

《SRE: Google运维解密》

图书基本信息

书名：《SRE: Google运维解密》

13位ISBN编号：9787121297264

出版时间：2016-10-1

作者：【美】Betsy Beyer（贝特西拜尔）等

页数：480

译者：孙宇聪

版权说明：本站所提供下载的PDF图书仅提供预览和简介以及在线试读，请支持正版图书。

更多资源请访问：www.tushu111.com

《SRE: Google运维解密》

内容概要

大型软件系统生命周期的绝大部分都处于“使用”阶段，而非“设计”或“实现”阶段。那么为什么我们却总是认为软件工程应该首要关注设计和实现呢？在《SRE：Google运维解密》中，Google SRE的关键成员解释了他们是如何对软件进行生命周期的整体性关注的，以及为什么这样做能够帮助Google成功地构建、部署、监控和运维世界上现存最大的软件系统。通过阅读《SRE：Google运维解密》，读者可以学习到Google工程师在提高系统部署规模、改进可靠性和资源利用效率方面的指导思想与具体实践——这些都是可以立即直接应用的宝贵经验。

任何一个想要创建、扩展大规模集成系统的人都应该阅读《SRE：Google运维解密》。《SRE：Google运维解密》针对如何构建一个可长期维护的系统提供了非常宝贵的实践经验。

作者简介

Betsy Beyer 是Google 纽约负责SRE 的一名技术文档作家。她之前曾为遍布全球的Google 数据中心与Mountain View 硬件运维团队编写文档。在搬到纽约之前，Betsy 是Stanford 大学技术性写作课程的讲师。她曾经学习国际关系与英文文学，并在Stanford和Tulane 获得学历。

Chris Jones 是Google App Engine 的一名SRE。Google App Engine 是一个PaaS 服务，每天处理超过280 亿个请求。他的办公室在旧金山，他之前的工作包括Google 广告统计、数据仓库，以及用户支持系统的维护。在之前，Chris 曾经在学校IT 行业任职，同时参与过竞选数据分析，以及一些BSD 内核的修改。他有计算机工程、经济学，以及技术政策学的学位。同时他也是一名有执照的职业工程师。

Jennifer Petoff 是Google SRE 团队的一名项目经理，工作地点在都柏林，爱尔兰。她曾经负责管理大型全球项目，包括：科学研究、工程、人力资源，以及广告等。Jennifer在加入Google 之前，曾在化工行业任职八年。她获得了Stanford 大学的化学博士与学士学位，同时她还拥有Rochester 大学的心理学学位。

Niall Murphy 是Google 爱尔兰团队广告SRE 的负责人。他拥有20 年互联网行业经验，目前是INEX（爱尔兰网络互联枢纽）的主席。他曾经写作以及参与写作很多科技文章与书籍，包括O’Reilly 出版的IPv6 Network Administration，以及很多RFC。他目前在参与书写爱尔兰互联网发展史。他拥有计算机科学、数学，以及诗歌学的学历（他当时一定是想错了！）。他目前与妻子和两个儿子居住在都柏林。

译者

孙宇聪，前Google SRE（2007-2015），山景城总部，曾参与构建运维Youtube 全球CDN网络，2008年奥运会直播项目，构建维护海量视频编码传输系统。后参与Google内部云平台运维工作，负责运维全球百万级别服务器集群，以及Borg、Omega等大规模集群理系统。2015年加入Coding，任CTO一职。回国后，积极推动国内容器化运维架构升级。目前是开放运维联盟之应用运维规范制定组，高可用运维规范制定者。

书籍目录

前言	xxxi
序言	xxxv
第 部分 概览	
第1章 介绍	2
系统管理员模式	2
Google 的解决之道：SRE	4
SRE 方法论	6
确保长期关注研发工作	6
在保障服务SLO的前提下最大化迭代速度	7
监控系统	8
应急事件处理	8
变更管理	9
需求预测和容量规划	9
资源部署	10
效率与性能	10
小结	10
第2章 Google 生产环境：SRE 视角	11
硬件	11
管理物理服务器的系统管理软件	13
管理物理服务器	13
存储	14
网络	15
其他系统软件	16
分布式锁服务	16
监控与警报系统	16
软件基础设施	17
研发环境	17
莎士比亚搜索：一个示范服务	18
用户请求的处理过程	18
任务和数据的组织方式	19
第 部分 指导思想	
第3章 拥抱风险	23
管理风险	23
度量服务的风险	24
服务的风险容忍度	25
辨别消费者服务的风险容忍度	26
基础设施服务的风险容忍度	28
使用错误预算的目的	30
错误预算的构建过程	31
好处	32
第4章 服务质量目标	34
服务质量术语	34
指标	34
目标	35
协议	36
指标在实践中的应用	37
运维人员和最终用户各关心什么	37

