阿里云云原生微服务ACP认证培训

阿里云云原生微服务概览

阿里云

课程目标

学习完本课程后，你将能够：

1. 了解云原生的发展历程和定义
2. 熟悉云原生微服务的由来和相关技术架构
3. 掌握阿里云云原生微服务的产品体系

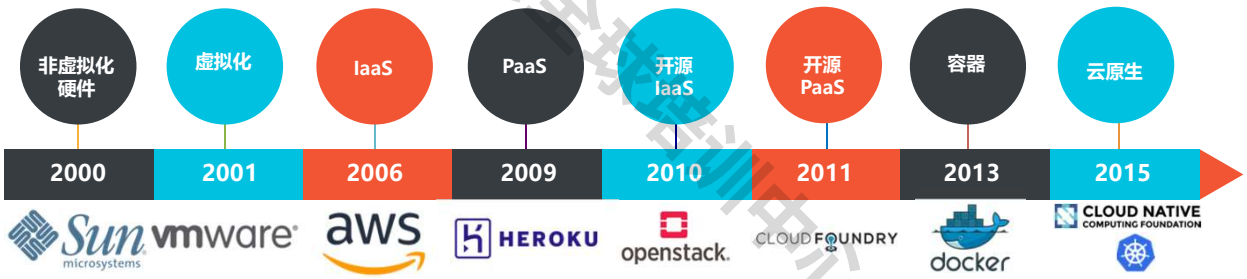
课程目录01

从虚拟化到云原生

CLOUD NATIVE COMPUTING FOUNDATION

kubernetes

- 云原生计算使用开源软件栈：
  - 根据微服务将应用进行细分，
  - 将每一部份打包放入对应的容器
  - 动态统筹管理容器，实现资源利用最大化



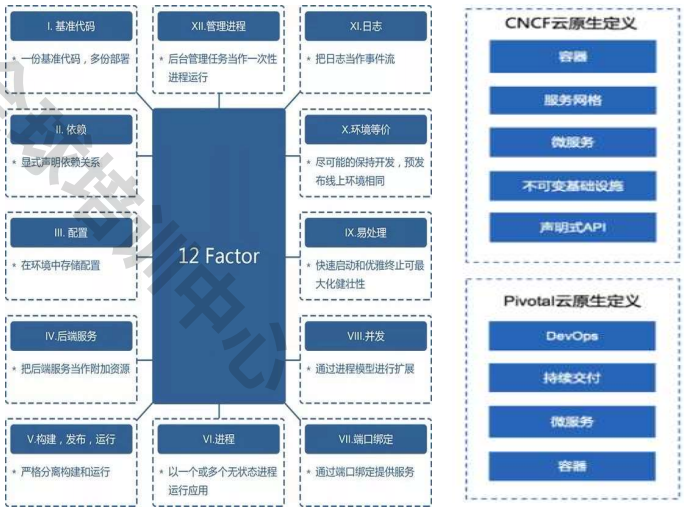
不断更新的云原生定义

很多人都会问“到底什么是云原生？”

Heroku 于 2011 年提出了**十二因子**的应用定义，该定义可以适用于任何编程语言，通常被认为是最早的云原生应用的技术特征。

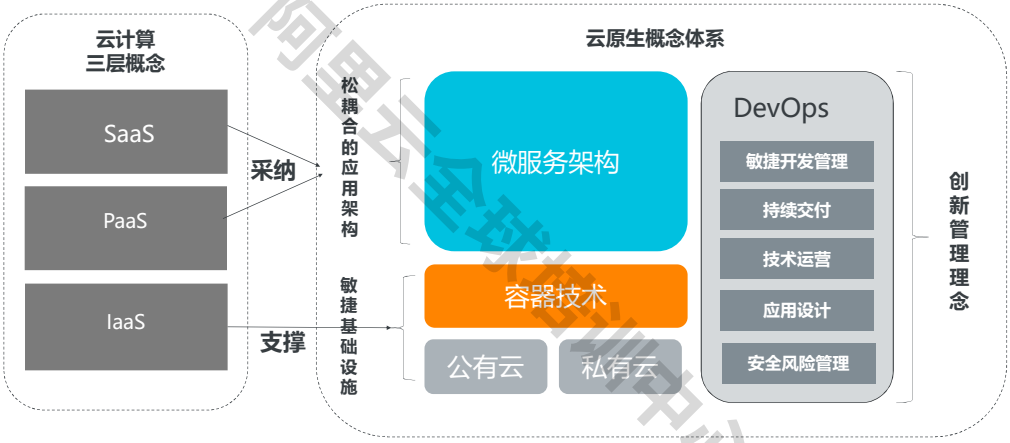
Pivotal于 2015 年提出了“Cloud Native”的概念：**云原生是一种可以充分利用云计算优势的构建和运行应用的方式。**

云原生计算基金会（CNCF）关于云原生定义的最新版本是：**云原生的代表技术包括容器、服务网格、微服务、不可变基础设施和声明式API。**



云原生是种架构模式及软件开发的新的思想理念

云原生基于云计算理念的深化，是面向云应用设计的一种新的架构设计理念，充分发挥云效能的最佳实践路径，帮助企业构建弹性可靠、松耦合、易管理可观测的应用系统，提升交付效率，降低运维复杂度。



云原生的作用及价值

释放云原生价值，推动行业数字化转型

云原生价值

从产业效能方面：

- 云原生极大的释放了云的红利，最大程度发挥云的优势
- 云原生成为驱动业务增长的重要引擎

从应用视角方面：

- 容器技术解决异构资源标准化
- 变革研发运营的生产方式提升交付效率
- 提升业务应用的迭代速度，赋能业务创新

从技术视角方面：

- 极致的弹性扩展能力，毫秒级弹性相应
- 服务自治故障自愈能力
- 大规模可复制能力，跨区域、平台快速复制

云原生的技术范畴

云应用定义与开发流程：

1. 应用定义与镜像制作
2. CI/CD
3. 消息和Streaming
4. 数据库

云应用编排与管理：

1. 应用编排与调度
2. 服务发现与治理
3. 远程调用
4. API网关
5. ServiceMesh

监控与可观测性：

1. 监控
2. 日志
3. Tracing
4. 混沌工程

云原生底层技术：

1. 容器运行时
2. 云原生存储技术
3. 云原生网络技术

云原生工具集：

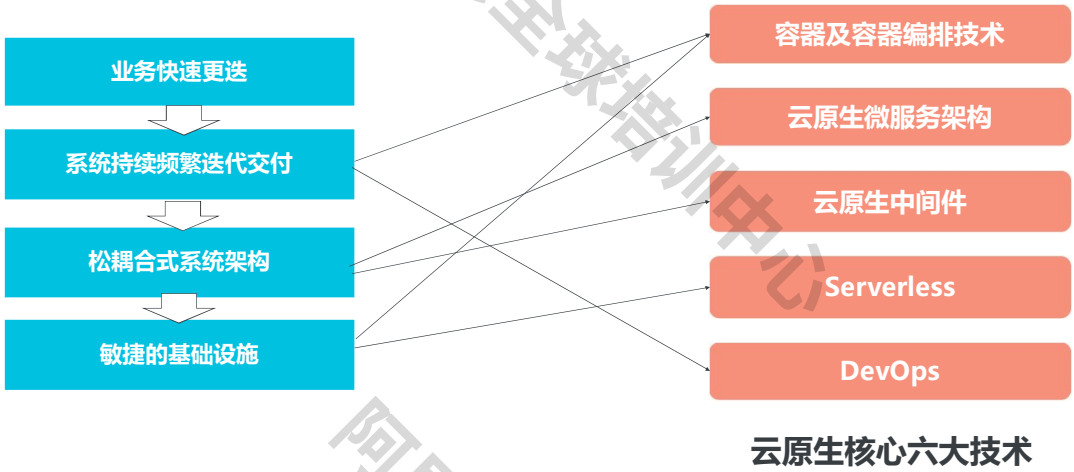
1. 流程自动化与配置管理
2. 容器镜像仓库
3. 云原生安全技术
4. 云端密码管理

Serverless：

1. FaaS
2. BaaS
3. Serverless计算

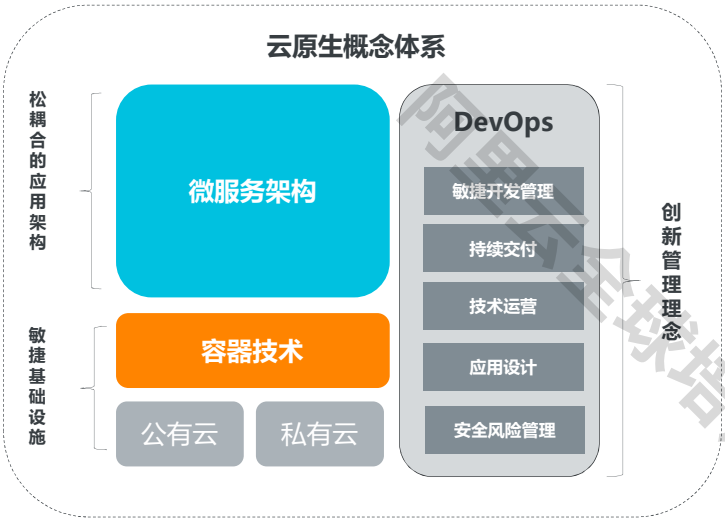
云原生技术学习主线解析

云原生的技术体系看似纷乱繁杂，但在不同视角都体现着“牵一发而动全身”的主线：



云原生核心六大技术

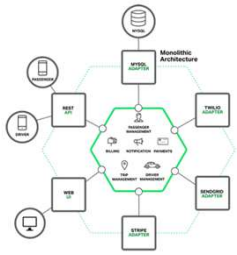
云原生的关键内涵



- **容器：**容器是一个标准的软件单元，它将代码及其所有依赖关系打包起来，使资源调度、微服务更容易，并具备强大的可移植性。
- **微服务：**微服务架构是一种架构模式，它提倡将单体应用划分成一组多个小服务，每个服务运行在其独立的进程中，服务与服务间采用轻量级的通信机制互相沟通。
- **DevOps:**高效组织团队之间如何通过自动化的工具协作和沟通来完成软件的生命周期管理，从而更快、更频繁地交付更稳定的软件。

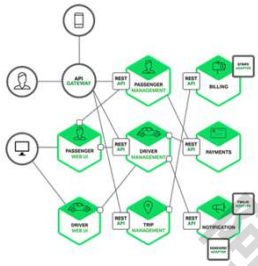
云原生微服务的架构

单体架构



- 任何变更必须部署整个应用
- 整个应用必须采用统一的技术栈
- 使用统一的数据库
- 发布阶段需要大量协调
- 所有功能在单个应用程序中，单个功能故障可能导致整个应用程序无法正常使用。

微服务架构



- **各个服务独立部署，相互协作**
- **各个服务采用独立的技术栈**
- **各个服务使用单独的数据库资源**
- **各个服务根据业务需求独立演进**
- **服务故障相互隔离。**

云原生微服务架构的特性

➢是每个微服务组件都是简单灵活的，能够独立部署。应用不需要一个庞大的应用服务器来支撑。

➢可以由一个小团队负责更专注专业，相应的也就更高效可靠。

➢微服务之间是松耦合的，微服务内部是高内聚的，每个微服务很容易按需扩展。

➢微服务架构与语言工具无关，自由选择合适的语言和工具，高效的完成业务目标即可。

课程目录02

阿里云对云原生的断言

成为云计算的新界面

- 向下封装基础设施，屏蔽底层架构差异性
- 链接异构算力，云边端一体化
- 向上分布式支撑工作负载，容器+K8S逐步成为云计算的新界面

重塑软件生命周期

- 解决研发生命周期上的挑战，包括研发、CI/CD等
- 云原生正在逐步改变并优化软件的开发和交付模式

加速信息产业转型升级

- 云原生带来了“一切即服务”的蓝海
- 使得任何IT能力都可以变成基于云的服务供企业使用，
- 从而推动软件产业的转型与升级

云原生成为释放云计算红利的最短路径

阿里十五年云原生实践

2006年-2015年
应用架构互联网化

2016年-2019年
核心系统全面云原生化

2020年-至今
全面升级下一代云原生技术

2006
自主研发互联网中间件

2009
淘宝和天猫合并建设业务中台，三大中间件核心系统上线

阿里云正式建立，自研飞天操作系统**开启云化时代**

2011
T4项目启动，**容器调度技术开始支撑集团在线业务**，云原生时代开启

2013
淘宝最后一台小型机下线，**自研飞天操作系统全面支撑集团业务**

2015
云原生技术全面商业化，容器技术对外开放

2017
实现规模化混合部署，底层资源池统一，支持百万级规模电商交易

2019
阿里巴巴核心系统100%上云，百万容器支撑双十一

2020
成立云原生技术委员会，**云原生升级为阿里技术新战略**，阿里巴巴核心系统全面使用云原生产品支撑大促。

阿里巴巴云原生技术全面升级，Severless时代开启

阿里云的云原生产品家族

阿里云拥有国内最丰富的的云原生产品家族，最全面的云原生开源贡献，最大规模的云原生应用实践，最大的容器集群和客户群体。

微服务产品家族

阿里云微服务是基于容器技术，全面集成Kubernetes，并深度支持 Dubbo 和Spring Cloud等微服务框架的一站式微服务解决方案。

应用 PaaS 及微服务产品

企业级分布式应用服务 EDAS 3.0
企业级云原生应用 PaaS 平台，提供 ECS/K8s 多语言应用生命周期管理及开源增强的微服务治理能力。

微服务引擎 MSE
以组件方式提供各类微服务能力，包括微服务依赖组件托管、无侵入的微服务治理，帮助用户更快捷，稳定、低成本的运行微服务。

应用工具

应用实时监控服 ARMS
云原生一体化可观测性平台，提供全栈式的性能监控和端到端的全链路追踪诊断能力。

性能测试 PTS
真实模拟多用户高并发访问业务场景，结合监控、流控等产品提供一站式高可用能力，高效检验和管理业务性能。

应用高可用服务 AHAS
提供应用架构探测感知、故障注入式高可用能力评测和流控降级高可用防护能力。

Prometheus 监控服务
基于开源 Prometheus 构建的数据监控服务，完全托管数据大盘，存储和报警能力，开箱即用。

阿里云微服务产品特点：

- 易接入
- 易运维
- 易管理

微服务组件围绕分布式、微服务体系，重点建设服务集成和整合能力，进而实现 PaaS 平台开放的生态体系。

消息队列产品家族

消息队列 RocketMQ 版

消息队列 Kafka 版

消息队列 RabbitMQ版

微消息队列 MQTT 版

阿里云消息服务 MNS

事件总线 EventBridge

Serverless产品

函数计算

函数计算（Function Compute）是一个事件驱动的全托管 Serverless 计算服务。

Serverless 应用引擎

Serverless 应用引擎（Serverless App Engine，简称 SAE）是面向应用的 Serverless PaaS 平台。

查看详情

FC（函数计算） 是一个事件驱动的全托管 Serverless 计算服务，用户无需管理服务器等基础设施，只需编写代码并上传，函数计算会准备好计算资源，并以弹性、可靠的方式运行业务代码。

SAE（Serverless 应用引擎） 实现了 Serverless 架构 + 微服务架构的完美融合，真正按需使用、按量计费，节省闲置计算资源，同时免去 IaaS 运维，有效提升开发运维效率；SAE 支持 SpringCloud、Dubbo 和 HSF 等流行的微服务架构。

阿里云微服务的典型应用场景

微服务架构场景

实现敏捷开发和部署落地，加速企业业务迭代

企业生产环境中，通过合理微服务拆分，将每个微服务应用存储在阿里云镜像仓库帮您管理。您只需迭代每个微服务应用，由阿里云提供调度、编排、部署和灰度发布能力

- 负载均衡和服务发现
  支持4层和7层的请求转发和后端绑定。
- 丰富的调度和异常恢复策略
  支持服务级别的亲和性调度，支持跨可用区的高可用和灾难恢复
- 微服务监控和弹性伸缩
  支持微服务和容器级别的监控，支持微服务的自动伸缩

云途时代基于阿里云中间件构建微服务架构

云途时代基于阿里云中间件（EDAS、PTS、AHAS、链路追踪等）构建了移动电影院应用，实现“分区+分众”精准放映；支持数字电影（DCI）行业2K标准分辨率在移动端放映；绑定 VR 一体机，提供移动巨幕体验；移动 3D 配合 VR 眼镜，实现 3D 效果；业务规模达 300+ 应用数，峰值实例数达500+，高峰 QPS 达 40000+。

课程目录03

阿里云的云原生开源贡献

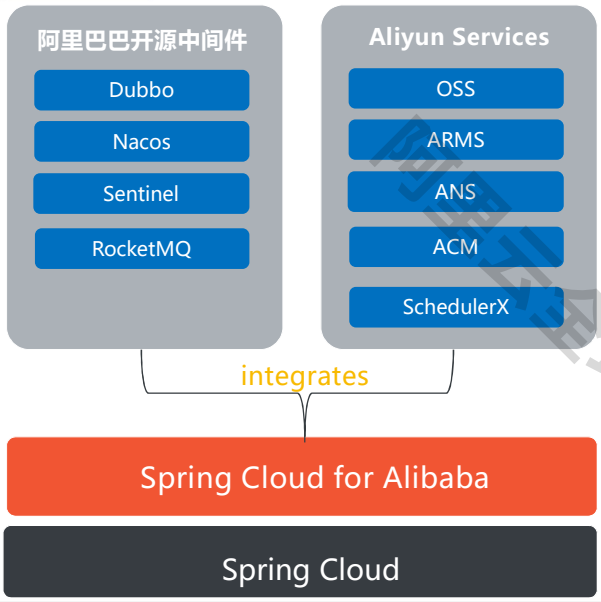
阿里云一直致力于回馈社区、积极拥抱开源，是国内在云原生领域的开源贡献最全面的科技公司，涵盖编排调度、作业管理、无服务器框架等。

作业管理	Streaming & Messaging	应用定义与镜像构建	CI/CD
• OAM • Kruise 作业管理 • Kubeflow/Arena AI • Argo pipeline 共建	• RocketMQ	• Helm 中国站（App Hub） • 应用 YAML 托管 • 云原生应用管理工具	• 云原生持续交付（CD） 插件
调度编排	**元信息存储**	**服务发现与RPC**	**Service Mesh + Gateway**
• Kubernete 核心上游投入 • Kubernetes 多租户插件 • Kubernetes 阿里云集成 • Cerebellum 高性能调度器	• etcd + etcd operator	• Dubbo • Nacos	• Envoy 增强插件 • Tengine • Isitio 上游共建
多云	**无服务框架**	**容器运行时**	**工具**
• Kubernetes 多云插件	• X Run Serverless 框架 • Virtual Kubelet	• Containerd、Kata • PouchContainer	• ChaosBlade • Dragonfly

云集成

• Alibaba Cloud controller manager	• CNI Terway • CSI	• Cluster AutoScaler • Cluster API	• 安全集成 RAM Authenticator • KMS plugin

阿里开源中间件Spring Cloud Alibaba



阿里 Spring Cloud 微服务开发的一站式解决方案

- 帮助用户使用阿里的开源组件构建微服务体系
- 帮助用户快速上云，从开源产品切换到云产品

优势：

- 完全遵循Spring Cloud规范，开箱即用
- 开源组件成套系，且有双11背书。

Spring Cloud Alibaba 与其他开源中间件的对比

	Spring Cloud Netflix	Spring Cloud 官方	Spring Cloud Zookeeper	Spring Cloud Consul	Spring Cloud Kubernetes	Spring Cloud Alibaba
分布式配置	Archaius	Spring Cloud Config	Zookeeper	Consul	ConfigMap	Nacos
服务注册/发现	Eureka	——	Zookeeper	Consul	Api Server	Nacos
服务熔断	Hystrix	——	——	——	——	Sentinel
服务调用	Feign	OpenFeign RestTemplate	——	——	——	Dubbo RPC
服务路由	Zuul	Spring Cloud Gateway	——	——	——	Dubbo PROXY
分布式消息	——	SCS RabbitMQ	——	——	——	SCS RocketMQ
负载均衡	Ribbon	——	——	——	——	Dubbo LB
分布式事务	——	——	——	——	——	Seata

总结与回顾

本章节主要讲述内容：

1. 云原生微服务的发展历程和技术范畴，以及云原生微服务的架构。
2. 云原生关键内涵主要包括微服务、容器技术和DevOps。
3. 阿里云云原生产品体系的组成，以及各个产品体系涉及的云服务。
4. 阿里云在云原生开源的贡献和开源中间件Spring Cloud Alibaba。

阿里云

阿里云云原生微服务ACP认证培训

企业级分布式应用服务EDAS的介绍

阿里云

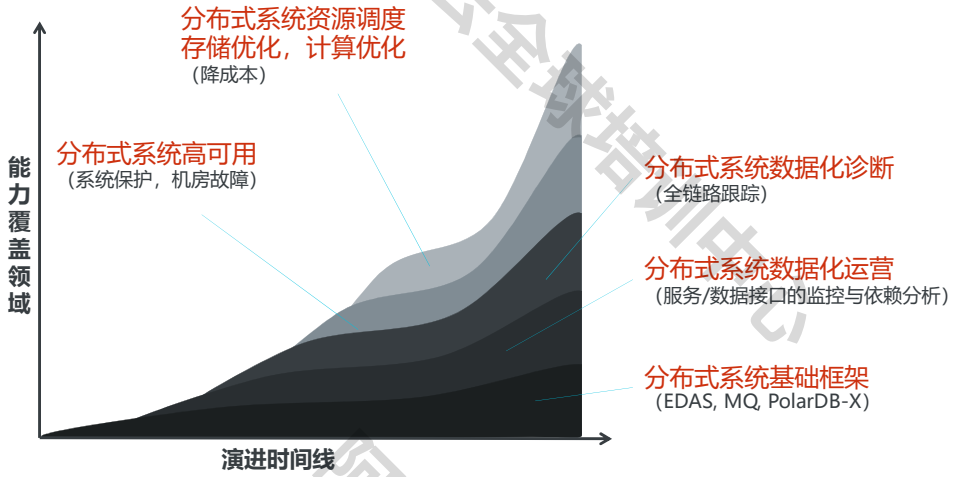
课程目标

学习完本课程后，你将能够：

1. 了解企业级分布式服务EDAS的产生背景和技术原理
2. 掌握企业级分布式服务EDAS的功能特性
3. 熟悉企业级分布式服务EDAS的基本使用和最佳实践

课程目录01

历经阿里10年技术沉淀打造的Aliware产品



基于阿里云中间件Aliware的业务架构

业务创新快　数据标准化　数据实时化　服务线性扩展　业务能力沉淀　业务能力开放

业务应用

PC　移动APP　物联网设备　第三方接入

上层业务应用　...　...　...　...　Aliware 云服务总线CSB

共享业务服务

业务PaaS能力

用户中心　商品中心　库存中心　营销中心　交易中心　支付中心　物流中心　评价中心　……

技术PaaS产品

EDAS 企业级分布式应用服务
SAE　Serverless应用引擎
CSB　云服务总线
GTS　全局事务服务

阿里云中间件服务
消息队列 Rocket 版
消息队列 Kafka 版
消息队列 MQTT 版
消息队列 AMQP 版

ARMS 业务实时监控
PTS　应用性能测试
AHAS　应用高可用服务
APDS　应用发现服务

IaaS

基础设施 – ECS / SLB / VPC / ACK / OSS / ACS / SLS
阿里云公共云、混合云、专有云

企业级分布式应用服务EDAS的定义

企业级分布式应用服务EDAS(Enterprise Distributed Application Service)是一个**应用托管和微服务管理的 PaaS 平台**，提供应用开发、部署、监控、运维等全栈式解决方案，同时支持Dubbo、Spring Cloud 等微服务运行环境，助力您的各类应用轻松上云。

多样的应用托管平台

- 独享实例的ECS集群
- 基于Kubernetes的容器服务集群
- 自建Kubernetes集群

丰富的微服务框架

基于以下开发应用和服务，并托管到EDAS中

- 原生Dubbo
- 原生Spring Cloud
- HSF

完整的应用管理

- 应用生命周期管理
- 服务治理
- 微服务管理

全面的监控和诊断

- 实时监控应用中资源和服务状态
- 实时监控Iaas层资源和服务状态
- 通过日志和诊断组件快速定位

企业级分布式应用服务EDAS的核心能力

提供完整的应用管控、微服务治理、系统高可用的解决方案

企业级分布式应用服务EDAS的定位

以应用为中心的 PaaS 产品，构建企业级在线应用运行环境。

定位：构建企业级在线应用运行环境

企业级分布式应用服务EDAS的典型应用场景

单体应用微服务化改造

解决应用耦合度高、难扩展，运维效率低，业务创新慢的问题，适用于电商、游戏类应用

应用云上托管

提供云上自动部署和运维能力，解决手工部署效率低、错误率高、升级困难、业务中断、监控定位难等应用运维问题，适用于政企IT系统及互联网类应用

混合云场景

提供公管私的的能力，统一管控使用公有云IAAS和本地数据中心部署的EDAS。适用于金融类应用、互联网、视频、游戏应用。

智能运维

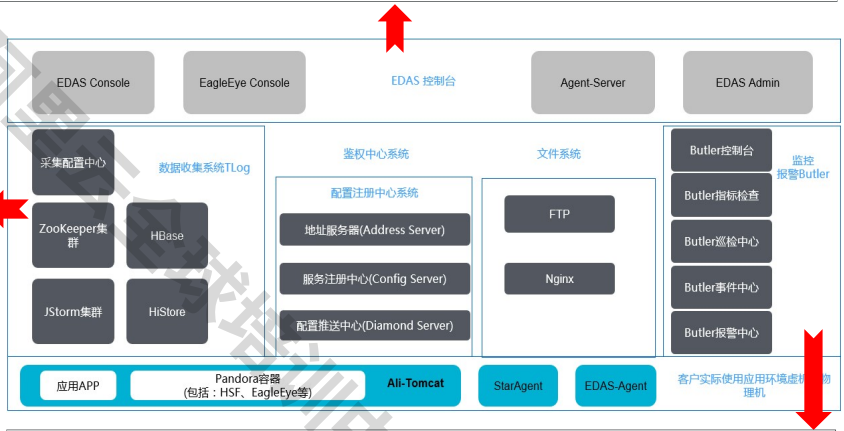
提供应用拓扑全栈可视的能力。提供全方位立体化监控和数据化运营，帮助企业智能运维。

课程目录02

企业级分布式应用服务EDAS的产品架构

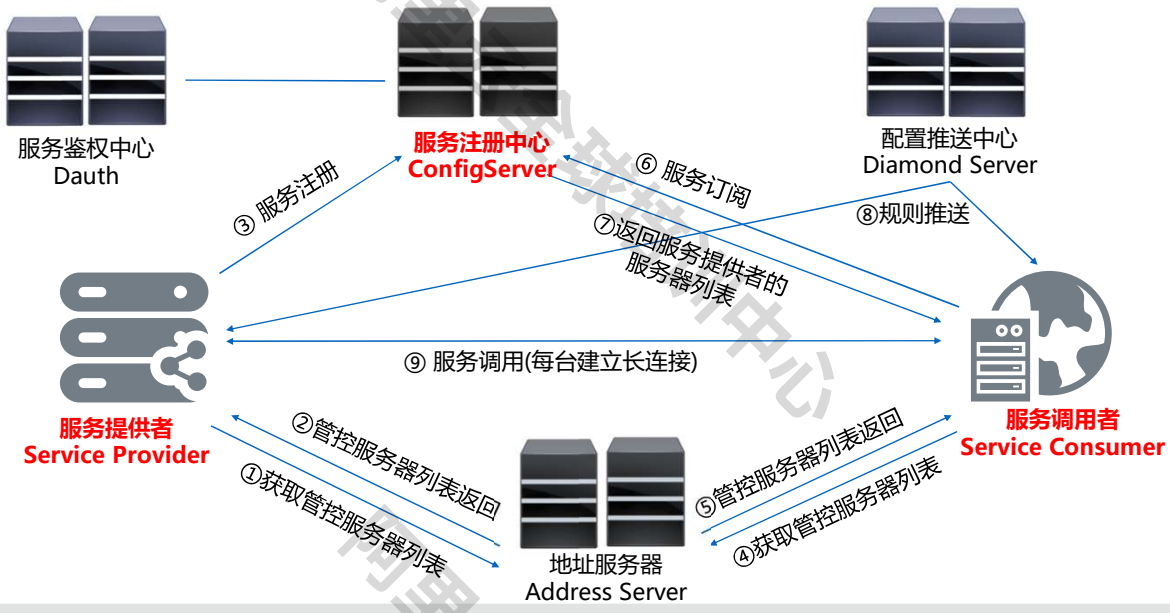
- **EDAS控制台**：EDAS系统功能的操作界面，是唯一可以让客戶直接使用的系统。用戶通过控制台可以 实现资源管理、应用生命周期管理、运维管控及服务治理、立体化监控及数字化运营等。

- **数据收集系统 Tlog**：实时收集EDAS集群及所有客戶应用机器的系统运行状态，调用链日志等，并进行实时汇总计算和存储。
- **配置中心注册系统**：服务发布及订阅的中心服务器，也是分布式配置配置推送的中心服务器。
- **鉴权中心系统**：对用戶的数据进行权限控制，以保证各个用戶的数据安全。
- **文件系统**：用于存放用戶上传的WAR包及JDK、Ali-Tomcat等必须组件。



- **监控报警系统**：EDAS对外输出的主要日常监控及报警工具。

企业级分布式应用服务EDAS的微服务管控



企业级分布式应用服务EDAS的产品功能

服务化

基于高性能分布式服务框架，IT系统沉淀共享资产，新需求基于共享服务层快速集成

应用生命周期管理

以应用为中心的中间件PaaS平台，提供一键应用部署与扩容，简化用户操作

立体化监控

从IaaS、应用和业务等多个层面实现对系统的全面立体的监控报警

运维管控

内置多种运维与管控工具，帮助用户实现服务治理、应用诊断

EDAS 功能一：服务化

EDAS 的服务化是基于高性能分布式服务框架，通过 IT 系统沉淀共享资产，新需求能基于共享服务层快速集成。

- 微服务管理
  - ✓ 支持微服务管理及服务治理
  - ✓ 支持多款主流微服务框架
- 对高性能、容错和服务安全性的保障
- 服务基础设施
  - ✓ 任务调度服务
  - ✓ 配置推送服务
  - ✓ 分布式事务支持

EDAS分布式服务兼容多种主流微服务框架

支持 Spring Cloud 和 Dubbo 等多种轻量级微服务架构，无需改变开发模式，可以零代码修改迁移到 EDAS 平台

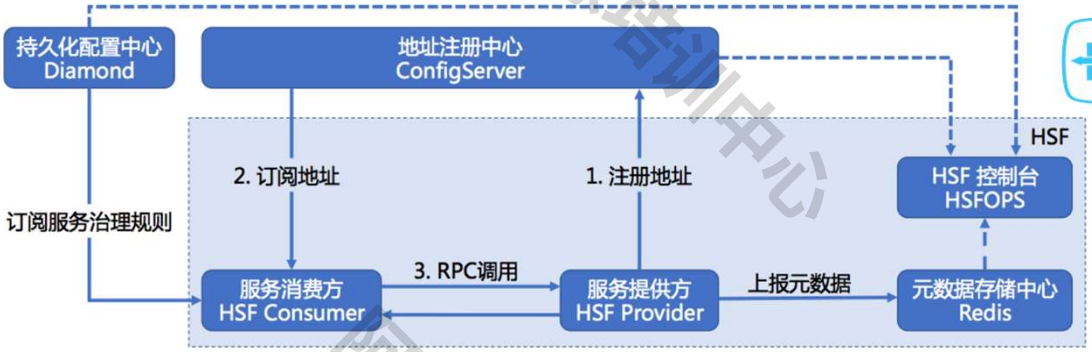
- HSF是在阿里巴巴内部广泛使用的分布式 RPC 服务框架
- 高性能网络通信
- 透明无侵入代理
- 强大的服务管控
- 天猫双11十年极致考验

- 完全兼容 Spring Cloud 开发模式
- 支持采用 Spring Boot 和 Spring Cloud 开发应用
- 无需修改代码，即可自动对接到 EDAS 内部的企业级中间件服务
- Jar 包方式部署运行

- 开源 Dubbo 项目持续维护
- Dubbo 成为 Apache 孵化项目
- GitHub 项目 22000+ 关注
- 连续多年国内最受欢迎开源软件 Top 5
- 核心 RPC 功能和阿里内部保持一致

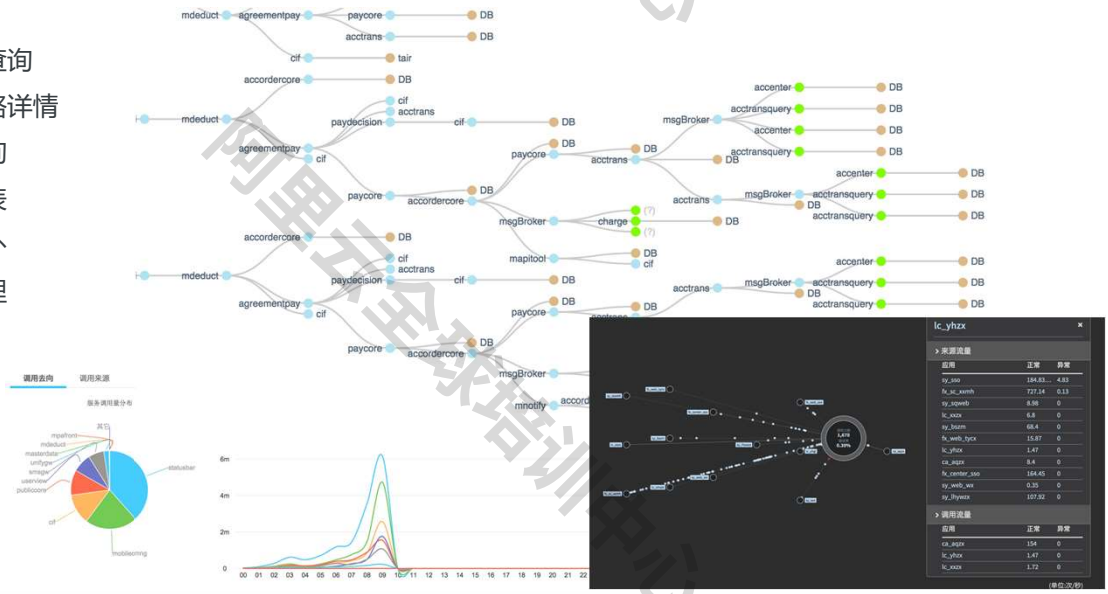
EDAS分布式服务兼容多种主流微服务框架

- 支持三大主流微服务框架 HSF、Apache Dubbo 和 Spring Cloud
  - 零代码入侵，完成 Apache Dubbo 和 Spring Cloud 应用上云
  - 无需构建 ZooKeeper、Eureka 和 Consul 等



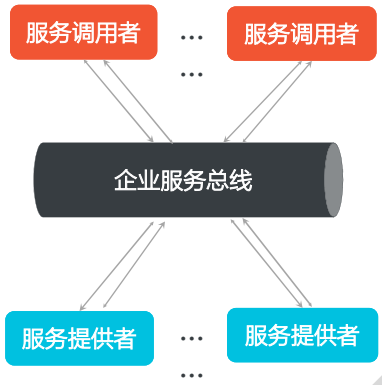
EDAS的微服务管理

- 调用链查询
- 调用链路详情
- 服务查询
- 服务报表
- 服务拓扑
- 服务治理

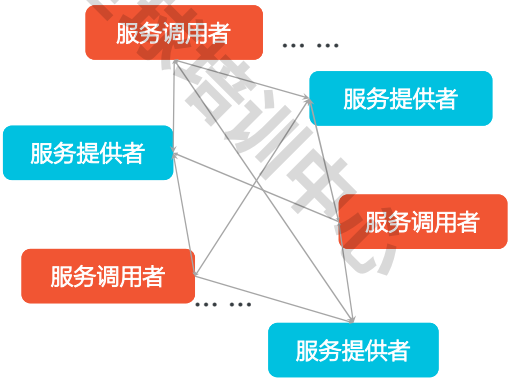


EDAS的高性能设计

传统“中心化”系统架构

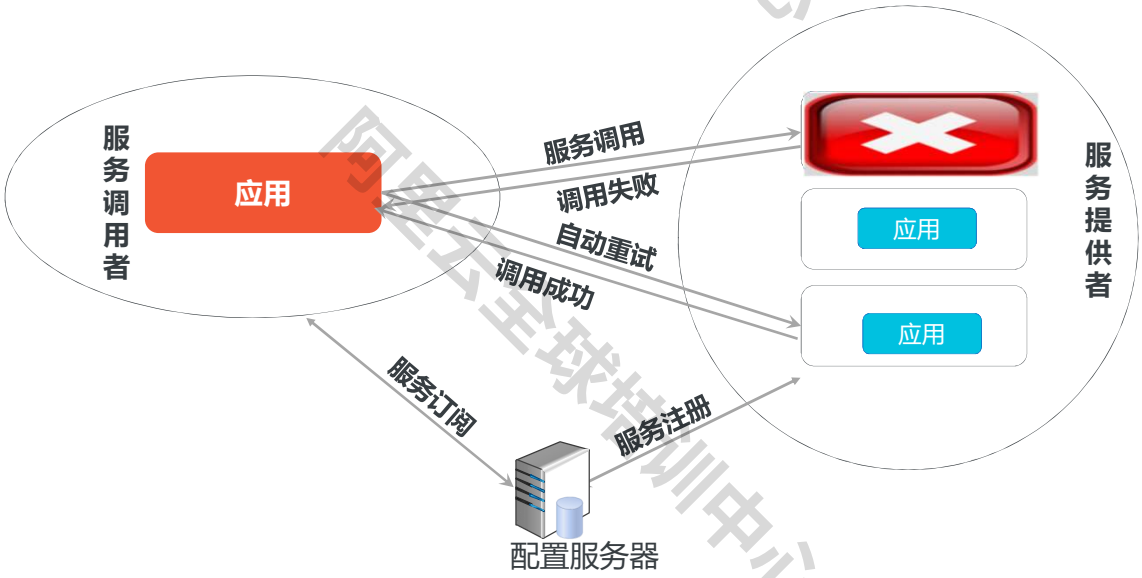


阿里“去中心化”系统架构



- 日均千亿次调用
- 峰值25亿次/分

EDAS对容错的支持



EDAS的服务安全性

基于服务组保证服务调用的安全性

- 发布、订阅和调用服务时必须使用合法的安全令牌
- 授权数据会下发到服务机器，避免性能瓶颈

EDAS服务化基础设施

分布式任务调度服务

允许用户配置任意周期性调度的单机或者分布式任务，秒级触发，高可用容灾，历史调度查询。适用于诸如每天凌晨2点定时迁移历史数据等任务调度场景

目前在阿里内部，任务个数多达数十万个

实时配置推送服务

提供高效的分布式配置管理，能够将分布式系统的配置信息在EDAS控制台上集中管理起来，做到一处配置，处处使用，秒级推送

目前在阿里内部，日均推送数亿次

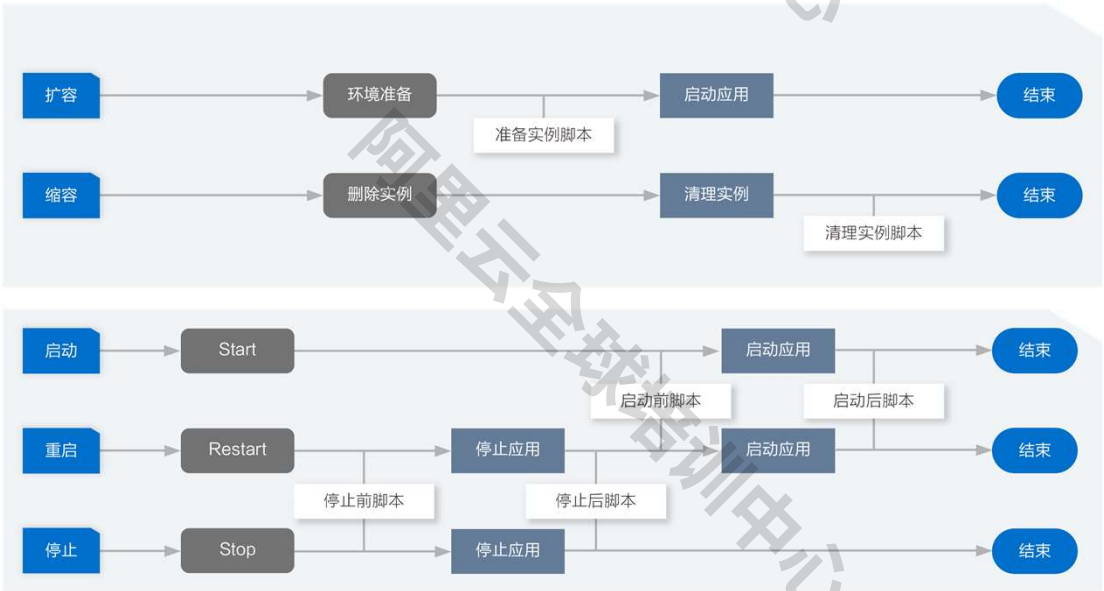
分布式事务处理

阿里巴巴自主研发分布式事务组件TXC，突破性的解决了分布式事务处理的难题，保障每一次分布式服务调用、消息收发和数据库操作都有事务保障。简单易用业务无侵入

EDAS 功能二：应用全生命周期管理

- 应用生命周期
  - ✓ 大规模集群应用管理
  - ✓ 应用Beta，分批发布
  - ✓ 弹性伸缩
  - ✓ 流量引导与灰度发布
- 账号与权限
  - ✓ 主子账号
  - ✓ 自定义角色
  - ✓ 资源组

设置应用生命周期挂载脚本

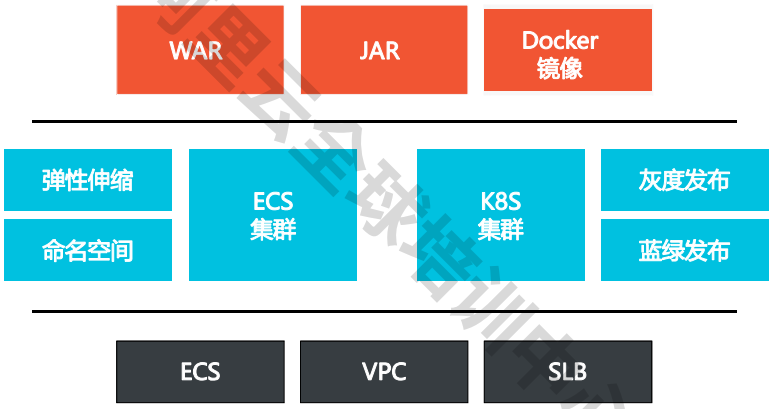


应用托管类型

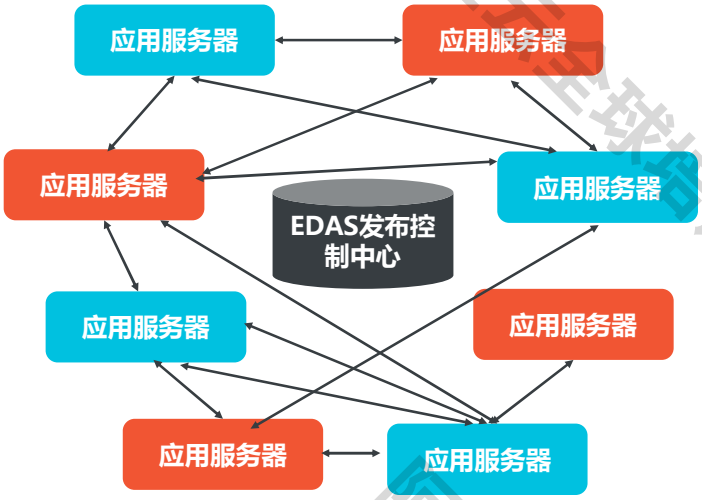
打包方式	WAR/JAR	WAR/JAR/镜像	WAR/JAR/镜像
可选集群	ECS集群	容器服务 Kubernetes集群	EDAS Serverless
应用	Spring Cloud、Dubbo、HSF		

应用部署

- 支持WAR 包、 JAR 包、Docker镜像等方式将应用部署至 EDAS集群，还可以使用可以在控制台或插件、云效等工具将应用部署至 EDAS集群



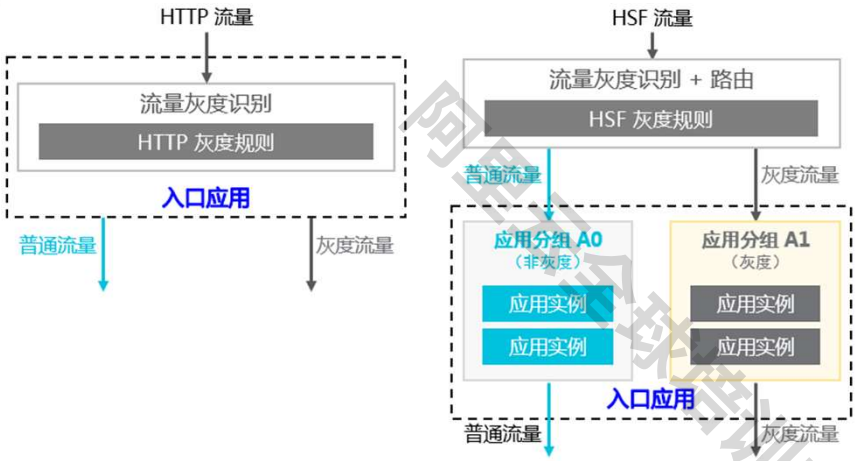
应用发布能力



- **单个应用上千台**机器的快速发布能力
- 应用**分批发布**能力
- 应用**灰度发布**能力

阿里云

灰度发布和流量管控

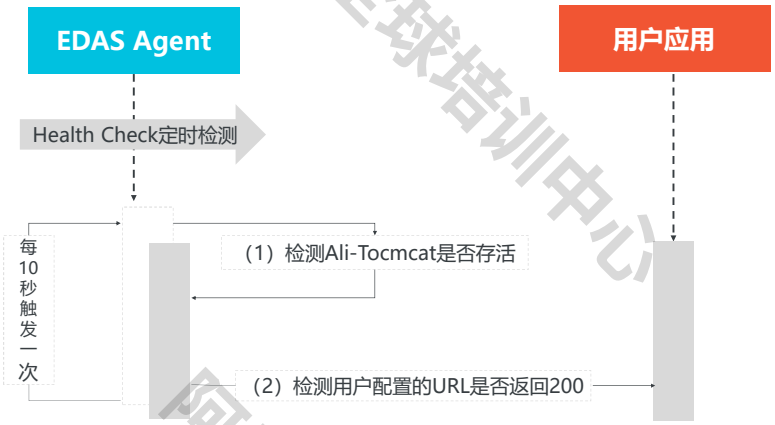


- 只需为要求灰度的部分应用准备实例资源，不用将业务系统整体再搭建一套
- 支持多重灰度，允许不同的应用有各自不同的灰度控制，甚至允许同一个应用同时参与多个灰度控制
- 支持链路灰度，允许多个应用同处于一个灰度控制，上游环节识别出来的灰度流量，经过非灰度的中间应用环节，在下游环节仍可路由到对应的灰度应用实例。

阿里云

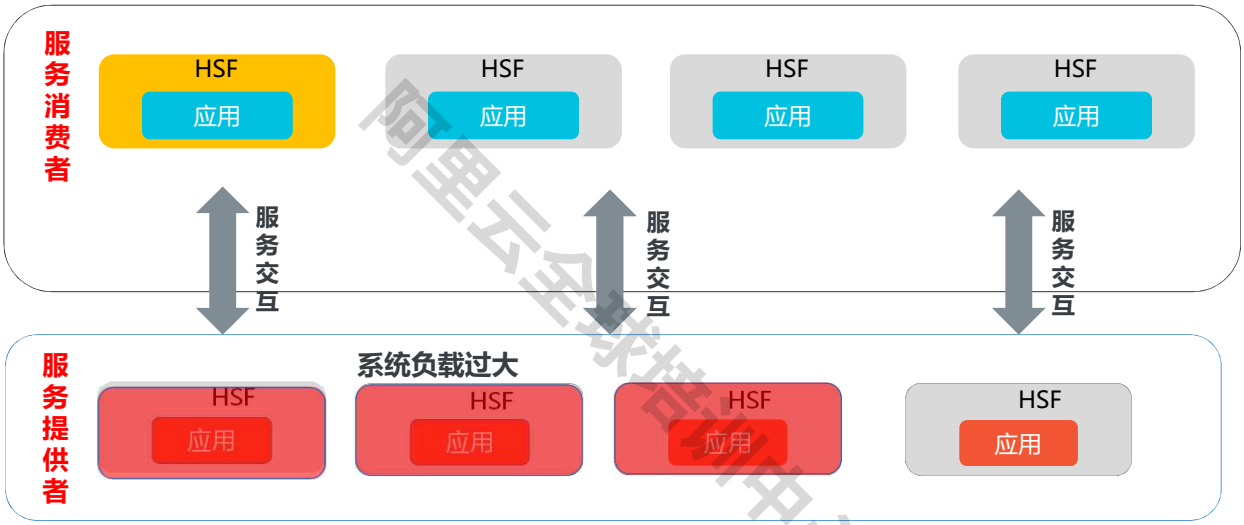
应用的健康检查

健康检查是指由 EDAS Agent（以下简称为 Agent）针对容器与应用进行定时检查与汇报，然后将结果反馈在控制台上的过程。健康检查能帮助您了解集群环境下整个服务的运行状态，从而为审查与定位问题提供帮助。



