



云原生技术在2B软件交付的实践



曾庆国

北京好雨科技有限公司
技术负责人



GopherChina 2021

目 录

2B软件交付的困局

01

云原生与云原生应用

02

面向交付的应用模型

03

2B交付版本的DevOps

04

第一部分

2B软件交付的困局



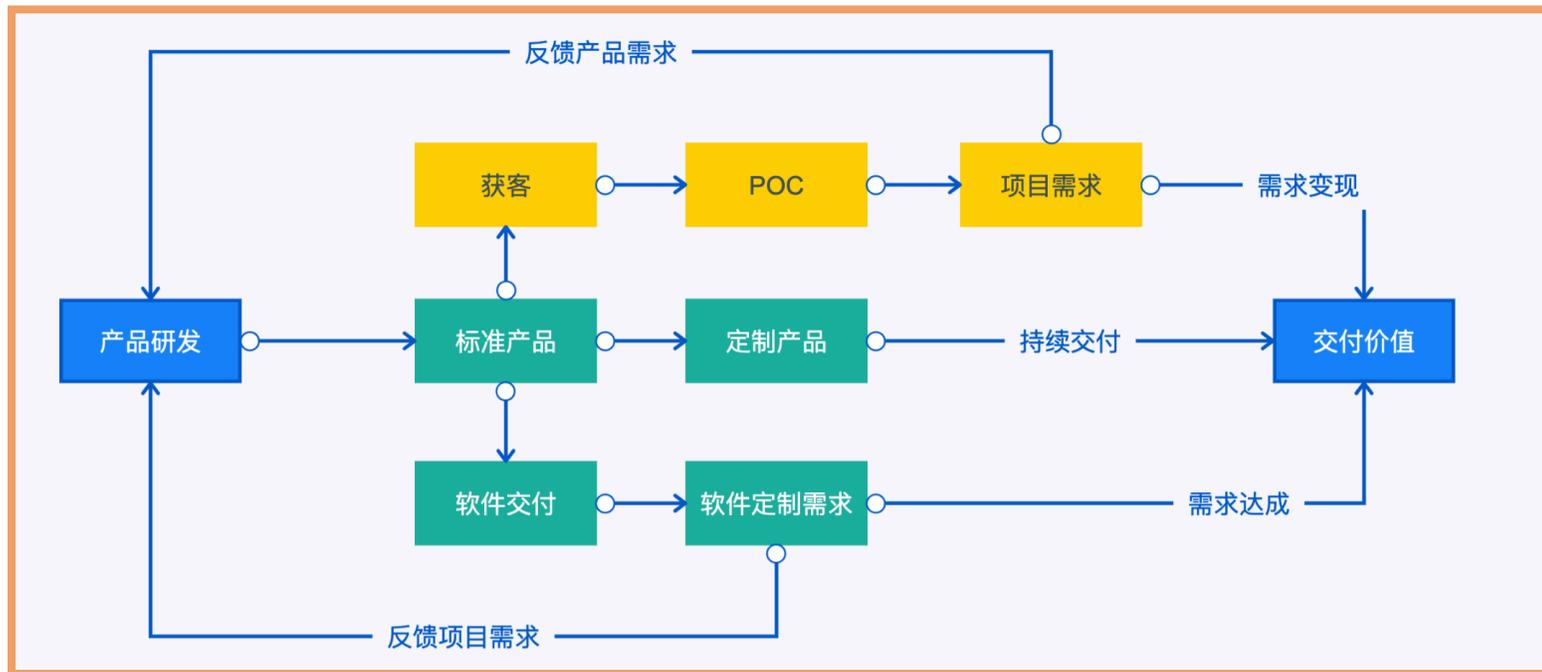
“

产业互联网升级使得2B软件服务市场需求旺盛



SaaS 服务模式高速发展，但目前大多数2B领域的软件交付，依然以传统交付模式为主。

什么是2B软件交付



