EC2 Container Service 上的微服务架构

李磊, AWS 解决方案架构师

May 2017

© 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.

注意

本文档仅供参考所用。文中所述仅代表文档发布之际 AWS 的产品信息和作者实践经验，如有更改，恕不另行通知。用户应对本文中所有信息准备确和 AWS 产品及服务的使用有自己的评估，并对此负责，本文不提供任何显式或隐式的保证。本文档不代表 AWS, 其附属公司,供应商或许可商提供任何保证、表示、合同承诺。AWS 与其用户间的所有责任和义务关系只由 AWS 使用协议控制，本文不参与，也不构成对 AWS 与其用户间任何协议的修改。

注意 2
简介 4
什么是微服务架构 4
微服务架构的特性 4
单体型架构 vs. SOA vs. 微服务架构 5
微服务的益处 6
微服务架构面临的挑战 8
微服务与云平台 9
ECS 上构建微服务架构 10
ECS 简介 11
ECS 的架构及工作原理 12
分层构建微服务方案 16
解决微服务架构中的常见问题 19
总结 26
客户案例 26
引用 31

简介

微服务是一种软件开发的组织和架构方法，它可以加快软件交付周期、增强创新和自主性，提高软件的可维护性和可伸缩、可扩展性，同时也提高了企业开发和发布软件服务的能力。使用微服务架构，软件产品将由多个独立的、可通过 API 进行交互的服务组成。这些服务将由各个小团队独自负责。

通过本白皮书，我们将首先总结一下微服务架构的特性，讨论构建微服务架构面临的挑战和难题，然后介绍 AWS 的 ECS 容器集群服务，以及如何利用 ECS 结合其他 AWS 提供的服务解决微服务架构中遇到的难题。

什么是微服务架构

微服务架构的特性

近些年，微服务架构的概念变得越来越火[1]，事实上微服务并非某项具体的技术或框架，它也不是软件工程学中新的概念，而是一些成功的，经过实践论证的概念的组合，比如面向对象方法论、敏捷开发、面向服务架构、API-first 设计、以及持续集成。

“微服务”这个名词是一个概括性的术语，很难对其进行精确的定义。但是所有微服务架构方法都具有以下共同特性：

- **去中心化**：微服务架构是由去中心化数据管理的分布式系统组成。它不依赖中心数据库上统一的数据库模式。每一个服务都有自己独立的数据模型。去中心化的另一个含义是，每个服务在开发、部署、管理和维护过程中也是相互独立的，不依赖某个中心系统。
- **独立**：微服务架构中各个服务组件都可以独立进行更改、升级和替换，而不会影响其他服务组件的正常功能。同样，负责各个服务组件的团队之间也都是相互独立的，互不干扰的。
- **专注做好一件事**：每个服务组件都是专注于某个问题域的功能集合，一旦某个服务复杂度到达一定程度，则应该考虑将其进行服务切分或对其进行再次微服务架构。
- **多语种**：微服务并不是“一体通用”的方法，它允许各个团队根据自己的喜好选择最适合自己问题域的开发语言、开发工具。因此，微服务无论在操作

系统、开发语言、数据存储还是工具，都是兼容的。也就是说，微服务是“多语种”的。

- **黑盒子：**微服务中的服务都是按照“黑盒”设计的，也就是说，它们将内部逻辑对外隐藏进来，各个服务间通过定义良好的 API 进行通信，避免了暗含的相互依赖。
- **谁构建，谁运行：**通常情况下，在生产环境中，构建了某个服务的团队负责该服务的运行和维护。这条规则也称之为 DevOps[2]。DevOps 除了能让各个团队按照自己的安排独立工作外，还可以让开发人员更贴近软件产品的真正使用者，能帮助他们更好地了解用户的需求。微服务对于企业组织架构上的影响也是不容忽视的，正如康威定律所言，系统架构是公司组织架构的反映[3]。

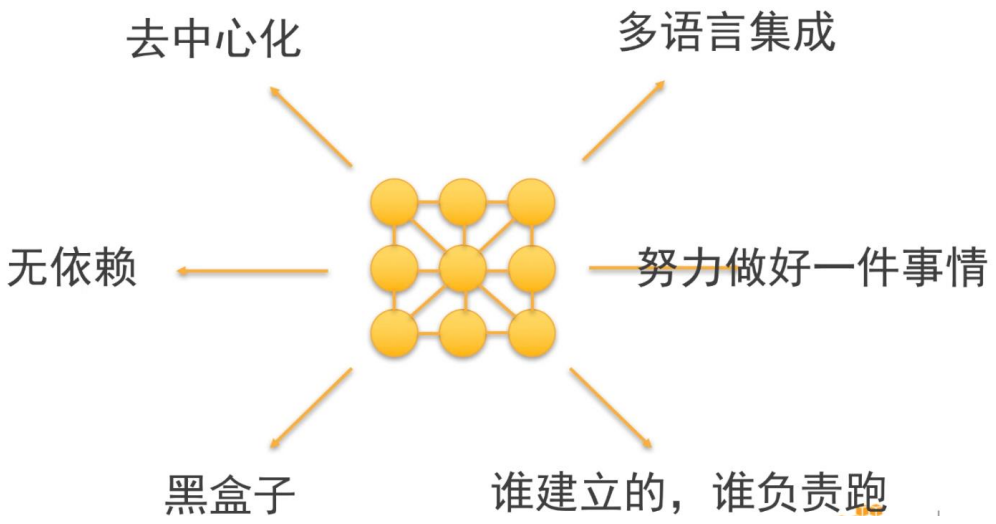

图 1：微服务架构的特征

单体型架构 vs. SOA vs. 微服务架构

技术为业务而生，架构也为业务而出现，无论传统单体型架构、SOA 和微服务都是因为业务的发展而出现。出现 SOA 以及微服务框架与业务的发展、平台的壮大密不可分。

单体型架构

当网站或者应用流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本，简化开发流程，适合于个人开发或者小团队开发一些功能单一的小产品使用。这是最传统的架构模式，一般整个系统都是有一个统一的数据中心，各

个功能模块显式进行相互依赖（比如直接调用模块公开的方法）。当网站流量随着业务发展不断变大时，这种架构的应用在可扩展性、容错性（一旦某个功能模块被流量击溃，整个系统瘫痪）上存在致命性的缺陷。

SOA

当网站或者应用流量变大，单体型架构无法招架之时，可以采取面向服务架构方式，将应用根据系统维度、功能维度、读写维度或者模块维度进行划分成各个独立的服务组件，服务组件间通过某个约定的方式（比如 Web Service 或者 HTTP REST API）进行通信。当服务越来越多，容量的难以准确评估，小服务导致资源浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率，这种架构一般仍然存在一个统一的数据中心或者调度中心。整个系统的可维护性、可扩展性相对于单体型架构得到提高，但在服务细粒度、隔度性、去中心化方面仍与微服务有差，另外就是 SOA 架构常常依赖于总线结构，利用可热插拔的中间件以及消息队列来完成服务与服务之间的解耦，对于总线结构的依赖也使得服务的独立性并没有被完全剥离。

微服务架构

在微服务架构中，每个服务都要去中心化，不存在单点故障问题。每个服务都要实现高可用、高负载，不会因为一个服务不可用而影响整套业务流。每个服务细粒度，专注于自己的问题域，与别的服务采用更好的 Restful 或者良好定义的 API 进行通信。服务都要高度通用化，即多种终端都可调用，不分语言和平台服务部署或升级简单，不会消耗大量人力并且部署过程不易出现人为错误，同时还具有快速注册与自动服务发现功能。

微服务的益处

许多 AWS 客户选择微服务架构来解决传统单体型架构中存在的敏捷性和可扩展性限制的问题。让我们来看看选择微服务架构能够带来哪些方面的益处吧。

敏捷

使用微服务的组织由多个负责运维独立服务的小团队组成。各个团队都是在界限明确的小范围内独立、快速地工作，从而起到减小迭代时间的效果。各个小团队效率的提升将使整个企业组织工作效率进行很大程序地提高。

