CCTC 2017 中国云计算技术大会
Cloud Computing Technology Conference 2017
CSDN

微服务架构转型的三大挑战及应对之策

王天青
首席架构师， DaoCloud云原生转型实验室

About Me

- 南京大学计算机科学与技术系硕士。
- 资深Java程序员，2003年开始从事J2EE开发，从软件开发做到架构设计。
- 08年加入EMC中国研究院，最高担任云平台主任研究员，长期从事云计算创新技术解决方案设计和实现。
- 2015年9月加入麻袋理财，任首席架构师
- 2016年12月加入DaoCloud，任首席架构师，负责云原生应用转型实验室
- 在工作期间，分别在中国和美国提交了20+专利申请，其中7项专利被美国专利局正式授权。

移动互联网时代的挑战

我要支持**100万**客户！！！

这个功能，我要**明天**上线！！！

又快又好

双十一怎么又**500**了？

公司IT**投入**怎么比业务发展快？？？

独角兽的成功秘诀

- Speed of innovation （天下武功，唯快不破）
- Always-available services （随时、随地可用）
- Web scale（从0到1，快速扩展）
- Mobile-centric user experiences （移动为王）

单体 vs. SOA vs. 微服务架构

	优势	劣势
单体	• **人所众知**：传统工具、应用和脚本都是这种结构 • **IDE友好**：Eclipse、IntelliJ等开发工具多 • **便于共享**：单个归档文件包含所有功能，便于共享 • **易于测试**：单体应用部署后，服务或特性即可展现，没有额外的依赖，测试可以立刻开始 • **容易部署**：只需将单个归档文件复制到单个目录下	• **不够灵活**：任何细修改需要将整个应用重新构建/部署，这降低了团队的灵活性和功能交付频率 • **妨碍持续交付**：单体应用比较大时，构建时间比较长，不利于频繁部署，阻碍持续交付。 • **受技术栈限制**：必须使用同一语言/工具、存储及消息，无法根据具体的场景做出其它选择 • **技术债务**：“不坏不修（Not broken，don’t fix）”， 系统设计/代码难以修改，偶合性高。
SOA	• **服务重用性**：通过编排你基本服务以用于不同的场景 • **易维护性**：单个服务的规模变小，维护相对容易 • **高可靠性**：使用消息机制及异步机制，提高了可靠性 • **高扩展和可用性**：分布式系统的特性 • **软件质量提升**：单个服务的复杂度降低 • **平台无关**：可以集成不同的系统 • **提升效率**：服务重用、降低复杂性，提升了开发效率	• **过分使用ESB**：使得系统集成过于复杂 • **使用基于SOAP协议的WS**：使得通信的额外开销很大 • **使用形式化的方式管理**：增加了服务管理的复杂度 • **需要使用可靠的ESB**：初始投资比较高
微服务	• **简单**：单个服务简单，只关注于一个业务功能。 • **团队独立性**：每个微服务可以由不同的团队独立开发。 • **松耦合**：微服务是松散耦合的。 • **平台无关性**：微服务可以用不同的语言/工具开发。 • **通信协议轻量级**：使用REST或者RPC进行服务间通信	• 运维成本过高 • 分布式系统的复杂性 • 异步，消息与并行方式使得系统的开发门槛增加 • 分布式系统的复杂性也会让系统的测试变得复杂

