持续部署

操作指南

产品文档

腾讯云

文档目录

操作指南
权限控制

最近更新时间：2021-11-26 14:39:32

本文为您详细介绍在 CODING 持续部署中的权限控制。

前提条件

使用 CODING 项目管理的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入管理设置

1. 登录 CODING 控制台，单击团队域名进入 CODING 使用页面。
2. 鼠标移至右上角头像后会出现下拉菜单，进入团队管理界面，单击权限配置进入管理界面。

权限控制

在默认情况下，CODING 持续部署的权限控制：

- 团队所有者：具备部署管理权限
- 团队管理员：具备部署管理权限
- 团队普通成员：不具备部署管理权限

CODING 推荐团队中建立独立的**运维**角色，并赋予**部署管理**的权限，并且指定团队成员担任运维角色，以实现最小权限原则。如下图所示，设置一个**运维**角色，可单击**添加用户组**设置更多角色。

推荐在 DevOps 实践中，将持续部署过程的操作简单地划分为两类角色：

- **运维：配置持续部署过程（配置应用、配置发布流程、配置审批流程）**
- **开发：提交发布单进行发布（提交发布单、等待审批完成、查看发布过程）**

部署控制台

最近更新时间：2021-08-26 10:45:33

本文为您详细介绍 CODING 持续部署中的控制台。

前提条件

使用 CODING 项目管理的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**立即使用**进入 CODING 使用页面。

2. 单击工作台首页左侧的  ，进入持续部署控制台。

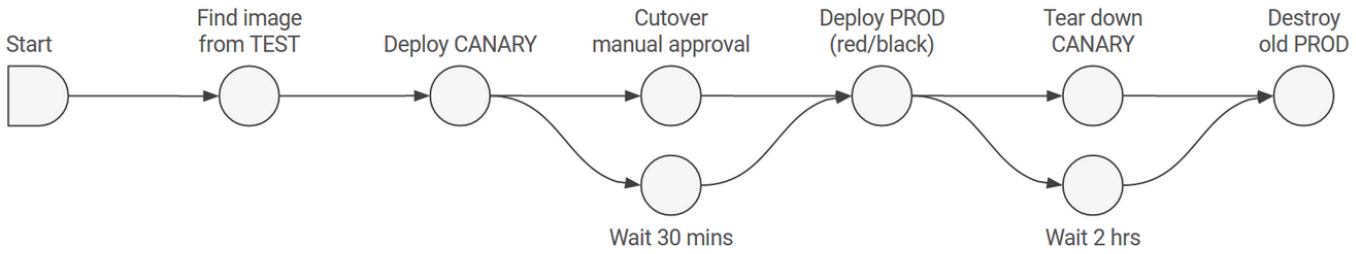
功能介绍

CODING 持续部署控制台基于 Spinnaker 实现，是一个能够管理应用与云账号的综合控制中心。控制台中运维类角色可以在控制台中管理待部署的应用列表，配置部署流程，查看和管理应用集群，对集群进行一些点对点的操作（扩缩容，停止，回退等）。

您可以在 CODING 团队首页中快速进入部署控制台。

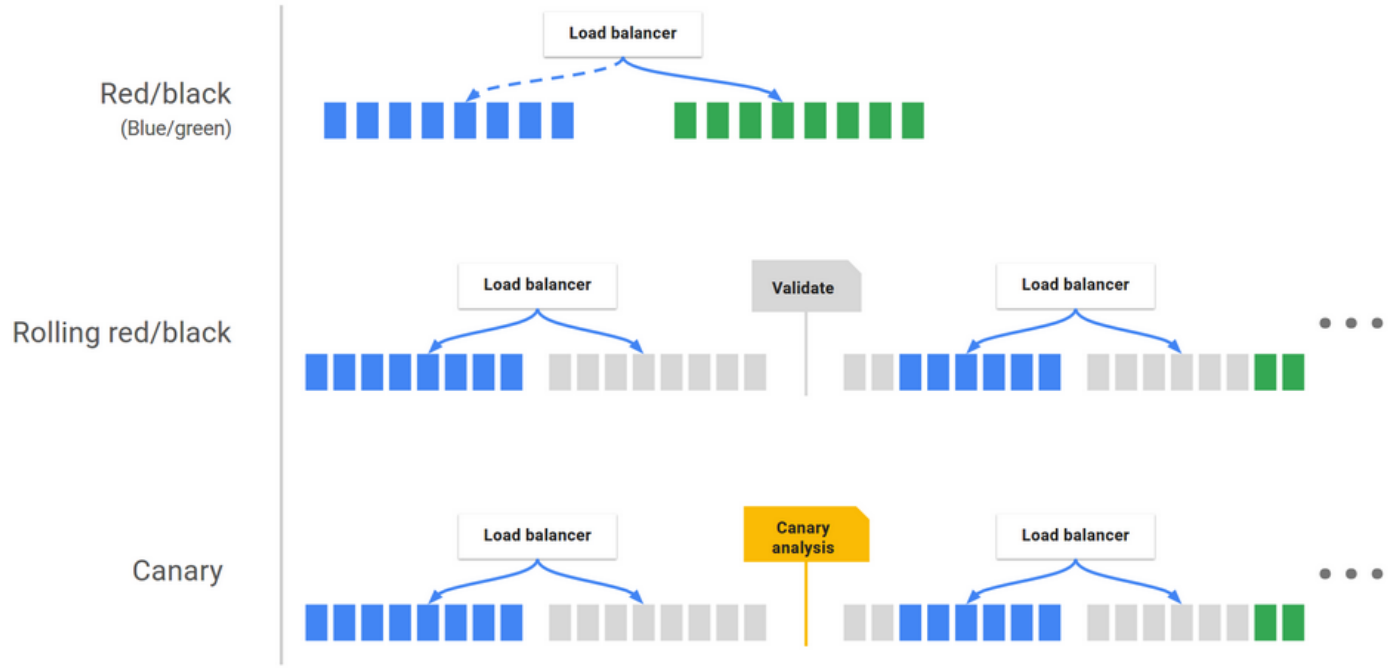
部署流程

部署流程由一系列的**阶段**组成，能够将持续部署流水线化。部署流程可以手动触发执行，也支持配置自动触发，例如由 CODING Docker 仓库触发、Webhook 触发、定时触发等。此外，可以配置引用制品、参数、通知和串行并行逻辑。**阶段**是持续部署流程里的一个自动构建模块，您可以在部署流程中定义各个阶段的执行顺序，以实现灵活的自动化部署。CODING 持续部署提供了很多阶段模板供您选择使用，例如人工确认、前置条件检查、Deploy（部署）等。



部署策略

CODING 持续部署支持精细化的部署策略，例如红/黑（蓝/绿）部署、滚动红/黑策略和灰度部署等。用户可以为每个环境使用不同的部署策略，可以在测试环境中使用红/黑策略，生产环境里使用滚动红/黑策略。在部署策略中已经对必要的步骤进行封装，不需要复杂操作就可以实现企业级发布。



