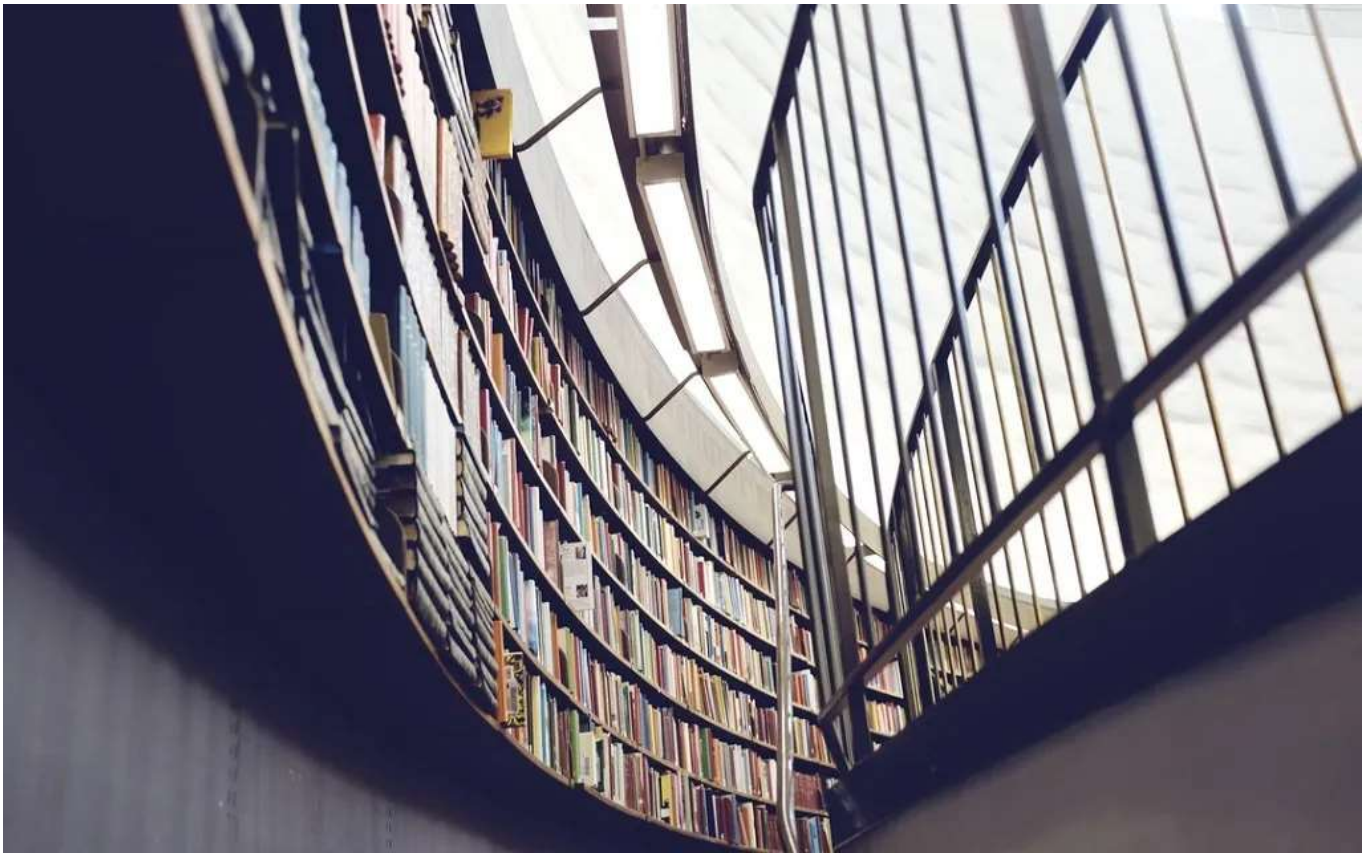
云上资源编排的思与悟

原创 元吟 阿里技术 3天前



一 背景

2018年7月9日，我通过校招加入阿里云，开启了职业生涯。有幸参与了资源编排服务从1.0到2.0的全部设计、开发、测试工作，这对我了解云上服务起到了启蒙作用。当然，本文源于我在设计开发过程中的思考和感悟。

在传统软件架构下，撇开业务层代码，都需要部署计算节点、存储资源、网络资源，然后安装、配置操作系统等。而云服务本质上是实现 IT 架构软件化和 IT 平台智能化，通过软件的形式定义这些硬件资源，充分抽象并封装其操作接口，任何资源均可直接调用相关 API 完成创建、删除、修改、查询等操作。