图 2 标示由多个小团队组成的企业组织架构与由一个大团队组成的企业组织架构，在发布部署一个应用时的工作效率区别。

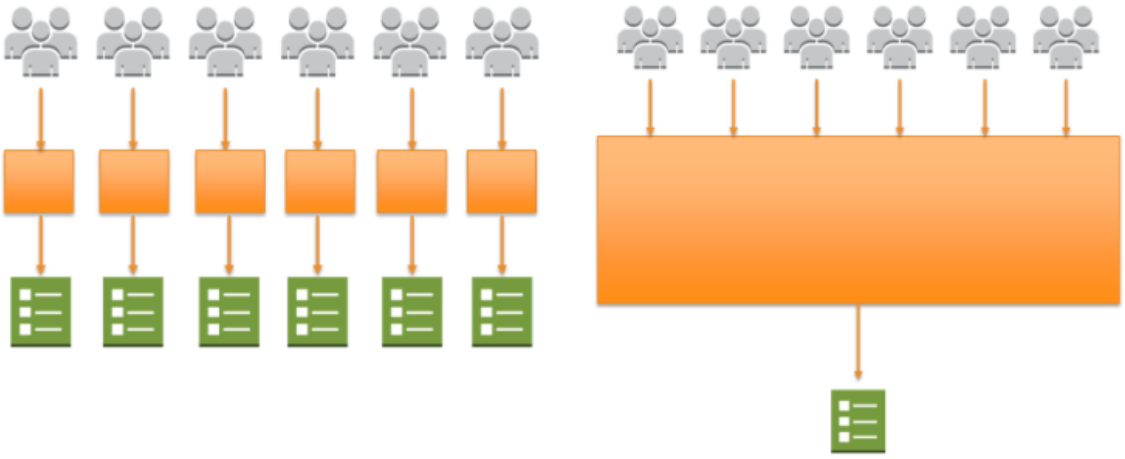

图 2：部署微服务

创新

小团队可以独立自主地针对自己负责的问题域选择相关的技术、框架和工具，这就是创新的体现。职责与义务，让团队产生对自己的服务的责任感。

DevOps 通过将开发与运维人员结合到一个团队中，让他们能更早地、更好地沟通，解决了开发部署过程中没必要的摩擦和矛盾。当应用程序部署时，敏捷的处理流程可以让开发和运维工作不再需要暂停下来，相反，整个软件交付周期，从提交代码到发布版本，整个流程将变成自动化的。测试一个新的想法，或者当新功能出现问题时及时回滚，将变得更加轻松。因为失败成本的降低，团队将更加乐于更改和创新。

质量

使用微服务架构您的应用程序，将一定程度上提高代码质量。这跟面向对象设计中提出的分模块开发有同样的效果：提高代码的可重用性、封装性和可维护性。

伸缩性

细粒度解耦的微服务架构另一个好处就是，它非常适合构建大规模系统。它允许针对不同的服务选择最适合的技术，这也是性能优化的体现。每个服务都可以使用最合适的开发语言和框架来实现，利用最优的数据持久化解决方案，并通过各自独立的服务配置进行调优。

合理解耦的微服务架构，可以独立地进行水平伸缩扩展。这相对于传统架构中使用的垂直伸缩有很大的优势。垂直伸缩指的是当业务量、流量增大时，升级硬件资源，将应用部署到“更大的机器”上，这会导致“机器硬件资源上限”和伸缩时宕机的问题。水平伸缩，通过将更多的服务部署到服务池（多个机器组成的集群）中，可以动态地适配流量，并且不会陷入单个机器资源上限问题。整个水平伸缩的过程将会是自动化的。另外，应用程序的可靠性将得到增加，因为出现问题的服务可以自动地被健康的服务替换掉，而不影响整个应用的运行。

可用性

微服务架构可以轻松实现故障隔离。通过健康检查、缓存、隔离仓、断路器等技术，可以减小当某个组件出现故障时的影响，从而保证应用的高可用性。

微服务架构面临的挑战

尽管拥有前面提到的那么多优点，但是微服务架构和所有其他架构一样，都不是解决所有问题的“银弹”。使用微服务架构也会面临一些挑战和难题。比如：

- 微服务架构都是分布式的，也就会存在所有“分布式计算误区”中提到的问题[4]。
- 从单体型架构迁移至微服务架构，需要您准确地划分好各个服务间的界限，并且从应用层到数据库层松耦合代码依赖。
- 除此之外，在组织管理方面还存在着一些挑战，比如如何组建一个高效的团队，如何将团队转变成 DevOps 的生产模式，如何合理化开发团队与运维团队的沟通等。

本白皮书主要关注的是架构和运维方面遇到的挑战，如果想了解更多关于 AWS 如何运用 DevOps 的知识，请访问 https://aws.amazon.com/devops[5]

架构复杂度

在传统单体型架构中，所有的复杂度和依赖都存在项目的代码库中。而对于微服务架构而言，复杂度变成了各个服务间的交互。

在架构上面临的挑战有处理异步通信、连锁故障、数据一致性问题、服务发现和授权认证等问题。

运维复杂度

运用微服务架构，你不再只是运行一个服务，而是数十个，甚至数百个服务。这也就带来了以下问题：

- 如何以可伸缩、低成本的方式配置好所需要的资源？
- 如何避免不同的团队重复“造轮子”，重复处理同一个问题，或重复安装配置相同的软件工具？
- 如何跟踪并管理好上百个代码流的部署过程和它们之间的相互依赖（不同服务可以单独部署）？
- 如何监控整个系统的健康并提前感知潜在的热点和流量高峰？
- 如何在整个系统各个服务间追踪和调试问题？
- 如何分析一个快速增长并超出预期需求的分布式系统的海量日志数据？
- 如何处理不同技术栈、开发环境间缺少统一标准的问题？
- 如何受益于不同技术带来的开发效率，而不受困于维护升级这些技术带来的困难？

微服务与云平台

微服务近些年的崛起与诸如 AWS 这样的公有云平台的发展密切相关。AWS 提供了可以直接解决微服务架构面临的挑战的服务。

- **按需资源调配：**资源可以快速根据资源进行配备。相对于传统的基础设施，云平台几乎可以说在资源上是没有上限的。运行于不同环境、不同版本的服务可以临时或永久地共存。不同需要进行困难的资源预测和评估。按需资源调配解决了如何以可伸缩、低成本的方式配置好所需要的资源的问题。
- **低成本低风险地进行尝试：**你只需为你使用的资源付费，从一定程序上降低了你尝试新想法的成本。新功能或者新服务可以快速地发布，并且当尝试失败时，可以快速关闭这些服务和其所需的资源。降低尝试新想法所需的成本和风险，将大力推动更多的创新。这也是跟微服务架构提倡的高度敏捷相符合。
- **可编程性：**AWS 服务自带 API、命令行工具（CLI）和针对不同语言的 SDK。服务器甚至整个架构都可以通过编码的方式进行克隆、关闭、扩展、监控并且在发生故障时实现自动恢复。标准化和自动化是构建速度、一

致性、重复性和可伸缩的关键。你可以通过编码的方式统一管理你所需的资源，减少运维微服务架构所需的精力。

- **基础设施即代码：**除了可以编写脚本来配置和管理你的资源，AWS 还支持你将基础设施代码化，像应用程序的代码一样放置到版本管理工具中进行管理。因此，各个版本的基础设施都可以随时进行重新部署。你可以保证基础设施代码与应用程序代码一样的质量和性能。回滚不再只与代码相关，也包含整个基础设施。可编程性和基础设施即代码可以解决运行、维护、扩展微服务架构时面临的问题。
- **持续交付：**云平台的可编程性推动了配置和部署流程的自动化。开发阶段的持续集成概念也因此可以扩展到运维阶段（因为运维也是可编码的了），从而形成持续集成与交付的概念。持续交付解决了如何同时管理多个代码流并行部署交付的问题。
- **自我管理的服务：**云平台最出色的优点就是服务都是自我管理的。你不再需要进行繁重的配置虚拟机、安装配置优化软件、处理资源伸缩扩展、管理备份等运维工作。诸如监控、安全、记录日志、伸缩和保持可用性等系统操作和特性都在云平台服务中内置好了。自我管理的服务从一定程度上缓解了运行微服务架构的复杂度和工作量。
- **面向服务：**AWS 本身就是微服务架构的。每一个 AWS 服务都是针对某个特定问题域的，并且都是通过 API 进行相互通信。你可以将这些服务像乐高积木一样组合成复杂的结构，用来解决特定的问题。这样就解决了重复“造轮子”的问题。
- **多语种：**AWS 提供了多种存储得数据库技术供你选择。AWS Marketplace 上有许多可以运行在 EC2 上的操作系统供你使用。AWS 的 SDK 支持各种各样主流开发语言。这可以让你根据特定问题选择最合适的技术方案，从而避免重复“造轮子”问题。

ECS 上构建微服务架构

在构建微服务架构时面临的一个主要问题就是，如果减轻运行、维护、扩展和管理微服务架构所需大规模分布式集群资源的工作量和复杂性。目前主流的解决此问题的方案之一就是通过容器化部署和运行服务，降低运维和部署的复杂性。AWS Elastic Container Service（ECS）是 AWS 提供的一个容器管理服务，能够应对和解决微服务架构的部署和运维中面临的种种挑战。

本章节首先介绍 Amazon ECS 及其架构和工作原理，然后介绍利用 Amazon ECS 及其他相关 AWS 服务分层构建的一个高可用、高可扩展性、容错的微服务，并讨论该方案中涉及的分布式系统监控、服务发现、异步通信等问题。

ECS 简介

什么是 ECS

Amazon ECS 是一个高度可扩展伸缩、高性能容器管理服务，支持 Docker 容器[6]并能够让你轻构地在 EC2 实例构建的集群中运行应用程序。

Amazon ECS 能够让你轻松地进行安装、操作和扩展你的集群资源。使用简单的 API 调用，你就可以启动和停止基于 Docker 的应用程序、查询你的集群状态、还可以使用其他熟悉的服务，诸如安全组、ELB、EBS 卷和 AWS IAM 角色。