指标的收集	37
汇总	38
指标的标准化	39
目标在实践中的应用	39
目标的定义	40
目标的选择	40
控制手段	42
SLO 可以建立用户预期	42
协议在实践中的应用	43
第5章 减少琐事	44
琐事的定义	44
为什么琐事越少越好	45
什么算作工程工作	46
琐事繁多是不是一定不好	47
小结	48
第6章 分布式系统的监控	49
术语定义	49
为什么要监控	50
对监控系统设置合理预期	51
现象与原因	52
黑盒监控与白盒监控	53
4个黄金指标	53
关于长尾问题	54
度量指标时采用合适的精度	55
简化,直到不能再简化	55
将上述理念整合起来	56
监控系统的长期维护	57
Bigtable SRE : 警报过多的案例	57
Gmail : 可预知的、可脚本化的人工干预	58
长跑	59
小结	59
第7章 Google 的自动化系统的演进	60
自动化的价值	60
一致性	60
平台性	61
修复速度更快	61
行动速度更快	62
节省时间	62
自动化对Google SRE 的价值	62
自动化的应用案例	63
Google SRE 的自动化使用案例	63
自动化分类的层次结构	64
让自己脱离工作: 自动化所有的东西	66
舒缓疼痛: 将自动化应用到集群上线中	67
使用Prodtest 检测不一致情况	68
幂等地解决不一致情况	69
专业化倾向	71
以服务为导向的集群上线流程	72
Borg : 仓库规模计算机的诞生	73

可靠性是最基本的功能	74
建议	75
第8章 发布工程	76
发布工程师的角色	76
发布工程哲学	77
自服务模型	77
追求速度	77
密闭性	77
强调策略和流程	78
持续构建与部署	78
构建	78
分支	79
测试	79
打包	79
Rapid 系统	80
部署	81
配置管理	81
小结	82
不仅仅只对Google 有用	83
一开始就进行发布工程	83
第9章 简单化	85
系统的稳定性与灵活性	85
乏味是一种美德	86
我绝对不放弃我的代码	86
“负代码行”作为一个指标	87
最小 API	87
模块化	87
发布的简单化	88
小结	88
第 部分 具体实践	
第10章 基于时间序列数据进行有效报警	93
Borgmon 的起源	94
应用软件的监控埋点	95
监控指标的收集	96
时间序列数据的存储	97
标签与向量	98
Borg 规则计算	99
报警	104
监控系统的分片机制	105
黑盒监控	106
配置文件的维护	106
十年之后	108
第11章 on-call 轮值	109
介绍	109
on-call 工程师的一天	110
on-call 工作平衡	111
数量上保持平衡	111
质量上保持平衡	111
补贴措施	112

安全感	112
避免运维压力过大	114
运维压力过大	114
奸诈的敌人—运维压力不够	115
小结	115
第12章 有效的故障排查手段	116
理论	117
实践	119
故障报告	119
定位	119
检查	120
诊断	122
测试和修复	124
神奇的负面结果	125
治愈	126
案例分析	127
使故障排查更简单	130
小结	130
第13章 紧急事件响应	131
当系统出现问题时怎么办	131
测试导致的紧急事故	132
细节	132
响应	132
事后总结	132
变更部署带来的紧急事故	133
细节	133
事故响应	134
事后总结	134
流程导致的严重事故	135
细节	135
灾难响应	136
事后总结	136
所有的问题都有解决方案	137
向过去学习，而不是重复它	138
为事故保留记录	138
提出那些大的，甚至不可能的问题：假如……	138
鼓励主动测试	138
小结	138
第14章 紧急事故管理	140
无流程管理的紧急事故	140
对这次无流程管理的事故的剖析	141
过于关注技术问题	141
沟通不畅	141
不请自来	142
紧急事故的流程管理要素	142
嵌套式职责分离	142
控制中心	143
实时事故状态文档	143
明确公开的职责交接	143