有赖于阿里云对资源的充分抽象以及高度统一的OpenAPI，这让基于阿里云构建一套完整的 IT 架构并对各资源进行生命周期管理成为可能。客户按需求提供资源模板，编排服务将会根据编排逻辑自动完成所有资源的创建和配置。

二 架构设计

伴随着业务场景的增加和业务规模的指数级增长，原有架构逐渐暴露出租户隔离粒度大、并发量小、服务依赖严重等问题，对于服务架构的重构迫在眉睫，其中最重要三个方面就是拓扑设计、并发模型设计和工作流设计。

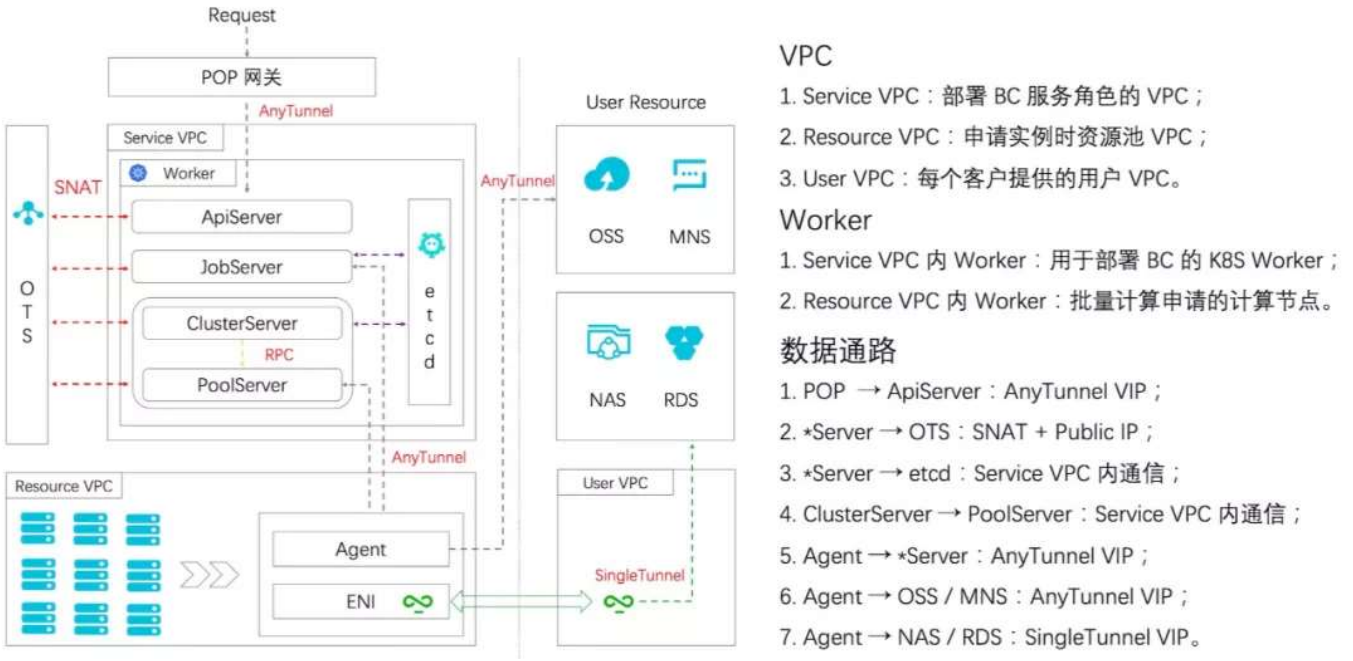
1 拓扑设计

拓扑设计的核心问题是明确产品形态和用户需求、解决数据通路问题。站在产品角度考虑的点包括: 1. 资源所有者(服务资源[计费单元]、用户资源)、2. 资源访问权限(隔离、授权)。站在用户角度需要考虑的点包括: 1. 服务类型(WebService型-需公网访问、数据计算型-阿里云内网访问)、2. 数据打通(源数据、目的数据)。

资源所有者分为服务账号和用户账号。资源属于服务账号的模式又叫做大账号模式，该模式优点有: 1. 管控能力更强；2.计费更容易。但易成为瓶颈的点包括：1.资源配额；2. 依赖服务的接口流控。很显然，全量资源托管是不现实的，比如VPC、VSwitch、SLB、SecurityGroup等资源客户往往需要和其他系统打通，这部分资源通常是用户提供的，而ECS实例则比较适合通过大账号创建。