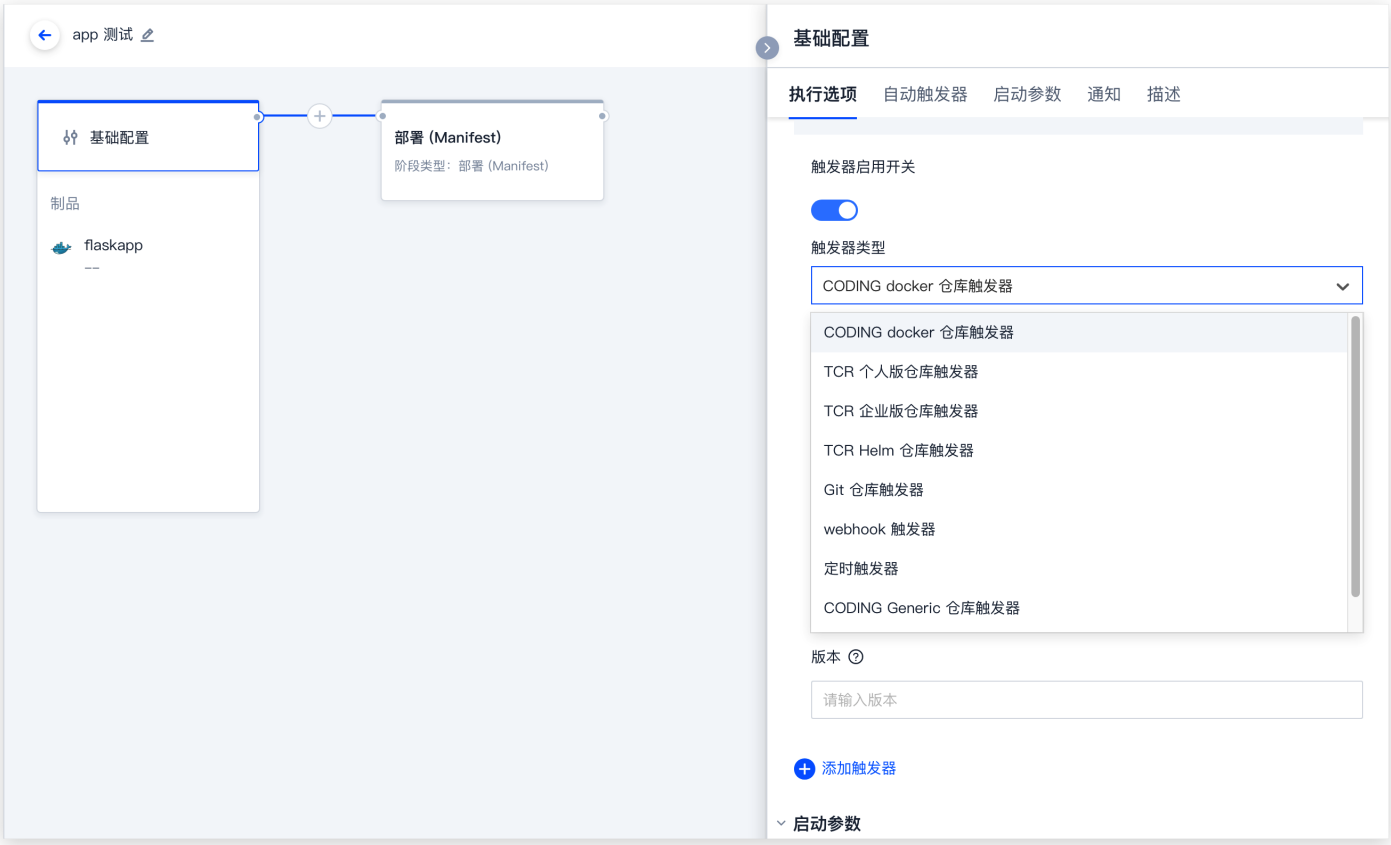
基础设施管理

CODING 持续部署基于 Spinnaker CloudDriver 组件开发，能够兼容适配不同的云平台，实现高效云资源管理。基础设施包含如下几块：

- 服务组：服务组是最基本的资源管理单元，用于标识可部署的制品（如：VM 镜像、Docker 镜像）以及实例数量、自动伸缩策略、元数据等可配置项。服务组还可能关联负载均衡器和安全组。当部署完成后，服务组就相当于一组运行中的软件实例集合（如：腾讯云弹性伸缩组、Kubernetes pods）。
- 负载均衡器：用于将外部网络流量重定向到服务组中的运行实例，支持指定一系列规则对运行实例做健康检测。
- 安全组：定义了网络访问权限，由 IP、端口和通信协议组成安全组规则。
- 集群：由用户定义的，对服务组的逻辑分组。
- 应用：CODING 持续部署以应用作为基本部署单位。应用包含若干个应用集群、负载均衡器和安全组等。应用通常代表您想要部署的服务、配置、以及运行所需的基础设置。推荐将一个应用对应至微服务架构中的一个服务。

触发器

在保留 Spinnaker 部分原生触发器类型的基础上， CODING 部署控制台扩充了触发器类型，使之能够与 CODING 上游制品库匹配。



制品类型改造

在保留 Spinnaker 部分原生制品类型的基础上，CODING 部署控制台在 **Git 仓库文件**制品类型中扩充了对 CODING 代码库的支持，在 **Docker 镜像**制品类型中扩充了对 CODING Docker 镜像制品的支持，已支持 War 包、Helm 包等更多的制品类型。

名词解释

- 实例：运行中的容器或 VM 实例。
- Stack：由用户自定义的，对集群的逻辑分组，如prod, staging, test。
- Detail：由用户自定义的，用于标识集群的三级字段。例如具有相同 \${Application}-\${Stack}-${Detail} 属性的服务组属于同一个集群。

云账号

最近更新时间：2023-08-10 11:07:41

本文为您详细介绍 CODING 持续部署中的云账号。

前提条件

使用 CODING 项目管理的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击团队域名进入 CODING 使用页面。
2. 进入 CODING 工作台首页后，选择左侧**功能设置** > **持续部署**，进入云账号管理页面。

功能介绍

云账号是访问云资源的凭证，只有配置了云账号，CODING 持续部署才能实现对云资源的部署管理和基础设施管理。目前支持三种云账号类型：

- 腾讯云 TKE：如果您是从腾讯云开发者平台入口注册登录，才会显示此类账号。
- Kubernetes：支持 Kubeconfig 和 Service Account 两个常用凭据。
- 腾讯云账号：即腾讯云 API 密钥。

您可以在 CODING 首页工作台中的**功能设置** > **持续部署** > **云账号**页面管理云账号。

腾讯云 TKE

1. 云账号类型选择腾讯云 TKE，按照指引完成与云账号名下的集群绑定。若没有集群请前往 腾讯云 TKE 创建集群。

2. 选择拟部署的集群，单击**确定**后会自动验证该账号名下的集群并完成互联。

Kubernetes

Kubernetes 云账号支持 Kubeconfig 和 Service Account 两种常用凭据。以 Kubeconfig 为例：

登录云计算网页控制台，复制 Kubeconfig，并将 CODING IP 段添加至集群外网访问控制列表（白名单）。

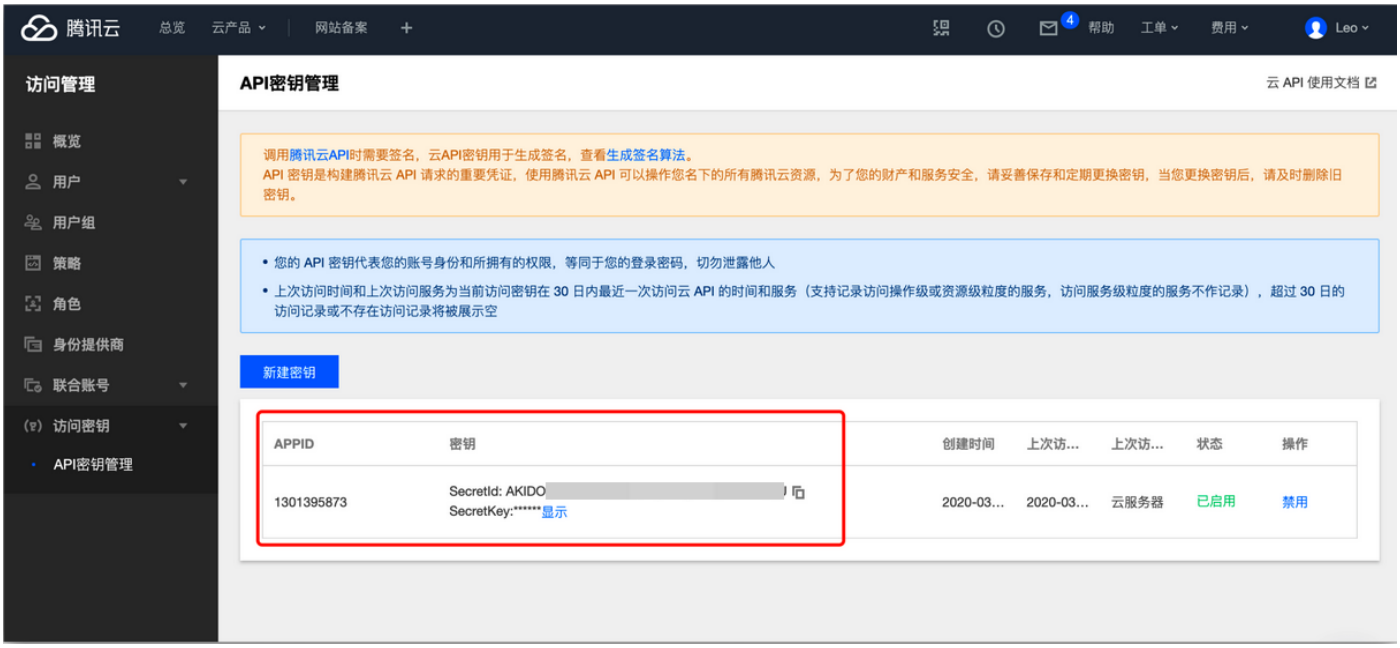
> CODING 持续部署的公网 IP 段：
> 212.64.105.0/24、212.129.144.0/24

