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

第一部分

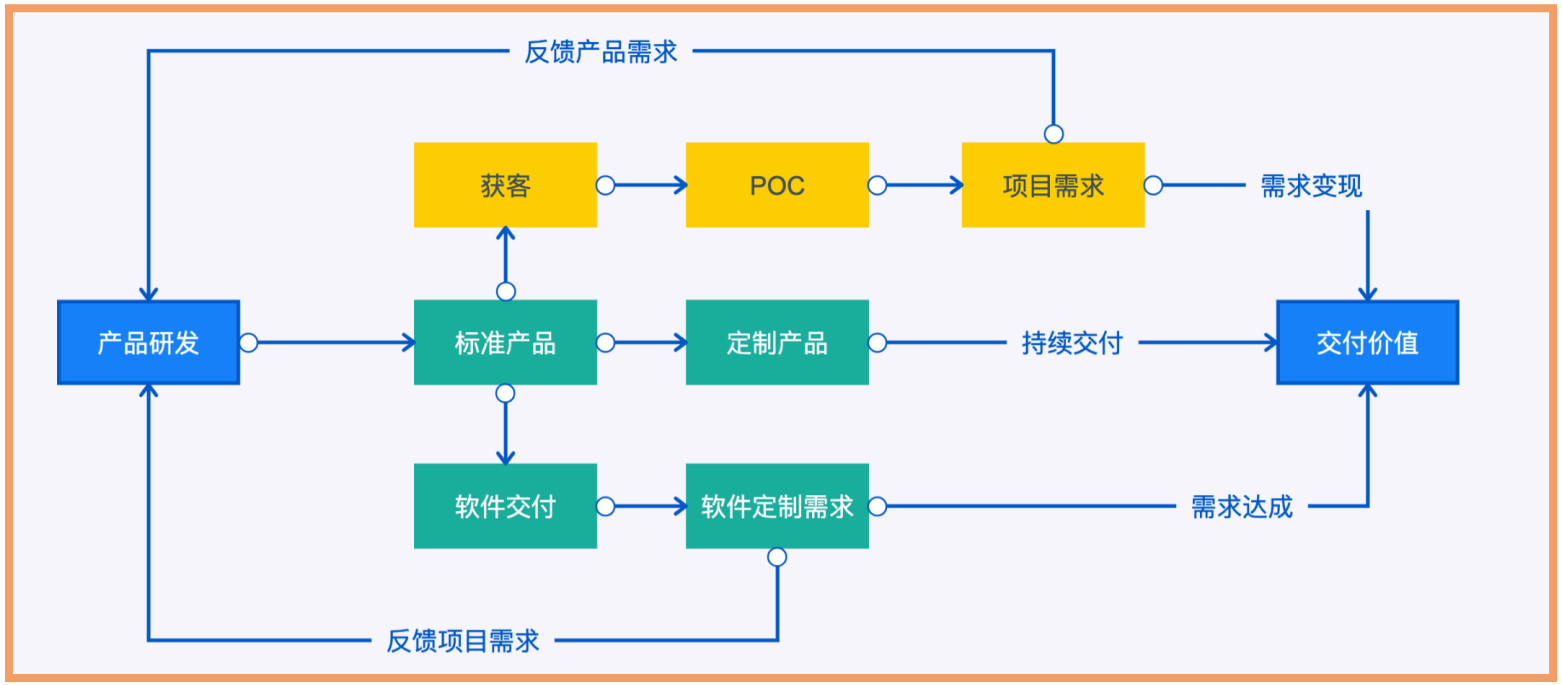
2B软件交付的困局

“

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

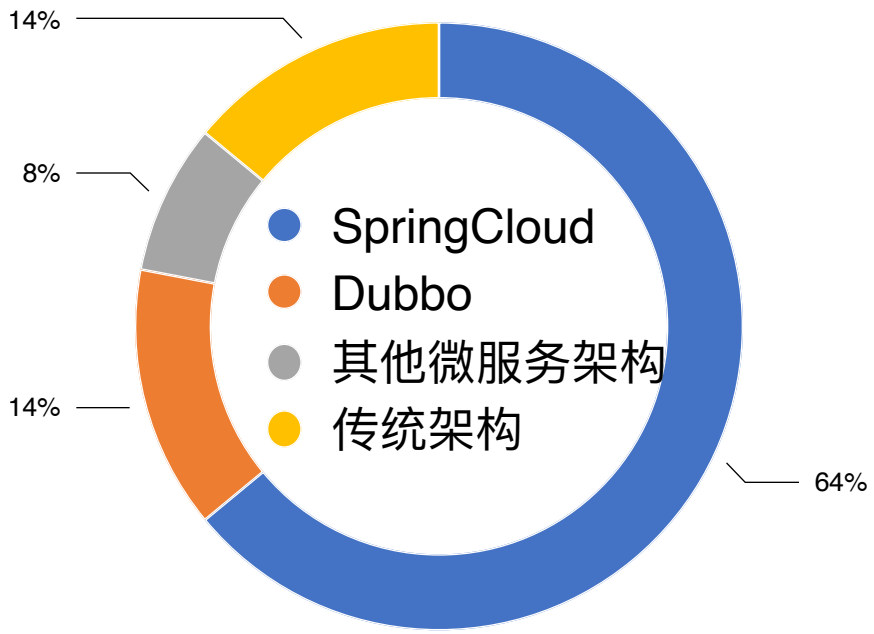
(6) 客户应用生产稳定性保障 (SLA)