多租户隔离在大账号模式下是非常重要的问题。既要保证某一用户的资源彼此可以相互访问，又要保证多个客户之间不能有越界行为。一个常见的例子是，所有用户的ECS均开在同一个服务VPC内，同一个VPC内实例默认是可以相互访问的，存在安全风险，因此在系统设计初期就需要考虑到相关问题的应对方案。

对于上述问题我们的设计是，ECS实例通过大账号模式创建在服务账号下的资源VPC内， 通过企业级安全组实现不同用户实例的访问隔离。涉及用户数据(NAS、RDS等)访问的操作时，需要用户提供这些访问点所在的VPC和Vswitch，通过在实例上创建ENI并绑定到用户VPC上，实现对用户数据的访问。具体数据通路如图所示。



常见的服务架构

2 并发模型设计

模型设计的核心是解决高并发(High Concurrency)、高性能(High Performance)、高可用(High Availability)问题。

资源编排的高并发主要指标为QPS(Queries-per-second)，对于动辄以分钟为单位的资源编排逻辑而言，同步模型显然不能支撑较高并发请求。资源编排的高性能主要指标为TPS(Transactions-per-second)，在根据用户资源模板编排资源的过程中，资源彼此间存在一定的依赖关系，线性地创建资源会导致大量时间处于忙等状态，服务吞吐严重受限。资源编排的高可用主要指标为SLA(Service Level Agreement)，在HA基础上若能解耦CRUD对内部服务的依赖，在服务升级或发生异常时就可以减小对SLA的影响。

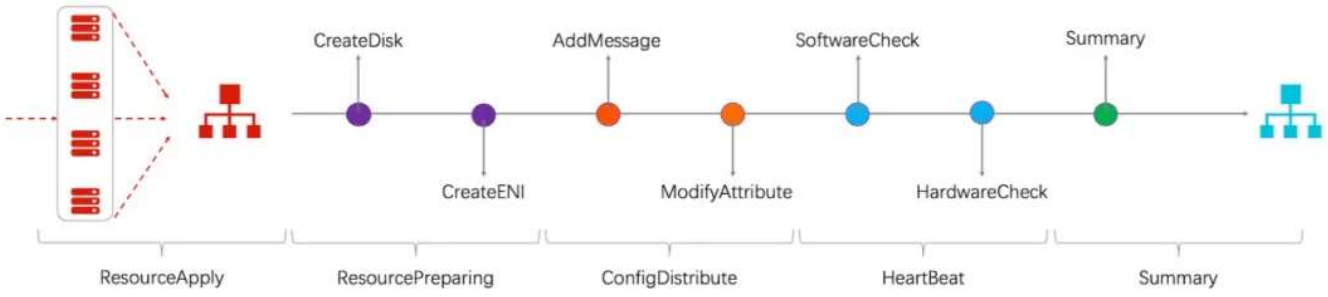
对于上述问题我们的设计是，在服务前端仅进行简单的参数检查后立即将用户模板写入持久化层，写入成功后立即返回资源ID，已持久化的资源模板将被视为未处理完成的任务等待调度处理。随后，我们周期性扫表探测任务，有序创建资源并同步其状态，如遇资源状态不满足向下推进的条件则立即返回，经过多轮次处理，最终达到期望的状态， 一个简化的分布式模型如图所示。

分布式并发模型

为了避免任务较多情况下的锁争抢问题，我们设计一套任务发现 + 租约续租的机制，一旦集群从数据库池子中被某个节点争抢到之后会被添加到该节点的调度池中并设定租约， 租约管理系统会对即将到期的租约进行续租(加锁)。这样可以确保一个集群在下一次服务被拉起前一直只被某个节点处理，如果服务重启，则任务会因超时自动解锁并被其他节点捕获。