将 Kubeconfig 粘贴到 CODING 中，选择 Cluster Context，完成云账号添加。

腾讯云账号

1. 云账号类型选择腾讯云账号，输入云账号名称，并选择区域。支持多选区域，CODING 持续部署将获得勾选区域的腾讯云资源管理权限。

2. 从腾讯云 访问管理控制台 拷贝密钥信息。



3. 将拷贝的 SecretID 和 SecretKey 粘贴到对应的文本框，单击**确定**完成云账号添加。

应用与项目

最近更新时间：2021-12-13 14:25:38

本文为您详细介绍 CODING 持续部署中的应用与项目。

前提条件

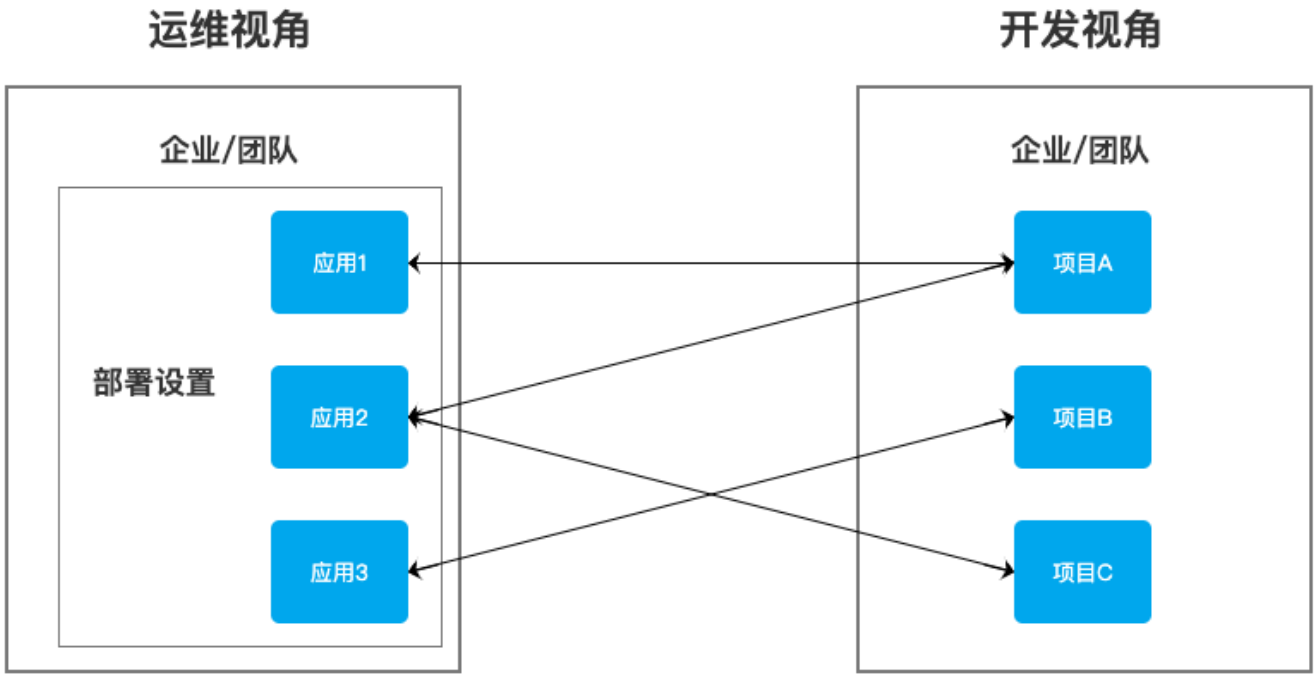
使用 CODING 项目管理的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**立即使用**进入 CODING 使用页面。
2. 单击工作台首页左侧的 ，进入持续部署控制台。

功能介绍

CODING 持续部署中的应用和项目都是属于企业/团队的一级资源，它们之间为一对多关系，即一个项目包含多个应用，一个应用从属于多个项目。



在该设计下，运维人员可以专注于应用的持续部署管理（部署流程、基础设施等），而非运维人员（一般指开发）只需要在项目维度操作（提交发布单，查看发布详情），使运维能够专注于在云的基础上做基础设施运维，开发在项目内就能够进行大部分业务运维并完成从需求到发布的完整闭环。

应用

应用是 CODING CD 中的基本部署单位。应用包含若干个应用集群，以及安全组和负载均衡器等。应用对部署的软件集合进行抽象，通常代表您想要部署的服务、配置、以及运行所需的基础设置。推荐的做法是一个应用对应微服务架构中的一个服务。

对应关系示例

在微服务架构下的微服务对应于一个 CODING 持续部署的应用，当然您也可以在了解对应关系的情况下依照自己的偏好来设定对应关系。下面是一个典型的团队、项目、应用、集群、云账号之间的关系示例：

团队：XXX 科技有限公司	
云账号	• 自建 Kubernetes Service Account • 腾讯云北京 TKE 集群 Service Account • 腾讯云中国香港 API Key
项目1：车载用品电商站点项目	• 应用1：车载电商后端 • 应用2：车载电商前端 • 应用3：物流管理服务
项目2：服装电商站点项目	• 应用1：服装电商后端 • 应用2：服装电商前端 • 应用3：物流管理服务