人人都在谈微服务，我也要上微服务

- 场景一
  - 2B场景
  - 用户数量是1W
  - 迭代周期半年
- 场景二
  - 需要集成多个系统
  - 各个系统接口不一样

业务驱动技术，技术推动业务

两个维度：创新速度 + 用户数量

快：只争朝夕

多：海量用户

创新速度

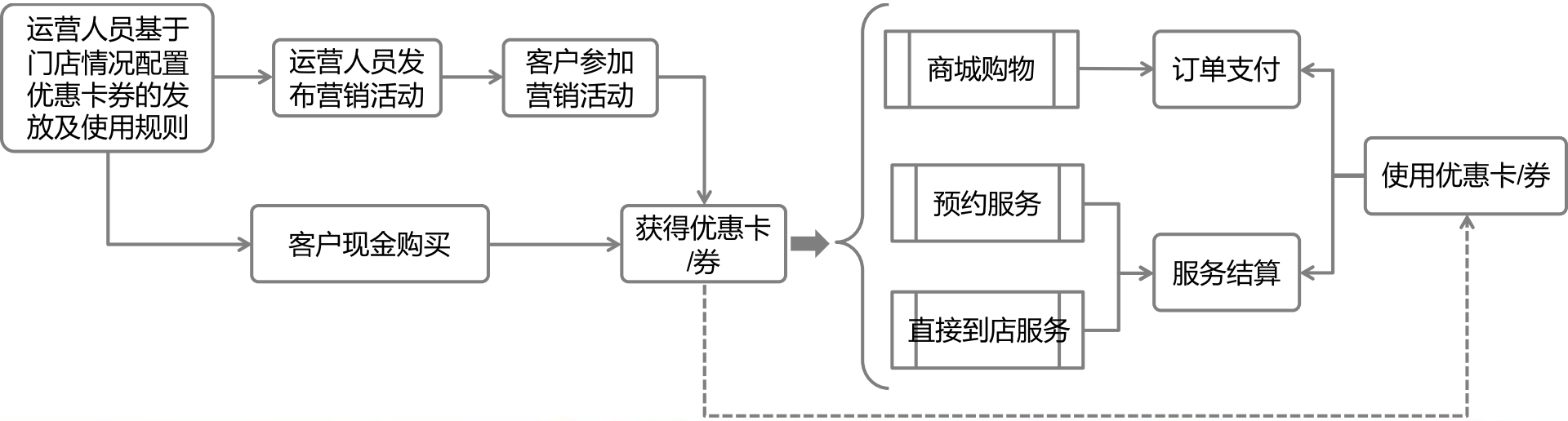
用户数量

人人都在谈微服务框架，用了就算落地了

	Dubbo	Spring Boot / Cloud
开发框架	否	是
服务调用方式	RPC	REST
服务注册	Zookeeper	Spring Cloud Netflix Eureka，Zookeeper，Consul
API网关	无	Spring Cloud Netflix Zuul
熔断器	不完善	Spring Cloud Netflix Hystrix
配置中心	无	Spring Cloud Config
分布式服务追踪	无	Spring Cloud Sleuth + Zipkin
消息总线	无	Spring Cloud Bus
社区活跃度	中	高
是否继续维护	否	是