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

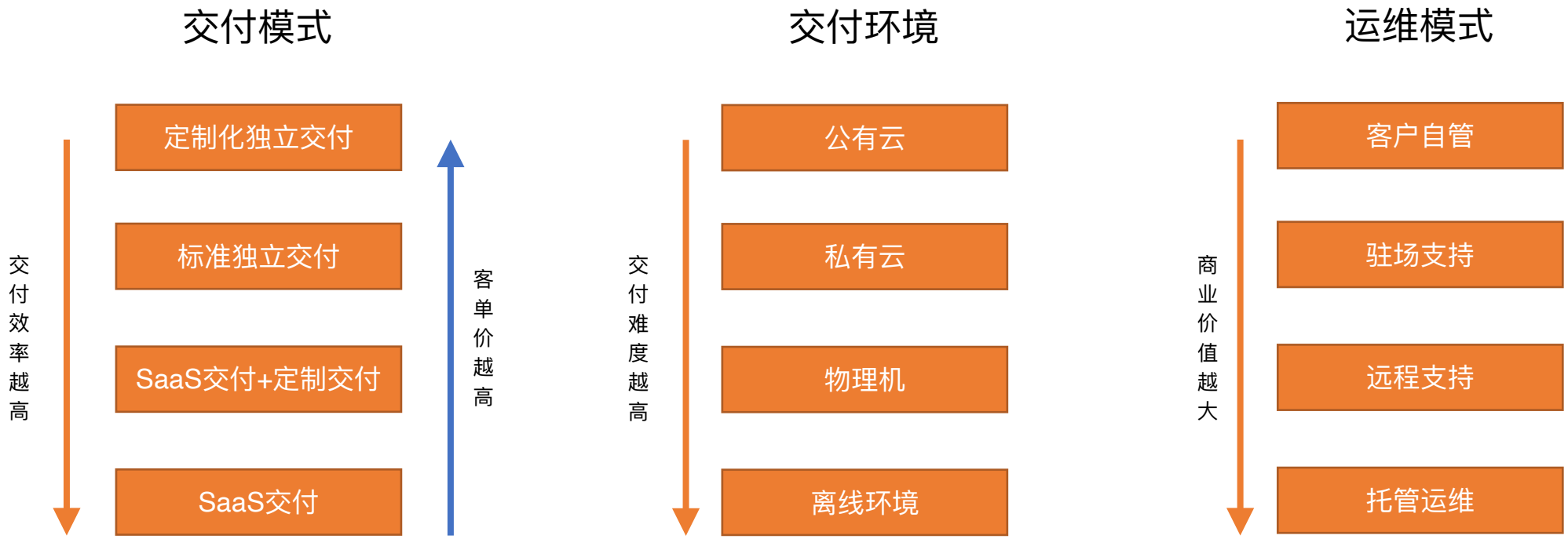
客户/项目要求

交付困难

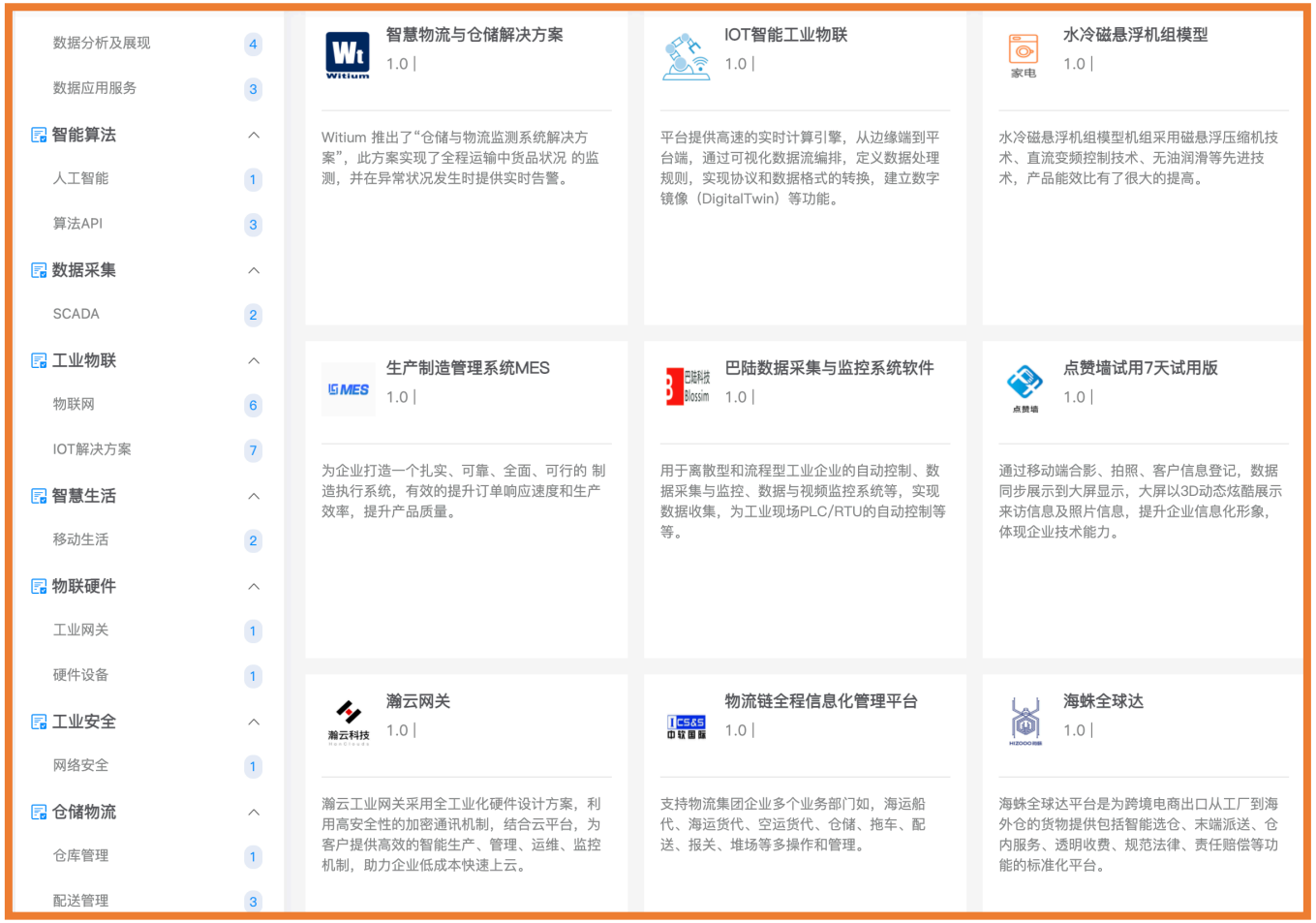
运维困难

分布式难题

2B软件交付需求多样性



2B软件交付的愿景



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

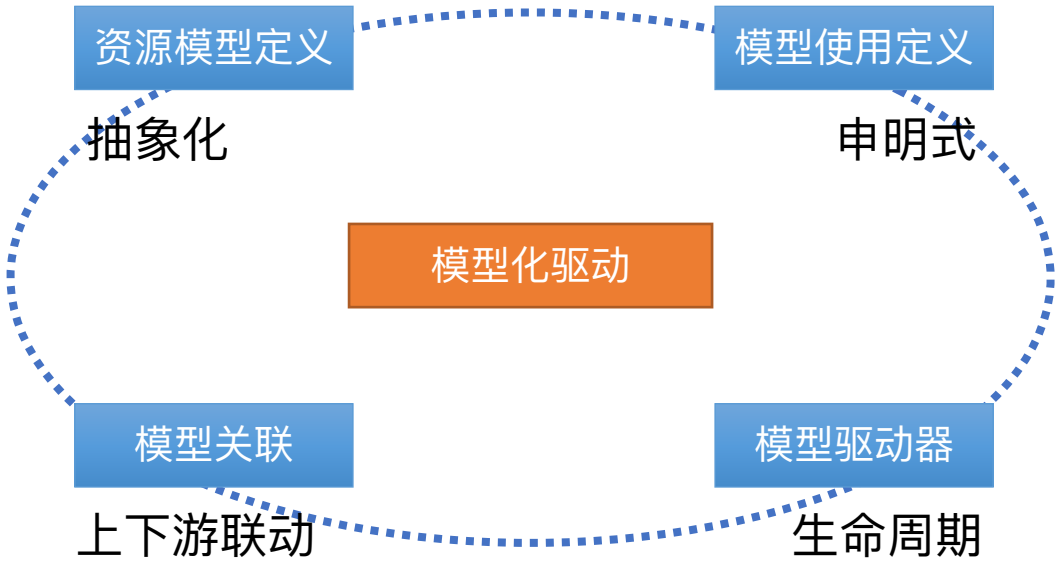
目前云原生技术很广泛，All IN 云原生；
它是一个指导思想、实践思路和行为方式；
我们仅从2B软件交付领域来谈云原生技术；

云原生技术概要

云原生四要素

- 微服务
- 容器化
- DevOps
- 持续交付

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

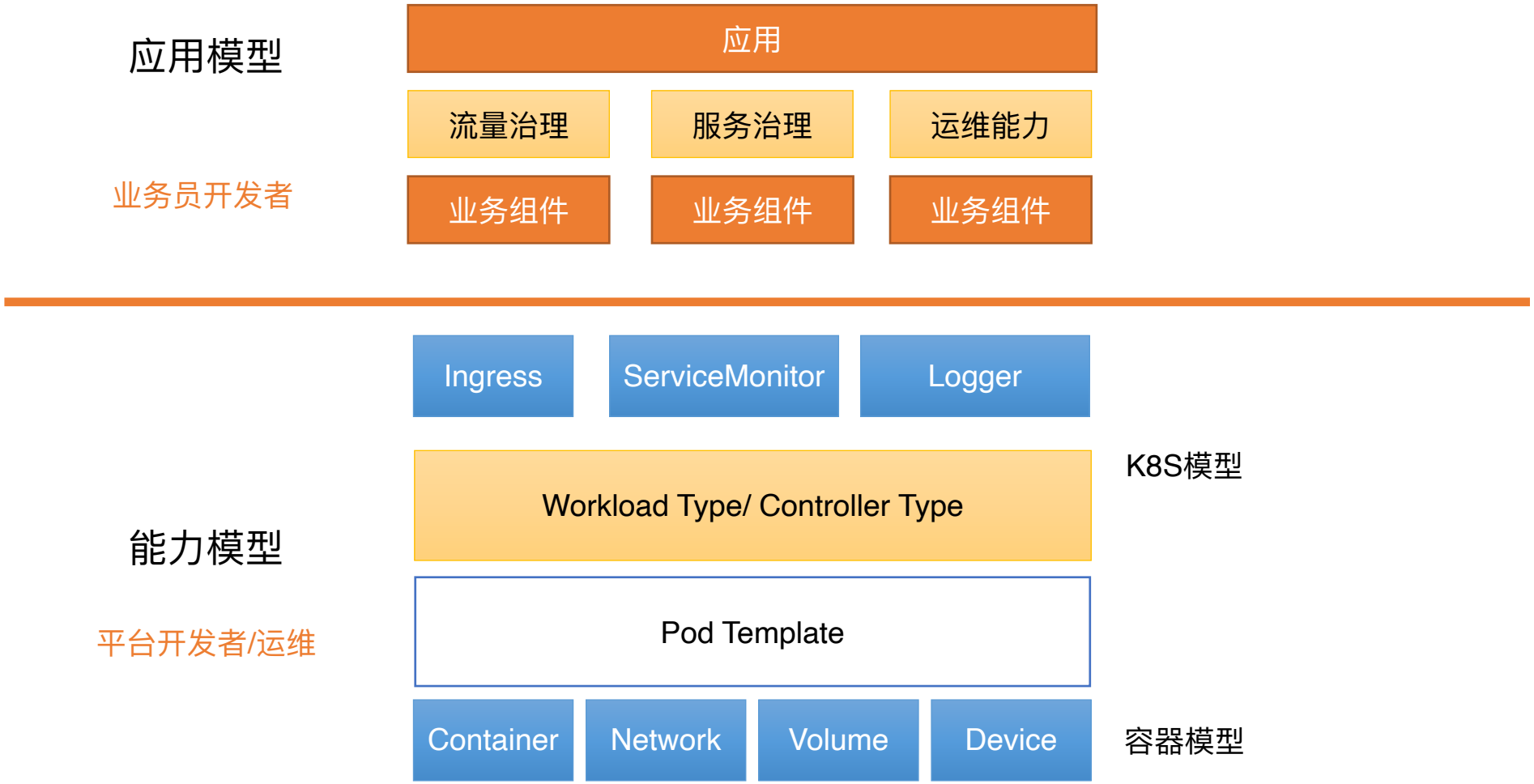
Deployment → Pod → Container

```go
// 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=repli

    // 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=s

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

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

    // 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,o

    // 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:

    // 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 ProgressDeadlineExceede
    // 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" pro
}
```

```go
// PodSpec is a description of a pod.
type PodSpec struct {
    Volumes []Volume `json:"volumes,omitempty" patchStrategy:"me
    InitContainers []Container `json:"initContainers,omitempty"
    Containers []Container `json:"containers" patchStrategy:"mer
    EphemeralContainers []EphemeralContainer `json:"ephemeralCon