ECS 的特点

- **与 Docker 完美兼容：**支持 Docker 并使您能够在 Amazon EC2 实例的集群间运行和管理 Docker 容器。Amazon ECS 所管理的集群中的每个 EC2 实例都运行有 Docker 守护程序，所以无论您将何应用程序打包为本地容器，它都将部署和运行在 Amazon ECS 上，无需进行任何配置更改。
- **内置集群托管：**管理自己的容器管理基础设施通常涉及安装、操作、扩展自己的集群管理软件、配置管理系统和监控解决方案。这些系统的可用性和可扩展性架构和管理很难。Amazon ECS 消除了容器管理工作的复杂性。使用 Amazon ECS，您只需要启动容器实例集群并指定想要运行的任务即可，所有集群管理工作将由 Amazon ECS 完成。
- **编程控制：**Amazon ECS 为您提供一组简单的 API，方便您集成和扩展服务。使用 API，您可以创建和删除集群、注册和取消注册任务、启动和终止 Docker 容器，并能提供集群及其实例的状态相关详细信息。您还可以使用 AWS CloudFormation[7] 预置 Amazon ECS 集群、注册任务定义和安排容器。
- **任务调度：**Amazon ECS 包含许多任务调度程序和内置的 Placement Engine，可以根据您的资源需求（例如 CPU 或 RAM）和可用性要求在集群中放置容器。通过使用可用的任务调度程序，您可以安排长期运行的应用程序和服务以及批量作业。Amazon ECS API 还能提供完整的集群状态信息，允许您编写自己的任务调度程序或集成现有的第三方调度程序（例如 Marathon）。Amazon ECS 是共享状态的乐观并发系统，可以将集群的完

整状态呈现给所有的计划程序。通过使用 Amazon ECS API 或 Blox (一个用于容器管理和编排的开源对象集合)，您可以开发自己的任务调度程序或集成第三方调度程序。

- **负载均衡**：Amazon ECS 已与 Application Load Balancing (ALB) 集成，支持您在多个容器之间分配流量。您只需指定任务定义和要使用的 ALB，Amazon ECS 服务计划程序将自动添加和删除 ALB 中的容器。您可以在任务定义中指定动态端口，这可为安排在 EC2 实例上的容器提供未使用的端口。您还可以使用基于路径的路由与多个服务共享 ALB。
- **监控**：Amazon ECS 为您的容器和集群提供了监控功能。您可以监控正在运行的任务的 CPU 和内存使用率及总和，这些任务通过 Amazon CloudWatch 按任务定义、服务或集群进行分组。您还可以设置 CloudWatch 警报，在您的容器或集群需要扩大或缩小规模时提醒您。
- **日志记录**：您可以将各个容器实例的 ECS 代理日志和 Docker 容器日志发送到 Amazon CloudWatch 日志，以简化问题诊断。借助 AWS CloudTrail，您还可以记录所有的 Amazon ECS API 调用并将日志文件发送给您。记录的信息包括 API 调用者的身份、API 调用的时间、API 调用者的源 IP 地址、请求参数以及 Amazon ECS 返回的响应元素。
- **安全性**：Amazon ECS 支持您为每项 ECS 任务指定一个 IAM 角色。如此一来，ECS 容器实例将获得遵循“最低权限”访问策略的最小角色，允许您分别管理实例角色和任务角色。您还可以查看哪项任务使用哪个角色，这些信息均记录在 CloudTrail 日志中。

ECS 的架构及工作原理

ECS 的架构

ECS 由以下主要组件构成：

- **Task Scheduler**：任务调度器负责将 task 和 service 以 task definition 文件描述的形式按照一定的调度策略，放置到相应的 Container Instance 上进行运行。
- **Resource Manager**：Resource Manager 负责对宿主机进行通信，申请、释放并管理任务运行时所需资源。
- **ECS Cluster**：ECS 集群是一组 EC2 实例的集合，由运行 Docker 容器的 Container Instance（容器实例）组成，构成了 ECS 运行的整体资源。当运

行任务时，ECS 会根据 task definition 的信息从 ECR 或者任何其他 Docker 镜像仓库中拉取对应的镜像文件，并在集群中运行。

- **Container Agent**：Container Agent 负责 Container Instance 与集群的心跳通信，负责访问 Container Instance、健康检查、发送命令等等。默认 Container Agent 是内置在 Amazon ECS-optimized AMI 中的，当您创建 Container Instance 就已经自动安装了 Container Agent。
- **Container Instance**：Container Instance 就是安装了 Container Agent 并在 ECS 集群上进行了注册的 EC2 实例。当您运行 ECS 任务时，ECS 会根据任务调度策略将任务放置到特定的 Container Instance 上运行。

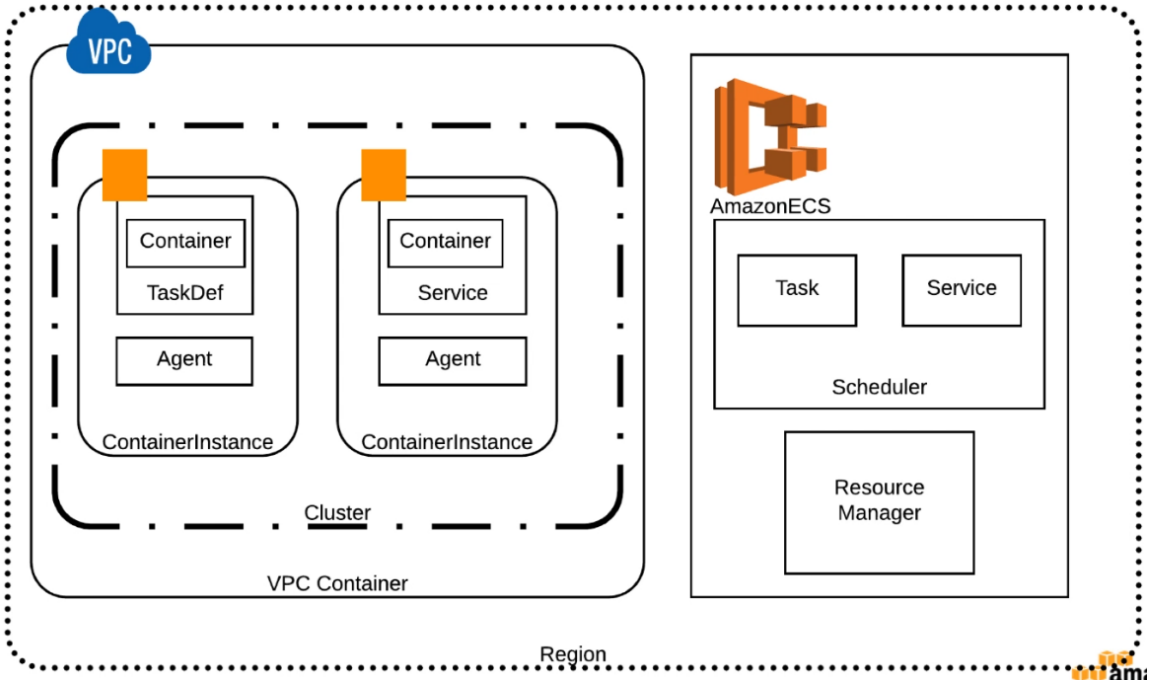

图 3：ECS 的架构

跨 AZ 保证高可用性

Amazon ECS 通过跨可用区（AZ）运行应用程序容器来保证高可用性。您需要在 VPC 中创建您的 ECS 集群，通过 task definition（任务说明）和 task service（服务说明）定义好 Docker 容器镜像文件，并其运行在您的 ECS 集群上。此外，您还可以通过 Amazon ECR，Docker Hub 或任何私有镜像仓库来存储和管理您的 Docker 容器镜像文件。