面向企业用户交付软件价值的过程

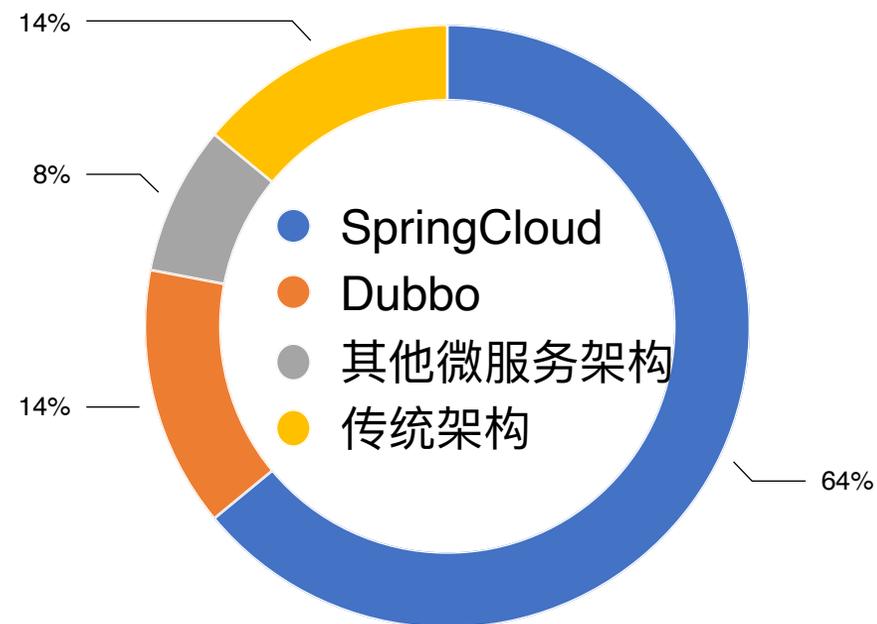
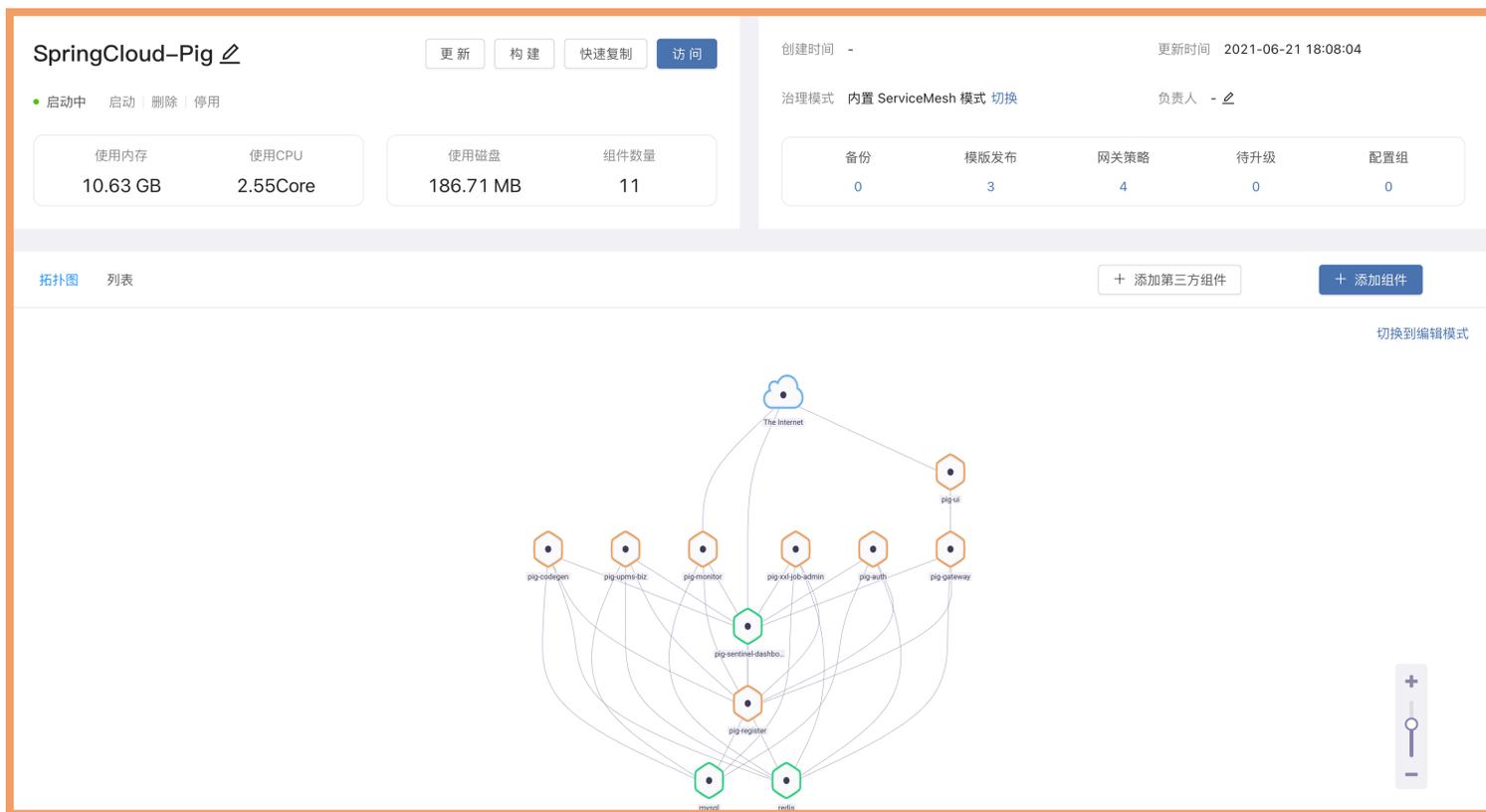
- (1) 产品研发流程管理
- (2) 产品版本管理
- (3) 概念验证, POC 管理
- (4) 客户个性化定制 (价值最大化的关键)
- (5) 客户应用的持续交付
- (6) 客户应用生产稳定性保障 (SLA)

追求价值最大化

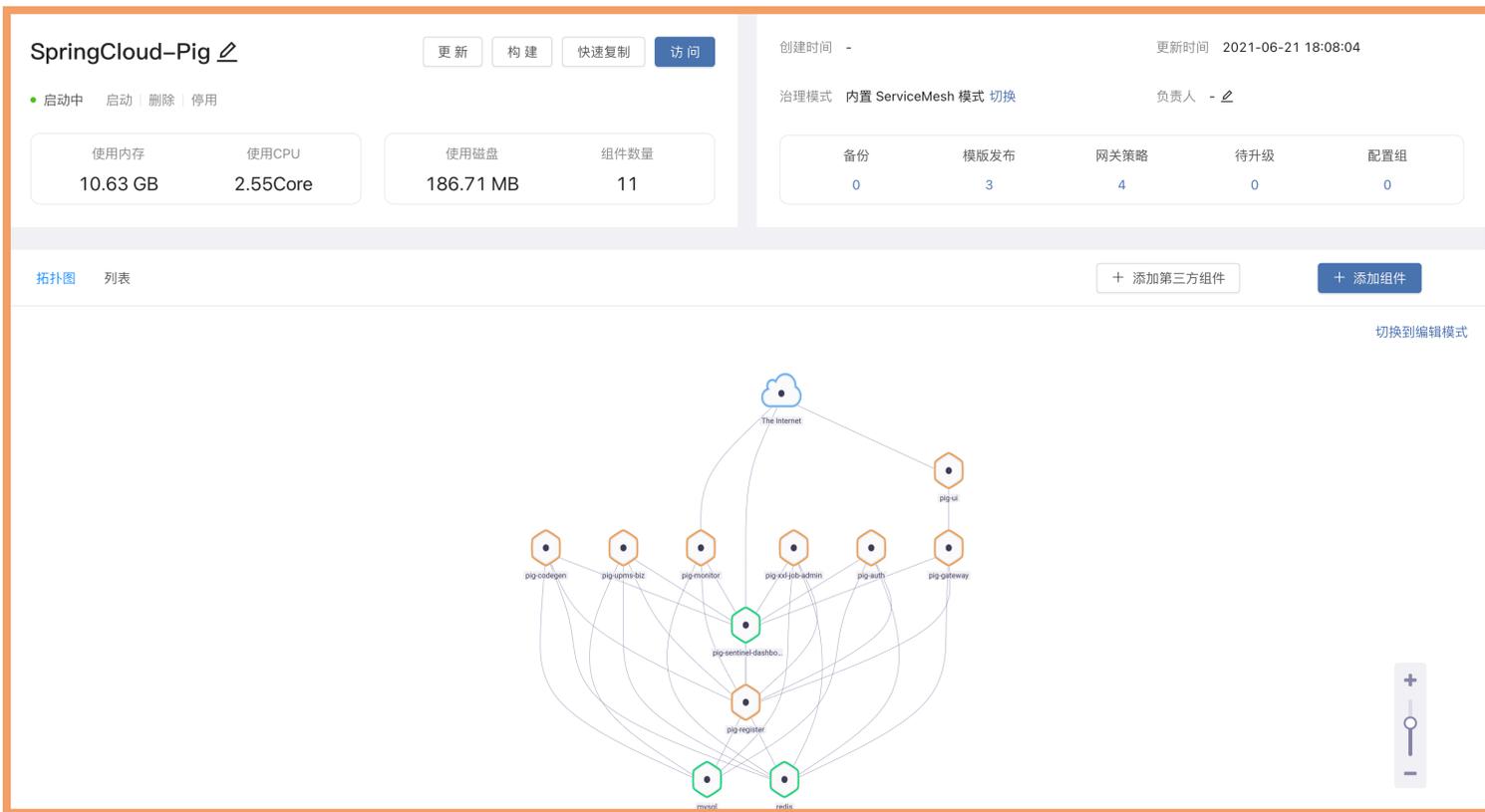
A. 高效的产品交付模式;