    RestartPolicy RestartPolicy `json:"restartPolicy,omitempty"
    TerminationGracePeriodSeconds *int64 `json:"terminationGrace
    ActiveDeadlineSeconds *int64 `json:"activeDeadlineSeconds,om
    DNSPolicy DNSPolicy `json:"dnsPolicy,omitempty" protobuf:"by
    NodeSelector map[string]string `json:"nodeSelector,omitempty
    ServiceAccountName string `json:"serviceAccountName,omitempt
    DeprecatedServiceAccount string `json:"serviceAccount,omitem
    AutomountServiceAccountToken *bool `json:"automountServiceAc
    NodeName string `json:"nodeName,omitempty" protobuf:"bytes,1
    HostNetwork bool `json:"hostNetwork,omitempty" protobuf:"var
    HostPID bool `json:"hostPID,omitempty" protobuf:"varint,12,o
    HostIPC bool `json:"hostIPC,omitempty" protobuf:"varint,13,o
    ShareProcessNamespace *bool `json:"shareProcessNamespace,omi
    SecurityContext *PodSecurityContext `json:"securityContext,o
    ImagePullSecrets []LocalObjectReference `json:"imagePullSecr
    Hostname string `json:"hostname,omitempty" protobuf:"bytes,1
    Subdomain string `json:"subdomain,omitempty" protobuf:"bytes
```

```go
// A single application container that you want to run within a po
type Container struct {
    Name string `json:"name" protobuf:"bytes,1,opt,name=name"`
    Image string `json:"image,omitempty" protobuf:"bytes,2,opt,nam