阿里云

应用服务弹性伸缩支持



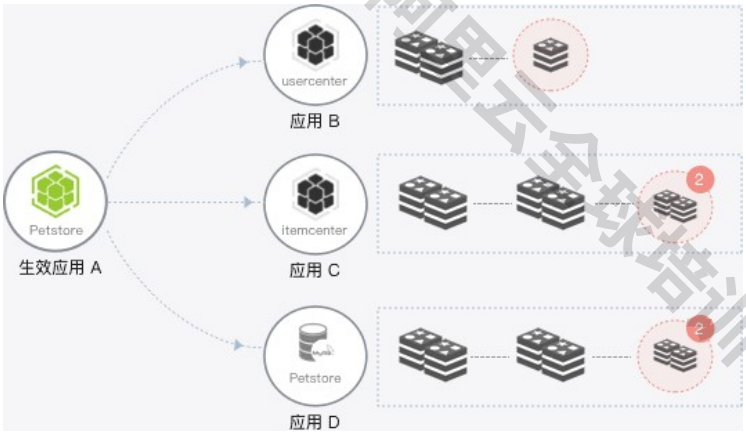
阿里云

应用服务弹性伸缩支持

线性扩展
秒级生效

离群实例的摘除

在微服务架构中，当服务提供者的应用实例出现异常，而服务消费者无法感知的时候，会影响服务的正常调用，并影响消费者的服务性能甚至可用性。离群实例摘除功能会检测应用和服务实例的可用性并进行动态调整，以保证服务成功调用，从而提升业务的稳定性和服务质量。

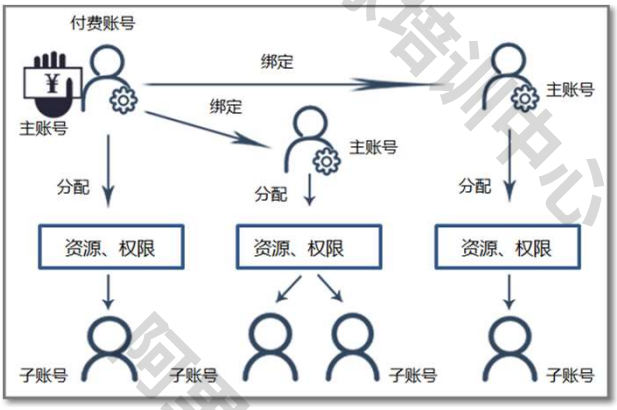


- 当提供者应用的异常实例数量过多（超过摘除实例比例上限）时，仅按照设置的比例摘除。
- 当提供者应用中仅剩最后一个可用实例时，即使错误率超过配置的阈值，也不会摘除该实例。

账号与权限的管理

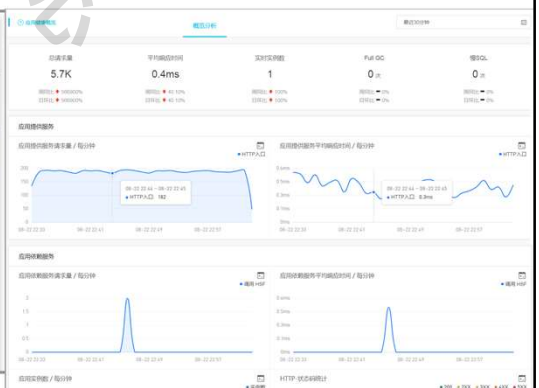
EDAS 提供了完善的主子账号管理体系。主账号可以分配权限和资源给多个子账号，实现按需为用户分配最小权限，从而降低企业信息安全风险，减轻主账号的工作负担。

接入阿里云访问控制（Resource Access Management，RAM）的账号体系之前，EDAS 本身拥有一套严谨的主子账号体系，实现了人权分离。2016年7月份升级后，EDAS 也同时支持了 RAM 主子账号体系。

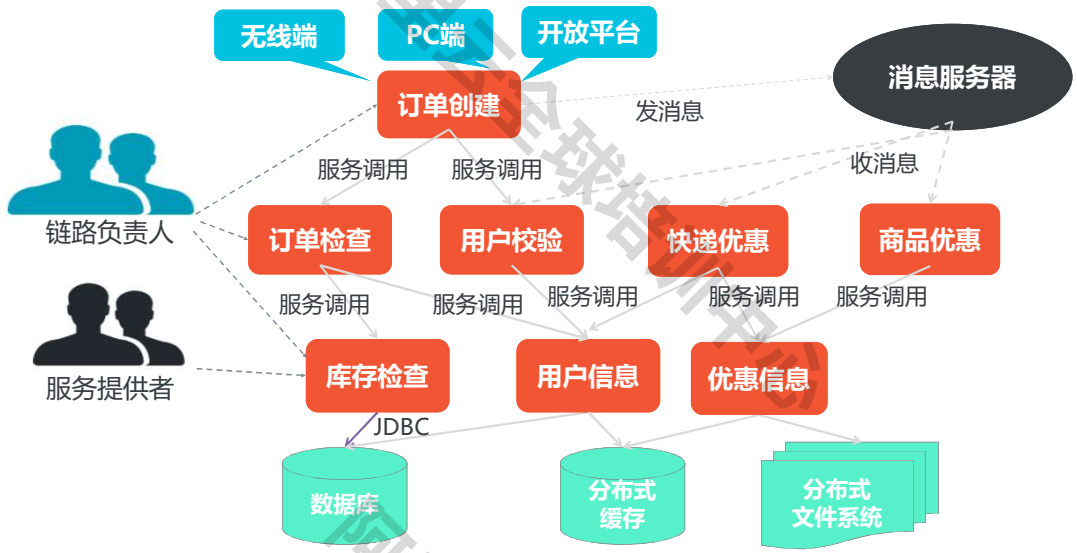


EDAS 功能三：立体化监控

- 立体化监控
  - ✓ 分布式调用跟踪
  - ✓ 应用监控
  - ✓ IAAS基础监控
  - ✓ 容器层监控
  - ✓ 业务监控大盘
  - ✓ 定制化报表大盘



大规模分布式服务调用的链路跟踪



鹰眼跟踪系统

2013-07-20 05:25:28,179 ERROR taobao.hsf - 基于 RPC 协议调用服务[com.taobao.wireless.trade.api.tmall.hsf.TmallBagInterface:1.0.0]的[bulidConfirmOrder]方法时出现错误:
所调用的服务目标地址为：[...]
参数信息为：[...]
TraceId=ac18287913742691251746923
错误原因为超时，请查看服务器端的执行日志是否也超时，执行时间为：3000毫秒。
HSFTimeOutException at com.taobao.hsf.....HSFResponseFuture.getResponse(HSFResponseFuture.java:52) at com.taobao.hsf.....SyncInvokeComponent.invoke(SyncInvokeComponent.java:51) at ...

链路分析报表

- 对海量调用链进行统计，得到链路各个依赖的稳定性指标

层次	名称	QPS	峰值QPS	调用比例	被调用均值	平均耗时	耗时比例	出错率	同机房	标记
0	http://[redacted]	82.66	8700	1.0000	1.0	312ms	16.52%	0.00%	0.0%	瓶颈
1	[redacted]	1.83	3445	0.0221	11.07	0ms	0.01%	0.00%	100.0%	
1	[redacted]	439.22	2220	5.3138	5.45	0ms	0.86%	0.02%	99.98%	
1	[redacted]	0.21	3660	0.0025	1.02	2ms	0.00%	0.00%	100.0%	
2	[redacted]	0.21	3660	0.0025	1.02	0ms	0.00%	0.00%	100.0%	
2	[redacted]	0.13	20	0.0016	1.0	1ms	0.00%	0.00%	100.0%	
1	[redacted]	80.61	8480	0.9752	1.0	5ms	1.49%	0.56%	100.0%	
2	[redacted]	0.0	130	0.0000	1.0	10ms	0.00%	0.00%	98.0%	
2	[redacted]	0.01	190	0.0001	2.13	10ms	0.00%	0.00%	81.25%	
2	[redacted]	0.01	130	0.0001	2.27	0ms	0.00%	0.00%	100.0%	
2	[redacted]	0.01	190	0.0001	2.13	0ms	0.00%	0.00%	100.0%	
1	[redacted]	79.45	8440	0.9612	1.0	0ms	0.08%	0.00%	100.0%	
1	[redacted]	0.85	60	0.0103	1.09	5ms	0.01%	0.01%	100.0%	强依赖 错误阻塞
1	[redacted]	0.15	520	0.0018	1.0	107ms	34.29%	0.00%	100.0%	瓶颈
1	[redacted]	0.08	30	0.0010	1.0	2ms	0.00%	0.00%	100.0%	
1	[redacted]	0.15	520	0.0018	1.0	0ms	0.00%	0.00%	100.0%	

对依赖的压力

易故障点

瓶颈点

阿里云

应用监控

test100 返回实例列表

基本信息
运行日志
基础监控
应用监控
通知报警
服务列表
限流降级
容器诊室
软件版本
操作记录

应用监控

系统概要 | HTTP入口 | 提供的HSF服务 | HSF调用来源 | HSF调用依赖 | ONS消息

HTTP入口数据统计汇总

QPS趋势 show	时间	all QPS	all 环比	all 耗时	zone1 QPS	zone1 环比	zone1 耗时	zone2 QPS	zone2 环比	zone2 耗时
	10:16	12793	↓1%	16.48	6843	↑3%	16.62	5950	↑3%	16.62
	10:15	12918	↑3%	16.62	7382	↑3%	16.62	5536	↑3%	16.62
	10:14	12511	↓0%	16.63	5334	↓0%	16.63	7177	↓0%	16.63
	10:13	12555	↓10%	16.59	4321	↓10%	16.57	8234	↓10%	16.57
	10:12	13894	↑0%	16.63	6754	↑0%	16.62	7140	↑0%	16.62

提供的HSF服务数据统计汇总

QPS趋势 show	时间	all QPS	all 环比	all 耗时	zone1 QPS	zone1 环比	zone1 耗时	zone2 QPS	zone2 环比	zone2 耗时
	10:16	12793	↓1%	16.48	6843	↑3%	16.62	5950	↑3%	16.62
	10:15	12918	↑3%	16.62	7382	↑3%	16.62	5536	↑3%	16.62

阿里云

IAAS 基础监控

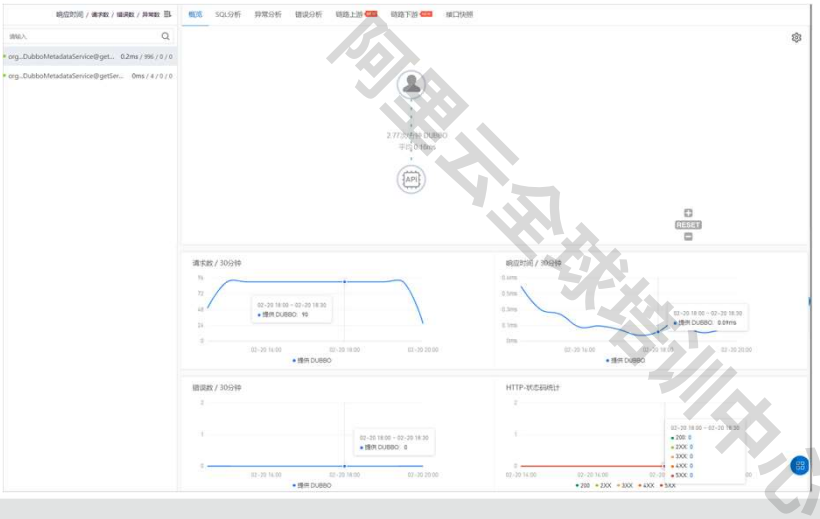
- CPU、内存、负载、网络、磁盘
- 实时~一周
- 单机、集群

网络速度数据统计汇总

查看大图	时间	发送速度 平均值	最大值--ECS实例	最小值--ECS实例
	07:20	2.5KB/秒	2.5KB/秒--i-28w7u7lkc	2.5KB/秒--i-28w7u7lkc
	07:19	2.98KB/秒	2.98KB/秒--i-28w7u7lkc	2.98KB/秒--i-28w7u7lkc
	07:18	2.97KB/秒	2.97KB/秒--i-28w7u7lkc	2.97KB/秒--i-28w7u7lkc
	07:17	3.38KB/秒	3.38KB/秒--i-28w7u7lkc	3.38KB/秒--i-28w7u7lkc
	07:16	3.39KB/秒	3.39KB/秒--i-28w7u7lkc	3.39KB/秒--i-28w7u7lkc

服务和接口监控

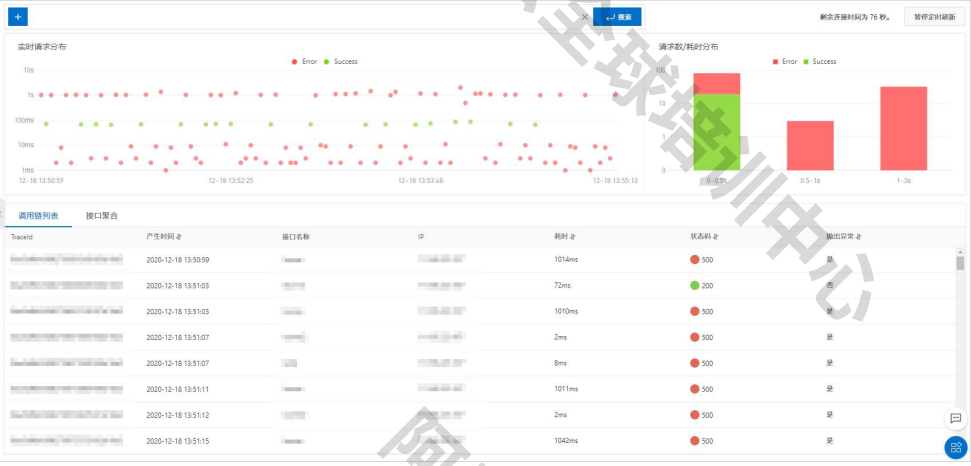
- 服务和接口监控功能用于监控应用下的接口调用详情，包括SQL分析、NoSQL调用分析、异常分析、错误分析、链路上下游和接口快照。



- 查看服务的详细调用拓扑，以及请求数、响应时间、错误数的时序曲线。

应用故障的实时诊断

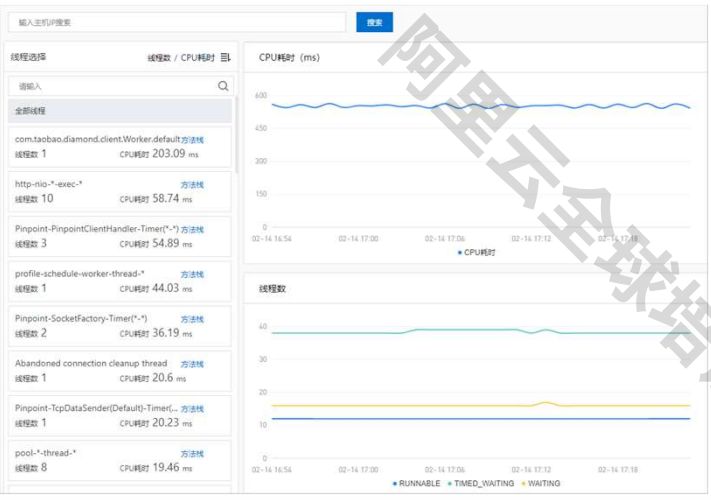
- 实时诊断功能适用于在短时间内密切监控应用性能和定位问题原因的场景。



- 实时诊断将于自动开启5分钟后自动终止。

应用故障的线程分析

- 线程分析功能提供线程粒度的CPU耗时和每类线程数量的统计，并且每5分钟记录一次线程的方法栈并聚合，可真实还原代码执行过程，快速定位线程问题。



应用故障的异常分析

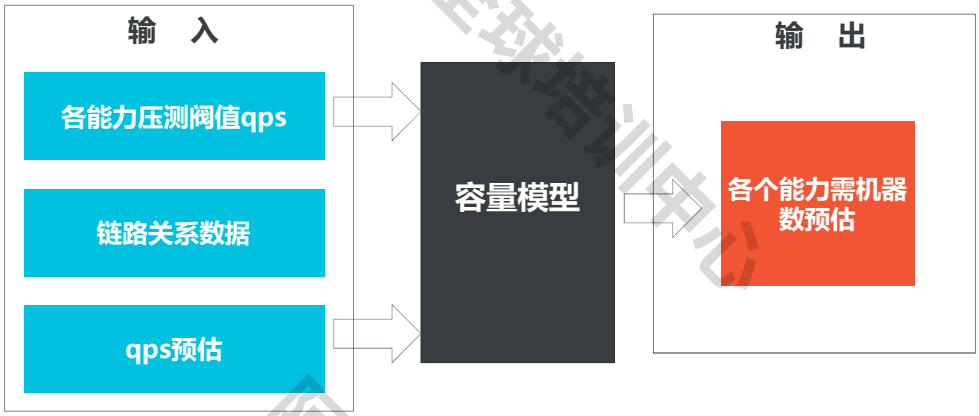
- 应用异常分析功能，包括异常数量统计、每类异常次数统计以及异常发生的端口等线程分析

EDAS 功能四：运维管控

- **服务治理**
  - ✓ 容量自动化压测
  - ✓ 容量规划
  - ✓ 限流降级
- **智能应用诊断**
  - ✓ 实时日志
  - ✓ 容器运行时诊断
- **组件中心**
  - ✓ 微服务组件
  - ✓应用诊断组件
  - ✓其他组件

链路容量计算

基于调用链查询的结果，查看慢业务或出错业务的调用链的详细信息，进行依赖梳理，包括识别易故障点、性能瓶颈、强依赖等问题；也可以根据链路调用比例、峰值 QPS 评估容量。



容量压测与规划

线上压测，容量规划轻松获取线上机器运行性能指标和实时运行水位。

预估水位　当前水位　加减机器

名称	类型	机房数	机器数	标准	单机负荷	单机能力	集群负荷	集群能力	水位	需机器数	机器增减	PE	操作
test1	type1	1	9	70.0%	0	11	0	98	0%	0	-9		实时容量
test2	production	1	323	70.0%	0	931	0	300778	0%	0	-323		实时容量
test3	testing	3	206	40.0%	0	1072	0	220842	0%	0	-206		实时容量
test4	development	1	212	70.0%	0	280	0	59296	0%	0	-212		实时容量

限流降级

EDAS 中的限流降级主要用于解决后端核心服务因压力过大造成系统反应过慢或者崩溃问题，通常用于例如商品秒杀、抢购、大促、防刷单等大流量场景。

- 限流

通过调节流量阈值来控制通过系统的最大流量值，保证系统安全可靠运行。

- 降级

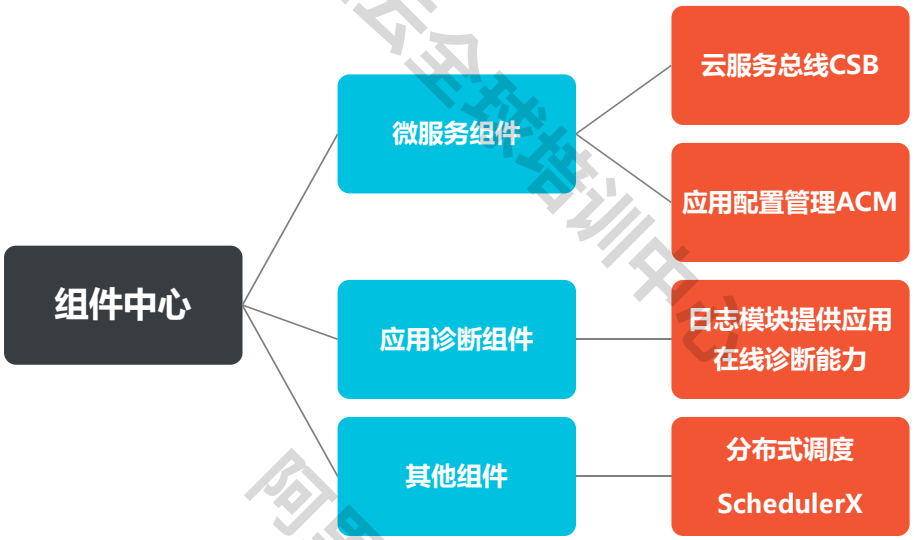
在 EDAS 中，降级通常用于对下游出现超时的非核心服务提供者进行低优先级调用，确保上游核心应用（服务消费者）不被影响。

日志管理

当应用出现异常情况的时候，可以通过查看实例和应用级别的日志来排查问题。

组件中心

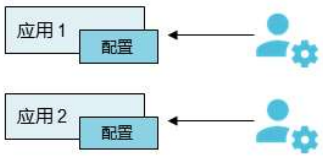
组件中心围绕分布式、微服务体系，重点建设服务集成和整合能力，进而实现 PaaS 平台开放的生态体系。



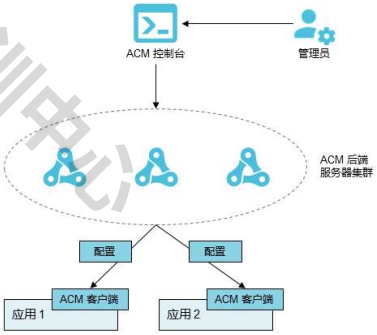
课程目录03

应用配置管理 ACM 的介绍

应用配置管理ACM（Application Configuration Management）是一款在分布式架构环境中对应用配置进行集中管理和推送的产品。凭借配置变更、配置推送、历史版本管理、灰度发布、配置变更审计等配置管理工具，ACM能集中管理所有应用环境中的配置，降低分布式系统中管理配置的成本，并降低因错误的配置变更造成可用性下降甚至发生故障的风险。



传统的手动修改配置



ACM的配置管理

应用配置管理 ACM 的发展历史

应用配置管理ACM前身为阿里中间件配置中心工具Diamond，是在阿里内部使用最广泛的中间件，于2017年10月正式成为阿里云产品。

2008

- 阿里中间件配置注册中心诞生，被运用于远程框架路由
- 百级应用，千级配置，万级推送

2010

- 配置中心和注册中心剥离，配置中心定位于保存持久化数据不丢失
- 中间件MetaQ，TDDL开始基于配置中心开发
- 千级应用，万级配置，十万级推送

2013

- 异步化改造，性能线性提高。
- 支持异地多活，双十一流控、预案、开关
- 千级应用，万级配置，百万级推送

2016

- 发布管控增强（支持灰度，审批，历史，回滚，推送轨迹，推送状态查询等）
- 万级应用，十万级配置，千万级推送
- ACM在EDAS中对外提供服务

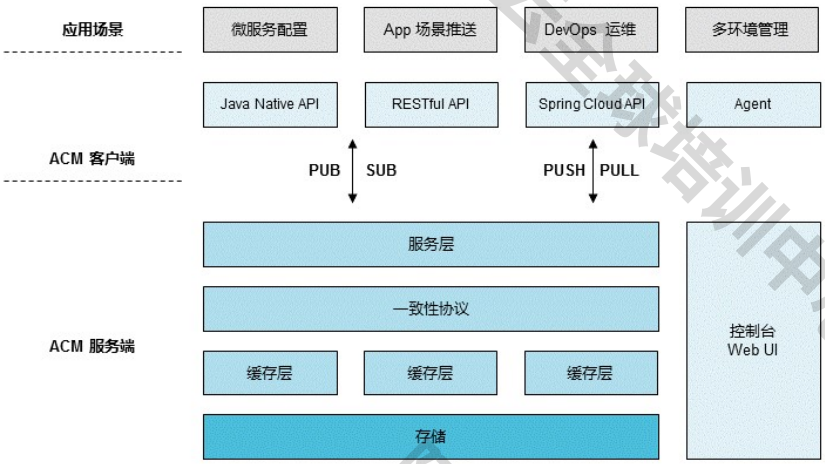
2017

- ACM上云支持多租户，鉴权，数据加密，容量管理
- 百级阿里云用户，万级应用，百万级配置，亿级推送

2018

- ACM商业化发布
- 开源版NACOS发布

应用配置管理 ACM 的技术架构



ACM 服务端，即 ACM 分布式服务节点，每一个节点包括以下部分：

- 服务协议层：用于进行协议转换，鉴权验证等。
- 一致性管理层：用于配置的一致性管理和配置推送。
- 配置缓存层：通过分布式缓存提高配置查询和推送效率。
  - 存储层：后台是一个分布式存储，用于存放配置，并具备高性能和高扩展性。
  - 控制台：ACM 控制台，用于配置管理。

ACM 客户端

- 客户端基于 RESTful API 设计，可实现跨语言访问。
- 提供 Java Native API 以及基于 Spring Cloud Config 的配置读取接口，方便您进行便捷开发。
- 在某些场景下，ACM 提供 Agent 来动态替换宿主机器上的配置文件。此时 ACM 配置项和配置文件对应关系需手动指定。

应用配置管理 ACM 的功能特性

核心功能

- 配置的“增删改查”
- 配置的导入导出
- 批量操作
- 配置描述
- 配置标签
- 在线编辑多种文件格式
- 多语言支持

高级功能

- 变更 Diff
- 灰度发布
- 变更历史
- 一键回滚
- 推送轨迹
- 命名空间

高可用性

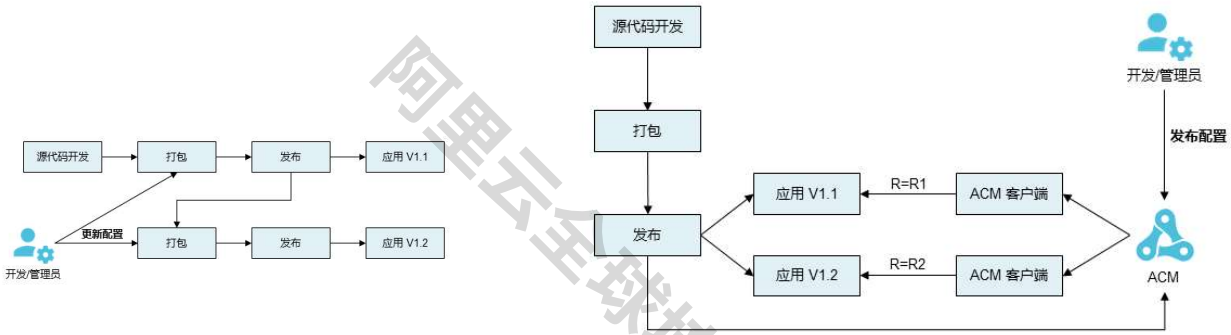
- 多级缓存
- 流量控制
- 容量管理
- 同城容灾

应用配置管理 ACM 与其他应用配置管理服务的对比

产品	ACM	Spring Cloud Config Server	ZooKeeper	ETCD
配置修改	直接在 ACM 控制台上进行配置修改	一般在 Git 仓库上进行配置修改	通过调用 ZK API 修改	通过调用 etcd API 修改
配置自动推送	修改过的配置自动推送到监听的客户端	客户端只能在启动的时候加载	修改过的配置自动推送到监听的客户端	修改过的配置自动推送到监听的客户端
API接口	基于 RESTful API，同时支持 Java Native 接口，Spring Cloud 接口，和其他语言类接口	基于 RESTful API 和 Spring Cloud 规范，同时也支持其他语言客户端	支持 Java 原生接口	基于 RESTful API 的接口
版本管理	在 ACM 上自动记录各个修改的版本信息	通过 Git 间接管理版本	不带任何版本控制	不带任何版本控制
配置推送追踪	可查询所有客户端配置推送状态和轨迹	无法查询配置推送历史	无法查询配置推送历史	无法查询配置推送历史

应用配置管理 ACM 的应用场景

微服务应用架构下的配置管理：



传统架构的应用发布过程

微服务应用架构的ACM应用发布过程

分布式任务调度 SchedulerX 的介绍

SchedulerX2.0是阿里巴巴基于Akka架构自研的新一代分布式任务调度平台

- 编排定时任务、工作流任务、分布式任务调度
- 在控制台配置、管理定时调度任务、查询任务执行记录和运行日志，还可以通过工作流进行任务编排和数据传递

系统组件

3类组件

- server组件
- console组件
- zookeeper

触发调度流程

4种方式

- 固定时间点
- 周期性
- 控制台手动
- 依赖触发

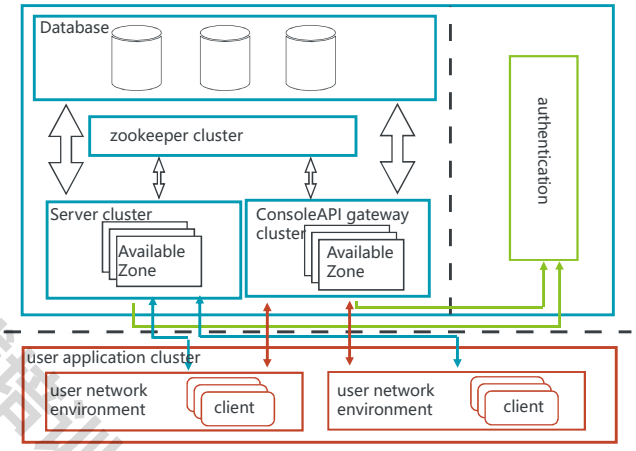
高可用保障

2个方面

- 服务端可用(自动备份)
- 客户端可用(分布式架构)

分布式任务调度 SchedulerX 的系统组件

01 console组件	1.Job元数据维护 2.API gateway 3.系统管控、监控报警
1.触发调度核心逻辑 2.Client连接管理 3.Job状态管理	**02 server组件**
03 zookeeper	1.分布式锁 2.Server、Console机器状态维护

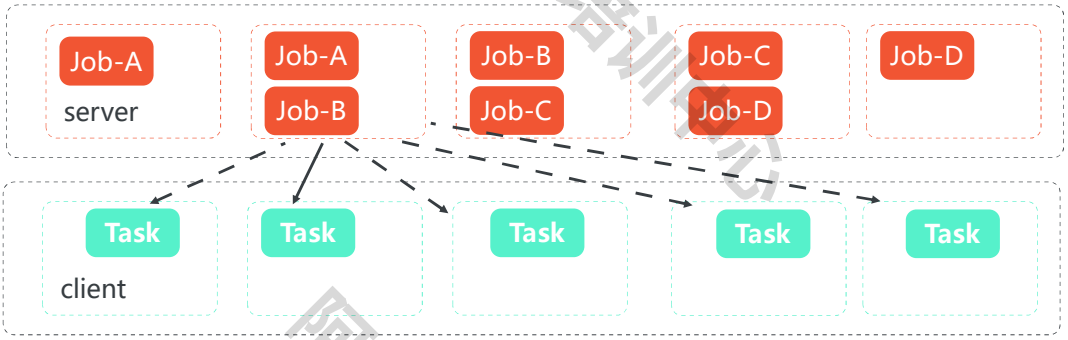


分布式任务调度 SchedulerX 的触发调度流程

01 固定时间点
例如：2016年11月11日0点执行的一次任务

02 周期性
例如：每秒钟执行一次的任务

03 控制台手动/API触发
例如：可以通过控制台手动触发任务的调度执行或者API



分布式任务调度 SchedulerX 的高可用保障

服务端可用

自动备份

每个Job在服务端奇数台不同机器上有一个备份，ScheduleX服务端任意宕掉仍然能保证Job正常调度

客户端可用

分布式架构设计

任务可以在多台客户端机器里的任何一台执行，如果客户端宕机，服务端会自动选择正常运行的客户端去执行Job

分布式任务调度 SchedulerX 的核心功能

SchedulerX主要提供调度、执行和运维三方面的功能。

定时调度	工作流调度	执行方式	运维能力
• Crontab • Fixed rate • Second delay • 日历 • 时区	• 使用有向无环图DAG（Directed Acyclic Graph）进行任务编排	• 单机 • 广播 • Map模型 • MapReduce模型 • 分片运行	• 数据大盘 • 查看日志 • 原地重跑 • 标记成功 • 停止调度任务

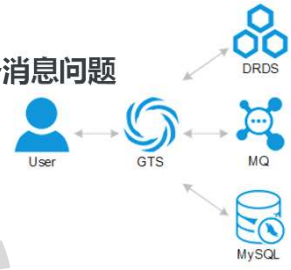
全局事务服务GTS

全局事务服务（Global Transaction Service，简称 GTS）是一款高性能、高可靠、接入简单的**分布式事务中间件**，用于解决分布式环境下的数据一致性问题。

分布式事务支持	支持多种数据源	配合多种微服务框架	多种事务支持
• 保证分布式事务 ACID 特性	• 数据源包括MySQL、RDS、DRDS等	• 配合EDAS、Dubbo等多种微服务框架使用	• 支持分布式数据库事务、多库事务、消息事务、服务链路级事务等

解决跨服务事务问题

GTS 配置 MQ 解决事务消息问题



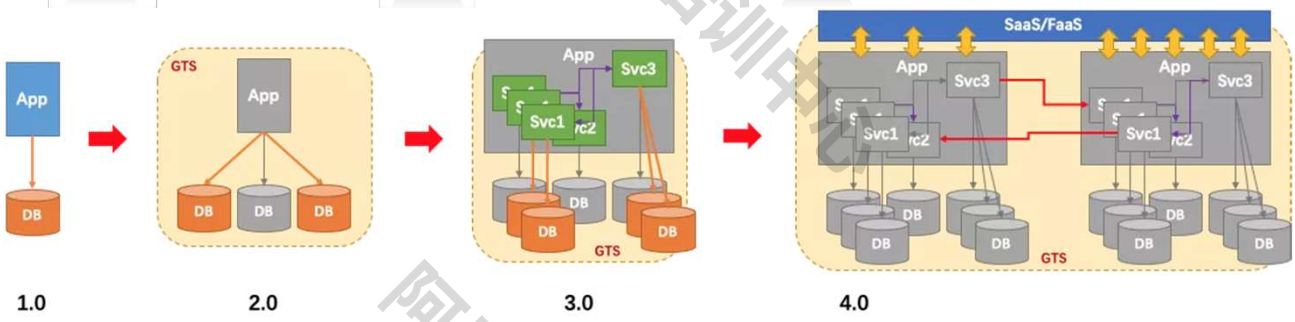
分布式事务面临的挑战

- 传统的事务主要是指单机数据库的ACID特性。
- **分布式事务**指事务的**发起者、资源及资源管理器、事务协调者**分别位于不同的分布式系统的不同节点之上。

• **跨库事务问题**

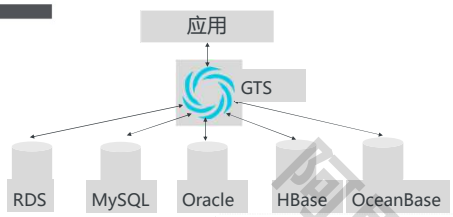
• **微服务化事务问题**

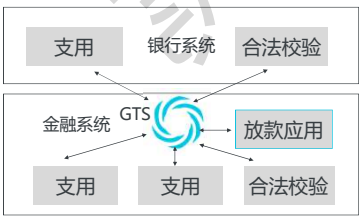
• **消息事务问题**



1.0 2.0 3.0 4.0

全局事务服务 GTS 应用场景



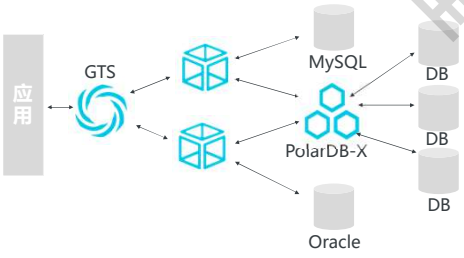


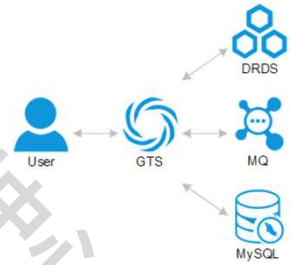
跨数据库事务

微服务调用事务

多系统集成事务

混合事务





分布式事务传统解决方案

资源层

XA 方案

- 最早解决跨库事务问题
- 优势：接口标准化，门槛较低，强一致性
- 阻塞协议，影响系统吞吐和可伸缩性
- 高并发下性能不理想，很难满足互联网大并发需求
- 缺乏容错机制
- 没办法解决微服务事务问题

补偿方案（TCC）

- 符合业务需求
- 对开发者要求高，实现复杂，各种异常情况难于处理
- 要求每个方案实现一个反向的回滚接口
- 运维成本高，扩展移植性不理想

消息方案

- 要求应用与消息系统紧耦合，增加并发成本
- 要求业务实现幂等
- 最终一致性、适用场景受限，不太适应一致性要求非常高的场景

应用层

全局事务服务 GTS 与传统分布式事务解决方案的对比

全局事务服务 GTS 的优势

完整解决方案

- 可以解决以下场景下的分布式事务问题：
  - 分布式数据库
  - 跨数据库
  - 服务化
  - 消息系统

侵入性极低

- 侵入性和使用门槛低
  - 一行注解即可实现事务接入
  - 提供API接入模式
- 节省开发和运维成本

性能超强

- 高达传统分布式事务10倍性能
- 4C8G 集群可达1.5万TPS
- 热点数据高效处理，无惧数据冲突

高可用

- 支持应用宕机、节点故障等各类异常情况下均可保持数据严格一致
- 支持同城主备及两地三中心部署

课程目录04

EDAS 使用流程

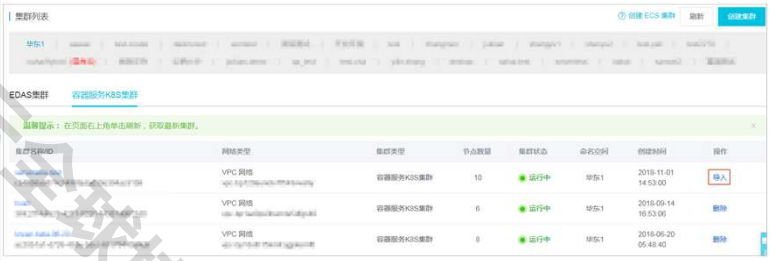
资源管理 → 应用部署 → 应用管理 → 应用监控 → 日志诊断

- EDAS 概览页

EDAS 使用流程

资源管理 → 应用部署 → 应用管理 → 应用监控 → 日志诊断

- 导入ECS集群

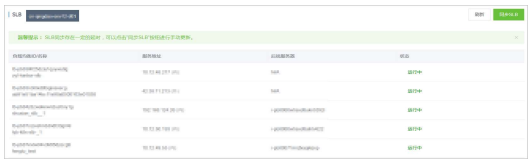


- 导入K8S集群

阿里云全球培训中心

EDAS 使用流程

资源管理 → 应用部署 → 应用管理 → 应用监控 → 日志诊断



- SLB 配置

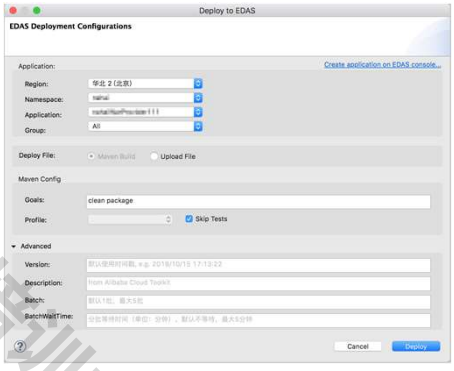
- VPC 配置

- EDAS Agent 安装

EDAS 使用流程

资源管理 → 应用部署 → 应用管理 → 应用监控 → 日志诊断

- 控制台部署

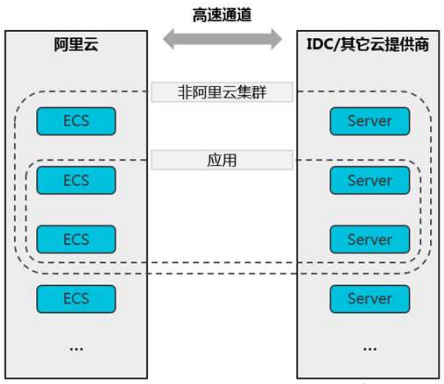


- 工具部署

阿里云全球培训中心

EDAS 使用流程

资源管理 → 应用部署 → 应用管理 → 应用监控 → 日志诊断

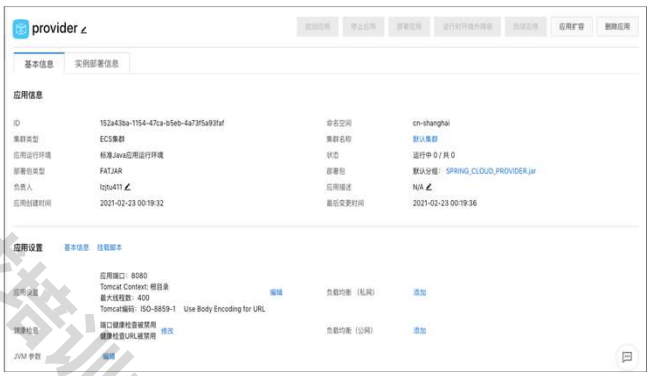


- 混合云部署

EDAS 使用流程

资源管理 → 应用部署 → 应用管理 → 应用监控 → 日志诊断

1、创建命名空间

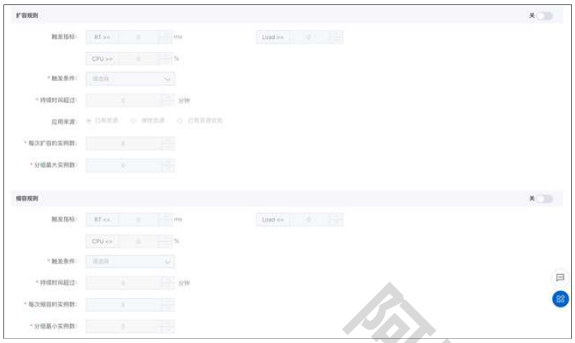


2、应用操作

EDAS 使用流程

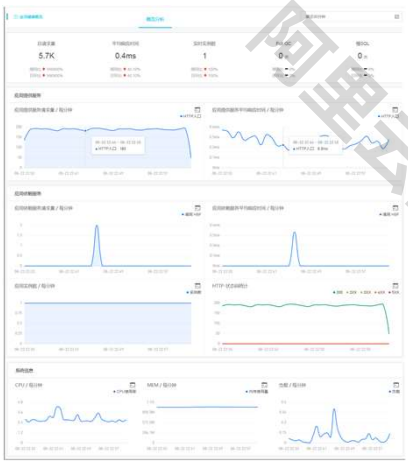
资源管理 → 应用部署 → 应用管理 → 应用监控 → 日志诊断

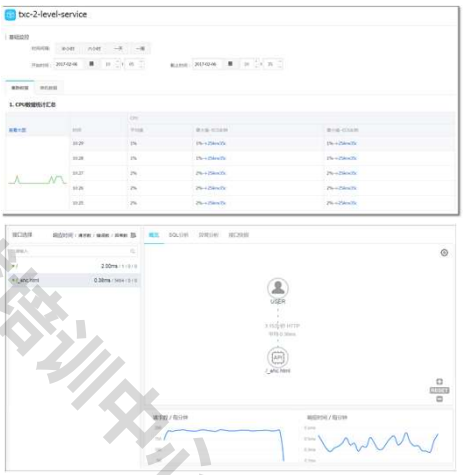
分组管理 — 健康检查 — 扩容/缩容 — 限流降级 — 配置管理 — 配置管理 — ……



EDAS 使用流程

资源管理 → 应用部署 → 应用管理 → 应用监控 → 日志诊断





阿里云全球培训中心

EDAS 使用流程

资源管理 → 应用部署 → 应用管理 → **应用监控** → 日志诊断

为EDAS应用接入ARMS高级监控后，可使用多项应用监控能力

自动发现应用拓扑

ARMS应用监控探针能够自动发现应用的上下游依赖关系，有效捕获、智能计算、自动展示不同应用之间通过RPC框架组成的调用链。

捕获异常事务和慢事务

您可以进一步获取接口的慢SQL、MQ堆积分析报表或者异常分类报表，对错、慢等常见问题进行更细致的分析。

自动发现并监控接口

ARMS应用监控能够自动发现和监控应用代码中常见的Web框架和RPC框架，并统计接口的调用量、响应时间、错误数等指标。

实时诊断

开启ARMS应用监控的实时诊断功能后，ARMS应用监控会持续监控应用5分钟，并在这5分钟内全量上报调用链数据。

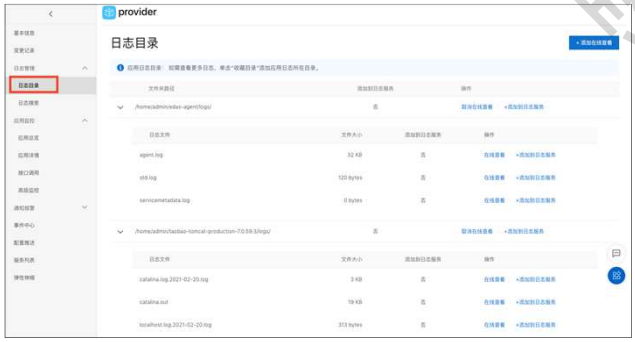
EDAS 使用流程

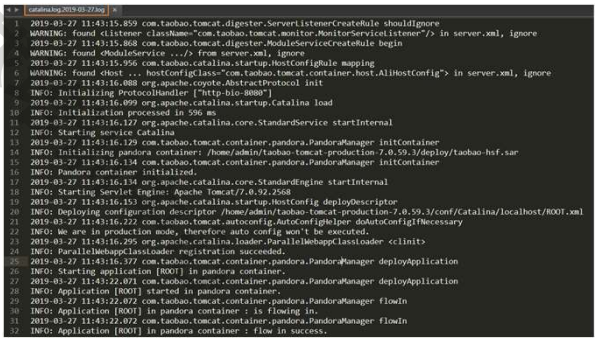
资源管理 → 应用部署 → 应用管理 → **应用监控** → 日志诊断

- 通知报警

EDAS 使用流程

资源管理 → 应用部署 → 应用管理 → 应用监控 → 日志诊断





- 通过控制台或 CLI 查看

EDAS 使用流程

资源管理 → 应用部署 → 应用管理 → 应用监控 → 日志诊断

- 记录日志: 查看日志定位问题

开发中遇到的很多问题都可以通过查看相关日记进行定位。

- 应用日志： catalina.out localhost.log.xxx
- HSF 相关日志： config.client.log hsf.log
- 轻量配置中心日志： config-center.log

日志文件名	日志文件含义
AliTomcat目录\logs\catalina.out IDE终端	最重要的日志，可以看到应用和 Tomcat 应用服务器的异常，这是开发最应该关注的日志。
Windows: C:\Users\你的用户名\configclient\logs/config.client.log Linux: /home/admin/configclient/logs/config.client.log	可以通过此日志查看服务发布订阅是否成功，"Register-ok"、"Publish-ok" 等关键字。
Windows: C:\Users\你的用户名\logs\hsf\hsf.log Linux: /home/admin/logs/hsf/hsf.log	HSF 服务的日志，有 HSF 服务调用过程的一些详细信息。如果 HSF 调用中出现异常，可以看看这个日志。

EDAS 使用流程

资源管理 → 应用部署 → 应用管理 → 应用监控 → 日志诊断

- 异常出现时，如何判断是自身代码错误，还是 EDAS 出现错误

查看错误堆栈的最后部分，判断具体问题。

Caused by: com.yourcompany.yourpack 业务代码问题。

Caused by: com.alibaba.xxxx EDAS框架问题。

Caused by: com.taobao.xxx EDAS框架问题。

Caused by: com.ali.xxx EDAS框架问题。

课程目录05

最佳实践一：Spring Cloud和Dubbo框架应用无缝迁移到EDAS

对于Spring Cloud Edgware及以上版本和Dubbo 2.5.3及以上版本的微服务应用，无需修改任何一行代码即可迁移至企业级分布式应用服务EDAS。

