China · Beijing

有关容器的三大路线之争

宋潇男

架构师

ArchSummit
全 球 架 构 师 峰 会 2016

[北京站]

主办方 Geekbang. 极客邦科技 InfoQ

自我介绍

我来自 PRIMETON · 普元

曾从事

You Americans, you're all engineers. You think that all the world's problems are puzzles that can be solved with money and material. You are wrong. All the world's great problems are not problems at all. They are dilemmas, and dilemmas cannot be solved. They can only be survived.

— Henry Kissinger

大纲

Docker Daemon与Systemd之争

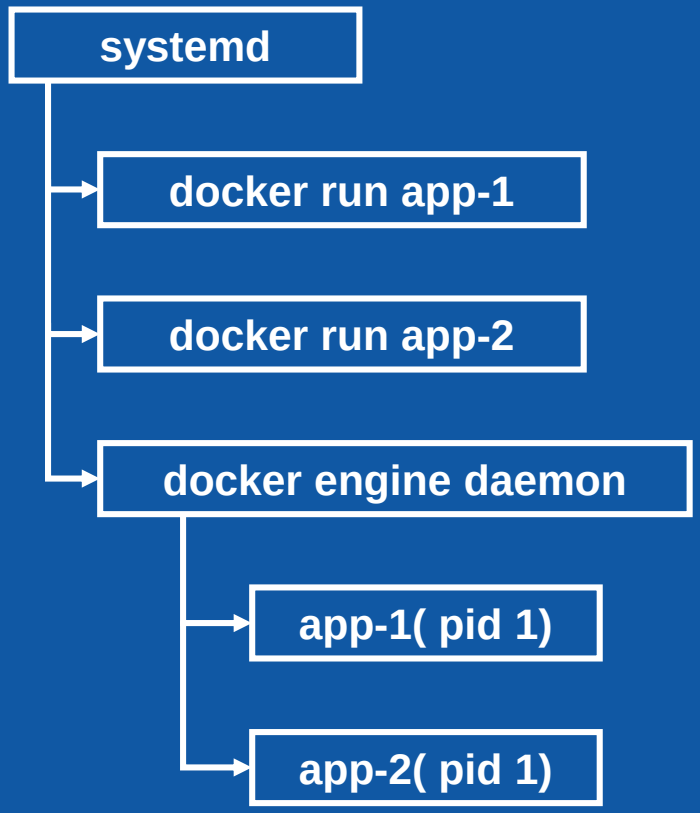
指令式运维和声明式运维之争

容器作为轻量的虚拟机还是应用管理的核心引擎？

From a security and composability perspective, the Docker process model - where everything runs through a central daemon - is fundamentally flawed.

— Alex Polvi, CEO of CoreOS

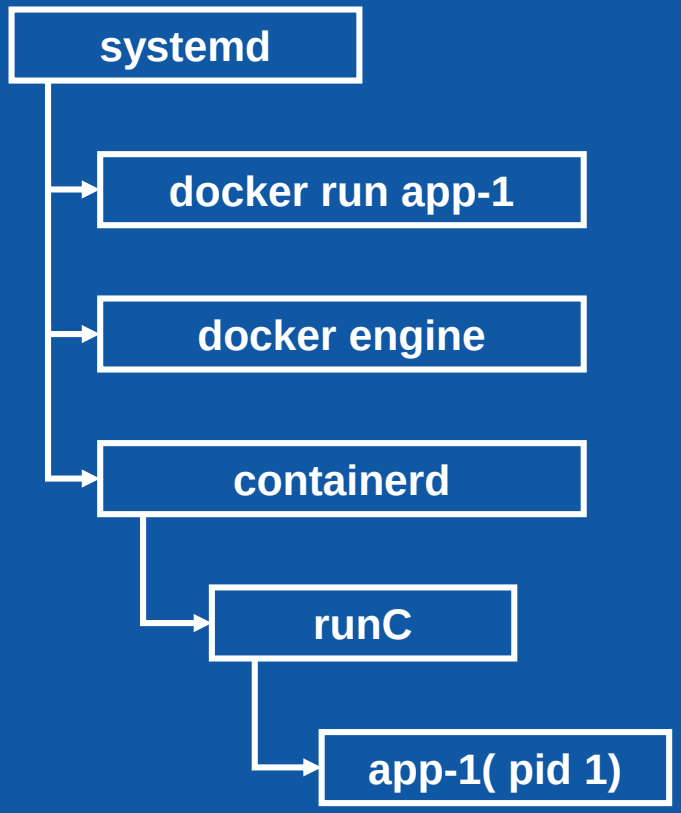
Docker < 1.11.0



问题：

- 存在单点故障
- 容器进程由docker daemon产生，破坏了Unix进程模型的自然继承关系
- 重启或升级docker会影响正在运行的容器
- 容器内没有真正的init系统，使用依赖init系统的应用时需要自行编写脚本、处理信号、回收进程和处理进程异常终止
- 无法利用sd_notify、socket activation等systemd功能