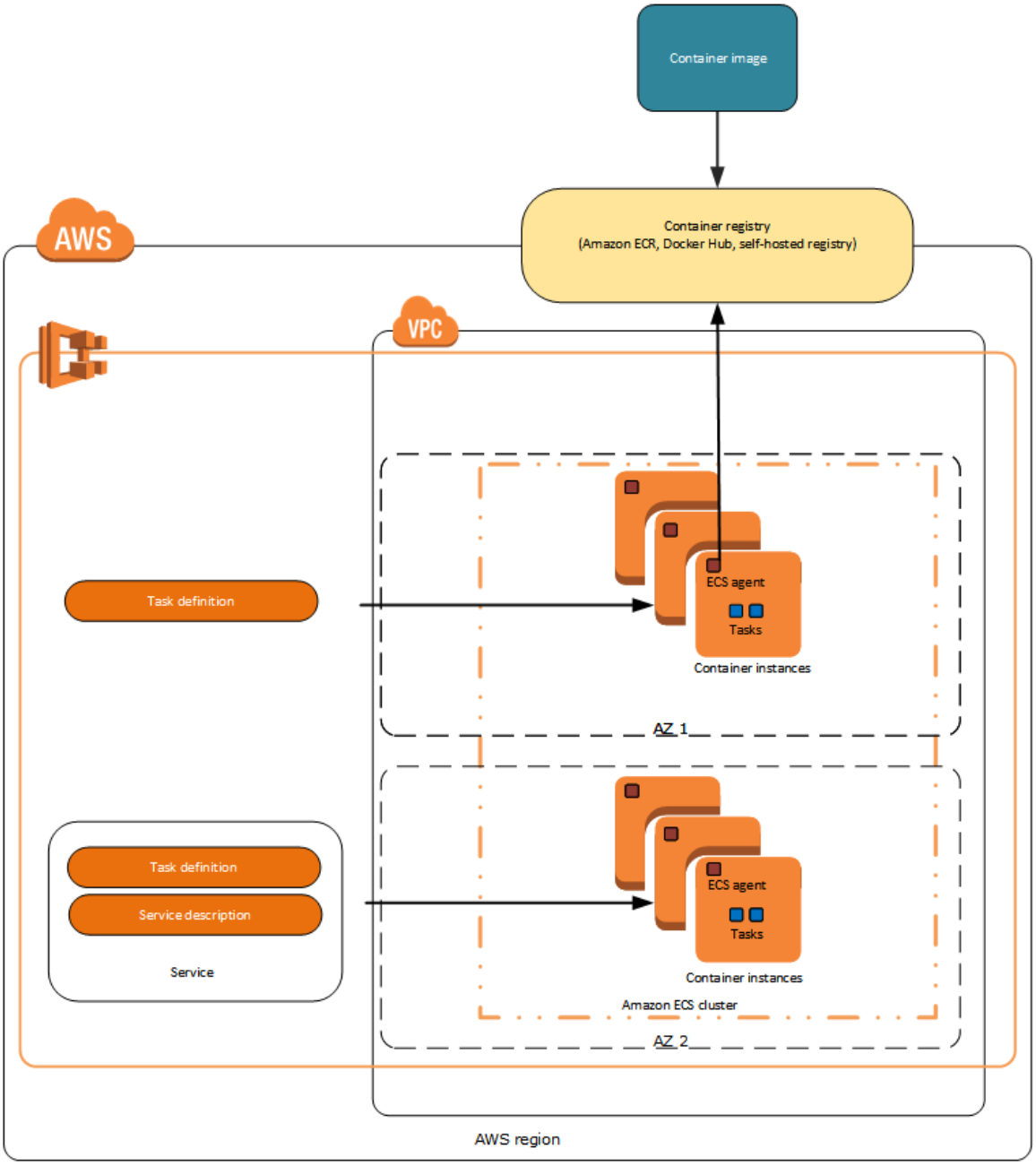

图 4：跨分区运行 ECS 集群

灵活的任务调度

Amazon ECS 将运行在容器中的作业分为两种类型：task（任务）和 service（服务）。

一个 task 通常指的是一次性的作业，比如定时触发器、批量数据处理等，task 需要由一个 task definition 文件定义其所需的资源和处理过程，并将它运行在您的集群上。您可以指定运行在您的 ECS 集群上的 task 个数。

一个 service 服务指的是永久运行在后台的作业，您同样需要在 task definition 文件中对其进行定义。Amazon ECS 允许您定义一定数量（由 desired count 参数指定）的任务进程来支持一个 service，当某个任务进程失败或者不健康，ECS 会自动启动新的任务进程，以保证该服务的任务数量达到 desired count。

那么如何将这些任务放到特定的容器实例上呢？Amazon ECS 提供的 Task Placement Engine 支持多种容器任务调度策略:

- Binpacking：最少资源利用（根据 CPU 或者内存）的实例会优先被放置，这种策略会先将任务放到可用资源最少的实例上，减少了启动过多的实例，提高了资源利用率。
- Spread：根据某个特定值进行平衡放置，这个特定值可以是一个键值对，也可以是 instanceId 或者 host。一般可使用这种策略实现跨 AZ 的任务放置。
- Affinity：组合条件的放置策略，比如两个符合条件的 task 不能同时放到某个实例中，或者两个符合条件的 task 必须同时放到某个实例中。
- Distinct Instance：将每项任务放置在不同的容器实例中。

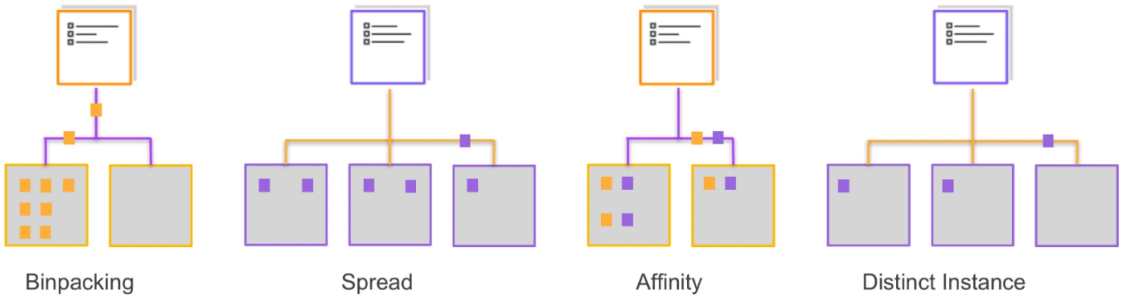

图 5：任务调度策略

此外，您可以任意组合多种调度策略，或者通过 BLOX 项目自定义您的任务调度策略。关于 BLOX，详见 https://blox.github.io/。

分层构建微服务方案

本节我们通过分布式应用中经典的 CDN 层、API 层、应用层以及数据持久层分层介绍使用 AWS ECS 及其他 AWS 服务构建一个高性能、高可用的微服务。

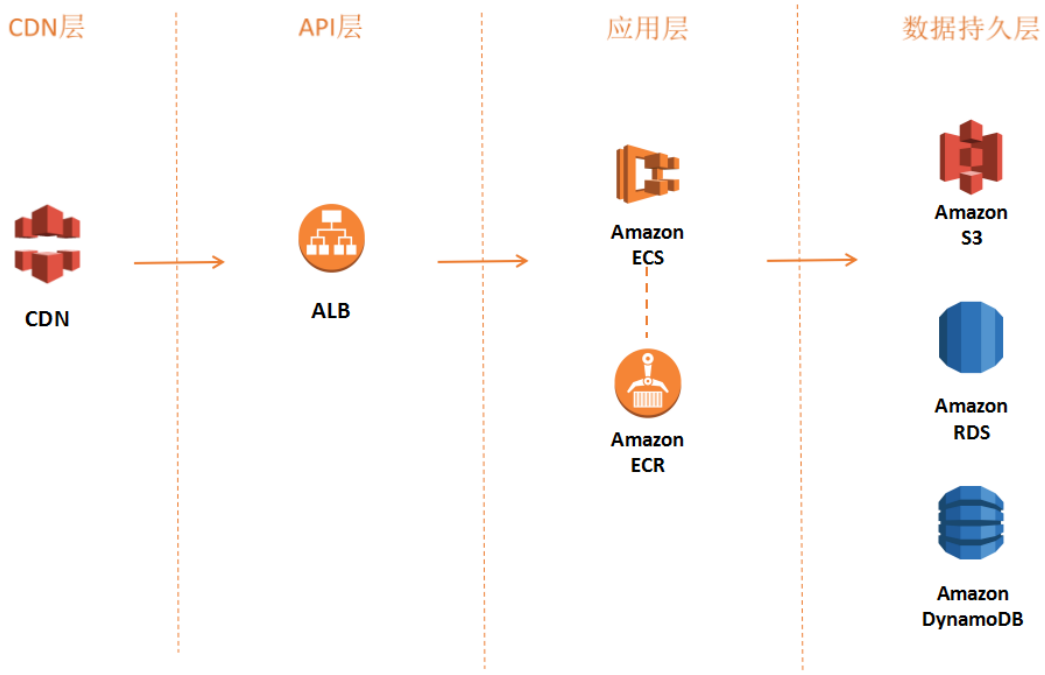

图 6：分层构建微服务

CDN 层

CDN 层的意义在于加速静态和动态资源的传输，并潜在地减轻后端服务器的压力。

使用 CDN，可以让您的微服务应用客户端从最近的端点或者代理服务器、缓存服务器上获取资源，或者可以让您通往原数据服务器的链路得到优化，从而降低延时， 加快您网站及应用上的图片、视频、脚本等数据的传输。

API 层

API 层是所有请求的入口，它可以隐藏应用程序的业务逻辑细节，通过诸如 HTTP REST API 接受外部请求。API 层一般还需要对所接收的请求进行流量管理、请求过滤、缓存及访问授权和控制。您可以使用 Application Load Balancer（ALB）来

实现您的 API 层。ALB 是 Amazon Elastic Load Balancing（ELB） 服务的一个负载均衡选项，支持您在运行于一个或多个 EC2 实例上的多个服务或容器之间基于内容定义路由规则。它可以做到：

- 基于主机的路由：您可以基于 HTTP 标头的“主机”字段路由客户端请求，以便您从同一个负载均衡器路由到多个域。
- 基于路径的路由：您可以基于 HTTP 标头的 URL 路径路由客户端请求。