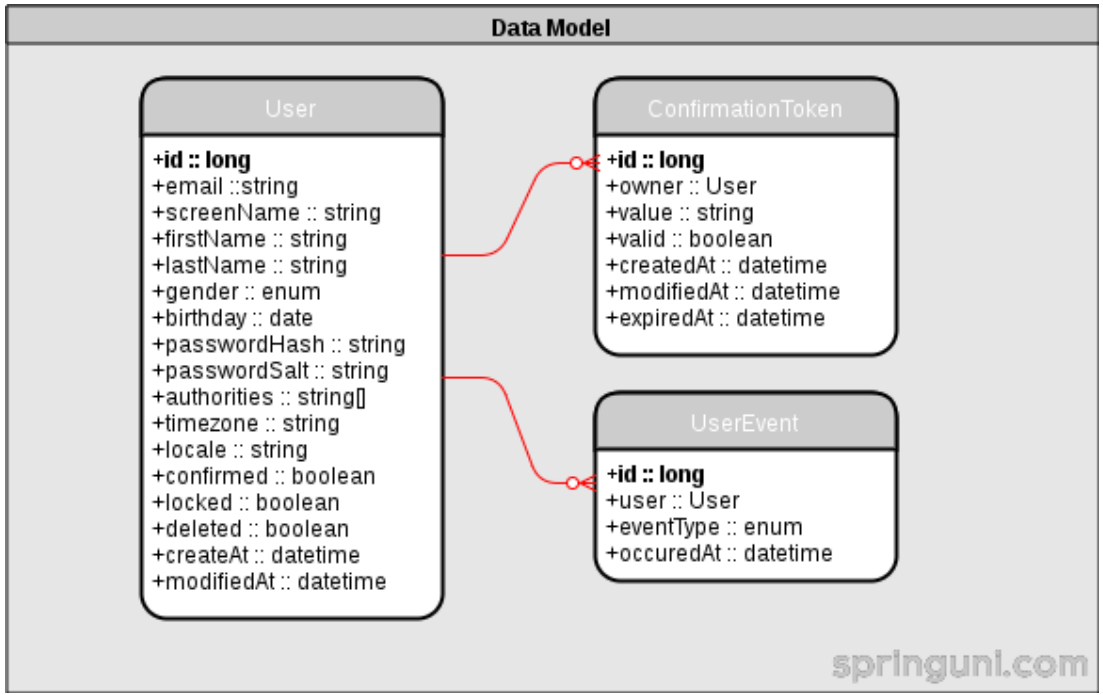
业务梳理

- 首先要理解业务，理解需求
- 需求要文档化，要明确业务场景，业务流程等信息



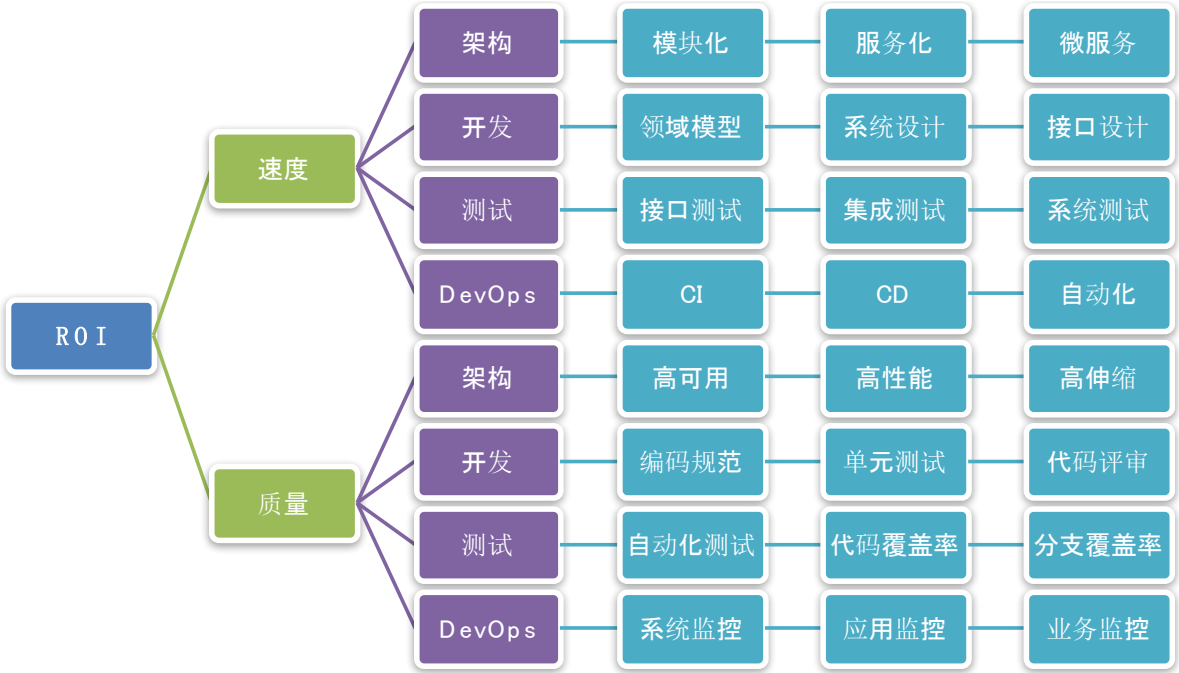
领域驱动设计（Domain Driven Design）

- 领域
- 子领域
- 上下文
- 通用语言



- 实体
- 值对象
- 聚合
- 领域事件
- 服务
- 存储库
- 模块

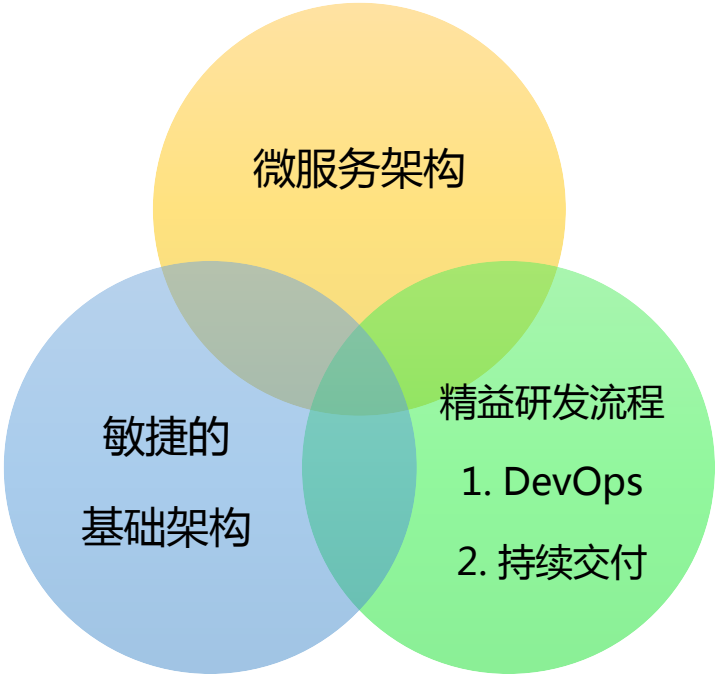
非功能性需求及关注地图



只要能够部署到生产服务器上就足够了

- 服务特性