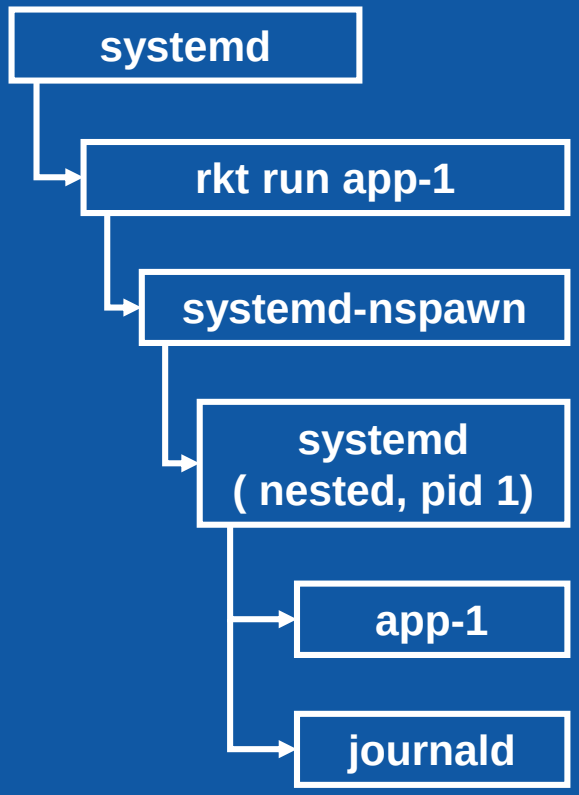
Docker 1.11.0+



问题：

- ~~存在单点故障~~
- 容器进程由containerd产生，破坏了Unix进程模型的自然继承关系
- ~~重启或升级docker会影响正在运行的容器~~
- 容器内没有真正的init系统，使用依赖init系统的应用时需要自行编写脚本、处理信号、回收进程和处理进程异常终止
- 无法利用sd_notify、socket activation等systemd功能

rkt + systemd



问题：

- ~~存在单点故障~~
- ~~容器进程由？？？产生，破坏了Unix进程模型的自然继承关系~~
- ~~重启或升级rkt会影响正在运行的容器~~
- ~~容器内没有真正的init系统，使用依赖init系统的应用时需要自行编写脚本、处理信号、回收进程和处理进程异常终止~~
- ~~无法利用sd_notify、socket activation等systemd功能~~

Is Docker Fundamentally Flawed?

Docker的考虑：

- 需要支持多种操作系统（新的、老的、Linux、Mac、Windows），并提供一致性的体验，而rkt在没有systemd的系统上会有功能缺失
- 主要面向开发人员，避免牵涉过多的复杂概念，而systemd的概念非常复杂
- 架构和rkt + systemd并无本质不同，只不过docker daemon占据了systemd的位置
- 是的，docker想成为systemd

From a security and composability perspective, the Docker process model - where everything runs through a central daemon - ~~is fundamentally flawed.~~

— *By Alex Polvi, CEO of CoreOS*

关于init系统……

- 系统启动的第一个进程，PID为1
- 以daemon的形式存在，负责产生所有其他进程，是所有其他进程的祖先，负责收养孤儿进程和收割僵尸进程
- 对signal拥有特权：如果init系统中没用signal A的handler，则不会对signal A做出响应
- Init进程退出 = 系统退出，所以一般不认为init进程为单点故障
- 常见的init系统有源于Unix的sysvinit、曾由Ubuntu牵头的Upstart、和当前被普遍采用的systemd

关于systemd……

- Linux上有史以来最具争议的项目，争议大到有人使用比特币雇佣职业杀手暗杀systemd的主要开发者Lennart Poettering
- 导致Debian社区分裂，Ian Jackson、Joey Hess等大佬退出
- 接管了太多事情，除了init，还接管了电源、设备、网络、日志、会话……当然还有我们正在讨论的容器
- Linus对systemd的看法：有很多小问题（非常愚蠢的、没有品味的……），但是大方向没错
- 我对systemd的看法：曾经略有抵触，现在觉得这玩意挺好

我的选择……

倾向于rkt + systemd，原因是：

- systemd可以为生产系统管理提供便利
- 支持pod-container结构，可以更好的管理多进程应用
- 容器（准确的说应该是pod）内部也可以有systemd，可以借助systemd管理pod内的应用
- rkt提供了更好的安全特性
- 乔布斯说兼容导致平庸，所以未来我们可能会基于docker做开发者环境，基于rkt + systemd做线上环境