- 迁移应用
- 迁移SLB或者修改域名
- 迁移存储和消息队列

Spring Cloud和Dubbo框架应用无缝迁移到EDAS的优势

迁移至EDAS前

DUBBO

1. 需独立搭建和维护ZK
2. 缺失链路跟踪能力
3. 无限流降级保护

代码零侵入、只需引入依赖

迁移至EDAS后

- ✓ 服务发现换成EDAS，**省硬件和维护成本。**
- ✓ **0成本**接入全链路跟踪，还支持慢SQL查询。
- ✓ **限流降级**规则动态可配，可视化监控。

spring cloud

1. 需独立搭建eureka、Consul等组件
2. 全链路跟踪需要搭建一系例组件
3. 配置管理方案成本较高
3. 无限流降级保护

代码零侵入、只需引入依赖配置

- ✓ 服务发现引入ANS starter，直接实现，**且自动安全加固。**
- ✓ **0成本**接入全链路跟踪，还支持慢SQL查询。
- ✓ 配置管理引入ACM starter，**实时推送配置，且推动状态和轨迹实时可查。**
- ✓ **限流降级**规则动态可配，可视化监控。
- ✓ **省硬件和维护成本。**

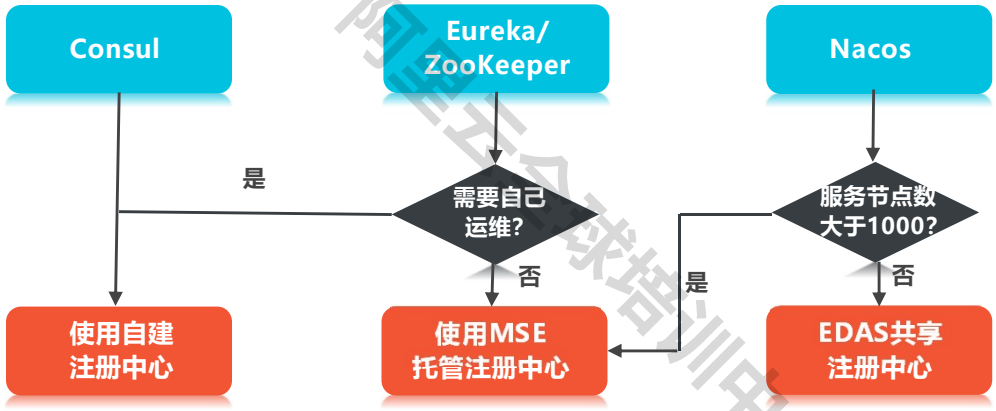
最佳实践二：基于EDAS构建开发环境

EDAS为您提供三种可选方案，可以根据实际需求决定在本地或者云上构建开发环境，以便开发和调试应用。

构建环境	方案	说明
本地	在本地搭建轻量配置中心实现服务注册和发现，在本地开发、调试。	轻量配置中心不具有生产环境的性能水平，当注册上来的服务较多的时候可能会有性能问题。因为是本地环境，也无法使用EDAS中的服务治理、监控和发布等功能。完全为您的自建环境。
阿里云	在云上创建开发环境，开发人员通过端云联调插件连接云端应用，进行开发、调试。	可以使用EDAS的全部能力。因为使用云上资源，成本比较高。
混合云	在混合云中创建开发环境，开发人员可以直接在本地进行开发、调试。	可以使用EDAS的全部能力。需要通过VPN或专线连通本地网络和阿里云VPC。注意：需要开通EDAS专业版或铂金版。

最佳实践三：注册中心的选型

微服务应用通过注册中心实现服务注册与发现。在开发应用时，可以根据实际需求选择注册中心：



微服务引擎MSE的介绍

微服务引擎（Microservice Engine， 简称 MSE ）是**微服务注册中心**和**配置管理**的全托管式平台，提供高可用且免运维的服务注册和配置管理集群。

- 全托管
- 100%兼容开源
- 开箱即用

客户场景

- 围绕服务治理、配置管理、分布式协调等领域
- 客户直接使用场景，如Dubbo、Spring Cloud分布式协调、配置管理等
- 大数据场景，如Hbase、Spark、Hadoop、Blink等
- EDAS微服务解决方案
- 容器服务

微服务引擎

- ZooKeeper
- Eureka
- Nacos
- Consul

高可用SLA指标保障

- 连接数、数据量、TPS
- 集群管理
  - 集群创建
  - 集群释放
  - 集群迁移及复制
- 数据管理
  - 数据添删改查
  - 数据备份、恢复
- 监控运维
  - 集群监控
  - 诊断工具

微服务引擎MSE的使用场景

微服务注册发现

- 在快速发展的云计算时代，微服务应用越来越广泛，采用Dubbo和Spring Cloud框架开发的微服务，需要稳定的服务注册和发现。MSE提供了高可用、免运维和稳定的服务注册中心。

分布式协调

- 如果企业正在使用HBase、Spark或Kafka等开源软件，那么可以使用微服务引擎提供的ZooKeeper、Eureka、Nacos，实现分布式系统的协调，助力企业增效降本。

总结与回顾

本章节主要讲述内容：

1.企业级分布式应用服务 EDAS 是一个**应用托管和微服务管理的 PaaS 平台**，提供应用开发、部署、监控、运维等全栈式解决方案，同时支持 Dubbo、Spring Cloud 等微服务运行环境；其定位是**构建企业级在线应用运行环境。**

2.企业级分布式应用服务EDAS的四大功能：

- 服务化：基于高性能分布式服务框架，IT系统沉淀共享资产，新需求基于共享服务层快速集成
- 应用生命周期：以应用为中心的中间件PaaS平台，提供一键应用部署与扩容，简化用户操作
- 立体化监控：从IAAS、应用和业务等多个层面实现对系统的全面立体的监控报警
- 运维管控：内置多种运维与管控工具，帮助用户实现服务治理、应用诊断

3.与企业级分布式应用服务EDAS结合使用的其他阿里云服务：应用配置管理ACM、分布式任务调度SchedulerX 全局事务服务GTS 和 微服务引擎MSE。

4.企业级分布式应用服务EDAS的使用与最佳实践

实验一：在EDAS的ECS集群中部署和管理Spring Cloud应用

实验概述

本实验将以Java开发的Demo应用程序，采用WAR包部署方式，展示如何将应用部署到企业分布式应用服务EDAS中，并快速实现服务治理。用户可以体验：

1. EDAS控制台的基本操作；
2. 在EDAS中创建和管理应用；
3. 在EDAS中实现应用的弹性伸缩管理；
4. 在EDAS中的应用监控和日志管理

实验目标

完成此实验后，可以掌握使用EDAS快速创建和部署应用，并对应用进行各种管理操作的能力；

课程目标

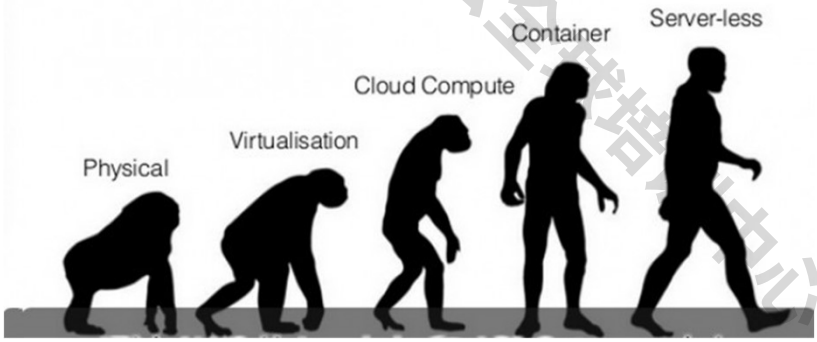
学习完本课程后，你将能够：

1. 了解Serverless发展的历程和理念
2. 掌握阿里云Serverless的产品体系和不同的分类
3. 深入理解Serverless应用引擎SAE的技术原理和功能特性

课程目录01

IT计算的发展历程

IT 史上演绎的解放生产力、发展生产力



Serverless 优点

- 降低运营成本
- 降低开发成本
- 自动扩展能力
- 更简单的管理
- “绿色”的计算

开发者

- 无需关注硬件资源，无需考虑多线程，无需考虑扩容，无需考虑流量瞬间峰值的未雨绸缪
- 极大解放生产力，专注做自己擅长的事情
- 安全免运维，真正细粒度的计算资源分配，按需服务

Serverless服务理念：让用户只专注于业务

除了实现业务逻辑本身，还需要解决：

1. 基础设施运维（ECS、集群）
2. 部署环境多样化
3. 应用运行时不一致
4. 基础服务能力集成（日志、监控、SLB）
5. 服务交互
6. 分布式缓存、队列的申请和回收等
7. 容量预估、弹性伸缩

.....

研发门槛？？ 研发效率？？企业创新效率?

应用	专注于业务需求实现
云	向上提供各种资源和通用能力

降低用户「运维服务」、「运行服务」、「开发服务」 的成本

阿里云Serverless服务体系

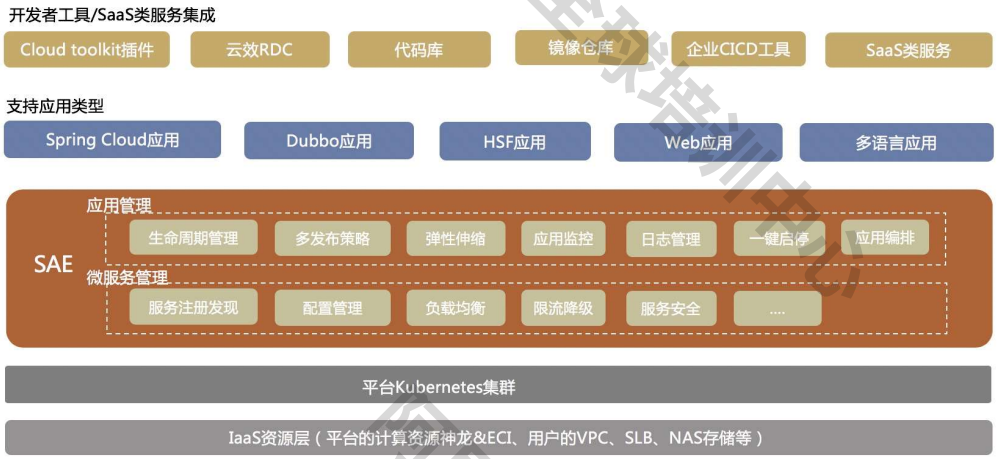
阿里云已帮助10万开发者专注业务开发，落地场景最丰富，是阿里经济体和企业客户 Serverless 最佳选型。

阿里云Serverless服务的分类

产品矩阵				编程模式	产品特点
函数计算（FC）				函数	面向函数，基于事件驱动与云产品间集成提供端到端的解决方案，用户只需编写代码上传后，为代码实际产生的请求资源消耗付费
函数调度器 + 资源管理	Serverless 应用引擎（SAE）			应用	面向应用，提供面向微服务的UI和API，抽象了应用的概念并对用户屏蔽底层，Kubernetes的技术细节，降低用户的使用门槛
	Serverless Infra 极致弹性组件	Serverless Kubernetes Serverless Addon		Kubernetes 容器编排	面向容器，底层使用 ECI，标准的 Kubernetes UI 和 API 用户界面，主要提供原生 Kubernetes 的生态；
ECS + 安全容器	ECI + 安全容器	ECI	ECI	容器实例	面向 IaaS 资源层，提供容器/Pod级别的运行环境，用户快速运行容器，只为容器实际消

Serverless应用引擎SAE：面向应用的Serverless PaaS平台

实现了**Serverless架构+微服务架构**的完美结合，支持多种微服务框架、多种部署渠道（UI、云效、插件等）、多种部署方式（war、jar、镜像）；核心场景主要面向**在线应用**：微服务应用、web应用、多语言应用等。



SAE 的产品优势

提供IaaS+PaaS集成的解决方案

基于Serverless架构，屏蔽底层IaaS运维和K8s细节，低门槛部署应用。

支持多种微服务框架，多种部署方式

完美支持各种主流微服务框架、支持镜像/war/jar三种部署方式。

精益成本，不为闲置浪费

按需使用，按分钟计费，无需长期保有固定IaaS资源。

高安全

底层基于安全容器运行应用，网络通过VPC隔离，双重保障应用运行时安全。

自动弹性

支持应用突发场景下秒级弹性伸缩，保障业务SLA。

服务集成，一站式体验

自动集成阿里云上的基础设施类产品：SLB、SLS、NAS等和微服务生态周边产品：ACM、ARMS、AHAS，提供一站式的解决方案。

SAE 的应用场景

中大型企业快速构建云上服务应用

屏蔽底层 IaaS 购买和运维细节，屏蔽底层镜像仓库和 Kubernetes 细节，低门槛通过 WAR/JAR 方式部署微服务应用，大幅提升运维效率，让企业聚焦核心业务本身。

应用环境灵活启停，降低资源成本

企业应用通常都有多套云上环境，除生产环境外，其它环境使用率低，但重新搭建一套环境的成本高。SAE提供了一键启停开发、测试环境的能力，即开即用，节省成本，方便运维

突发性流量洪流，从容应对

提供了弹性伸缩的能力，从容应对突发性洪流，帮助应用轻松应对流量高峰，保证 SLA 的同时也节省了资源成本。

课程目录02

SAE 的功能总览

应用开发

- 支持原生 Spring Cloud、Dubbo、HSF 微服务框架的应用开发

应用部署

- 支持控制台部署和开发工具部署

应用发布

- 应用发布的三板斧：可灰度、可观测和可回顾

应用管理

- 对应用本身及其相关的资源进行管理
- 监控管理和日志管理

应用开发

SAE 支持原生 Spring Cloud、Dubbo、HSF 微服务框架，在该框架下开发的应用只需添加依赖和修改配置，即可获取 SAE 企业级的应用托管、应用治理、监控报警和应用诊断等能力，实现代码零代码量应用迁移。

Spring Cloud开发

Dubbo开发

HSF开发

应用部署

控制台部署

开发工具部署

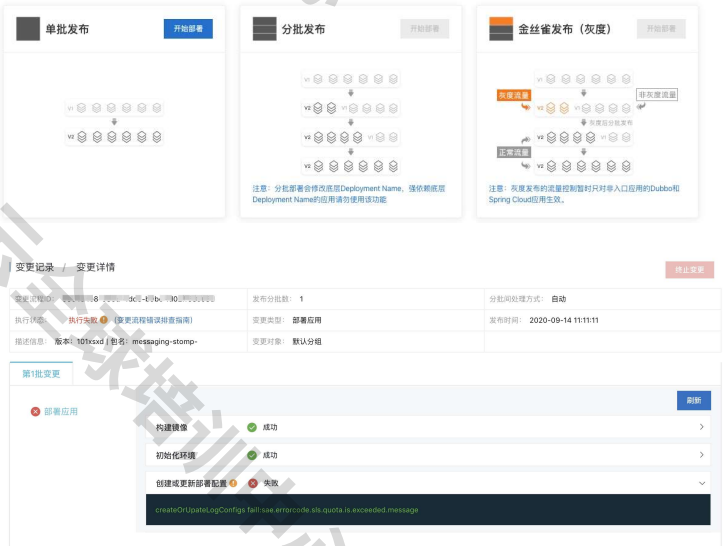
应用举例	部署方式	运行环境
原生 Spring cloud	WAR、JAR、镜像	WAR包部署： apache-tomcat 相关版本的运行环境 JAR 包部署：标准 Java 应用运行环境
原生 Dubbo	WAR、JAR、镜像	
HSF	WAR、JAR、镜像	**EDAS-Container** 相关版本的运行环境
多语言应用	镜像	

灰度发布　应用升级　分批发布

应用发布

应用发布**三板斧**（**可灰度、可观测、可回滚**）

- 支持单批、分批、金丝雀等发布策略；支持按流量灰度；批次间自动/手动任选
- 发布过程可监控：白屏化实时查看发布日志和结果，及时定位
- 允许人工介入控制发布流程：异常中止、一键回滚



应用管理

应用托管到 SAE 以后，通过 SAE 控制台对应用本身及其相关的资源进行管理。

1. **绑定SLB**
   通过添加公网 SLB 实现公网访问应用；添加内网 SLB 实现同 VPC 内所有节点通过私网负载均衡访问应用。
2. **应用生命周期管理**
   通过 SAE 对已部署的应用进行启停、更新、删除和扩缩容等管理操作。
3. **查看变更记录**
   记录 SAE 应用在生命周期内所有的操作记录，便于管理应用及定位问题。
4. **变更实例规格**
   SAE 支持按需修改应用实例规格，确保应用正常运行。
5. **配置弹性伸缩**
   通过自动弹性伸缩实现应用实例数高效利用，降低应用资源成本。
6. **一键启停应用**
   将处于同一命名空间的应用一键批量启停。
7. **配置管理**
   SAE 集成了应用配置管理（ACM），对应用配置进行集中管理

监控管理

基础监控 CPU、内存、负载、网络 4 个主要指标视图

应用监控 自动发现及监控应用代码中常见的 Web 框架和 RPC 框架，进行JVM监控、主机监控、SQL分析等

日志管理

实时日志

通过监控大盘查看该应用的实例分组下各个 Pod 的实时日志。当应用出现异常情况的时候，可以通过查看 Pod 的实时日志来排查问题。

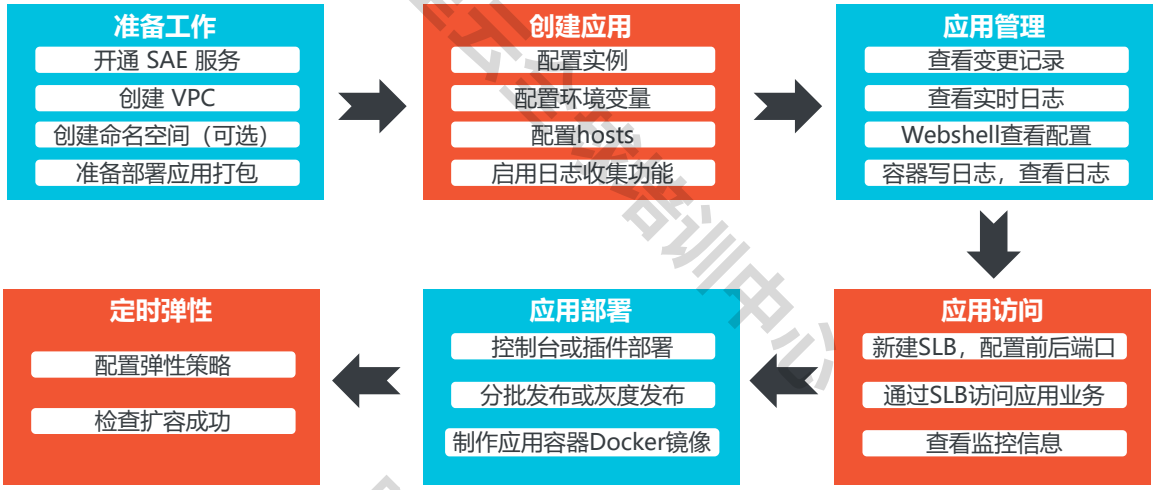
文件日志

如果启用了文件日志收集功能，SAE 将业务文件日志（不包含stdout 和 stderr 日志）收集并输入 SLS 中，您可以无限制行数的查看所收集日志，并进行自聚合分析。

日志源	消费端	创建时间	操作
/root/logs/spas/spas_sdk.log	SLS日志服务	2019-08-21 17:53:18	查看文件日志

课程目录03

SAE 的使用流程



应用部署

部署方式

- SAE控制台部署
- 通过toolkit-maven-plugin部署
- 通过Alibaba Cloud Toolkit for IntelliJ IDEA部署
- 通过Alibaba Cloud Toolkit for Eclipse部署

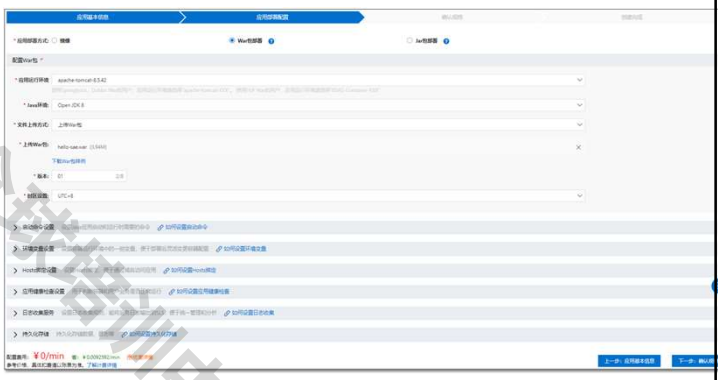
可部署的应用类型

应用举例	部署方式
原生Spring Cloud	WAR、JAR、镜像
原生Dubbo	WAR、JAR、镜像
HSF	WAR、JAR、镜像
多语言应用	镜像

通过控制台部署应用

①

②



③

通过toolkit-maven-plugin部署应用

1. 添加插件依赖，在pom.xml中增加如下依赖

```
<build>
    <plugins>
        <plugin>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>toolkit-maven-plugin</artifactId>
            <version>1.1.2</version>
        </plugin>
    </plugins>
</build>
```

2. 配置插件

2.1 账号配置，在打包工程的根目录下创建文件格式为 YAML 的账号配置文件，命名为toolkit_profile.yaml并填入如下信息：

```
regionId: #应用所在区域，如北京为`cn-beijing`，上海为`cn-shanghai`，杭州为`cn-hangzhou`。
accessKeyId: #访问阿里云资源的AK，建议使用子账号的AK降低安全风险
accessKeySecret: #访问阿里云资源的SK，建议使用子账号的SK降低安全风险
```

通过toolkit-maven-plugin部署应用

2.2. 打包配置，在打包工程的根目录下创建文件格式为 YAML 的打包配置文件。如果打包工程为Maven的子模块，则需要在子模块的目录下创建该文件，并命名为toolkit_package.yaml，填入如下信息：

```
apiVersion: V1
kind: AppPackage
spec:
    packageType: #应用部署包类型，支持WAR、FatJar、Image三种格式。
    packageUrl: #如果应用部署包类型为WAR或FatJar，可填入此字段，不填则使用当前maven构建的包进行部署。
   imageUrl: #如果部署包类型为Image，可填入此字段；Image类型也可以在本地构建Docker镜像进行部署，请参考打包文件参数章节设置相关参数。
```

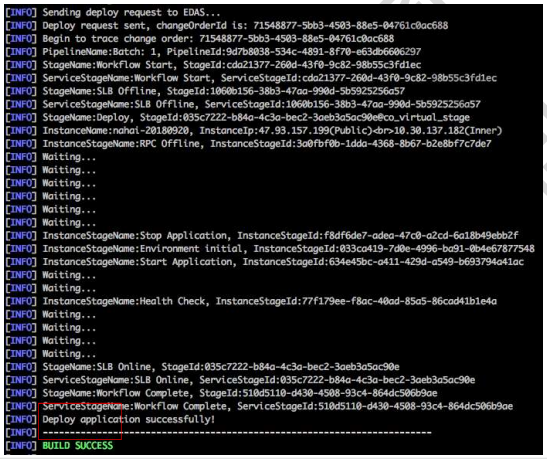
2.3 部署配置，在打包工程的根目录下创建文件格式为 YAML 的部署文件，命名为toolkit_deploy.yaml，并填入如下信息：

```
apiVersion: V1
kind: AppDeployment
spec:
  type: serverless
  target:
      appId: #部署应用的ID，如果配置了appId则无需配置namespaceId和appName
      namespaceId: #所属区域，如不清楚appId，可使用此所属区域及应用名称进行部署
      appName: #应用名称，如不清楚appId，可使用此应用名称及命名空间进行部署
```


通过toolkit-maven-plugin部署应用

3 构建应用，进入pom.xml所在的目录（如果部署Maven子模块，则进入子模块pom.xml所在的目录），执行如下命令：

```
mvn clean package toolkit:deploy -Dtoolkit_profile=toolkit_profile.yaml -Dtoolkit_package=toolkit_package.yaml -Dtoolkit_deploy=toolkit_deploy.yaml
```



当执行结果出现 BUILD SUCCESS，则部署成功

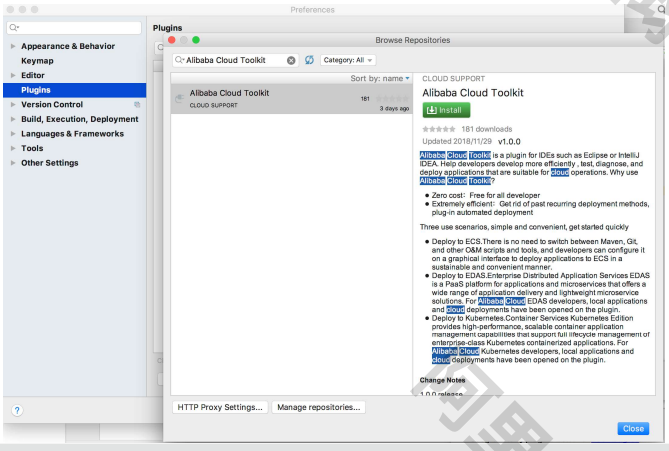
通过Alibaba Cloud Toolkit for IntelliJ IDEA部署应用

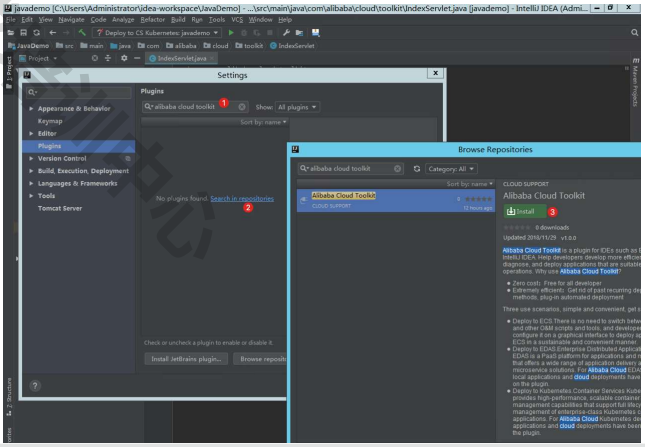
1 安装前提： JDK1.8、IntelliJ IDEA 2018.3或更高版本

2 安装Cloud Toolkit

Mac系统：进入Preference配置页面，进入Plugins选项，搜索Alibaba Cloud Toolkit，单击Install。

Windows 系统：进入**Plugins**选项，搜索**Alibaba Cloud Toolkit**，并单击**Install**。



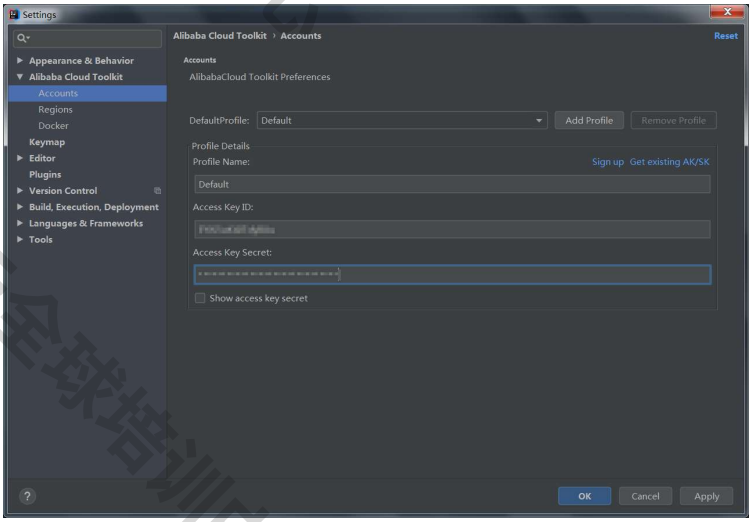


通过Alibaba Cloud Toolkit for IntelliJ IDEA部署应用

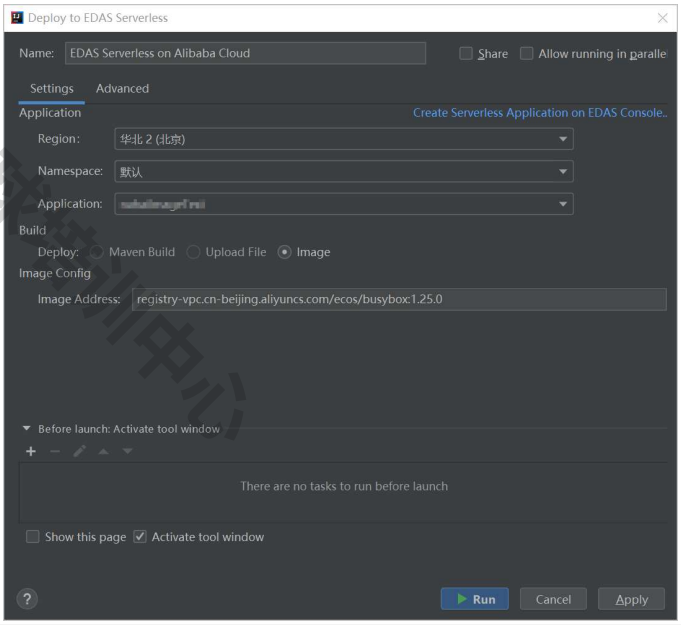
3 重启IntelliJ IDEA。插件安装成功后，重启IntelliJ IDEA，在工具栏显示Cloud Toolkit图标

4 配置Cloud Toolkit账号

单击Cloud Toolkit图标，在下拉列表中单击**Preference...**，在设置页面左侧导航栏选择**Alibaba Cloud Toolkit > Accounts** 。在**Accounts**界面中设置**Access Key ID**和**Access Key Secret**，并单击**OK**。



通过Alibaba Cloud Toolkit for IntelliJ IDEA部署应用



5 将应用部署到SAE。

- 在IntelliJ IDEA上单击Cloud Toolkit 图标，并在下拉列表中选择 **Deploy to EDAS Serverless**。
- 在**Deploy to EDAS Serverless**运行配置页面，配置应用部署参数。配置完成后单击**Apply**保存设置。
- 在Settings页面中，根据需求选择应用的Region、Namespace和Application 。Region：应用所在地域。Namespace：应用所在命名空间。Application：应用名称。
- 设置构建方式。Maven Build：Maven Build方式构建应用，默认添加Maven任务构建部署包。
- Upload File：Upload File方式构建应用，上传待部署的WAR包或者JAR包并进行部署。
- Image：选择镜像方式构建应用，需要注入镜像地址并进行部署。

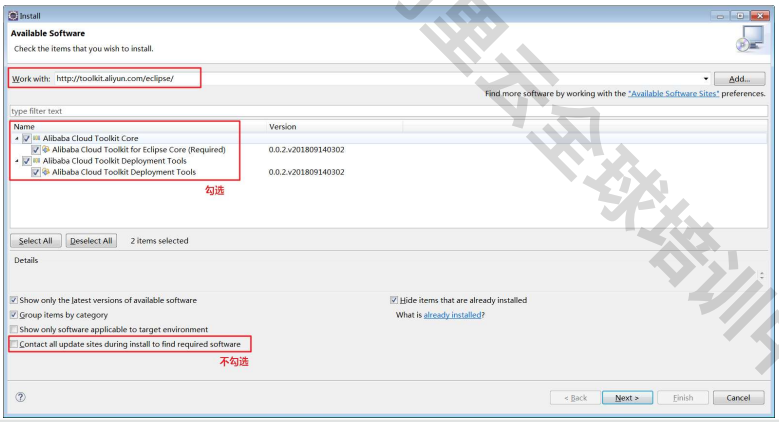
6 单击Run。IntelliJ IDEA的Console区域打印部署应用运行日志。您可以根据日志信息检查部署结果。

通过Alibaba Cloud Toolkit for Eclipse部署应用

1 安装前提： JDK1.8、Eclipse IDE、4.5.0（代号：Mars）或更高版本

2 安装Cloud Toolkit

2.1 启动Eclipse，在菜单栏中选择Help > Install New Software。在Available Software对话框的Work with文本框中，输入Cloud Toolkit for Eclipse的URL。组件配置如下：

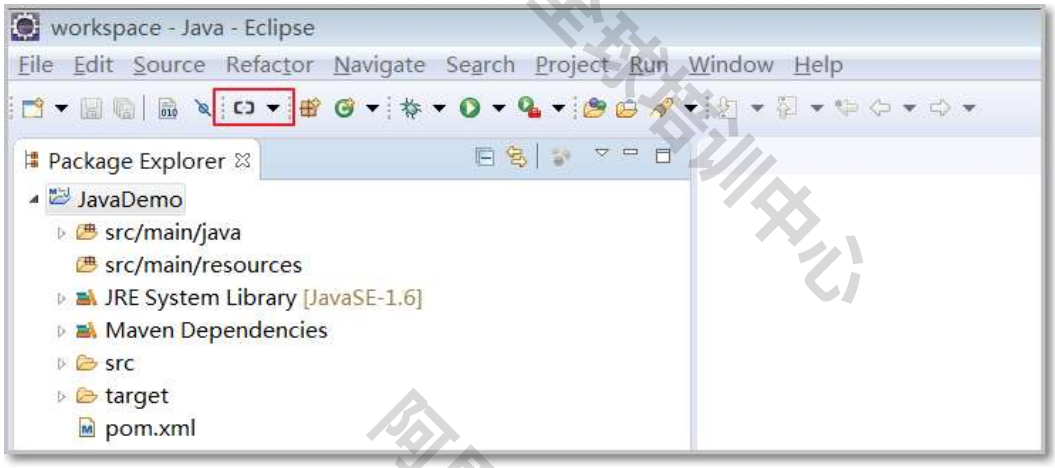


- 在type filter text列表区域中，勾选需要的组件Alibaba Cloud Toolkit Core和Alibaba Cloud Toolkit Deployment Tools。
- 在下方Details区域中，清除勾选Connect all update sites during install to find required software。
- 单击Next。
- 按照Eclipse安装页面的提示，完成后续安装步骤。

通过Alibaba Cloud Toolkit for Eclipse部署应用

2 安装Cloud Toolkit

2.2重启Eclipse。Cloud Toolkit插件安装完成后，重启Eclipse。重启后在工具栏显示Alibaba Cloud Toolkit 图标。



通过Alibaba Cloud Toolkit for Eclipse部署应用

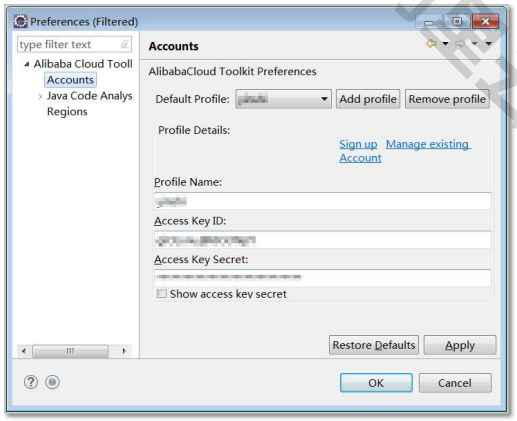
3 通过Cloud Toolkit部署应用到SAE

3.1 配置Cloud Toolkit账号

安装Cloud Toolkit完成后，需要使用Access Key ID和Access Key Secret配置Cloud Toolkit账号。启动Eclipse。

在工具栏Cloud Toolkit图标右侧下拉菜单中，选择 **Alibaba Cloud Preference... > Alibaba Cloud Toolkit > Accounts**。

在**Accounts**界面中设置**Access Key ID**和**Access Key Secret**，并单击**OK**。



- 已有阿里云账号：在Accounts界面中单击Get existing AK/SK，进入并登录阿里云登录页面，系统自动跳转至安全信息管理页面，获取Access Key ID和Access Key Secret。
- 没有阿里云账号：在Accounts界面中单击Sign up，进入阿里云账号注册页面并完成注册。注册完成后并获取Access Key ID和Access Key Secret。

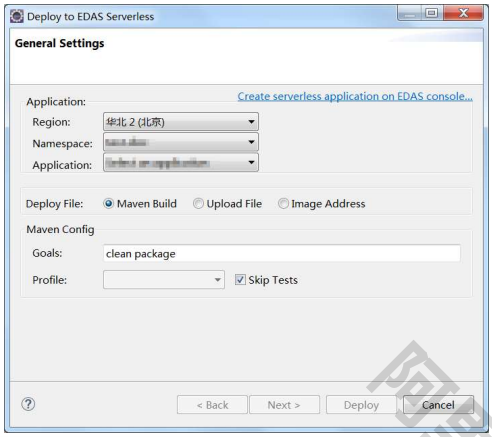
通过Alibaba Cloud Toolkit for Eclipse部署应用

3 通过Cloud Toolkit部署应用到SAE

3.2 Cloud Toolkit插件目前仅支持将应用以WAR包、JAR包或镜像方式部署到SAE。

在Eclipse界面左侧的Package Explorer中选择创建的应用工程名，并在弹出的下拉菜单中选择Alibaba Cloud > Deploy to EDAS Serverless...。

在Deploy to EDAS Serverless对话框中，依据需求选择应用的Region、Namespace、Application，并设置部署方式，配置完成后单击Deploy。



- Region：应用所在地域。
- Namespace：应用所在命名空间。
- Application：应用名称。
- 部署过程中，Eclipse的Console区域回显应用的部署日志，依据日志信息检查部署结果。

查看应用事件

SAE支持查看K8s原生的应用事件，帮助运维人员快速了解应用运行时的状态，方便快速聚焦问题。

1. 登录SAE控制台。
2. 在左侧导航树单击**应用列表**，**应用列表**页面单击具体应用名称。
3. 在**应用详情**页左侧导航栏中单击**应用事件**。

4. 设置事件搜索过虑条件，并单击搜索。搜索条件如下：

来源类型：包括**应用实例（Pod）**、**SLB（Service）**、**应用（Deployment）**。

来源名：输入事件的来源名，例如应用名称、应用实例名称。

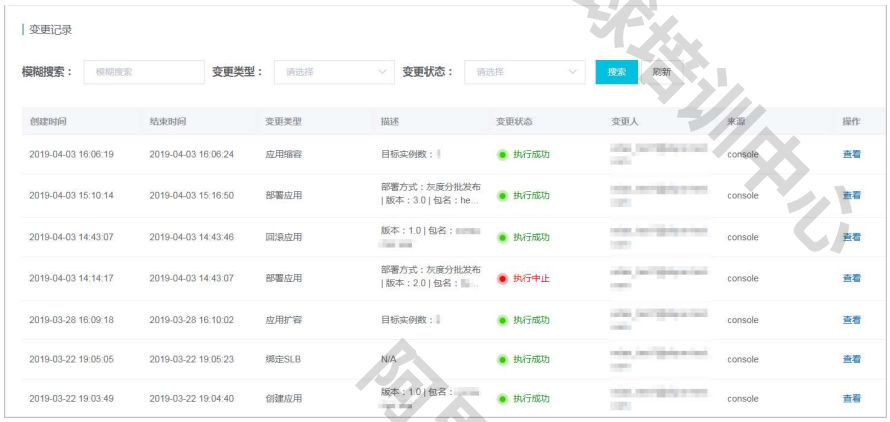
事件原因：输入事件原因，例如Pod的FailedScheduling。

事件等级：包含Waring和Normal，Waring级别事件需要您关注。

查看变更记录

在SAE上进行应用部署、启动、扩容/缩容等生命周期操作后，可以在应用详情页查看当前变更状态，也可以通过变更记录页面查看该应用的历史变更记录。

1. 登录SAE控制台。
2. 在左侧导航树单击应用列表，应用列表页面单击具体应用名称。
3. 在应用详情页面左侧导航栏中单击变更记录，查看该应用的所有变更操作以及该操作的变更状态



查看变更记录

在SAE上进行应用部署、启动、扩容/缩容等生命周期操作后，可以在应用详情页查看当前变更状态，也可以通过变更记录页面查看该应用的历史变更记录。

4. 在操作列单击**查看**，查阅变更详情及操作的详细信息。

- 该页面包含变更概要信息和变更流程执行信息。
- 变更概要信息：包括变更流程 ID、执行状态、变更类型等信息。
- 变更流程执行信息：包含整个变更流程，以及变更阶段中具体的变更任务。

5. 在变更流程执行信息区域的右侧展开具体任务，可以查看该变更过程中本环节的操作日志。

配置弹性伸缩

弹性伸缩是很重要的运维能力。弹性伸缩能够感知应用内各个实例的状态，并根据状态动态实现应用扩容、缩容。SAE支持定时弹性或者监控指标（CPU和内存）弹性功能，实现应用实例数自动增加和减少，在保证服务质量的同时，高效的利用应用资源、降低应用资源成本。

- 定时弹性伸缩：适用于资源画像存在周期性的应用场景，多用于证券 、医疗政府和教育等行业 。
- 监控指标弹性伸缩：适用于突发流量和典型脉冲的应用场景，多用于互联网、游戏和社交平台等行业。

配置弹性伸缩 —— 定时弹性伸缩

1. 登录SAE控制台。
2. 在左侧导航树单击应用列表，在应用列表页面单击具体应用名称。
3. 在应用详情页面选择实例部署信息 > 弹性伸缩，展开弹性伸缩折叠页签。
4. 配置定时弹性伸缩策略。

- 选择时间：根据需求选择短期或者长期执行所设弹性伸缩策略。
- 周期：执行弹性伸缩策略的时间周期，每天、每周、每月。
- 触发时间：设置弹性伸缩策略触发时间，以及该时间段内需要保持的应用实例数。

配置弹性伸缩 —— 监控指标弹性伸缩

监控指标弹性伸缩与定时弹性伸缩配置流程基本一样，仅配置规则不同。

- **触发条件**：支持**CPU使用率**和**Mem使用率**。
- **最大应用实例数**：触发弹性伸缩条件后，应用扩容，其实例数可达到的目标值。
- **最小应用实例数**：触发弹性伸缩条件后，应用缩容，其实例数可达到的目标值。
- 单选CPU使用率和Mem使用率时，当前应用的CPU使用率或者Mem使用率大于或者等于所设的目标值，则对应用进行扩容，其应用实例数不超所设的最大应用实例数；反之，进行缩容，其应用实例数不低于所设的最小应用实例数。
- 全选CPU使用率和Mem使用率时，如果任意一个指标的使用率大于或者等于所设目标，则应用进行扩容，其应用实例数不超所设的最大应用实例数；反之，进行缩容，其应用实例数不低于所设的最小应用实例数。

配置网关路由

应用托管SAE后，如果业务请求需要分发给其他服务或者应用，可以为应用配置网关路由，实现请求路由分发。

1 配置前提：

- 使用SLB
- 应用托管至SAE
- 待配置网关路由的应用以及后端接收请求的应用需要处在同一命名空间内。

2 网关路由功能适用场景：

- 单应用或多应用使用相同域名但存在不同路径流量转发。
- 单应用或多应用有不同域名流量转发，即不同域名解析的访问ip是同一个。

网关路由配置完成后，可以通过“ http://域名:访问端口/Path”的方式访问相应的后端服务或者应用。

配置网关路由

1. 登录SAE控制台。
2. 在左侧导航树单击应用列表，在应用列表页面单击具体应用名称。
3. 在应用详情页面的左侧导航树单击网关路由，并在网关路由页面单击新建。
4. 配置入口网关

   4.1在新建路由规则页面设置入口网关的基本信息。

- 名称：填写路由规则的名称。
- 网关类型：选择待转发请求的网络类型，包含公网和私网。通过私网网关转发仅在当前VPC内部转发。
- 请选择SLB：选择SLB控制台创建的SLB实例。
- 协议类型：选择请求转发协议，目前支持HTTPS和HTTP两种协议。
- SSL证书：如果您使用HTTPS协议，需要选择其相应的SSL证书。如果账户下没有SSL证书，那么请单击下方SLB控制台进行创建。
- 访问端口：用来接收请求，并向后端服务或者应用进行请求转发的监听端口，端口范围为1～65535，例如80。

配置网关路由

5. 配置转发策略。

- 如果没有配置自定义策略转发的请求，那么指定默认的转发策略。如果不设置，那么将会导致访问报错。

5.1 配置自定义转发策略。

域名：输入要转发的请求域名。

Path：输入请求转发路径，例如https://www.aliyun.com/product/sae，/product/sae为请求转发路径。

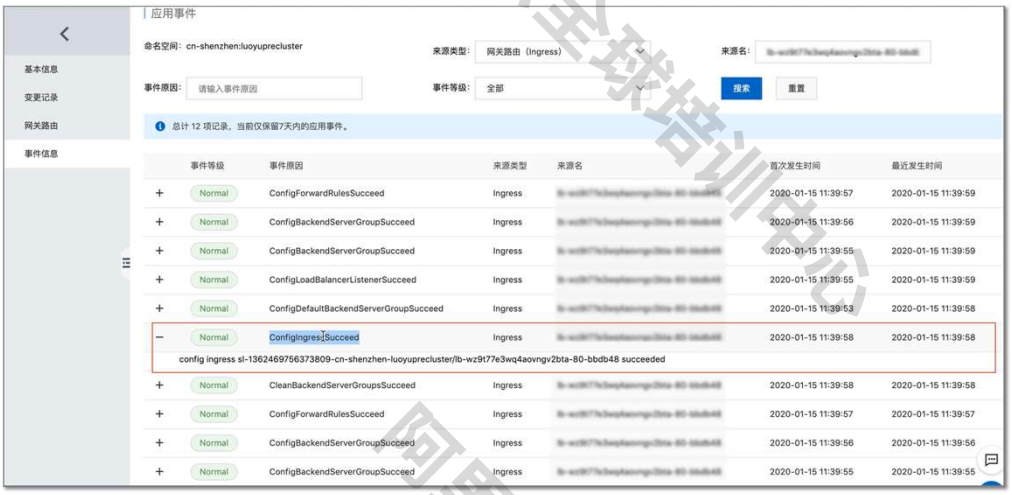
后端应用：接收转发请求的应用，该应用须与转发请求的应用在同一个命名空间内。

容器端口：设置访问您后端应用的容器端口，例如8080。

配置网关路由

6. 验证网关路由是否设置成功。

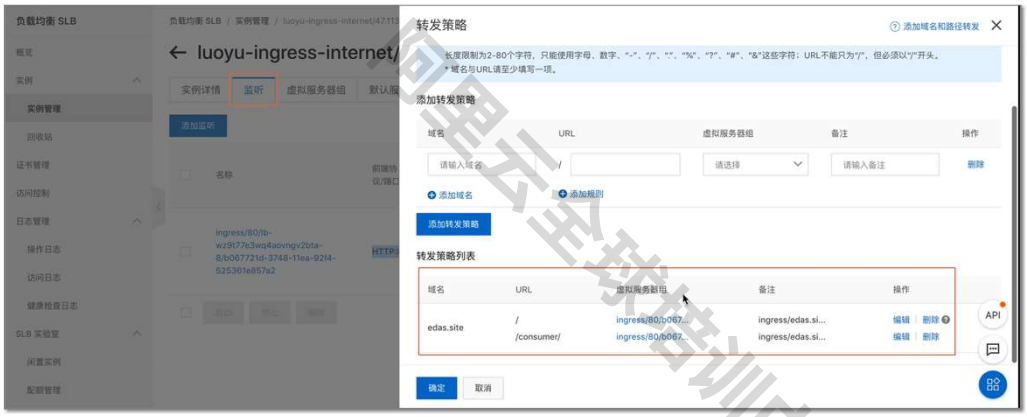
方法一：在应用事件页面查看是否配置成功，具体操作请参见**查看应用事件**。



配置网关路由

6. 验证网关路由是否设置成功。

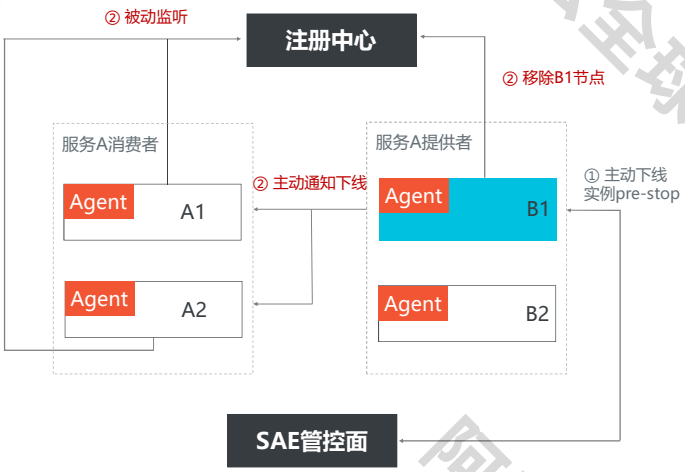
方法二：在SLB控制台查看监听是否已经配置。



方法三：通过域名+路径访问应用，查看是否生效。

微服务无损下线

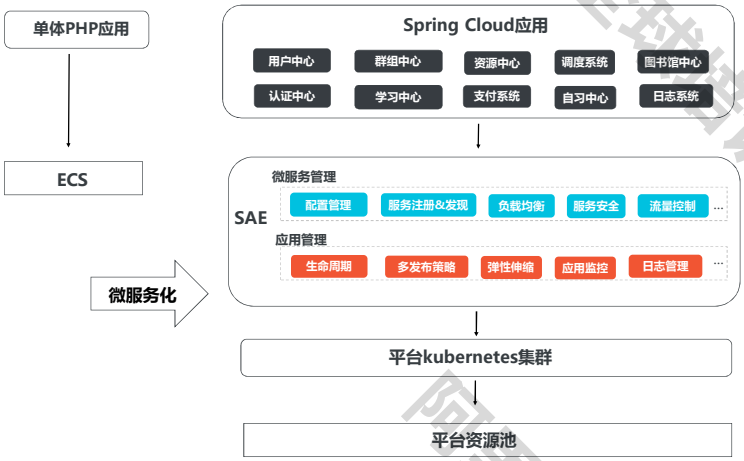
如果需要托管至SAE的微服务应用其处理完请求后再停止，可以使用SAE优雅下线功能（即无损下线）。



课程目录04

高效弹性、低门槛微服务架构转型的解决方案

某教育企业尝试开源自建微服务架构+APM，因技术门槛和人力不足一直没有成功落地生产环境。业务具有潮汐特性，受疫情影响春节期间峰值流量暴增，面临成本压力。



核心价值：

提效：支撑新业务快速上线，让企业专注于业务本身。

省成本：比自建方案节省人力成本。IaaS按需使用按量计费。

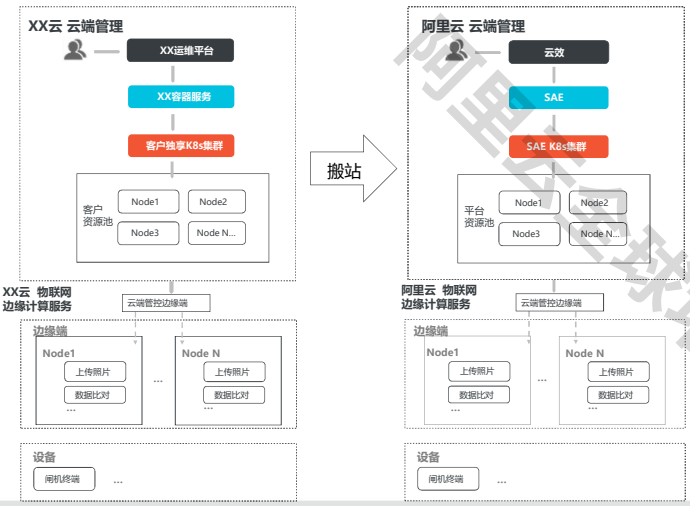
业务稳定：秒级自动弹性，轻松应对流量暴增，保障SLA。

低门槛：零容器基础、开箱即用微服务套件。

阿里云

在线应用一站式的搬站上云

某物联网公司经历了微服务-云原生-物联网架构的转化，对迭代交付效率要求高，想把应用服务转成Serverless架构。期望搬站过程中一并完成DevOps，应用部署能和CICD打通。



核心价值：

快速交付：免运维，聚焦核心业务的开发，提升交付效率。

节省成本：无需固定保有IaaS主机，按需使用，按分钟计费。

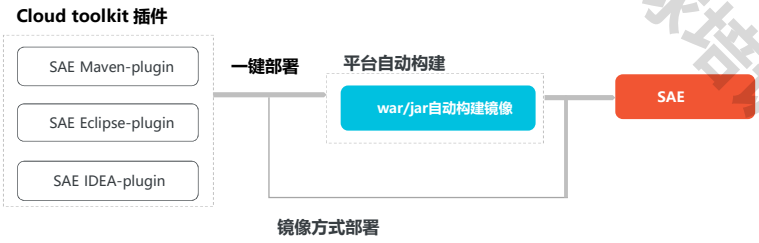
一站式体验：打通微服务生态各产品，Arms应用监控、云效、其它阿里云基础产品（SLB、NAS、SLS等）。

零改造：客户零代码改造迁移Spring Cloud应用。

阿里云

本地一键部署到云端的提效方案

某零售企业之前自建Dubbo框架，经常踩各种稳定性坑，tomcat启不起来。想自建k8s集群解决应用部署密度，学习门槛太高。期望提升部署效率，本地开发完能一键部署到云端。



核心价值：

提效：本地应用开发完通过插件一键部署到云端，并即将可以进行远程调试。

业务稳定：线上微服务应用的稳定性得到极大的提升。

零门槛降成本：零容器基础，war/jar部署也能享用K8s技术红利

一键启停开发测试环境的降本方案

某医疗企业规模较大，有多套测试环境，且光开发测试环境就有20-30台高配的ECS，一般晚上都不使用，长期保有闲置浪费高，降本诉求迫切。

核心价值：

省成本：使用SAE一键启停，按需释放闲置资源，节省2/3开发测试环境机器成本。

总结与回顾

本章节主要讲述内容：

1. Serverless的发展历程，以及服务理念：让用户只专注于业务。

2. 阿里云Serverless服务体系和分类，主要包括：函数计算FC、Serverless应用引擎SAE、Serverless Kubernetes和ECI等。 Serverless应用引擎SAE实现了Serverless架构+微服务架构的完美结合，支持多种微服务框架、多种部署渠道（UI、云效、插件等）、多种部署方式（war、jar、镜像）；核心场景主要面向在线应用：微服务应用、web应用、多语言应用等。

3. Serverless应用引擎SAE的应用场景包括：中大型企业快速构建云上服务应用；应用环境灵活启停，降低资源成本；突发性流量洪流，从容应对。

4. Serverless应用引擎SAE的的四大功能：应用开发、应用部署、应用发布和应用管理。

5. Serverless应用引擎SAE的使用流程和最佳实践。

实验二： 基于Serverless应用引擎SAE部署和管理应用

实验概述

本实验将以Java开发的Demo应用程序，采用WAR包部署方式，展示如何将应用部署到SAE（Serverless应用引擎），并通过绑定公网SLB，让应用可以被公网访问。用户可以：

1. 申请和开通SAE服务
2. 如何在SAE中创建和管理应用；
3. 如何在SAE中启停和删除应用；

实验目标

完成此实验后，可以掌握使用SAE快速创建和部署应用，并对应用进行各种管理操作的能力；

课程目标

学习完本课程后，你将能够：

1. 了解函数计算的特性
2. 掌握阿里云函数计算的功能特性和基本技术原理
3. 熟悉阿里云函数计算的使用和最佳实践

课程目录01

Serverless开启上云2.0时代

企业上云关注什么?

