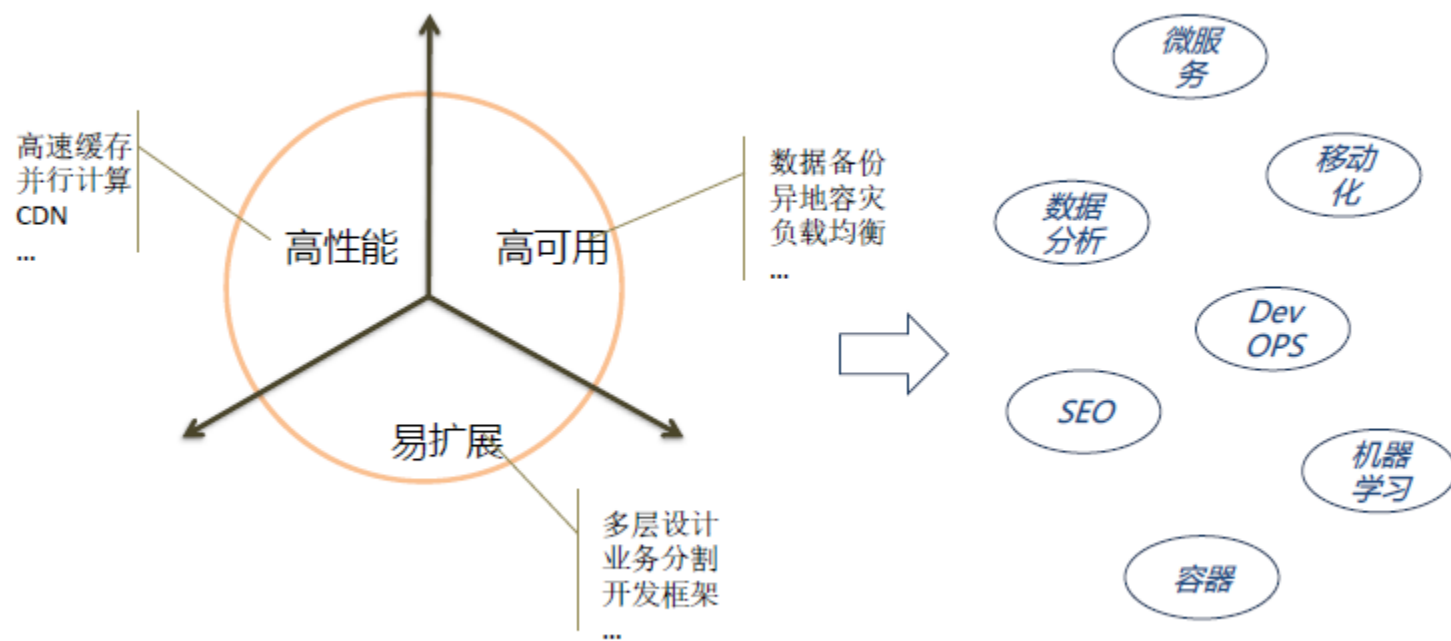




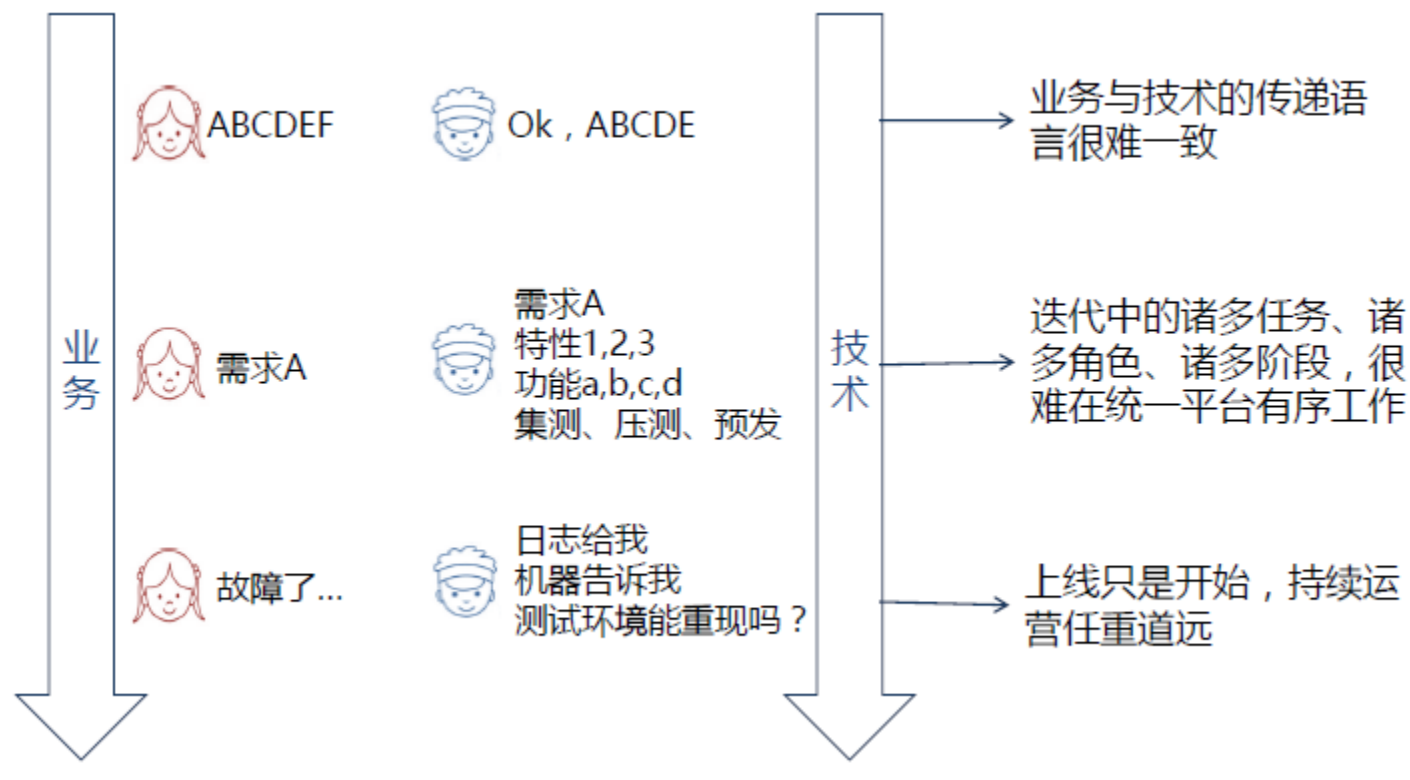
DevOps

罗晶