B. 高效的产品定制开发模式;

微服务应用成为2B软件的架构主流



微服务应用成为2B软件的架构主流



微服务是目前大多数B端业务的首选架构

组件复用

按需运维

灵活定制

客户/项目要求

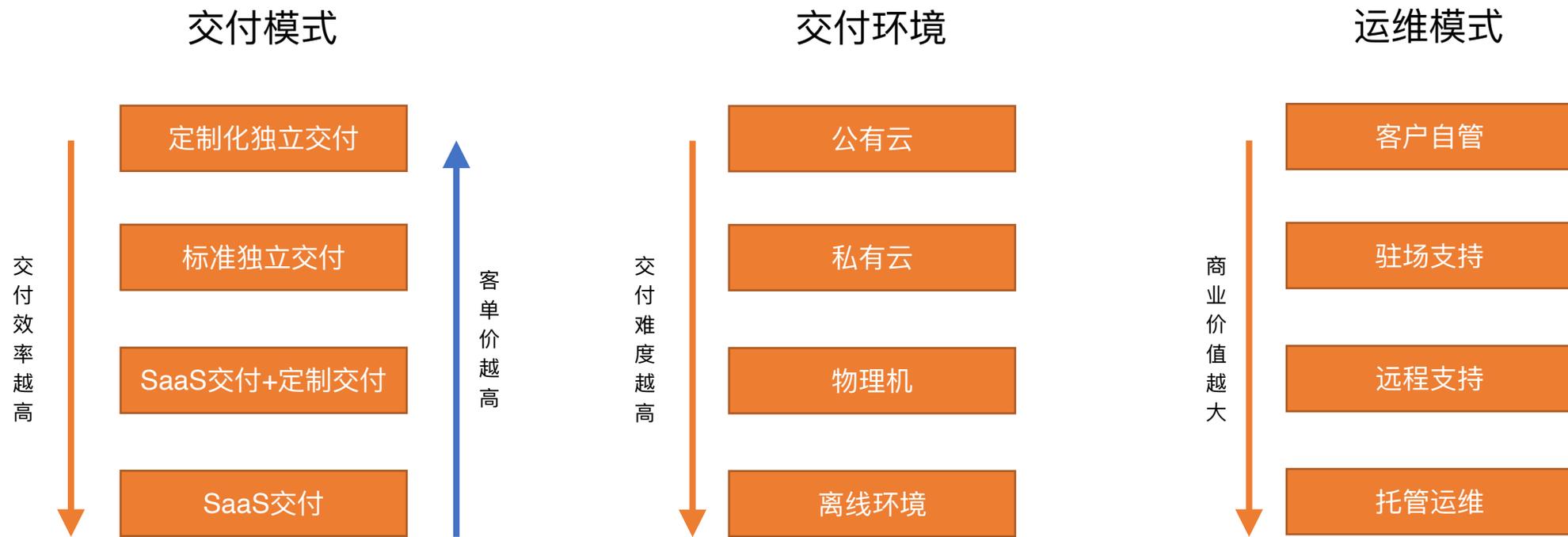


交付困难

运维困难

分布式难题

2B软件交付需求多样性



2B软件交付的愿景

数据分析及展现	4	 智慧物流与仓储解决方案 1.0	 IOT智能工业物联 1.0	 水冷磁悬浮机组模型 1.0
数据应用服务	3	Witium 推出了“仓储与物流监测系统解决方案”，此方案实现了全程运输中货品状况的监测，并在异常状况发生时提供实时告警。	平台提供高速的实时计算引擎，从边缘端到平台端，通过可视化数据流编排，定义数据处理规则，实现协议和数据格式的转换，建立数字镜像（DigitalTwin）等功能。	水冷磁悬浮机组模型机组采用磁悬浮压缩机技术、直流变频控制技术、无油润滑等先进技术，产品能效比有了很大的提高。
智能算法	^			
人工智能	1			
算法API	3			
数据采集	^			
SCADA	2			
工业物联	^			
物联网	6			
IOT解决方案	7			
智慧生活	^			
移动生活	2			
物联硬件	^			
工业网关	1			
硬件设备	1			
工业安全	^			
网络安全	1			
仓储物流	^			
仓库管理	1			
配送管理	3			
		 生产制造管理系统MES 1.0	 巴陆数据采集与监控系统软件 1.0	 点赞墙试用7天试用版 1.0
		为企业打造一个扎实、可靠、全面、可行的制造执行系统，有效的提升订单响应速度和生产效率，提升产品质量。	用于离散型和流程型工业企业的自动控制、数据采集与监控、数据与视频监控系统等，实现数据收集，为工业现场PLC/RTU的自动控制等等。	通过移动端合影、拍照、客户信息登记，数据同步展示到大屏显示，大屏以3D动态炫酷展示来访信息及照片信息，提升企业信息化形象，体现企业技术能力。
		 瀚云网关 1.0	 物流链全程信息化管理平台 1.0	 海蛛全球达 1.0
		瀚云工业网关采用全工业化硬件设计方案，利用高安全性的加密通讯机制，结合云平台，为客户提供高效的智能生产、管理、运维、监控机制，助力企业低成本快速上云。	支持物流集团企业多个业务部门如，海运船代、海运货代、空运货代、仓储、拖车、配送、报关、堆场等多操作和管理。	海蛛全球达平台是为跨境电商出口从工厂到海外仓的货物提供包括智能选仓、末端派送、仓内服务、透明收费、规范法律、责任赔偿等功能的标准化平台。