上云1.0 如何架构改造

上云1.0 如何数据迁移

上云2.0 如何专注业务创新

上云2.0 如何带来商业价值

基于IaaS/CaaS的资源托管

核心价值：资源的弹性和成本

面向资源交付

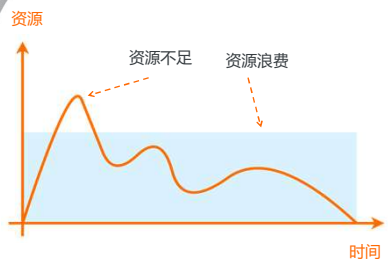
基于函数计算的 Serverless 理念

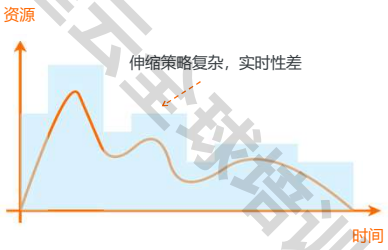
核心价值：免运维和专注业务实现

面向业务交付

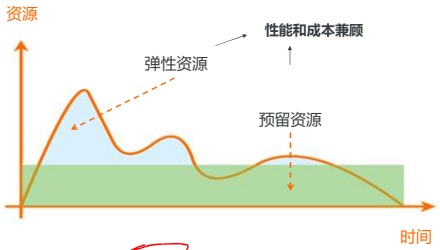
Serverless 连接云生态孤岛

函数计算引领的变革





上云1.0：基于虚机/容器的业务资源模型



上云2.0：基于函数计算 Serverless 架构的业务资源模型

资源管理革命	从人工运维	到云平台工具运维	到 Serverless 免运维	免运维
资源利用率革命	从预算采购低利用率	到有限弹性高利用率	到 Serverless 100%资源利用率	高弹性
资源成本革命	从固定成本支出	到根据资源策略伸缩	到 Serverless 根据业务策略适配	低成本

传统应用与基于函数计算的应用构建方式的对比

传统应用构建方式：

1. 购买服务器
2. 更新操作系统
3. 搭建开发环境
4. 构建/部署应用
5. 配置负载均衡机制
6. 搭建日志分析和监控系统
7. 监控应用运行情况

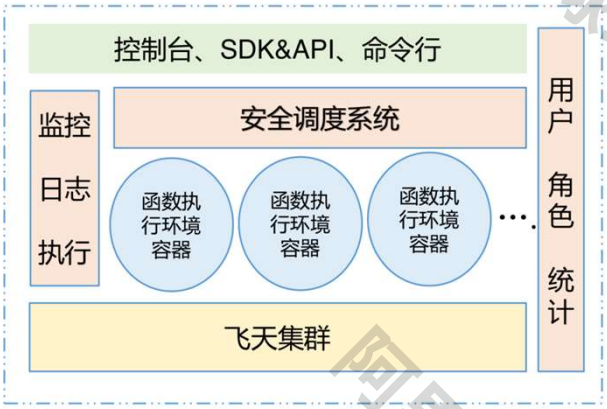
基于函数计算的应用构建方式：

1. 购买服务器
2. 更新操作系统
3. 搭建开发环境
4. 构建/部署应用
5. 配置负载均衡机制
6. 搭建日志分析和监控系统
7. 监控应用运行情况

阿里云函数计算的定义

一个无服务器的全托管的运行环境，客户只需编写代码并设置运行的条件，即可以弹性、安全地运行。函数计算会自行维护服务器资源，网络资源，以及消息分发和负载均衡等功能，函数计算是按运行时长计费，不执行0费用。

函数计算功能框图



优点:

- 能大规模并行执行代码
- 无需管理运行环境
- 容器自动配置扩展，松耦合
- 事件触发运行代码方式
- 不为空闲的资源付费

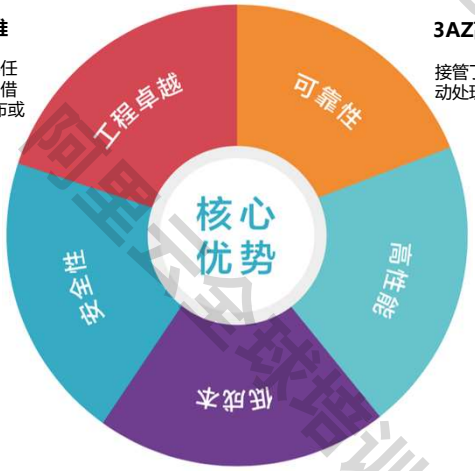
阿里云函数计算的核心优势

专注业务逻辑开发、免运维

只需专注业务代码编写，无需关心任何底层基础设施及软件网络配置，借助函数版本/别名功能实现灰度发布或A/B 测试

VM级别租户隔离

身份认证、访问控制、数据安全、运行时安全、网络安全方面对用户的应用进行全方位的保护



3AZ高可用架构

接管了基础设施的管理工作，系统能自动处理基础设施层的各种错误

毫秒级自动弹性伸缩

短时间内处理大量数据文件，或者在请求流量快速增加的同时仍能保持稳定的延时， 是明显波峰波谷应用最佳终结者

按需付费 + 预付费

按需付费，只需为代码实际运行消耗的资源付费，预付费可以优化负载较为平稳业务的财务成本

阿里云函数计算的典型应用场景

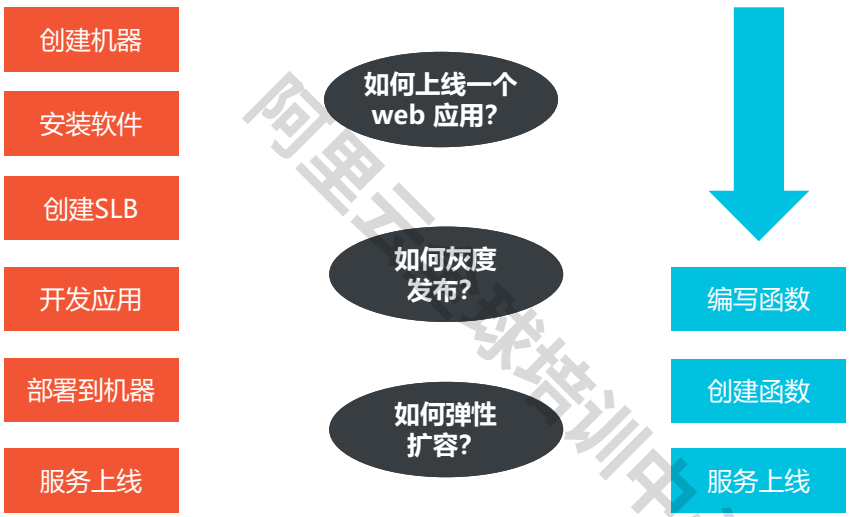
- Web应用
- 音视频处理系统
- 异步数据处理系统
- 实时流处理系统
- 图文处理系统
- AI推理系统

课程目录02

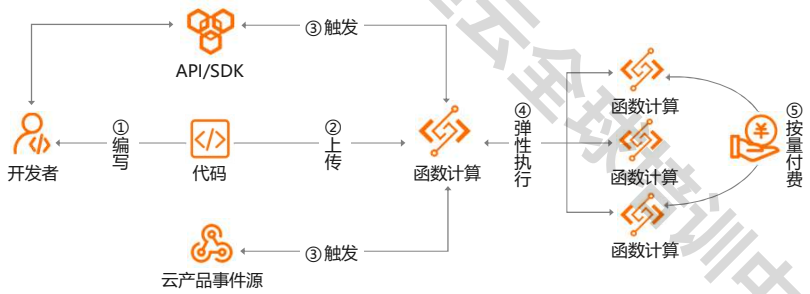
1. 函数计算的概述
2. **函数计算的功能特性**
   - **2.1 函数计算的架构**
   - **2.2 函数计算的工作流程**
   - **2.3 函数计算的事件驱动**
3. 函数计算的基本使用
4. 函数计算的最佳实践

函数计算的整体架构

函数计算的快速上线功能



函数计算的工作流程



- 通用 Serverless 计算平台
- 云端事件源无缝集成
- 自动弹性伸缩，按量付费
- 只需专注业务逻辑开发即可

调用方式

- 同步调用
- 异步调用

丰富的编程语言

✓ nodejs ✓ python ✓ java ✓ php ✓ C#

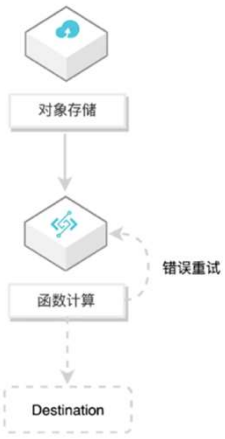
✓ 自定义，可以使用任何语言，比如 go

函数调用：同步调用

- 服务端会立刻返回计算结果
- 执行过程中遇到错误，会将错误返回客户端

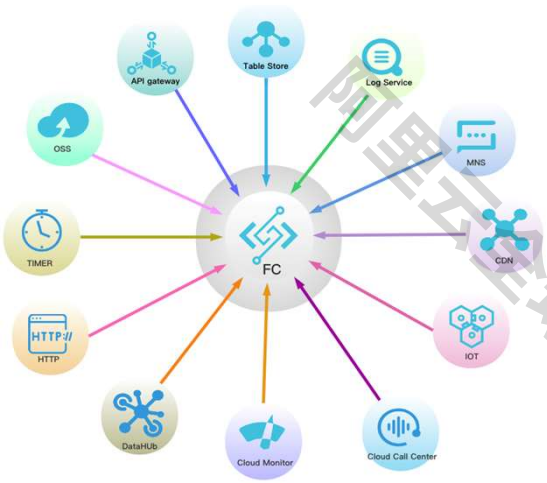
注：函数计算不会对错误进行重试，需要客户端添加重试机制

函数调用：异步调用



- 异步调用将触发请求放到队列中就返回，不会等待函数调用结束。
- 执行过程中遇到错误，会对错误进行重试，函数错误重试三次，系统错误会以指数退避方式无限重视。
- 适用于批量数据处理

函数计算的事件驱动



丰富的事件源

事件源还在增加中 ...

✓ 对象存储	✓ API 网关	✓ 消息服务
✓ 表格存储	✓ 日志服务	✓ HTTP
✓ 定时器	✓ IOT	✓ DataHub
✓ CDN	✓ 云监控	✓ 云呼叫中心

事件驱动，简化编程模型，编写少量的代码即可串联多个服务实现复杂的功能

课程目录03

函数计算的使用流程

流程说明如下：

1. 创建服务。
2. 创建函数，编写代码，将应用部署到函数中。
3. 以事件源触发函数。
4. 查看执行日志。
5. 查看服务的监控

创建服务

1. 登录**函数计算**控制台。
2. 在顶部菜单栏，选择地域。
3. 在左侧导航栏中，单击**服务/函数**。在服务/函数页面右上角单击新增服务。

创建函数

1. 登录**函数计算**控制台。
2. 在顶部菜单栏，选择地域。
3. 在左侧导航栏中，单击**服务/函数**。在服务/函数页面右上角单击**新增函数**。

创建函数

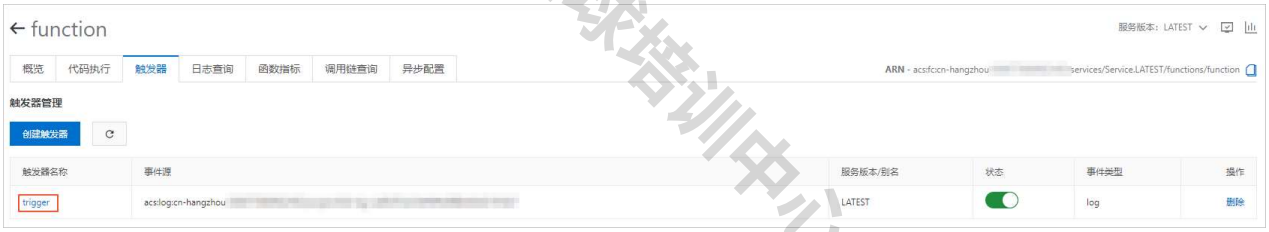
- 在**新建函数**页面选择创建的函数类型或函数模板，然后单击**配置部署**。

创建函数

- 在**新建函数**区域，设置相关参数，然后单击**完成。**

创建触发器

- 在函数详情页面，单击**触发器**页签。
- 在触发器列表中，单击目标触发器名称。然后在**修改触发器**区域，根据需要更新触发器的配置



课程目录04

日志的大规模流式处理和并行计算

用户访问日志存入日志服务　　数据流触发函数执行，根据负载自动扩容　　存储处理结果

敏捷开发免运维： 聚焦顶层设计，释放大量的运维人员，降低人力成本。

弹性高可用：用户访问产生海量的访问日志巨大， 白天的某些时间段有明显的峰值，使用 FC 能避免资源浪费，降低资源成本。

降低成本：用户访问产生海量的访问日志有明显波峰波谷， 凌晨一般来说访问日志产生较少， 按量付费节约了成本。

海量图片的处理和快速访问

敏捷开发免运维：只需要专注业务逻辑代码的编写， 工程效率大幅提高，同时云平台提供完善的监控设施。

弹性高可用：毫秒级伸缩扩容，保证应用在负载动态变化时仍能保持稳定的延时，平滑处理海量调用请求。

多版本灰度：只多版本功能助力用户安全灰度升级函数。

降低成本：按需付费，客户不需要为峰值预留计算资源。

全终端全平台的网络视频业务

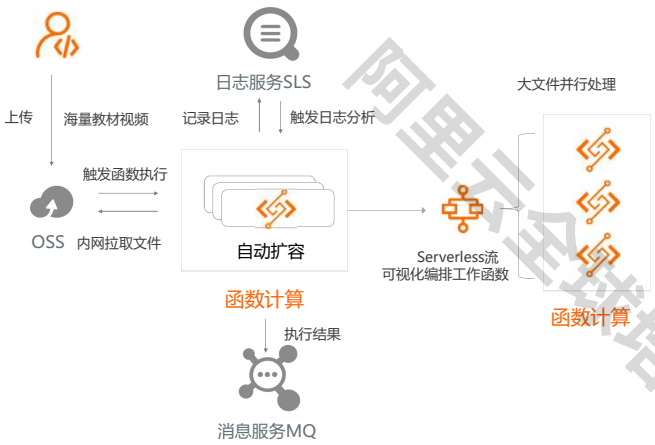
全Serverless架构：整个流程高度自动化，视频的上传、处理、播放、分发在极短的时间内完成。

弹性高可用：计算力不再是瓶颈，事件驱动能力保证实时性，加快处理速度，提高处理效率。

稳定性：即使在上传视频高峰期，视频处理也不会有任何的排队。保障了服务的稳定性。

降低成本：有效应对短视频的突发在线流量，按需付费，提高资源利用率，降低总体成本。

直播、推流、转码一站式解决方案

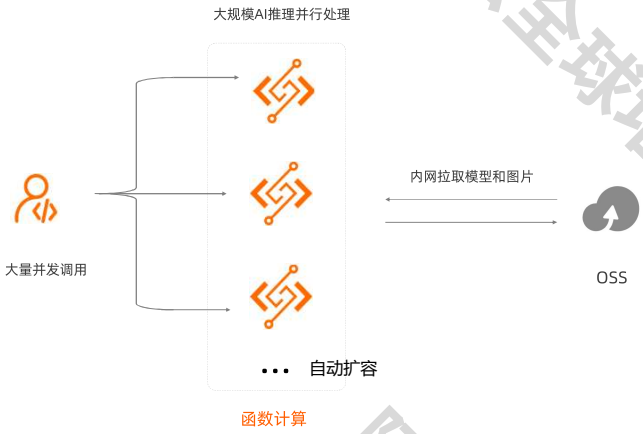


快速上云：充分享受云原生的技术红利，完全按需调用的 Serverless 方案，任务编排实现复杂的并行处理方计算案落地，人力成本从>30人日优化到~3人日。

弹性免运维：能够在短时间内迅速调集上万个实例的计算资源，实现视频处理任务的快速执行；不需要预留计算资源，也不需要对底层的软硬件进行维护，极大地降低了开发和运营成本；

降低成本：相比于传统的方式，基于函数计算的方案帮助客户节省了 60% 左右的 IT 成本投入。

Serverless AI建模：



降低成本： AI 推理CPU密集型，基于函数计算的推理CPU利用率高， 整体节约了30% 的成本

预付费 + 后付费：预付费支撑日常资源消耗，降低70%，后付费应对峰值时高弹性需求。

预留资源：预留资源应对峰值计算需求，无冷启动，保障业务高性能、高可用

总结与回顾

本章节主要讲述内容：

1. Serverless新生态：函数计算+阿里云，连接云生态孤岛。
2. 函数计算引领的变革：免运维、高弹性和低成本。
3. 传统应用与基于函数计算的应用构建方式的对比。
4. 函数计算的定义和应用场景，以及函数计算的整体架构和工作流程。
5. 函数计算的使用流程和最佳实践。

课程目标

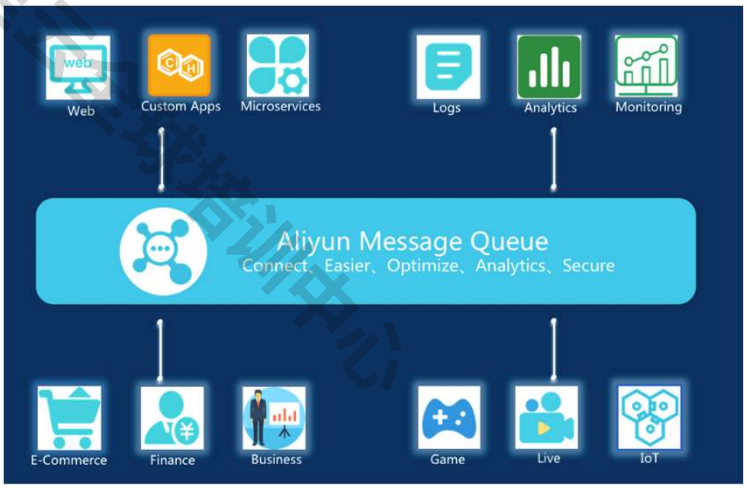
学习完本课程后，你将能够：

1. 了解阿里云消息队列产品RocketMQ
2. 掌握消息队列RocketMQ产品功能特性和基本技术原理

课程目录01

消息队列的应用场景

- 应用解耦
- 削峰填谷
- 异步通知
- 大数据处理
- 分布式事务
- IM 实时通信
- Cache 同步
- 移动互联网（Apps、互动直播）
- 物联网（连联网）



应用场景和优势

消息队列RocketMQ版是阿里云基于Apache RocketMQ构建的低延迟、高并发、高可用、高可靠的分布式消息中间件。

- 分布式应用系统提供异步解耦和削峰填谷
- 海量消息堆积、高吞吐、可靠重试

生态丰富

- 消息类型：定时消息、顺序消息、事务消息
- 多种订阅：Pub/Sub、Tag 过滤、SQL 过滤
- 云产品集成：IoT、大数据、音视频等各领域

性能优越

- 海量消息堆积能力
- 毫秒级端到端延迟
- 千万级高并发处理能力

Apache RocketMQ

The Apache Software Foundation Announces Apache® RocketMQ™ as a Top-Level Project

便捷的运维体系

- 消息回溯、死信队列、监控告警
- 多维度消息查询 、全链路消息轨迹
- 主子账号、TLS 传输加密

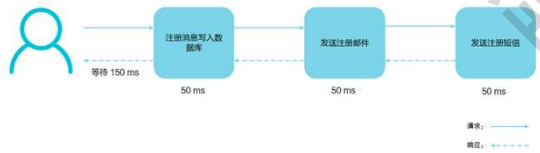
服务健壮

- 99.99% 服务可用性
- 99.99999999% 数据可靠性
- 熔断机制、消息重投机制

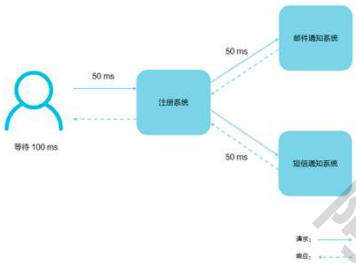
异步解耦场景

场景描述：用户注册后需要发送注册邮件和短信通知两部分内容

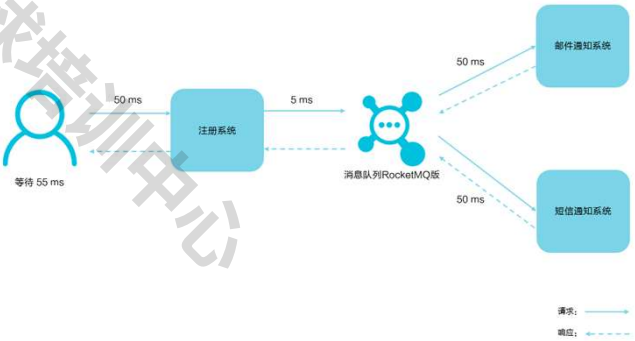
1.串行方式，共需要150ms



2.并行方式，共需要100ms



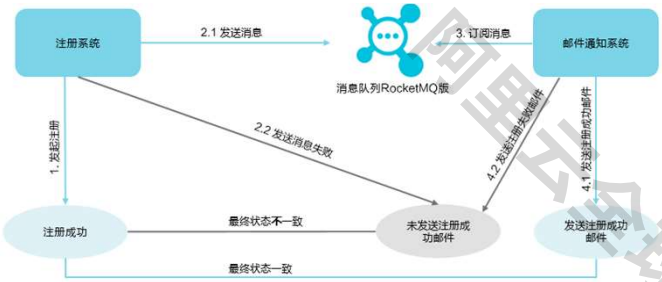
3.异步解耦，引入RocketMQ后，共需要55ms



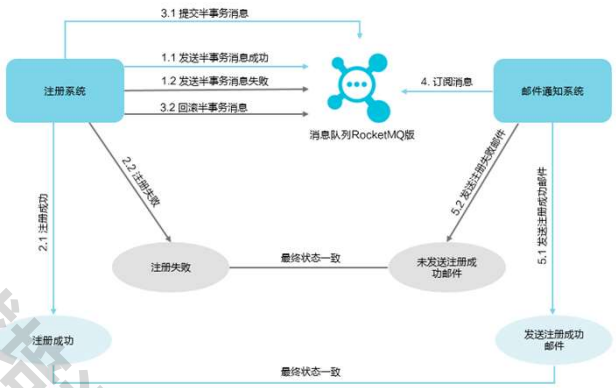
阿里云

分布式事务一致性场景

场景描述：用户注册流程中，注册系统和邮件通知系统通过消息队列，保持最终一致



1.系统解耦完成，但是没有做到数据最终一致



2.通过消息队列的事务消息，实现数据最终一致

阿里云

消息顺序收发和削峰填谷场景

➢ **顺序收发场景**

- 全局顺序

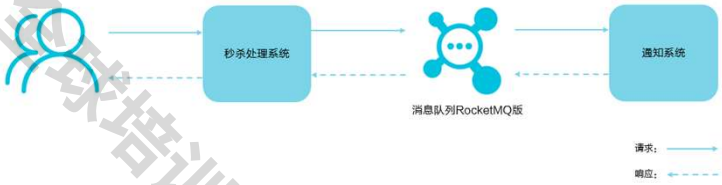
对于指定的一个Topic，所有消息将按照严格的先入先出（FIFO）的顺序，进行顺序发布和顺序消费

- 分区顺序

对于指定的一个Topic，所有消息根据Sharding Key进行区块分区，同一个分区内的消息将按照严格的FIFO的顺序

例如，在注册场景中，可使用用户ID作为Sharding Key来进行分区。

➢ **削峰填谷场景**



1.秒杀处理系统按照秒杀处理逻辑将成功请求发送至消息队列RocketMQ版。

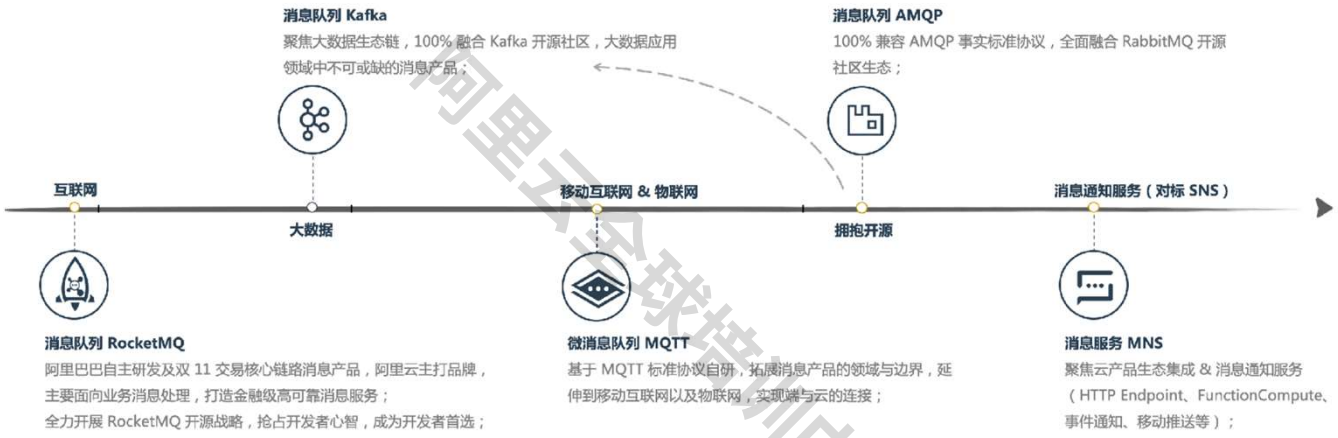
2.通知系统订阅消息队列RocketMQ版的秒杀相关消息，异步发送通知。

3.用户收到秒杀成功的通知。

4.用户发起海量秒杀请求到秒杀业务处理系统。

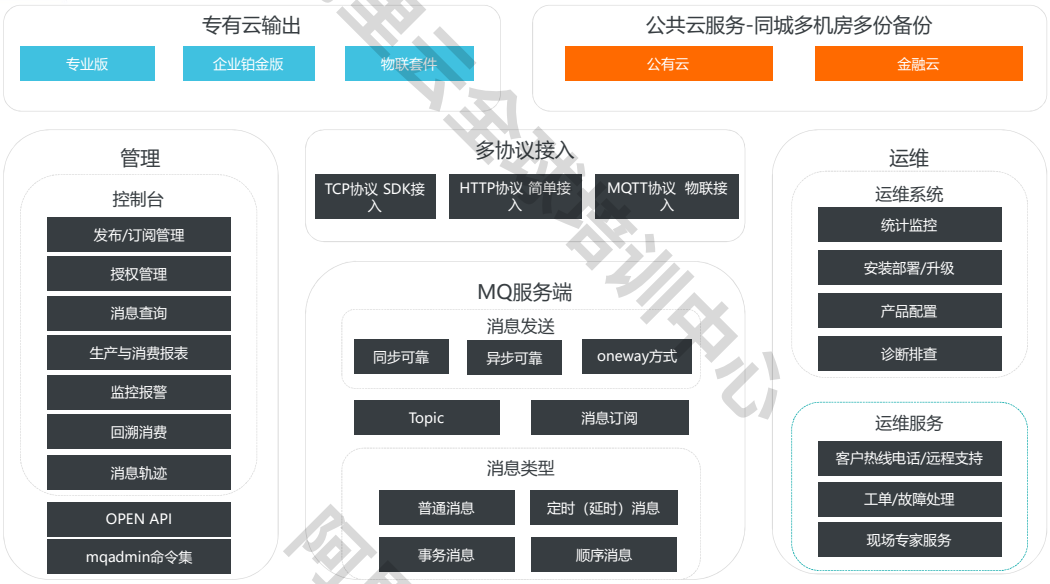
阿里云

消息队列产品家族



阿里云

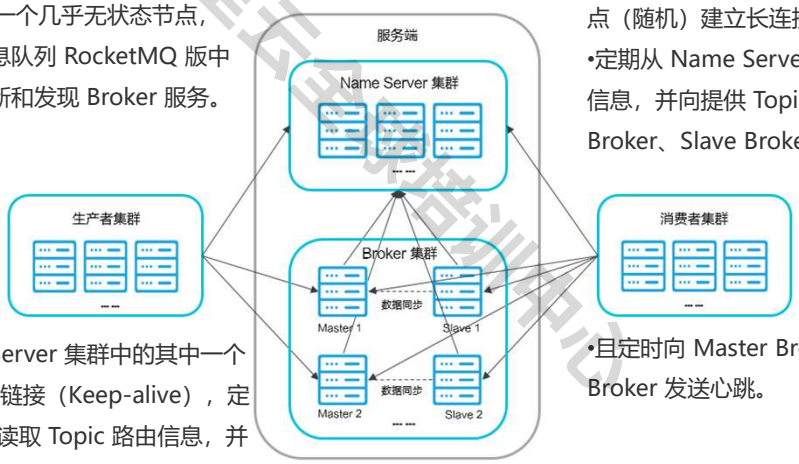
产品功能和概览



课程目录02

产品核心模块

- Name Server：是一个几乎无状态节点，可集群部署，在消息队列 RocketMQ 版中提供命名服务，更新和发现 Broker 服务。



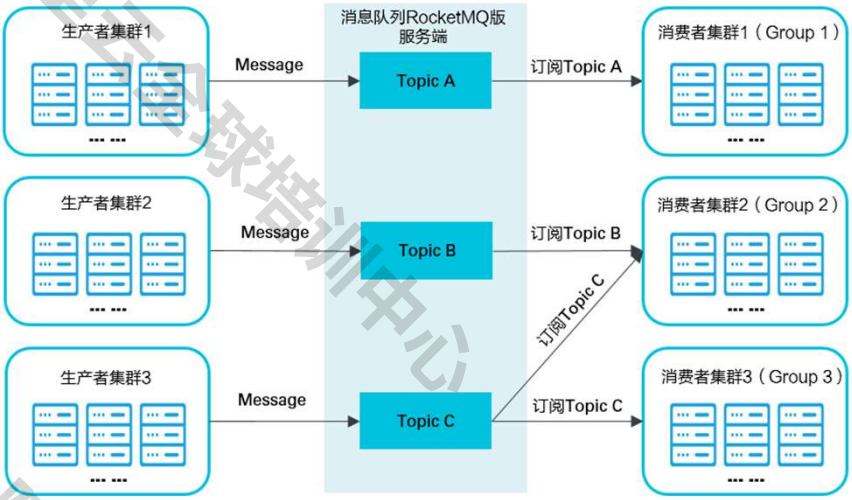
- 消费者：
  - •与 Name Server 集群中的其中一个节点（随机）建立长连接
  - •定期从 Name Server 拉取 Topic 路由信息，并向提供 Topic 服务的 Master Broker、Slave Broker 建立长连接
  - •且定时向 Master Broker、Slave Broker 发送心跳。
- 生产者：与 Name Server 集群中的其中一个节点（随机）建立长链接（Keep-alive），定期从 Name Server 读取 Topic 路由信息，并向提供 Topic 服务的 Master Broker 建立长链接，且定时向 Master Broker 发送心跳。
- Broker：消息中转角色，负责存储消息，转发消息。

关键名词解释

- ✓ **Topic** 消息主题，一级消息类型，通过 Topic 对消息进行分类。
- ✓ **Tag** 消息标签，二级消息类型，用来进一步区分某个 Topic 下的消息分类。
- ✓ **Group ID** 是Group的标识，一类Producer或Consumer，这类Producer或Consumer通常生产或消费同一类消息，且消息发布或订阅的逻辑一致。
- ✓ **Message ID** 消息的全局唯一标识，由 MQ 系统自动生成，唯一标识某条消息。
- ✓ **Message Key** 消息的业务标识，由消息生产者（Producer）设置，唯一标识某个业务逻辑

消息收发模型

- 消息队列 RocketMQ 版支持发布/订阅模型
  - 消息生产者应用创建 Topic 并将消息发送到 Topic
  - 消费者应用创建对 Topic 的订阅以便从其接收消息
- 通信可以是：
  - 一对多（扇出）
  - 多对一（扇入）
  - 多对多



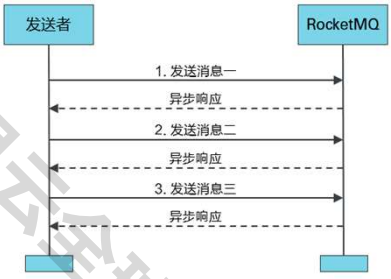
消息发送模式

同步发送

➢ 应用场景

适用场景广泛，
例如重要通知邮件、报名短信通知、营销短信系统等。

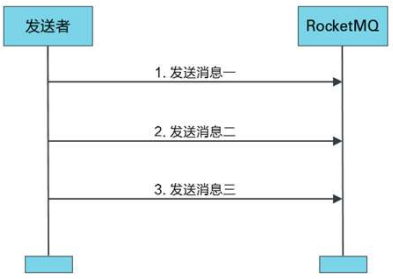
异步发送



➢ 应用场景

适用于链路耗时较长，
对响应时间较为敏感的业务场景，
例如，视频上传后通知启动转码服务，转码完成后通知推送转码结果等。

单向发送

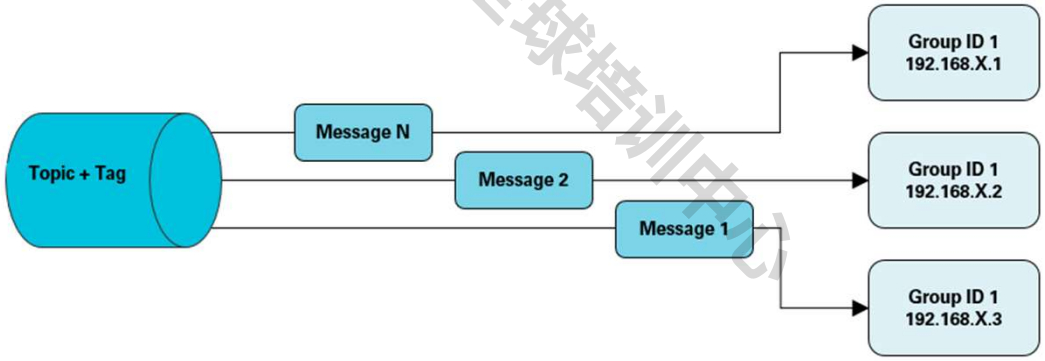


➢ 应用场景

适用于某些耗时非常短，但对可靠性要求并不高的场景，
例如日志收集。

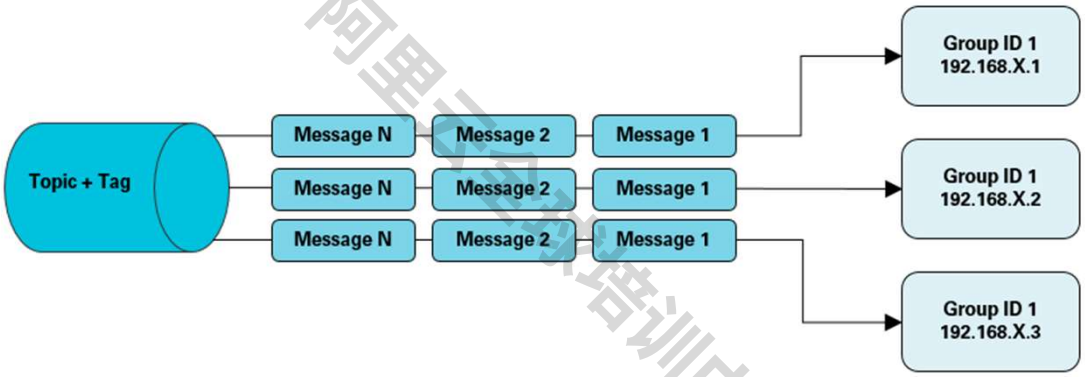
集群消费模式

- 集群消费模式：适用于消费端集群化部署，每条消息只需要被处理一次的场景。此外，由于消费进度在服务端维护，可靠性更高。



广播消费模式

- 广播消费模式：适用于消费端集群化部署，每条消息需要被集群下的每个消费者处理的场景。



消息类型总览

普通消息

- 指消息队列 RocketMQ 版中无特性的消息，区别于有特性的定时/延时消息、顺序消息和事务消息。

定时/延时消息

- 消息在发送到消息队列 RocketMQ 版服务端后并不会立马投递，而是根据消息中的属性推迟到某个时间或延迟固定时间后才投递给消费者。

顺序消息

- MQ提供的一种严格按照顺序进行发布和消费的消息类型。顺序消息指 消息发布和消息消费都按顺序进行。

事务消息

- MQ提供类似X/Open XA的分布事务功能，通过MQ事务消息能达到分布式事务的最 终一致。

全局顺序消息

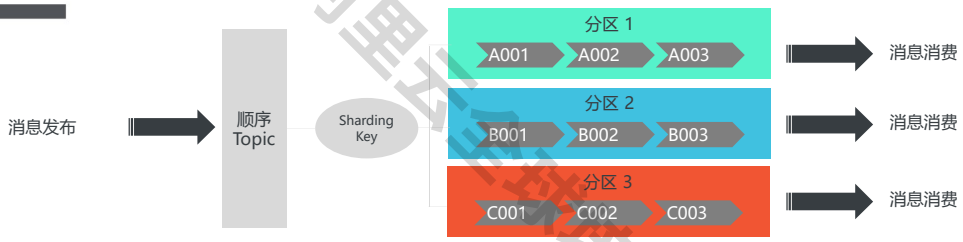
全局顺序

对于指定的一个 Topic，所有消息按照严格的先入先出（FIFO）的顺序进行发布和消费。

适用场景

对于性能要求不高的全局性业务场景，如：证券交易平台中，同一股别的交易是全平台按照订单的创建时间顺序进行交易，则可以采用全局顺序。

分区顺序消息

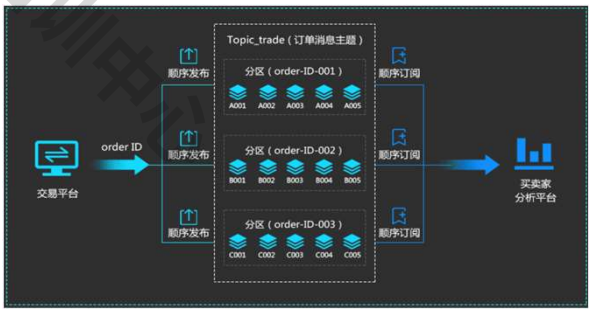


分区顺序

对于指定的一个 Topic，所有消息根据 sharding key 进行区块分区。 同一个分区内的消息按照严格的 FIFO 顺序进行发布和消费。

适用场景

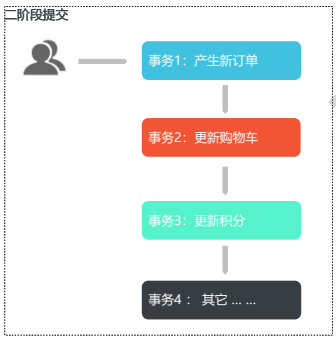
对于性能要求高，非全局性有序，可拆分出最小有序单元的系统，如：电商系统，一个完整的订单处理流程为最小有序单元，以订单 ID 作为 sharding key，那么同一个订单相关的创建订单消息、订单支付消息、订单退款消息、订单物流消息都会按照先后顺序来发布和订阅。



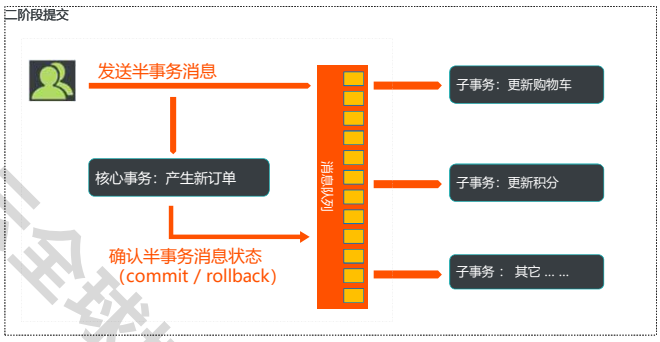
阿里云

事务消息

传统事务



分布式事务：应用之间松耦合 & 数据最终一致性



传统的事务处理

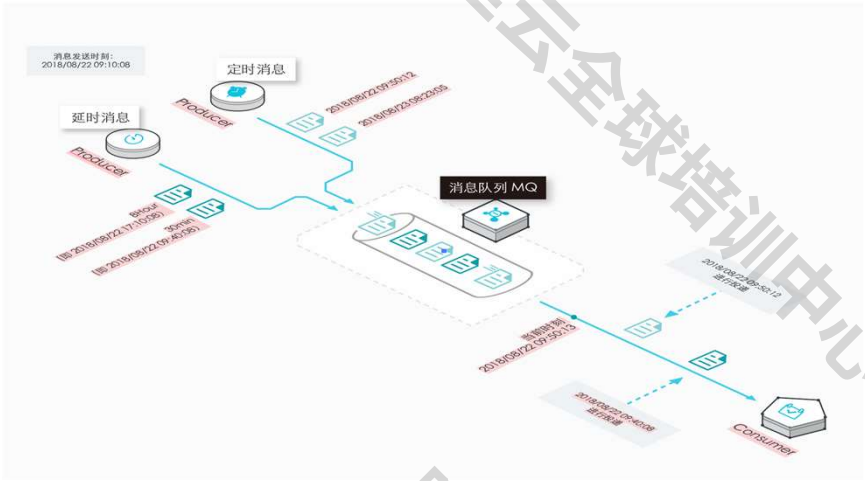
- 多个系统之间的交互耦合到一个事务中，响应时间长，影响系统可用性；

引入分布式事务消息

- 交易系统与消息队列之间，组成一个事务处理；
- 下游业务系统（购物车、积分、其它）相互隔离，并行处理；

阿里云

定时消息&延时消息



交易场景

交易中超时未支付关闭订单的场景，在订单创建时会发送一条延时消息。这条消息将会在 30 分钟以后投递给消费者，消费者收到此消息后需要判断对应的订单是否已完成支付，如支付未完成，则关闭订单。