一次流程管理良好的事故	144
什么时候对外宣布事故	144
小结	145
第15章 事后总结：从失败中学习	146
Google 的事后总结哲学	146
协作和知识共享	148
建立事后总结文化	149
小结以及不断优化	151
第16章 跟踪故障	152
Escalator	152
Outalator	153
聚合	154
加标签	155
分析	155
未预料到的好处	156
第17章 测试可靠性	157
软件测试的类型	158
传统测试	159
生产测试	160
创建一个构建和测试环境	163
大规模测试	165
测试大规模使用的工具	166
针对灾难的测试	167
对速度的渴求	168
发布到生产环境	170
允许测试失败	170
集成	172
生产环境探针	173
小结	175
第18章 SRE 部门中的软件工程实践	176
为什么软件工程项目对SRE 很重要	176
Auxon 案例分析：项目背景和要解决的问题	177
传统的容量规划方法	177
解决方案：基于意图的容量规划	179
基于意图的容量规划	180
表达产品意图的先导条件	181
Auxon 简介	182
需求和实现：成功和不足	183
提升了解程度，推进采用率	185
团队内部组成	187
在SRE 团队中培养软件工程风气	187
在SRE 团队中建立起软件工程氛围：招聘与开发时间	188
做到这一点	189
小结	190
第19章 前端服务器的负载均衡	191
有时候硬件并不能解决问题	191
使用DNS 进行负载均衡	192
负载均衡：虚拟IP	194
第20章 数据中心内部的负载均衡系统	197

- 理想情况 198
- 识别异常任务：流速控制和跛脚鸭任务 199
- 异常任务的简单应对办法：流速控制 199
- 一个可靠的识别异常任务的方法：跛脚鸭状态 200
- 利用划分子集限制连接池大小 201
- 选择合适的子集 201
- 子集选择算法一：随机选择 202
- 子集选择算法二：确定性算法 204
- 负载均衡策略 206
- 简单轮询算法 206
- 最闲轮询策略 209
- 加权轮询策略 210
- 第21章 应对过载 212
- QPS 陷阱 213
- 给每个用户设置限制 213
- 客户端侧的节流机制 214
- 重要性 216
- 资源利用率信号 217
- 处理过载错误 217
- 决定何时重试 218
- 连接造成的负载 220
- 小结 221
- 第22章 处理连锁故障 223
- 连锁故障产生的原因和如何从设计上避免 224
- 服务器过载 224
- 资源耗尽 225
- 服务不可用 228
- 防止软件服务器过载 228
- 队列管理 229
- 流量抛弃和优雅降级 230
- 重试 231
- 请求延迟和截止时间 234
- 慢启动和冷缓存 236
- 保持调用栈永远向下 238
- 连锁故障的触发条件 238
- 进程崩溃 239
- 进程更新 239
- 新的发布 239
- 自然增长 239
- 计划中或计划外的不可用 239
- 连锁故障的测试 240
- 测试直到出现故障，还要继续测试 240
- 测试最常用的客户端 241
- 测试非关键性后端 242
- 解决连锁故障的立即步骤 242
- 增加资源 242
- 停止健康检查导致的任务死亡 242
- 重启软件服务器 242
- 丢弃流量 243

进入降级模式	243
消除批处理负载	244
消除有害的流量	244
小结	244
第23章 管理关键状态：利用分布式共识来提高可靠性	246
使用共识系统的动力：分布式系统协调失败	248
案例1：脑裂问题	249
案例2：需要人工干预的灾备切换	249
案例3：有问题的小组成员算法	249
分布式共识是如何工作的	250
Paxos 概要：协议示例	251
分布式共识的系统架构模式	251
可靠的复制状态机	252
可靠的复制数据存储和配置存储	252
使用领头人选举机制实现高可用的处理系统	253
分布式协调和锁服务	253
可靠的分布式队列和消息传递	254
分布式共识系统的性能问题	255
复合式Paxos：消息流过程详解	257
应对大量的读操作	258
法定租约	259
分布式共识系统的性能与网络延迟	259
快速Paxos 协议：性能优化	260
稳定的领头人机制	261
批处理	262
磁盘访问	262
分布式共识系统的部署	263
副本的数量	263
副本的位置	265
容量规划和负载均衡	266
对分布式共识系统的监控	270
小结	272
第24章 分布式周期性任务系统	273
Cron	273
介绍	273
可靠性	274
Cron 任务和幂等性	274
大规模Cron 系统	275
对基础设施的扩展	275
对需求的扩展	276
Google Cron 系统的构建过程	277
跟踪Cron 任务的状态	277
Paxos 协议的使用	277
领头人角色和追随者角色	278
保存状态	281
运维大型Cron 系统	282
小结	283
第25章 数据处理流水线	284
流水线设计模式的起源	284