像安装手机APP一样；

在任意环境之上；

实现快速、灵活地交付；

和智能的运维能力。

第二部分

云原生与云原生应用



“

云原生技术



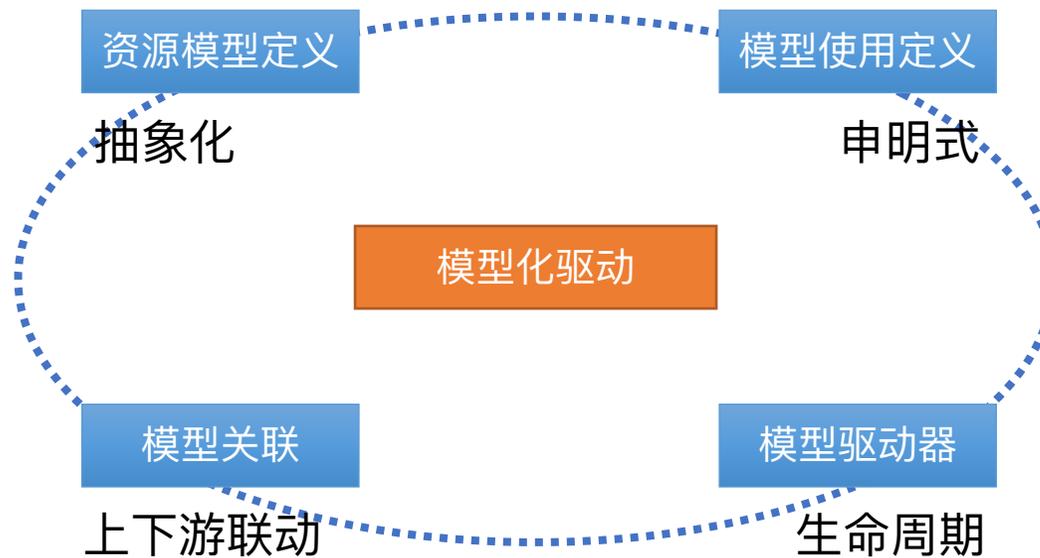
目前云原生技术很广泛，All IN 云原生；
它是一个指导思想、实践思路和行为方式；
我们仅从2B软件交付领域来谈云原生技术；

云原生技术概要

云原生四要素



云原生主要实践方式



云原生发展的热点就是解决交付问题

“

云原生应用



按照云原生技术规范设计研发的业务应用，覆盖了应用运维、交付等层面的要求。

云原生应用定义

云原生 12 因素应用规范

I. 基准代码

一份基准代码，多份部署

II. 依赖

显式声明依赖关系

III. 配置

在环境中存储配置

IV. 后端服务

把后端服务当作附加资源

V. 构建，发布，运行

严格分离构建和运行

VI. 进程

以一个或多个无状态进程运行应用

VII. 端口绑定

通过端口绑定提供服务

VIII. 并发

通过进程模型进行扩展

IX. 易处理

快速启动和优雅终止可最大化健壮性

X. 开发环境与线上环境等价

尽可能的保持开发，预发布，线上环境相同

XI. 日志

把日志当作事件流

XII. 管理进程

后台管理任务当作一次性进程运行

面向开发

单个服务

基础规范

云原生应用定义

面向交付的“三级”云原生应用定义

L1: 基础云原生应用

- (1) 容器化运行
- (2) 计算与数据分离
- (3) 满足12 因素
- (4) 可伸缩性
- (5) 可配置性
- (6) 基础可观测性

L2: 具备远程交付能力

- (1) 完全的模版化定义
- (2) 模版自动实例化
- (3) 数据自动初始化
- (4) 业务高容错性
- (5) 业务高可用性

L3: 具备持续升级能力

- (1) 高容错化数据升级
- (2) 高容错化版本升级
- (3) 版本可回滚
- (4) 业务高观测性

第三部分

面向交付的应用模型



K8S资源的模型定义

Deployment



Pod



Container

```
// DeploymentSpec is the specification of the desired behavior of the Deployment
type DeploymentSpec struct {
    // Number of desired pods. This is a pointer to distinguish between explicit
    // zero and not specified. Defaults to 1.
    // +optional
    Replicas *int32 `json:"replicas,omitEmpty" protobuf:"varint,1,opt,name=replicas"`

    // Label selector for pods. Existing ReplicaSets whose pods are
    // selected by this will be the ones affected by this deployment.
    // It must match the pod template's labels.
    Selector *metav1.LabelSelector `json:"selector" protobuf:"bytes,2,opt,name=selector"`

    // Template describes the pods that will be created.
    Template v1.PodTemplateSpec `json:"template" protobuf:"bytes,3,opt,name=template"`

    // The deployment strategy to use to replace existing pods with new ones.
    // +optional
    // +patchStrategy=retainKeys
    Strategy DeploymentStrategy `json:"strategy,omitEmpty" patchStrategy:"retainKeys"`

    // Minimum number of seconds for which a newly created pod should be ready
    // without any of its container crashing, for it to be considered available.
    // Defaults to 0 (pod will be considered available as soon as it is ready)
    // +optional
    MinReadySeconds int32 `json:"minReadySeconds,omitEmpty" protobuf:"varint,5,opt,name=minReadySeconds"`

    // The number of old ReplicaSets to retain to allow rollback.
    // This is a pointer to distinguish between explicit zero and not specified.
    // Defaults to 10.
    // +optional
    RevisionHistoryLimit *int32 `json:"revisionHistoryLimit,omitEmpty" protobuf:"varint,6,opt,name=revisionHistoryLimit"`

    // Indicates that the deployment is paused.
    // +optional
    Paused bool `json:"paused,omitEmpty" protobuf:"varint,7,opt,name=paused"`

    // The maximum time in seconds for a deployment to make progress before it
    // is considered to be failed. The deployment controller will continue to
    // process failed deployments and a condition with a ProgressDeadlineExceeded
    // reason will be surfaced in the deployment status. Note that progress will
    // not be estimated during the time a deployment is paused. Defaults to 600s
    ProgressDeadlineSeconds *int32 `json:"progressDeadlineSeconds,omitEmpty" protobuf:"varint,8,opt,name=progressDeadlineSeconds"`
}
```