大纲

Docker Daemon与Systemd之争

指令式运维和声明式运维之争

容器作为轻量的虚拟机还是应用管理的核心引擎？

一个运维萝卜

@bhatiakishore

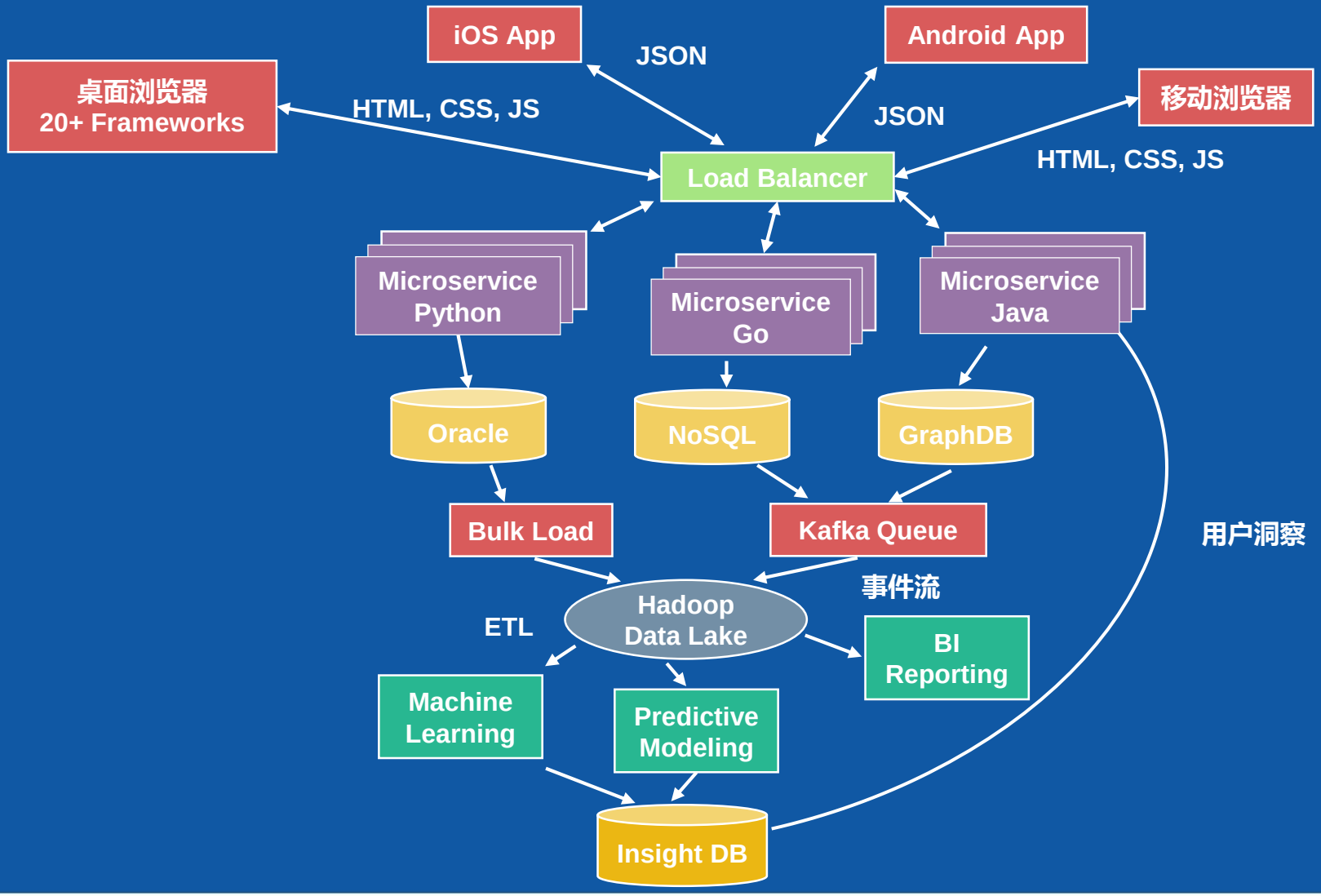
另一个运维故事

@bhatiakishore

Imperative vs. Declarative

- 指令（命令）式编程：命令机器如何去做事情
- 声明（描述）式编程：告诉机器你想要的是什么
- 相对而言，我们更喜欢声明式的工具（不仅限于编程语言）：比如SQL和各种所见即所得工具，再比如容器领域的CoreOS和Kubernetes
- 在容器和微服务流行之后，声明式思想显得格外重要

微服务架构下，指令式的运维已难以为继



指令式运维的问题

- 需要step-by-step的编写脚本，需要设想目标环境的各种状况、处理各种edge cases，高度依赖编写者的经验
- 脚本在不同的环境里运行可能会产生不同的结果
- 编写脚本时需要假定目标环境保持稳定，这在分布式共享环境下不太容易实现
- 运维流程很难具备事务性，如果脚本在执行过程中被意外情况打断，很可能产生意想不到的中间状态
- 只有到线上环境查看才能确切了解部署结果，想了解上次部署的结果，对不起，可能已经被覆盖了……
- 需要另行编写文档描述运维流程，如果需要多人维护同一个脚本，协作往往会非常困难

声明式运维的好处

- 使用配置文件直接描述最终状态，不必考虑流程和目标环境细节，易于编写，甚至可以由开发人员直接编写
- 易于理解，易于多人共同维护和code review
- 易于文档化和版本管理
- 由于配置文件直接描述了最终状态，那么配置文件版本化既是部署结果的版本化
- 重复部署不会产生不一致的结果
- 在不同的目标环境下可以产生一致的部署结果
- 天然具备事务性，要么成功，要么什么都不做
- 天然符合不可变基础设施理念

以Kubernetes为例，说明声明式运维思想

What is Kubernetes?

Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure.

……

Additionally, Kubernetes is not a mere “orchestration system”; it eliminates the need for orchestration. The technical definition of “orchestration” is execution of a defined workflow: do A, then B, then C. In contrast, Kubernetes is comprised of a set of independent, composable control processes that continuously drive current state towards the provided desired state. It shouldn’t matter how you get from A to C: make it so. Centralized control is also not required; the approach is more akin to “choreography”. This results in a system that is easier to use and more powerful, robust, resilient, and extensible.

Kubernetes的设计原则

Design principles

Declarative > imperative: State your desired results, let the system actuate

Control loops: Observe, rectify, repeat

Simple > Complex: Try to do as little as possible

Modularity: Components, interfaces, & plugins

Legacy compatible: Requiring apps to change is a non-starter

Network-centric: IP addresses are cheap

No grouping: Labels are the only groups

Bulk > hand-crafted: Manage your workload in bulk

Open > Closed: Open Source, standards, REST, JSON, etc.

