货拉拉技术稳定性保障实践

货拉拉
已认证帐号

2021年10月22日，在云栖大会的《云上运维最佳实践》分论坛，货拉拉技术副总监陈永庭发表了主题为“基于云的货拉拉技术稳定性保障实践”的演讲，为大家分享了货拉拉在过去一段时间是如何做到技术稳定性保障的，希望给有同类型业务场景的同行提供一种思考方式。

图：货拉拉技术副总监陈永庭

以下是根据他的演讲整理成的文章，主要分为四个部分：

一、货拉拉业务形态。

二、基础架构治理。

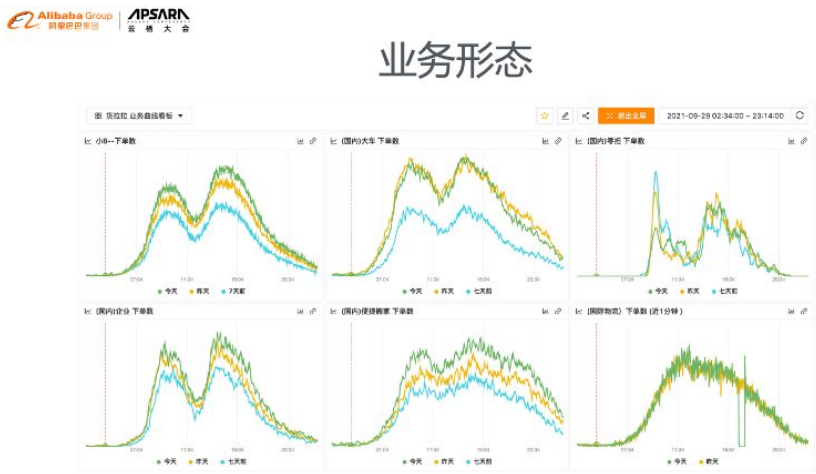
三、技术保障能力的建设。

四、跨云的思考与实施。

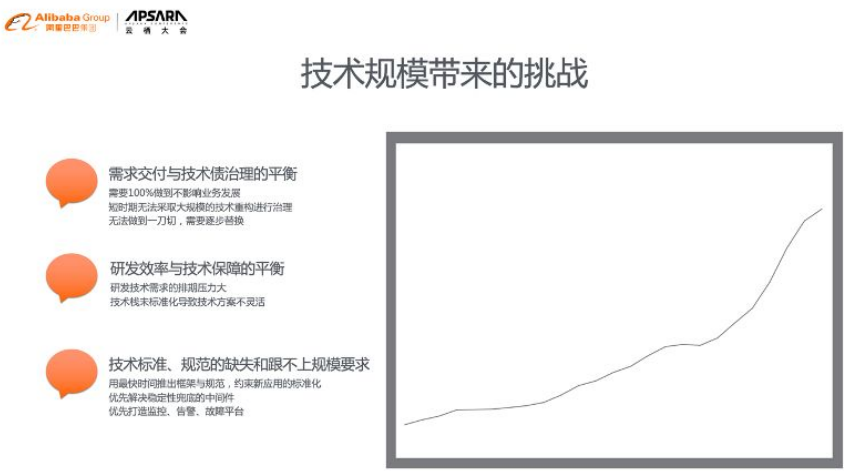
一、货拉拉业务形态

货拉拉从成立到现在已经有接近十年的时间，从2013年开始我们主要做同城货运与国际货运。到现在我们有企业、跨城、搬家、零担，多条业务线。这些业务线有共性，也有不一样的地方。在技术系统的容量保障上，与业务特征有很强的关系。平台提供司机运力的供给，帮助配对用户货物的订单需求。

所以，在供需出现不平衡的时候，会对我们的系统带来一些压力，譬如：运力不足；用户反复下单；调度系统努力帮助订单寻找配对的运力；这期间会产生大量的计算需求，系统的容量峰值会是正常峰值的好几倍，但我们系统常态下没有足够的空闲容量。



这是货拉拉的各个业务的流量形态图。业务形态是典型的高峰低峰的分布。这是正常的状态图，但不正常的情况也有很多。比如有的业务线周一到周五是流量比较小的，周末流量是平时的好几倍。还有在节假日前夕，出现很多司机运力不足的问题。这个时候，我们的取消单、待配对单等指标就会增长，我们的司机调度系统的也会频繁的告警。所以在过去1-2年里，我们会重点关注，在当前的业务形态下，技术系统和系统容量如何去设计？我们在日常保障中如何合理的评估、压测系统的容量？