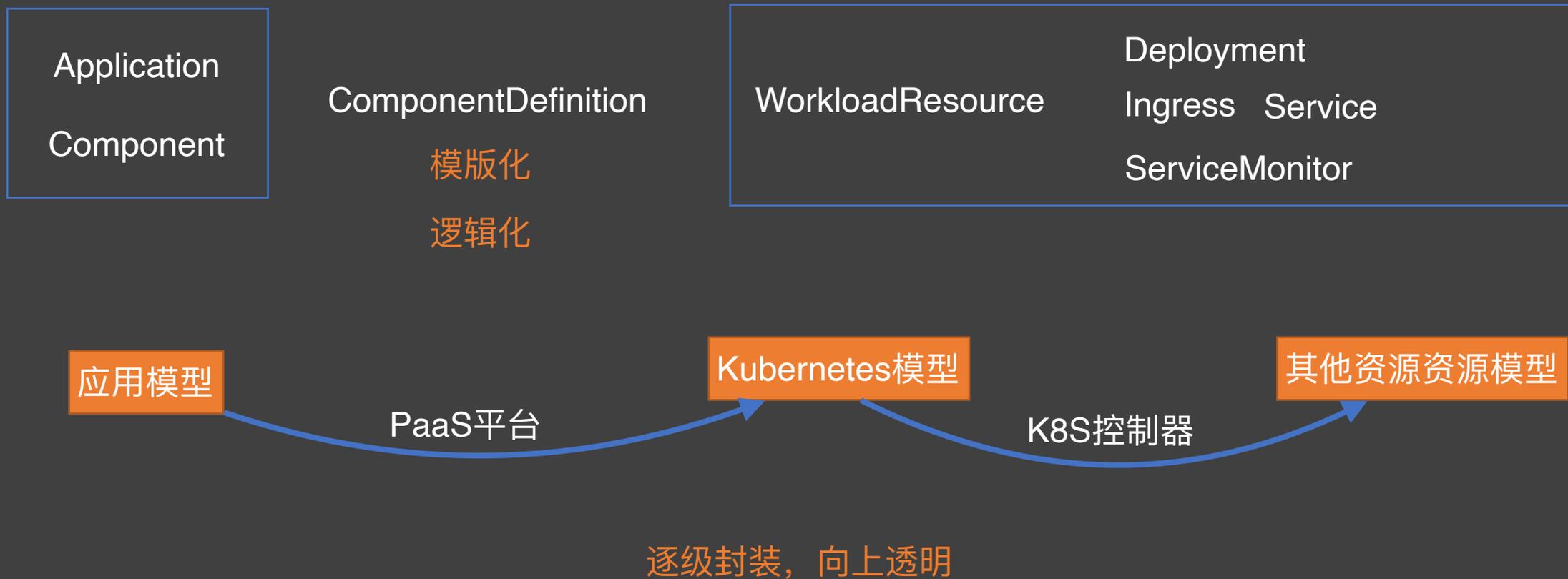
```
// PodSpec is a description of a pod.
type PodSpec struct {
    Volumes []Volume `json:"volumes,omitEmpty" patchStrategy:"merge,retainKeys" protobuf:"bytes,1,opt,name=volumes"`
    InitContainers []Container `json:"initContainers,omitEmpty" protobuf:"bytes,2,opt,name=initContainers"`
    Containers []Container `json:"containers" patchStrategy:"merge,retainKeys" protobuf:"bytes,3,opt,name=containers"`
    EphemeralContainers []EphemeralContainer `json:"ephemeralContainers" protobuf:"bytes,4,opt,name=ephemeralContainers"`
    RestartPolicy RestartPolicy `json:"restartPolicy,omitEmpty" protobuf:"bytes,5,opt,name=restartPolicy"`
    TerminationGracePeriodSeconds *int64 `json:"terminationGracePeriodSeconds,omitEmpty" protobuf:"varint,6,opt,name=terminationGracePeriodSeconds"`
    ActiveDeadlineSeconds *int64 `json:"activeDeadlineSeconds,omitEmpty" protobuf:"varint,7,opt,name=activeDeadlineSeconds"`
    DNSPolicy DNSPolicy `json:"dnsPolicy,omitEmpty" protobuf:"bytes,8,opt,name=dnsPolicy"`
    NodeSelector map[string]string `json:"nodeSelector,omitEmpty" protobuf:"bytes,9,opt,name=nodeSelector"`
    ServiceAccountName string `json:"serviceAccountName,omitEmpty" protobuf:"bytes,10,opt,name=serviceAccountName"`
    DeprecatedServiceAccount string `json:"serviceAccount,omitEmpty" protobuf:"bytes,11,opt,name=serviceAccount"`
    AutomountServiceAccountToken *bool `json:"automountServiceAccountToken,omitEmpty" protobuf:"boolean,12,opt,name=automountServiceAccountToken"`
    NodeName string `json:"nodeName,omitEmpty" protobuf:"bytes,13,opt,name=nodeName"`
    HostNetwork bool `json:"hostNetwork,omitEmpty" protobuf:"boolean,14,opt,name=hostNetwork"`
    HostPID bool `json:"hostPID,omitEmpty" protobuf:"boolean,15,opt,name=hostPID"`
    HostIPC bool `json:"hostIPC,omitEmpty" protobuf:"boolean,16,opt,name=hostIPC"`
    ShareProcessNamespace *bool `json:"shareProcessNamespace,omitEmpty" protobuf:"boolean,17,opt,name=shareProcessNamespace"`
    SecurityContext *PodSecurityContext `json:"securityContext,omitEmpty" protobuf:"bytes,18,opt,name=securityContext"`
    ImagePullSecrets []LocalObjectReference `json:"imagePullSecrets,omitEmpty" protobuf:"bytes,19,opt,name=imagePullSecrets"`
    Hostname string `json:"hostname,omitEmpty" protobuf:"bytes,20,opt,name=hostname"`
    Subdomain string `json:"subdomain,omitEmpty" protobuf:"bytes,21,opt,name=subdomain"`
}
```

```
// A single application container that you want to run within a pod.
type Container struct {
    Name string `json:"name" protobuf:"bytes,1,opt,name=name"`
    Image string `json:"image,omitEmpty" protobuf:"bytes,2,opt,name=image"`
    Command []string `json:"command,omitEmpty" protobuf:"bytes,3,opt,name=command"`
    Args []string `json:"args,omitEmpty" protobuf:"bytes,4,opt,name=args"`
    WorkingDir string `json:"workingDir,omitEmpty" protobuf:"bytes,5,opt,name=workingDir"`
    Ports []ContainerPort `json:"ports,omitEmpty" patchStrategy:"merge" protobuf:"bytes,6,opt,name=ports"`
    EnvFrom []EnvFromSource `json:"envFrom,omitEmpty" protobuf:"bytes,7,opt,name=envFrom"`
    Env []EnvVar `json:"env,omitEmpty" patchStrategy:"merge" protobuf:"bytes,8,opt,name=env"`
    Resources ResourceRequirements `json:"resources,omitEmpty" protobuf:"bytes,9,opt,name=resources"`
    VolumeMounts []VolumeMount `json:"volumeMounts,omitEmpty" patchStrategy:"merge" protobuf:"bytes,10,opt,name=volumeMounts"`
    VolumeDevices []VolumeDevice `json:"volumeDevices,omitEmpty" protobuf:"bytes,11,opt,name=volumeDevices"`
    LivenessProbe *Probe `json:"livenessProbe,omitEmpty" protobuf:"bytes,12,opt,name=livenessProbe"`
    ReadinessProbe *Probe `json:"readinessProbe,omitEmpty" protobuf:"bytes,13,opt,name=readinessProbe"`
    StartupProbe *Probe `json:"startupProbe,omitEmpty" protobuf:"bytes,14,opt,name=startupProbe"`
    Lifecycle *Lifecycle `json:"lifecycle,omitEmpty" protobuf:"bytes,15,opt,name=lifecycle"`
    TerminationMessagePath string `json:"terminationMessagePath,omitEmpty" protobuf:"bytes,16,opt,name=terminationMessagePath"`
    TerminationMessagePolicy TerminationMessagePolicy `json:"terminationMessagePolicy,omitEmpty" protobuf:"bytes,17,opt,name=terminationMessagePolicy"`
    ImagePullPolicy PullPolicy `json:"imagePullPolicy,omitEmpty" protobuf:"bytes,18,opt,name=imagePullPolicy"`
    SecurityContext *SecurityContext `json:"securityContext,omitEmpty" protobuf:"bytes,19,opt,name=securityContext"`
    Stdin bool `json:"stdin,omitEmpty" protobuf:"boolean,20,opt,name=stdin"`
    StdinOnce bool `json:"stdinOnce,omitEmpty" protobuf:"boolean,21,opt,name=stdinOnce"`
    TTY bool `json:"tty,omitEmpty" protobuf:"boolean,22,opt,name=tty"`
}
```