Observe -> Diff -> Act

Control loops

Drive **current state** -> **desired state**

Act independently

APIs - **no shortcuts** or back doors

Observed state is truth

Recurring pattern in the system

Example: ReplicationController

Control Loop

Replication Controllers

A type of *controller* (control loop)

Ensure N copies of a pod always running

- if too few, start new ones
- if too many, kill some
- group == selector

Cleanly layered on top of the core

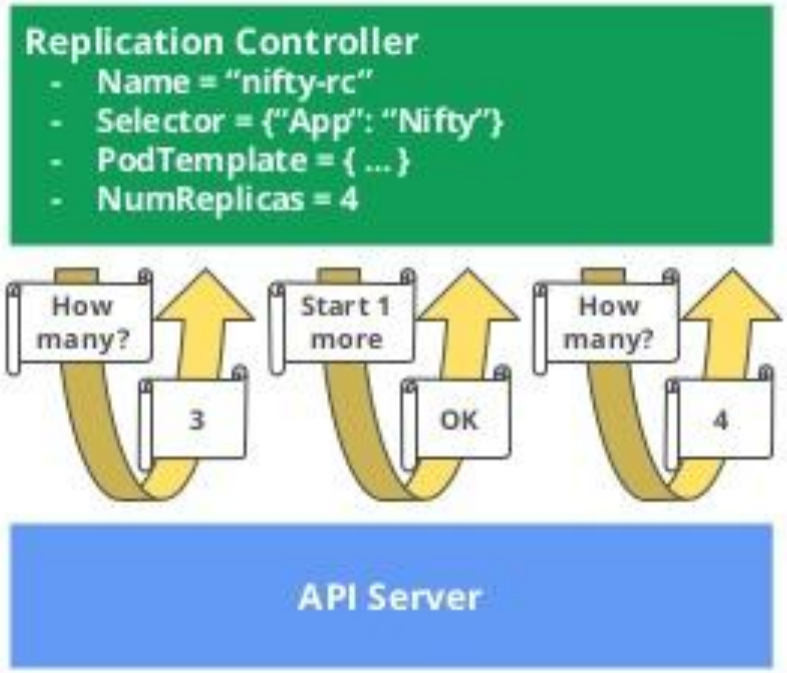
- all access is by public APIs

Replicated pods are fungible

- No implied ordinality or identity

Other kinds of controllers coming

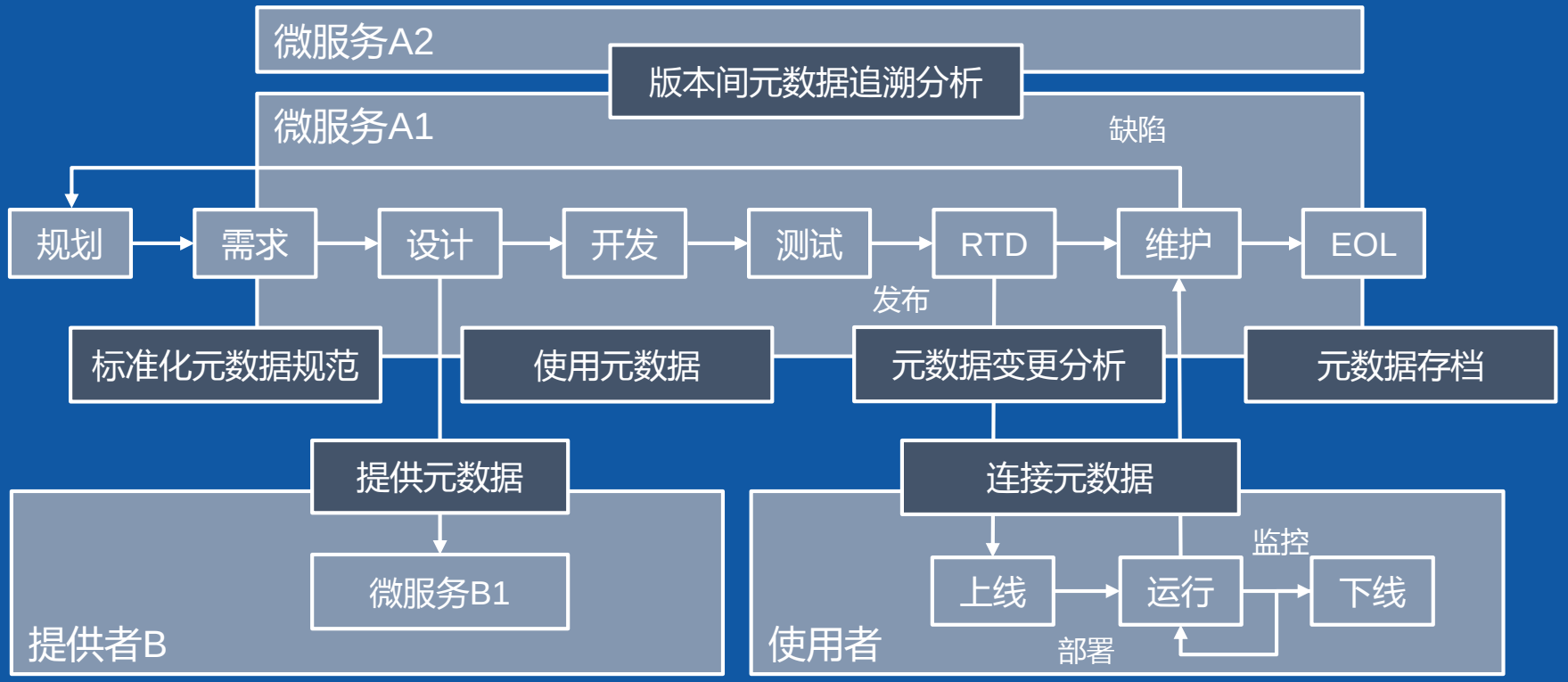
- e.g. job controller for batch



@puja108

声明信息既为系统元数据

DevOps的关键之一：有效地识别和管理元数据，以便高效、高质量的将系统动态组装并运行起来



Orchestration vs. Choreography

- Orchestration：为管弦乐配器，中文为编配
- Choreography：（尤指芭蕾舞的）编舞艺术，中文为编排
- Orchestration指的是自动执行一个工作流，即用某种语言定义工作流，并让引擎执行这一工作流。Orchestration并不描述参与方之间的协调交互。
- Choreography指的是对参与方之间协调交互的描述，是一种交互的议定模型

大纲

Docker Daemon与Systemd之争

指令式运维和声明式运维之争

容器作为轻量的虚拟机还是应用管理的核心引擎？

While the adoption of VMs is driven by IT operations management, the adoption of containers is driven primarily by developers, both in the name of "agility."