    Command []string `json:"command,omitempty" protobuf:"bytes,3,r
    Args []string `json:"args,omitempty" protobuf:"bytes,4,rep,nam
    WorkingDir string `json:"workingDir,omitempty" protobuf:"bytes
    Ports []ContainerPort `json:"ports,omitempty" patchStrategy:"m
    EnvFrom []EnvFromSource `json:"envFrom,omitempty" protobuf:"by
    Env []EnvVar `json:"env,omitempty" patchStrategy:"merge" patch
    Resources ResourceRequirements `json:"resources,omitempty" pro
    VolumeMounts []VolumeMount `json:"volumeMounts,omitempty" patc
    VolumeDevices []VolumeDevice `json:"volumeDevices,omitempty" p
    LivenessProbe *Probe `json:"livenessProbe,omitempty" protobuf:
    ReadinessProbe *Probe `json:"readinessProbe,omitempty" protobu
    StartupProbe *Probe `json:"startupProbe,omitempty" protobuf:"b
    Lifecycle *Lifecycle `json:"lifecycle,omitempty" protobuf:"byt
    TerminationMessagePath string `json:"terminationMessagePath,om
    TerminationMessagePolicy TerminationMessagePolicy `json:"termi
    ImagePullPolicy PullPolicy `json:"imagePullPolicy,omitempty" p
    SecurityContext *SecurityContext `json:"securityContext,omitem
    Stdin bool `json:"stdin,omitempty" protobuf:"varint,16,opt,nam
    StdinOnce bool `json:"stdinOnce,omitempty" protobuf:"varint,17
    TTY bool `json:"tty,omitempty" protobuf:"varint,18,opt,name=tt
}
```