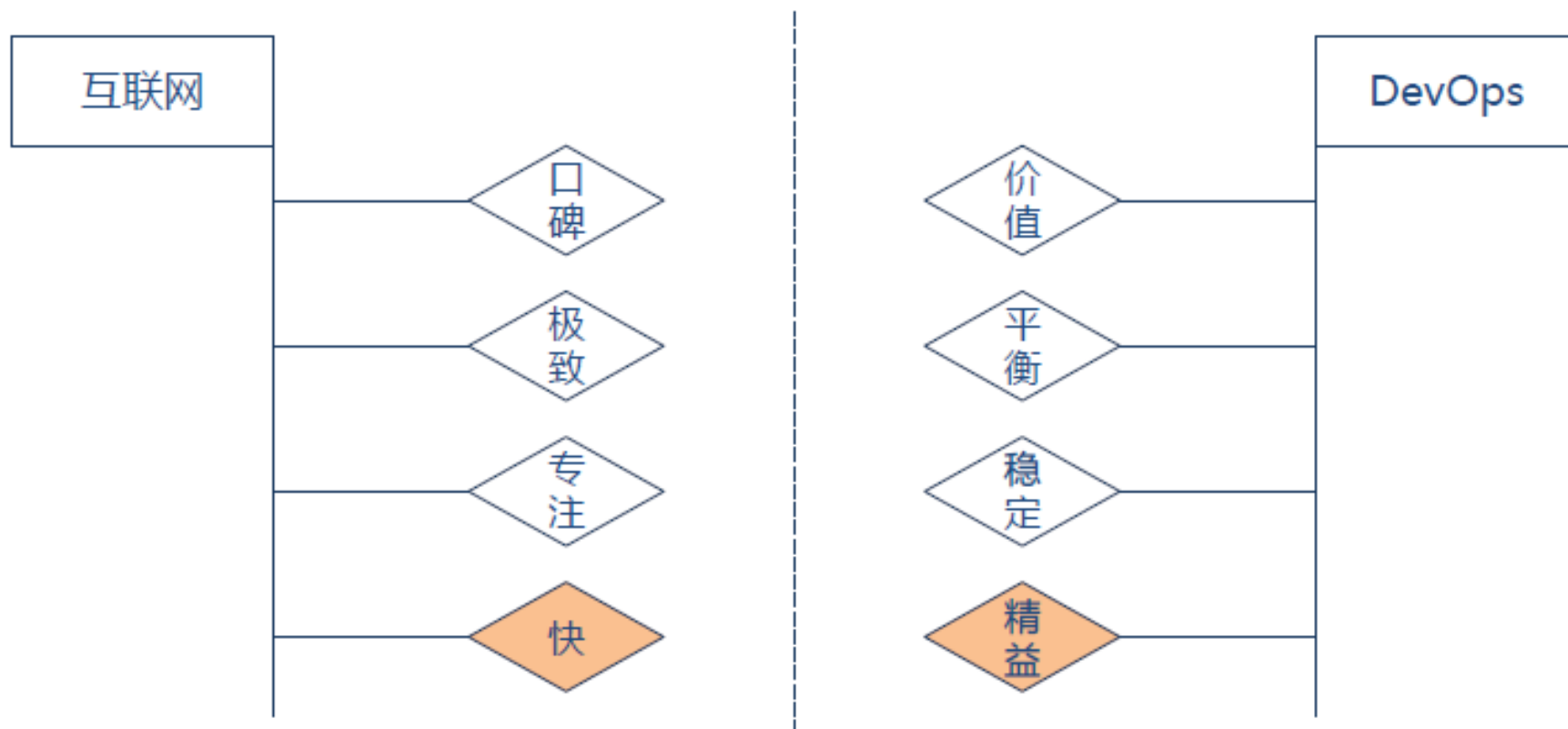
- ▶ 介绍DevOps
- ▶ 别人家的DevOps
- ▶ 生产环境
- ▶ 如何进行DevOps
 - 评估 管理
 - 服务质量和目标
 - 减少琐事