Gartner: Virtual Machines and Containers Solve Different Problems

Java EE 与容器

- Java EE是一个标准化的交付形态、一个通用的运行环境和基础架构，并由此衍生出一个完整的IT产业链
- 容器也是一个标准化的交付形态、一个通用的运行环境和基础架构，并即将由此衍生出一个完整的IT产业链

Java EE的劣势

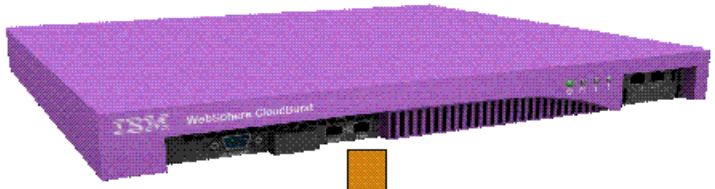
严重依赖于应用服务器，但是应用服务器存在如下问题：

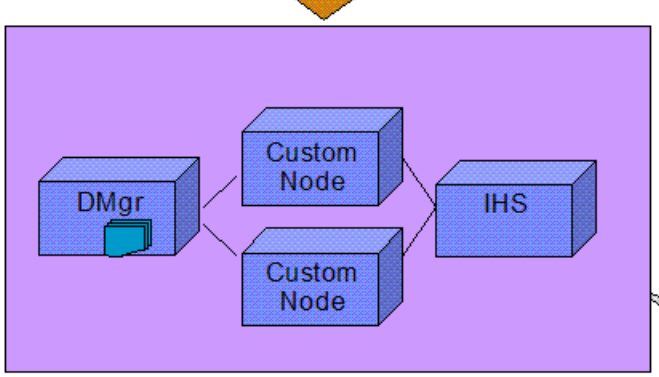
- 资源隔离差：无法对CPU、内存、文件系统、IO进行隔离
- 依赖管理差：内部经常jar版本冲突，外部只能管管数据库和Web服务器
- 与应用紧耦合：需要为应用做针对性的配置和优化
- CI/CD不友好：体积庞大，配置复杂
- ……

WebSphere Cloudburst Appliance

Environment profiles give users more control over these steps

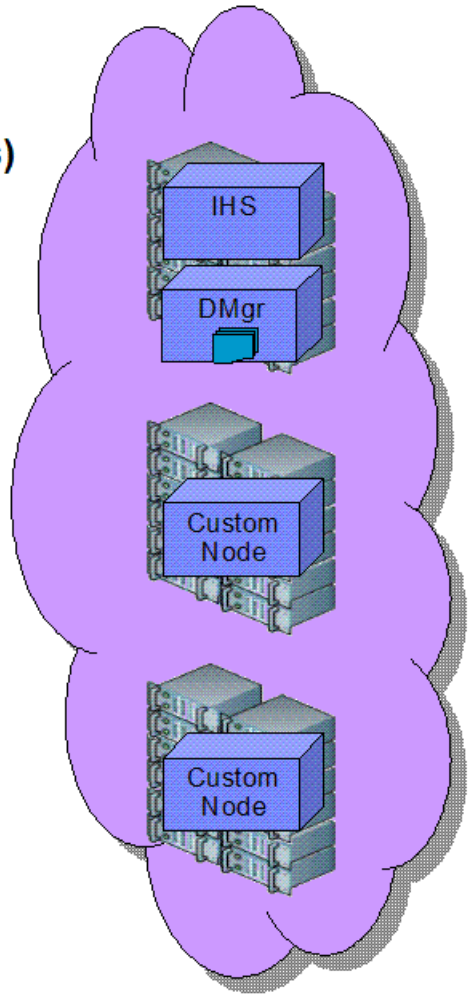
1. Choose hypervisor(s)
2. Create virtual machines
3. Inject IP addresses
4. Start VMs and WAS
5. Run scripts