  - 单体的项目拆分成了若干个服务
  - 服务之间启动有依赖关系
  - 服务需要依赖第三方中间件服务
- 部署方式
  - 手工或者脚本部署

黄金三角



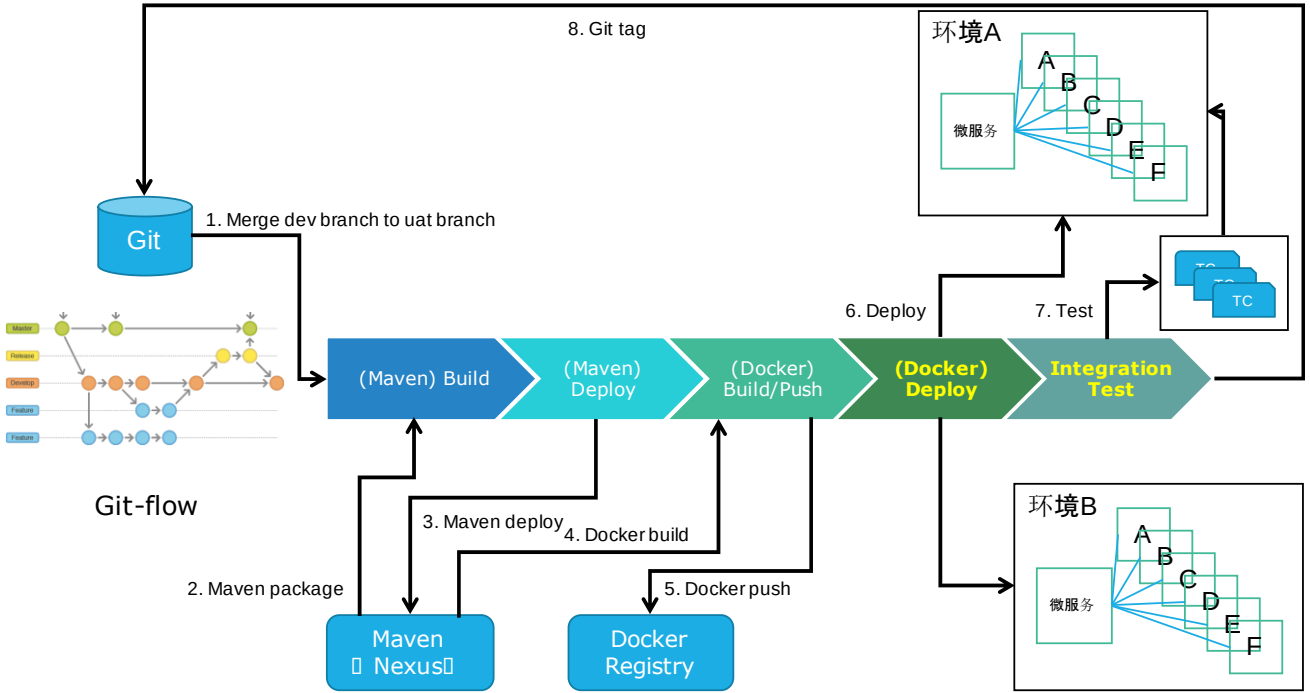
Cloud Native Application

*Cloud Native describes the patterns of high performing organizations **delivering** software **faster**, **consistently** and **reliably** at scale.（又快又好）*