部署控制台	• 应用1：车载电商后端
	  ◦ 测试集群
	  ◦ 生产集群
	• 应用2：车载电商前端

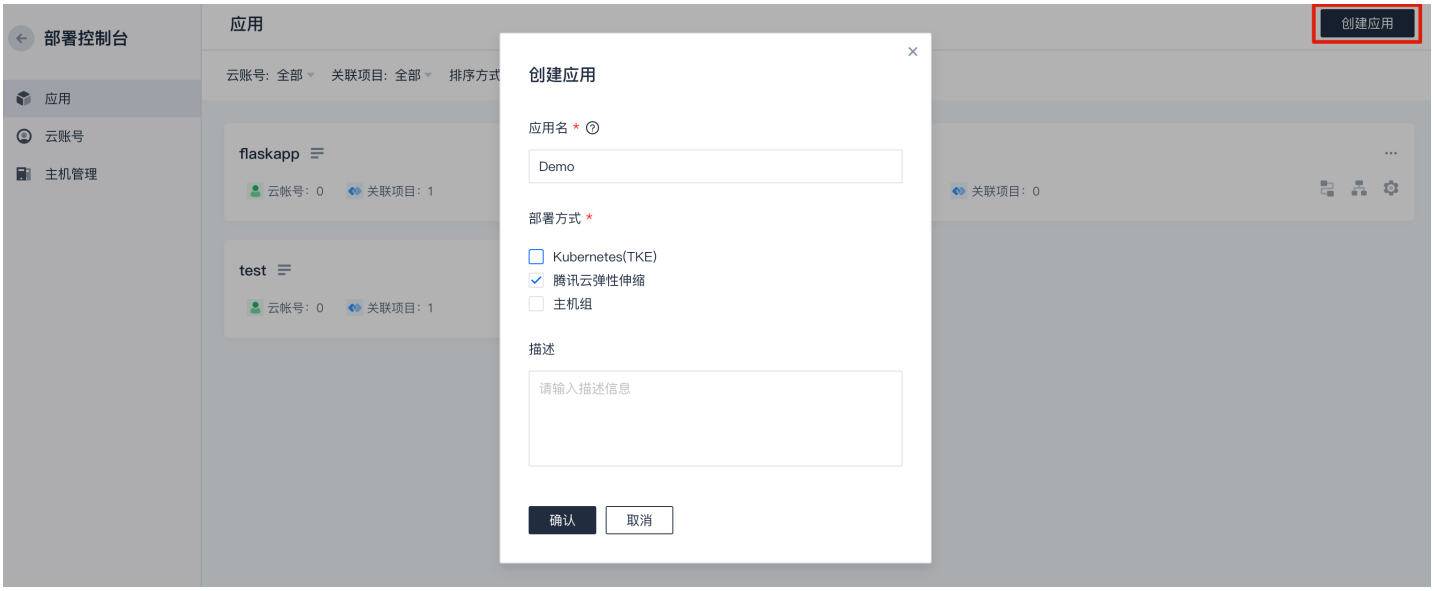
	◦ 测试集群 ◦ 生产集群 • 应用3：物流管理服务 ◦ 车载电商测试集群 ◦ 车载电商生产集群 ◦ 服装电商测试集群 ◦ 服装电商生产集群 • 应用4：服装电商后端 ◦ 测试集群 ◦ 生产集群 • 应用5：服装电商前端 ◦ 测试集群 ◦ 生产集群

云账号绑定

云账号是访问云资源的凭证，进入 CODING 部署控制台创建应用，单击导航栏**应用 > 创建应用**。在进行应用创建之前，请确保您已经完成了 云账号绑定。

应用创建

单击首页左侧的控制台，单击右上角**创建应用**。



与项目关联

在部署控制台内完成应用创建后，可以直接在控制台首页将应用与项目相关联。

新建发布单

当运维人员完成对应用的 部署流程配置 后，开发人员在项目内就可以实现从项目协同到应用发布的 DevOps 闭环。典型的场景是当有新版本需要发布时，开发人员在**持续部署** > **Kubernetes** 页面新建发布单，发布单自动触发

部署流程执行，开发可随时查看发布状态和历史详情。

管理应用

在部署控制台新建应用后可以在应用的配置中调整应用属性与通知，或删除应用。

应用通知

目前支持 CODING 站内通知、企业微信、钉钉和飞书四种通知方式：

显示 / 隐藏功能入口

对于不需要显示的功能入口，可以在**特性**栏将其禁用，这里的禁用并不会删除相应的数据，仅表示在控制台界面隐藏。可以隐藏部署流程、集群、负载均衡器和安全组功能入口：

添加实例的自定义属性链接

在**集群 > 服务组 > 实例详情**面板可以查看运行实例的自定义链接，自定义链接提供关于实例的简略信息，如：日志、健康状态等。

自定义链接对应的 IP 可以是公有或私有 IP，默认端口为 80；如果需要设置其他端口号，在 Path 文本框以 : 开头，如： :7002/health 。

1. 在**链接**一栏，单击 **Add Section**。
2. **Section Heading** 输入自定义链接标题。
3. **Links** 输入自定义链接名称，以及 URL。

> **说明：**
> URL 字段支持使用表达式引用更多的实例属性。例如对于腾讯云实例，可以使用 {region} 引用实例的所在地域。

5. 单击 **Add Link** 在同一属性下添加更多链接。
6. 单击 **Add Section** 添加新的自定义属性链接。
7. 单击**撤销**取消添加操作。**撤销**不会删除已保存的自定义属性链接。
8. 单击**保存**完成操作。

流量保护

> **说明：**
> 流量保护旨在确保任何时间至少有一个实例处于正常运行状态。

启用流量保护功能后，如果用户或者脚本尝试删除、禁用或对服务组进行伸缩容操作，CODING 控制台会对操作进行验证以确保集群中至少有一个实例在正常运行，否则将会拒绝用户或脚本请求。

1. 在**流量保护**栏，单击**添加流量保护**。
2. 以下是需要填写的字段：

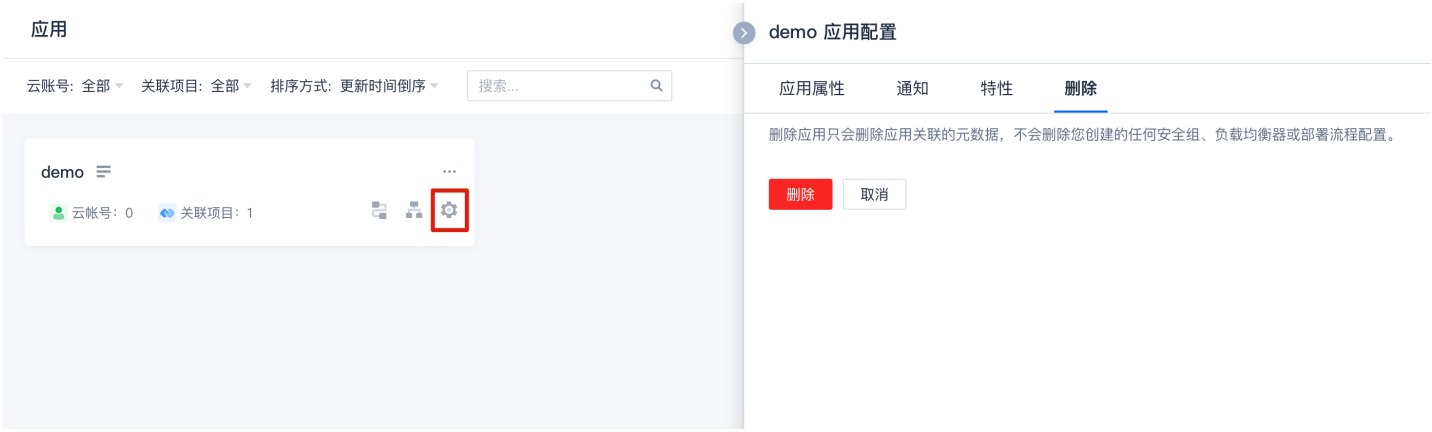
字段	必填项	说明
云账号	是	设置流量保护的云账号
地域	是	可选的地域，* 表示选择所有地域
分组	否	设置流量保护的集群分组，如果留空表示选择不属于任何分组的集群
详情	否	详情是区分集群的三级字段，具有相同 ${Application}-${Stack}-${Detail} 的服务组属于同一个集群

3. 单击**保存**使配置生效。

应用删除

如果应用中有服务组，需要先删除服务组。

进入 部署控制台 后，单击应用右下角**齿轮图标**，进入应用配置页后单击**删除**。



部署流程
部署流程介绍

最近更新时间：2021-11-26 14:47:02

本文为您详细介绍 CODING 持续部署中的部署流程。

前提条件

使用 CODING 项目管理的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**立即使用**进入 CODING 使用页面。
2. 单击工作台首页左侧的  ，进入持续部署控制台。