Placement of Virtual Images

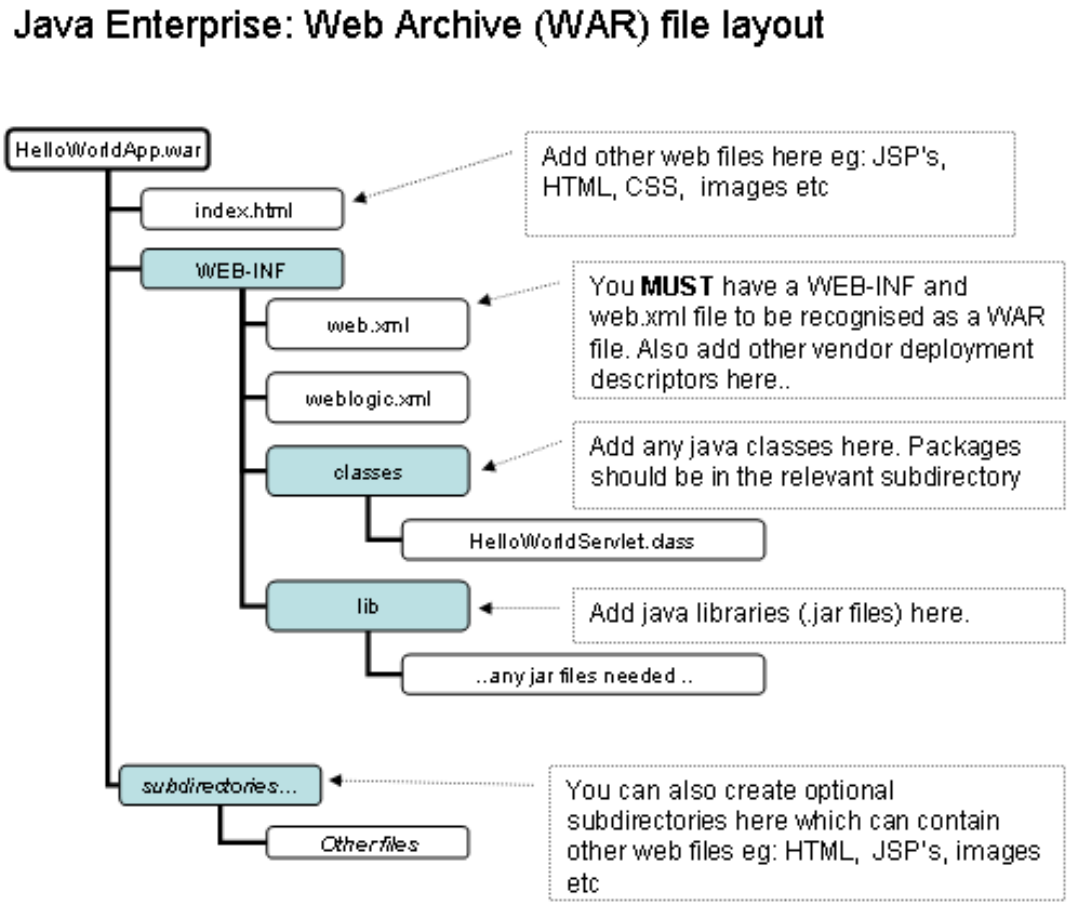
Pattern

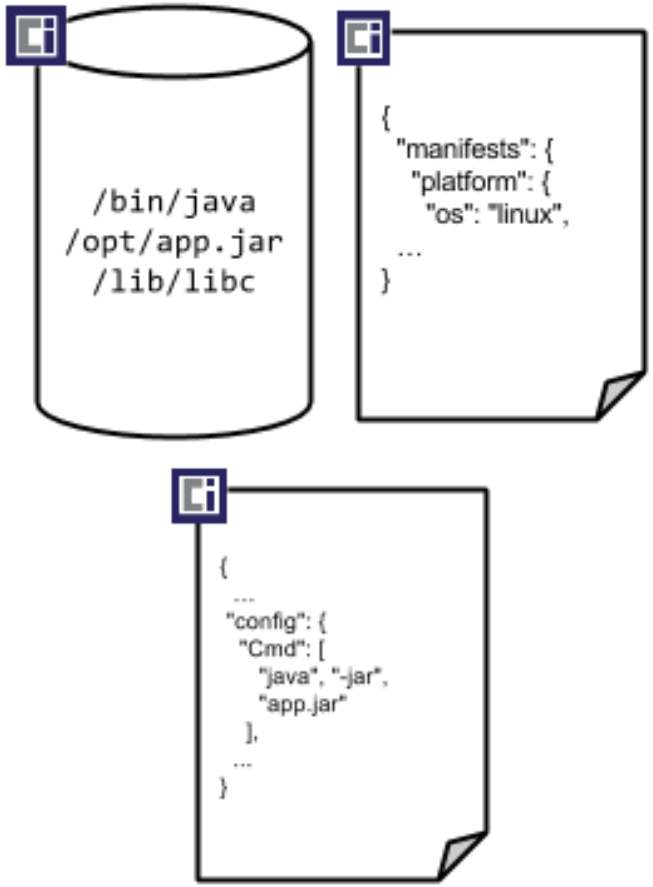


而容器和容器管理平台的到来

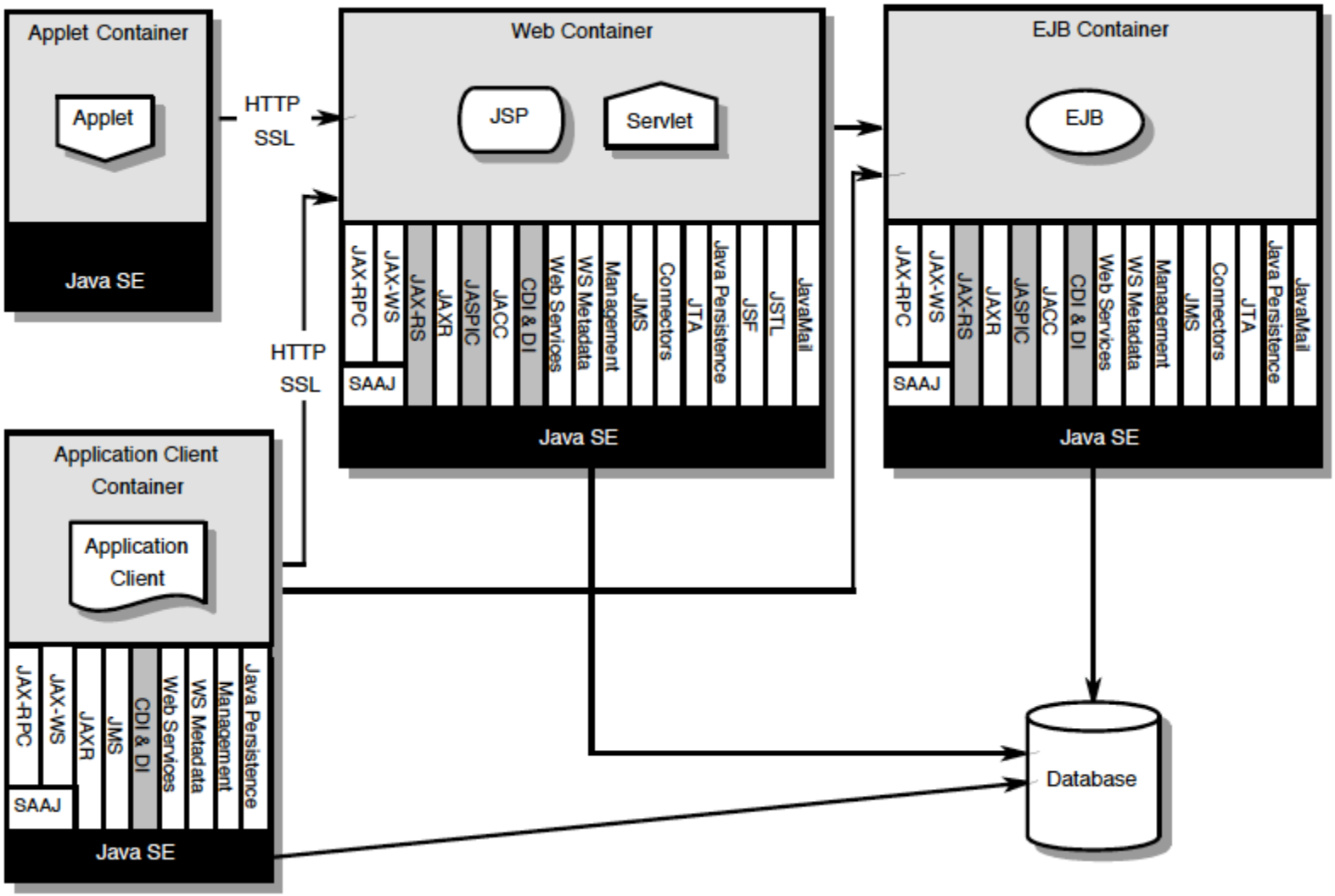
完美的解决了这些问题

WAR 与 OCI Image

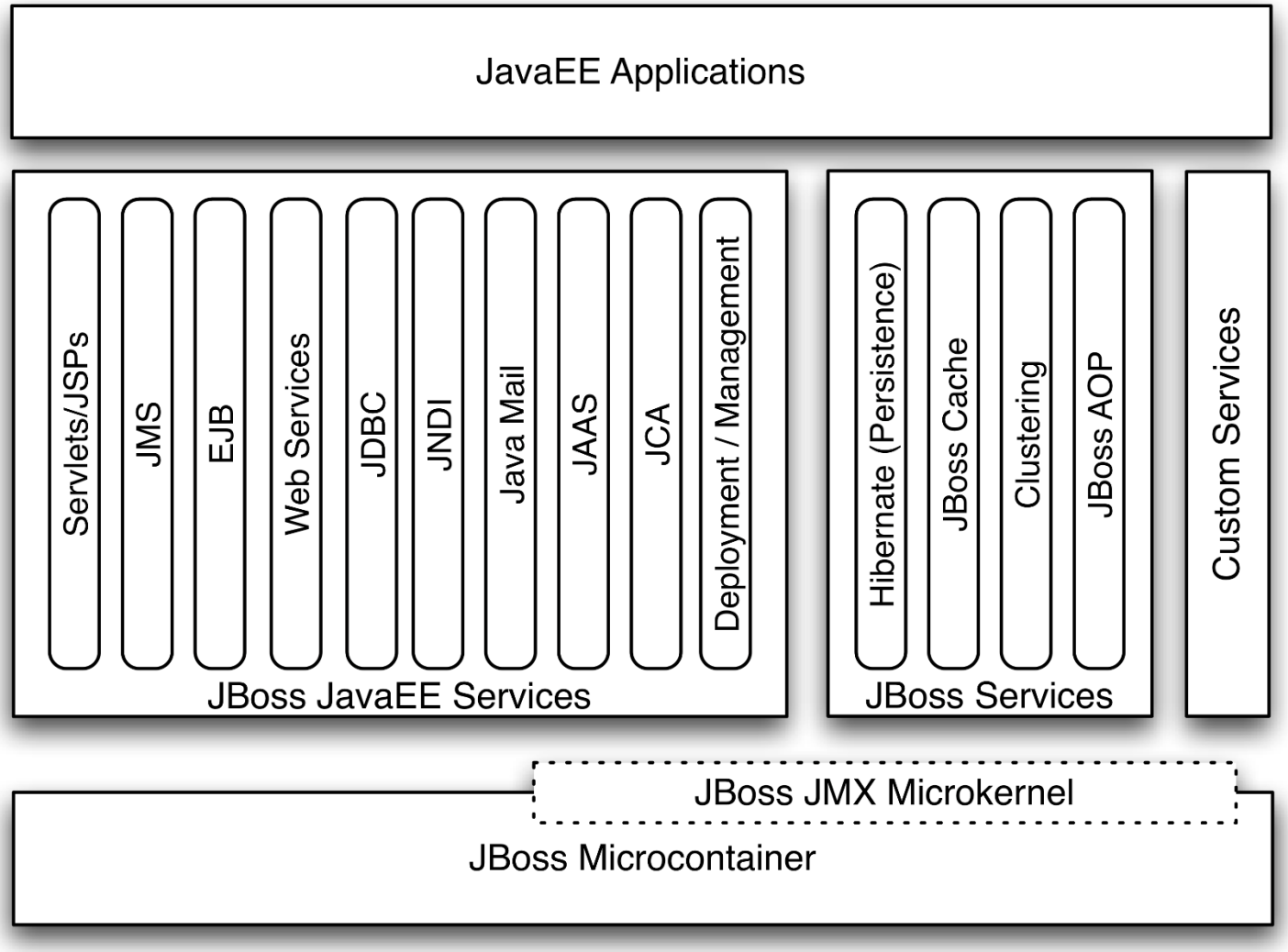