简单流水线设计模式与大数据	284
周期性流水线模式的挑战	285
工作分发不均造成的问题	285
分布式环境中周期性数据流水线的缺点	286
监控周期性流水线的问题	287
惊群效应	287
摩尔负载模式	288
Google Workflow 简介	289
Workflow 是模型—视图—控制器 (MVC) 模式	290
Workflow 中的执行阶段	291
Workflow 正确性保障	291
保障业务的持续性	292
小结	294
第26章 数据完整性：读写一致	295
数据完整性的强需求	296
提供超高的数据完整性的策略	297
备份与存档	298
云计算环境下的需求	299
保障数据完整性和可用性：Google SRE 的目标	300
数据完整性是手段，数据可用性是目标	300
交付一个恢复系统，而非备份系统	301
造成数据丢失的事故类型	301
维护数据完整性的深度和广度的困难之处	303
Google SRE 保障数据完整性的手段	304
24 种数据完整性的事故组合	304
第一层：软删除	305
第二层：备份和相关的恢复方法	306
额外一层：复制机制	308
1T vs. 1E：存储更多数据没那么简单	309
第三层：早期预警	310
确保数据恢复策略可以正常工作	313
案例分析	314
Gmail—2011年2月：从GTape上恢复数据（磁带）	314
Google Music—2012年3月：一次意外删除事故的检测过程	315
SRE 的基本理念在数据完整性上的应用	319
保持初学者的心态	319
信任但要验证	320
不要一厢情愿	320
纵深防御	320
小结	321
第27章 可靠地进行产品的大规模发布	322
发布协调工程师	323
发布协调工程师的角色	324
建立发布流程	325
发布检查列表	326
推动融合和简化	326
发布未知的产品	327
起草一个发布检查列表	327
架构与依赖	328

集成	328
容量规划	328
故障模式	329
客户端行为	329
流程与自动化	330
开发流程	330
外部依赖	331
发布计划	331
可靠发布所需要的方法论	332
灰度和阶段性发布	332
功能开关框架	333
应对客户端滥用行为	334
过载行为和压力测试	335
LCE的发展	335
LCE 检查列表的变迁	336
LCE 没有解决的问题	337
小结	338
第 部分 管理	
第28章 迅速培养SRE 加入on-call	341
新的SRE 已经招聘到了，接下来怎么办	341
培训初期：重体系，而非混乱	344
系统性、累积型的学习方式	345
目标性强的项目工作，而非琐事	346
培养反向工程能力和随机应变能力	347
反向工程：弄明白系统如何工作	347
统计学和比较性思维：在压力下坚持科学方法论	347
随机应变的能力：当意料之外的事情发生时怎么办	348
将知识串联起来：反向工程某个生产环境服务	348
有抱负的on-call 工程师的5个特点	349
对事故的渴望：事后总结的阅读和书写	349
故障处理分角色演习	350
破坏真的东西，并且修复它们	351
维护文档是学徒任务的一部分	352
尽早、尽快见习on-call	353
on-call 之后：通过培训的仪式感，以及日后的持续教育	354
小结	354
第29章 处理中断性任务	355
管理运维负载	356
如何决策对中断性任务的处理策略	356
不完美的机器	357
流状态	357
将一件事情做好	358
实际一点的建议	359
减少中断	361
第30章 通过嵌入SRE 的方式帮助团队从运维过载中恢复	363
第一阶段：了解服务，了解上下文	364
确定最大的压力来源	364
找到导火索	364
第二阶段：分享背景知识	365

书写一个好的事后总结作为示范	366
将紧急事件按类型排序	366
第三阶段：主导改变	367
从基础开始	367
获取团队成员的帮助	367
解释你的逻辑推理过程	368
提出引导性问题	368
小结	369
第31章 SRE 与其他团队的沟通与协作	370
沟通：生产会议	371
议程	372
出席人员	373
SRE 的内部协作	374
团队构成	375
高效工作的技术	375
SRE 内部的协作案例分析：Viceroy	376
Viceroy 的诞生	376
所面临的挑战	378
建议	379
SRE 与其他部门之间的协作	380
案例分析：将DFP 迁移到F1	380
小结	382
第32章 SRE 参与模式的演进历程	383
SRE 参与模式：是什么、怎么样以及为什么	383
PRR 模型	384
SRE 参与模型	384
替代性支持	385
PRR：简单PRR 模型	386
参与	386
分析	387
改进和重构	387
培训	388
“接手”服务	388
持续改进	388
简单PRR 模型的演进：早期参与模型	389
早期参与模型的适用对象	389
早期参与模型的优势	390
不断发展的服务：框架和SRE 平台	391
经验教训	391
影响SRE 的外部因素	392
结构化的解决方案：框架	392
新服务和管理优势	394
小结	395
第 部分 结束语	
第33章 其他行业的实践经验	398
有其他行业背景的资深SRE	399
灾难预案与演习	400
从组织架构层面坚持不懈地对安全进行关注	401
关注任何细节	401