在过去1-2年里，我们的技术团队规模快速增长，同时我们也遇到了一些问题需要去解决。首先，我们的研发同学不但要快速地交付业务需求，而且要清还历史技术的债务，尤其是前期，债务比较

另外，我们非常关注研发效率与技术保障之间的平衡。前期，在很多领域，我们会优先通过技术手段来提升普适性场景的效率，并且不需要业务研发同学投入太多时间（避免影响他们的业务交付效率）。如果要做到足够好的技术稳定性保障，研发同学需要在自己的服务代码里加上更多额外的保护措施。这个工作量是比较大的，对一些业务需求交付节奏比较快的团队来说是很难的。举个例子：我们最早期监控框架SDK在研发接入的时候，不需要研发人员修改代码，只需要把JAR包集成到服务里。

最后，在技术规模快速增长的过程中，我们的技术标准，落后于我们的诉求。公司内部出现较多非标准化的技术操作。在这个时期，我们会考虑一些优先级比较高的事情先做。比如说，我们会在特定的时间里，优先解决监控问题、告警问题、故障预案平台的问题，然后再去解决其他非标领域的事情。

二、基础架构治理

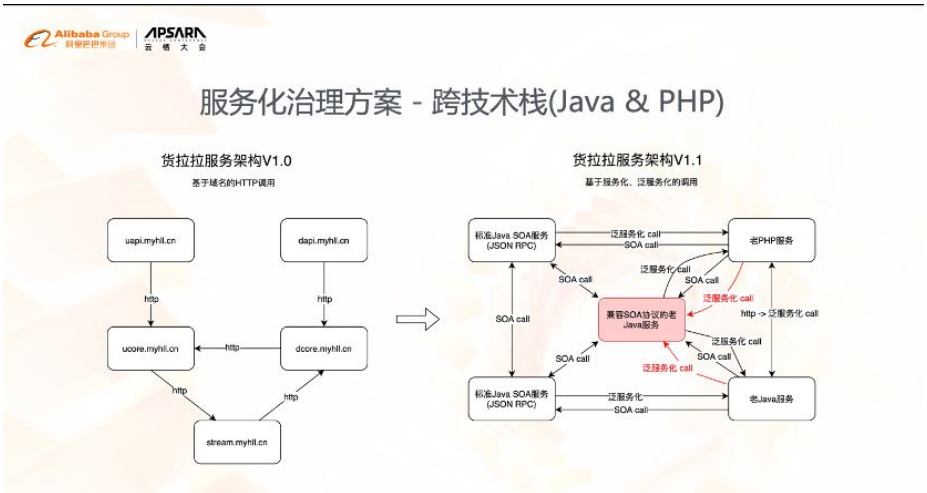
货拉拉的基础架构是如何治理的呢？如何通过基础架构的治理来提升基础的稳定性?前期我们主要聚焦在两个方面：一是货拉拉结合自身情况去做服务化治理，其次，如何让研发同事在相对可控、有保障的前提下，支撑他们方便快捷的发布？甚至可以在高峰期时间发布，且不会出现无法预料的风险。

下图是货拉拉最早期的技术架构版本。它实际上很简单，支持快速开发，支撑快速交付。由于我们的研发工程师效率非常高，今天很快可以把业务需求的代码写完，明天就能测试上线。这种方式快速支撑了业务的高速发展，但这种“快”也带来一些隐患。当我们的业务规模、数据规模、服务规模在百万订单级别下发生了很大的变化。“快”引发的一些问题变得越来越凸显。

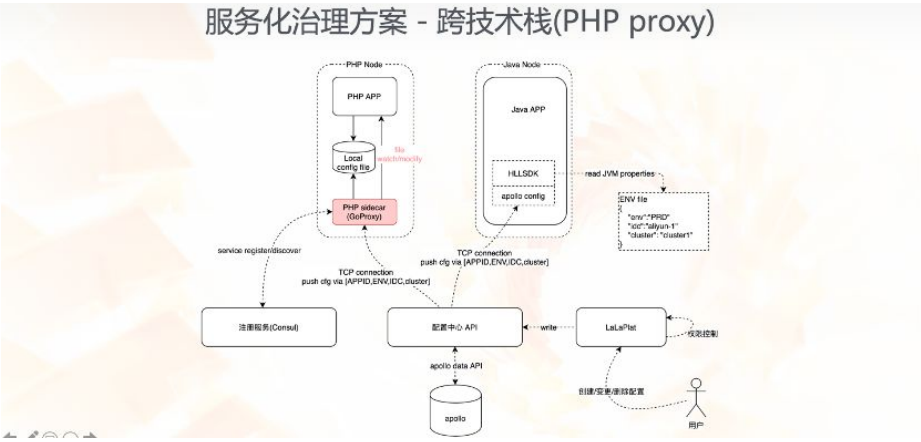
首先，业务链路服务不可靠；关键服务与非关键服务之间不清晰，并且相互依赖。这很容易因非关键服务导致整个主链路发生故障。**其次，部分核心服务臃肿庞大**，不易维护。**第三，服务自愈能力比较弱**，在瞬间几倍高峰流量下很容易发生系统瘫痪。**第四，我们过去排障效率低，**故障应急处理时间长。这是我们最困难，最黑暗的一段时间。