定时任务

通过消息触发一些定时任务，比如在某一固定时间点向用户发送提醒消息。

阿里云

高级特性

消息重试

消息过滤

消息去重

消息幂等

阿里云

消息重试

顺序消息的重试

当消费者消费消息失败后，MQ会自动不断进行消息重试(每次间隔时间为1 秒)，这时，应用会出现消息消费被阻塞的情况。因此，建议您使用顺序消息时，务必保证应用能够及时监控并处理消费失败的情况

无序消息的重试

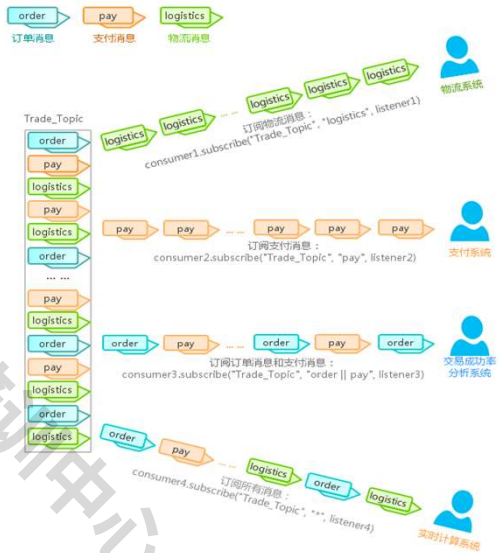
当消费者消费消息失败时，您可以通过设置返回 状态达到消息重试的结果。只针对集群消费方式生效;广播方式不提供失败重试特性，即消费失败后，失败消 息不再重试，继续消费新的消息。

- MQ默认允许每条消息最多重试16次

第几次重试	与上次重试的间隔时间	第几次重试	与上次重试的间隔时间
1	10 秒	9	7 分钟
2	30 秒	10	8 分钟
3	1 分钟	11	9 分钟
4	2 分钟	12	10 分钟
5	3 分钟	13	20 分钟
6	4 分钟	14	30 分钟
7	5 分钟	15	1 小时
8	6 分钟	16	2 小时

消息过滤

- MQ允许消费者按照Tag对消息进行过滤，确保消费者最终只消费到他关心的消息类型。
- 从客户下单到收到商品这一过程会生产一系列消息，比如订单创建消息（order）、支付消息（pay）、物流消息（logistics）。
- 这些消息会发送到Topic为Trade_Topic的队列中，被各个不同的系统所接收，比如支付系统、物流系统、交易成功率分析系统、实时计算系统等。其中，物流系统只需接收物流类型的消息（logistics），而实时计算系统需要接收所有和交易相关（order、pay、logistics）的消息

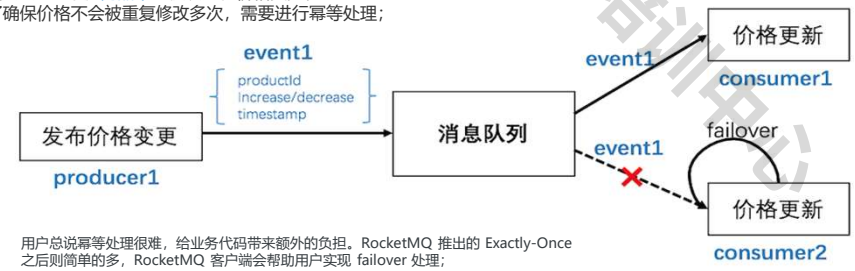


消息去重（Exactly-Once）

业务概念：
- At-Least-Once：消息至少投递一次，RocketMQ 的默认支持方式；
- At-Most-Once：消息至多投递一次；
- Exactly-Once：消息投递有且仅有一次；

经典案例：
以电商系统为例，交易、商品、店铺、活动会场均以微服务方式构建：
- 上游运营支撑系统通过实时计算模块发布商品价格变更的信息；
- 异步通知到下游商品中心模块进行价格变更；

为了确保价格不会被重复修改多次，需要进行幂等处理；



用户总说幂等处理很难，给业务代码带来额外的负担。RocketMQ 推出的 Exactly-Once 之后则简单的多，RocketMQ 客户端会帮助用户实现 failover 处理；

消费幂等

消息幂等概念
- 当出现消费者对某条消息重复消费的情况时，重复消费的结果与消费一次的结果是相同的，并且多次消费并未对业务系统产生任何负面影响，那么这整个过程就实现可消息幂等。

消息重复场景
- 发送消息时重复
- 投递消息时重复
- 负载均衡时消息重复

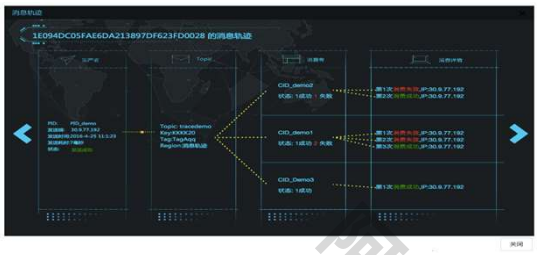
处理方法
- Message ID 有可能出现冲突（重复）的情况，所以真正安全的幂等处理，不建议以 Message ID 作为处理依据。
- 最好的方式是以业务唯一标识作为幂等处理的关键依据，而业务的唯一标识可以通过消息 Key 设置。

消息运维体系

- 消息查询

- 消息回溯 & 清除

- 消息轨迹



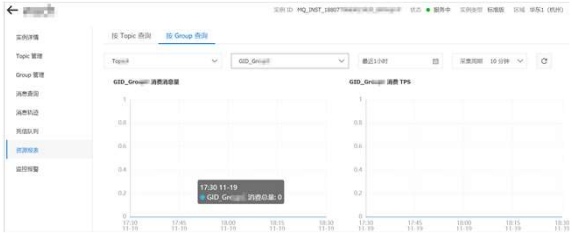
- 监控告警

阿里云

消息运维

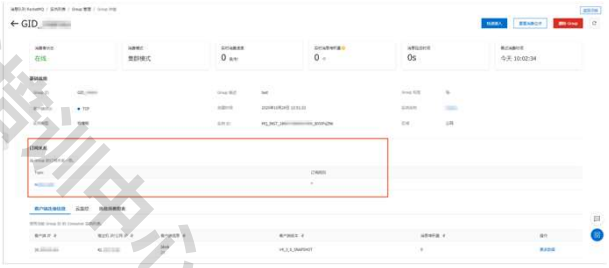
- 查看生产消息量

- 查看消费消息量



- 查看Topic被哪些Group ID订阅

- 查看Group ID订阅的Topic



阿里云

消息轨迹

- 是一条消息从生产方发出到消费方消费处理；
- 整个过程中的各个相关节点的时间地点等数据汇聚而成的完整链路信息。

消息轨迹数据

每条消息的生产方、服务方、消费方三个角色全链路可视化追踪

阿里云

消息轨迹

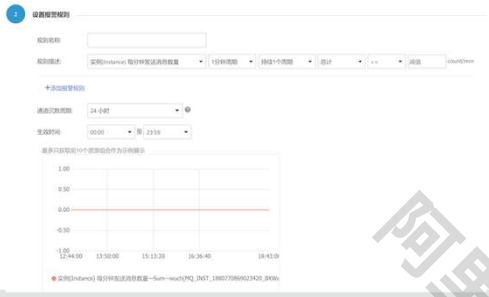
- 消息体中设置 Key，记录业务的唯一标识
- 消息发送完成时，记录 Message ID

阿里云

监控告警

1.关联资源实例

2.设置报警规则



3.设置通知方式

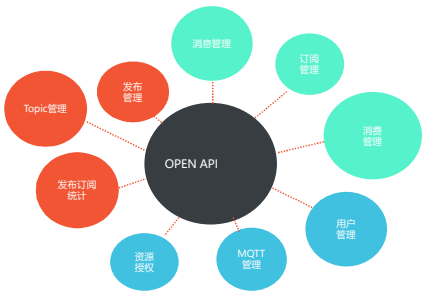
阿里云

OpenAPI使用介绍

Open API 是 MQ 提供给用户的管控方式，用于实现一系列资源管理和运维功能。

HTTP 协议
采用 HTTP RESTful 标准，方便易用，快速接入，跨网络能力强；

7 种多语言客户端
Java / C++ / .NET / PHP / Go / Node.js / Python



➢ **OpenAPI使用举例**

1、公共参数部分

https:/ons.cn-hangzhou.aliyuncs.com/?Action=OnsRegionList &Format=JSON &**Version**=2019-02-14 &**AccessKeyId**=key-test &**Signature**=Pc5WB8gokVn0xfeu%2FZV%2BiNM1dgI%3D &**SignatureMethod**=HMAC-SHA1 &**Timestamp**=2020-01-01T12:00:00Z &SignatureNonce=15215528852396 &SignatureVersion=1.0 ...

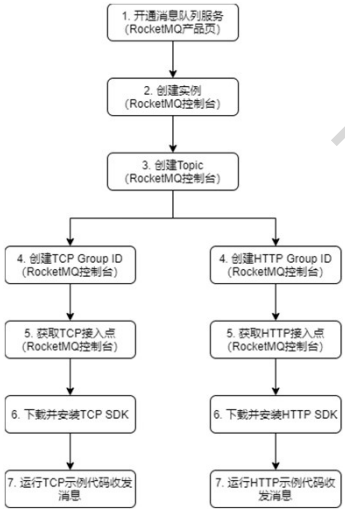
2、创建Topic操作示例

http(s)://ons.cn-zhangjiakou.aliyuncs.com/?**Action=OnsTopicCreate** &InstanceId=MQ_INST_188077086902****_BXSuW61e &MessageType=0 &Topic=**test** &<公共请求参数>

阿里云

课程目录03

完整使用流程概述



- 基本使用流程四步曲

  步骤1： 创建资源与实例

  步骤2： 创建Topic与Group ID

  步骤3： 获取接入点

  步骤4： 使用SDK收发消息

- 注意事项

  1. 同一个Group ID不能混用于TCP协议和HTTP协议。

  2. 同一个实例，需要分别获取TCP协议和HTTP协议的SDK来使用对应协议的接入点，不能混用。

  3. 跨地域使用消息队列RocketMQ版的场景，推荐使用HTTP协议。

创建资源与实例

1.开通消息队列服务

2.选择网络类型

3.创建RocketMQ实例

注意事项

1.网络访问限制，只有在同一个地域下的同一个实例中的Topic和Group ID才能互通。

如果需要通过**公网**使用消息队列RocketMQ版服务，请将Topic和Group ID都创建在**公网**地域下的实例中。

2.标准版以按量付费的方式计费，铂金版以包年包月的方式计费，价格差异较大，按需选择

阿里云

创建Topic与Group ID

➢ 创建Topic

➢ 创建Group ID

注意事项

1.Topic不能跨实例使用

2.同一个实例中的Topic名称必须唯一

3.每个Topic的消息类型唯一

注意事项

1.同一实例中Group ID必须唯一；

2.Group ID和Topic的关系是N:N

阿里云

获取接入点

- 获取RocketMQ的接入点信息，生产端和消费端都需要配置改接入点信息。

TCP 接入点

使用 RocketMQ 的 TCP SDK 进行消息收发时，所需要配置的接入地址。

客户端协议	网络	Endpoint
TCP	公网访问	http://MQ_INST_188[redacted]_BXWFqZNk.mq-internet-access.mq-internet.aliyuncs.com:80
TCP	内网访问	n/a

注意事项

1.同地域不同实例的接入点连接地址不一样

2.公网地域的实例才有TCP协议的公网接入点，其余地域的的只有TCP协议的内网接入点

3.TCP协议的接入点不可跨地域使用

使用SDK收发消息

```java
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.OnExceptionContext;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.SendCallback;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;

import java.util.Properties;

public class ProducerTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // AccessKey ID阿里云身份验证，在阿里云服务器管理控制台创建。
        properties.put(PropertyKeyConst.AccessKey, "XXX");
        // AccessKey Secret阿里云身份验证，在阿里云服务器管理控制台创建。
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        //设置发送超时时间，单位毫秒。
        properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");
        // 设置TCP协议接入点，进入控制台的实例详情页面的TCP协议客户端接入点区域查看。
        properties.put(PropertyKeyConst.NAMESRV_ADDR,
          "XXX");

        Producer producer = ONSFactory.createProducer(properties);
        // 在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
        producer.start();

        Message msg = new Message(
                // Message所属的Topic。
                "TopicTestMQ",
                // Message Tag，可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过
                "TagA",
                // Message Body，任何二进制形式的数据，消息队列RocketMQ版不做任何干预，需要P
                "Hello MQ".getBytes());

        // 设置代表消息的业务关键属性，请尽可能全局唯一，以方便您在无法正常收到消息情况下，可通
        // 注意：不设置也不会影响消息正常收发。
        msg.setKey("ORDERID_100");

        // 异步发送消息，发送结果通过callback返回给客户端。
        producer.sendAsync(msg, new SendCallback() {
            @Override
            public void onSuccess(final SendResult sendResult) {
                // 消息发送成功。
                System.out.println("send message success. topic=" + sendResult.getTopic
            }

            @Override
            public void onException(OnExceptionContext context) {
                // 消息发送失败，需要进行重试处理，可重新发送该条消息或持久化该条数据进行补偿
                System.out.println("send message failed. topic=" + context.getTopic() +
            }
        });

        // 在callback返回之前即可取得msgId。
        System.out.println("send message async. topic=" + msg.getTopic() + ", msgId=" +

        // 在应用退出前，销毁Producer对象。 注意：如果不销毁也没有问题。
        producer.shutdown();
    }
}
```

1.根据协议类型和开发语言版本，下载不同类型的SDK

2.调用TCP协议的SDK发送普通消息

3.通过控制台“消息查询”查看消息是否发送成功

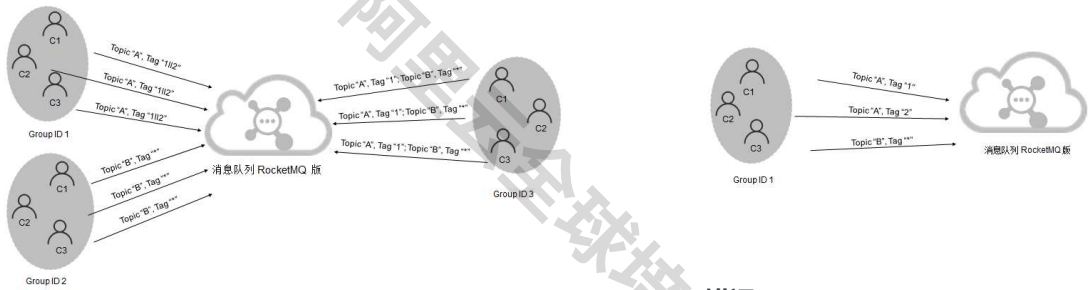
```java
import com.aliyun.openservices.ons.api.Action;
import com.aliyun.openservices.ons.api.ConsumeContext;
import com.aliyun.openservices.ons.api.Consumer;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.MessageListener;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;

import java.util.Properties;

public class ConsumerTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // 您在控制台创建的Group ID。
        properties.put(PropertyKeyConst.GROUP_ID, "XXX");
        // AccessKey ID阿里云身份验证，在阿里云服务器管理控制台创建。
        properties.put(PropertyKeyConst.AccessKey, "XXX");
        // AccessKey Secret阿里云身份验证，在阿里云服务器管理控制台创建。
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        // 设置TCP协议接入点，进入控制台的实例详情页面的TCP协议客户端接入点区域查看。
        properties.put(PropertyKeyConst.NAMESRV_ADDR,
          "XXX");
        // 集群订阅方式（默认）。
        // properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.CLUSTERING);
        // 广播订阅方式。
        // properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.BROADCASTING)

        Consumer consumer = ONSFactory.createConsumer(properties);
        consumer.subscribe("TopicTestMQ", "TagA||TagB", new MessageListener() { //订阅多个Tag
            public Action consume(Message message, ConsumeContext context) {
                System.out.println("Receive: " + message);
                return Action.CommitMessage;
            }
        });

        //订阅另外一个Topic，如需取消订阅该Topic，请删除该部分的订阅代码，重新启动消费端即可。
        consumer.subscribe("TopicTestMQ-Other", "*", new MessageListener() { //订阅全部Tag。
            public Action consume(Message message, ConsumeContext context) {
                System.out.println("Receive: " + message);
                return Action.CommitMessage;
            }
        });

        consumer.start();
        System.out.println("Consumer Started");
    }
}
```

4.调用TCP协议的SDK消费普通消息

课程目录04

订阅关系一致最佳实践

- 订阅关系一致指的是同一个消费者 Group ID 下所有 Consumer 实例所订阅的 Topic、Group ID、Tag 必须完全一致。



正确

1. 订阅的 Topic 必须一致
2. 订阅的 Topic 中的 Tag 必须一致

错误

1. 单个Group ID订阅了多个Topic
2. Group ID里的多个消费者实例的订阅关系没有一致

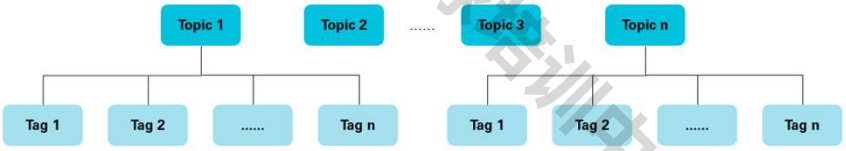
Topic & Tag 设计最佳实践

➢ Topic（一级分类）

消息主题，通过 Topic 对不同的业务消息进行分类。

➢ Tag（二级分类）

消息标签，用来进一步区分某个 Topic 下的消息分类，消息从生产者发出即带上的属性。



➢ 什么时候用Topic，什么时候用Tag做区分？

- ✓ 消息类型是否一致
- ✓ 业务是否相关联
- ✓ 消息优先级是否一致
- ✓ 消息量级是否一致

➢ 最佳实践

- ✓ 以天猫交易平台为例，订单消息和支付消息属于不同业务类型的消息，分别创建 **Topic_Order** 和 **Topic_Pay**，
- ✓ 其中订单消息根据商品品类以不同的 Tag 再进行细分，列如电器类、男装类、女装类、化妆品类等被各个不同的系统所接收。

消息幂等控制最佳实践

➢ 消息幂等

消费者对某条消息进行重复消费时，重复消费的结果和消费一次的结果是相同的

➢ 适用场景

发送时消息重复

投递时消息重复

负载均衡时消息重复

➢ 最佳实践

不要以Message ID作为处理依据，以业务唯一标识作为幂等处理的关键依据

```
Message message = new Message();
message.setKey("ORDERID_100"); SendResult
sendResult = producer.send(message);
```

① 消息的 Key 设置为订单号，作为幂等处理的依据

```
consumer.subscribe("ons_test", "*", new
MessageListener() { public Action consume(Message
message, ConsumeContext context) { String key =
message.getKey() // 根据业务唯一标识的 Key 做幂等处
理 } });
```

② 消费者收到消息时可以根据消息的 Key，即订单号来实现消息幂等

客户案例 – 众安保险

• 背景及使用情况：

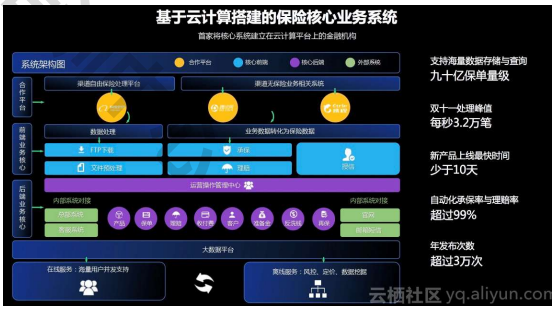
国内首家互联网保险公司，作为纯线上的互联网保险公司，IT 是支撑业务灵活快速发展的战略基础设施；传统的保险公司每日的保单量可能不会超过一万笔，但是众安保险在上线初期，在业务场景很单一的情况下（主要来源是淘宝运费险），每天的保单量在三百万至五百万件。众安保险的业务需求是小额高频场景，大家都知道运费险是一个单价非常低的保险种类，一般的保险公司不屑于或者不愿意去做这种尝试，因为技术系统要求特别高，难度也比较大；其次是要满足高并发交易，每天交易数据量非常庞大；

• 消息队列 RocketMQ 优势：

面对海量的交易，海量的数据，高性能、系统稳定是整个平台健康稳定运行的核心痛点。众安则大量的使用 MQ 进行系统间的解耦以及数据同步；

• 类似客户：

红岭创投，盒子科技、在线支付，横琴人寿



阿里云

客户案例 – 中国邮政EMS

• 背景及使用情况：

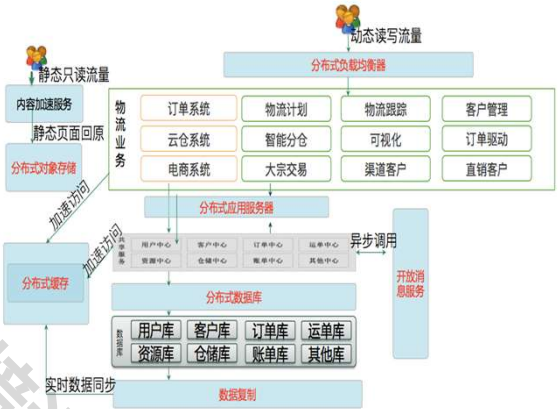
运单的状态在物流多个环节的中实时变动，产生的数据量是订单的几倍甚至十几倍，邮政每日需要上传的运单状态平均在3000万到5000万条。双十一过后，随着订单的突增，每日的运单上传接近3亿条。在这个数量级下，传统的数据采集方式无法解决数据堆积的问题，因为数据的处理速度要小于采集上传的速度，在高峰期堆积的数据极易造成流量崩溃。

• 消息队列 RocketMQ 优势：

线下数据采集平台：消息队列 MQ 具有海量消息堆积的能力，单个Topic可堆积100亿条消息，同时一份消息多份落盘存储，即使突然发生断电，也不会造成消息丢失。EMS项目采用MQ作为运单上传的媒介，系统运行至今，包括双十一期间，没有一类运单数据上传失败的问题产生。

多系统的数据中心：从线下收集的运单信息要交给多个系统，利用MQ高性能的网络传输能力，保障业务消息可以被多方同时消费。

全业务流程监控：借助业务上的设计及MQ自身的监控能力，MQ成为EMS系统中全业务流程监控的关键点



• 类似客户：

中石化，中石油、中化工，中国联通，人社部，国家税务

阿里云

客户案例 – 民生银行

• 背景及使用情况：

支持民生银行互联网银行核心技术改造，两地双中心创新型容灾方案建设。在消息系统方面，客户可采用的 ActiveMQ 在性能、可用性以及数据可靠性上都遇到了瓶颈，尤其在数据可靠性方面， KahaDB只支持单副本，目前KahaDB主要依赖EMC完成底层数据存储的复制；

• 消息队列 RocketMQ 优势：

多渠道消息通知：用户账号的动态变更变更的实时通知，会对接各种外围系统和通讯软件，外围系统可用性与稳定性相对较为不稳定，如短信、微信、银行App等；利用消息队列 MQ pub/sub 模型，提高对接各种外围系统和通讯软件的稳定性以及可扩展性；

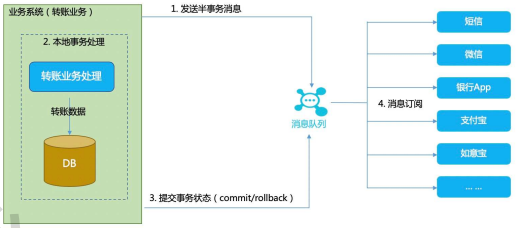
分布式事务：引入消息队列 MQ 的分布式事务消息，简化业务系统中事务处理环节的复杂度，减少对DB的操作，降低对DB带来的压力；

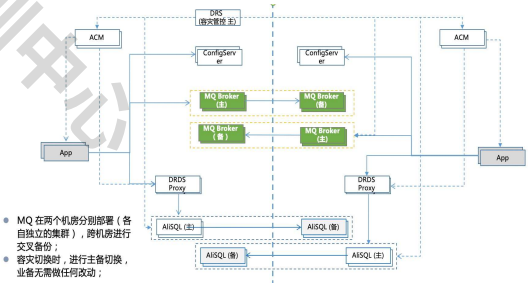
双核心系统数据实时同步：民生银行新系统（DCBC）、老系统（ BPP ）之间，客户号变更信息同步；

同城双活：支持民生银行异地双活系统构建；

• 类似客户：

平安银行、浙商银行、广发银行、广东农信，太平人寿





阿里云

总结及回顾

本章节主要讲述内容：

1.消息队列的应用场景，异步解耦，分布式事务一致性，削峰填谷等。

2.阿里云RocketMQ的功能特性

- 消息收发模型
- 消息类型，包括普通消息，顺序消息，分布式消息，定时&演示消息
- RocketMQ消息运维相关，消息轨迹查询，消费查询，监控告警等

3.阿里云RocketMQ的基本使用步骤

从创建实例与资源，到Topic和GroupID的创建，通过接入SDK演示程序侧使用方式

4.阿里云RocketMQ最佳实践与案例

阿里云

课程目标

学习完本课程后，你将能够：

1. 了解阿里云消息队列产品Kafka
2. 掌握消息队列Kafka产品功能特性和基本使用

课程目录01

1. 消息队列Kafka的概述

1.1应用场景介绍

1.2应用生态与产品优势

2.消息队列Kafka的功能特性

3.消息队列Kafka的基本使用

4.消息队列Kafka的最佳实践

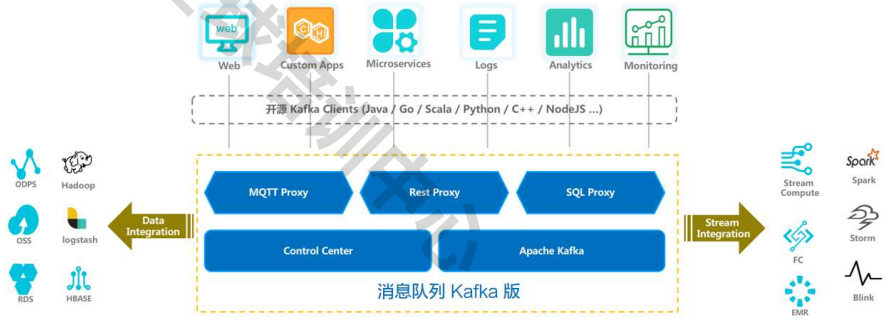
Kafka的应用场景

消息队列Kafka是整个消息引擎领域的执牛耳者，在实际业务系统中实现消息引擎应用、应用程序集成、流处理应用的开发和部署都可以依赖Kafka实现

- 大数据领域
网站行为分析、日志聚合、应用监控、流式数据处理、在线和离线数据分析等领域

- 数据集成
将消息导入MaxCompute、OSS、RDS、Hadoop、HBase等离线数据仓库

- 流计算集成
与StreamCompute、E-MapReduce、Spark、Storm等流计算引擎集成

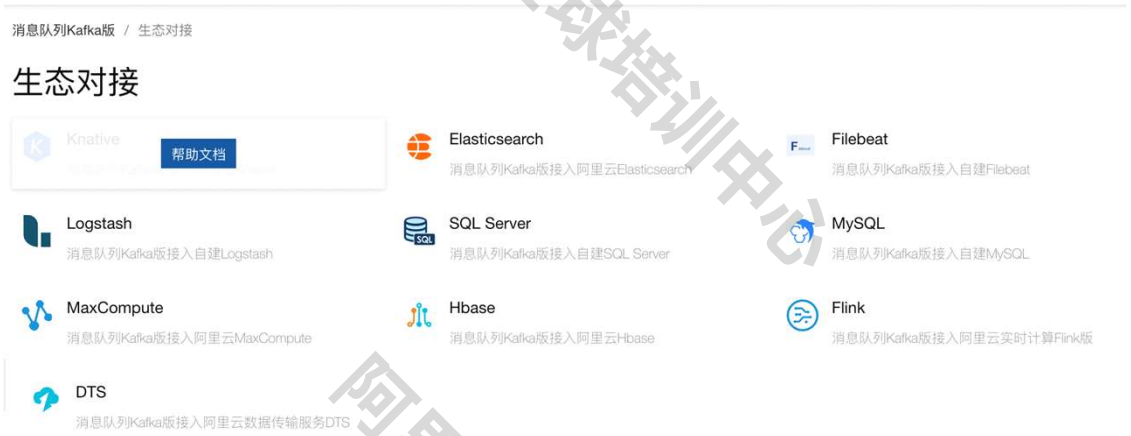


Kafka的应用场景

消息中间件	跟踪网站活动	数据监控	日志聚合	提交日志	增量数据捕捉
在应用系统中，实现消息队列和消息的发布/订阅。Kafka拥有更好的吞吐量、内置分区、具有复制和容错的功能，大型消息处理应用。	LindIn公司最初的Kafka使用场景。发布到特定的topic，订阅topic，用以做数据的实时处理、实时监控、加载到离线数据仓库做离线分析等。	分布式应用中，相关的监控数据汇集到Kafka中，生成可操作的集中数据源，供相关系统使用。	日志收集中心，将位于不同服务器的物理日志文件收集起来，并抽象为一个更加清晰的消息流，然后由数据分析平台统一处理。	从外部为分布式系统提供日志提交功能。日志有助于记录节点和行为间的数据，采用重新同步机制可以从失败节点恢复数据。	将数据库增量变化数据，按照发生的顺序完整地记录到Kafka，这些增量数据可以后续用于异构数据备份、微服务间状态共享、更新缓存以及事件监控等场景。

Kafka的应用生态

消息队列 Kafka 针对 **Apache Kafka 提供全托管服务**，兼容 Apache Kafka 的生态，彻底解决开源产品长期以来的痛点，是**大数据生态**中不可或缺的产品之一。对底层技术实现对开源做了优化，用户只需专注于业务开发，**无需运维，更便捷、更稳定，更专业、更可靠**。



Kafka的产品优势

全托管服务
提供全托管服务，提供SLA 兜底，用户只需专注于业务开发，无需部署运维，更专业、更弹性、更可靠

易用性
业务代码无需任何改造，可快速上云；提供意丰富Demo；提供完善的 GUI 驱动运维管理

高性价比
入门门槛低于自建；提供标准版、企业版，高写、高读等多种模式选择

高弹性
提供磁盘、任意集群规格处理能力秒级别扩容，扩容期间对业务几乎无影响

大规格
集群处理能力达到2GB/s 的业务数据规模，支持万级别Topic，十几万级别分区

生态支持
自动支持阿里云上的大数据产品接入，包括MaxCompute、OSS、FC等，并提供存储Hbase、搜索Elasticsearch、数据库MySQL，计算Spark、实时引擎Flink的等一站式的解决方案。

Kafka的产品优势

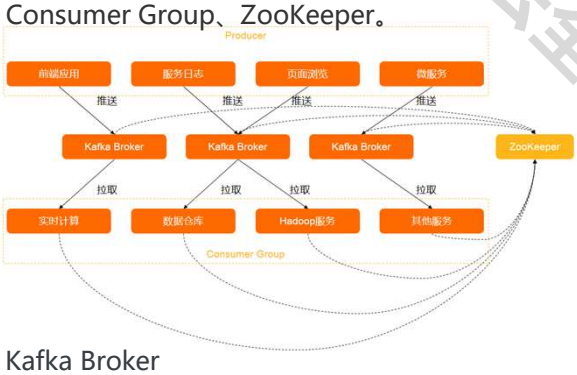
	功能	消息队列 Kafka	Apache Kafka
治理能力	版本升级	一键自助升级	手工操作易出错
	Metrics 曲线	能看到完整 Metrics 曲线，追踪流量、排查问题必备	只能看到实时 Metrics，历史数据较难查看
	堆积告警	告警及时发现问题	无
	订阅关系	完整的订阅关系	比较简略
	分区状态	可以看到完整的状态图	比较简略
	发送消息	控制台直接发送消息	只能命令行操作，成本高
	查询消息	控制台根据时间或者位点直接查看消息	命令行可以消费，但无法根据位点或者时间直接定位到具体的消息
稳定性保证	磁盘水位	磁盘写满删除旧数据	磁盘写满直接宕机
	线程池隔离	读冷数据仍可以保证写入基本正常	读冷数据直接导致线程堵塞，数据写入大量失败
	巡检系统	针对死锁、宕机等问题进行自动发现和修复	无
	Bug 修复	及时发现并修复	只能等社区缓慢修复，且通常要等新版发布，周期长
	同城容灾	支持指定多个可用区部署	无
内核优化	弹性能力	秒级弹缩，业务几乎无感知	小时级弹缩，期间会因为复制流量加大，对集群造成影响
	存储成本	专业版提供高可靠云存储，节省大量存储空间	出于可用性和可靠性考虑，业界通常都是 3 副本存储，存储压力大
	分区规模	万级分区仍然可以保证稳定写入	千级分区就会出现大量抖动

课程目录02

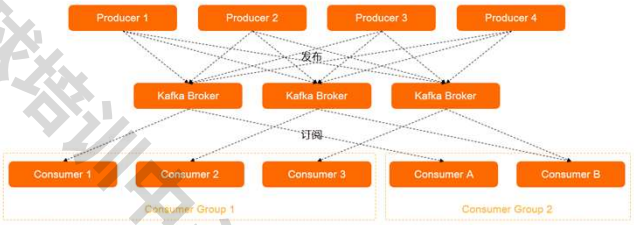
阿里云全球培训中心

产品部署结构

- 消息队列Kafka版集群包括Producer、Kafka Broker、Consumer Group、ZooKeeper。



- 消息队列Kafka版的发布/订阅模型



✓ Kafka Broker

支持水平扩展，节点越多，集群的吞吐率越高

✓ Zookeeper

管理集群的配置，选举leader，在Consumer Group发生变化时，进行负载

✓ Consumer Group和Topic关系 N:N

✓ Topic只能被同一个Consumer Group内的任意一个Consumer消费

关键名词解释

- **Topic**

  特指kafka处理的消息源的不同分类

- **Partition**

  Topic物理上的分组，一个topic可以分为多个partition，每个partition是一个有序的队列。

  partition中的每条消息都会被一个有序的id(offset)

- **Message**

  消息，是通信的基本单位，每个producer可以向一个topic发布一些消息

- **Producers**

  消息和数据生产者，向kafka的一个topic发布消息的过程叫做producers

- **Consumers**

  消息和数据消费者，订阅topics并处理其发布的消息的过程叫做consumers

- **Broker**

  缓存代理，kafka集群中的一台或多台服务器统称为broker

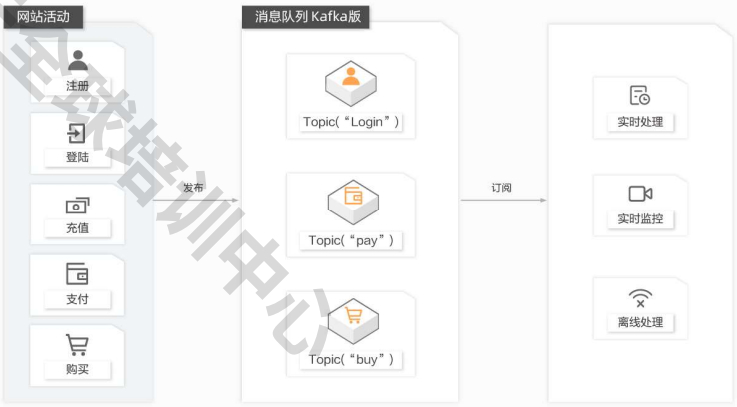
- **接入点**

  Producer或Consumer连接消息队列Kafka版时使用的地址，由Broker的URI和端口号拼接而成。

网站活动跟踪

通过消息队列 Kafka 版，可以实时收集网站活动数据（包括用户浏览页面、搜索及其他行为等），并通过“发布/订阅”模型实现：

- ✓ 根据不同的业务数据类型，将消息发布到不同的 Topic
- ✓ 通过订阅消息的实时投递，将消息流用于实时监控与业务分析或者加载到 Hadoop、ODPS 等离线数据仓库系统进行离线处理与业务报告。



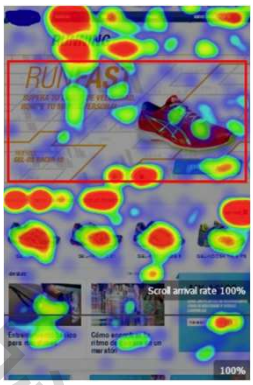
网站活动跟踪案例

业务场景举例

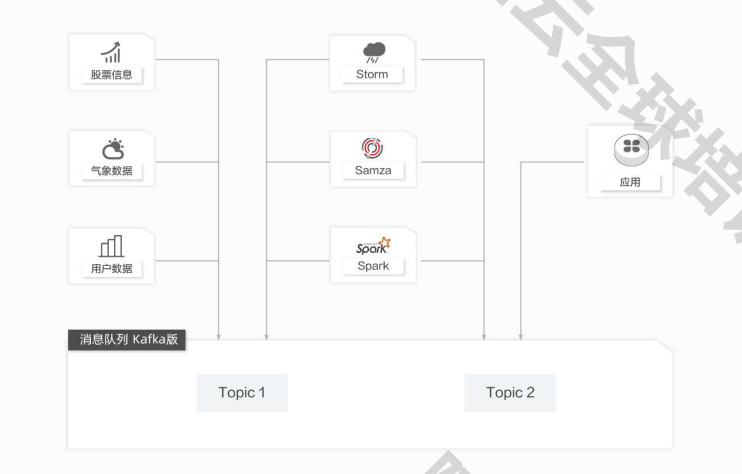
- 用户行为分析（用户喜好、停留、热点区域）
- 改善页面布局，照片展示，用户打标

背后技术实现

通过消息队列Kafka可收集活动网站上的日志，用于页面分析、用户打标和业务调整



流计算处理



➢ 流计算处理场景

1.数据产生快，实时性强且量大

2.流计算模型能实现在数据流动的过程中对数据进行实时捕获和处理

➢ 处理优势

流动的数据：构建应用系统和分析系统的桥梁，并将它们之间的关联解耦

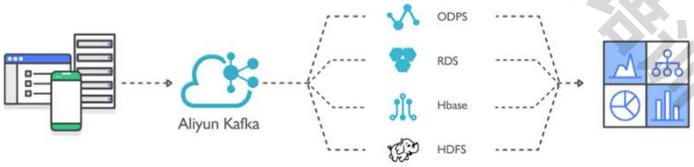
高可扩展性：由于数据产生非常快且数据量大，需要非常高的可扩展性；

流计算引擎：可对接开源 Storm/Samza/Spark 以及 EMR、Blink、StreamCompute 等阿里云产品。

流计算处理案例

业务场景：适合实时类数据监控、展示等场景。如业务大盘，销售大盘，系统运行大盘等

技术实现：用Kafka收集数据流数据，并将其加载到云数据存储中



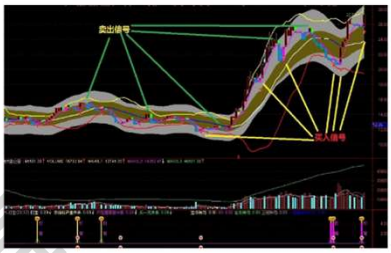
Step1：业务数据写入Kafka

Step2：消息数据流入到离线存储，数据库等

Step3：BI处理，批量数据分析

流计算处理案例

➢ **业务场景**：适合各类实施业务分析、预测、决策等场景，如实时路况+导航、台风路径预测、股票走势预测等

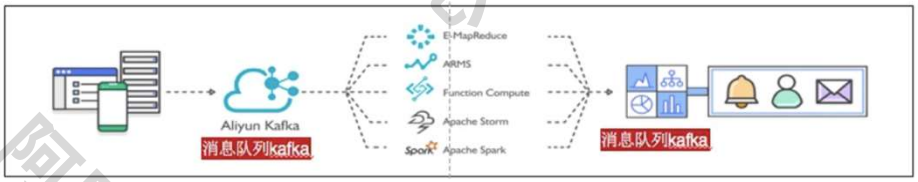


➢ **技术实现**：通过Kafka+流计算引擎，构建实时应用程序

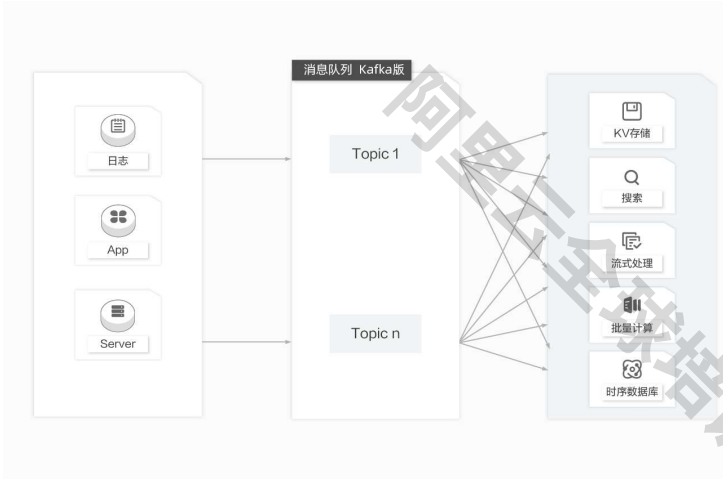
Step1：各类网站点击流、金融交易和应用程序日志等流数据写入到Kafka进行缓存

Step2：消息数据流入到实时计算引擎（如EMR、Storm、Spark），以及Function Compute等进行实时业务分析、预测

Step3：用户分析、运营平台、Alert、Monitor等进行分析，发起决策和指挥



数据中转枢纽



➢ **数据中转枢纽场景**

1.原始数据需要被投递到各个系统中使用

2.KV存储，搜索，流失处理，时序数据库订阅Kafka消息，起到中转枢纽作用

➢ **处理优势**

- **高容量存储**

  能在商业硬件上存储高容量的数据，实现可横向扩展的分布式系统

- **一对多消费模型**

  “发布/订阅”模型，支持同份数据集能同时被消费多次

- **同时支持实时和批处理**：

  支持本地数据持久化和 Page Cache，在无性能损耗的情况下能同时传送消息到实时和批处理的消费者。

数据中转枢纽案例

➢ **业务场景**：大数据的部分数据中心，适合运用于

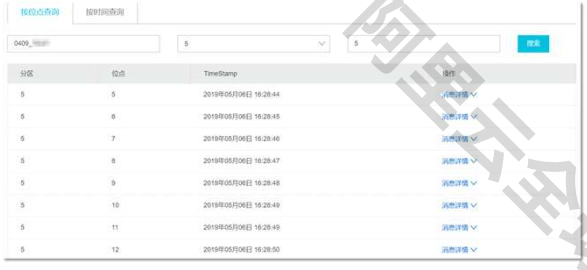
用户画像库：生活圈、社交关系、消费层次等标签画像库

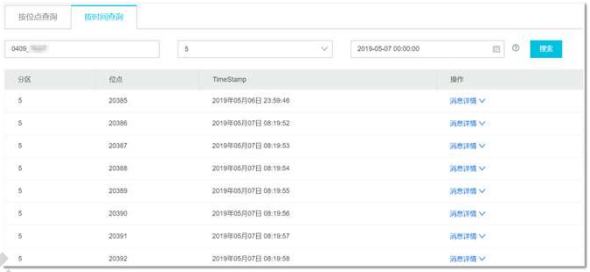
客户服务：用户使用问题定位，故障排查等

➢ **技术实现**：用Kafka收集网站系统等海量日志，并将其加载到云数据存储中，成为日志收集中心

Kafka消息运维

➢ **消息运维**



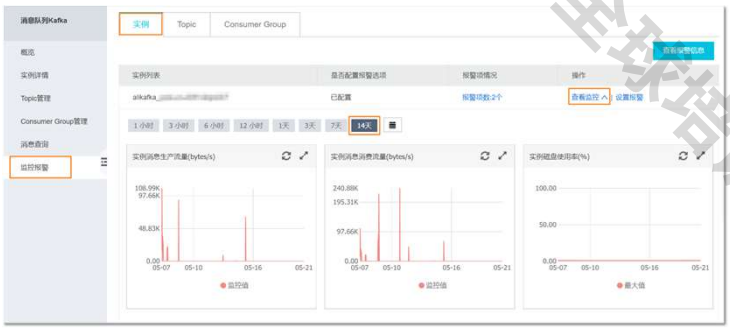


• **按位点查询**：一个位点指向一条消息，在确定需要查询的消息所在位置的情况下，可以指定位点查询相应的消息。

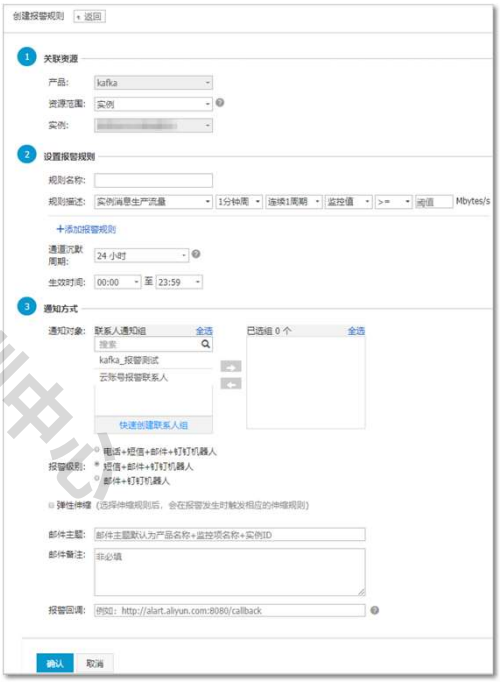
• **按时间查询**：不确定消息的位置，但确定消息发送的时间段，可以指定该时间段中的某一个时间点查询该时间点附近的消息。

Kafka消息运维

- 控制台查看监控信息



- 设定告警规则



Kafka的幂等性功能

- 幂等性的生产者

  创造一个幂等的Producer很简单，只需要设置一个参数即可(0.11.0.0之后的版本)，设置之后，producer自动升级为幂等的producer，kafka会自动做消息的去重。

```
//设置幂等
props.put(ProducerConfig.ENABLE_IDEMPOTENCE_CONFIG,true);
```

- 幂等性的作用范围

  首先，只能保证单分区的幂等性，即一个幂等的producer能够保证某个主题的一个分区不出现重复消息，无法实现多个分区的幂等性。

  其次，只能实现单会话的幂等性。会话，可以理解为producer进程的一次运行，当重启producer进程后，幂等性就丧失了。

事务性生产者

- 事务性的生产者
  kafka的事务，主要是在read committed隔离级别上做事情。可以保证多条消息原子性地写入到目标分区，同时保证consumer智能看到事务成功提交的消息。

- 事务型的producer能够保证消息原子性地写入到多个分区中。这批消息要么全部写入成功，要么全部失败。

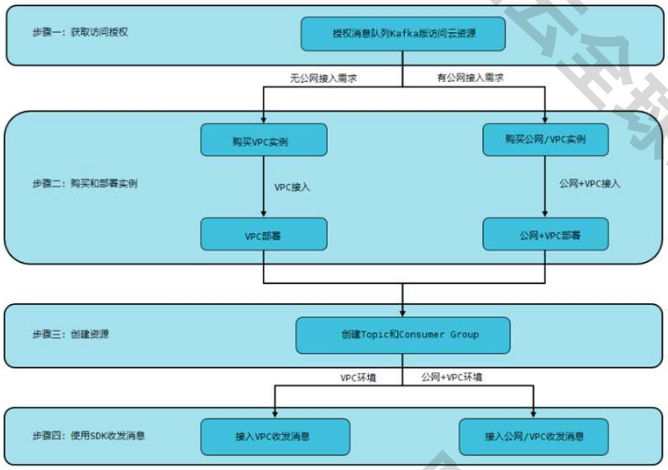
- 事务型的producer不被进程重启影响，重启后，kafka依然保证发送消息的精确一次处理。

```
//设置事务性消息
props.put(ProducerConfig.ENABLE_IDEMPOTENCE_CONFIG,true);
//设置transctional.id,为其设置一个有意义的名字
props.put(ProducerConfig.TRANSACTIONAL_ID_CONFIG,"*****");
```

```
producer.initTransactions();

try {
    producer.beginTransaction();
    //批量获取 futures 可以加快速度, 但注意, 批量不要太大
    List<Future<RecordMetadata>> futures = new ArrayList<Future<RecordMetadata>>( initialCapacity: 128);
    for (int i =0; i < 100; i++) {
        //发送消息, 并获得一个Future对象
        ProducerRecord<String, String> kafkaMessage =  new ProducerRecord<>(topic,  value: value + ": " + i);
        Future<RecordMetadata> metadataFuture = producer.send(kafkaMessage);
        futures.add(metadataFuture);

    }
    producer.flush();
    producer.commitTransaction();
    for (Future<RecordMetadata> future: futures) {
        //同步获得Future对象的结果
        try {
            RecordMetadata recordMetadata = future.get();
            System.out.println("Produce ok:" + recordMetadata.toString());
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }

} catch (Exception e) {
    //客户端内部重试之后, 仍然发送失败, 业务要应对此类错误
    //参考常见报错: https://help.aliyun.com/document_detail/68168.html?spm=a2c4g.11186623.6.567.2OMgCB
    System.out.println("error occurred");
    e.printStackTrace();
}finally {
    producer.abortTransaction();
}
```