功能介绍

部署流程是实现持续部署最核心的模块。其强大之处在于支持阶段以任意的顺序组合，这样的能力让部署流程具备出色的灵活性、一致性和可重复性。

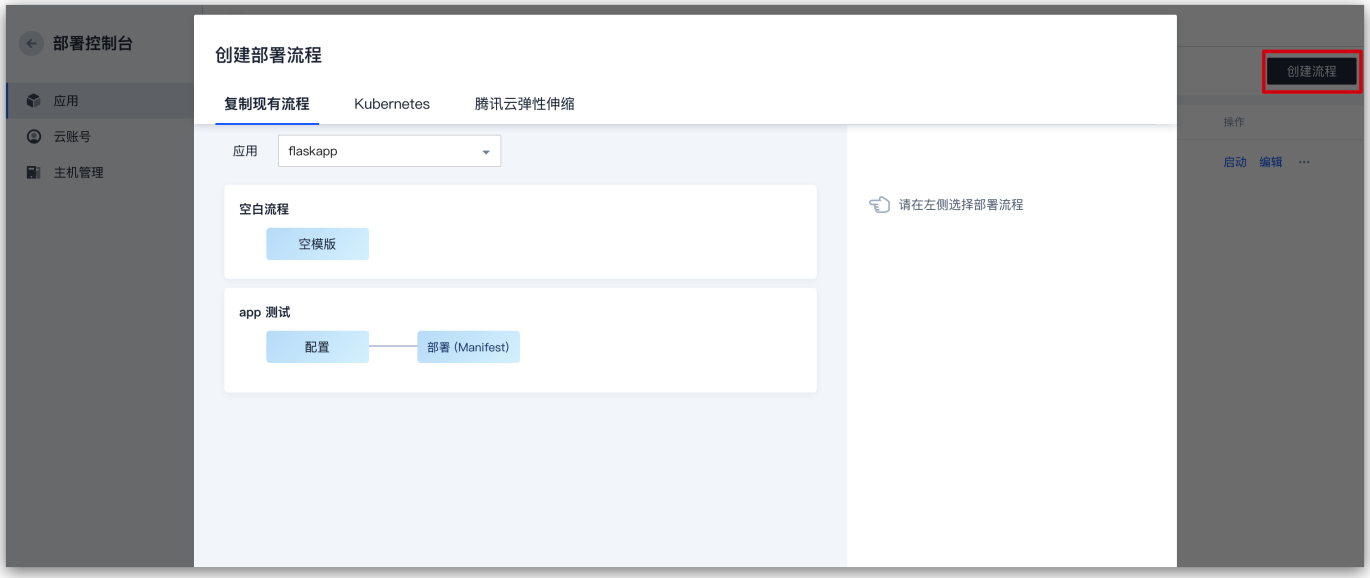
- 灵活性：支持串行、并行控制
- 一致性：支持多种部署策略，回滚能力，确保发布结果符合预期
- 可重复性：部署流程可重复执行，阶段可被其他部署流程复制使用

您可以配置完全自动化的部署流程，也可以在某些阶段加入手工判断条件。此外部署流程支持多种事件的自动化触发，如 Webhook 触发、由其他部署流程触发等。

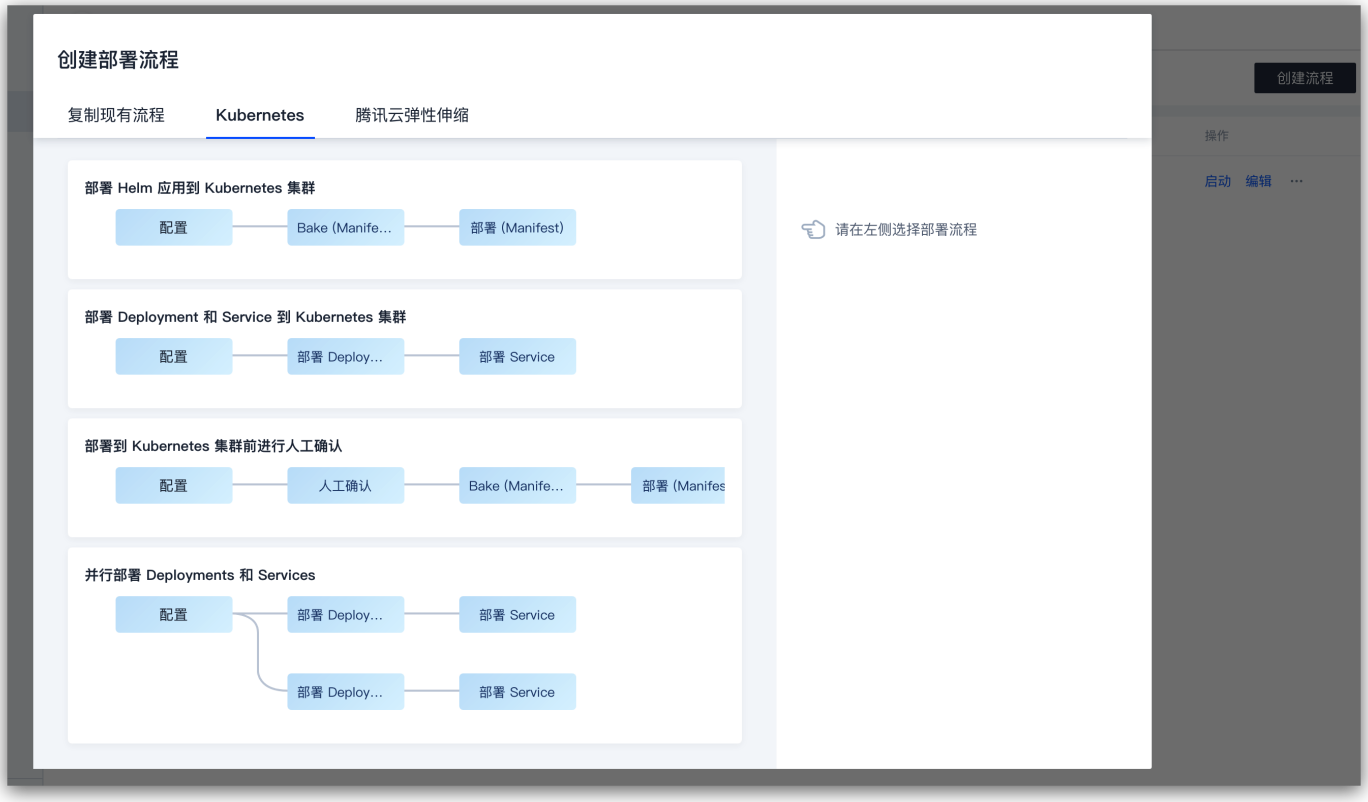
新建部署流程

前往部署控制台，单击应用卡片右下方的部署流程按钮。

1. 单击右上角的**创建流程**按钮。



2. 您可以复制在其他应用中创建的流程，或通过空白流程自行创建。CODING 亦提供了 Kubernetes 与腾讯云弹性伸缩参考流程模板。



基础配置

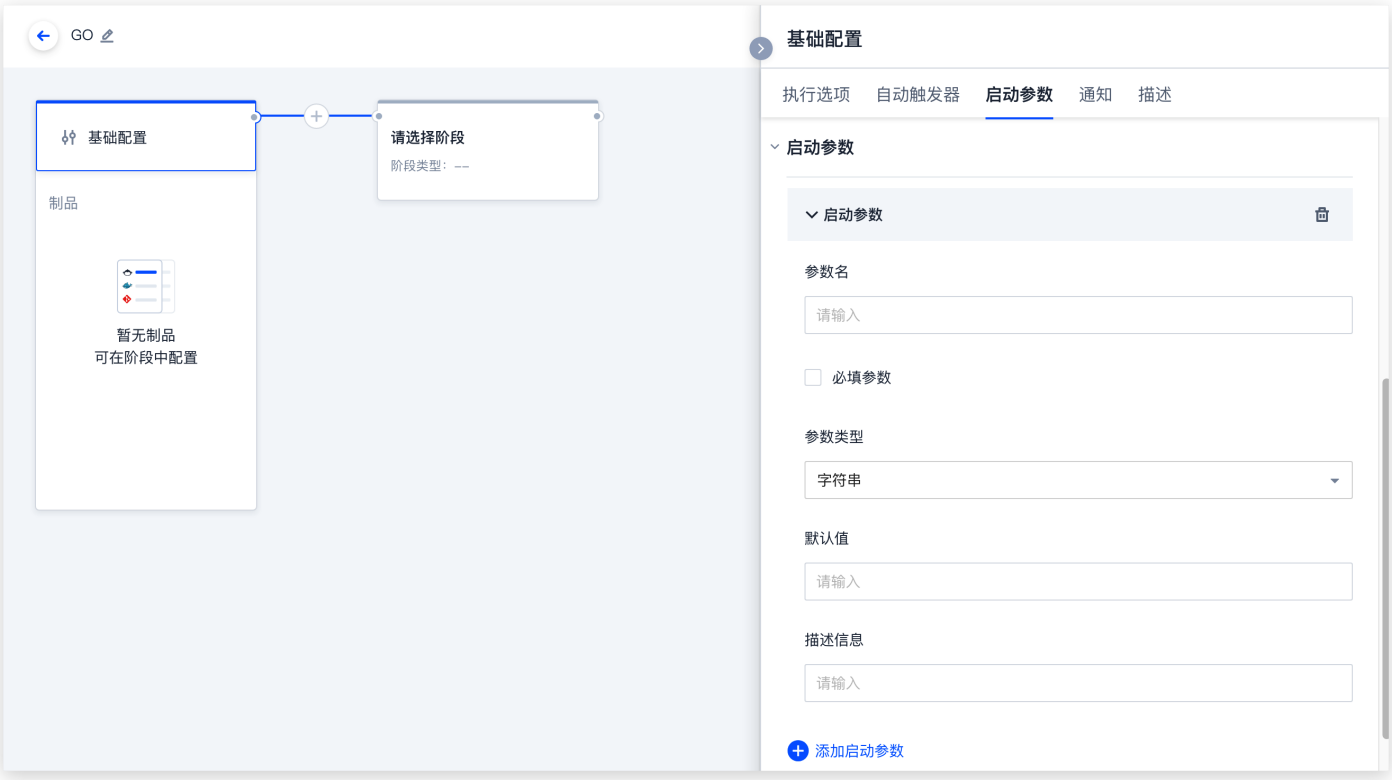
应用的基础配置可以理解为构建整体的初始环节，既可以设置触发条件，也可以配置部署流程的通知方式等。

