行，即可连续多步、完整执行单个用例，可批量执行多个用例，有一定调度能力。机器能够对执行时机进行主动判断（如出现被测对象时进行执行），并对有效性进行主动判断（如用例非因bug失败时，可主动恢复）。一键执行用例比例>80%。具备精准测试的能力；任务编排和调度：提升机器利用率，降低执行时间 | 4 机器基于机器训练的模型解析结果，结果变化时模型可自主识别，极少需要人工介入；实际使用用例占比>=80% | 4 机器基于机器学习的模型进行结果判定，可进行复杂逻辑判定，新功能可复用，无须人介入；发现业务逻辑bug占比>=70%,准确率>=90% | 4 机器基于机器训练的模型进行定位，可定位到接口、代码，新类型问题无需人介入；定位的准确率>=70%，自动定位占比>=70% | 4 机器基于模型提供修复建议，人工确认修复方案，机器自动进行匹配修复;问题修复比例>=20% | 4   机器基于模型识别线上异常，发现率>=95%，误报率<20% |
| L5 | 全智能测试 | 5 机器基于机器学习训练出的模型，进行分析，报告，新类型的分析无需人介入训练新模型（机器自学习） | 5 无须人介入，机器全自主分析筛选场景，生成用例，可供下一环节执行，且用例可被人理解 |  | 5 无人介入，机器自动建立感知模型，并输出感知结果，感知结果可被人理解。 | 5 无人介入，机器全自主识别预期结果或预期判定逻辑，进行复杂逻辑判定。 | 5 机器全自主定位，可定位到代码行，并给出要根因分析。 | 5 无人介入，机器自动生成修复方案，并执行修复 | 5 无人介入，机器自动发现线上异常 |

# 六、 云原生测试发展趋势

## (一) 云原生时代测试角色的变化

随着云原生时代的到来，应用架构也从单体架构演变成分布式化、微服务化、容器化和网格化，单个服务更加轻量化，服务自治和独立部署，服务之间也更加松耦合。

在云原生时代，开发模式也由传统的瀑布开发演变为更加快速灵活的敏捷开发。在传统的开发模式中，开发和测试有很明确的界限，产品开发完成后交给测试进行验收测试。在敏捷开发模式中，开发和测试的活动开始融合在一起，测试人员可以参与故事改进、同行代码审查、单元测试和实践活动中，测试活动左移到研发过程中。

云原生的应用和服务都部署在公有云、私有云或者混合云之上，依赖云上的公共服务和组件，测试程序开始转向服务化，测试过程也向云上转化，测试活动也开始“右移”至发布后的生产环境，更多的测试工具在线上展开。云原生应用之间的低耦合高内聚的特性，也要求测试纵向的更深入到单个服务的内部进行专项测试。

因此，在云原生时代，测试的角色不断的向左深入到扩展到研发过程中，向右扩展到发布后运维过程中，同时还要在服务内部进行更深入的专项测试。测试不再是一个单纯的验收的角色，而是全面的参与到产品开发的各个环节，从整体上推动提高产品的质量。

## (二) 云原生测试对技术要求越来越强

在云原生时代，软件的迭代速度越来越快，和新技术的不断涌现， 对测试的要求也是越来越高。云原生应用的基础环境大部分都是Kubernetes来管理的，Kubernetes本身也是一个非常复杂的系统，对维护和使用都有较高的要求。

云原生应用基本都是分布式、微服务化和容器化的应用，一个应用可能包括众多微服务，微服务之间通过Restful API或者GRPC进行通信。云原生应用在解决了传统单体应用的缺陷，并利用云资源的便利性快发构建产品的同时，也引入了复杂性和更多的技术依赖性。

相比之前的传统应用的测试，云原生测试需要测试人员能够快速的学习更多新的技术和知识，如Istio的Service Mesh、ChaosBlade和ChaosMesh等混沌测试工具、pact契约测试、AI测试、云设施的配置和使用、以及云上测试工具和自动化测试的开发等，都对测试人员的技术提出了越来越高的要求。

## (三) 云原生测试平台持续演进

根据云原生端到端的业务测试需求，对面向云原生的不同类型自动化测试能力进行组合编排，如功能 Web UI自动化测试、接口自动化测试、全链路压测、基于故障测试的混沌工程相结合，实现业务功能快速验证后，排查分析并解决性能瓶颈，探索未知故障的复合场景，推动系统质量和业务稳定性不断提高。一站式的测试平台加快整合面向云原生的自动化测试能力或测试工具，基于云化的测试环境，持续提升平台的测试智能化能力，来提高云原生的测试效能。

# 附录 云原生测试最佳实践案例

## (一) \*\*\*\*\*

## (二) \*\*\*\*\*

## (三) \*\*\*\*\*

中国信息通信研究院

地址：北京市海淀区花园北路 52 号

邮政编码：100191

联系电话：010-62300072

传真：010-62304980

网址：www.caict.ac.cn