所以，我们立马开始着手服务化治理。让所有业务服务是处于一种可治理状态，让整个业务链路是清晰的。这个技术本身不难，业内有诸多成熟的开源微服务治理框架可以直接使用。但对我们来说，难点是我们生产环境有很多老的服务在运行着，有的是JAVA实现的，有的是PHP实现的。我们没有办法做到让研发一次性把老代码全部铲掉，按照标准方式重新开发。这个是无法落地的。针对这个问题，我们的方案是采取泛服务化的设计方式来解决。

何为泛服务化？泛服务化就是在保留原有HTTP URL API调用原型的情况下，达到服务化治理的目的。研发同学无须对老服务进行大量的代码改造，只需要轻量的调整HTTP URL call的代码调用方式，动几行代码即可。这样就具备了基本的服务注册与服务发现与路由的机制。这种设计方式实现了大量老服务无须花太多代价就达到了服务治理的目标。在这样轻量改造下，使得我们可以快速地在生产环境进行落地，让所有的老服务具备了基本的服务化治理的能力。在生产环境的稳定性技术保障过程中我们就有了更多的治理与应急手段。



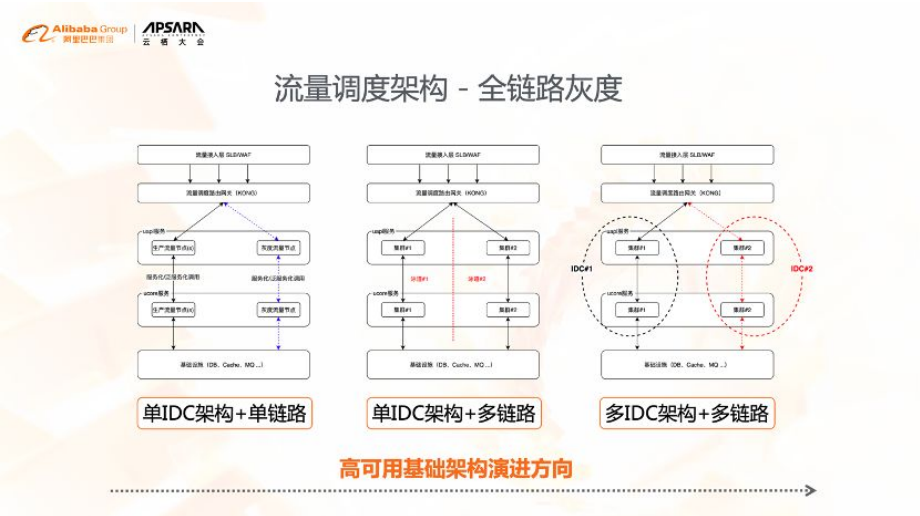
我们业务的主要技术栈是Java和PHP，在过渡期我们的系统就会存在多种类型的服务并存。这些类型的服务他们在同一个大的运行环境里，可以方便的相互通讯，随着老服务的慢慢改造成标准微服务架构，最终形成了全量的标准微服务链路。做到了不“一刀切”（全量高成本改造）也能在初期让服务架构具备治理能力。



因为我们第一代技术栈是以PHP技术为主。PHP高效快速的支持了业务的发展。但是在过渡期，我们如何让PHP服务跟Java服务在大的服务架构里互联通讯？我们中间件团队用了PHP sidecar的方式。帮助PHP做服务注册和发现和服务配置，让PHP和Java可以在同一个池子里运行。

做完服务治理之后，生产环境的链路就变得相对明确清晰。我们的监控工具可以把服务链路清晰的画出来。当出现技术故障或者冒烟时，我们也会有更多手段去做一些降级、限流、熔断等应急操作。我们可以更快的解决问题，让故障恢复的时间变的更短。