面向交付的应用模型



应用模型定义用例



应用模型定义用例

```
type RainbondApplicationConfig struct {
    AppKeyID      string `json:"group_key"`
    AppName       string `json:"group_name"`
    AppVersion    string `json:"group_version"`
    TemplateVersion string `json:"template_version"`
    Components    []*Component
    Plugins        []*Plugin
    AppConfigGroups []*AppConfigGroup
    IngressHTTPRoutes []*IngressHTTPRoute
    IngressStreamRoutes []*IngressStreamRoute `json:"ingress_stream_routes,omitempty"`
}
```

★ 定义了一个完整应用的部署模型

应用模型定义用例

```
type Component struct {  
    Memory          int          `json:"memory"`  
    CPU             int          `json:"cpu"`  
    GPU            int          `json:"gpu"`  
    Probes          []ComponentProbe `json:"probes"`  
    AppImage        ImageInfo    `json:"service_image"`  
    ComponentID     string       `json:"service_id"`  
    DeployType     DeployType  `json:"extend_method"`  
    ServicePluginConfigs []ComponentPluginConfig `json:"service_related_plugin_config,omitempty"`  
    ComponentMonitor []ComponentMonitor `json:"component_monitors"`  
    ComponentGraphs []ComponentGraph `json:"component_graphs"`  
    Envs            []ComponentEnv `json:"service_env_map_list"`  
    Ports          []ComponentPort `json:"port_map_list"`  
    ServiceVolumeMapList ComponentVolumeList `json:"service_volume_map_list"`  
}
```



定义业务组件运行、运维等需求

应用模型定义的UI化

The screenshot displays the nginx management interface. At the top, there's a navigation bar with the nginx logo and several action buttons: 'Web终端', '构建', '更新(滚动)', and '访问'. Below this is a sub-navigation bar with tabs for '总览', '监控', '日志', '伸缩', '环境配置', '依赖', '存储', '端口', '插件', '构建源', and '其他设置'. The '伸缩' (Scaling) tab is currently selected and highlighted with an orange border.

The main content area is divided into three sections:

- 实例情况 (Instance Status):** Shows a circular gauge for '运行内存' (Running Memory) at 0.25%. Below it, there's a '查询命令' (Query Command) field containing 'grctl service get gr57b3e5 -t y7lcz42i' and a '复制' (Copy) button. A note below reads: '查询详细的组件实例信息,请复制查询命令到集群管理节点查询'.
- 手动伸缩 (Manual Scaling):** Features three input fields with '设置' (Set) buttons:
 - 内存 (Memory):** Set to 512M.
 - GPU显存 (GPU Memory):** Set to 0 MiB. A note below states: 'GPU 设置为0则不分配GPU资源, 请求显存超过集群中单颗显卡的最大可用容量时无法进行调度。'
 - 实例数量 (Instance Count):** Set to 1. A note below states: '该值定义组件实例数量的初始值, 实际值可能被自动伸缩调整'.
- 自动伸缩 (Automatic Scaling):** Includes a '功能开关' (Feature Toggle) and a table for '最小实例数' (Minimum Instance Count) and '最大实例数' (Maximum Instance Count). The table shows values 0 and 1 respectively, with a '+' button to the right.

应用模型定义的UI化

Redis4.0

重启 | 关闭 | 修改所属应用 | 删除

总览 监控 日志 伸缩 环境配置 依赖 **存储** 端口 插件 构建源 其他设置

存储配置发生变化后需要更新组件才能生效

存储设置

存储名称	挂载路径	存储类型	存储容量
GRE1F1EA_1	/data	共享存储（文件）	不限制

共享其他组件存储

本地挂载路径	目标存储名称	目标挂载路径	目标存储类型	目标所属组件
暂无数据				

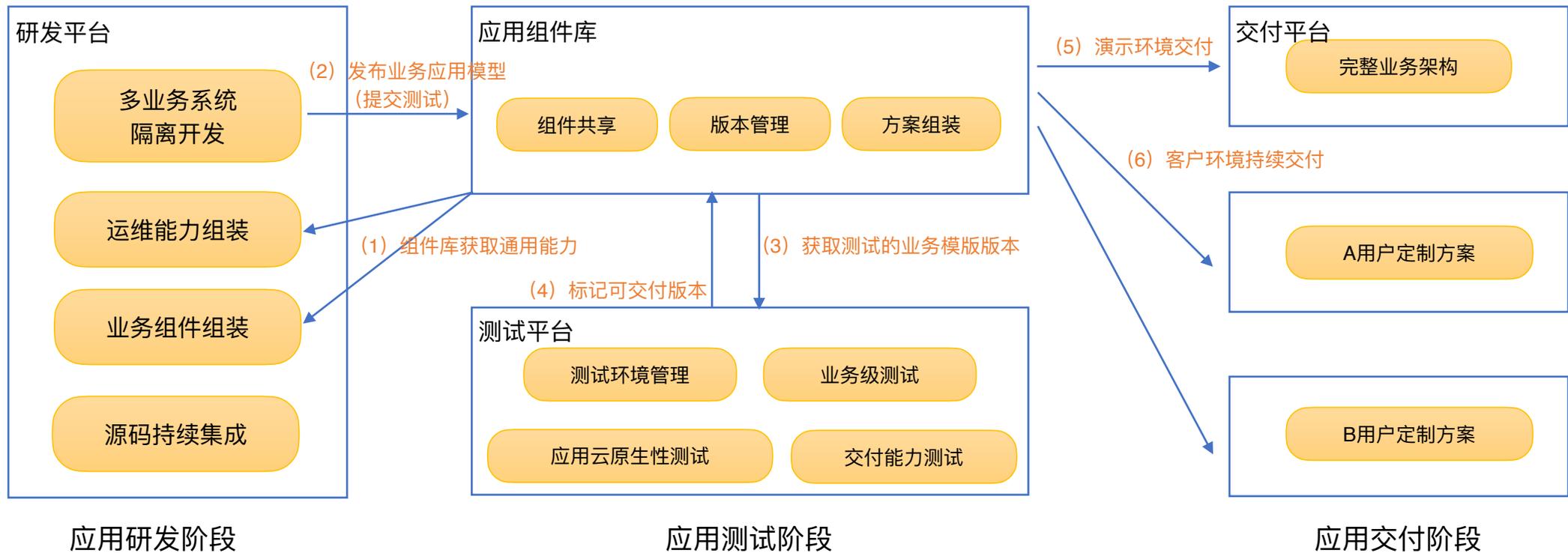
取消 确认

第四部分

2B交付版本的DevOps



2B交付版本的DevOps全景图



应用模型定义实践-开发者

应用组装和研发

CLOUD-CONSOLE-5.3...