课程目录03

完整使用流程概述



- 基本使用流程四步曲
  - 步骤1： 获取访问授权
  - 步骤2： 购买和部署实例
  - 步骤3： 创建资源
  - 步骤4： 使用SDK收发消息

- 注意事项
  1. 实例接入点有两种方式

  默认接入点支持VPC网络使用，SSL接入点支持公网接入

获取访问授权

第一步：RAM权限授予，需要赋予AliyunKafkaDefaultRole的权限角色

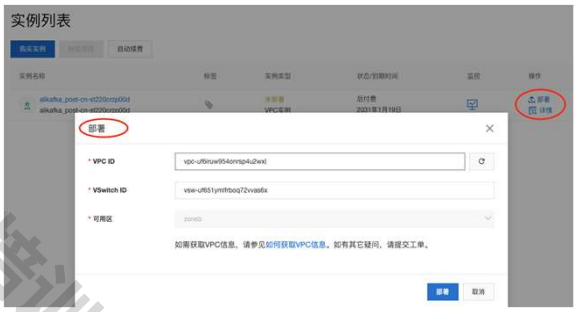
购买和部署实例

➢ 购买实例类型

1.实例规格

2.磁盘类型和容量

3.消息保留最长时间



➢ 部署实例

1.VPC接入

购买好的Kafka实例规格需要部署到指定的VPC中

2.公网+VPC接入

在部署实例过程中，多了一个设置用户名和密码的步骤

实例部署预计10分钟~30分钟

创建Topic与Consumer Group

➢ 创建Topic

标准版填写名称和分区数即可

专业版可以选择存储引擎和消息类型

1.存储引擎类型包括云存储和Local存储两种

2.消息类型包括普通消息和分区顺序消息两种

➢ 创建Consumer Group

1.名称和备注都是必填项

2.可以增加标签做区分和标记

使用SDK收发消息

- Kafka的Demo和SDK使用，在界面上提供了大量的接入案例，包括

1.消息对接Kafka接入Logstash

2.通过MirrorMaker迁移集群数据

3.Spring框架客户端库

使用SDK收发消息

- 准备配置

1.创建Log4j配置文件

```
log4j.rootLogger=INFO, STDOUT

log4j.appender.STDOUT=org.apache.log4j.ConsoleAppender
log4j.appender.STDOUT.layout=org.apache.log4j.PatternLayout
log4j.appender.STDOUT.layout.ConversionPattern=[%d] %p %m (%c)%n
```

2.创建Kafka配置文件

```
## 配置接入点，即控制台的实例详情页面显示的默认接入点。
bootstrap.servers=xxxxxxxxxxxxxxxxxxxx
## 配置Topic，可以在控制台上创建Topic。
topic=alikafka-topic-demo
## 配置Consumer Group，可以在控制台创建Consumer Group。
group.id=CID-consumer-group-demo
```

3.创建配置文件加载程序

```
public class JavaKafkaConfigurer {
    private static Properties properties;
    public synchronized static Properties getKafkaProperties() {
       if (null != properties) {
           return properties;
       }
       //获取配置文件kafka.properties的内容。
       Properties kafkaProperties = new Properties();
       try {
           kafkaProperties.load(KafkaProducerDemo.class.getClassLoader().getResourceAs
       } catch (Exception e) {
           //没加载到文件，程序要考虑退出。
           e.printStackTrace();
       }
       properties = kafkaProperties;
       return kafkaProperties;
    }
}
```

使用SDK收发消息

➢ 发送消息

1.初始化发送者参数配置

```
public class KafkaProducerDemo {

    public static void main(String args[]) {
        //加载kafka.properties。
        Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaProperties();

        Properties props = new Properties();
        //设置接入点，请通过控制台获取对应Topic的接入点。
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty(

        //Kafka消息的序列化方式。
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, "org.apache.kafka.commo
        //请求的最长等待时间。
        props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
        //设置客户端内部重试次数。
        props.put(ProducerConfig.RETRIES_CONFIG, 5);
        //设置客户端内部重试间隔。
        props.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
        //构造Producer对象，注意，该对象是线程安全的，一般来说，一个进程内一个Producer对象即可。
        //如果想提高性能，可以多构造几个对象，但不要太多，最好不要超过5个。
        KafkaProducer<String, String> producer = new KafkaProducer<String, String>(prop
```

2.Producer发送消息

```
//构造一个Kafka消息。
String topic = kafkaProperties.getProperty("topic"); //消息所属的Topic，请在控制台申请之后，
String value = "this is the message's value"; //消息的内容。

try {
    //批量获取Future对象可以加快速度，但注意，批量不要太大。
    List<Future<RecordMetadata>> futures = new ArrayList<Future<RecordMetadata>>(128);
    for (int i =0; i < 100; i++) {
        //发送消息，并获得一个Future对象。
        ProducerRecord<String, String> kafkaMessage =  new ProducerRecord<String, Str
        Future<RecordMetadata> metadataFuture = producer.send(kafkaMessage);
        futures.add(metadataFuture);

    }
    producer.flush();
    for (Future<RecordMetadata> future: futures) {
        //同步获得Future对象的结果。
        try {
            RecordMetadata recordMetadata = future.get();
            System.out.println("Produce ok:" + recordMetadata.toString());
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }
} catch (Exception e) {
    //客户端内部重试之后，仍然发送失败，业务要应对此类错误。
    System.out.println("error occurred");
    e.printStackTrace();
}
```


使用SDK收发消息

➢ 消费消息

1.初始化Consumer参数配置和Topic消费关系

```
public class KafkaConsumerDemo {

    public static void main(String args[]) {
        //加载kafka.properties。
        Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaProperties();

        Properties props = new Properties();
        //设置接入点，请通过控制台获取对应Topic的接入点。
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getPropert
        //两次Poll之间的最大允许间隔。
        //消费者超过该值没有返回心跳，服务端判断消费者处于非存活状态，服务端将消费者从Consumer Gro
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
        //每次Poll的最大数量。
        //注意该值不要改得太大，如果Poll太多数据，而不能在下次Poll之前消费完，则会触发一次负载均衡
        props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
        //消息的反序列化方式。
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.com
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.c
        //当前消费实例所属的消费组，请在控制台申请之后填写。
        //属于同一个组的消费实例，会负载消费消息。
        props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.
        //构造消息对象，也即生成一个消费实例。
        KafkaConsumer<String, String> consumer = new org.apache.kafka.clients.consume
        //设置消费组订阅的Topic，可以订阅多个。
        //如果GROUP_ID_CONFIG是一样，则订阅的Topic也建议设置成一样。
        List<String> subscribedTopics =  new ArrayList<String>();
        //如果需要订阅多个Topic，则在这里添加进去即可。
        //每个Topic需要先在控制台进行创建。
        String topicStr = kafkaProperties.getProperty("topic");
        String[] topics = topicStr.split(",");
        for (String topic: topics) {
            subscribedTopics.add(topic.trim());
        }
        consumer.subscribe(subscribedTopics);
```

2.循环消费消息

```
//循环消费消息。
while (true){
    try {
        ConsumerRecords<String, String> records = consumer.poll(1000);
        //必须在下次Poll之前消费完这些数据，且总耗时不得超过SESSION_TIMEOUT_MS_CONFIG。
        //建议开一个单独的线程池来消费消息，然后异步返回结果。
        for (ConsumerRecord<String, String> record : records) {
            System.out.println(String.format("Consume partition:%d offset:%d", record.
        }
    } catch (Exception e) {
        try {
            Thread.sleep(1000);
        } catch (Throwable ignore) {

        }
        e.printStackTrace();
    }
}
```


阿里云全球培训中心

课程目录04

1.消息队列Kafka的概述

2.消息队列Kafka的功能特性

3.消息队列Kafka的基本使用

4.消息队列Kafka的最佳实践

4.1 发布者最佳实践

4.2 订阅者最佳实践

4.3 消息对接迁移实践

发布者最佳实践

➢ **失败重试**

消息对接Kafka版是VIP网路结构，会主动断开空闲连接控制参数：

- retries重试次数
- retry.backoff.ms重试间隔

➢ **Acks实践说明**

- acks=0：性能较高、丢数据风险较大。
- acks=1：性能中等、丢数据风险中等。
- acks=all：性能较差、数据较为安全

一般建议选择acks=1，重要的服务可以设置acks=all

➢ **Batch提高吞吐**

- batch.size：发往每个分区（Partition）的消息缓存量（消息内容的字节数之和
- linger.ms：每条消息待在缓存中的最长时间

➢ **OOM**

- buffer.memory 默认大小32MB，超过数值消息发往服务器。
- 消息生产，没必要启用多个Producer避免触发OOM

订阅者最佳实践

➢ 消息分区数量

- 消费实例数量不要大于分区数量
  订阅Topic的分区，平均分配给每个消费实例
- 默认分区个数12，分区增加后不能减少，建议小幅调整

```
String topicStr = kafkaProperties.getProperty("topic");
String[] topics = topicStr.split(",");
for (String topic: topics) {
subscribedTopics.add(topic.trim());
}
consumer.subscribe(subscribedTopics);
```

➢ Consumer Group订阅多个Topic均匀消费

- Topic被多个Consumer Group订阅独立消费，相互独立

➢ 消息位点重置

Java客户端可以通过auto.offset.reset来配置重置策略，主要有三种策略：

latest：从最大位点开始消费

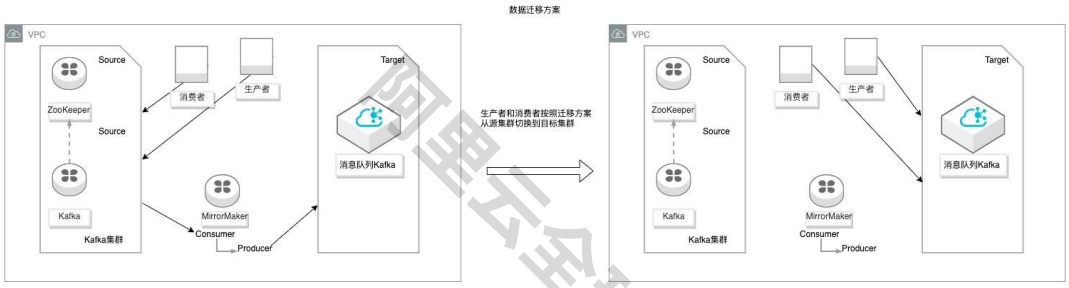
earliest：从最小位点开始消费

none：不做任何操作，即不重置

建议设置成latest，如果是自己管理位点，可以设置成none

在线迁移至Kafka

➢ 迁移示意图



➢ 推荐通用迁移方案

1.迁移Metadata(Topic和Consumer Group)

2.开启消息队列Kafka的消费者50%

3.生产者切换到目标消息队列

4.源Kafka堆积消费完成，消费者整体切换到目标Kafka

5.下线源Kafka集群

不建议通过MirroMaker做数据的迁移

总结及回顾

本章节主要讲述内容：

1.消息队列Kafka的应用场景，完整的应用生态

2.消息队列Kafka的功能特性

- 产品部署结构
- 网站活动跟踪，流计算处理和数据中转枢纽的三个场景
- Kafka消息运维

3.消息队列Kafka的基本使用步骤

从创建实例与资源，到Topic和Consumer Group的创建，通过接入SDK演示程序侧使用方式

4.消息队列Kafka最佳实践

发布者和订阅者的最佳实践，Kafka在线数据迁移

阿里云云原生微服务ACP认证培训

应用实时监控ARMS

阿里云

课程目标

学习完本课程后，你将能够：

1. 了解阿里云应用实时监控ARMS产品
2. 掌握ARMS功能特性和基本技术原理

阿里云

课程目录01

企业IT监控需求

企业IT覆盖度愈发广泛，IT环境日益复杂，企业对IT监控需求增加

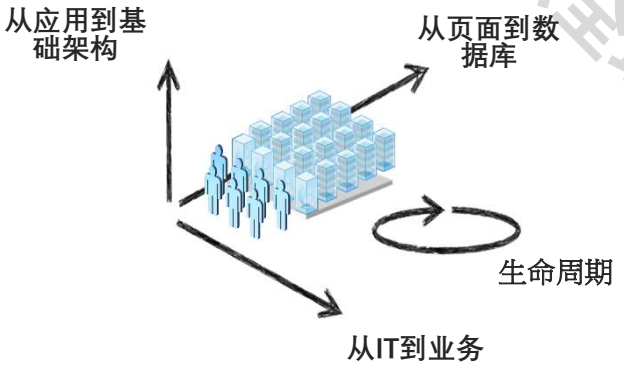
- 单个业务操作需要跨越多IT系统

- 多终端的访问设备

- 新的应用开发架构与技术，创造了IT系统的多样性

- 新的应用交付模式，让IT资产分布多样化

智能运维从监控做起



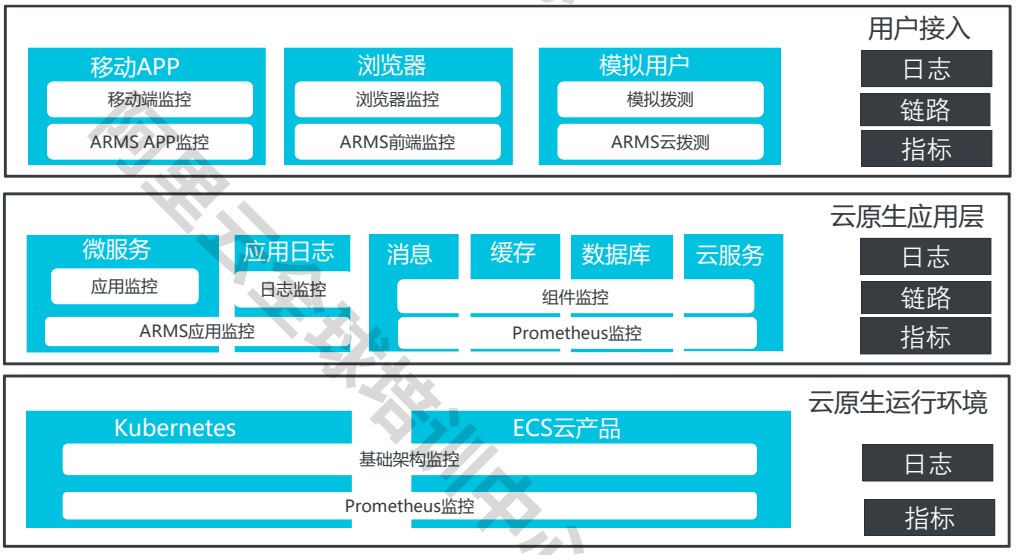
Step 3：智能运维
利用大数据和人工智能技术，解决运维各个环节的效率问题，全面提升IT运维及业务运营管理质量

Step 2：业务运维
从业务视角实时感知业务及系统运行状态，实现业务和IT的双向驱动，确保业务连续性，持续提升业务效能

Step 1：大数据运维
面向企业IT，基于大数据技术建立一体化监控平台及数据应用体系

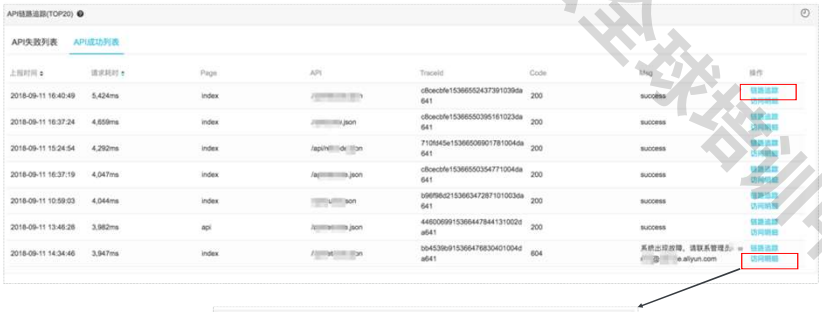
立体监控体系

- 用户接入层
  APP监控，前端监控，云拨测
- 云原生接入层
  应用监控，Prometheus监控
- 基础设施层
  云监控，基础架构监控

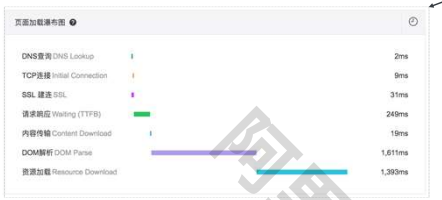


从页面到数据库的监控

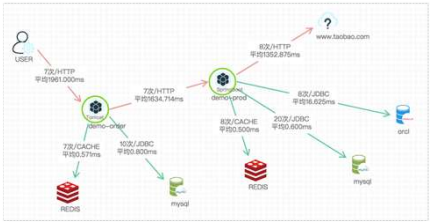
页面执行监控



页面加载监控

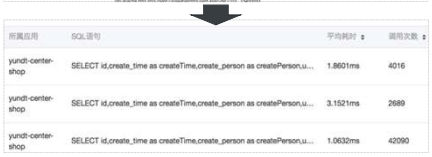


调用链监控



慢服务监控

慢SQL监控



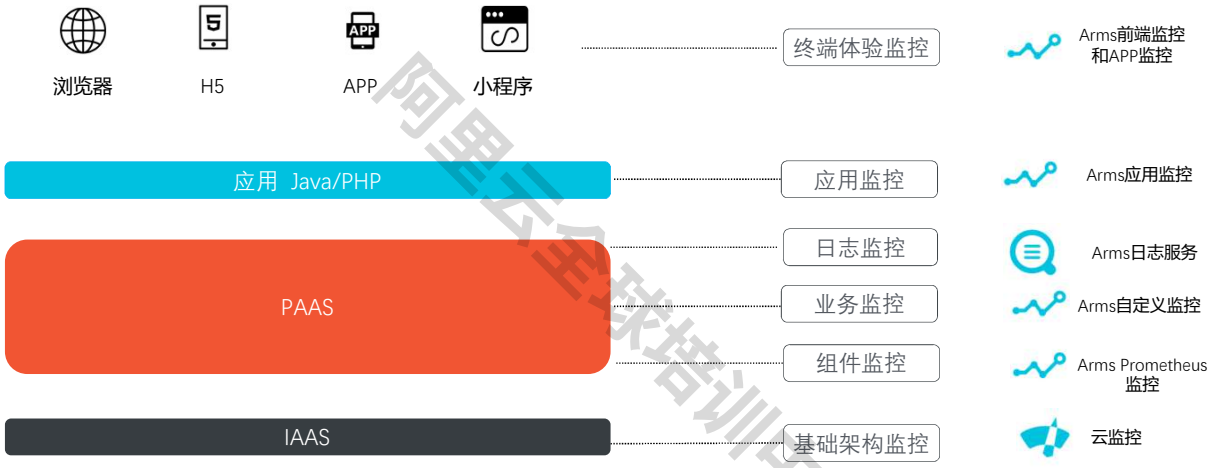
从应用到基础架构的监控

➢ 从应用的角度拉通JVM、主机、数据库及周边组件的统一监控视图

从技术到业务的监控

- 能提供业务调用链路的回溯功能，排查业务链上的错误
- 提供自定义监控功能，可以监控比如：新用户或订单等数据的同比增长、环比增长；

全栈监控构成



课程目录02

ARMS功能模块概览

- 本课程重点覆盖内容
- App监控
- 应用监控
- 前端监控
- Prometheus监控
- 容器监控
- 业务监控

应用监控

面向分布式架构，监控Java/PHP应用，查看应用TOP、接口调用、异常事务、慢事务等

前端监控

从页面打开速度、稳定性和外部调用成功率角度监测Web页面和小程序的健康度

Prometheus监控

全面对接开源Prometheus生态，支持丰富的组件监控

APP监控

专注监控移动设备上的应用性能和用户体验，涵盖崩溃分析、性能分析和远程日志

容器监控

面向部署在kubernetes的集群，监控节点和容器的实时数据，提供性能数据可视化

业务监控

从业务视角衡量应用性能和稳定性，实现对业务全链路监控，提供丰富性能指标和诊断能力

APP监控——ARMS移动监控

APP 线上问题痛点

感知难　分析难　定位难

解决方案：APP监控

- 【采集】构建端到云体验数据采集体系。
- 【度量】建立用户可用性度量标准。
- 【分析】通过多维分析、智能算法模型识别问题特征。
- 【定位】拉取本地全量日志，精准分析定位代码问题。

自动感知 → 智能分析 → 快速定位

10S　自动告警　智能分析　全量日志

10S 内发现问题

多种告警方式
支持短信、邮件
站内信、阿里云APP

智能分析问题
全面的度量体系、影响用户体量
问题特征自动识别、关联问题分析

全量日志还原问题现场
动态拉取全量端上日志，还原
问题现场，快速定位问题

APP监控使用场景

场景1：发现崩溃问题

Crash和异常错误监控和及时告警

崩溃告警 ← 超过阈值 ← 聚合分析 ← 日志上报 ← APP

场景2：发现体验问题

启动、页面卡顿、页面加载慢等问题度量和监控

启动慢、卡顿 ← 问题度量 ← 聚合分析 ← 日志上报 ← APP

场景3：定位线上问题

拉取全量日志，快速定位问题，降低MTTR

精准定位问题 ← 分析日志 ← 还原现场 ← 拉取全量日志 ← APP

性能分析

- 无痕埋点的方式，开发接入成本极低
- 精确度高，与用户直接感受接近
- 通用性强，线下线上都适用

崩溃分析

•问题分析

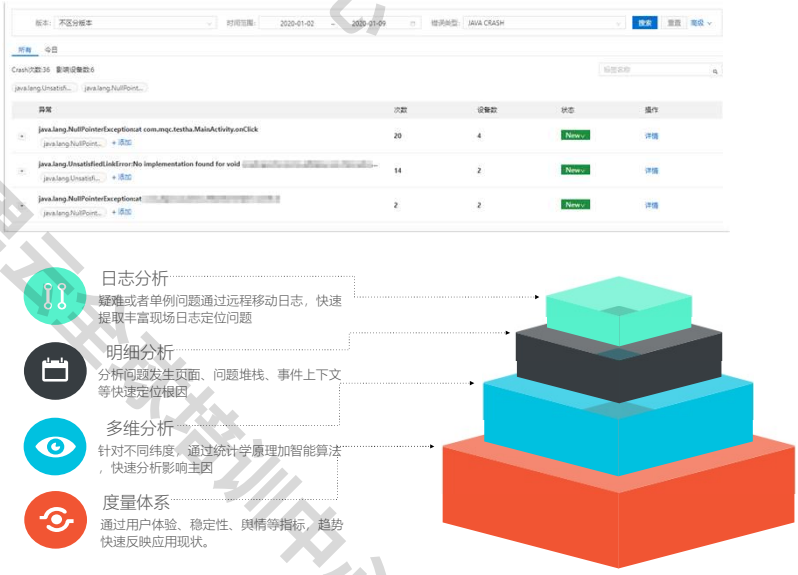
以折线图展示该条异常在不同日期内发生次数的对比。

•崩溃列表

展示异常的基本信息、崩溃堆栈、内存信息、存储信息和Consolelog。

•关联聚合

列出关联异常、异常状态和可执行的操作。



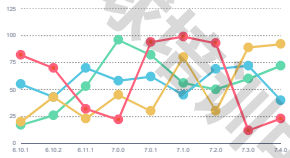
远程日志

远程日志主要针对App上线后的问题分析和定位：

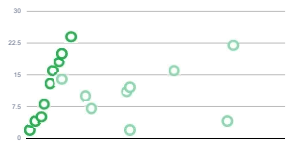
•全量拉取移动终端的异常日志，还原现场，快速定位复杂问题。

•单独客户异常问题，快速拉取日志进行排查

性能对比分析

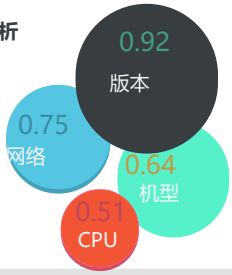


性能长尾分析

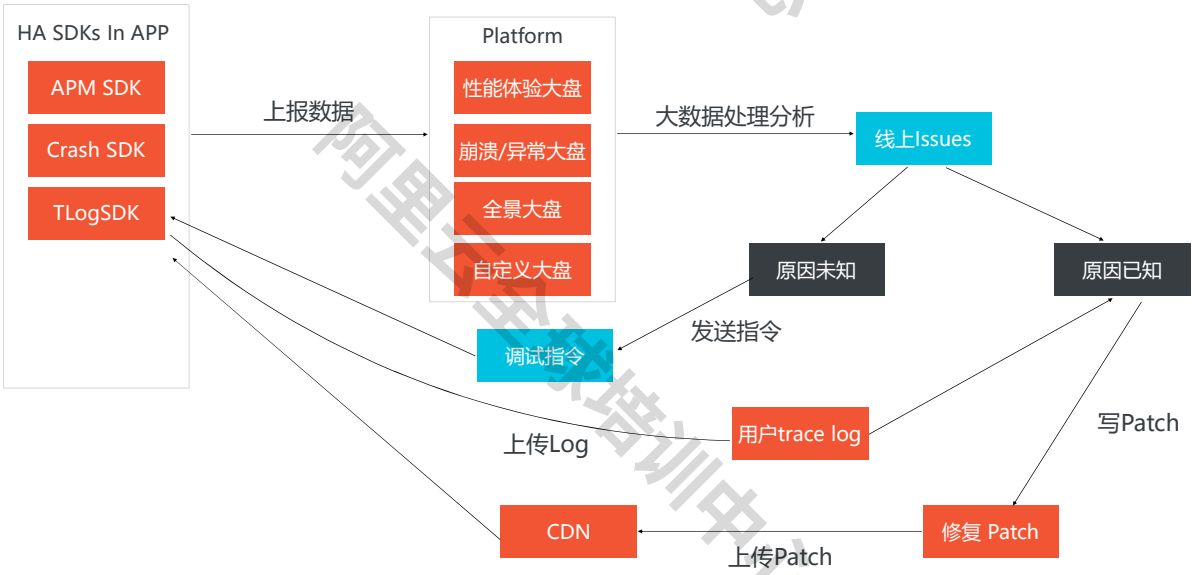


性能指标看板

相关度分析

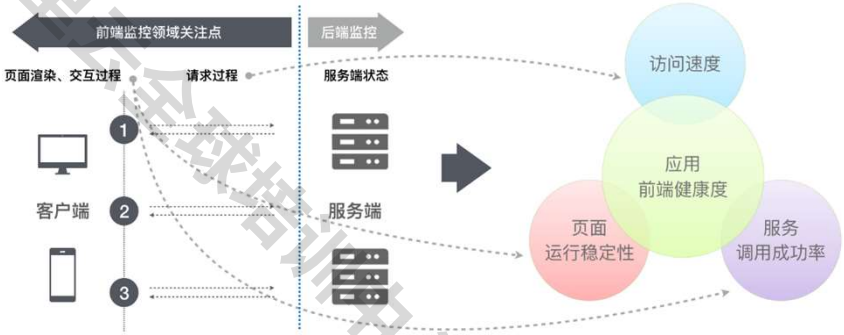


APP监控接入过程



前端监控——ARMS前端监控

- ARMS前端监控专注于对Web场景、Weex场景和小程序场景的监控，从以下三个方面监控
  - 页面访问速度
  - 页面稳定性（JS 错误诊断）
  - API请求监控



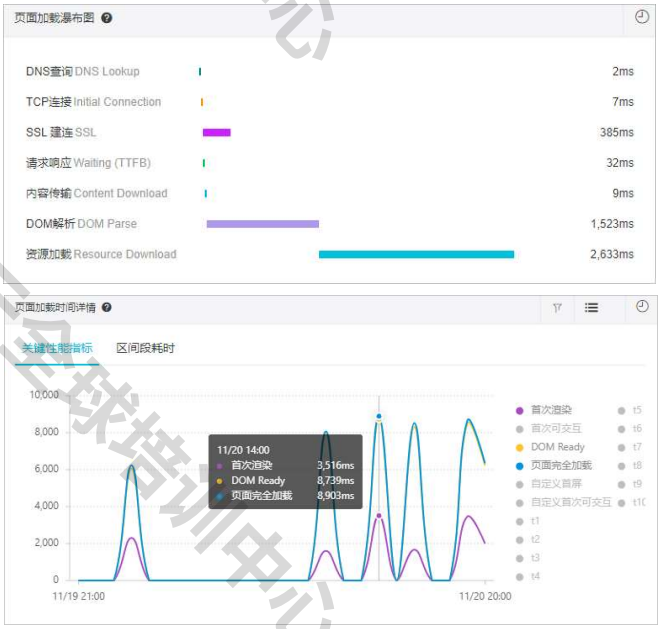
- 支持多种场景的前端监控
  - Web场景
  - Weex场景
  - 小程序场景

访问速度监控

应用场景

快速定位性能瓶颈，有针对性能优化

- 快速确定是DNS、TCP或者DOM解析瓶颈
- 明确不同国家、不同省份地域的性能情况
- 定位不同浏览器、设备上的性能问题



页面稳定性监控

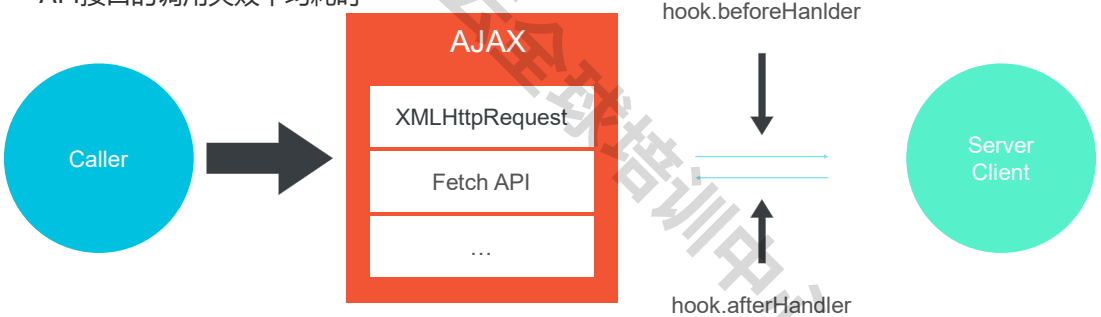
➢ 应用场景

JS错误的发生直接影响前端应用质量，JS错误诊断困难挑战。

- 复现困难
- 缺少监控信息，无法深入排查

服务调用监控

- ➢ 前端监控中最典型的服务调用：API请求
- ➢ 阿里云ARMS前端监控的API请求模块，可展示以下信息：
  - 每个API的成功率
  - API返回信息
  - API接口的调用成功平均耗时
  - API接口的调用失败平均耗时



前后端链路追踪

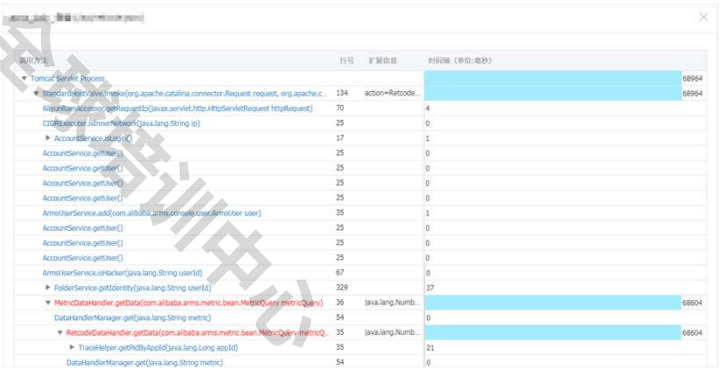
➢ 应用场景

前端与后端串联，一站式的问题排查

- 应用监控可提供API在后端的处理性能及调用链路，但这些数据未必能准确反映用户的真实体验
- 前端监控只能监控到API从发送到返回的整体耗时及状态，无法提供后端服务的调用链路及性能数据。

① 前端监控的整体耗时和后端应用的调用时序图

② 查看请求的完整后端调用链



应用监控——ARMS应用监控

ARMS 应用监控是一款针对应用性能管理的监控工具，结合 Google Dapper 的理论模型和阿里巴巴集团内部实践，提供更加全面的面向应用的业务实时监控。

- 应用拓扑的自我发现
- 基于常用诊断场景的指标下钻分析
- 异常事务和慢事务捕获
- 基于调用链的事务快照查询
- 即席多维排查功能
- PaaS 平台集成，一键集成其他阿里云产品服务

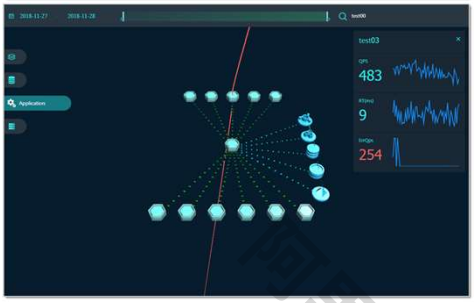
应用监控总览

- **应用总览**

展示当前应用的调用总览信息，包括应用拓扑图、应用对应的实例信息。

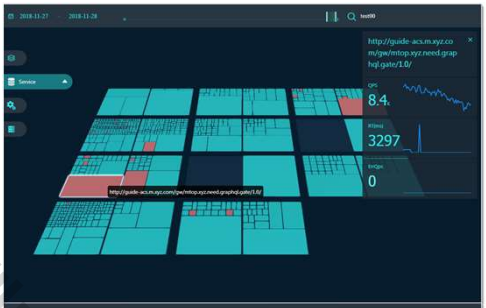
- **应用层**

展示应用及其上下游依赖关系，包括依赖的中间件在内。



- **服务层**

展示应用所依赖的服务信息



- **主机层**

展示应用的主机详情

查询调用链路

➢ 查询调用链

- 采用以下方法之一查询调用链。
  - 精确查询

    在 TraceId 精确查询搜索框中输入 TraceId查询

  - 高级查询

    接口名称

    客户端应用名

    服务端应用名

- **通过TraceID查看调用链路**

- **调用链路查询**

分析调用链路

关键技术点：

- 通过埋点关键接口 API 对所有经过的堆栈进行耗时统计。
- 能有效捕捉各类异常，如异常抛错，SQL 语句等。
- 支持100%上报堆栈，性能损耗平均3%不到，随开随停。

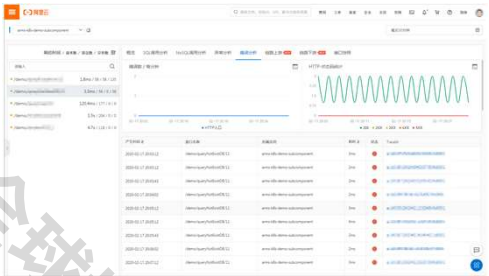
接口调用详情

接口调用详情，包括SQL调用分析、NoSQL调用分析、异常分析、错误分析、链路上下游和接口快照。

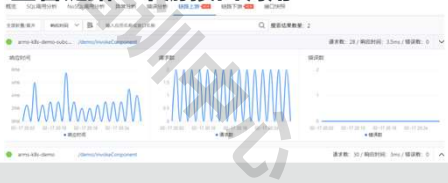
查看接口概览

查看SQL和NoSQL调用分析

查看错误分析



查看链路上下游接口调用



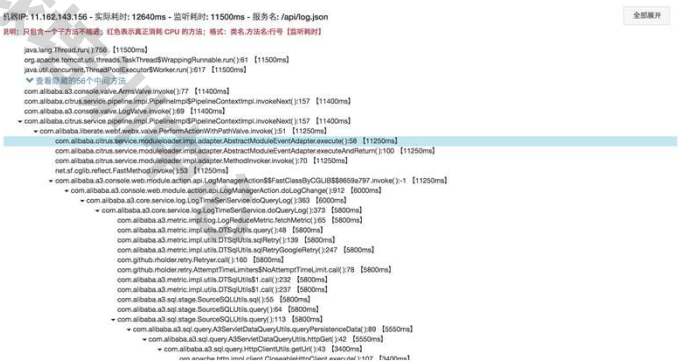
应用监控线程分析

➢ **线程分析**

- 查看全部线程
- 根据CPU耗时统计快速发现异常线程
- 分析CPU耗时与线程数变化

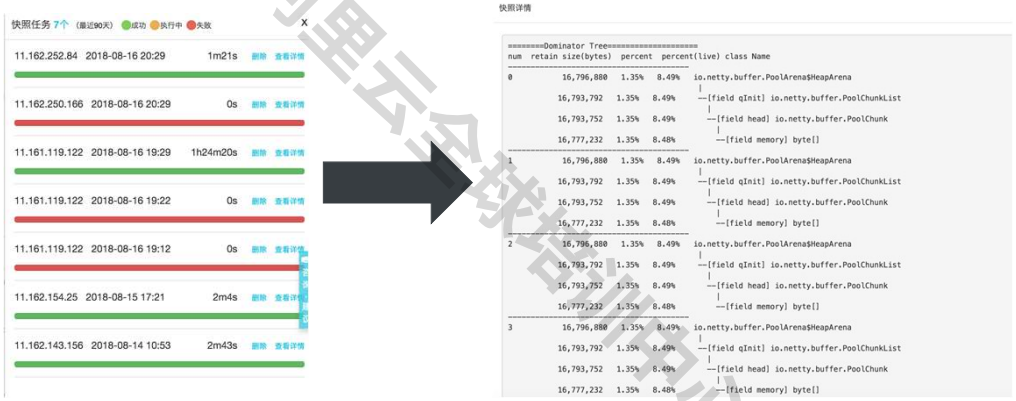
➢ **异常线程方法栈**

- 捕获所有超过阈值(如500ms)的调用。
- 捕获堆栈为所有全堆栈。
- 阈值动态设定，随改随生效。



应用监控内存诊断

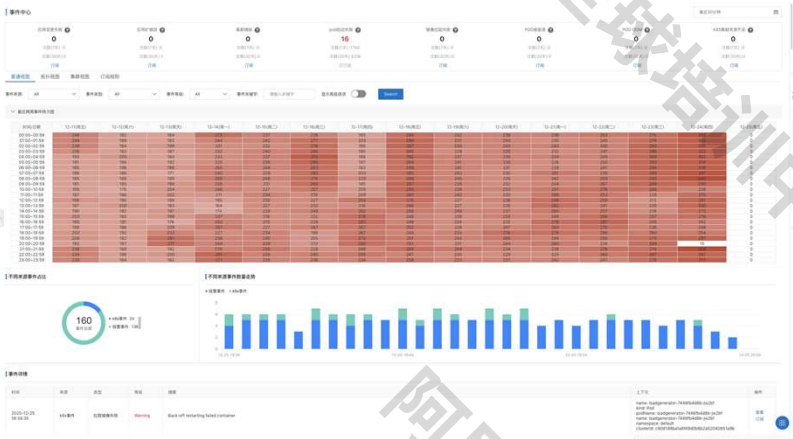
- **快速分析内存泄漏:**
  - 支持JVM相关性能统计如 Heap, GC 。
  - 根据内存信息，支持一键 Dump 内存并自动分析，根据对象大小进行排序。



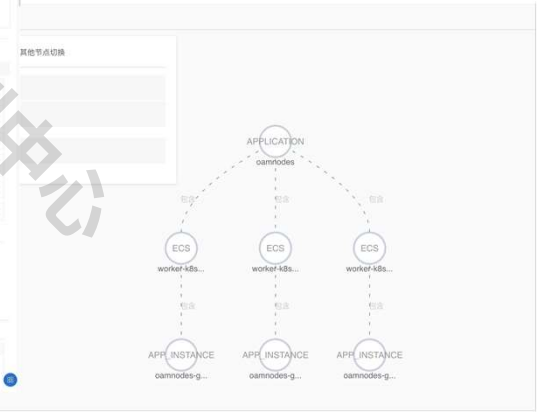
应用监控事件中心

应用监控的事件中心，将云产品产生的事件数据进行统一管理，存储，分析和展示。已经接入的事件包括EDAS的变更事件、ARMS的报警事件、MES的微服务管控事件、ACK集群事件等。

- 事件视图总览



- 拓扑视图，按照应用拓扑展示事件信息，包含应用事件，云监控事件，操作审计事件



Prometheus监控——ARMS组件监控

Prometheus监控是CNCF(Cloud Native Computing Foundation)下的开源监控报警系统和时序列数据库(TSDB)，目前广泛用于容器应用的监控。

主要由三部分组成：

•Server 主要负责数据采集和存储，提供PromQL查询语言的支持

•AlertManager 警告管理器，用来进行报警

•Push Gateway 支持临时性Job主动推送指标的中间网关

支持多种组件类型

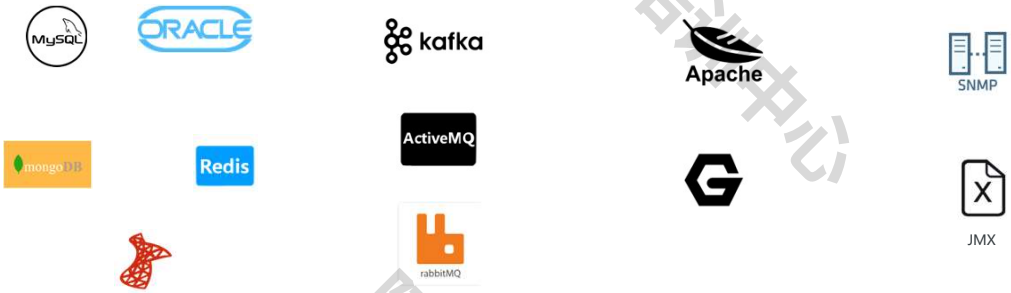
开箱即用的监控大盘

全托管的Prometheus服务

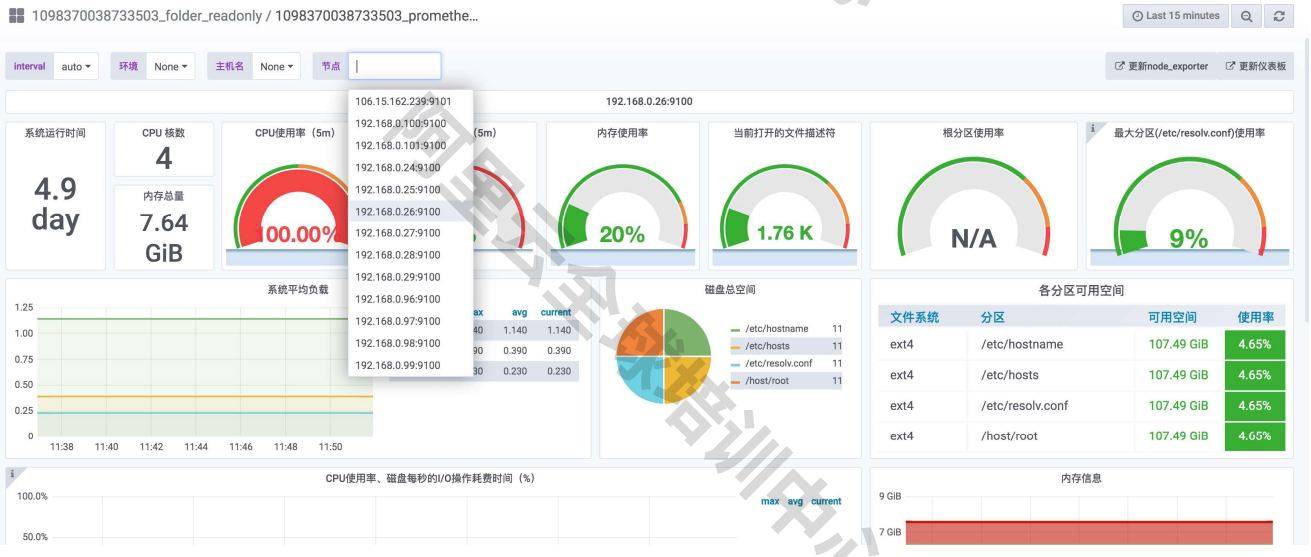
Prometheus监控介绍

完全兼容Prometheus生态，支持各种组件的监控

数据库　　消息　　HTTP　　其他

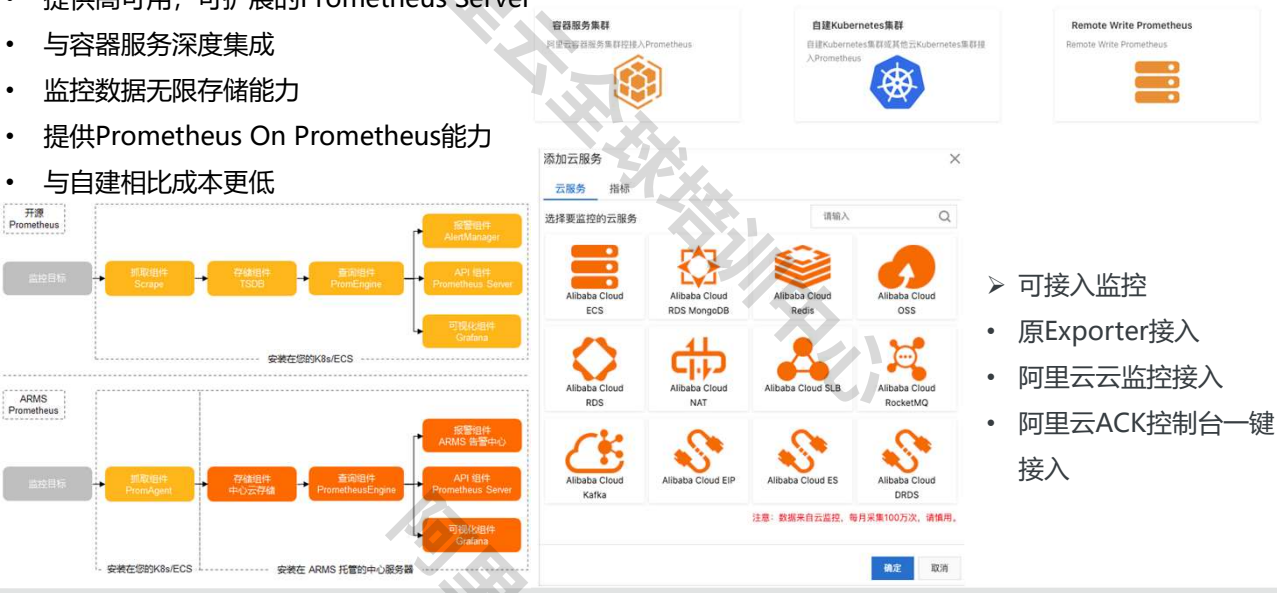


Prometheus监控大盘



Prometheus监控优势与范围

- 提供高可用，可扩展的Prometheus Server
- 与容器服务深度集成
- 监控数据无限存储能力
- 提供Prometheus On Prometheus能力
- 与自建相比成本更低



- ➢ 可接入监控
- 原Exporter接入
- 阿里云云监控接入
- 阿里云ACK控制台一键接入

容器监控——ARMS容器监控

ARMS容器监控能够对阿里云容器服务Kubernetes版的节点机器上的资源及容器进行实时监控和性能数据采集，并进行可视化展示，提供容器化环境端到端的监控排查路径。

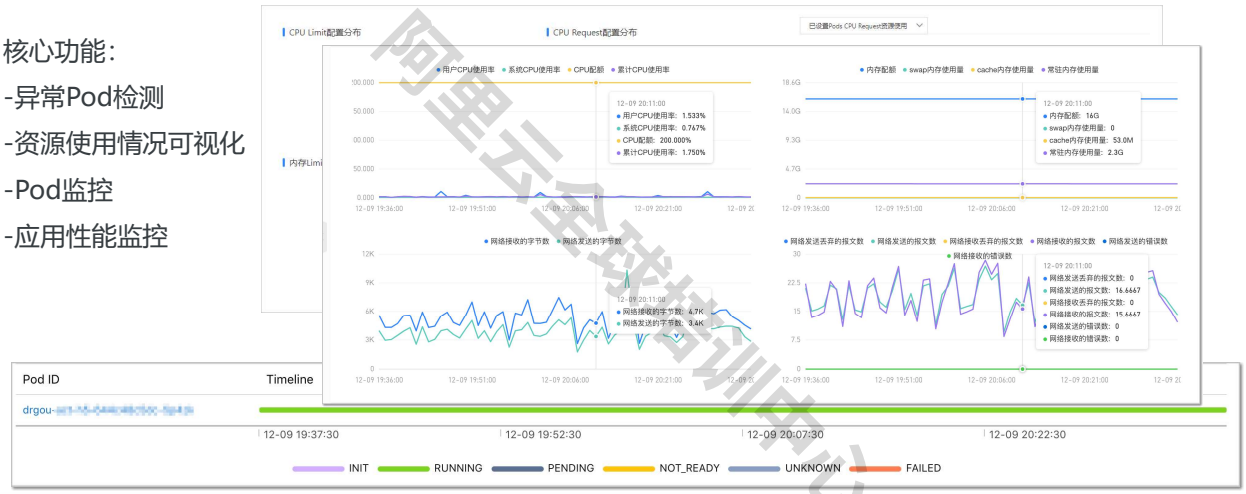
核心功能：

-异常Pod检测

-资源使用情况可视化

-Pod监控

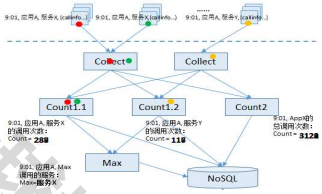
-应用性能监控



业务监控——ARMS业务监控

业务监控（原自定义监控），快速复制能力：

1. 鹰眼，前端
2. 其他业务统计：销售额，购物车等

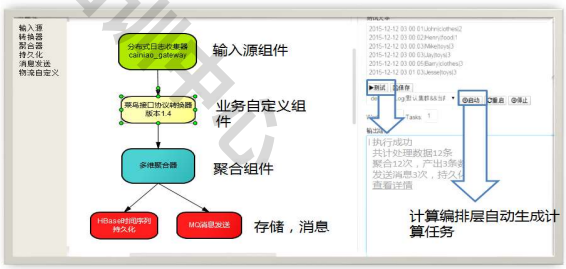


之前做法 -- 流式计算编程：

```
public static class ProduceRecordSpout extends BaseRichSpout {

    private static final long serialVersionUID = 1L;
    private static final Log LOG = LogFactory.getLog(ProduceRecordSpout.class);
    private SpoutOutputCollector collector;
    private Random random;
    private String[] records;

    public ProduceRecordSpout(String[] records) {
        this.records = records;
    }

    @Override
    public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
        this.collector = collector;
        random = new Random();
    }

    @Override
    public void nextTuple() {
        Utils.sleep(100);
        String record = records[random.nextInt(records.length)];
        List<Object> values = new Values(record);
        collector.emit(values, values);
        LOG.info("Record emitted: record=" + record);
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("record"));
    }
}
```

```
    @Override
    public void cleanup() {
        // print count results
        LOG.info("Word count results:");
        for(Entry<String, AtomicInteger> entry : counterMap.entrySet()) {
            LOG.info("\tword=" + entry.getKey() + ", count=" + entry.getValue().get());
        }
    }
}
```

新的做法 -- 可视化编程：



改进后的新指标接入效率：几天 -> 几分钟

业务监控的能力

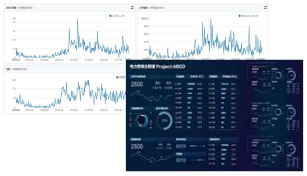
ARMS 应用实时监控提供一站式服务平台的功能性介绍：

- 丰富的数据接入和输出层，用户接入门槛低。
- 基于可视化编程的实时计算编排，MOLAP存储，智能报警和展示的一站式服务，开发效率高，
- 监控平台高可靠，高性能，秒级响应，数据一致。
- 应用场景丰富，各类行业模板开箱即用。

ARMS 实际适用的产品定位

- 业务监控大屏
- 先计算后存储的实时计算 + OLAP平台
- 数据管道 (Loghub, MQ -> MaxCompute, DataV等)

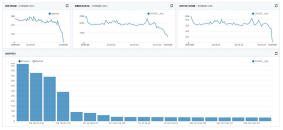
业务监控模版



- 电商总体销售额
- 渠道分布监控
- 新老用户注册监控
- 事件及流失率分析
- 终端用户分析

- 零售总额，件数，单数统计
- 区域销售量统计
- 主管销售量统计
- 畅销物品类目统计

- 总体访问量
- URL调用详情
- 响应时间监控
- 返回值错误占比报警
- 爬虫情况，搜索推荐情况



- 车辆在线数
- 车辆故障数
- 车辆行驶区域统计
- 平均 / 最高速度统计

课程目录03

完整开通使用流程

- ARMS实时监测服务的类别较多，各子产品的开通，收费以及使用方式差别较大。
- 各子产品的开通使用流程单独介绍

付费子产品	15天免费试用期每日额度
应用监控	240 Agent×小时（例如10个Agent分别运行24小时）
前端监控	2万次数据上报
Prometheus监控	2000万条自定义指标上报

应用监控接入概述

- 按部署环境开始监控

EDAS
- 为EDAS中的应用快速安装探针

阿里云容器服务K8s集群
- 为容器服务Kubernetes版Java应用安...
- 为容器服务Kubernetes版PHP应用安...