***Why**, **how** and **what** of the cloud natives：*

Continuous delivery
DevOps
Microservices

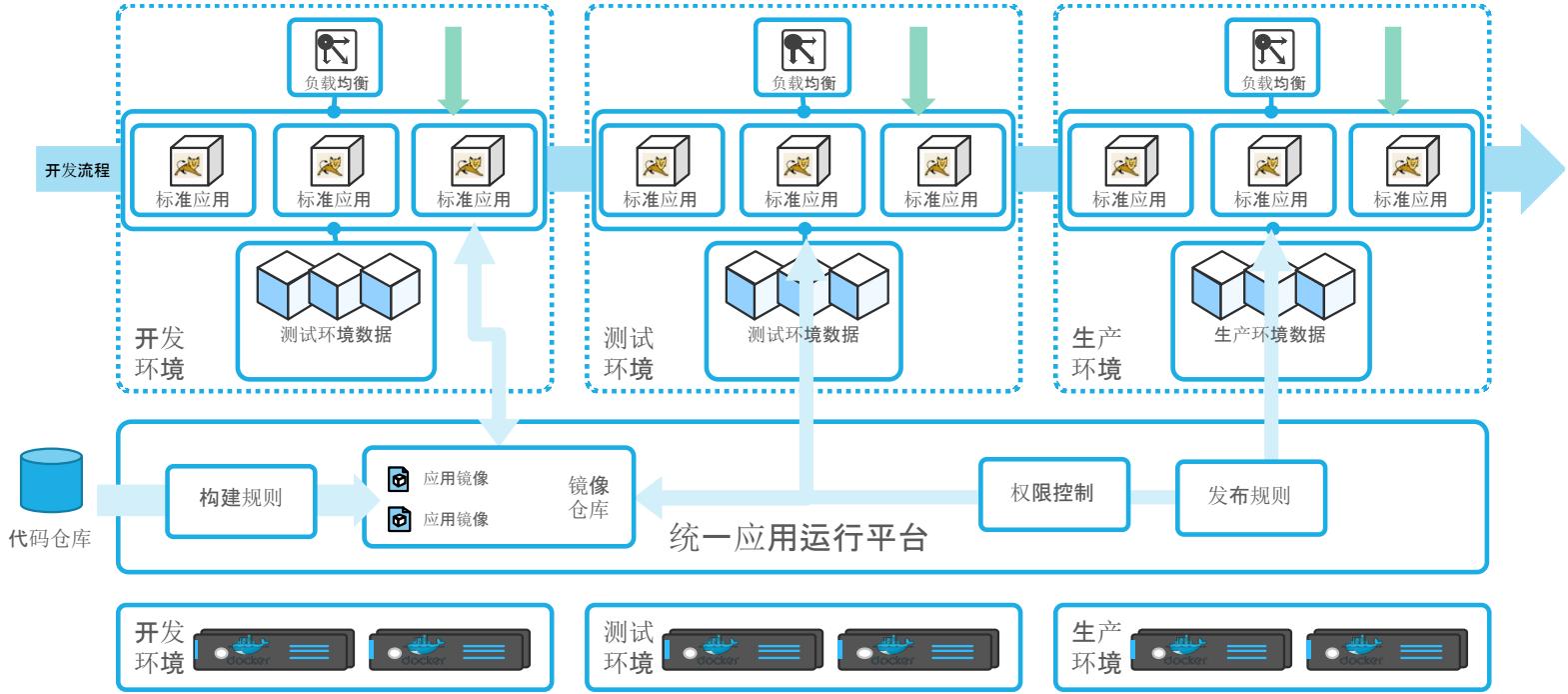
持续交付流水线


8. Git tag
环境A
A
B
C
D
E
F
微服务
Git
1. Merge dev branch to uat branch
TC
6. Deploy
7. Test
(Maven) Build
(Maven) Deploy
(Docker) Build/Push
(Docker) Deploy
Integration Test
Git-flow
3. Maven deploy
4. Docker build
2. Maven package
5. Docker push
Maven Nexus
Docker Registry
环境B
A
B
C
D
E
F
微服务

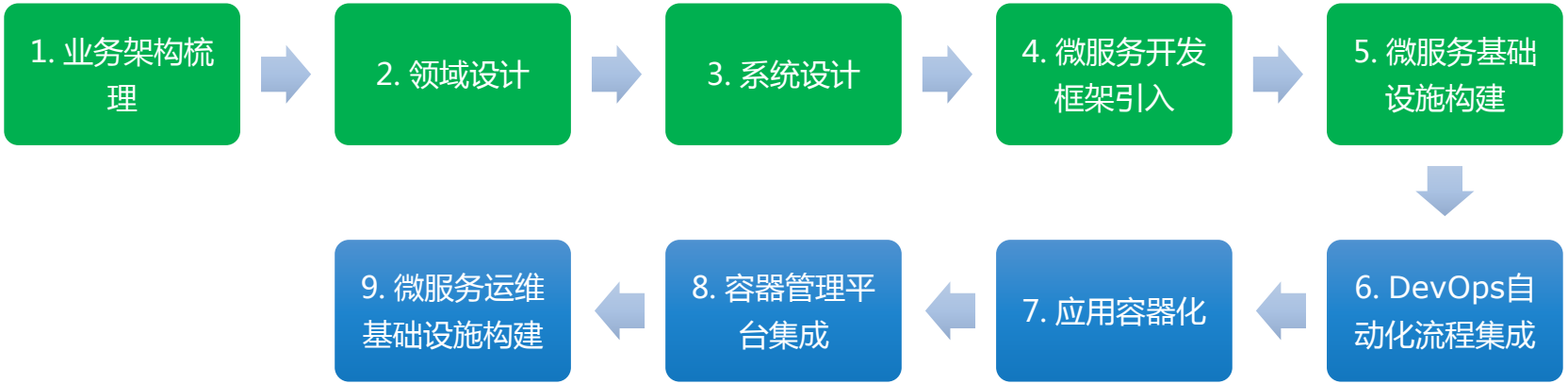
轻量级弹性容器基础架构

- 服务编排
- 服务注册与发现
- 负载均衡
- 自动伸缩
- 监控告警
- 日志

轻量级弹性容器基础架构



转型流程


1. 业务架构梳理
2. 领域设计
3. 系统设计
4. 微服务开发框架引入
5. 微服务基础设施构建
6. DevOps自动化流程集成
7. 应用容器化
8. 容器管理平台集成
9. 微服务运维基础设施构建

流程说明

业务架构梳理	•输出PRD文档，包括业务规则，业务流程图等
领域设计	•使用Domain-Driven-Design设计原则，划分系统层次，领域边界等
系统设计	•根据领域设计输出，定义不同的服务及其职责
微服务开发框架引入	•使用Spring Boot / Cloud建立服务代码框架
微服务基础设施构建	•根据需求构建配置中心，注册中心，熔断器，系统管理等微服务基础设施
DevOps自动化流程集成	•构建持续集成和持续发布体系，同时划分开发，测试，生产环境
应用容器化	•构建基础镜像和应用镜像，同时DevOps自动化流程集成对容器的支持
容器管理平台集成	•使用DCE对生产容器集群进行管理
微服务运维基础设施构建	•应用运维与容器管理平台集成

Q&A