更新 构建 快速复制 访问

运行中 启动 | 停用

使用内存	使用CPU	使用磁盘	组件数量
4.06 GB	1.19Core	4.61 MB	7

创建时间 2021-04-16 18:34:41 更新时间 2021-06-08 15:56:52

治理模式 内置 ServiceMesh 模式 切换 负责人 张震

备份	模版发布	网关策略	待升级	配置组
0	1	4	1	1

拓扑图 列表 + 添加第三方组件 + 添加组件 切换到编辑模式

添加组件

从源代码开始

- 自定义仓库
- Gitlab (gitlab)
- Github (Github)

注:支持 Dockerfile Java PHP Python Nodejs NodeJSStatic Go Netcore Html 等语言规范

从源镜像开始

- 指定镜像
- 指定DockerRun命令
- 指定DockerCompose配置

从应用市场开始

- 开源应用商店
- 工业物联网应用商店
- 码云GVP应用商店

- IOT 智能工业... 版本: 1.0
- 海模智云模具... 版本: 1.0
- 洗涤塔系统模型 版本: 1.0

查看更多...

应用模型定义实践-开发者

开发者

源码定义规范

源码持续构建

选择中间件

实现数据初始化

实现业务可配置

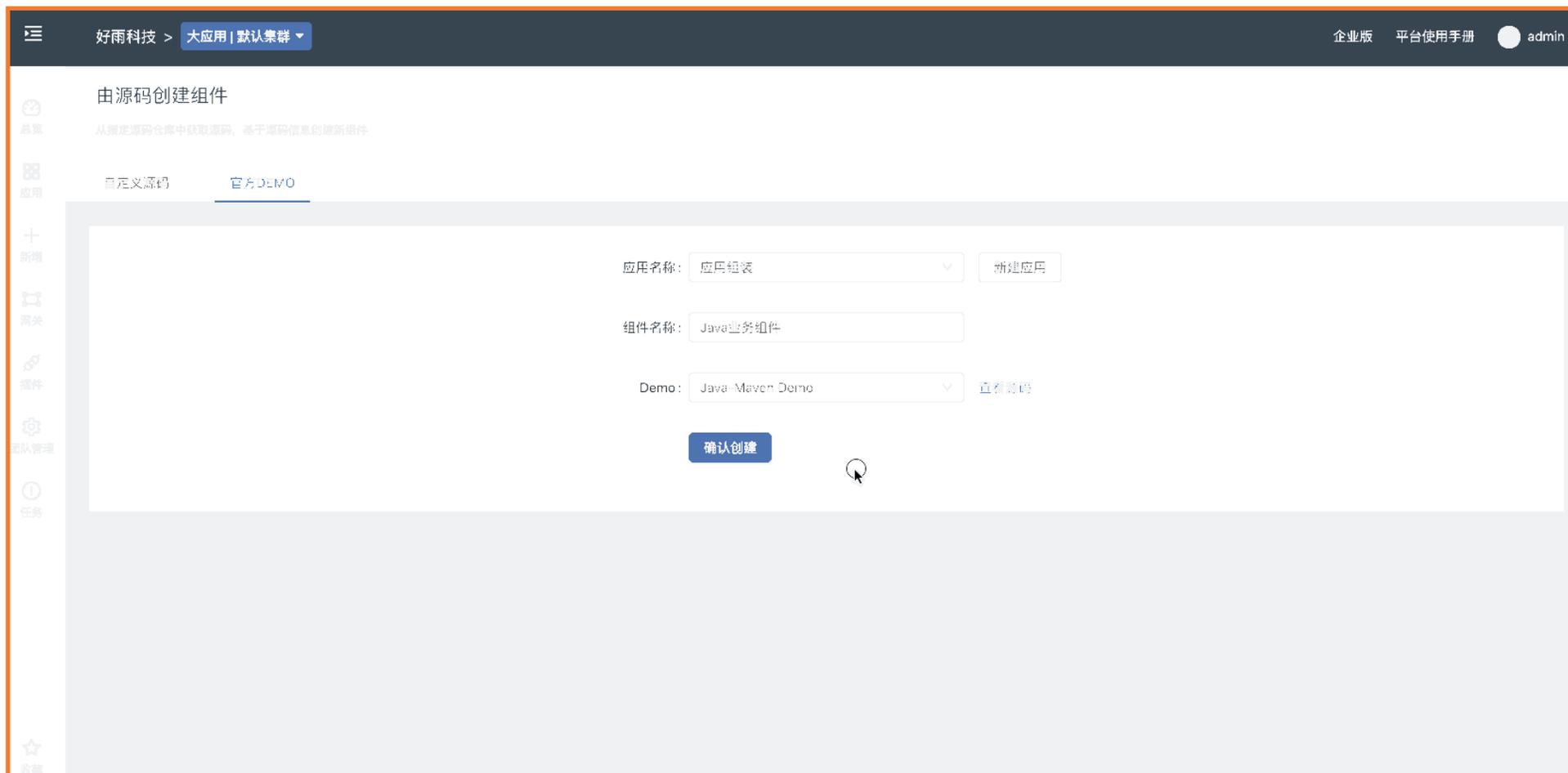
云原生平台

属性识别

智能赋值

UI化管理

模型打包



应用模型交付实践-交付平台

交付

应用模型一键交付

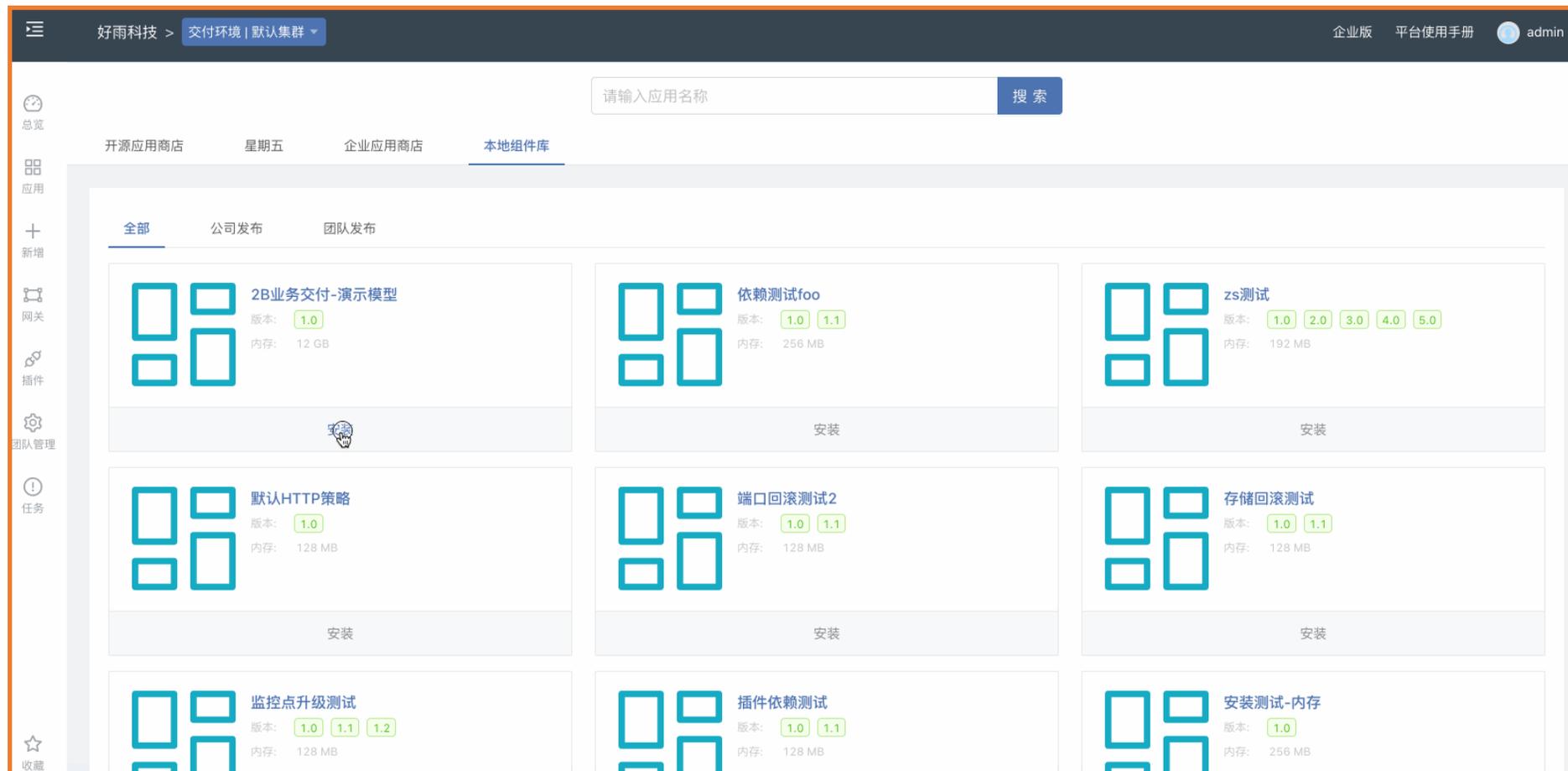
自动资源调度

自动数据初始化

分配访问策略