开源K8s集群
- 为开源Kubernetes环境中的应用安装...

Docker集群
- 为Docker中的应用安装探针

其他环境（如自建IDC）
为Java应用手动安... 为Java应用自动安...
为普通PHP应用安... 为单机多站点PHP...

- 按应用语言开始监控

开始监控Java应用
为Java应用手动安... 为Java应用自动安...
为EDAS中的应用快... 为Docker中的应用...
为容器服务Kuberne...

开始监控PHP应用
为普通PHP应用安... 为单机多站点PHP...
为容器服务Kuberne...

开始监控Go应用 *
- 通过Jaeger上报Go应用数据
- 通过Zipkin上报Go应用数据

开始监控Node.js应用 *
- 通过Jaeger上报Node.js应用数据

开始监控 .NET应用 *
- 通过Jaeger上报 .NET应用数据
- 通过Zipkin上报 .NET应用数据

开始监控C++ 应用 *
- 通过Jaeger上报C++ 应用数据

ARMS应用监控三步曲

ARMS应用监控可以监控运行在多种环境下的Java应用和PHP应用。现以ECS实例中Tomcat环境下运行的Java应用为例，介绍如何创建一个应用监控任务。

1. 获取LicenseKey，修改Tomcat配置文件

打开{TOMCAT_HOME}/bin/catalina.sh配置文件
JAVA_OPTS="$JAVA_OPTS -
javaagent:/workspace/ArmsAgent/arms-bootstrap-1.7.0-
SNAPSHOT.jar -Darms.licenseKey=<licenseKey> -
Darms.appName=<appName>"

2.安装Java探针

3.重启Tomcat应用

ARMS前端监控接入概述

ARMS前端监控针对Web场景，Weex 场景和小程序场景的监控，分三种不同的接入方式
其中，Web场景接入，又分为CDN方式安装探针和NPM方式安装探针两种

前端监控接入-Web场景

➢ Web场景——以CDN方式接入前端监控

- 异步加载：复制指定代码操作后，重启应用

- 同步加载：复制指定代码操作后，重启应用

➢ Web场景——以npm方式接入前端监控

- 安装alife-logger

在npm仓库中安装 `alife-logger` 。

```
npm install alife-logger --save
```

- 初始化配置

SDK以 `BrowserLogger.singleton` 方式初始化。

```
const BrowserLogger = require('alife-logger');
                // BrowserLogger.singleton(conf) conf传入config配置。
                const __bl = BrowserLogger.singleton({
                pid: 'your-project-id',
                // 设定日志上传地址:
                // 部署新加坡地域可设为`https://arms-retcode-sg.aliyuncs.com/r.png?`。
                // 部署美西地域可设为`http://arms-us-west-1.console.aliyun.com/r.png?`。
                imgUrl: 'https://arms-retcode.aliyuncs.com/r.png?',
                // 其他config配置。
                });
```


前端监控接入-Weex场景

- 导入npm包，由WeexLogger模块上报日志

```
npm install alife-logger --save
```

- 创建monitor.js，完成SDK初始化

```
// 初始化SDK。
const wxLogger = WeexLogger.singleton({
    pid: 'your-project-id',
    uid: 'zhangsan',
    // 设置uid，用于生成UV报告。
    page: 'Lazada | Home',
    // 设置初始页面名称，SDK将在初始化完成后发送PV日志。
    imgUrl: 'https://arms-retcode.aliyuncs.com/r.png?',
    // 设定日志上传地址，如需部署至新加坡地域，则改为'https://arms-retcode-sg.aliyuncs.com/r.png?'。
    // 设置GET上报方法，示例如下：
    sendRequest: (data, imgUrl) =>{
        const url = imgUrl + serialize(data);
        fetch({
            method: 'GET',
            url
        });
    },
    // 设置POST上报方法，示例如下：
    postRequest: (data, imgUrl) =>{
        fetch({
            method: 'POST',
            type: 'json',
            url: imgUrl,
            body: JSON.stringify(data)
        });
    }
});
```

- 上报，通过实例调用相应上报方法

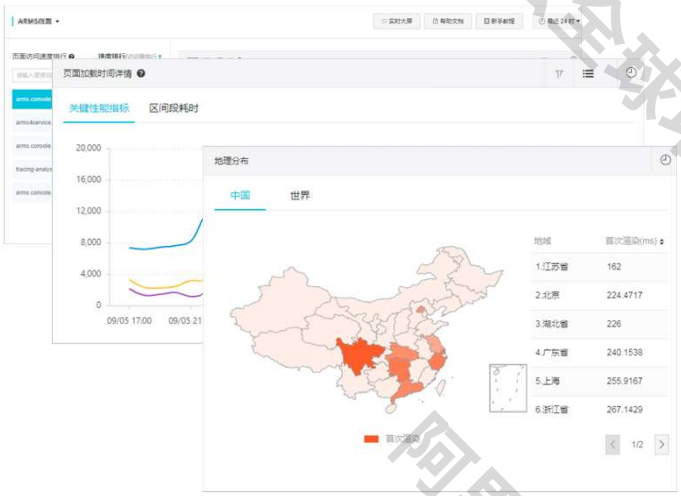
```
// in some biz module
import wxLogger from '/utils/monitor';
wxLogger.api('/search.do', true, 233, 'SUCCESS');
```

- 通用SDK配置项参考

参数	类型	描述
pid	String	项目唯一ID，由ARMS在创建站点时自动生成。
uid	String	用户ID，用于标识访问用户，可手动配置，用于根据用户ID检索。如果不配置，则由SDK自动生成且每半年更新一次。
tag	String	传入的标记，每条日志都会携带该标记。
release	String	应用版本号。建议您配置，便于查看不同版本的上报信息。
environment	String	环境字段，取值为：prod、gray、pre、daily和local，其中： • prod表示线上环境。 • gray表示灰度环境。 • pre表示预发环境。 • daily表示日常环境。 • local表示本地环境。
sample	Integer	日志采样配置，值为 1 、10 或 100 ，性能和成功API日志按照 1/sample 的比例采样。

前端监控接入效果

- 页面访问速度



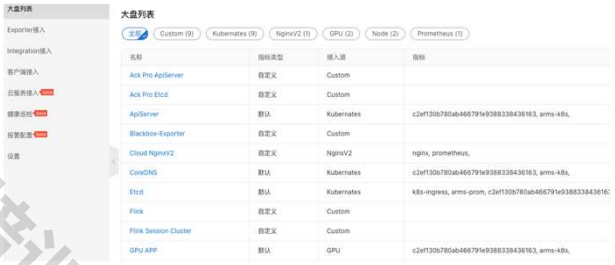
- JS错误诊断
  - ✓ 错误数：选定时间段内的JS错误总数。
  - ✓ JS错误率：选定时间段内发生过错误的PV占总PV的比例。
  - ✓ 影响用户数：JS错误影响到的用户数量和比例。

Prometheus监控接入

- 以阿里云容器服务集群接入Prometheus监控为例，先选择地域，然后进入安装引导界面

- 选择对应的集群，创建应用目录

- 接入监控指标



创建ARMS报警

ARMS报警按照不同的监控类别，分不同报警模板

- 创建应用监控报警

- 创建前端监控报警

阿里云全球培训中心

创建ARMS报警

- 创建通用报警

✓ 报警类型

应用监控，前端监控，自定义监控

✓ 报警维度

无，=，遍历

✓ 通知方式

邮件，短信，钉钉机器人以及Webhook

字段	含义	说明
应用站点	已创建的监控任务。	在下拉菜单中选择。
报警类型	报警指标的类型。	三种报警的报警指标类型各不相同： • 应用监控报警：应用入口调用、应用调用类型统计、数据库指标、JVM监控、主机监控和异常接口调用。 • 前端监控报警：页面指标、API指标、自定义指标和页面API指标。 • 自定义监控报警：基于已有下钻数据集创建报警和基于已有通用数据集创建报警。
报警维度	配置报警指标（数据集）的维度，可选择为：无、=、遍历。	• 配置为无：报警内容中透出这个维度所有数值的和。 • 配置为=：具体内容需手动填写。 • 配置为遍历：会在报警内容中透出实际触发报警的维度内容。
最近N分钟	报警判断最近N分钟内数据结果是否达到触发条件。	N的范围为：1-60分钟。
通知方式	支持邮件、短信、钉钉机器人和Webhook四种方式。	可勾选多种方式。若需设置钉钉机器人报警请参见，设置钉钉机器人报警。
报警静默期开关	可选择为开启或关闭，默认为开启状态。	• 开启报警静默期开关：若数据一直处于触发状态，首次触发报警后，24小时后才会发送第二次报警信息。当数据恢复正常，会或判断或恢复通知并解除报警。若数据再次触发报警，则会再次发送报警信息。 • 关闭报警静默期开关：若报警数据触发，将会每分钟发送一次报警信息。
报警级别	包括警告、错误和致命。	无
通知时间	报警发送时的通知时间。此时间范围外将不发送报警通知，但仍会有报警事件记录。	查看报警事件记录请参见管理报警。
通知内容	自定义的报警通知内容。	您可以编辑默认模板，在模板中，除$报警名称、$筛选、$报警时间和$报警内容等4个变量（暂不支持其它变量）为固定搭配，其余内容均可自定义。

阿里云全球培训中心

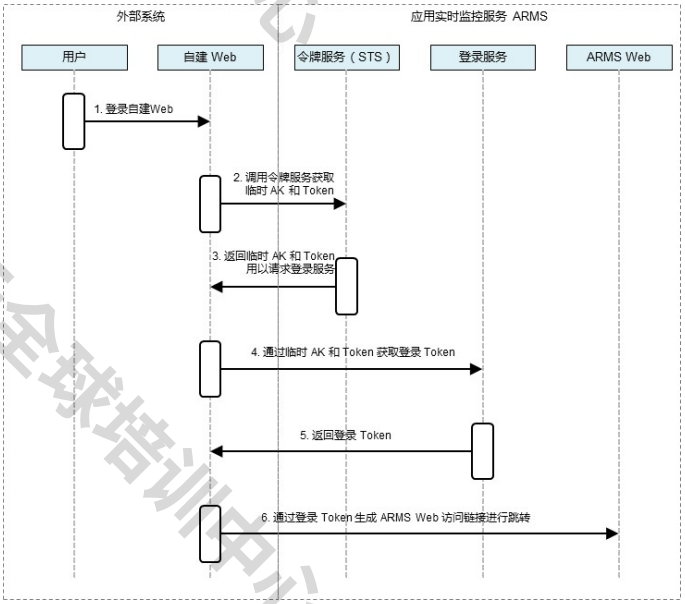
课程目录04

1. 应用实时监测ARMS的概述
2.应用实时监测ARMS的功能特性
3.应用实时监测ARMS的基本使用
4.应用实时监测ARMS的最佳实践

将ARMS页面内嵌到自建的Web应用

通过集成可以实现：

- 可登录您的自有系统并浏览嵌入的应用列表、应用详情、调用查询等页面。
- 可隐藏 ARMS 控制台页面的顶部导航栏和左侧导航栏。
- 可通过访问控制 RAM 控制操作权限，例如将完整权限改为只读权限等。



将ARMS页面内嵌到自建的Web应用

一、准备工作：创建 RAM 用户并添加权限

1. 登录 RAM 控制台，在左侧导航栏中选择**人员管理 > 用户**，并在**用户**页面上单击**新建用户**。
2. 在**新建用户**页面的**用户账号信息**区域框中，输入**登录名称**和**显示名称**。在**访问方式**区域框中，勾选**编程访问**，并单击**确认**。

注意

RAM 会自动为 RAM 用户创建 AccessKey（API 访问密钥）。出于安全考虑，RAM 控制台只提供一次查看或下载 AccessKeySecret 的机会，即创建 AccessKey 时，因此请务必将 AccessKeySecret 记录到安全的地方。

将ARMS页面内嵌到自建的Web应用

一、准备工作：创建 RAM 用户并添加权限

3. 在**手机验证**对话框中单击**获取验证码**，并输入收到的手机验证码，然后单击**确定**。 创建的 RAM 用户显示在**用户**页面上，但此时没有任何权限。
4. 在**用户**页面上找到创建好的用户，单击**操作**列中的**添加权限**。
5. 在**添加权限**面板的**选择权限**区域框中，通过关键字搜索需要添加的权限策略 **AliyunSTSAssumeRoleAccess**，并单击权限策略将其添加至右侧的**已选择**列表中，然后单击**确定**。

6 在添加权限的授权结果页面上，查看授权信息摘要，并单击完成。

将ARMS页面内嵌到自建的Web应用

二、准备工作：创建 RAM 角色并添加权限

1 登录 RAM 控制台，进入RAM 角色管理，新建 RAM 角色

2 在新建 RAM 角色面板中，依次

类型选择阿里云账号 -> 输入角色名称 -> 为角色授权

3 添加角色权。

4 可选： 在添加权限面板的授权结果页面上，查看授权信息摘要，并单击完成

将ARMS页面内嵌到自建的Web应用

三、集成开发

步骤一：获取临时 AccessKey 和 Token

1 登录自建 Web 应用后，在 Web 服务端调用 STS AssumeRole 接口获取临时 AccessKey 和 Token，即临时身份。请选择一种方式调用该接口：

通过 OpenAPI 在线调用

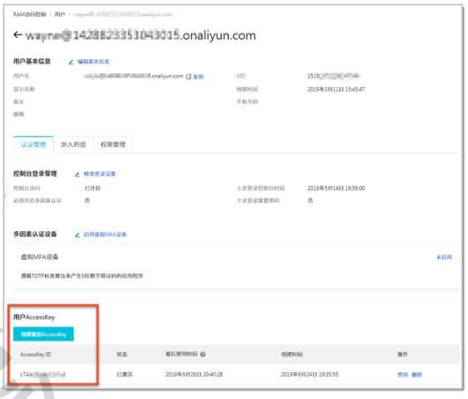
通过 Java SDK 调用

2 Demo中以Java SDK 调用的方式为例。Demo下载：示例代码　　注意：首先需要将以下demo的参数替换为真实的值。

```
String accessKey = "xxxxxx"; //准备工作中创建的 RAM 用户的 AccessKeyId
String accessSecret = "xxxxx"; //准备工作中创建的 RAM 用户的 AccessKeySecret
String roleArn = "acs:ram::xxxxxx:role/xxxxxx"; //准备工作中创建的 RAM 角色的标识 ARN
```

accessKey 和 accessSecret 是准备工作中创建的 RAM 用户的 AccessKeyId 和 AccessKeySecret。

roleArn 是准备工作中创建的 RAM 角色的标识 ARN，可在 RAM 控制台的 RAM 角色基本信息页面获取。



将ARMS页面内嵌到自建的Web应用

三、集成开发

步骤二：获取获取登录 Token

在通过 STS AssumeRole 接口获取临时 AccessKey 和 Token 后，调用登录服务接口获取登录 Token。

注意 STS 返回的安全 Token 中可能包含特殊字符，请对特殊字符进行 URL 编码后再输入。

代码样例如下：

```
http://signin.aliyun.com/federation?Action=GetSigninToken
    &AccessKeyId=<STS 返回的临时 AccessKeyId>
    &AccessKeySecret=<STS 返回的临时 AccessKeySecret>
    &SecurityToken=<STS 返回的安全 Token>
    &TicketType=mini
```


将ARMS页面内嵌到自建的Web应用

三、集成开发

步骤三：生成免登录链接

利用获取到的登录 Token 与待嵌入的 ARMS 控制台页面链接生成免登录访问链接，以最终实现在自建 Web 中免登录访问 ARMS 控制台页面的目的。

说明 由于 Token 有效期为 3 小时，建议在自建 Web 应用中将链接设置为每次请求时生成新登录 Token。

1. 在 ARMS 控制台获取待嵌入的控制台页面链接。例如杭州地域的**应用列表**页面的链接为：
   https://arms.console.aliyun.com/?hideTopbar=true&hideSidebar=true#/tracing/list?regionNo=cn-hangzhou
2. 利用步骤二中获取到的登录 Token 与 ARMS 控制台页面链接生成免登录访问链接。 请求样例：

```
http://signin.aliyun.com/federation?Action=Login
    &LoginUrl=<登录失效跳转的地址，一般配置为自建 Web 配置 302 跳转的 URL>
    &Destination=<ARMS 控制台页面链接>
    &SigninToken=<获取到的登录 Token>
```

调试，并完成

阿里云全球培训中心

总结及回顾

本章节主要讲述的内容：

1. 构建完整企业IT监控体系需要覆盖的内容

2.全栈监控包含的内容：应用监控，前端监控，APP监控，容器监控，业务监控等

3.应用实时监测ARMS的功能使用介绍

- APP监控的性能分析，奔溃分析，远程日志
- 前端监控的访问速度，页面稳定性和服务调用链路
- 应用监控的查询调用链路，分析调用链路，接口调用详情
- Prometheus的监控大盘，优势与涵盖范围等

4.应用实时监测ARMS的最佳实践

阿里云云原生微服务ACP认证培训

性能测试服务PTS

阿里云

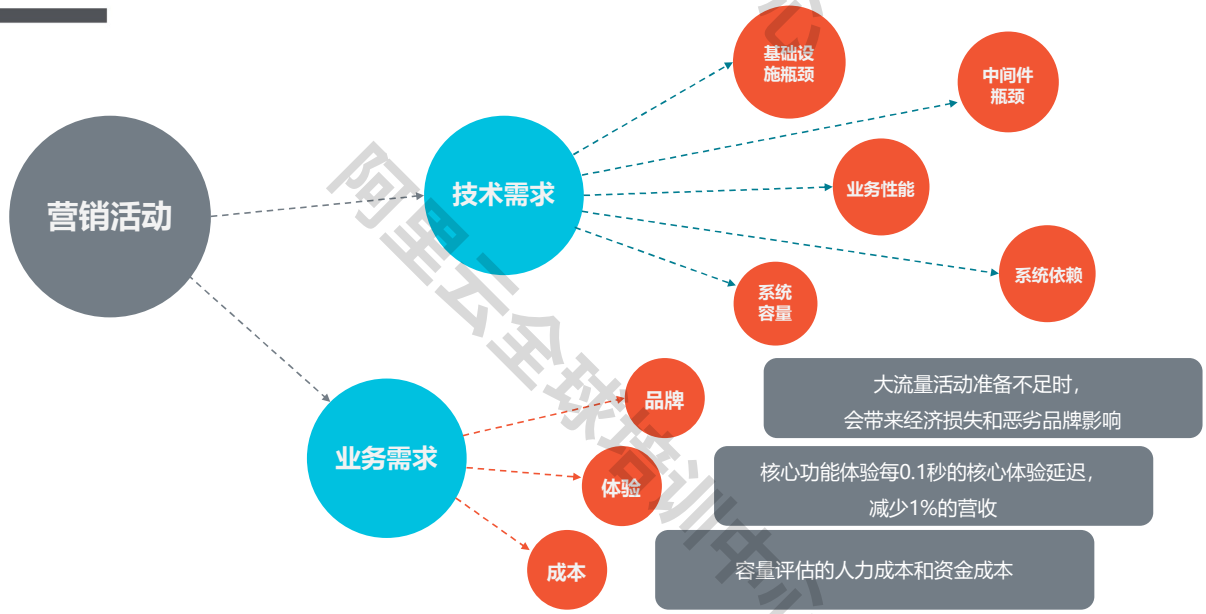
课程目标

学习完本课程后，你将能够：

1. 了解阿里云性能测试产品PTS
2. 掌握性能测试PTS的功能特性和基本技术原理

课程目录01

营销活动对性能测试的需求



性能测试的场景

新系统上线

通过 PTS，准确探知站点能力，防止系统一上线即被用户流量打垮

技术升级验证

大的技术架构升级后进行性能评估，验证新技术场景的站点性能状态

业务峰值稳定性

大促活动等峰值业务稳定性考验，保障峰值业务不受损

站点容量规划

对站点进行精细化的容量规划，分布式系统机器资源分配

性能瓶颈探测

探测系统中的性能瓶颈点，进行针对性优化

PTS产品功能简介

PTS是面向所有技术背景人员的云化测试工具

A 非阿里云客户能使用吗?

可以，无论客户是公有云/专有云/混合云/自建IDC，无论什么云厂商，只要在公网可访问就能通过PTS来压测。

B 使用上的门槛

不需要专业测试背景，只需要懂一点http协议即可进行压测场景的设置。也可以基于云端录制器和日志回放进行压测。

PTS在各行业的应用

游戏/视频	电商	互联网APP	网站类	各平台应用
手游/页游/直播/短视频 • 点赞 • 留言 • 聊天 • 赠送礼物 • 广播等	**电商交易类(PC或APP)** • 浏览 • 秒杀 • 购物车 • 下单	**社交、互金、出行、生活、知识付费等** • 拉新 • 推广 • 自身各种核心业务场景	**几乎所有行业** • 首页 • 注册 • 登陆 • 内部系统 • 核心业务等	**各平台上的应用** • 支付宝 • 微信 • 钉钉 • 各类开放平台上的第三方业务和页面

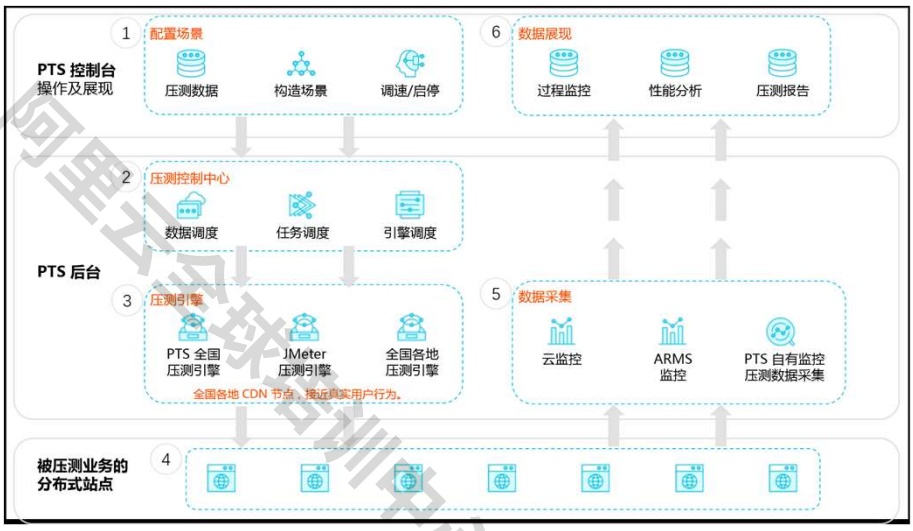
主流性能测试工具对比

	同类型云化测试工具	开源&商用工具
其他性能压测工具	• 其他云计算厂商的云化测试工具 华为云 云性能测试服务 CPTS 腾讯云 压测大师 LM	• 基于开源的Jmeter，或商业软件LoadRunner构建压测平台
PTS优势	• 功能最全 • 易用度最高 • 价格透明且较低 • 平台稳定 • 阿里双十一等场景全链路压测	• 全SaaS化形态，即买即用 • 不需专人维护，不用维护、机器、环境、维修 • 流量真实，施压能力无上限 • 配套完善 • 综合成本更低

课程目录02

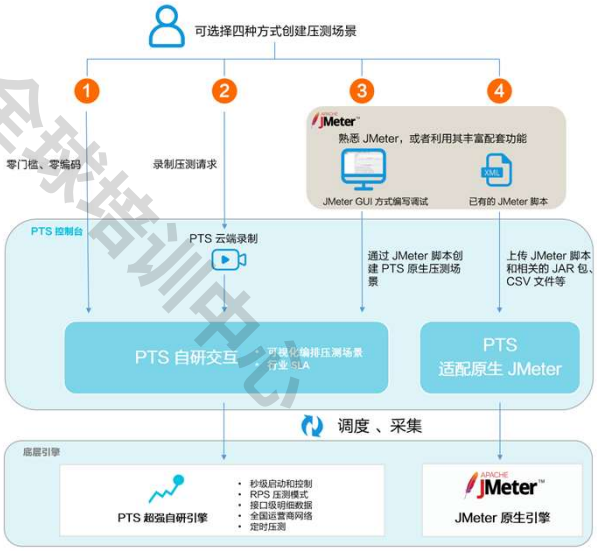
PTS压测流程

1. 在PTS控制台构造压测场景和配置
2. 后台压测控制中心进行自动调度
3. 压测引擎从各节点发起压测流量
4. 站点接受压测流量测试
5. 过程中实时采集压测数据
6. 实时压测数据和事后压测报告



PTS功能一 —— 4种压测创建方式

1. PTS 自研零编码可视化编排
2. PTS 自研云端录制器
3. JMeter 脚本倒入 PTS 自研交互
4. Jmeter 脚本导入原生 Jmeter 引擎

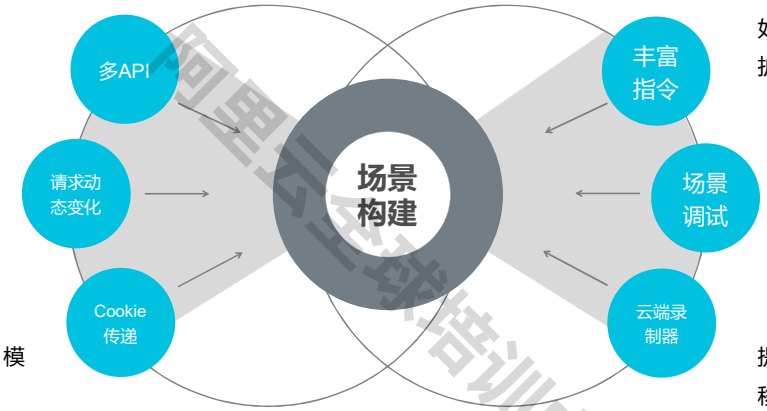


PTS功能一 —— 压测场景构建

支持多个 API 并行或者有序串行

支持 API 地址中添加参数，实现请求的动态变化

支持 Cookie 传递，模拟用户登录场景



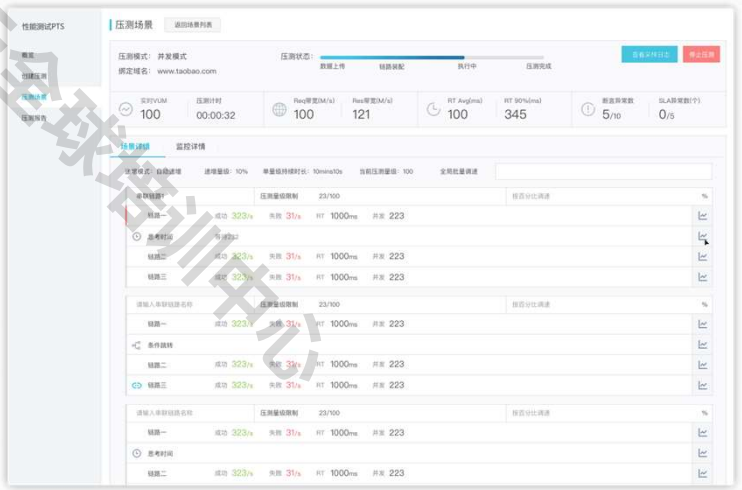
提供丰富的指令功能，如集合点、思考时间等，扩展场景的仿真度

支持压测前的场景调试，可进行复杂场景的数据流向校验

提供云端录制器，便于移动端的请求抓取，可一键导入到压测场景中

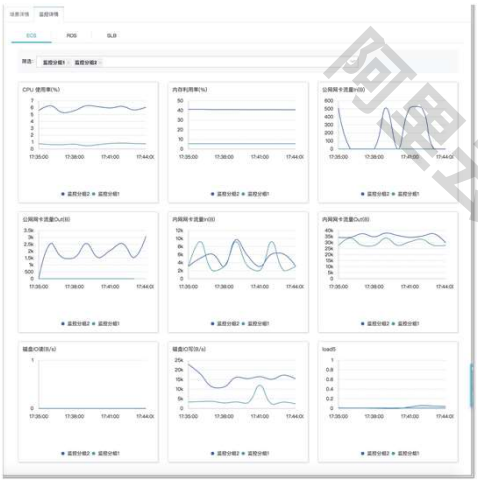
PTS功能二 —— 压测流量控制

- 除并发模式还支持特有的**吞吐量压测模式**-RPS模式，即时启动。
- 容量评估功能可协助探测系统的最佳压力点、极限压力点和破坏压力点，帮助评估系统容量。
- 支持定时压测，结合服务等级定义 SLA 指标监控，触发告警或停止压测，实现智能压测。
- 支持两种调速模式：自动递增和手动调整。压测流量的调整秒级生效

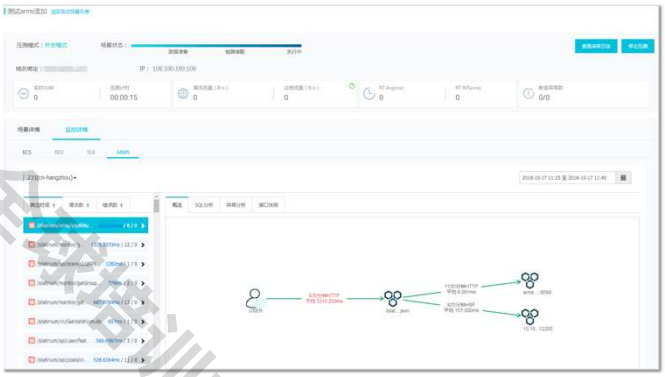


PTS功能三 —— 压测数据监控

- 云监控



- ARMS监控



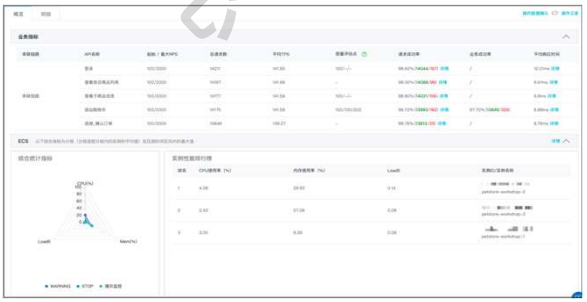
PTS功能四 —— 压测报告之压测场景概览

指标	说明
并发	峰值：压测周期内场景的最高并发值 上限：根据该场景配置的压测量级，限制的最大并发数
RPS	Requests Per Second，即每秒发出的压测请求数量。 - 峰值：压测周期内，该场景的最高 RPS - 上限：根据该场景配置的压测量级，限制的最大 RPS
来源 IP	发起压测流量的 IP 地址个数。 - 最小：压测周期内，该场景的最小来源 IP 数； - 配置：场景施压配置中配置的来源 IP 个数） 单击**分配详情**，可查看发起压测流量的 IP 地址所在的地理位置和所属运营商。该功能仅国内公网压测可见。
平均流量	压测周期内的平均流量，PTS 采样统计的基于7层（HTTP）请求体或者响应体的最大值（与4层网络带宽完全不同）。
峰值流量	压测周期内的最大流量，算法同**平均流量**。
异常数	单击异常数，可快速查看其采样日志。 - 请求异常数：压测过程中，请求失败个数（包括 4XX、5XX、各种异常、超时等）。 - 业务异常数：设置了断言的 API 业务失败的请求数。
总请求数	整个压测过程中，共发起的请求个数。

PTS功能四 —— 压测报告之串联链路概览和详情

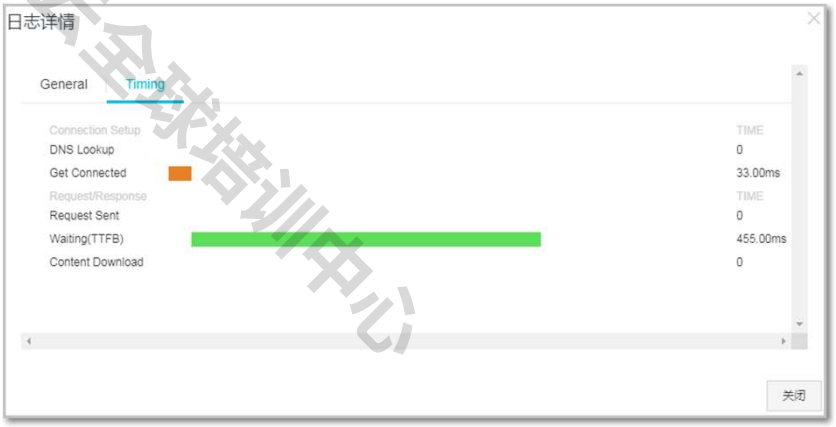
显示整个场景下所有的串联链路和 API 的压测情况；如添加了云监控产品，ECS、SLB 或者 RDS，可以看到这些产品在压测期间的性能表现。

业务指标	说明
串联链路	并发模式下，串联链路的起始/最大并发量；RPS 模式下，API 的起始/最大RPS。
请求成功率	压测中此 API 的请求成功率。 - 单击请求成功/失败个数，可快捷查看对应日志。 - 单击**详情**，查看 3XX、4XX、5XX 和其他异常导致的请求失败的个数。
平均响应时间	压测中此 API 的平均响应时间。点击**详情**，查看最大、最小及其各分位的响应时间。
容量评估点	开启容量评估模式下，三个压力值信息。



PTS功能四 —— 压测报告之采样日志

- 采样日志按照 1% 频率进行收集，保存30天。
- 压测采样日志包括压测API的：
  - 请求详情
  - 响应详情
  - 请求核心生命周期耗时信息



业务指标	说明
DNS Lookup	执行 DNS 查询所用的时间。
Get Connected	建立连接所用的时间。
Request Sent	发出网络请求所用的时间。
Waiting(TTFB)	等待初始响应所用的时间
Content Download	接收响应数据所用的时间。
Reuse	复用连接所用的时间。

课程目录03

完整PTS压测流程概述

场景配置
1、自定义API，配置基本请求
2、串联链路
3、场景调试

施压配置
1、压力来源
2、施压模式
3、配置量级及数据

高级设置
1、通用设置：连接超时时间、成功状态码
2、域名绑定

监控集成
1、云监控：ECS，RDS，SLB
2、ARMS监控：应用监控

场景SLA
1、配置场景SLA
2、SLA结果验证及统计

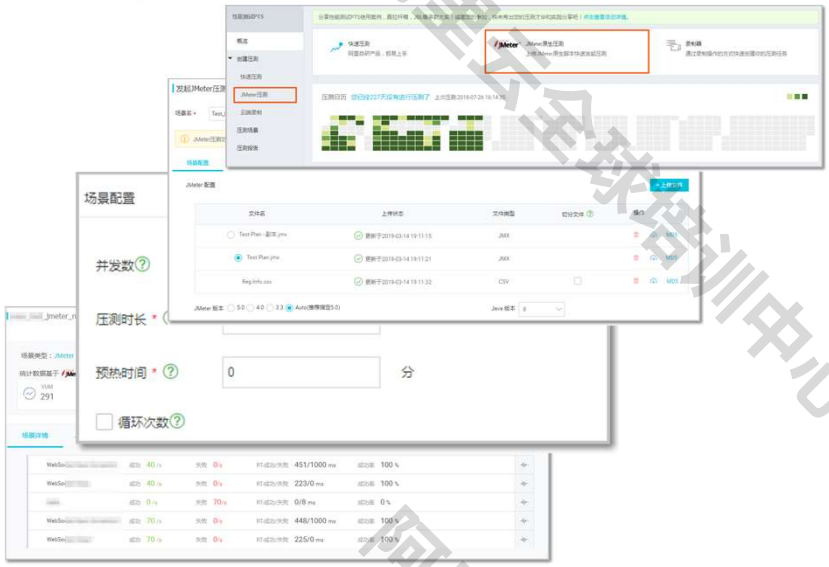
压测及调速
1、启动压测
2、调速：自动递增、全场景调速、串联链路调速、手工模式

压测报告
压测报告包括：
1、压测场景概览
2、串联链路信息
3、监控信息

配置场景-快速压测

创建快速压测
↓
填写压测信息
↓
开始压测
↓
查看压测报告和日志

配置场景-Jmeter压测



发起Jmeter压测

↓

上传Jmeter脚本

↓

配置压测场景

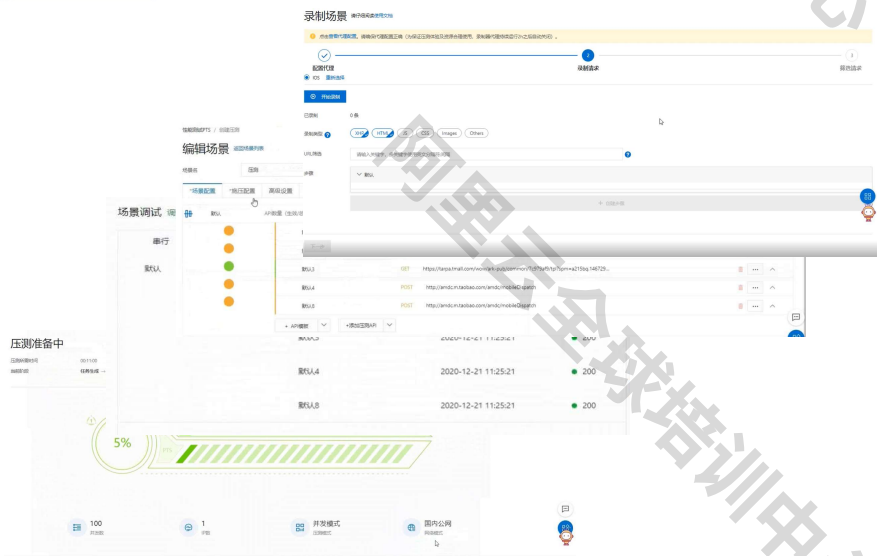
↓

发起压测

↓

查看压测报告和日志

配置场景-云端录制器



使用录制器录入

↓

录制压测请求导入场景

↓

配置压测场景

↓

发起压测

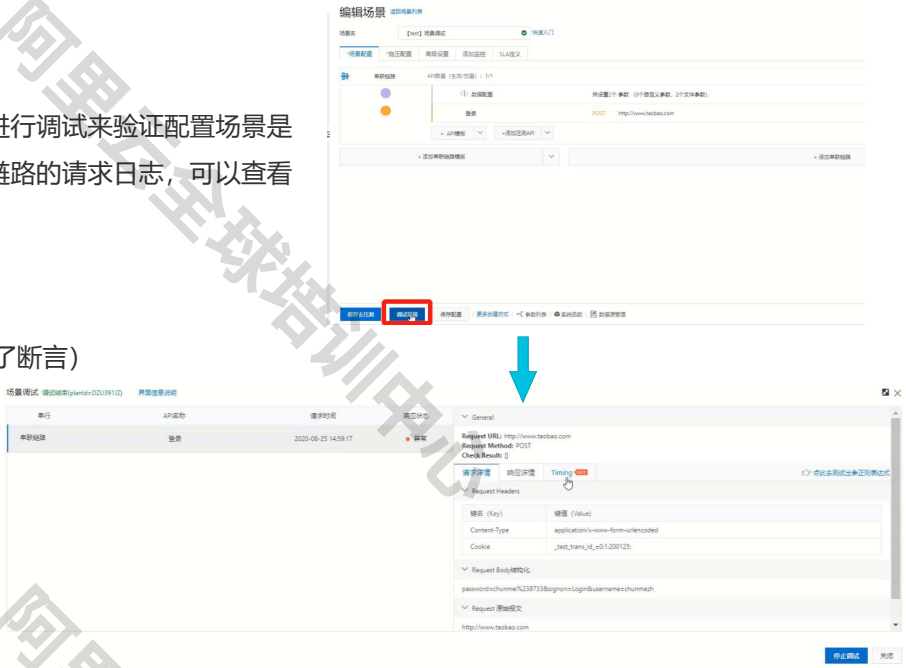
↓

查看压测报告和日志

场景调试

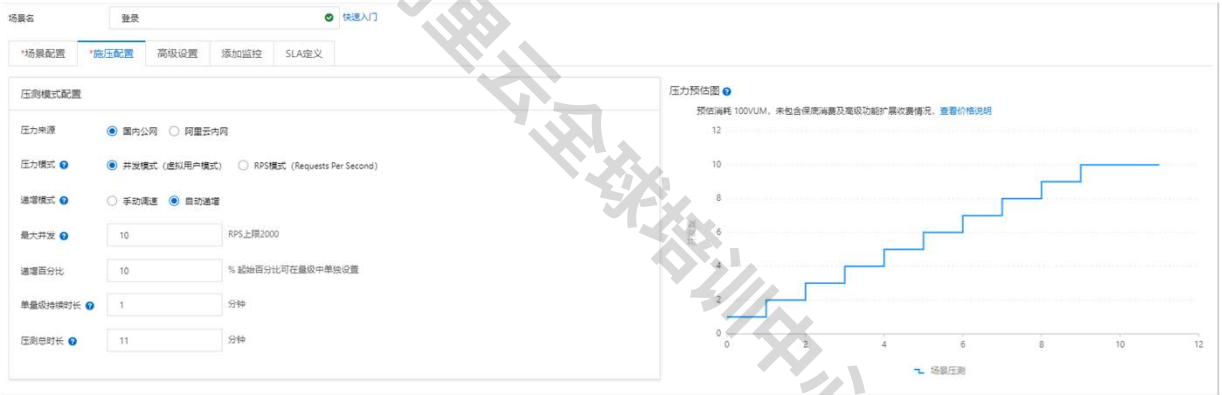
编写好场景之后，需要先对场景进行调试来验证配置场景是否合理。在调试结果中查看所有链路的请求日志，可以查看该API的调试详情，包括：