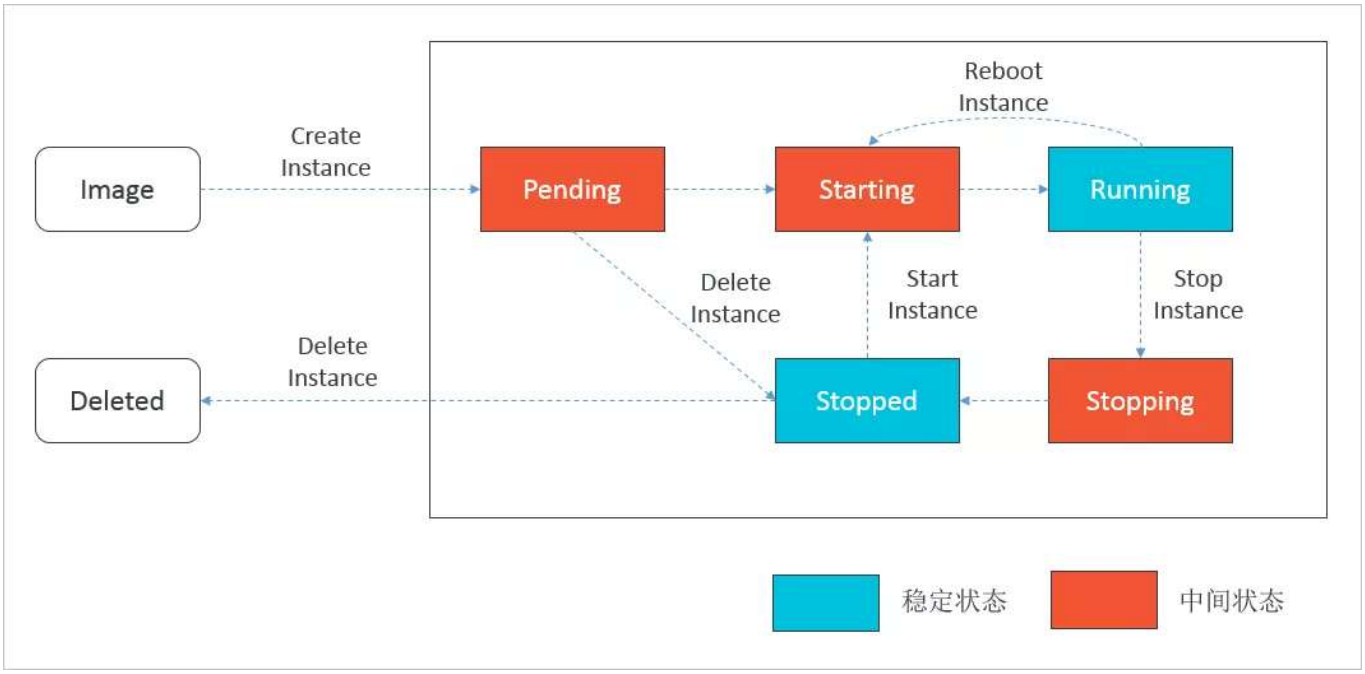
3 工作流设计

流程设计的核心是解决依赖问题。依赖问题包含两种情况：前序资源的状态不符合预期和资源本身状态不符合预期。我们假设各资源的状态只有可用和不可用，并且假定可用的资源不会跳转到不可用状态，最简单的情况就是一个线性任务，如图所示。考虑到部分子资源的编排工作可以并行，编排过程就可以看作是一个有向无环图(DAG, Direct Acyclic Graph)任务。



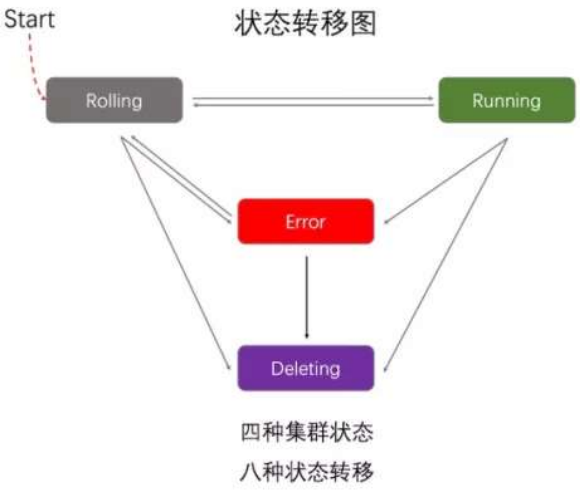
资源线性编排结构

世界不只是非黑即白，资源的状态也是一样，有向无环成为了美好的愿望，有向有环才符合真实世界的运行规律。对于这种情况，简单的工作流很难覆盖复杂的流程，只有进一步对工作流抽象，设计符合要求的有限状态机(FSM, Finite State Machine)。有限状态机说起来过于抽象，但ECS实例的状态转移大家都接触过，下图就是ECS实例的状态转移模型。



ECS实例状态转移模型

结合实际业务需求，我设计了如下图所示的集群状态转移模型。该模型简化了状态转移逻辑，有且仅有Running这一稳态，其他三种状态(Rolling、Deleting、Error)均为中间态。处于中间态的资源会根据当前资源状态尝试向着稳态越迁，每次状态越迁过程均按照一定的Workflow执行相关操作。