在我们服务化治理达到预期后，常规的生产故障会在很短的时间内被快速修复掉。从系统的流量角度看，我们初期是一个单IDC单链路的架构。链路容错性比较差。在很长一段时间，我们的生产效率比较低，大版本发布经常持续到凌晨。为了提升发布效率，提升技术稳定性。我们演进架构到单IDC多链路方式（如中间的图）。

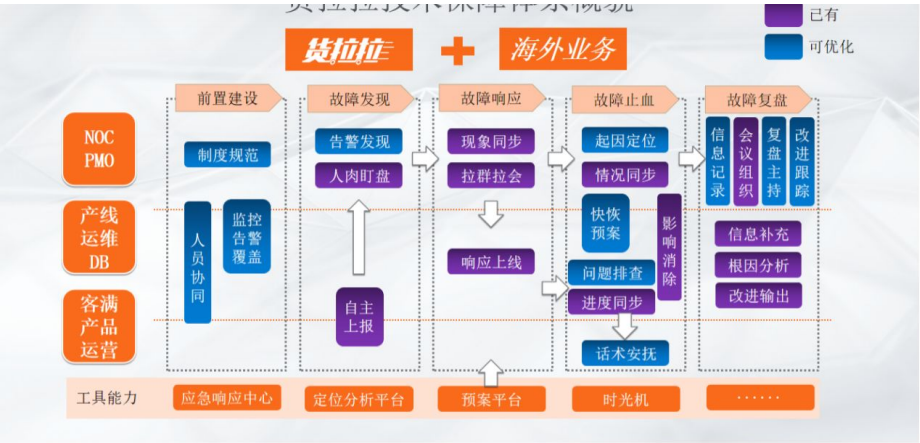


虽然所有服务依然在一个数据中心内，但可以创建出多条不同逻辑的流量链路。链路之间相互隔离，我们内部称呼为泳道。这种设计方式，解决了全链路高峰期生产灰度发布的场景问题。

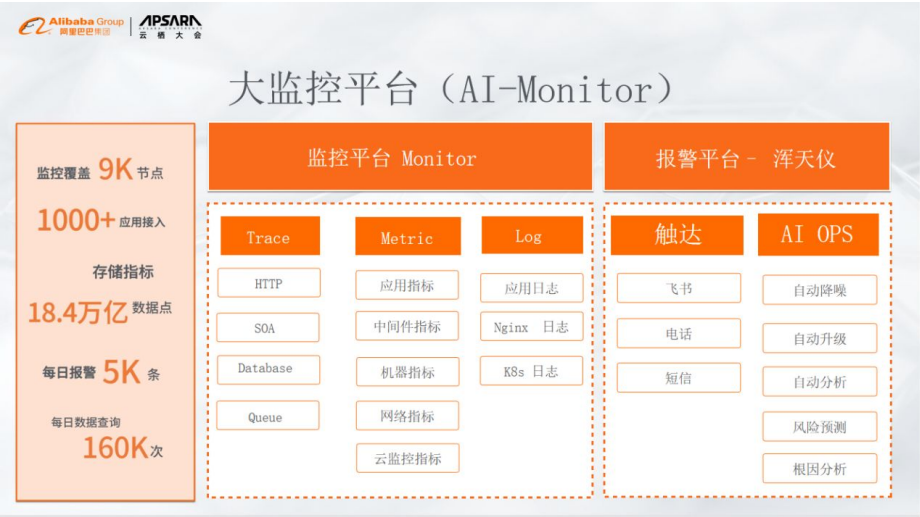
这样的流量调度架构不但解决了我们生产发布效率与低风险的问题，而且它也是作为我们系统容量发生严重故障时的一种应急手段。如果系统容量出现了致命的瓶颈故障并且无法快速恢复。系统容量不足无法扛住全网全业务的流量，我们可以紧急配置，修改流量调度策略，保住比较重要的城市和区域业务。

三、技术保障能力的建设

货拉拉在完成服务化架构（含泛服务化）的重构与治理之后，技术稳定性保障有了更多的手段与措施。在这个时期，技术团队也专门成立了全局稳定性团队。团队的核心职责是建设与完善货拉拉技术保障体系平台，目前的体系概貌如图所示。