应用级持续升级

支持上百个组件一键交付



应用模型运维实践-插件化运维能力

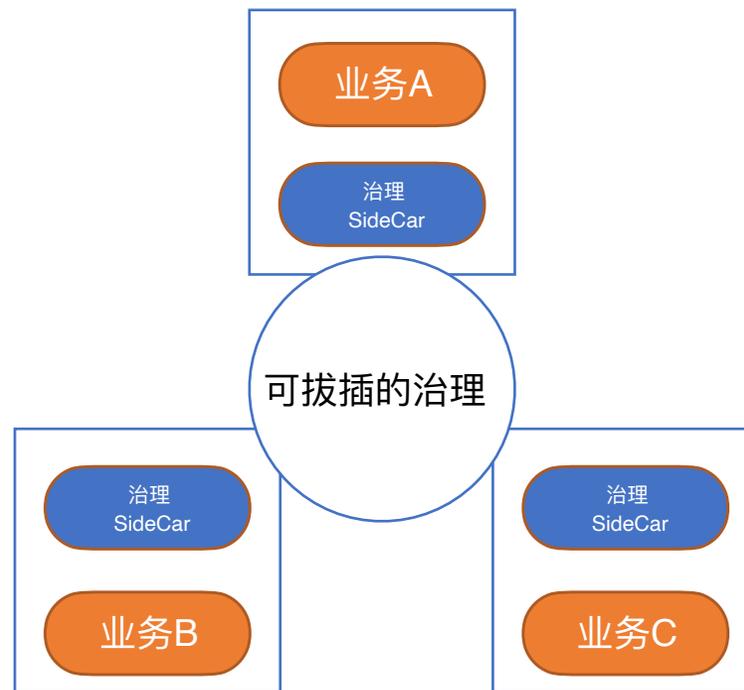
应用治理架构与业务架构解耦合

(1) 业务开发者定义业务架构

(2) 运维/架构师定义业务治理架构

网络治理模式 日志收集模式 链路跟踪 访问日志记录

(3) 业务不受治理架构的变化影响



应用模型运维实践-插件化运维能力

常用运维能力插件

我的插件

应用插件是标准化的为应用提供功能扩展，与应用共同运行的程序

+ 新建插件

 服务综合网络治理插件 出口入口共治网络 该插件支持服务的出站和入站网络治理，包括服务动态路由、限流、熔断等功能	 fileBeat日志收集插件 一般类型 该插件支持通过fileBeat日志收集器收集日志信息到ELK集群中，完成日志的收集、分析功能(需要访问公网安装)	
 阿里云日志服务收集插件 一般类型 该插件支持通过logtail日志收集器收集日志信息到阿里云日志服务中，完成日志的收集、分析功能(需要访问公网安装)	 MySQLD Exporter 监控 用于暴露 MySQL 的 Prometheus 指标	 出口网络治理插件 出口网络 实现智能路由、A/B测试、灰度发布、端口复用等微服务治理功能
 服务实时性能分析 性能分析 实时分析应用的吞吐率、响应时间、在线人数等指标	 数据初始化插件 性能分析 实时分析应用的吞吐率、响应时间、在线人数等指标	 链路追踪agent 性能分析 实时分析应用的吞吐率、响应时间、在线人数等指标

欢迎关注开源项目：

<https://github.com/goodrain/rainbond>

云原生且易用的应用管理平台



社区钉钉群



微信