面向交付的应用模型


应用模型
应用
流量治理
服务治理
运维能力
业务员开发者
业务组件
业务组件
业务组件
Ingress
ServiceMonitor
Logger
K8S模型
能力模型
Workload Type/ Controller Type
Pod Template
平台开发者/运维
Container
Network
Volume
Device
容器模型

应用模型定义用例

Application

Component

ComponentDefinition

模版化

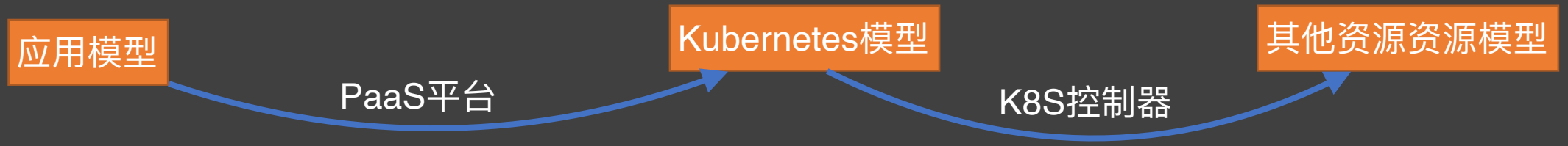
逻辑化

WorkloadResource

Deployment

Ingress Service

ServiceMonitor



逐级封装，向上透明

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"`
    TempleteVersion    string                `json:"template_version"`
    Components         []*Component          `json:"apps"`
    Plugins            []*Plugin             `json:"plugins,omitempty"`
    AppConfigGroups    []*AppConfigGroup     `json:"app_config_groups,omitempty"`
    IngressHTTPRoutes  []*IngressHTTPRoute   `json:"ingress_http_routes,omitempty"`
    IngressSreamRoutes []*IngressSreamRoute `json:"ingress_stream_routes,omitempty"`
}
```

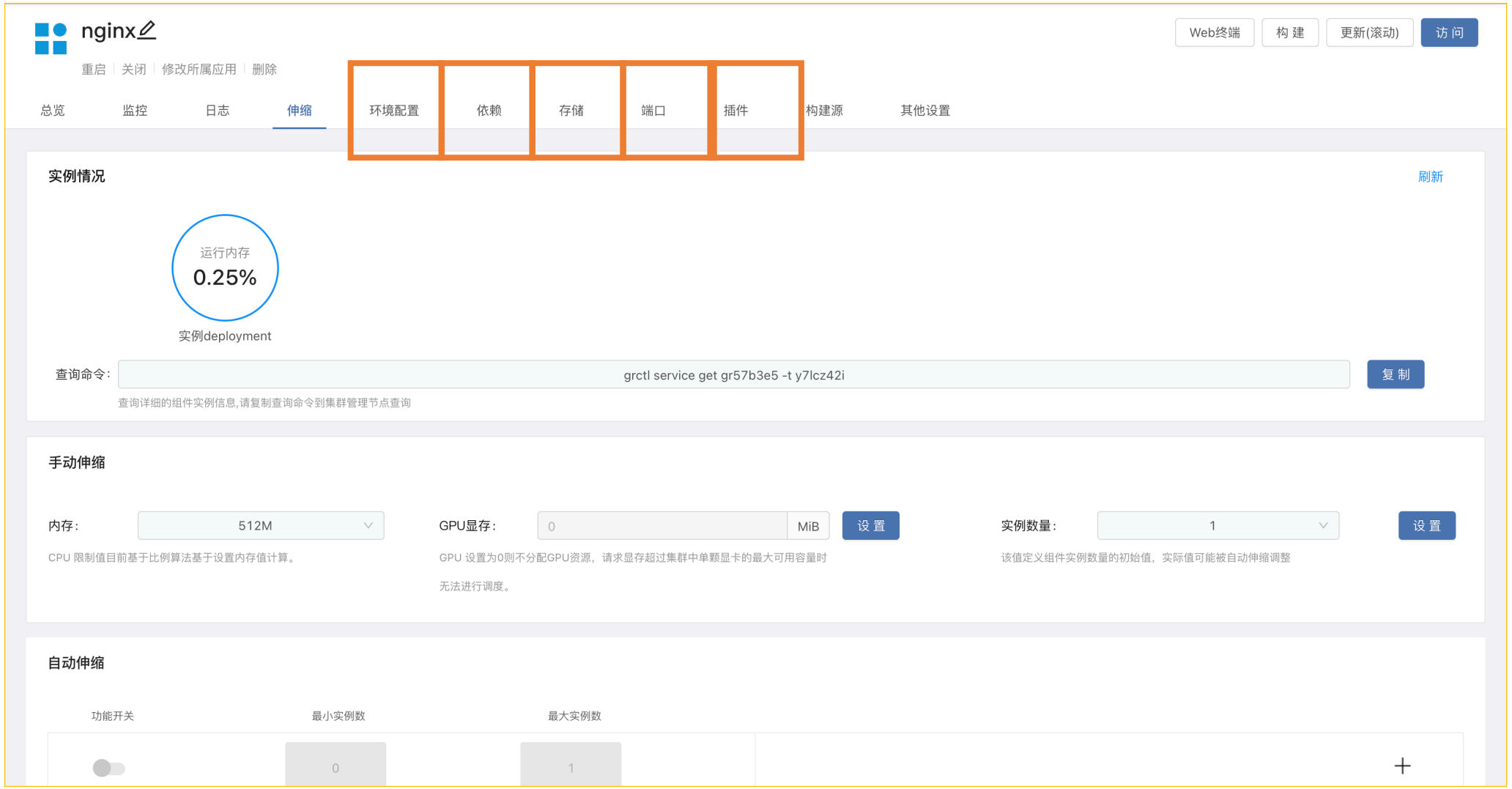
- 定义了一个完整应用的部署模型

应用模型定义用例

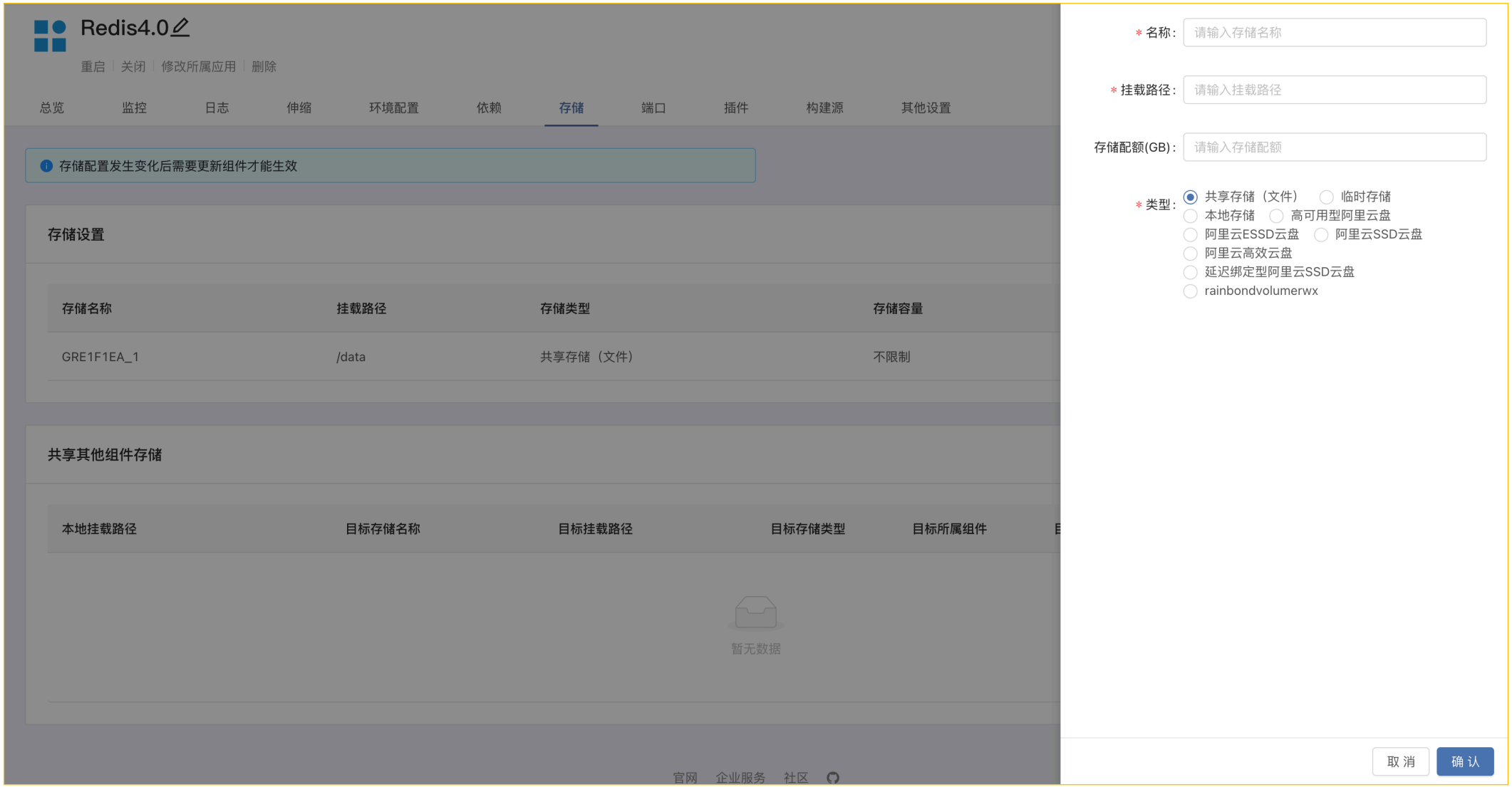
```
type Component struct {
    Memory                int                    `json:"memory"`
    CPU                   int                    `json:"cpu"`
    GPU                   int                    `json:"gpu"`
    Probes                []ComponentProbe       `json:"probes"`
    AppImage              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