沟通失真与业务响应时间



▶ “DevOps思维” 不就是 “互联网思维”



► DevOps不一定需要用微服务和容器支撑

Vert.x - 从零基础迈向微服务之巅（下）

2016-07-27 Clement, Julien Vert.x北京用户组

目录

- 1. 序言
- 2. Vert.x
- 3. 解道微服务
- 4. 微服务应用 —— 微型交易系统
- 5. 环境准备
 - 5.5.1. 没有安装IDE ?
 - 5.2.1. Docker工具箱/安装包
 - 5.2.2. 镜像
 - 5.2.3. 创建Docker主机映射

Docker与实现DevOps的三种方式之系统思维

2016-05-11 周小睦 译 灵雀云

Devops2.0工具集黑宝书-读书笔记之6 - Docker世界里的配置管理

2016-07-27 原创 Martin Liu DevOps教练



① 在系统架构的自然演进中，诞生了微服务

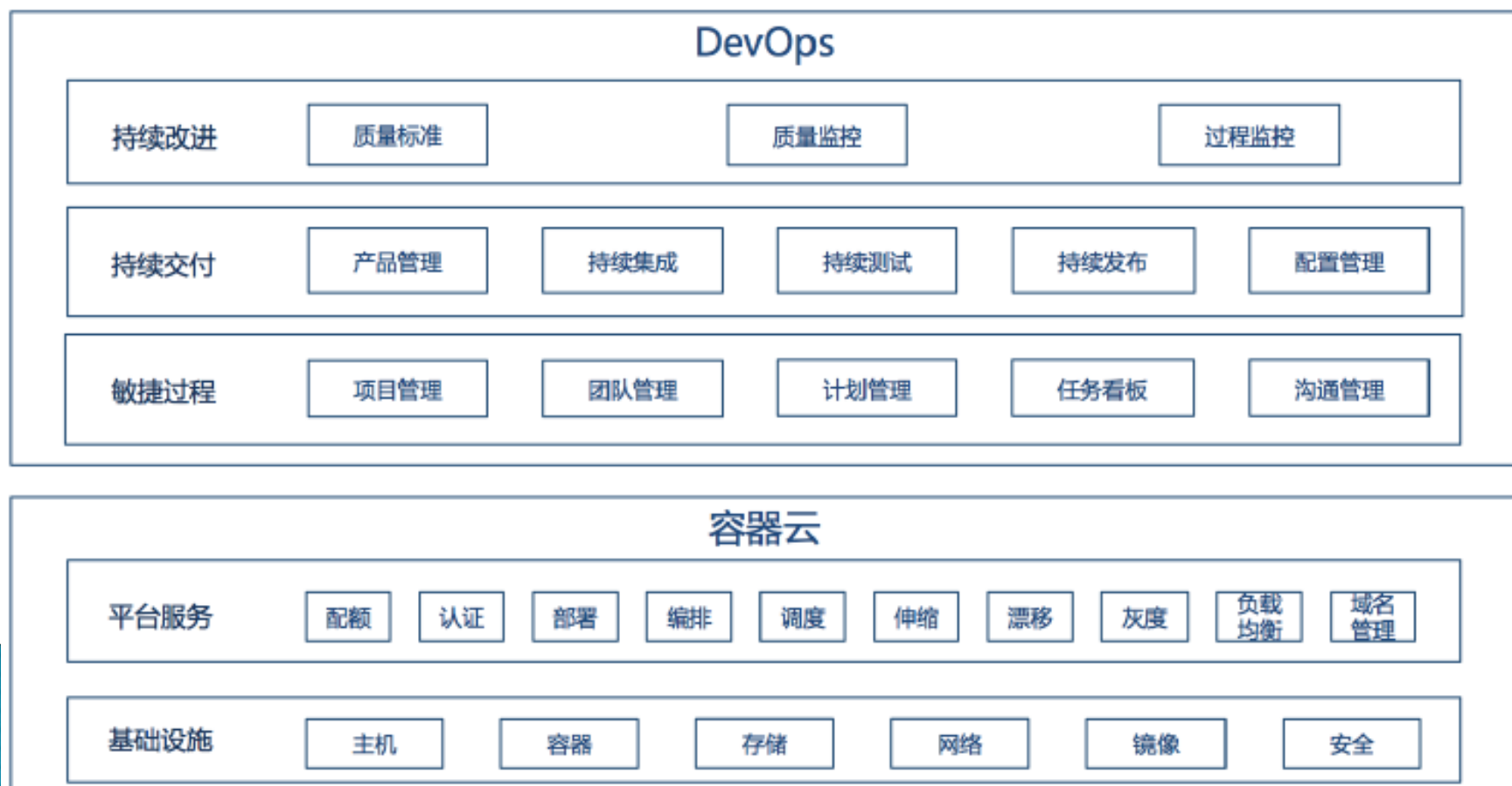


② 微服务架构让容器技术再次风靡



③ 容器让DevOps的某些环节更简单

► DevOps云平台功能架构



► DevOps打通工具链

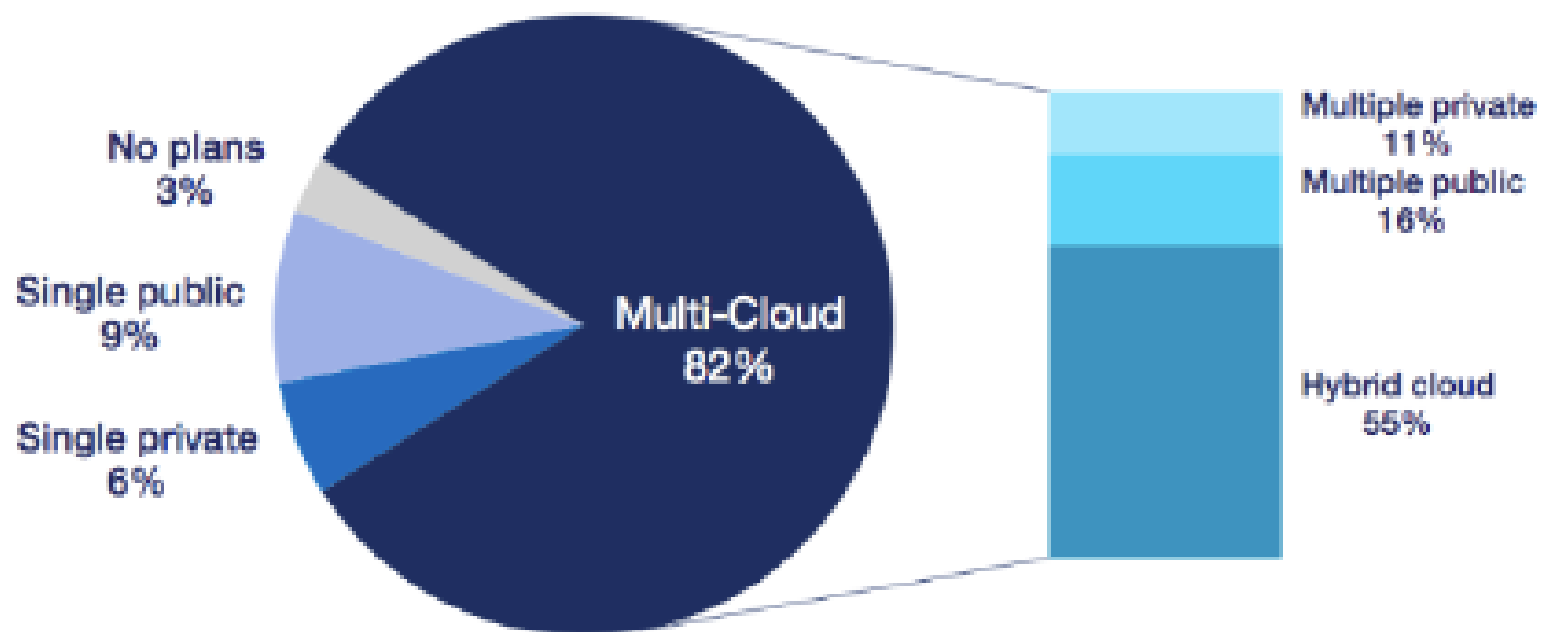


▶ 上云后DevOps自动化运维现状需求

- 混合云多云是多数企业的选择，企业基础设施形态为混合云多云混合IT形态