冗余容量	401
模拟以及进行线上灾难演习	402
培训与考核	402
对详细的需求收集和系统设计的关注	402
纵深防御	403
事后总结的文化	403
将重复性工作自动化，消除运维负载	404
结构化和理性的决策	406
小结	407
第34章 结语	408
附录A 系统可用性	411
附录B 生产环境运维过程中的最佳实践	412
附录C 事故状态文档示范	417
附录D 事后总结示范	419
附录E 发布协调检查列表	423
附录F 生产环境会议记录示范	425
参考文献	427
索引	439__

《SRE: Google运维解密》

精彩短评

- 1、详细解释了google sre方面的理论和实践
- 2、很快速的翻了一遍，SRE观点应该是每个进行系统研发和运行的团队都应该关注的，也是每个工程师都应该关注的。主要讲了Google实践的一些观点、方法论和实践经验
- 3、SRE百科全书，每个人都能得到自己的收获，Google实现自动化之路是将所有东西都封装成API，包括屏蔽Shell脚本。
- 4、作为某大互联网公司的运维，国内没有公司将运维做的这么规范，这么好，google的sre也是我们一直努力的方向。这本书也算是传统op向devops转型的指南。
- 5、对系统设计有一定启发
- 6、值得放在办公桌边，经常翻起审视和改进自己公司的业务。
- 7、其实不用那么神话，整本书有些章节非常的啰嗦还说不清楚，但是开头结尾不错。记住三点：第一，SRE都是通过google开发笔试的人，这保障了开发的效率和质量。第二，良好的体制，on-call机制和SLO机制。第三，非常重视事后总结，多个章节有提到。
- 8、买一本放在工位上，常常审视自己的项目是否到这种标准
- 9、粗读，没有读完，内容很多，自己功力不够
- 10、很不错的索引书
- 11、非常受用，连锁故障部分需再消化
- 12、作为一名OP，正在向OPs以及「FRE」转变。
- 13、绝不只是给运维读的书，所有的IT工程师应该都看看。
- 14、任何一个架构和程序设计没有从运维角度进行思考都是错误的
- 15、谷歌给我们的运维工作证名了，有规划了

1、原文来自：<http://blog.csdn.net/xindoo/article/details/52723114> 《SRE》这本书英文版已面世半年后，中文版终于面世。从4月、5月的时候，我就一直在尝试看英文版，由于自己英文水平有限，阅读进度和深度实在有限，看到中文版，对很多章节的内容才算是有了较深入的理解，一句话评价此书，这是一本运维转型的指导性书。看过原版，再对照中文版，从内容上，并不比原版少什么，所以各位读者不必担心内容相对原版是否缺失，如果各位英语不好、但又想了解Google的SRE，放心大胆的买中文版吧，因为译者也是Google的前SRE，翻译的不能说原汁原味，但也八九不离十。我自己本身也在国内某大厂做运维，我们也面临着传统运维向devops的转型，接下来我就结合自己实际工作的经历，谈谈我对这本说的理解。这本书基本上可以分成几个大部分。* SRE的诞生 * Google内部软硬件环境 * SRE和Dev的协作 * SRE自己是如何做事的 SRE是为了解决op和dev相互之间的矛盾和割裂的问题，用一些工程和规范来让op和dev之间有个平衡，并且最优化系统的发展。书中举出大量dev和sre系统的方法和规范，比如错误预算、部分运维工作交还dev、SRE协助dev团队健康发展等..... 从我自己的经验来看，其实作为一个op，一天到晚有一堆乱七八糟的事，曾经因为这些事，搞的我情绪都不太好。不同于国内一些公司，google考虑到了这些，制定了一系列的标准来平衡SRE工作上的问题，比如最多50%运维工作、完善的轮值机制、完善的SRE培训体系.....，前两天还看过google的《重新定义公司》，从他们内部各种福利政策来看，google是一个非常人性化的公司。运维工作中，有些是管理层需要做的事，但也有些内容能让你自己提升自己运维的效率。这么多年，SRE总结出了一套完善的方法论，比如和Dev团队的协作沟通，SRE在风险管理、on-call、故障排查、问题处理、故障后总结.....，google都总结出了想当好的经验。书中也介绍了google的一些软硬件环境，比如数据中心、网络、borg、Chubby (zookeeper)、监控系统、负载均衡、cron等。书中介绍了一些这类软硬件设计的思路，可以给那些想自己设计软硬件系统的公司一些方向。> We want our systems that are automatic, not just automated. >上面这句话是英文版原文，如何理解这句话?我们想让要系统是自治的，而不仅仅是自动的。这是一个设计系统先进的理念，想想我们往常是怎么设计系统的，是不是专注于解决一个问题，流程在这里卡了，需要人为干预，甚至是再做一个新的系统来解决某些问题。举个例子，一个应用有数十台服务器，服务器会宕机，然后需要把服务器下线，再扩容一台。然后就会有一个监控系统发现问题，再弄个服务器下线的系统，自动化扩容的系统，然后都需要手工提单。这个时候，有人嫌提单麻烦，又写了个自动调其他三个系统解决问题的系统。就因为服务不是自治的，而我们一味强调自动化，导致系统越来越复杂，越来越难以自动化。其实我举的这个例子，google的borg已经很好的解决了，我认为borg基本上就是一个自治的系统。总结一把，我觉得这本书并不是直接告诉你应该怎么做，因为不同的公司在不同的阶段关注的重点是不一样的，做的事也不可能和google相同，盲从某些方法论可能会得到相反的结果，所以我的建议是把这本书当成一种方向性的指导。