自动触发器

自动触发器支持 CODING Docker 制品仓库、TCR 个人版仓库触发器、Git 仓库触发器等触发条件。

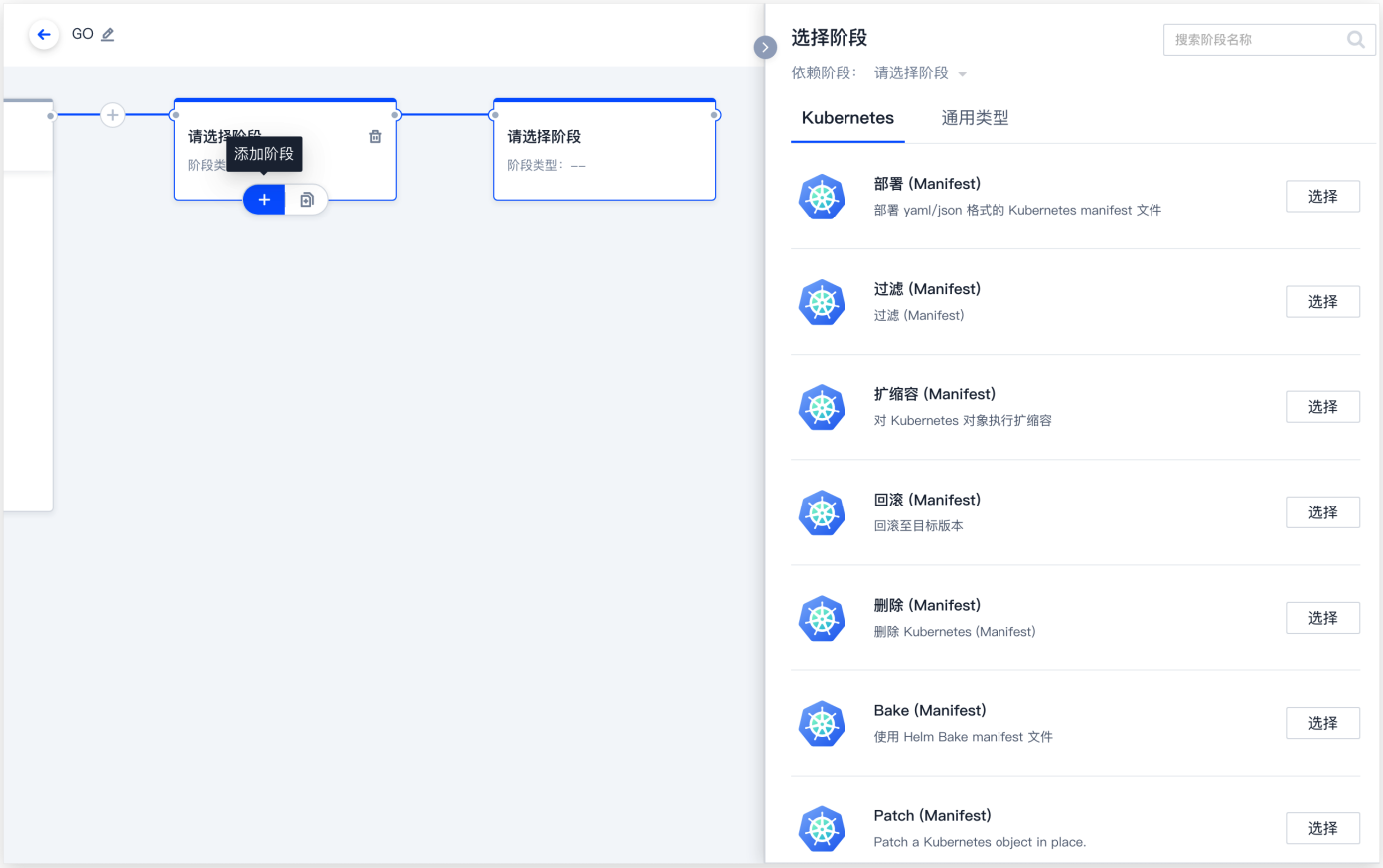
添加部署流程参数

在部署流程配置页面，单击添加启动参数，即可开始填写参数。



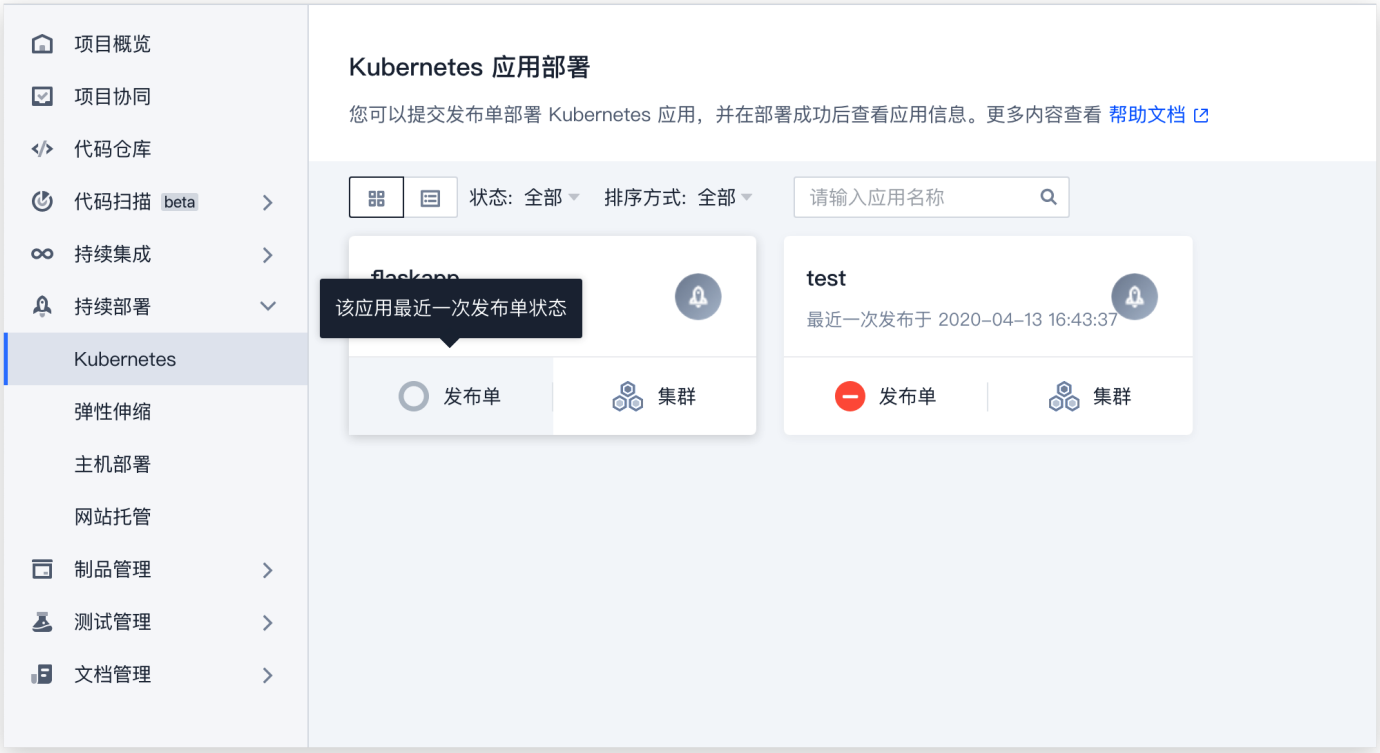
添加阶段

在部署流程配置页面单击+即可添加新的阶段，右侧列表中支持选择阶段类型。