- 应用分布物理机，虚拟化平台，公有云，私有云，容器云
- 多数企业上云后应用直接运行在IaaS，而不是PaaS，容器云

Enterprise Cloud Strategy

1000+ employees



Source: RightScale 2016 State of the Cloud Report

- ▶ 介绍DevOps
 - ▶ 别人家的DevOps
 - ▶ 生产环境
 - ▶ 如何进行DevOps
 - 评估 管理
 - 服务质量和目标
 - 减少琐事
- 

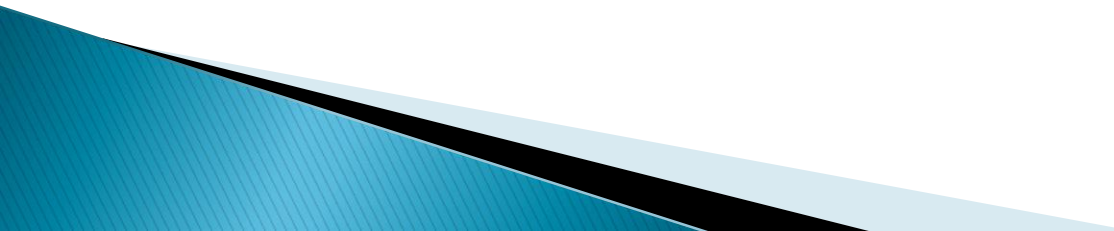
▶ 云DevOps自动化运维现状，需求和目标

▶ 现状：

- 混合云多云是多数企业的选择，企业基础设施形态为混合云混合IT
- 上云后应用分布运行在物理机，虚拟化，IaaS以及容器云
- 上云后多数应用直接运行在IaaS上，而不是PaaS上
- 云提供了可编程的基础设施和全栈自动化基础
- 云平台本身及传统现有工具离实际的需求有一段距离

- ▶ 需求目标：实现混合云混合IT环境下DevOps高效自动化运维
 - 充分基于云可编程特性提升自动化程序，实现全栈自动化
 - 现境上实现混合IT混合云多云环境下的DevOps自动化运维
 - 应用上支持各种类型的应用，平台，中间件，操作系统运行环境
 - 实现上能够落地，能够以小代价无风险方式支持已有主机环境应用自动化