ALB 提供了对容器化应用程序的支持，您可以对应用程序负载均衡器进行配置，以在单个 EC2 实例的多个端口之间对容器进行负载均衡。借助 Amazon EC2 Container Service (ECS)，您可以在 ECS 任务定义中指定一个动态端口，以便在将容器安排到 EC2 实例上时为其提供一个未使用的端口。ECS 计划程序会自动使用该端口将任务添加到 ALB Target Group。

如下图所示：您可以将 ALB 作为 API 层的入口，接收外部 API 调用请求，可以对请求进行过滤和访问控制，并根据一定的分发策略将请求分发给其代理的 ECS 集群，从而起来负载均衡的作用。关于更多 ALB 的介绍，请参考：http://docs.amazonaws.cn/elasticloadbalancing/latest/application/introduction.html

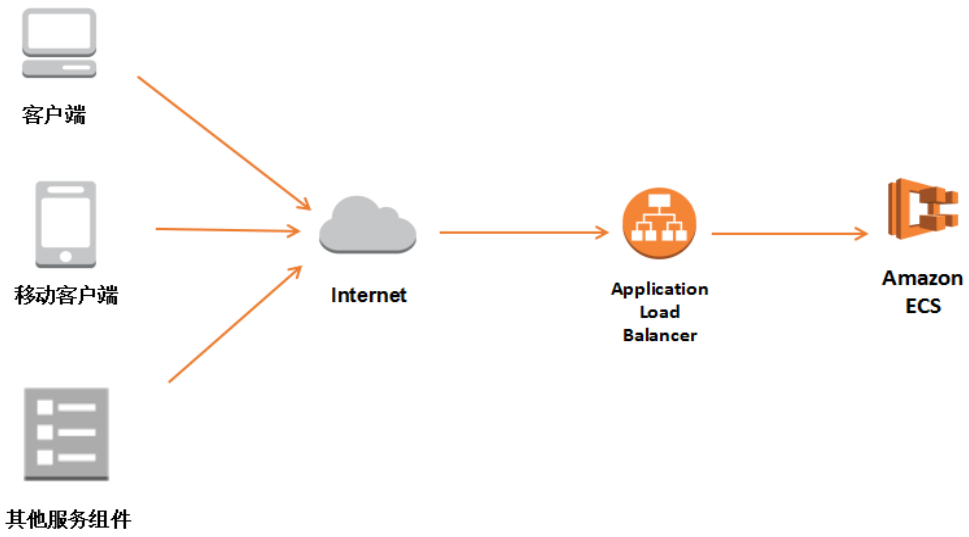

图 7：使用 ALB 实现 API 层作为服务调用入口

应用层

应用层实现了具体的应用程序业务逻辑。AWS 提供了多种方式来运行应用程序，您可以将程序直接部署到 EC2 上、或者使用 Elastic Beanstalk 服务快速构建运行环境，或者使用无服务器服务 AWS Lambda、当然也可以使用容器管理服务 ECS。这里介绍 ECS 利用容器技术运行应用层的好处。

- **一致性：**容器镜像文件具有一致性，无论您将同一个容器镜像部署到开发者的笔记本还是测试环境，抑或是生产环境，容器都会保持一致的工作。
- **灵活性：**容器技术将应用程序划分为各个独立的、松耦合的服务，这一点跟微服务架构思想是完美融合的。容器可以使微服务架构的灵活性得到更好的表现。
- **高资源利用率：**容器允许您预先定义需要使用的资源数据（比如 CPU 和内存），可以在多个下层宿主机构成的集群上共享资源，达到比虚拟机技术更好的资源利用率。
- **高工作效率：**容器具有一致性、版本公开性、松耦合和可轻松回滚等特性的可以重复使用的工作集。您不需要重复“造轮子”，无论从开发和运维上都可以提高您的工作效率。

数据持久层

数据持久层负责将业务中使用到的数据进行存储和管理。将数据管理和持久化单独封装一层，可以让应用层实现无状态化，从而便于应用层进行水平伸缩扩展和实现容错机制。

静态数据一般可以存储在 AWS Simple Storage Service（S3）服务上，并通过 AWS CloudFront 进行分发。关于 S3，可以参考 https://www.amazonaws.cn/s3/

会话数据和常用的数据，可以存储在数据缓存中间件中，比如 Memcached 或者 Redis。AWS 提供了 ElasticCache 服务管理这两种缓存服务。关于 ElasticCache，可以参考 https://www.amazonaws.cn/elasticache/

对一致性要求较高的业务数据可以存储在关系型数据库中，AWS Relational Database Service（RDS）服务可以管理六大常见关系型数据库引擎（Microsoft SQL Serve,Oracle,MySQL,MariaDB,PostgreSQL, Amazon Aurora）。关于 RDS，可以参考 https://www.amazonaws.cn/rds/

关系型数据库不利于伸缩和业务扩展，利用 NoSQL 数据库得到了更好的可扩展性、可用性和性能。您可以使用 Amazon DynamoDB 创建可以存储任意数据、应对大量访问的数据库表。DynamoDB 可以将表中数据分布到足够多的服务器上，以满足高性能、大数据量、大访问量的存储要求。关于 DynamoDB，可以参考 https://www.amazonaws.cn/dynamodb/

解决微服务架构中的常见问题

在讨论如何分层构建微服务后，通过 ECS 和 AWS 相关服务，我们已经很大程序上减轻了架构和运维微服务应用所需的工作量。本节我们将讨微服务架构中另外几个常见的问题，并结合 AWS 提供的相关服务对这些问题进行分析和处理。

服务发现

如何让服务发现彼此并进行交互，这是微服务架构必须解决的问题。服务发现包括检查服务的健康状态以及自动发现新服务上线。在 AWS 中提供了几种方式来实现服务发现：

基于 Application Load Balancer 实现的服务发现

ALB 具有健康检查和服务上线注册下线注销的功能，将这些功能与 DNS 解析服务相结合，可以低成本地轻松解决服务发现问题。

您可以给每个微服务设置自定义的域名，然后通过一个 CNAME 入口，将这些服务绑定到某个 Elastic Load Balancer 的 DNS 下。这些服务的 DNS 名将会被发布给所有需要访问服务的应用程序, 同时结合 ALB 基于路径路由的功能，能够利用 Target Group 和 Path 的结合分发给不同的微服务，节省资源和成本。

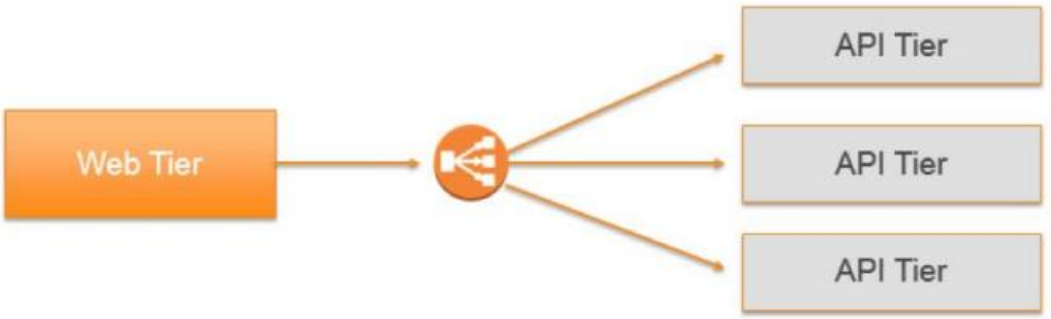

图 8：基于 Elastic Load Balancer 的服务发现

基于 Key-Value 对存储的服务发现

您还可以使用 Key-Value 键值对存储的形式来实现服务发现，虽然这种方式实现起来需要您花费比其他方式更多的时间，但是它能提供更好的灵活性和扩展性，并且不用担心 DNS 缓存问题。这种方式天生与客户端负载均衡技术相吻合，客户端负载均衡可以解决性能瓶颈并简化流量管理。

图 10 展示的是一个利用 Amazon DynamoDB 作为 Key-Value 键值存储，并结合 DynamoDB Stream 以生产者消费者模式来实现服务状态更改通知的架构。

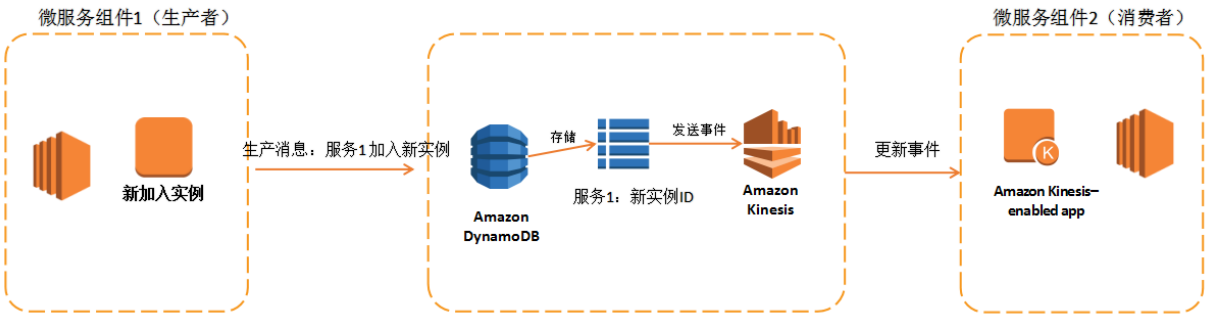

图 9：基于 Key-Value 对存储的服务发现

基于第三方开源组件的服务发现

您还可以使用开源社区中的服务发现组件来构建服务发现，例如 Hadoop 的子项目 ZooKeeper 或者 Hashicorp 公司的 Consul[8], 这种方式的好处在于合理利用了已有的开源社区中的优秀框架来提供基于 Key-Value 的特定服务，并且这些开源社区的组件对 AWS 的集成也非常对友好，能够非常快速的搭建服务发现的方案。下图 11 展示了 Consul 与 ECS 的协同工作, 宿主机将会首先利用 Registrator[9] 的 Agent 在 Docker 启动时将 Consul Agent 注册到 Consul Server，由于 Consul 基于消息订阅的模型设计，所以其他服务很快收到订阅知道新的组件上线并且能够通过其 API 了解到其他服务的状态.