nginx
重启 | 关闭 | 修改所属应用 | 删除
Web终端
构 建
更新(滚动)
访 问
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
实例情况
刷新
运行内存
0.25%
实例deployment
查询命令:
grctl service get gr57b3e5 -t y7lcz42i
复 制
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
内存:
512M
GPU显存:
0
MiB
设 置
实例数量:
1
设 置
CPU 限制值目前基于比例算法基于设置内存值计算。
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。
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
功能开关
最小实例数
最大实例数
0
1

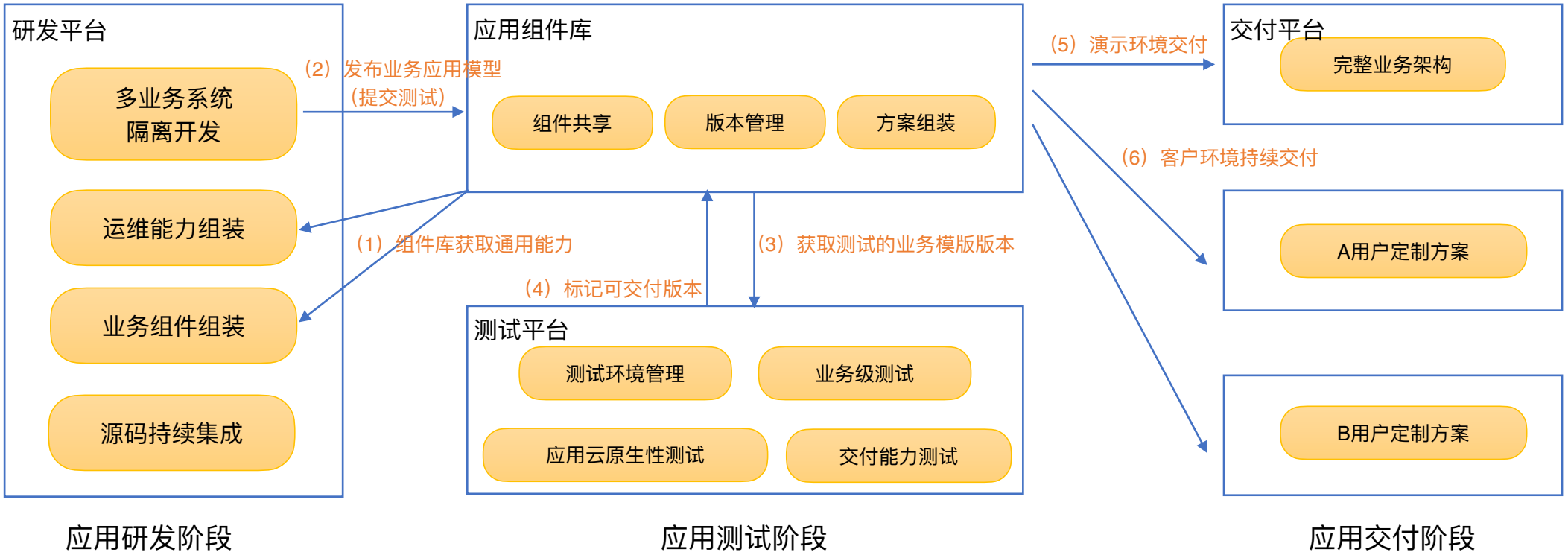
应用模型定义的UI化


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
官网 企业服务 社区
* 名称: 请输入存储名称
* 挂载路径: 请输入挂载路径
存储配额(GB): 请输入存储配额
* 类型: 共享存储（文件） 临时存储
本地存储 高可用型阿里云盘
阿里云ESSD云盘 阿里云SSD云盘
阿里云高效云盘
延迟绑定型阿里云SSD云盘
rainbondvolumerwx
取 消
确 认