- ▶ SRE site Reliability Engineering
- ▶ 系统管理员模式：系统管理员的日常工作和研发工程师相差甚远，通常分属两个不同的部门：开发部（Dev）和运维部（Ops）
- ▶ Dev/Ops分离的团队模型存在些问题：
 - ▶ 1 **直接成本** 系统管理员团队大部分依赖人工处理系统维护事件以及变更实施。
 - ▶ 2 **间接成本** 研发团队和系统运维团队分属两个部门所带来的间接成本就没那么容易度量了，但是间接成本往往大的多

- ▶ SRE团队里基本上有两类工程师
 - ▶ 第一类标准软件工程师，具体来说就是那些能够正常通过软件工程招聘的人
 - ▶ 第二类基本满足软件工程师标准，同时具有一定程度其他技术能力的工程师
- 

- ▶ DevOps这个名词是2008年年末流行起来，目的是想将IT相关技术与产品设计和开发过程结合起来，着重强调自动化而不是人工操作，以及利用软件工程手段执行运维任务等。SER是DevOps模型在google具体实践。
- ▶ SER团队要几类职责：可用性改进，延迟做强化，性能优化，效率优化，变更优化，效率优化，变更管理，监控，紧急事务处理以及容量规划与管理

- ▶ SRE团队的运维工作限制在50%以内，SRE团队应该将剩余时间在研发项目上。
- ▶ 将生产环境中发现Bug和产生的工单转给研发管理人员去分配，或者将开发团队成员加入on-call体系中共同承担轮值压力等。
- ▶ 产品事故都应该对应的事后总结，无论有没有触发警报。