章节试读

1、《SRE: Google运维解密》的笔记-第15章，事后总结：从失败中学习

事后总结的主要目的是为了保证该事故被记录下来，理清所有的根源性问题，确保实施的有效措施在未来重现的几率降低甚至避免重现。基本的时候总结条件为：1. 用户可见的宕机时间或者服务质量降级程度达到一定标准2. 任何类型的数据丢失3. on-call工程师需要人工介入的事故4. 问题解决耗时过长5. 监控是由人工发现的，而非警报系统事后总结应该对事不对人，一篇事后总结的重点必须是关注如果造成这次事件的根本问题，而不是指责某个人或某团队的错误或者不恰当的举动。如果时候总结的重点是指责某人的错误举动，那人们就会逃避时候总结，模糊事情的原因。引入时候总结机制的时候，最大的阻力来源于对投入产出比的质疑。帮助面对挑战的策略：1. 逐渐引入事后总结流程，首先进行几次完整的和成功的事后总结，证明它们的价值，同时帮助确定具体书写事后总结的条件。2. 确保对有效的书面总结提供奖励和庆祝。不光通过前面提到的公开渠道，也应该在团队或个人的绩效考核中体现。3. 鼓励公司高级管理层认可其中

2、《SRE: Google运维解密》的笔记-第184页

任何足够复杂的软件工程项目都会遇到一定程度的不确定性，要么是组件的设计方法不确定，要么是问题的解决方式不确定。

3、《SRE: Google运维解密》的笔记-序言

可靠性就像安全性，越早关注越好。

4、《SRE: Google运维解密》的笔记-第1页

只有靠着对细节的不懈关注，做好充足的灾难预案和准备工作，时刻警惕着，不放过一切机会去避免灾难发生。— 这是SRE最重要的理念

5、《SRE: Google运维解密》的笔记-第118页

值得注意的是，在某些系统中某一类问题可能全被排除了。例如，在细心设计的集群文件系统实现中，由于某个磁盘出现问题导致延迟问题是非常不可能的。

6、《SRE: Google运维解密》的笔记-第23页

* 目标

* 没有 100% 可靠的服务, 达到一定程度的可靠性之后, 应把精力转向他处.

* ** ” 当设立了一个可用性目标为99.99%时, 我们即使要超过这个目标, 也不会超过太多, 否则会浪费为系统增加新功能, 清理技术债务或者降低运营成本的机会. ” **

* 可靠性目标成为错误预算: 提供明确和客观的指标决定服务在一个季度中接受多少不可靠性(用于 SRE 部门和产品部门的沟通). 只要错误预算耗尽, 新版本的发布就会暂停(?但是错误率由 SRE 部门提供, 而发布由产品决定?) -> 认为风险由产品开发决定, 一个变通是, 当错误预算即将用尽时, 降低发布的频率. 即使是网络中断或者数据中心故障影响了错误率, 发布频率也会降低, 因为 ” 每个人 ” 都有义务保障服务的正常运行.