- 请求URL
- 请求Header
- 请求参数断言情况（如果设置了断言）
- 应答Body
- 应答Header
- 应答的异常（如果存在）



配置压测模式

压力来源：国内公网 / 阿里云内网
压力模式：并发模式 / RPS模式
递增模式：自动递增 / 手动调速
压测总时长：设置压测的时间长度



设置量级及数据

设置好压测模式后，需要设置压测起始量级与最大量级；每个API可以视为业务系统的一个节点，处理能力不同导致可承载的业务量也不一致。

注意事项：各场景最大值的总和不可超过该账户下对应资源包的最大VU、RPS。

启动压力测试

启动压测后如右图：

1.数据信息和配置信息区域

2.单链路（API）实时状态区域

3.单个调速

4.批量调速

5.查看图表

注意事项

1.自动递增模式下，调速操作会将递增模式从自动递增切换到手动调速，并按照输入压测值进行压测

2.在压测过程中，可以随时查看系统自动生成的操作记录，也可以随时在场景操作记录面板中输入笔记内容

3.可以通过定时压测功能，定期执行压测任务，来检验核心链路

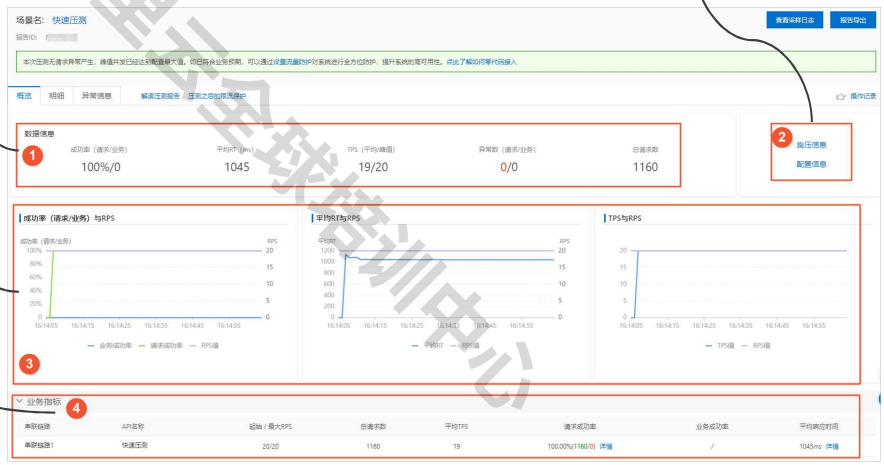
查看压测报告

当前压测场景的压力分布、压力来源等信息。

压测的主要指标，例如成功率、TPS、异常数等

重要指标的关系图和趋势图，包括请求成功率、业务成功率与RPS之间的关系和趋势等

各个串联链路和API的压测情况和主要业务指标



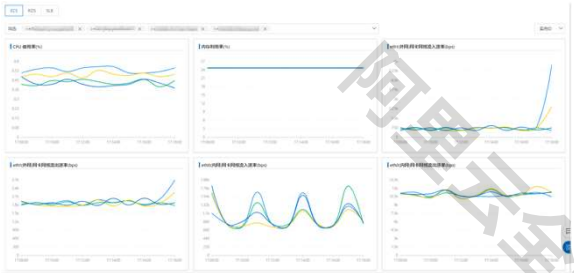
注意事项

1.选择明细页签，可以显示全场景视图和串联链路中单个API的业务详情

2.若压测过程中有异常请求，可以在异常信息页签查看错误信息和有异常API的信息

查看监控详情与采样日志

ECS、RDS、SLB云监控详情



采样日志详情

注意事项

- 若添加了架构监控，可以看到各产品的架构情况，通过点击产品图标查看该服务的实例详情

注意事项

- 采样日志中Timing仅统计请求在核心生命周期的耗时情况，方便定位问题，未覆盖请求所有阶段的耗时

课程目录04

最佳实践——电商压测场景

业务场景：

业务A：浏览产品A

业务B：登录→浏览产品B→加入购物车→提交订单

场景设置：

串联链路1：浏览产品A

串联链路2：登录及购买一干业务

链路1和链路2是并行关系

设置场景并发起压测

压测结果：

1.验证了系统的承压能力，估测系统可承受的访问压力

2.查找定位有性能问题的功能模块或接口，针对性做调优

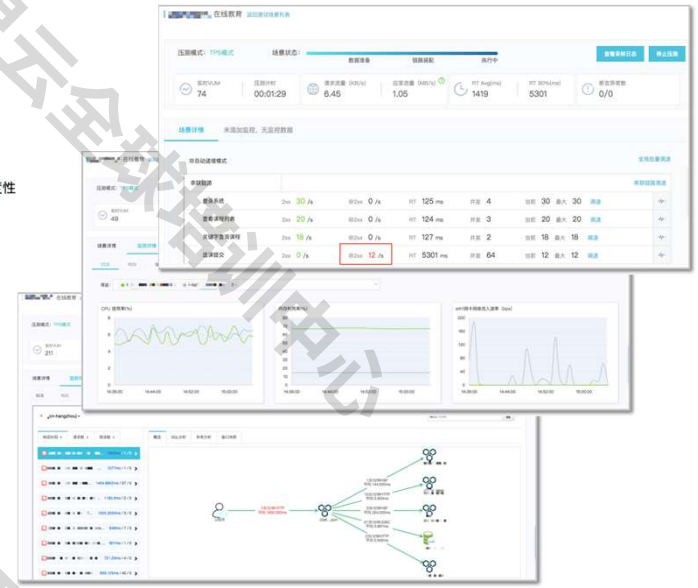
最佳实践——PTS+ARMS典型应用

PTS

- 0编码，复杂场景可视化构建。
- 云化资源按需购买，流量极速发起，过程实时调速。

ARMS

- 应用、接口、节点等多维度性能监控。
- 性能瓶颈、故障异常诊断。



A. 使用PTS快速构建高仿真业务压测

B. 通过PTS发起压测，并进行短端到端的全链路监控，了解性能水位

C. 通过ARMS集成快速找到瓶颈点，分析耗时原因

最佳实践——PTS+AHAS典型应用

PTS 与 AHAS 共同组成的压测流控方案，通过 PTS 进行系统验证，提早发现性能短板的同时，通过 AHAS 进行流量防控，进一步确保系统的稳定性。

A. 通过 PTS 构建高仿真业务压测，验证系统能够支持的最大并发请求量

B. 压测停止后，在压测报告中可查看当前系统的最大并发请求量

C. 重要接口已经达到达到了业务预期的 QPS

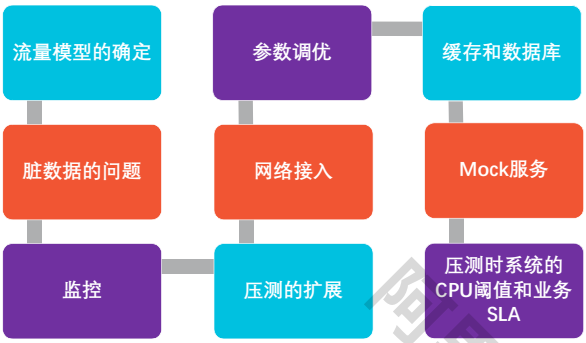
D. 默认情况下，新建的流控或降级规则立即生效。在 AHAS 流控降级监控页面，您可以实时查看该接口资源的通过的 QPS 和拒绝的 QPS

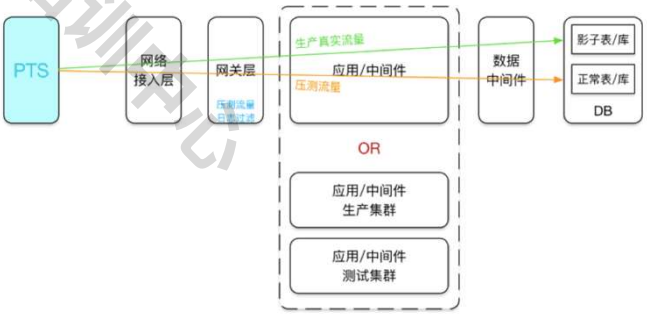
罗辑思维全链路压测分析

业务场景：

业务的知名度越高，其背后技术团队承受的压力就越大。一旦出现技术问题，就有可能被放大，尤其是当服务的是对知识获取体验要求颇高的用户群体，所以需要在生产环境中建立起一套全链路压测的验证机制，来验证各个生产环节都是能经受住各类流量的访问，成为保障服务的可用性和稳定性的重中之重。

全链路压测的构建过程：





总结及回顾

本章节主要讲述的内容：

1. 进行一次性能测试需要覆盖的内容
2. 性能测试PTS压测场景配置
   - 场景设置
   - 施压配置
   - 监控集成
   - 定时压测
3. 压测报告：PTS压测结束之后，系统会自动获取例如压测场景指标、业务详情数据、监控详情数据和API采样日志等压测过程中的数据，形成压测报告供查看和导出
4. 性能测试PTS的最佳实践

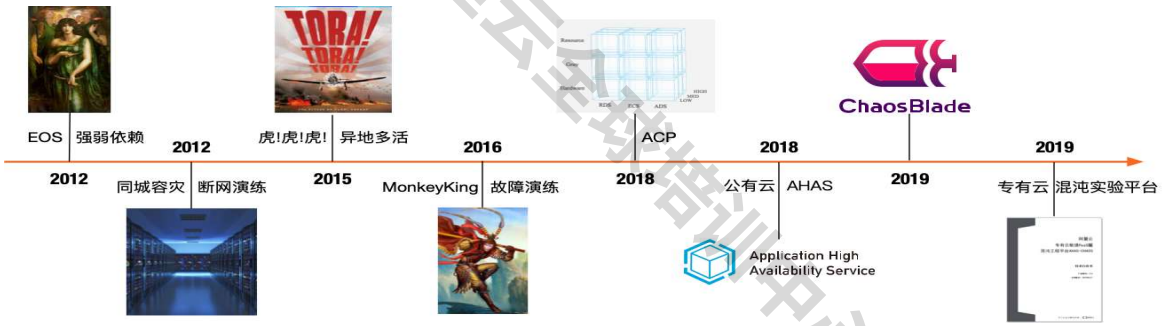
课程目标

学习完本课程后，你将能够：

1. 了解混沌工程的发展历程和目的
2. 熟悉阿里云应用高可用服务AHAS的定义和使用场景
3. 掌握阿里云应用高可用服务AHAS的功能特性和基本技术原理

课程目录01

混沌工程的发展历程



通过观察分布式系统在受控的故障注入测试中的行为变化发掘系统弱点，并针对性的改进，从而提高系统可靠性，建立系统抵御失控条件的能力的信心。所以，混沌工程并不是一个新概念，常见的异地容灾测试也是混沌工程的一种应用。

混沌工程的实践原则

建立一个围绕稳定状态行为的假说

- 关注可测量输出，而不是系统内部属性。
- 短时间内的度量结果，代表了系统的稳定状态。
- 验证系统是否工作，而不是如何工作。

多样化真实世界的事件

- 混沌变量反映了现实世界中的事件。
- 通过潜在影响或预估频率排定事件的优先级。
- 任何能够破坏稳态的事件都是混沌实验中的一个潜在变量。

在生产环境中运行实验

- 系统的行为会根据环境和流量模式有所不同。
- 为了保证系统执行方式的真实性与当前部署系统的相关性，混沌工程强烈推荐直接采用生产环境流量进行实验。

持续自动化运行实验

- 手动运行实验是劳动密集型的，最终是不可持续的，所以我们要把实验自动化并持续运行。
- 混沌工程要在系统中构建自动化的编排和分析。

最小化爆炸半径

- 在生产中进行试验可能会造成不必要的客户投诉。但混沌工程师的责任和义务是确保这些后续影响最小化且被考虑到。

混沌工程的目的

架构师：验证系统架构的容错能力

开发&运维：提高故障的应急效率

测试：提早暴露线上问题，降低故障复发率

产品&设计：提升客户使用体验

混沌工程的适用场景

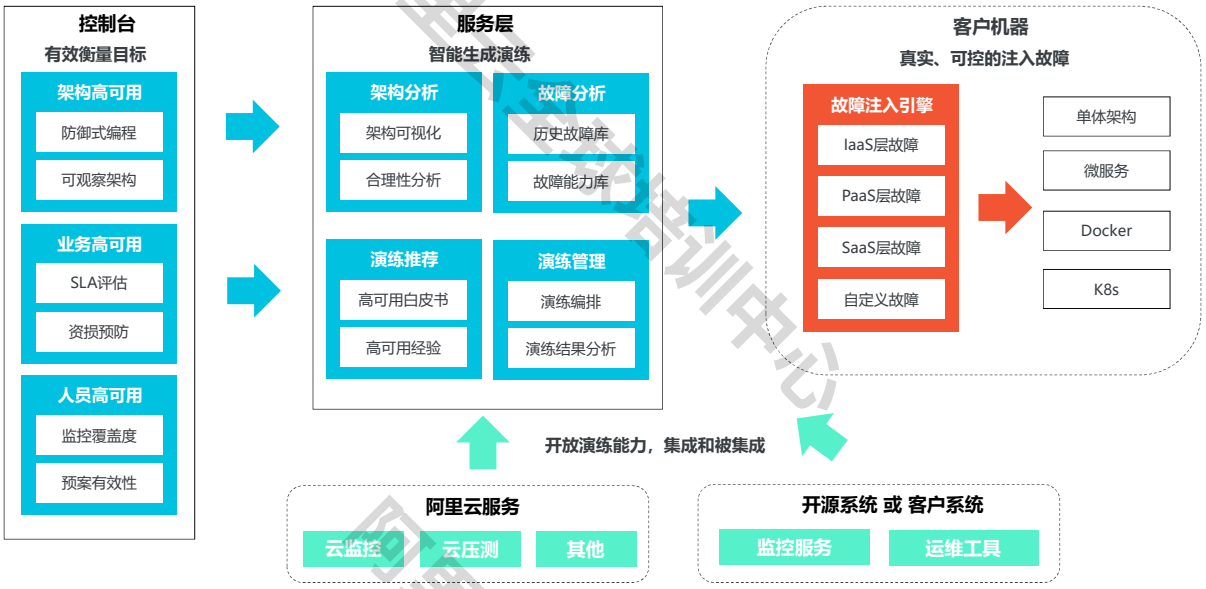
类似阿里服务化架构客户，演练应用分级是否合理，业务依赖是否合理，有无降级预案

出现过的故障是否完全修复，同样的故障在其他位置是否有可能发生

新系统、新架构上线，基于交付标准演练系统，防止系统上线后遇到预期外异常

监控覆盖率、预案有效性、参数合理性（限流、弹性、心跳……）、故障处理流程、人员应急能力

应用高可用服务AHAS：提供一套云原生+的混沌工程实现

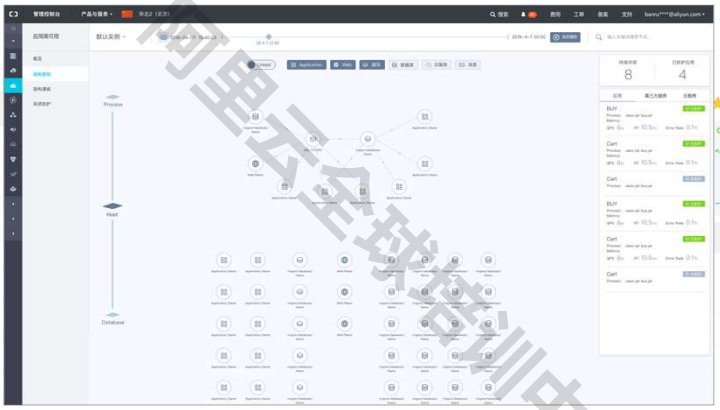


应用高可用服务AHAS的产品定义

业界首款专注于提高应用高可用能力的SaaS产品，提供应用架构自动探测，故障注入式高可用能力评测和一键应用限流降级等功能，可以让用户快速低成本的提升应用可用性

高可用评测

基于架构自动推荐故障场景验证高可用能力



架构感知

自动感知应用架构，智能识别三方组件和云服务

高可用防护

一键式流控降级保护应用可用性

应用高可用服务AHAS的使用场景

- **应用生命周期管理**

准备/架构设计 → 代码实现/编码 → 上线前验证/演练 → 应用交付 → 线上防护

- **各种应用场景对于高可用的需求**

- **流量洪峰**

秒杀、大促、下单、订单回流处理

- **API 精准流量控制**

精准控制API调用次数

- **削峰填谷**

消息类型应用，平滑消息消费速度

- **慢启动**

平滑控制流量速率，实现慢启动

应用高可用服务AHAS的产品优势

自动化

自动探测应用的架构组件和依赖关系，构建架构拓扑并持续跟踪变化

智能化

智能识别主流的三方组件和大部分阿里云服务，通过AI不断学习架构特征

全面性

来自线上真实的故障类型和演练模板，通过故障演练来全面评测应用的高可用能力

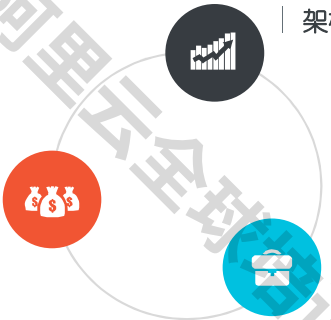
稳定性

历经双十一等技术大考的流控降级保护等防护手段，确保应用万无一失

课程目录02

AHAS 三大功能模块

应用高可用服务（Application High Availability Service）包含**架构感知、故障演练和流控降级**（即限流降级）三大独立的功能模块。



架构感知

- 自动感知应用的拓扑结构
- 以可视化的方式直观呈现应用对基础架构的依赖关系和组件间的依赖关系
- 持续记录上述依赖关系

故障演练

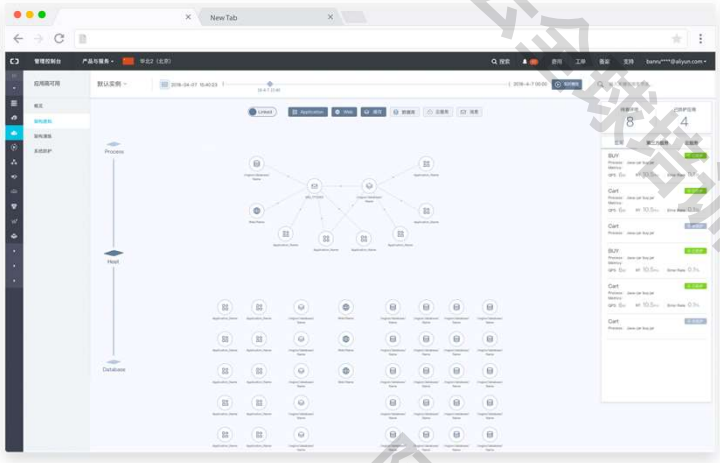
- 提供基于真实线上故障的高可用能力演练服务
- 根据您的应用架构智能推荐故障演练场景

流控降级

- 专业化多样化的限流手段
- 实时秒级的监控
- 立即生效的规则管理

功能模块一：架构感知

自动感知应用的拓扑结构，绘制组件间依赖关系和应用对基础架构的依赖，并持续记录。



第三方组件与云服务识别

自动感知识别Redis，Mysql，ZooKeeper等常用的三方组件和RDS，SLB，EDAS等云服务

Kubernetes集群感知

自动识别容器服务、Kubernetes环境中的node、Pod、service、container及其之间的依赖关系

架构感知的概念

AHAS 的架构感知模块以可视化的方式直观呈现应用对基础架构的依赖关系，以及组件间的依赖关系。

- 架构分为水平和垂直两个维度：
  - 水平架构：进程拓扑、容器拓扑、主机拓扑
  - 垂直架构：进程、容器、主机之间的依赖关系
- 架构组件是指架构的组成部分，包含进程（应用进程、第三方组件进程、云服务）、容器、主机。

- AHAS 架构感知的工作流程包括四个步骤：

数据采集 → 关系构建 → 特征识别 → 架构可视化

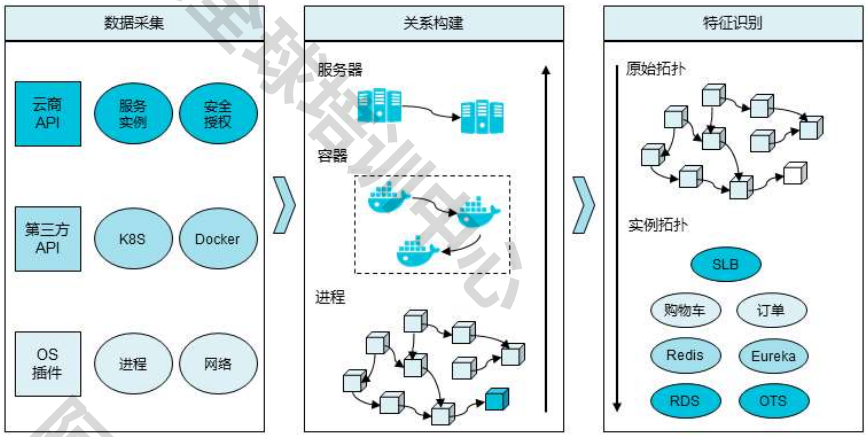
架构感知的自动探测智能识别

通过对操作系统和三方的标准接口进行**采集和分析**，**构建进程级的调用关系**，**基于特征库算法识别进程使用的技术组件**，并通过server、container、process三个维度进行**可视化架构**展示

多数据源自动采集

基于网络流量的进程间依赖关系
进程-容器-云服务器依赖关系

进程特征匹配智能学习



阿里云

架构感知支持的服务

可识别的第三方组件

- Apache
- Java
- Jetty
- Memcache
- MongoDB
- MySQL
- Ngnix
- RabbitMQ
- Redis
- Tomcat
- ZooKeeper

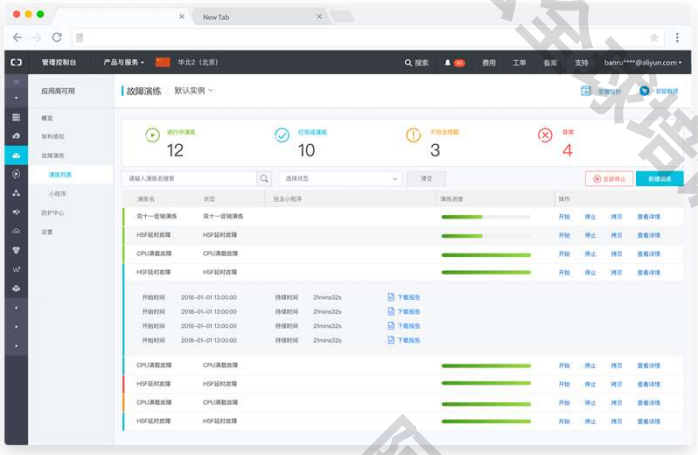
可识别的云服务

- Alibaba Cloud ACM
- Alibaba Cloud DRDS
- Alibaba Cloud Elastic Search
- Alibaba Cloud HBase
- Alibaba Cloud HiTSDB
- Alibaba Cloud KVStore
- Alibaba Cloud MQ
- Alibaba Cloud OCS
- Alibaba Cloud OSS
- Alibaba Cloud OTS
- Alibaba Cloud RDS HBase
- Alibaba Cloud RDS Memcache
- Alibaba Cloud RDS MongoDB
- Alibaba Cloud RDS MySQL
- Alibaba Cloud RDS PostgreSQL
- Alibaba Cloud RDS Redis
- Alibaba Cloud RDS SQL Server
- Alibaba Cloud SLB

阿里云

功能模块二：故障演练

提供基于真实故障的演练场景来测试应用系统的高可用能力，是混沌工程在云上的最佳实践。



小程序扩展

通过故障演练小程序，可以方便的添加各种准备，监控，检测等故障演练环节

演练库

不断积累的基于真实故障场景的演练库

通用故障梳理

Application	进程Hang	进程被杀	启动异常	心跳异常
	环境错误	包错误或损坏	配置错误、误删、获取超时	
Data	系统单点	异步阻塞同步	依赖超时	依赖异常
	业务线程池满	流控不合理	监控错误	OOM
Runtime	负载均衡失效		缓存热点	缓存限流
Middleware	数据库热点	数据库宕机	数据同步延迟	
O/S	数据库主备延迟		数据库连接满	数据库热点
	CPU抢占	内存抢占	内存错乱	上下文切换
Virtualization	服务器宕机&假死	断电	超卖	混部
Storage	磁盘满、慢、坏		不可写	不可读
Networking	网络抖动、丢包、超时	网卡满	DNS故障	断网

以Iaas，Paas，saas的维度划分

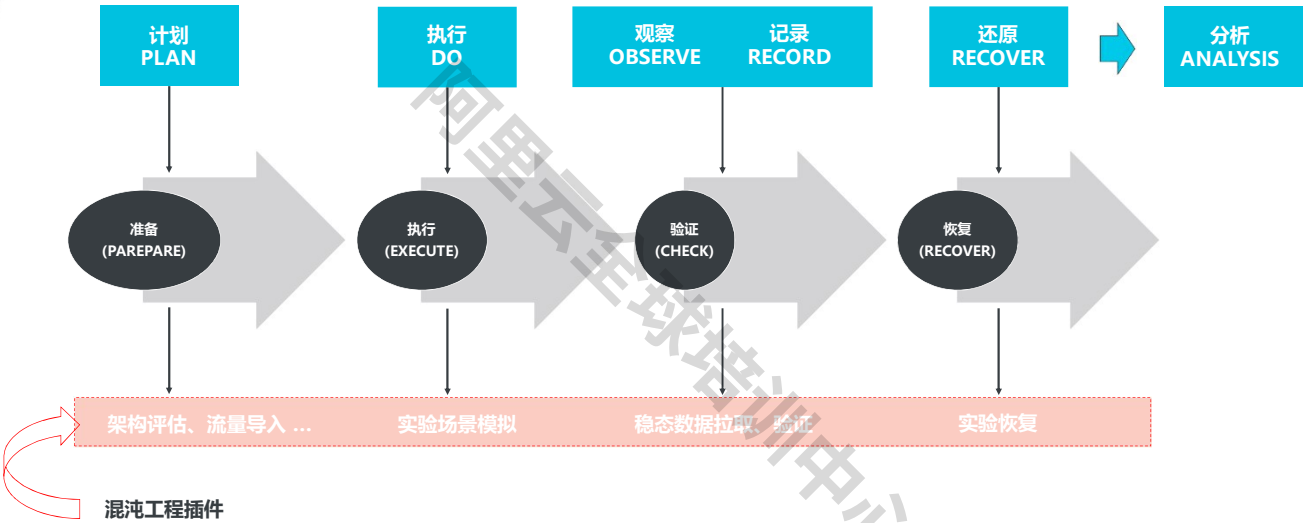
另一维度的故障画像

AHAS提供的故障场景

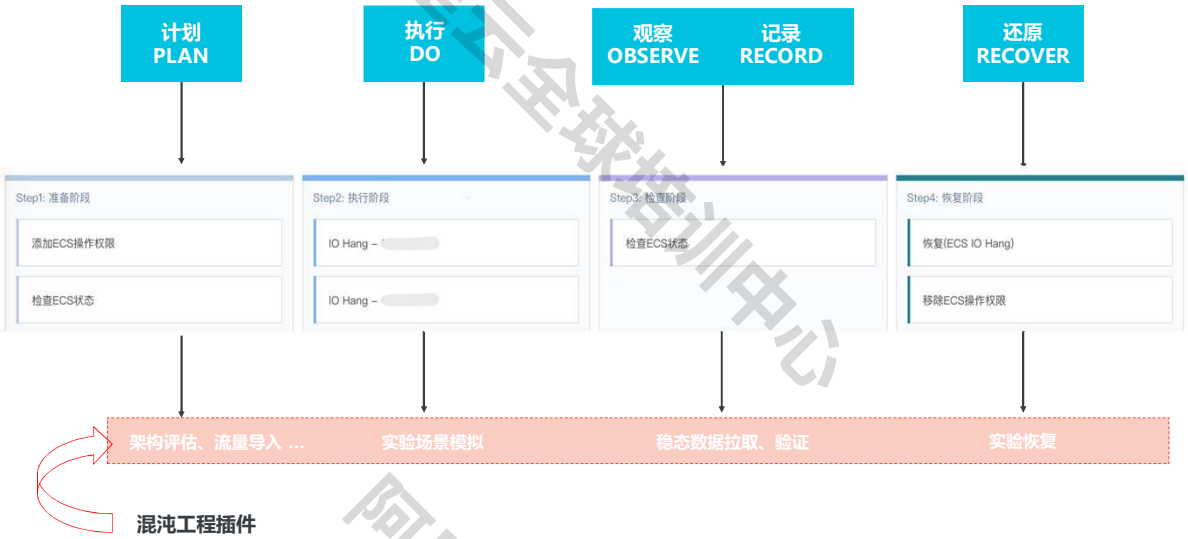
AHAS Chaos 包括了以下场景：

- 常见的基础设施资源例如CPU、内存、磁盘等。
- 应用级别的故障注入，目前只支持 Java 应用，后续将陆续推出对于 NodeJs 和 C++ 的应用故障注入。
- 云原生领域的演练场景。

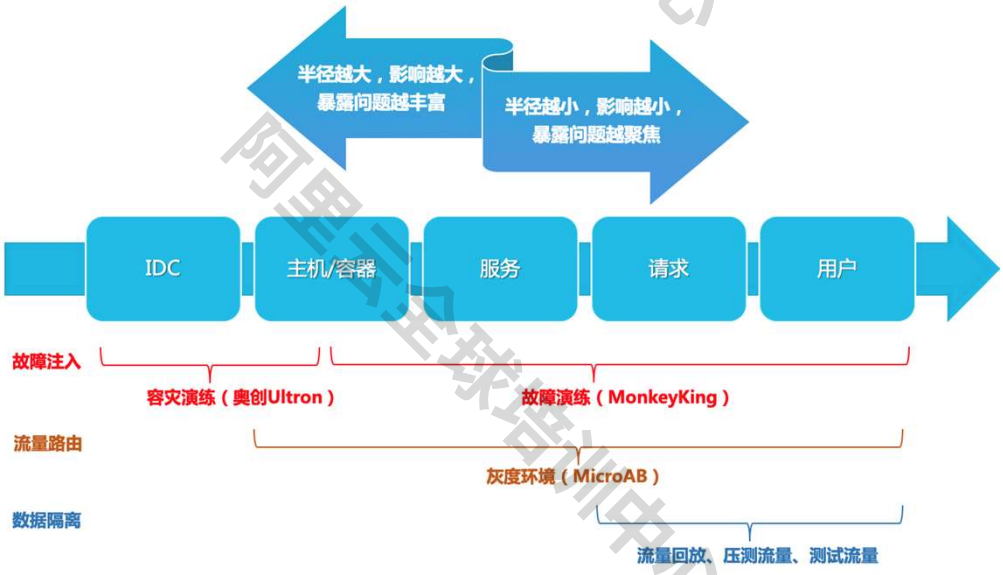
故障演练的流程



AHAS通过平台能力标准化实验流程

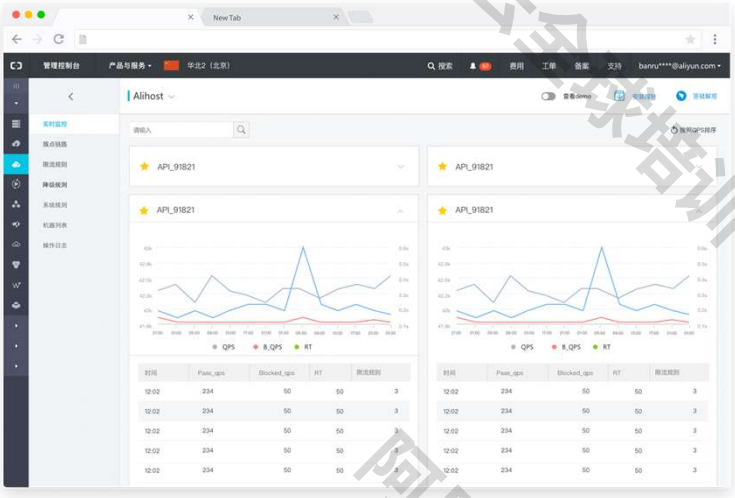


爆炸半径控制影响范围



功能模块三：流控降级

提供应用流控降级和网关限流等可用性防护手段，零代码改动，一键启用。



低成本接入

支持SDK和Agent方式接入应用，提供常用接口和自定义代码的限流降级能力

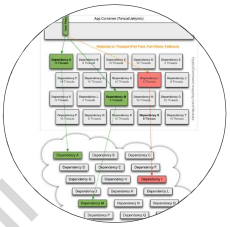
网关限流

提供SpringCloudGateway和Zuul等常用网关的限流功能

经典流量控制场景

流量控制

- 应对洪峰流量: 秒杀，大促，下单，订单回流处理
- 消息型场景: 削峰填谷，冷热启动
- 付费系统:根据使用流量付费



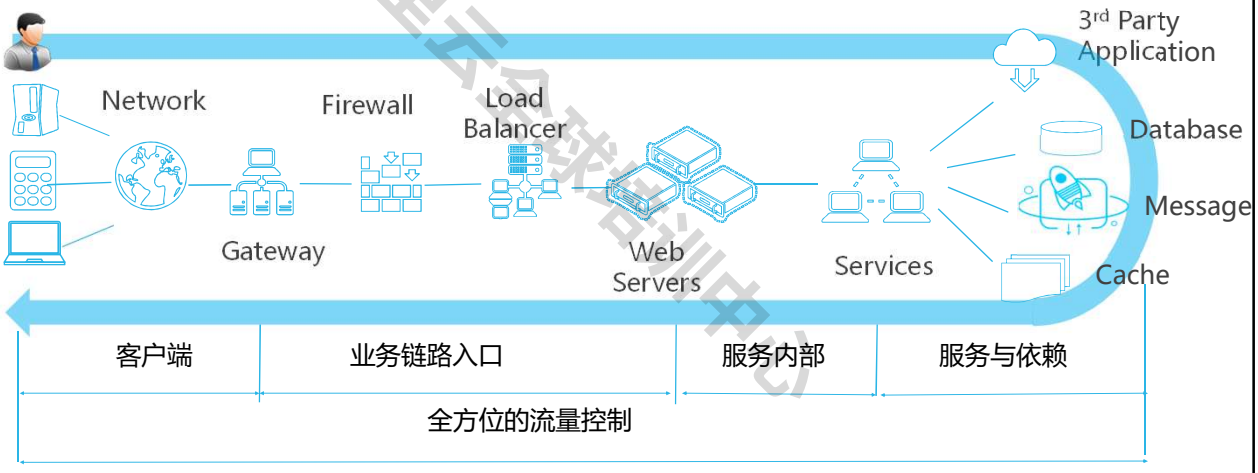
熔断降级

- 适用于任何结构复杂的应用。
- 当系统内部或者外部出现不稳定因素，迅速降级不稳定因素，让应用保持稳定

系统保护

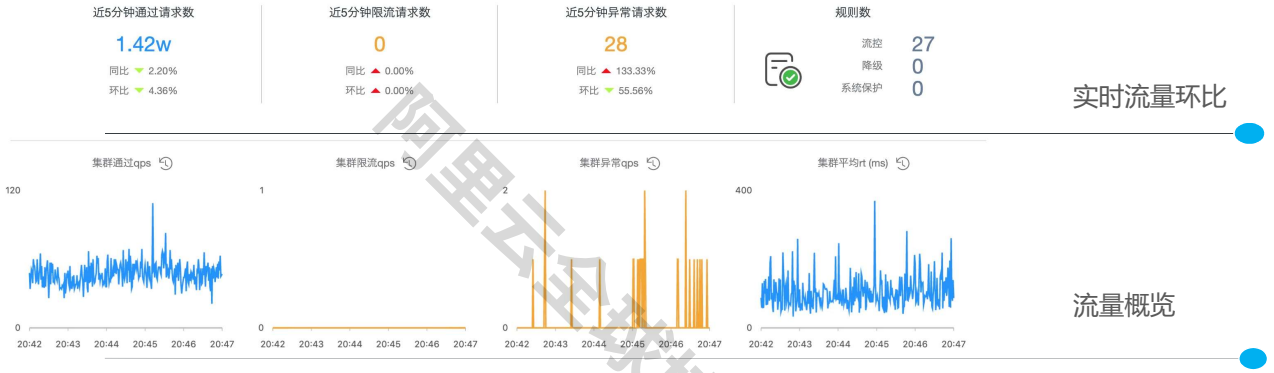
- 根据RT动态调节入口流量

典型的流量布局



- 流量实时监控
- 水位诊断分析
- 链路入口流控
- 热点漏斗
- 按照服务水位流控
- 消峰填谷
- 匀速器
- 慢SQL发现熔断
- 慢方法熔断
- 热点探测

实时秒级流量监控大盘



实时流量环比

流量概览

流量分布热力图

如何使用流量控制

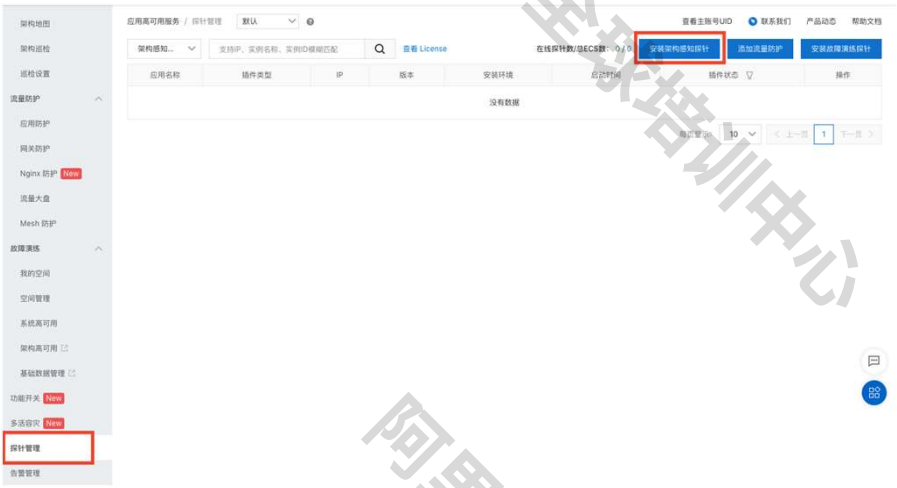
- 关键入口接口要出现在监控之中（事前）
- 校验限流体验，当限流发生的时候用户有一个好的体验（事前）
- 保证每台机器设置了负载保护（事前）
- 当发现请求响应时间过长，通过设置并发线程数来隔离异常（事中）
- 当发现请求响应时间过长，且应用不是强依赖，可以通过熔断来降级应用（事中）
- 根据服务能力来设置QPS流量保护。配合限流使用效果更佳（最佳实践）

课程目录03

架构感知的接入

1 登录 AHAS 控制台，然后在页面顶部导航栏选择地域。

2 在控制台左侧导航栏中选择**探针管理**，然后在**探针管理**页面右上角单击**安装架构感知探针**。



架构感知的接入

1 在**选择环境**页面单击**阿里云 ECS**。

2 在**安装应用高可用探针**页面安装探针。

- 若为单台主机安装探针，在目标主机右侧**操作**列单击**单击安装**。
- 若为多台主机批量安装探针，请勾选目标主机并在页面左下角单击**批量安装**。

架构巡检

1 登录 AHAS 控制台，然后在页面顶部导航栏选择地域。

2 在控制台左侧导航栏中选择架构感知 > 架构巡检。

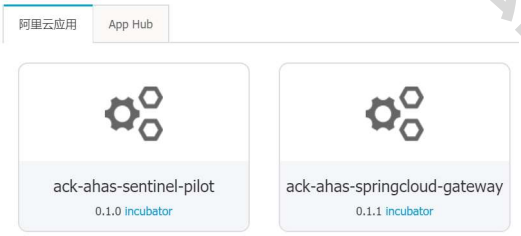
3 可选：在架构巡检页面单击再次巡检。

架构巡检

- 单击巡检不通过的诊断项，右侧面板中将列出该诊断项的巡检详情，以及影响的主机。单击主机 ID，可跳转至架构地图查看具体架构。

流量控制的接入方式（容器）

1. 应用目录安装ack-ahas-sentinel-pilot



2. 创建应用

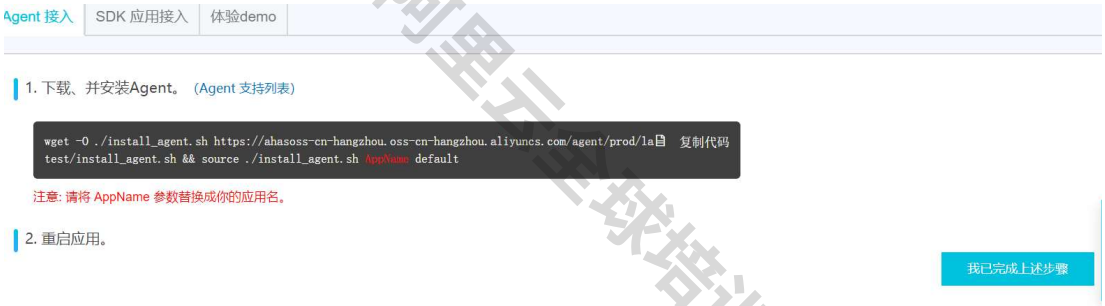
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: agent-foo
  labels:
    name: agent-foo
spec:
  replicas: 1
  selector:
    matchLabels:
      name: agent-foo
  template:
    metadata:
      labels:
        name: agent-foo
      annotations:
        ahasPilotAutoEnable: "on"
        ahasAppName: "K8sFooTest"
        #ahasLicense: "dkjfldjlfjdljlfjd"
        # ahasNamespace: "default"
    spec:
      containers:
      - name: foo
        image: registry.cn-hangzhou.aliyuncs.com/sentinel-docker-repo/foo:0.1.1
        imagePullPolicy: Always
        env:
        - name: JAVA_TOOL_OPTIONS
          value: "-Dname=env"
```


流量控制的Agent接入方式

1 登录 AHAS 控制台，然后在页面顶部导航栏选择地域。

2 在控制台左侧导航栏中选择**流量防护** > **应用防护**。

3 在应用流控右上角单击**新应用接入**。在Agent 接入 页面查看 License 信息（非公网地域不需要）。



不需要改动代码，秒级接入

适合人群场景: 运维，紧急情况

流量控制的SDK接入方式

适合人群场景: 开发，更精粒度的限流

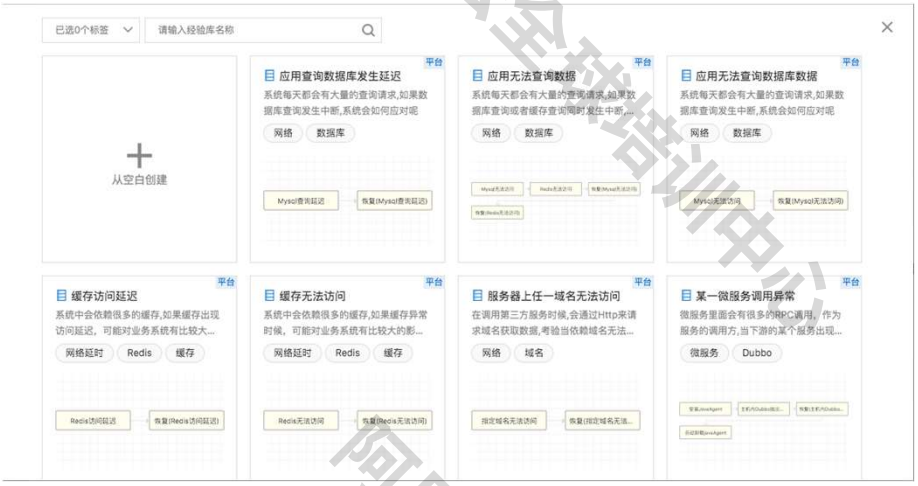
故障演练：创建演练

1 登录 AHAS 控制台，然后在页面顶部导航栏选择地域。

2 在控制台左侧导航栏中选择**故障演练** > **我的空间** ，单击**新建演练。**

故障演练：创建演练

- 在新建页面中选择创建的模板



故障演练：演练配置

- 在**演练配置**页面，填写**演练名称**、**演练描述**和**演练标签**。

故障演练：演练配置

- 配置全局配置。在**全局配置**页面完成以下配置。

故障演练：执行演练

1 在左侧导航栏中选择**故障演练** > **演练列表。**

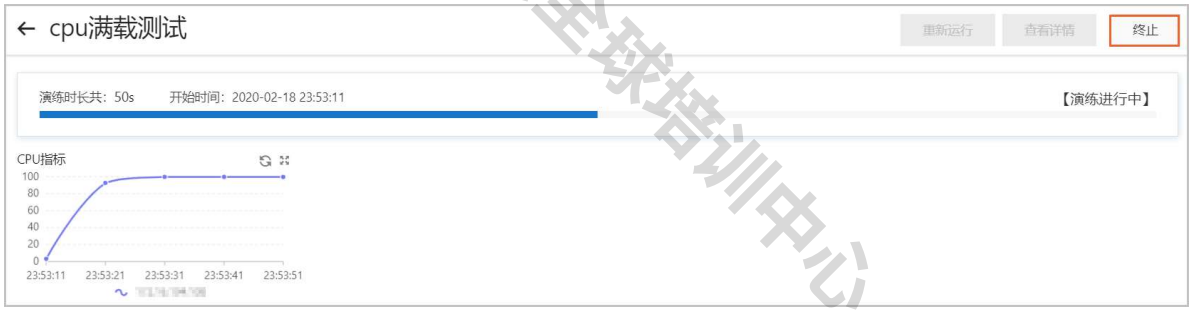
2 在**演练列表**中单击目标演练任务右侧操作栏的演练，然后在**开始执行演练**对话框中单击**确定。**

可以看出故障开始注入之后，目标机器的CPU指标开始增加，说明故障已经生效。

故障演练：停止演练

在左侧导航栏选择**故障演练 > 演练列表**，然后在**演练列表**中单击目标演练任务右侧**操作**列的**停止**。

- 在演练详情页右上角单击**终止**。



- 在演练详情页**执行情况**区域执行节点后单击 。

课程目录04

针对慢 SQL 的自动防护

对慢SQL进行自动防护的步骤：

接入 AHAS 应用流控降级 > 查看监控 > 配置慢 SQL 防护规则

步骤一：接入AHAS应用流控降级

AHAS 流控降级通过自动检测常见的 DAO 类、JDBC 驱动类等自动识别应用中的 SQL 语句，用户可以通过 Java Agent 或者 JAVA SDK 两种接入方式来实现 SQL 的监控和拦截。

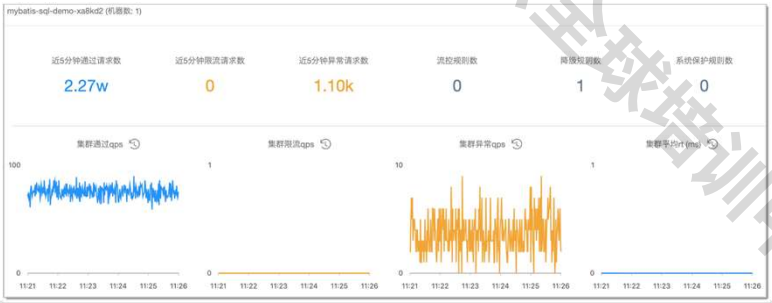
说明 其中 Java Agent 目前支持 MySQL JDBC 和 Oracle JDBC 驱动，SDK 目前支持 MyBatis 框架下的 SQL 识别。第三方组件和框架的版本支持情况详见支持列表。

针对慢 SQL 的自动防护

步骤二：查看监控

为应用接入流控降级服务后，您可以监控应用和资源 API 维度的实时数据（细化至秒级），从而评估系统的整体表现，并为流控降级规则的配置提供重要依据。具体监控指标包括 QPS、响应时间、流控降级接口数等。

1. 登录 AHAS 控制台，在控制台左上角选择应用接入的**地域**。
2. 在控制台左侧导航栏中选择**流控降级 > 应用流控降级**。
3. 在**应用列表**页面单击目标应用的资源卡片。
4. 在**应用概览**页面查看应用的限流指标详情、QPS 热力图、集群的平均 CPU 和负载和不同 SQL 语句执行的 Top 情况（包括请求 QPS Top、拒绝 QPS Top、RT Top，以及对应的单机 TOP 数据）。



针对慢 SQL 的自动防护

步骤二：查看监控

为应用接入流控降级服务后，您可以监控应用和资源 API 维度的实时数据（细化至秒级），从而评估系统的整体表现，并为流控降级规则的配置提供重要依据。具体监控指标包括 QPS、响应时间、流控降级接口数等。

5. 登录 AHAS 控制台，在控制台左上角选择应用接入的**地域**。在左侧导航栏单击监控详情，在监控详情页面查看每条 SQL 语句的调用及执行情况。

针对慢 SQL 的自动防护

步骤三：配置慢SQL防护规则

根据 AHAS 自动识别的 SQL 语句，可以对出现慢 SQL 的应用配置线程数维度的流控或降级规则，当出现慢 SQL 调用时限制同一时刻执行的 SQL 数量，防止过多的慢 SQL 语句执行把资源耗尽。

流控规则

- **直接模式**：在该模式下，阈值配置为当前 SQL 资源，超过设置的值后多余的请求将被拒绝。

针对慢 SQL 的自动防护

步骤三：配置慢SQL防护规则

根据 AHAS 自动识别的 SQL 语句，可以对出现慢 SQL 的应用配置线程数维度的流控或降级规则，当出现慢 SQL 调用时限制同一时刻执行的 SQL 数量，防止过多的慢 SQL 语句执行把资源耗尽。

流控规则

- **关联模式**：阈值配置为关联SQL资源，关联的资源请求超过设置的之后，该资源的调用将被拦截。

针对慢 SQL 的自动防护

步骤三：配置慢SQL防护规则

根据 AHAS 自动识别的 SQL 语句，可以对出现慢 SQL 的应用配置线程数维度的流控或降级规则，当出现慢 SQL 调用时限制同一时刻执行的 SQL 数量，防止过多的慢 SQL 语句执行把资源耗尽。

降级规则

慢 SQL 防护的还可以使用降级规则，根据 SQL 执行的 RT 设置对应的阈值以及时间窗口，超过指定的 RT 值后在时间窗口内 SQL 执行将被降级，抛出包装好的异常。

资源被流控降级后会报 BlockException 类异常（限流会抛流控异常 FlowException，降级会抛出降级异常 DegradeException），可以根据异常信息进行后续业务处理。

总结与回顾

本章节主要讲述内容：

1. 混沌工程的发展历程和实践原则。
2. 混沌工程的适用场景：新系统新架构上线、分布式系统依赖治理、业务连续性和故障修复验证。
3. 应用高可用服务AHAS的定义和使用场景：流量洪峰、API精准流量控制、削峰填谷和慢启动。
4. 应用高可用服务AHAS的三大功能模块：架构感知、故障演练和流控降级。
5. 应用高可用服务AHAS的使用流程和最佳实践。

阿里云