- ▶ 监控系统：
- ▶ SRE团队监控服务质量和可用性的一个主要手段。
所以监控系统的设计策略是针对某个特定的情况或者监控值，一旦出现情况或者监控值超过阈值就触发E-mail警报。但是这样的报警并不是很有效果。

- ▶ 如何提高报警效率

- ▶ 1 紧急警报

- ▶ 意味着收到警报用户需要立即执行某种操作，目标是解决某种已经发生的问题，或者是避免即将发生的问题

- ▶ 2 工单

- ▶ 意味着接受工作单的用户应该执行某种操作，但非立即执行。系统羡慕不能自动解决目前情况，但是如果一个用户几天内执行这项操作，系统 也不会受到任何影响

- ▶ 3 日志

- ▶ 平时没有人去关注日志信息，但是日志信息依然被收集起来以备调试和事后分析时使用。正确的做法是平时没人会去主动阅读日志，除非有特殊需要。

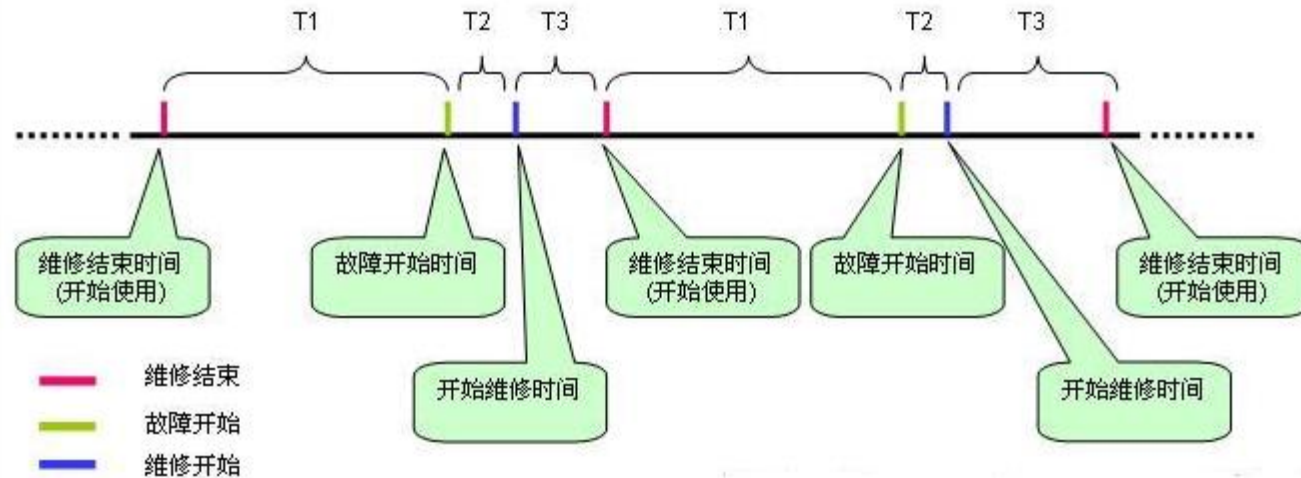
▶ 应急事件处理

▶ MTTF (Mean Time To Failure, 平均无故障时间), 指系统无故障运行的平均时间, 取所有从系统开始正常运行到发生故障之间的时间段的平均值。 $MTTF = \sum T1 / N$

MTTR (Mean Time To Repair, 平均修复时间), 指系统从发生故障到维修结束之间的时间段的平均值。 $MTTR = \sum (T2 + T3) / N$

MTBF (Mean Time Between Failure, 平均失效间隔), 指系统

图解 MTTR、MTTF、MTBF



- ▶ 运维手册 通常可以使MTTR 降低3倍以上。初期几个万能的工程师的确可以解决生产问题，但是长久看来一个手持“运维宝典”经过多次演习的on-call工程师才是正确之路。
- ▶ 变更管理
- ▶ SRE经验告诉我们，70%的生产事故由某种部署的变更而触发。
 - 采用渐进式发布机制
 - 迅速而准确地检测到问题的发生
 - 当出现问题时，安全迅速地回退改动

▶ 需求预测和容量规划

- 必须有一个准确的自然增长需求预测模型，需求预测的时间应该超过资源获取的时间
- 规划中必须有准确的非自然增长的需求来源的统计
- 必须有周期性压力测试，以便准确地将系统原始资源信息与业务容量对应起来

▶ 资源部署

- ▶ 增加现有容量经常需要启动新的实例甚至是整个集群，这通常需要大幅度修改现有的集群配置（配置文件，负载均衡，网络等）同时还要执行一系列测试，确保新上线的容量可以正确地服务用户。