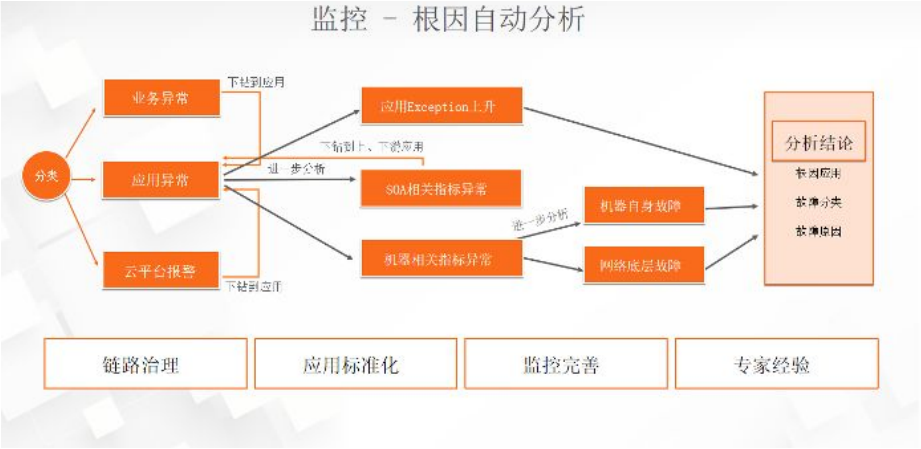
整套体系涵盖了前置建设、故障发现、故障响应、故障止血与故障复盘等领域。并且为每个领域优先打造了比较急需的能力。其中最重要的有两个方面。第一，NOC团队。这个团队的职责就是做稳定性大盘监控与职守，以及指挥生产故障的应急响应。他们需要做到快速止血、快速修复。我们要求这个团队在1分钟之内发现问题，5分钟之内响应这个问题。NOC是我们技术保障体系的守门人。第二，组织保障很重要。我们通过设置单独的组织架构与组织职责，重视故障的复盘与复盘整改的跟进闭环，定义稳定性过程指标和持续跟进这些指标。



大监控平台是我们技术保障体系最重要的领域。初期阶段，我们给予监控平台最高优先级去建设。内部团队一直称呼为AI-Monitor项目。项目启动之初，大家一致选择一条艰难的路，希望开发出的监控平台既能提供监控告警的常规能力（应用接入采集、数据存储计算、查询展示与告警），也能解决一些深层次的问题，譬如智能发现、智能健康扫描、自动分析与定位。所以大家在项目名称中带上了AI。目前为止，监控平台已经覆盖了所有应用(1000+)，覆盖所有的节点(9000+)，每天会收到5000+条告警（降噪前）。除了监控平台的功能性覆盖外，我们也会关注监控指标数据的实时性，查询响应延迟，把这些指标作为监控平台自身的关键技术指标对待。

这里重点提一下监控平台AI OPS模块里面的风险预测功能。平台无时无刻都在对每一个业务应用进行“健康体检”。捕获应用运行机器的OS指标(CPU、IO、Mem、Network等)，服务自身进程空间的指标信息（CPU、Thread、Mem、TCP等），应用服务的上下游服务，应用吞吐能力（QPS、Latency）、应用依赖的基础设施和中间件服务(DB、Cache、MQ等)，Java应用的JVM信息。把这些指标信息汇合在一起，检查是否有处于不合理范围区间的指标。一些指标也会进行“日环比/周同比”检查，背后专家团队也会持续的修订和定义这些指标的合理区间，如果发现有任何一个指标出现问题，平台就会发出预警，提醒我们研发同学做进一步的诊断。

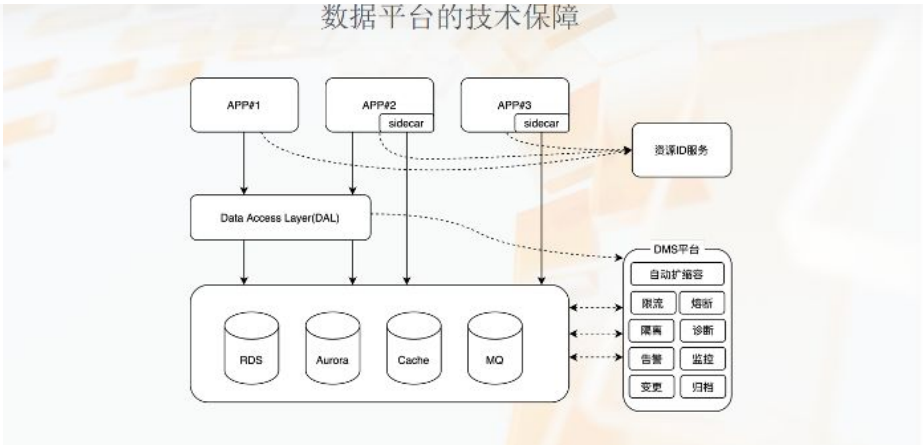
这种风险预测能力在货拉拉内部被广泛使用,我们统计有超过70%的技术冒烟事件都是可以通过这些常规的预测进行提前感知的，帮助我们把一些可能会引发严重技术故障的起因扼杀在最早期。