状态转移表

状态转移	描述
Rolling → Running	集群所需资源全部准备完毕，达到用户预期
Rolling → Error	资源未分配完毕前集群动态检查失败
Rolling → Deleting	用户在集群尚未准备完毕前发起删除集群请求
Running → Rolling	集群遇升级或某些异常导致机器被释放
Running → Error	资源分配完毕后集群动态检查失败
Running → Deleting	用户在集群尚未准备完毕前发起删除集群请求
Error → Rolling	动态检查异常恢复逻辑
Error → Deleting	用户在集群动态检查失败下发起删除集群请求

集群状态转移模型

从这时起，服务的整体架构和设计思路基本确立。

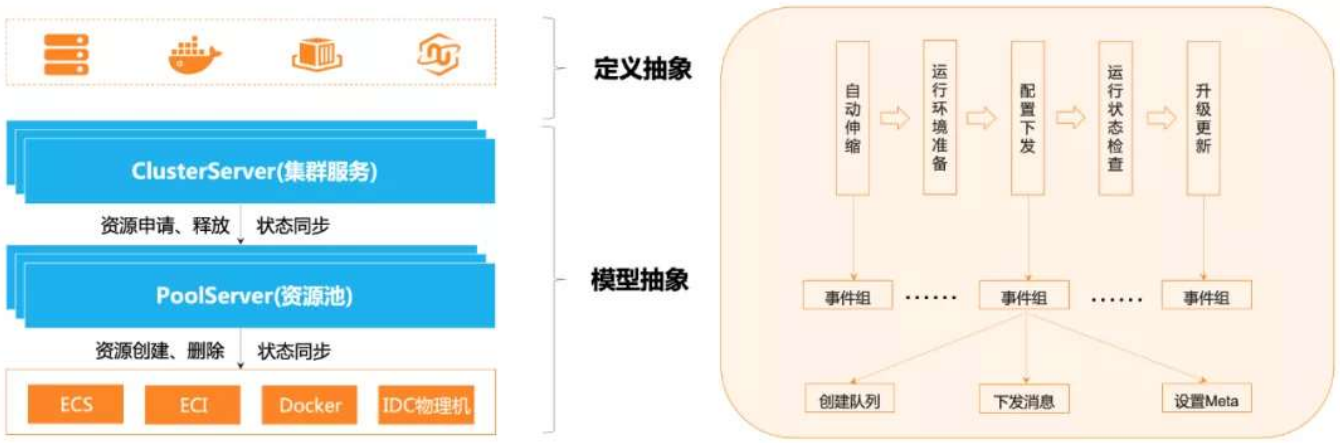
三 核心竞争力

资源(ECS)短缺问题日益严峻，加上粗粒度的扩缩容、升降配功能已不能满足客户的需求，资源池化(Resource Pooling)、自动伸缩(Auto Scaling)、滚动升级(Rolling Update)被提上日程并成为提升产品竞争力的一大利器。

1 资源池化

资源池化简单来说就是提前预留某些资源以备不时之需，很显然，资源池化的前提一定是大账号模式。对开发者而言，线程池不是陌生的词汇，但资源池却相对比较遥远，实际上，资源池解决的就是资源创建、删除时间开销很大以及库存不可控的问题。当然， 池化资源另一个假设是，被池化的资源会被频繁使用且可被回收利用(规格、配置相对单一)。

由于计算资源创建周期较长且经常被资源库存等问题困扰，加之产品期望在业务上有所拓展，因此我们设计了如图所示的资源池化模型并对多种计算资源进行抽象，提供了一套可以应对异构资源的处理逻辑。