▶ 效率与性能

- 资源使用情况：用户需求，可用容量，软件资源使用效率
- 软件系统一般来说在负载上升的时候，会导致延迟升高。

- ▶ 介绍DevOps
- ▶ 别人家的DevOps
- ▶ **生产环境**
- ▶ 如何进行DevOps
 - 评估 管理
 - 服务质量和目标
 - 减少琐事

- ▶ 硬件
- ▶ 数据中心：供电系统 制冷系统 网络系统 计算机硬件
- ▶ 数据中心拓扑结构：机柜 机柜排 集群 数据中心 园区

▶ 管理服务器的系统管理软件

◦ 管理物理服务器 Mesos

- 负责运行用户提交“任务”，每一个具体的任务分配一个名字和编号。将具体资源分配给每一个任务

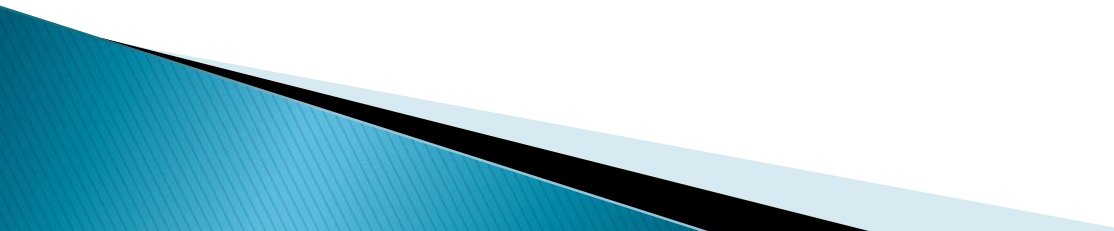
◦ 存储

- 底层服务：磁盘
- 集群文件系统
- 数据库：Nosql，SQL数据库

▶ 网络 SDN

- 高级智能路由器，普通非智能交换组件+控制器
- 网络带宽合理分配。给每一个任务分配计算资源

▶ 其他系统软件：

- 分布式锁服务
 - 监控与警报系统
- 

- ▶ 介绍DevOps
 - ▶ 别人家的DevOps
 - ▶ 生产环境
 - ▶ 如何进行DevOps
 - 评估 管理
 - 服务质量和目标
 - 减少琐事
- 

- ▶ 用户通常不会注意到一项服务在**高可靠性和极端可靠性**之间差异，因为用户体验主要是受较不可靠的组件主导，用户在一个有着99%可靠性的智能手机上是不能分辨出99.99%和99.999%的服务可靠性的区别。
- ▶ 服务的可用性目标通常取决于它提供的功能。
 - 用户期望是什么水平
 - 服务直接关系人
 - 有偿服务还是免费服务
 - 如果市场竞争对手服务水平

▶ 成本

- 可用性目标时，可能考虑以下问题：
- 构建和运维可用性再多一个9，收益会增加多少
- 额外收入是否能够抵消可靠性水平所付出的成本

- ▶ 介绍DevOps
 - ▶ 别人家的DevOps
 - ▶ 生产环境
 - ▶ 如何进行DevOps
 - 评估 管理
 - **服务质量和目标**
 - 减少琐事
- 

- ▶ SLI 服务质量指标：服务质量具体量化指标大部分服务都将请求延迟—处理请求所消耗的时间—作为一个关键SLI，其它常见SLI包括错误率（请求处理失败的百分比）系统吞吐量（每秒请求数量）
- ▶ SLO服务质量目标：定义是 $SLI < \text{目标值}$ 或者 $\text{范围下限} < SLI < \text{范围上限}$ ，要求搜索请求的平均延迟小于100ms
- ▶ 注：全球Chubby服务计划内停机

- ▶ SLO 和SLA的一简单方法是问“如果SLO没有达到时，有什么后果”
- ▶ 运维人员最终用户各关心什么
- ▶ 我们不应该将监控系统中的所有指标都定义为SLI; 只有理解用户对系统的真实需求才能真正决定哪些指标是否有用。

▶ 常见的服务，根据它们的相关SLI通常会归类为以下几个大类

- 用户可见服务系统：是否能正确处理请求？每个请求花费的时间多少？多少请求可以被处理？
- 存储系统通常强调：延迟，可用性和数据持久性。读写数据需要多少时间？是否可以随时访问数据？数据是否一段时间内还能被读取？
- 大数据系统，数据处理流水线系统，一般来说关心吞吐量和端到端延迟。
- 所有的系统都应该关注：正确性。是否返回了正确的回复，是否读取了正确的数据，或者进行了正确的数据分析操作。