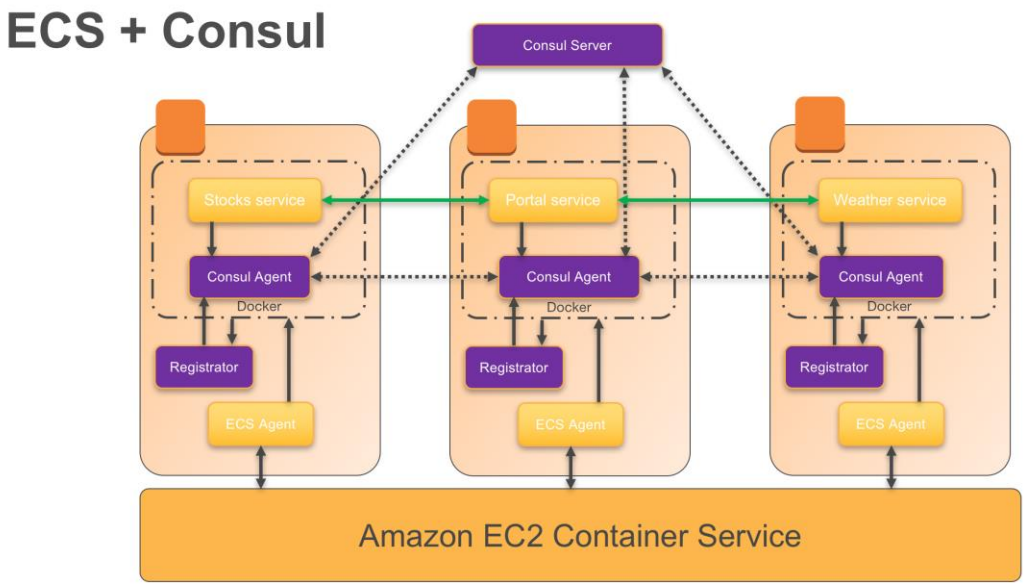

图 10：基于 Consul 服务发现

数据一致性

单体型架构的应用一般都是将数据存储在一个数据库中，应用程序中所有模块统一使用一致的数据模型。微服务则不能使用这些中心数据库，因为这与微服务去中心化、要求独立型的原则不符。每一个微服务组件都可以拥有自己的一套数据持久化方案。那么如何在服务通信过程中，保持数据的一致性呢？

通常情况下，微服务中状态的变化将会影响整个系统数据的一致性。在这种情况下，事件驱动模型被实践证明可以用于实现数据的最终一致性。事件驱动模型的核心就是将应用程序每一次状态变化当作一个事件记录下来，与持久化数据状态不同，数据是通过一连串事件流存储起来的。数据库事务日志和版本控制就是应用事件驱动模型最出名的两个例子。

在微服务场景中，事件驱动模型以“订阅者-消费者”形式进行应用程序各服务间的解耦，在各个应用不同数据系统的微服务组件之间共享事件流数据，事件驱动模型还可用于读写分离场景，提升读和写的性能。

下图展示如何利用 AWS Kinesis Stream 作为中心事件流，捕获各个服务的数据变化作为事件，并将事件流记录在 S3 上。图中各个微服务组件由 ALB、ECS 和 RDS 组成，如图所示，微服务组件 Microservice 1 由于某个状态改变，向 Kinesis

发布事件，Kinesis 将事件持久化到 S3 中（客户可以使用 Redshift 服务对这些事件流进行分析），其他微服务组件如 Microservice 2 和 3 订阅了这个事件，会拉取并过滤事件，并在本服务组件中进行相应的处理。通过这种机制，可以实现不同数据系统间数据的最终一致。

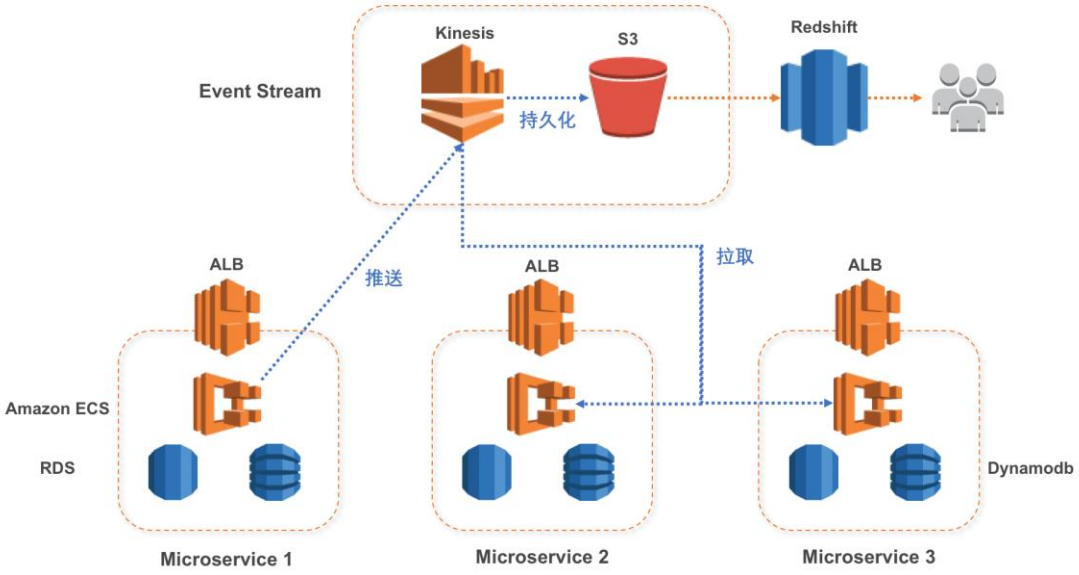

图 11：Kinesis 实现事件驱动

异步通信

在单体型架构的应用中，各个模块间的通信是非常简单的，可以是简单的方法调用或者通过内部事件发布机制。而当使用微服务架构时，服务间的通信必须通过网络远程调用。

常用的微服务通信机器除了 HTTP REST API 进行 API 调用外，对于一些对实时性要求不高，又耗时较长的通信场景，经常需要使用到消息队列实现异步通信。Amazon Simple Queue Service（SQS）和 Amazon Simple Notification Service（SNS）都是 AWS 提供的用于异步通信、通知推送的服务。

服务监控与日志记录

在微服务架构中，您需要对各个服务进行监控，以判断服务是否健康，是否可用。您还需要对服务的 API 调用记录、访问记录等日志进行记录，以实现问题发现和追踪。

您可以使用 Amazon CloudWatch 来收集系统指标，集中管理和监控日志文件，设置预警并自动适应您的运行环境中发生的变化。Amazon CloudWatch 可以监控的 AWS 资源包括 EC2 实例、DynamoDB 表和 RDS DB 实例，也可以监控任何您的应用和服务生成的自定义指标和日志文件。

监控

利用 Amazon CloudWatch，您可以轻松地对微服务组件进行监控，可以监控整个系统的资源利用率、应用程序的性能和服务的健康状态。CloudWatch 非常简单，只要进行一些简单的设置，您就可以拥有一套可靠、可伸缩、灵活的监控方案。您不再需要自己去维护一个复杂的监控系统。在微服务架构中运用 CloudWatch 的另一个好处就是，CloudWatch 提供的自定义收集指标可以让开发人员针对特定服务定制特定指标。另外，基于收集到的自定义指标，自动伸缩也可以得到轻松地实现。

记录日志

持续记录日志对于排查和发现问题起到非常重要的作用。微服务让产生的更新发布频率比以往传统架构更多，并鼓励工程师团队更多地尝试新的功能和特性。了解用户的影响对整个应用的逐步提升起来决定性的作用。

为了更好地调试和总览整个分布式系统的状况，我们必需将日志存储在一个统一的中心。在 AWS 中，您可以使用 Amazon CloudWatch Logs 来监控、存储和访问您 EC2 实例上、CloudTrail 或者其他资源上的日志文件。