监控平台另一个比较重要的功能是根因自动分析。这个功能在内部的探索已经初具成效。我们依靠监控平台的自动分析，做到快速定位原因。平台的自动分析能力会聚合所有的异常指标，结合内部自动构建的服务链路关系，给出初步的分析建议，这大大加速了我们工程师人工排查的效率。我们可以快速定位故障的“根源”，第一时间用最合理的措施进行止损与恢复，最大程度降低了故障导致对业务的影响。

全链路容量压测，是货拉拉技术保障体系中的最核心的战场。在稳定性保障过程中，必须要知道系统容量可以支撑多大的业务量？核心主链路服务的容量短板在哪些地方？货拉拉在实现全链路容量压测平台时，也遇到过一些场景的设计与选择。

一，技术改造的选型。货拉拉选择生产真实库表。二，数据和流量模型的设计。由于主链路业务服务的流量比例不同，我们针对这些服务进行流量请求的额外补偿，这样让整个链路上的所有服务都得到1.5倍、2倍的流量压力。让整个系统的容量压测更全面和精准。三、容量治理与演练。我们每隔两周做一次全链路容量压测，跟进解决容量压测发现的技术设计、性能问题。避免系统技术架构长时间迭代出现腐化。另外，技术保障团队会定期结合压测进行生产故障模拟演练，让保障团队的研发同学始终保持对生产系统的稳定性敏感度。



技术保障体系中的数据平台建设，是货拉拉非常重视的一块领域。我们通过自建的DMS平台管理和治理数据类基础服务。平台的建设过程中会比较关注三个方面的能力。

第一，当业务体量快速发展，数据规模成倍增长，DMS在数据隔离、服务限流和实时扩缩容做了很多自动化建设。第二，DAL是我们快速引入并推动落地的数据库访问中间件。使数据库具备灵活弹性的分库分表、读写分离架构能力，来快速支撑业务规模的增长。第三，资源ID化设计。我们在交付数据服务时，是不会将服务的真实物理信息（链接串）暴露给研发的。而且我们的框架和工具是完全集成支持资源ID。研发同学做到开箱即用，不需要去了解资源ID背后各个环境的集群、部署和配置等信息。技术保障同学也可以通过资源ID快速定位是哪个业务应用在使用服务（譬如：DB、Cache、MQ）时出现了问题。

四、跨云的思考与实施

最后，我们聊聊货拉拉在多云场景下，我们是如何平衡效率，成本，稳定性设计。 主要有三个方面：磨平多云的差异化、云抖动的防范、IT成本的治理措施。

第一，磨平多云的差异化。因为货拉拉的业务在多云场景下，云商服务的API会存在一些差异性。所以我们内部做了一个LCloud工具平台来对接所有的云商。工具会磨平云商的差异。我们内部的研发、运维同学通过LCloud可以获得一致的操作体验。

第二，云是一定会抖动的。如果我们跑在云上的服务不具备足够的弹性，那么云上的“抖动”一定会影响你的业务。一旦出现网络的短暂抖动，就会导致整个服务响应延迟升高导致服务不可用。这个服务的吞吐能力会全部掉底，导致业务链路受损。

第三，云服务的IT成本。货拉拉目前的成本管控措施除了使用K8S基础设施做资源的弹性调度外，还大量使用了云商的一些特性。譬如：竞价实例、预留实例的购买。容器化团队也在研究如何在低峰时间弹出闲置资源，提供给离线任务场景的使用。

还没有评论

写下你的评论...

文章被以下专栏收录

货拉拉技术
科技改变物流

推荐阅读

阿里ACP大数据认证笔记--DataIDE篇

复习资料： 阿里云官方文档云顶云每日一题和复习视频（未看完）淘宝题库DATAIDE 概述提供强大的调度能力，支持按照时间、依赖关系的任务触发机制，支持每日千万级别的任务按照DAG关系准确、...

噜啦啦啦啦

Introduction to OCC: Part 1

南昌市湿地公园

一文教会你如何写复杂业务代码

阿里开发者

一文教

简介： 这
代码。面
场景，如
应对，是
题，我进
究。结合

阿里云云