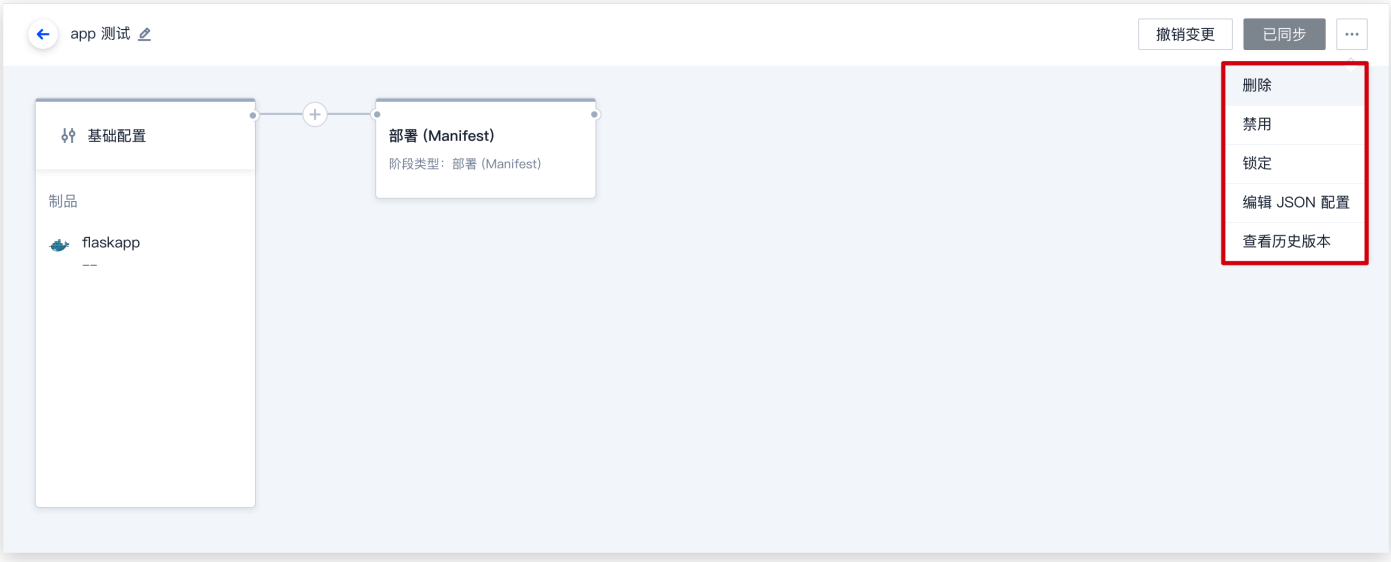
执行部署流程

部署流程配置完成后，您可以通过设置好的触发器以提交自动执行，或在持续部署中提交发布单手动触发部署流程。



部署流程配置

部署流程支持删除、禁用、锁定、查看历史版本与编辑 JSON 配置。



删除部署流程

设置后，将删除此部署流程。

禁用部署流程

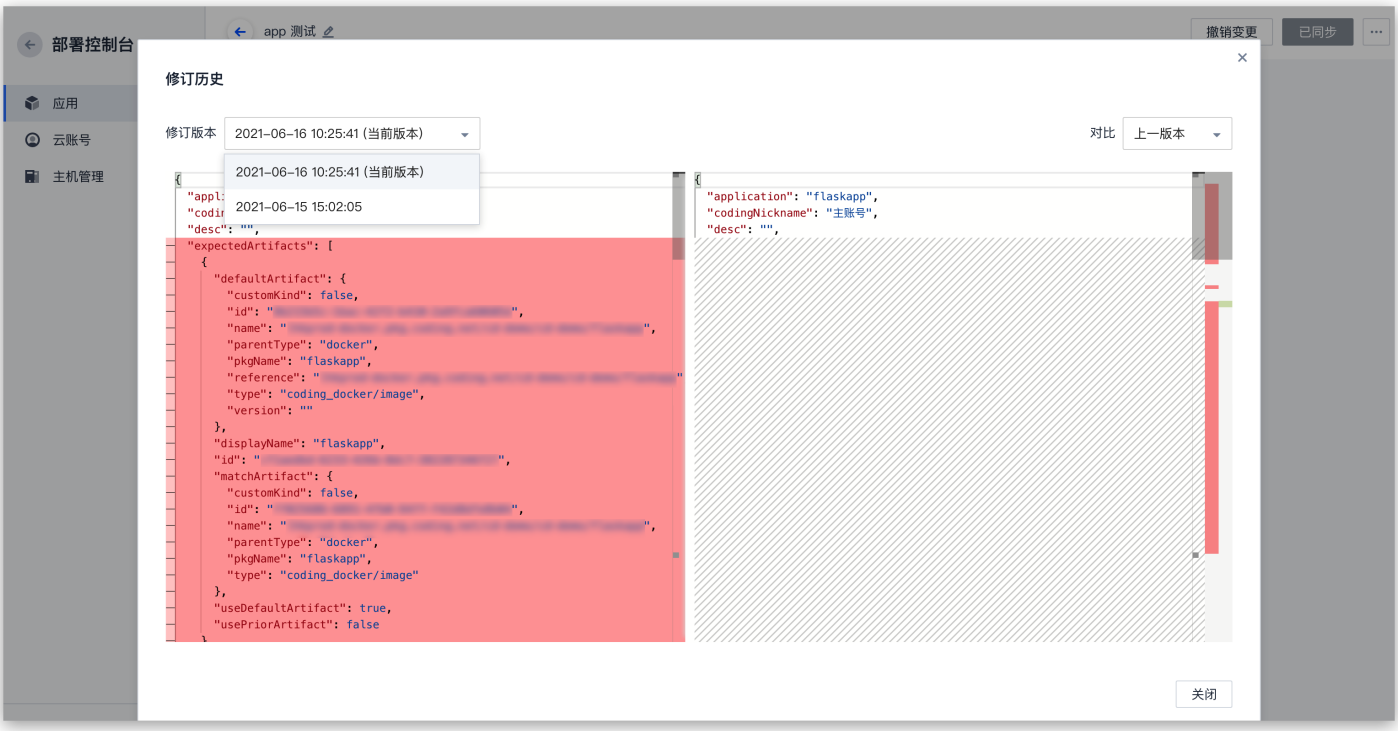
设置后，将禁止任意触发器启动部署流程，包括手动触发。可以选择在团队内整体禁用或仅在项目内禁用。