资源池化模型

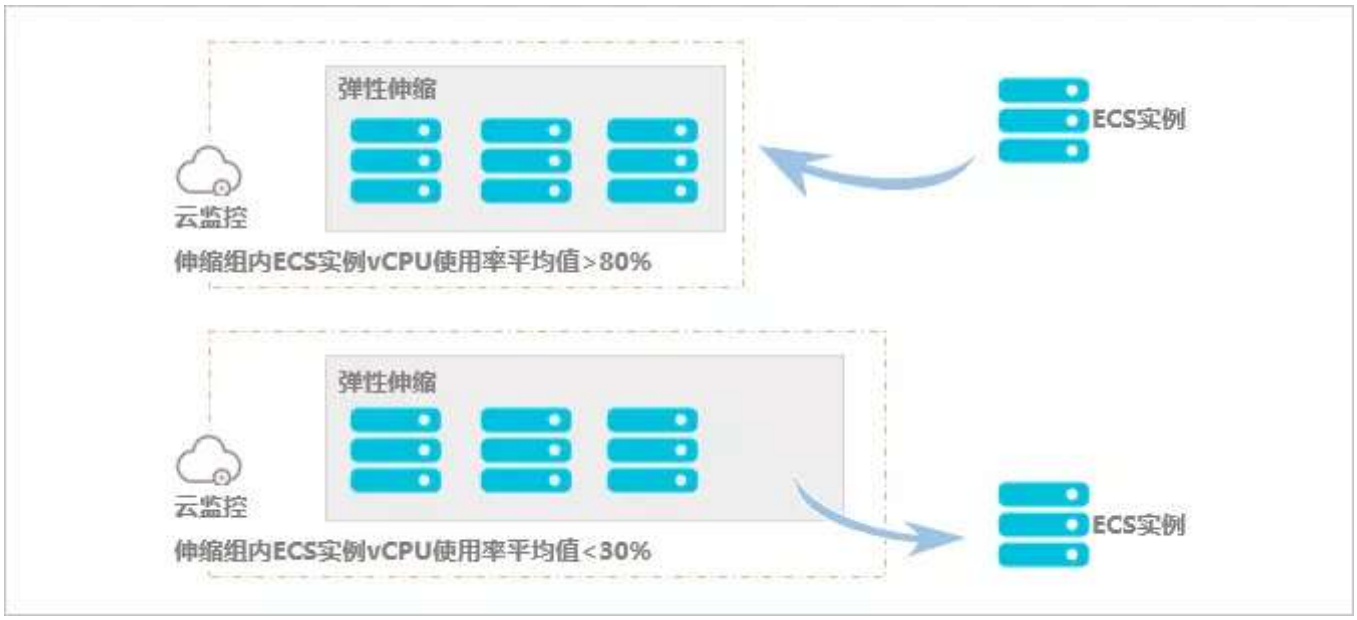
资源池化可以大大缩短资源创建等待时间，解决库存不足问题，另外，它可以帮上层使用到资源的服务解耦复杂的状态转移逻辑，对外提供的资源状态可以精简到Available和Unknown两种，所得即可用。但不得不考虑的问题包括：

- ECS实例的创建是否受用户资源的限制(如用户提供VSwitch会限制ECS可用区)。
- 如何解决资源闲置问题(成本问题)。

对于第一个问题，目前受制于VSwitch由客户提供，暂时还没有比较好的解法，只能尽量要求客户提供的VSwitch覆盖更多的可用区，如果VSwitch属于服务账号，就可以比较好规划资源池建在哪个AZ。对于第二个问题，资源池本身也是一种资源，成本控制我们可以从接下来提到的自动伸缩上得到答案。

2 自动伸缩

云计算最大的吸引力就是降低成本，对资源而言，最大的好处就是可以按量付费。实际上，几乎所有线上服务都有其峰谷，而自动伸缩解决的正是成本控制问题。它在客户业务增长时增加ECS实例以保证算力，业务下降时减少ECS实例以节约成本，如图所示。