第四部分

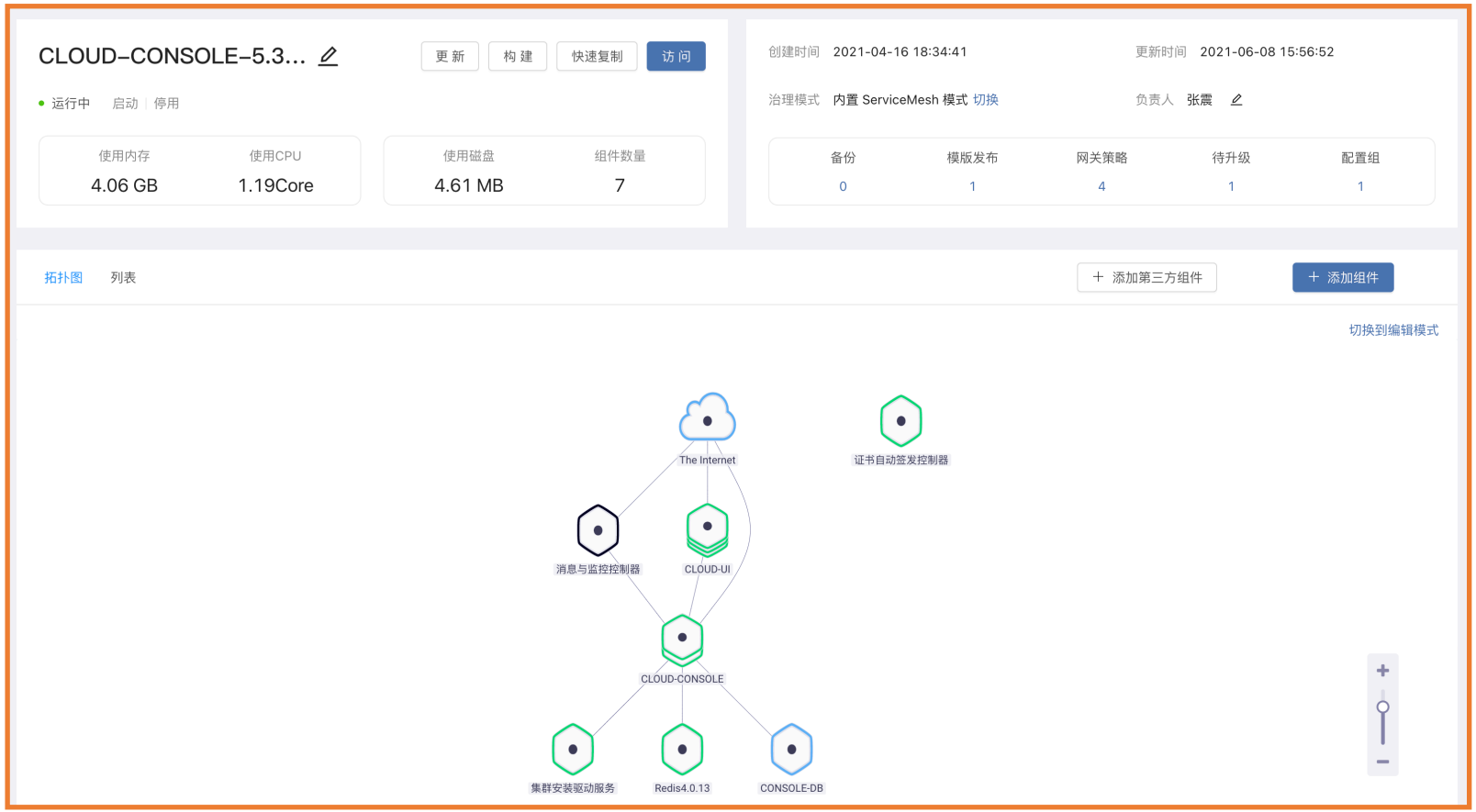
2B交付版本的DevOps

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



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

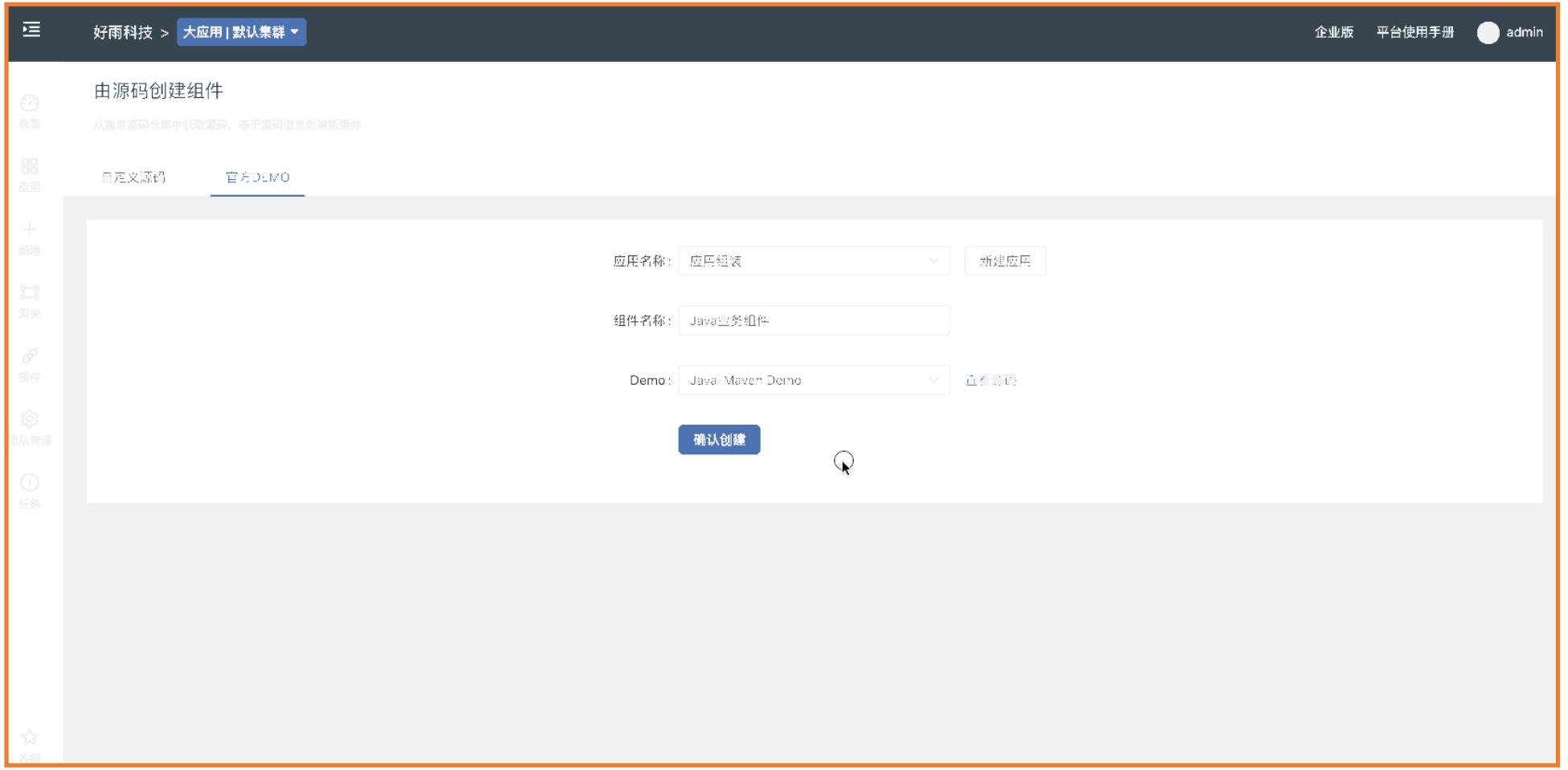
云原生平台

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