集中管理日志

您可以有很多不同的选择来集中管理日志文件。大部分 AWS 服务已经“开箱即用”地集中了日志文件。在 AWS 中，首要的存放日志文件的地方就是 Amazon S3 和 Amazon CloudWatch Logs。图 13 展示的是一些 AWS 服务的日志记录能力。日志文件对于每个系统来说都是重要的部分，基本每一个系统操作都会生成一条日志记录。中心化的日志管理方案就是将所有日志文件放在一个统一的中心位置，这

样便于日志的访问、查询和分析。

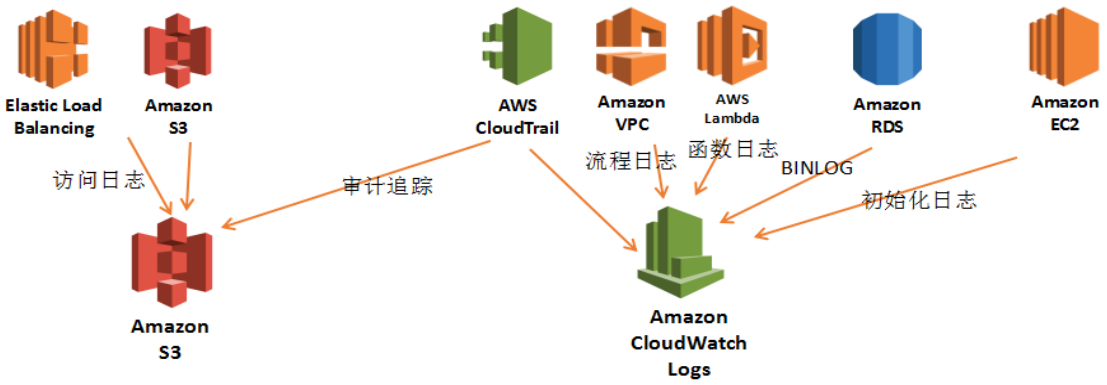

图 12：AWS 部分服务的日志记录能力

Correlation IDs（关联 ID）

在很多场景中，一个请求需要由多个微服务共同合作进行处理。想象一下，一个由数十个微服务组成的复杂的系统，当服务链中某个服务出现一个错误时的对整个服务链所造成的影响有多大。即使每一个微服务都有进行很好的日志记录，并集中化日志管理，要追踪出现的错误，需要将所有这些跟这个错误关联的日志都找出来，这是非常困难的。

关联 ID 的中心思想就是，当负责面向用户的服务收到请求时创建一个特定的 ID。这个 ID 可以放在 HTTP 头（比如，使用类似“x-correlation-id”这个的头字段）然后在服务链中传递到其他服务中去。这个 ID 就可以在各个服务独立的日志中记录，这样针对特定请求，就可以根据这个关联 ID 将所有关联的日志记录都找出来了。

为了得到日志文件中正确的服务调用顺序，可以在 HTTP 头中传递一个计数器，每经过一个服务就累加一次。

审计

在微服务架构中需要处理的另一个挑战，就是潜在的数百个分布式服务如何既确保所有用户操作的可见性，又能够在组织层面获得良好的整体感观。为了加强安全管理，对资源访问和引起系统变化的操作进行审计就显得非常重要。对系统的更改无论在独个服务中，还是在跨服务的整个系统层面，都必须进行严格追踪记录。在微服务架构中，更改是经常需要发生的，这也就让对更改的审计变得更加重要。

审计追踪

AWS CloudTrail 是一个非常有用的在微服务中追踪更改的工具，因为它能将所有在 AWS 平台上的 API 调用进行记录，并实时传送到 CloudWatch Logs 或者几分钟内传送到 Amazon S3 中。它可以记录从 AWS Management Console、AWS CLI、SDK 发起或者直接调用的记录。

所有用户的行为和自动化运行的系统行为都变成可以查询和分析的。CloudTrail 记录的信息包括用户账号、时间戳、调用的目标服务、调用者的 IP 地址、请求参数和返回响应元素。

事件和实时操作

整个系统结构中有一些系统状态的变化是必须对其进行快速响应的，必须快速进行修复或者必须及时跟进处理这些变化。Amazon CloudWatch Events 实现了一个近实时的事件系统。它可以声明一些适配某些系统状态变化的规则，并定义好自动响应这些变化的措施。

Amazon CloudWatch Events 与 CloudTrail 结合，可以为所有 AWS 服务上的所有 API 调用生成事件。它还支持自定义事件或者根据某个特定的调度策略生成事件（比如定时事件）。当一个事件发生并且符合您系统中定义的规则时，CloudWatch Events 将立即通知您预先定义好的系统中的某个角色或人物，以便他们及时做出响应措施。更好的做法是，当事件触发时，您可以定义自动化处理的工作流或者调用一个自定义的 Lambda function。

图 14 展示的是使用 CloudTrail 和 CloudWatch Events 处理在一个微服务架构系统中进行审计和系统修复的需求。所有的微服务都由 CloudTrail 进行追踪、审计并将审计信息存储在 S3 bucket 中。当一个对您系统的更改发生时，CloudWatch Events 能在 CloudTrail 基础上触发事件警告，及时通知系统管理员。您还可以对所有这些日志信息使用 Elasticsearch 等搜索引擎技术进行日志查询。

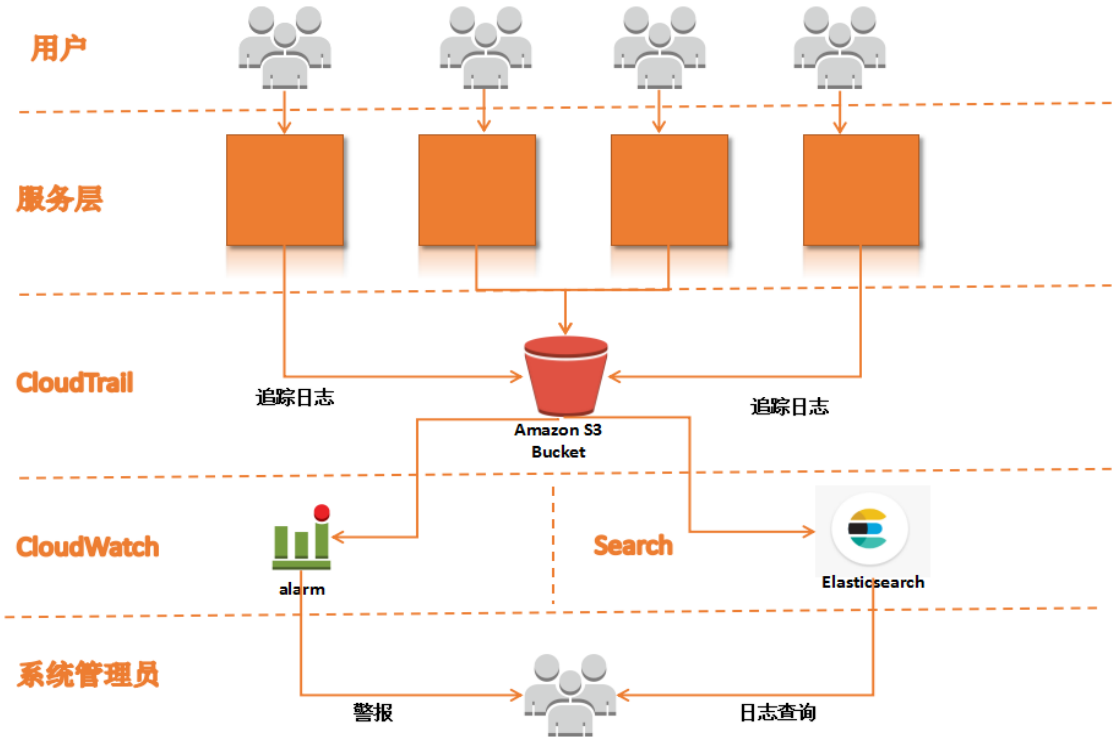

图 13：审计追踪和系统修复

总结

微服务架构可以突破传统单体型架构遇到的种种限制，它是松耦合的，去中心化的，黑盒的，多语种的，在可用性、可靠性、可伸缩性、可扩展性、安全性、性能和工作效率上更加优秀。但微服务架构也面临着像资源管理、服务发现、消息通信、数据一致性等诸多问题和挑战，利用 AWS 提供的 ECS 及其他相关服务，可以解决微服务架构面临的挑战和难题，轻松架构微服务应用，更好地拥抱微服务带来的种种便利。

客户案例

Remind

(*https://aws.amazon.com/cn/solutions/case-studies/remind/*)

Remind 是一款 Web 和移动应用程序，教师可以使用该应用程序向学生发送短信并与家长保持联系。Remind 在自己的平台上拥有 2500 万名用户和 150 多万名教师，该应用程序每月可发送 1.5 亿条消息。

“迁移到 Amazon ECS 后，我们服务的性能得到了显著提升。我们将服务响应时间的第 99 个百分位降低了 50%。” - Jason Fischl 工程师副总裁