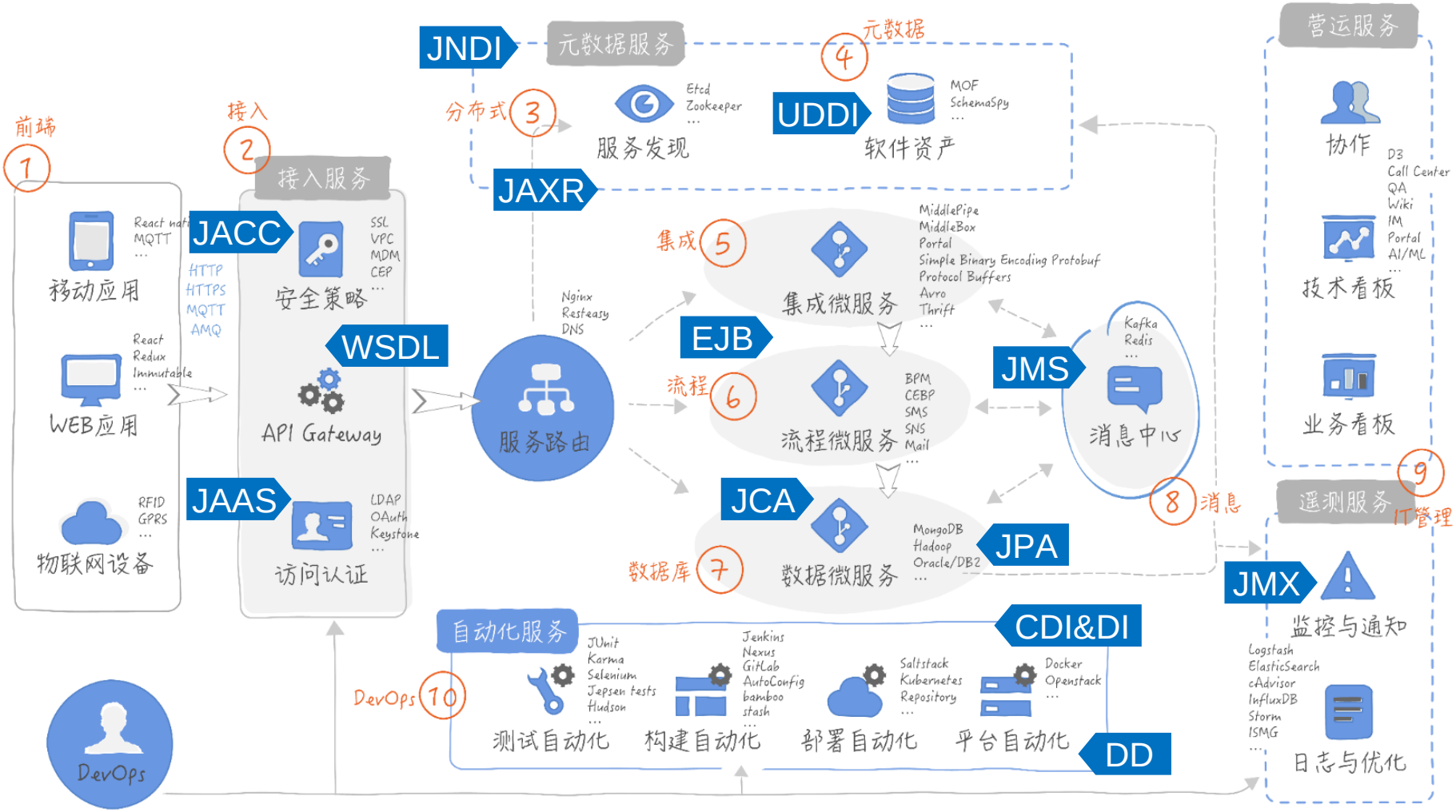
Java EE Architecture



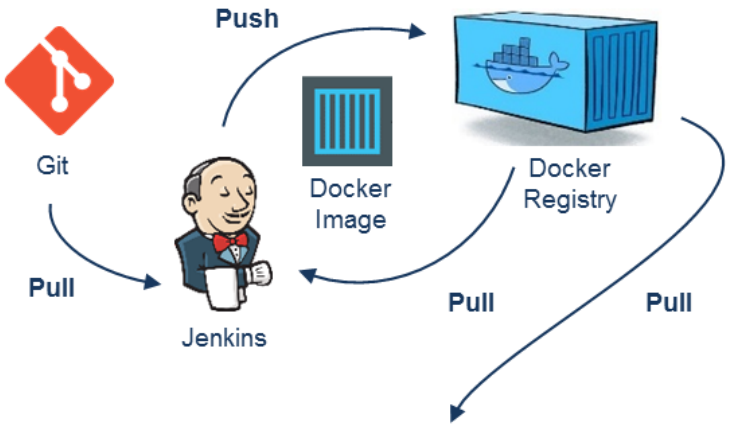
Jboss Architecture



Primeton The Platform Architecture



你只需上传你的镜像包




```
$ kubectl rollout history app/crm
deployments "crm":
REVISION      CHANGE-CAUSE
…
7             kubectl set image app/crm app=crm:1.1
8             kubectl set image app/crm app=crm:1.2
$ kubectl rollout undo app/crm --to-revision=7
deployment "crm" rolled back
```

THANKS

ArchSummit
全 球 架 构 师 峰 会 2016

[北京站]

主办方 Geekbang. 极客邦科技 InfoQ