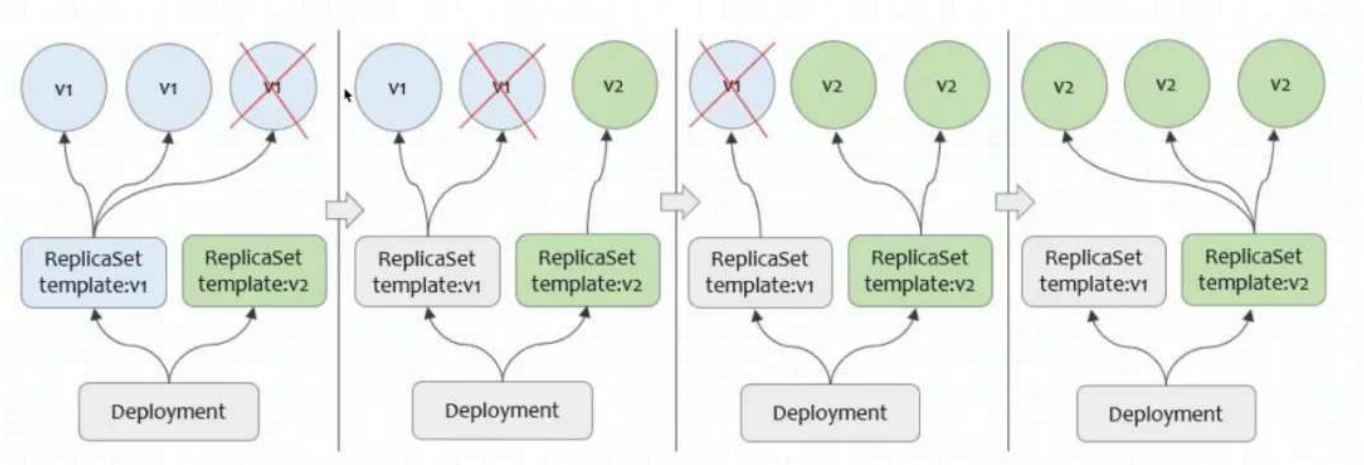


自动伸缩示意图

我对自动伸缩的设计思路是，先对时间分片触发定时任务，再对时间段内配置伸缩策略。伸缩策略也包含两部分，一部分是最大ECS规模和最小ECS规模，它指定了该时间段内集群规模的浮动范围，另一部分是监控指标、耐受度和步进规则，它提供了伸缩依据和标准。这里监控指标是比较有意思的点，除了采集云监控的CPU、Memory利用率外，还可以通过对ECS空闲、忙碌状态的标记，计算出工作节点占比率，一旦超出耐受范围，即可按步进大小触发一次扩容或缩容事件。

3 滚动升级

客户服务架构的修改往往涉及复杂的重建逻辑，在重建过程中不可避免的会影响服务质量，如何优雅平滑地做升降配成为了诸多客户的刚需。滚动升级正是解决不停服、可调控的升降配问题的。



滚动升级示意图

一次简化的滚动升级过程如上图所示。滚动升级的核心是对升级进行灰度，按照一定比例开出Standby资源直到它们可以顺利服役，随后再下线掉相应台数的资源。经过多次滚动之后，使其全部资源更新到最新预期，通过冗余实现升级不停服。

四 可观测性

服务可观测性将来必将成为云服务的核心竞争力之一，它包括面向用户的可观测行和面向开发者的可观测性两部分。时至今日，仍然记得半夜被客户电话支配的恐惧，仍记得对着海量日志调查问题的不知所措，仍记得客户一通抱怨后毫无头绪的茫然。

1 面向用户

是的，我希望用户在向我们反馈遇到的问题时，提供的信息是有效的，甚至是能直接指向病灶的。对用户而言，能够直接通过API获取资源编排所处的阶段以及各阶段对应资源的状态信息，确实能够极大地提高用户体验。针对这个问题，我分析了系统处理流程， 设计了面向"阶段 - 事件 - 状态"的运行状态收集器。

具体包括：对的业务流程进行拆分得到多个处理阶段，对每个阶段依赖的事件(资源及其状态)进行整理，对每个事件可能出现的状态做结构化定义(尤其是异常状态)。一个典型的样例如代码样例所示。

```
[
    {
        "Condition":"Launched",
        "Status":"True",
        "LastTransitionTime":"2021-06-17T18:08:30.559586077+08:00",
```