* 可用性指标: 请求成功率. **用我们记录的请求成功率与用户期望的服务水平做对比.**

* 成本

* 可用性: 99.9% 到 99.99%; 收入: 1000000刀 -> 改进后的价值: $1000000 * 0.09\% = 900$ 刀

* 需求

* 面向消费者需要低延迟(队列空为好), 离线计算需要吞吐量(队列满为好). 需要分别响应不同的需求.
-> 两个集群, 低延迟/高吞吐量

7、《SRE: Google运维解密》的笔记-第71页

某个最主要任务是加速现存的集群上线的团队是没有动力去减少服务运维团队在生产流程后期运维服务产生的技术负债的。

一个不亲自运行自动化的团队是没有动力去建设一个能够很容易自动化的系统的。

一个产品经理的时间表如果不受低质量的自动化影响, 他将永远优先新功能的开发, 而不是简化和自动化。

8、《SRE: Google运维解密》的笔记-第3页

研发部门最关注的是如何能够更快速地构建和发布新功能。运维部门更关注的是如何能在他们值班期间避免发生故障。研发部门想要：“随时随地发布新功能，没有任何阻拦”，而运维部门则想要：“一旦一个东西在生产环境中正常工作了，就不要再进行任何改动。”

运维团队宣称，任何变更上线前必须经过运维团队制定的流程，这有助于避免事故的发生。研发团队吃过苦头之后则宣称不再进行大规模的程序更新，而是……

9、《SRE: Google运维解密》的笔记-第一章

监控系统不应该依赖人来分析报警信息，而应该由系统自动分析，仅当需要用户执行操作的时候，才需要通知用户

10、《SRE: Google运维解密》的笔记-第36页

我们在这里采用了一个很有趣的解决办法：SRE保证全球Chubby服务能够达到预定义的SLO，但是同时也会确保服务质量不会大幅超出该SLO。每个季度，如果真实故障没有将可用性指标降低到SLO之下，SRE会有意安排一次可控的故障，将服务停机。利用这种方法，我们可以很快找出那些对Chubby全球服务的不合理依赖，强迫服务的负责人尽早面对这类分布式系统的天生缺陷。

11、《SRE: Google运维解密》的笔记-第8页

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。

12、《SRE: Google运维解密》的笔记-第86页

鼓励删代码，统计源代码库里删除代码的行数

13、《SRE: Google运维解密》的笔记-第208页

无法预知的性能因素：坏邻居、任务重启

14、《SRE: Google运维解密》的笔记-第39页

我们应该从思考（或者调研）用户最关心的方面入手，而非从现在能度量什么入手。

15、《SRE: Google运维解密》的笔记-第6页

可用性改进，延迟优化，性能优化，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。

16、《SRE: Google运维解密》的笔记-第85页

我们的工作最终是在系统的灵活性和稳定性上维持平衡。

17、《SRE: Google运维解密》的笔记-第44页

Google员工每周要写一个短小的无固定格式的工作总结，称为“Snippets”。

18、《SRE: Google运维解密》的笔记-第16页

主要有以下几种方式使用监控系统：

对真实问题进行报警。

对比服务更新前后的状态变化：新的版本是否让软件服务器运行得更快了？

检查资源使用量随时间的变化情况，这个信息对合理制定资源计划很有用。

19、《SRE: Google运维解密》的笔记-第71页

自动化代码和单元测试代码一样，当维护团队不再关心它与它所覆盖的代码仓库同步的时候就会逐渐死去。

20、《SRE: Google运维解密》的笔记-第58页

两个例子的决策都很值得思考

21、《SRE: Google运维解密》的笔记-第12页

Google并不会使用专门的物理服务器运行专门的软件服务器。例如，并不存在一个具体的物理服务器运行我们的邮件系统，而是采用一套集群管理系统进行资源分配，它的名称为Borg。

22、《SRE: Google运维解密》的笔记-第118页

相关性（correlation）不等于因果关系（causation）。

23、《SRE: Google运维解密》的笔记-第1页

不能将碰运气当成战略。

24、《SRE: Google运维解密》的笔记-第79页

所有代码都默认提交到主分支上（mainline），然而大部分的项目都不会直接从主分支上进行直接发布。我们会基于主分支的某一个版本创建新分支，新分支的内容永远不会再合并入主分支。Bug修复先提交到主分支，再cherry picking到发布分支上。这种方式可以避免在第一次构建之后，再引入主