- ▶ 汇总
- ▶ 大部分指标都应该以“分布”，而不是平均值来定义
- ▶ 关于统计性谬误
 - SRE更倾向于分析一组数据的百分比分布，而非其算术平均值。长尾效应比算术平均值 更有特点，使用百分比分布能够更清晰地进行分析。
 - 例如，请求延迟必须大于0，同时如果超时设置为1000ms,则不可能有成功请求超过这个时间

► 指标的标准化

- 汇总间隔：每1分钟汇总一次
- 汇总范围：集群中的全部任务
- 度量频率：每10秒一次
- 包含哪些请求：从黑盒监控任务发出HTTP GET 请求
- 数据如何获取：通过监控系统获取服务器端信息得到
- 数据访问延迟：从收到请求到最后一个字节被发出

▶ 目标的定义

- SLO应该具体指出它们是如何被度量的，以及其有效条件，
 - 99%（在1分钟时间内）GET RPC调用小于100ms的时间内完成（包括全部后端服务）
 - 99% GET RPC 会在100ms内完成（这一句与上一句一样，只是利用了SLI 模板中信息减少重复信息）

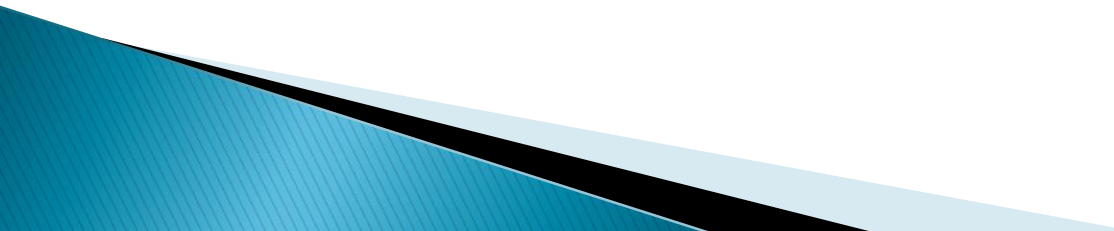
- ▶ 如果性能曲线也很重要的话，我们可以指定多个SLO目标
 - 90% Get RPC 会在1ms内完成
 - 99% GET RPC 会在10ms内完成
 - 99.9%GET RPC会在100ms内完成
- ▶ 可能应该为每种负载指定单独SLO目标
 - 95%批处理用户 SET RPC 应该在1s之内完成
 - 99%交互式用户Set RPC，并且RPC负载小于1K应该在10ms之内完成

- ▶ SLA需要业务部门和法务部门选择合适后果条款，SRE在这个过程中作用帮助部门理解SLA 的SLO达标概率和困难程度。许多针对SLO的建议也同样适用于SLA.最好用户宣传方面保持保守，因为受众越广，修改和删除一个不合适或者很有困难达到SLA就越困难。

- ▶ 介绍DevOps
- ▶ 别人家的DevOps
- ▶ 生产环境
- ▶ 如何进行DevOps
 - 评估 管理
 - 服务质量和目标
 - **减少琐事**

▶ 定义 50%法则

- 手动性
- 重复性的
- 可以被自动化的
- 战术性的（突然出现的）
- 没有持久价值
- 与服务同步线性增长

- ▶ 职业停滞
 - ▶ 士气低落
 - ▶ 造成误解
 - ▶ 进展缓慢
 - ▶ 开创先例（SRE接受工作）
 - ▶ 促进摩擦产生
 - ▶ 违反承诺
- 

温馨提示



- ITIL先锋论坛专家直播讲堂，每周四晚上8:30指定QQ大群
- 专家讲堂视频&PPT合集，请猛击[链接](#)
- 看预告&PPT更新，请关注右边二维码
- 找培训，请看下图：



咨询QQ群
119205977

电话咨询
400 8060 230

基础 - 实战 - 专家

打基础 迎实战 成专家 ITIL先锋为您一站达成

ITIL Expert	¥ 24 万元/人	ISO20000 Auditor	¥ 5.4 千元/人
Prince2 双证	¥ 7.5 千元/人	ITIL Foundation	¥ 2.7 千元/人
ITSS 项目经理	¥ 4.2 千元/人	ISO27001 Foundation	¥ 3.2 千元/人
ITIL 流程实操及 iTOP 软件实施	¥ 2.5 千元/人	PMP 精品班	¥ 1380~4880 元/人
云安全 C-CCSK	¥ 5880 元/人		