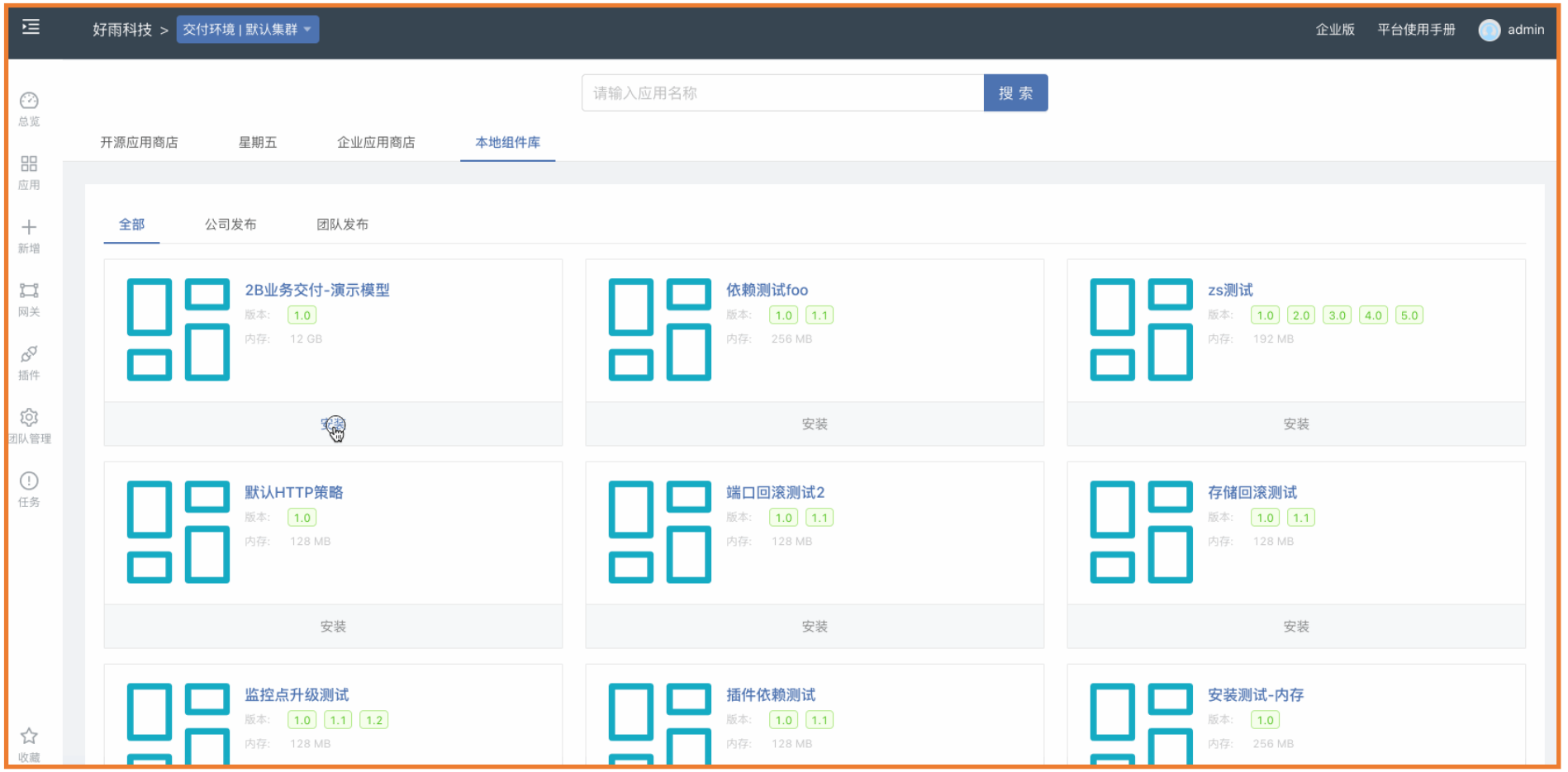
自动资源调度

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

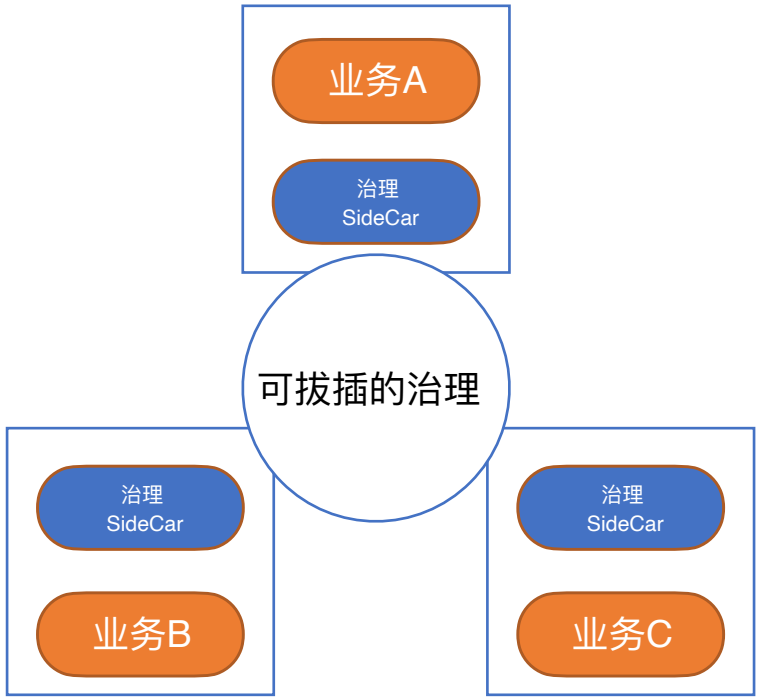
应用治理架构与业务架构解耦合

（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

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