分支上的其他的无关改动。利用这种分支与cheery picking的方法，可以明确每个发布版本中包含的全部改动。

25、《SRE: Google运维解密》的笔记-第34页

- * 质量度量
 - * 请求延迟 (Req time)
 - * 错误率 (Web errors)
 - * 吞吐量 (Web QPS) -> **分应用展示?*
 - * Google 云计算的可用性指标: 99.95% -> $60 \times 24 \times 365 \times 0.0005 = 262.8 \text{ min/year}$ -> 我们的可用性?
- * 特色
 - * 4/5个指标, 多/少都不好.
 - * 监控, y 轴指数分布
 - * 数据收集每10秒一次, 每一分钟汇总一次. 目标像这样: **99% 的 get RPC 调用在 < 100ms 的时间内完成.** 每天可以出一个这样的报表.
- * 总结
 - * 指标越少越好, 少到不能更少
 - * 性能指标保持简单
 - * 从松散的目标开始, 逐渐收紧. 不要一开始就追求完美
 - * 对内指标要求可以比对外高一些, 留有余地

26、《SRE: Google运维解密》的笔记-第11页

- * 集群资源分配: Borg(分布式集群操作系统), 下一代 Kubernetes(2014)
- * Large-scale cluster management at Google with Borg
- * Borg, Omega, and Kubernetes
- * 负责运行用户提交的任务. 每个任务由多个实例组成, Borg 会为每一个实例安排一台物理服务器, 执行具体的程序启动它
 - * 负责任务的监控, 如果异常, 终止并重启
 - * 命名: BNS: /bns/<cluster>/<user>/<task>/<instance>
 - * 任务需要在配置中声明其所需的具体资源(cpu/mem), 超过则立即 kill
- * 存储
 - * 分布式存储, 小文件和大文件进不同的集群.
 - * 单个集群一年内会损失上千块硬盘, 数据中心有专门的团队来处理
- * 网络
 - * 这些概念比较陌生, OpenFlow 的软件定义网络, 带宽控制器优化带宽.
 - * 从地理位置, 用户服务和远程调用三层进行负载均衡
- * 监控报警
 - * 定时抓取指标, 超出触发报警
 - * **新旧版本的对比: 新版本是否让软件服务器更快了?*
 - * 检查资源用量随时间的变化, 制定资源计划.
- * 服务

《SRE: Google运维解密》

- * 所有服务使用 RPC 通信, 开源实现为 gRPC
- * 格式为 Protocol Buffer(与 Apache Thrift 相比) (大小比 xml 小 3-10 倍, 序列化/反序列化快 100 倍) (和 json 比?)
- * 服务和存储根据流量分散到各大洲的机房

- * 开发
- * Code review

27、《SRE: Google运维解密》的笔记-第1页

- * DevOps 在 Google 的实践

传统开发/运维分离的解决方案在规模扩大后沟通成本上升(“随时发布” vs. “不再改动”) -> 新型运维团队 SRE(50%-60%标准开发, 其他为85%-99%能力的开发, 为了开发系统代替手工操作) -> 最多 50% 时间用于运维工作, 余下开发系统来自动化

* SRE 方法论

- * 运维工作最多占用 50% 时间
- * 遇到故障事后写总结
- * 因为信息系统的特点, 不是也不该追求 100% 可靠, 给出现实的可靠性. 在实现这个可靠性的前提下, SRE 可以做各种创新
- * 监控, 通过预案/手册缩短平均恢复时间
- * 70% 的事故源于部署变更 -> 渐进发布, 精确检测, 回滚机制

28、《SRE: Google运维解密》的笔记-第42页

理解系统行为与预期的符合程度可以帮助决策是否需要投入力量优化系统, 使其速度更快、更可用, 或者更可靠。如果服务一切正常, 可能力量应该花在其他的优先级上, 例如消除技术债务、增加新功能, 或者引入其他产品等。

29、《SRE: Google运维解密》的笔记-第246页

理解如何构建分布式共识实际上就是理解某个服务应用如何实现强一致性和高可用性。

30、《SRE: Google运维解密》的笔记-第7页

事后总结应该包括以下内容: 事故发生、发现、解决的全过程、事故的根本原因, 预防或者优化的解决方案。

31、《SRE: Google运维解密》的笔记-第14页

一个缓慢的不断重启的实例要好过一个永远不重启一直泄露资源的实例。

32、《SRE: Google运维解密》的笔记-第120页

在日志中支持多级记录是很重要的, 尤其是可以在线动态调整日志级别。

33、《SRE: Google运维解密》的笔记-第76页

从代码修改提交到部署到生产环境一共需要多长时间（也就是发布速度）

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:www.tushu111.com