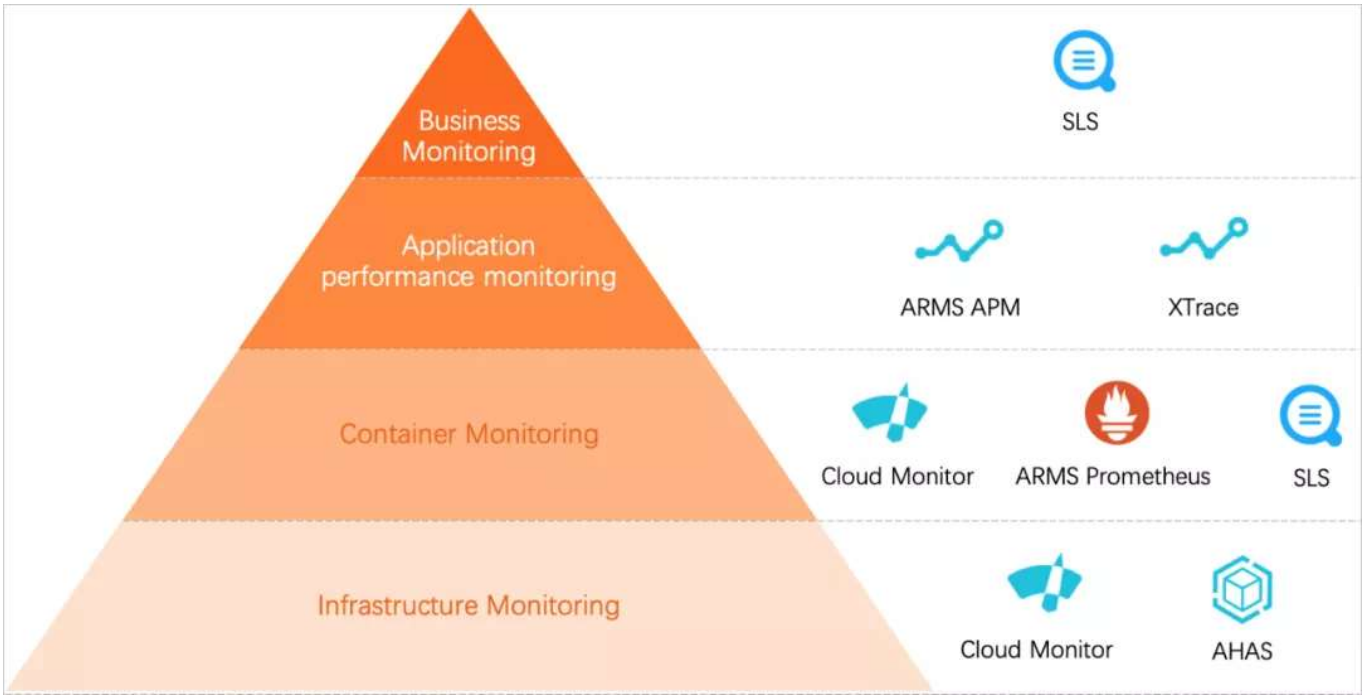
```
            "LastProbeTime":"2021-06-18T14:35:30.574196182+08:00"
        },
        {
            "Condition":"Authenticated",
            "Status":"True",
            "LastTransitionTime":"2021-06-17T18:08:30.941994575+08:00",
            "LastProbeTime":"2021-06-18T14:35:30.592222594+08:00"
        },
        {
            "Condition":"Timed",
            "Status":"True",
            "LastTransitionTime":"2021-06-17T18:08:30.944626198+08:00",
            "LastProbeTime":"2021-06-18T14:35:30.599628262+08:00"
        },
        {
            "Condition":"Tracked",
            "Status":"True",
            "LastTransitionTime":"2021-06-17T18:08:30.947530873+08:00",
            "LastProbeTime":"2021-06-18T14:35:30.608807786+08:00"
        },
        {
            "Condition":"Allocated",
            "Status":"True",
            "LastTransitionTime":"2021-06-17T18:08:30.952310811+08:00",
            "LastProbeTime":"2021-06-18T14:35:30.618390582+08:00"
        },
        {
            "Condition":"Managed",
            "Status":"True",
            "LastTransitionTime":"2021-06-18T10:09:00.611588546+08:00",
            "LastProbeTime":"2021-06-18T14:35:30.627946404+08:00"
        },
        {
            "Condition":"Scaled",
            "Status":"False",
            "LastTransitionTime":"2021-06-18T10:09:00.7172905+08:00",
            "LastProbeTime":"2021-06-18T14:35:30.74967891+08:00",
            "Errors":[
                {
                    "Action":"ScaleCluster",
```

```
                "Code":"SystemError",
                "Message":"cls-13LJYthRjnrdOYMBug0I54kpXum : destroy worker fa
                "Repeat":534
            }
        ]
    }
]
```

代码样例：集群维度状态收集

2 面向开发者

对开发者而言，可观测性包含监控和日志两部分，监控可以帮助开发者查看系统的运行状态，而日志可以协助问题的排查和诊断。产品从基础设施、容器服务、服务本身、客户业务四个维度进行了监控和数据聚合，具体用到的组件如图所示。



各级别监控、告警体系

基础设施主要依托云监控(Cloud Monitor)追踪CPU、Memory等使用率；容器服务主要依赖普罗米修斯(Prometheus)监控部署服务的K8S集群情况。对服务本身，我们在各个运行阶段都接入了Trace用于故障定位；对最难处理的客户业务部分，我们按通过SLS收集客户使用情况，通过UserId和ProjectId进行数据聚合，并整理出普罗米修斯的DashBoard，可以快速分析某个用户的使用情况。

除监控外，已接入云监控告警、普罗米修斯告警和SLS告警，系统、业务分别设置不同告警优先级，并整理了丰富的应急响应方案。

五 其他

从懵懂到能够独立负责资源编排服务的设计、开发工作，阿里云提供了宝贵的学习平台。如今秋招大幕已经拉开，欢迎应届同学加入我们，简历请投递邮箱: yuanyin.lw@alibaba-inc.com. 此时此刻，非你莫属。

技术公开课

虚拟化技术入门

本课程共7个课时，主要讲解云计算技术的核心技术之一——虚拟化技术，介绍说明虚拟化技术的主要作用以及常见实现方法，并针对硬件中常用的虚拟化技术（CPU、内存、IO）进行详细的讲解，最后针对目前流行的开源虚拟化项目，讲解其出现的漏洞以及阿里云是如何进行漏洞分析和处理的。