Coursera

(https://aws.amazon.com/cn/solutions/case-studies/coursera-ecs/)

Coursera 是一家教育技术公司，致力于在全球范围内提供世界上最好的课程。该公司与世界各地的顶尖大学和机构合作，为所有人免费提供各种在线课程。Coursera 拥有来自 190 个国家/地区的 1300 多万用户，提供来自 119 个机构的 1000 多个课程，涵盖从 Python 编程到作曲的各个主题。

“借助 Amazon ECS，Coursera 可以集中精力发布新软件，无需花时间管理群集。” - Frank Chen 软件工程师

Shippable

(https://aws.amazon.com/cn/solutions/case-studies/shippable/)

Shippable 是一个提供托管式持续集成、测试和使用 GitHub 和 Bitbucket 等存储库进行部署的平台。该公司在从应用程序创建一直到向最终用户交付软件的整个过程中为开发人员和运营团队提供帮助，并在整个过程中实现完全的自动化和完全的控制。Shippable 的平台有助于消除实现重复性持续集成/持续交付和工作流中的阻碍和难题，这样，开发人员便能够专注于交付高质量代码。

“借助 Amazon ECS，我们的开发人员几乎不用花时间在运营相关工作上。过去，我们的高级开发人员需要将 80% 的时间花费在后端基础设施管理功能方面，而现在则将 80% 的时间投入在客户功能上。” - Avi Cavale 首席执行官兼联合创始人

Segment

(https://aws.amazon.com/cn/solutions/case-studies/segment/)

Segment 为企业提供一种可从枢纽中心收集客户数据，以备日后用于分析、市场营销和其他用途的服务。该公司的总部位于旧金山，拥有广泛的客户群，从初创公

司到大型企业，其中包括 Nokia、Angie’s List、Conde Nast、The Motley Fool 和 Salesforce Foundation 等。

“转而使用 Amazon ECS 大大简化了服务的运行，无需担心预配置或可用性。” - Calvin French-Owen 联合创始人兼首席技术官

mytaxi

(https://aws.amazon.com/cn/solutions/case-studies/mytaxi/)

mytaxi 基于 AWS 设计了一款采用速度快且可轻松扩展的 Docker 容器的微服务架构，以应对新年除夕等特别日子的需求高峰。该公司运行着欧洲领先的出租车应用程序，此应用程序在 40 个城市中将 1000 万用户与 45000 辆出租车联系起来。整个基础设施都构建在 AWS 之上，其中的 Amazon EC2 和 Amazon ECS 等服务可为 mytaxi 的 Docker 容器提供支持。

“2015 年 11 月，我们将 Docker 容器架构移动到了 Amazon ECS，而在 12 月，我们有史以来第一次能够庆祝新年，因为我们的系统可以处理大量请求，并且没有发生任何崩溃或中断，我们对这项成就感到极为自豪。” - Sebastian Herzberg 系统工程师

Burda Studios

(https://aws.amazon.com/cn/solutions/case-studies/burda-studio/)

BurdaStudios 是全球媒体集团 Hubert Burda Media 的一个部门，拥有近 10000 名员工和 500 多个品牌，包括家庭和生活方式类杂志以及企业间出版物。BurdaStudios 专门从事影视制作和数字出版。该公司的旗舰网站是 Bunte.de，一个在德语欧洲国家/地区 (德国、瑞士和奥地利) 处于领先地位的名人八卦门户。该网站是了解好莱坞演员、体育明星和皇室成员最新新闻的首选网站。它吸引了 600 万独立用户，月访问量高达 3000 – 3500 万次 (其独立用户的数量比最大竞争对手高出约 20%)，通过展示、视频和本地广告盈利。

“我们通过 AWS 和微服务架构所做的事情正是数字出版业的未来。” - Hansjörg Posch BurdaStudios 首席技术官

Airtime

(https://aws.amazon.com/cn/solutions/case-studies/airtime/)

使用 AWS 服务重新设计其应用程序之后，Airtime 能够以更加快速可靠的方式为客户提供社交体验，而且不会产生任何延迟。Airtime 是一家社交媒体公司，也是一款移动应用程序，可让用户在 iOS 和 Android 设备上实时分享自己喜爱的音乐、视频，并进行消息收发。该公司通过使用 Amazon EC2、Amazon Route 53 和 Amazon S3 等基本服务开始了采用 AWS 的过程，然后重新设计了自己的平台，以便能在采用 Amazon EC2 Container Service (Amazon ECS) 和 Docker 容器的微服务架构上运行。除了改善客户体验之外，Airtime 还通过滚动部署简化了内部流程，而且可以利用 Elastic Load Balancing Connection Draining 等较新的 AWS 功能。

Abema TV

(https://aws.amazon.com/cn/solutions/case-studies/abema-tv/)

AbemaTV 是一家互联网媒体服务公司，运营着日本领先的流媒体平台之一 – FRESH!by AbemaTV。目前， FRESH! by AbemaTV 可提供大约 1400 个频道和将近 37000 个节目：从广播电台的流媒体服务到由 CyberAgent (AbemaTV 的母公司) 开发的原版影视剧，应有尽有。AbemaTV 使用 Amazon Aurora 数据存储来执行其写入密集型微服务 (例如时间表和聊天)，并利用 Amazon Relational Database Service (Amazon RDS) 上的 MySQL 数据库来处理剩余的微服务 API。所有 API 都通过 Amazon CloudFront 和 Amazon API Gateway 进行管理。对于可能会影响 API 响应时间的流程，AbemaTV 实施了一套可通过 Amazon Simple Queue Service (Amazon SQS) 队列接收任务的工作程序。

“Amazon EC2 Container Service 对于我们构建具有高可用性、稳定性的微服务，以及受益于各种 AWS 服务而言至关重要。” - Akinori Yamada 工程团队

Linden Lab

(https://aws.amazon.com/cn/solutions/case-studies/linden-lab/)

Linden Lab 是一家总部位于旧金山的互联网公司，因推出 Second Life 虚拟世界而广为人知。Second Life 为用户提供了一个可生成 3D 内容并与之进行互动的平台。用户可通过 Linden Lab 的客户端程序或通过可选的第三方查看器访问 Second Life 虚拟世界。该公司的其他产品包括 Blocksworld，它让用户可以构建虚拟 3D 块并使用这些 3D 块进行游戏；还包括“Project Sansar”，这是计划于 2016 年发布的用于提供虚拟体验的新平台代码名称。

“使用 Amazon EC2 Container Service，我们可以切实地提高开发速度，从而将构建和部署时间缩短 50% 或更高。借助 Amazon ECS，我们拥有了一个非常稳定的平台，这让我们可以大幅扩展产品规模。” - Landon McDowell 运营副总裁

CyberAgent

(https://aws.amazon.com/cn/solutions/case-studies/cyberagent/)

CyberAgent 是一家互联网媒体服务公司，运营着日本领先的流媒体平台之一 – FRESH!。目前，FRESH! 可提供大约 1400 个频道和将近 37000 个节目，从广播电台的流媒体服务到原版影视剧，应有尽有。CyberAgent 使用 Amazon Aurora 数据存储来执行其写入密集型微服务 (例如时间安排和对话)，并利用 Amazon Relational Database Service (Amazon RDS) 上的 MySQL 数据库来处理其余的微服务 API。所有 API 都通过 Amazon CloudFront 和 Amazon API Gateway 进行管理。对于可能会影响 API 响应时间的流程，CyberAgent 实施了一套可通过 Amazon Simple Queue Service (Amazon SQS) 队列接收任务的工作程序。

“Amazon EC2 Container Service 对于我们构建具有高可用性、稳定性的微服务，以及受益于各种 AWS 服务而言至关重要。” - Akinori Yamada 工程团队

Meteor Development Group

(https://aws.amazon.com/cn/solutions/case-studies/meteor-development-group/)

Meteor Development Group 是一款开源的跨平台 JavaScript 应用程序平台，开发人员可以使用它为 Web、Android 和 iOS 应用程序编写代码。Meteor 由总部位于旧金山的 Meteor Development Group 创建，使用分布式数据协议和发布-订阅模式自动将更改传送给客户，无需开发人员编写同步代码。Meteor 是 GitHub 上领先的 Web 应用程序框架，每天有数以千计的公司要依靠该平台构建现代化应用程序。

“AWS 的稳定性大家有目共睹。我们面临的难题是：‘我们能够扩展运行所有客户应用程序所需的计算资源数量吗？’以及，“我们能够扩展协调所有这些资源的机制吗？”使用 AWS，我们可以对这两个问题回答‘能’。” - Matt DeBergalis 联合创始人兼产品副总裁

引用

[1] https://www.google.com/trend/explore#q=Microservices

[2] https://en.wikipedia.org/wiki/DevOps

[3] https://en.wikipedia.org/wiki/Conway%27s_law

[4] https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing

[5] https://aws.amazon.com/devops/

[6] https://www.docker.com

[7] https://aws.amazon.com/cloudformation/

[8] https://www.consul.io/

[9] http://gliderlabs.github.io/registrator/latest/