锁定部署流程

锁定部署流程后，将不能通过部署控制台编辑部署流程。可以选择是否允许通过 API 接口对部署流程进行更新。

查看修订历史

保存新的部署流程配置后，旧版本将会添加到修订历史。您可以在修订历史页对比各版本信息，选择并还原到任意历史版本。



编辑 JSON 配置

在部署控制台中所做的任何更改最终都会以 JSON 格式文件保存，直接编辑部署流程的 JSON 内容可以为部署流程添加新属性或自定义 UI 界面尚未显示的配置项。

> **注意：**
>
> 此种方式允许用户将在文本框内自由编辑部署流程，但有可能会破坏部署流程的可用性，我们提供了从修订历史中恢复到任意指定版本的能力。


部署控制台
应用
云账号
主机管理
app 测试
撤销变更
已同步
编辑 JSON 配置
{
"codingNickname": "主账号",
"desc": "",
"expectedArtifacts": [
{
"defaultArtifact": {
"customKind": false,
"id": "
"name": "
"parentType": "docker",
"pkgName": "flaskapp",
"reference": "
"type": "coding_docker/image",
"version": ""
},
"displayName": "flaskapp",
"id": "
"matchArtifact": {
"customKind": false,
"id": "
"name": "
"parentType": "docker",
"pkgName": "flaskapp",
"type": "coding_docker/image"
},
"useDefaultArtifact": true,
"usePriorArtifact": false
}
取消
保存变更

阶段类型

最近更新时间：2021-11-26 11:23:09

本文为您详细介绍在 CODING 持续部署中部署流程的阶段类型。

前提条件

使用 CODING 项目管理的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击**立即使用**进入 CODING 使用页面。
2. 单击工作台首页左侧的  ，进入持续部署控制台。

通用类型

编辑部署流程时可以为阶段选择阶段类型。

预置条件检查

在执行下一步之前检查前置条件，例如检查集群规模或某个阶段的状态。

自定义变量

添加自定义变量（ key/value 键值对），在此阶段的定义的变量可以被下游阶段引用。

人工确认

在执行下一步之前等待人工确认。可以在人工确认阶段增加指引说明帮助确认人进行人工确认，或者添加输入选项让用户选择。这些输入选项可以决定下游阶段的执行行为。例如，可以使用 预置条件检查 来确保只有满足特定的条件时才执行相应的阶段。

部署流程

将其他部署流程作为子部署流程执行。您可以执行当前应用的部署流程，也可以执行具有访问权限的其他应用的部署流程。在阶段执行结束前可以选择是否等待子部署流程的执行结果。如果选择等待执行结果，此阶段的状态即为子部署流程的最终执行状态；否则，只要子部署流程开始执行此阶段的状态就会被标记为“成功”。

配置选项说明：

字段	是否必填	说明
应用	是	列出所有具体访问权限的应用
部署流程	是	列出应用下所有的部署流程
是否等待执行结果	否	如果选择等待执行结果，此阶段的状态即为子部署流程的最终执行状态；否则，只要子部署流程开始执行此阶段的状态就会被标记为“成功”。

等待

等待一定的时间段后继续执行。在部署流程执行过程中可以手动减少等待时间或直接跳过等待。等待时间支持表达式。

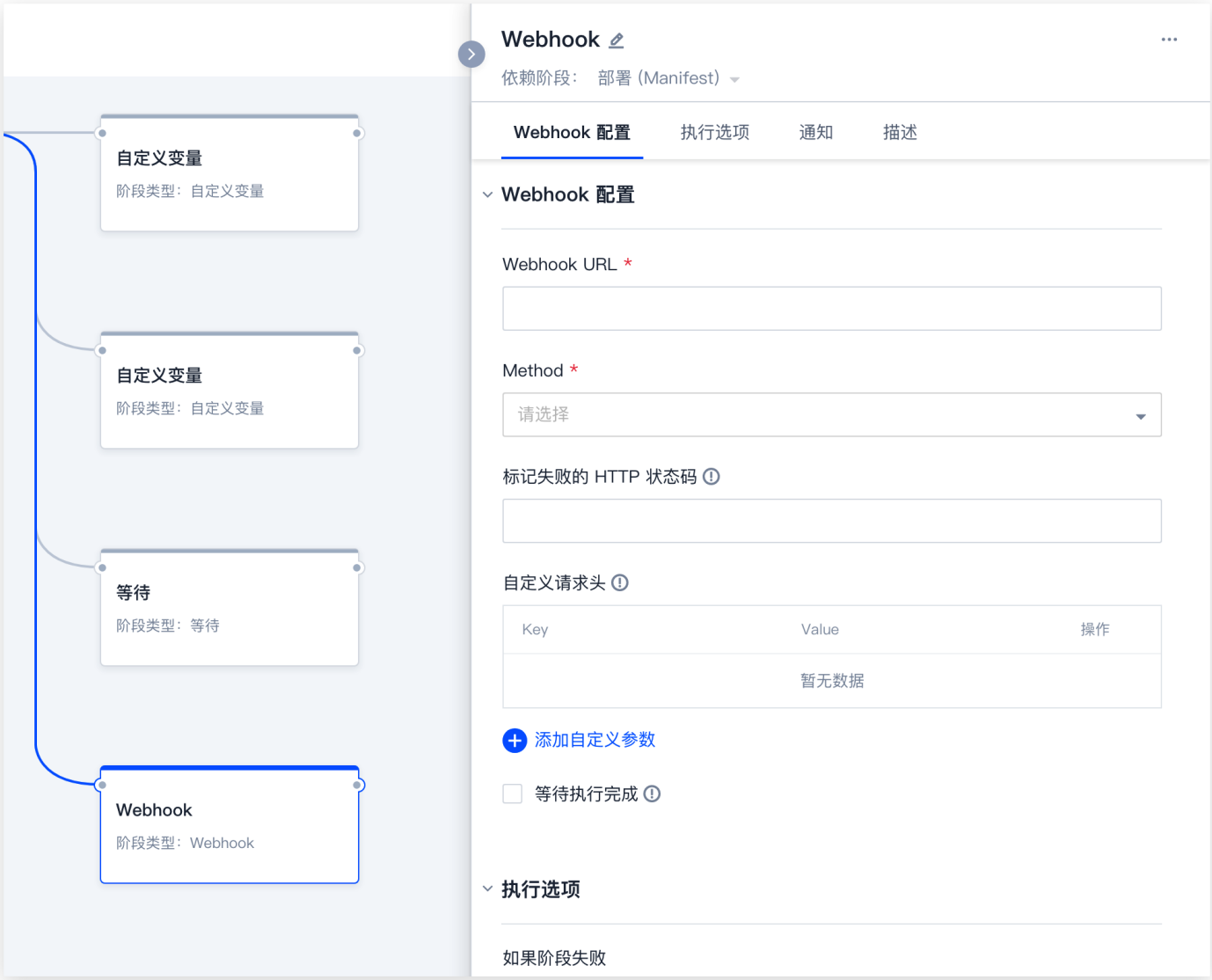
Webhook

支持调用外部系统 API 作为部署流程的阶段。

利用指定 Webhook 的目标 URL 和 HTTP 方法，支持自定义 header 和 JSON 格式的 payload。默认情况下，如果 Webhook 调用返回 2XX 或 3XX 表示阶段执行成功，返回 4XX 或 5XX 表示执行失败。Webhook 的 URL、payload 调用的最终状态都会展示在部署流程执行详情中。

用户可以在 URL 字段和 payload 中使用部署流程表达式。当阶段执行完成后，阶段上下文的 Webhook 对象包含了 payload 内容，可以在后续的部署流程表达式中引用 payload。例如以下表达式可以获取 Webhook 执行的最终状态：

```
${#stage("My Webhook Stage")["context"]["webhook"]["statusCode"]}
```



禁用集群

禁用集群意味着让集群保持运行状态但不处理流量。如果需要的话，用户可以指定运行一定数量的服务组而禁用剩下的。

配置选项说明：

- **云服务（Provider）选择 Kubernetes**

字段	是否必填	说明
云服务	是	云服务类型，目前支持 Kubernetes 和 腾讯云
云账号	是	管理资源对象的云账号
Namespaces	是	服务组所属的命名空间

集群	是	服务组所属的集群
禁用选项	是	指定具体的禁用规则

- 云服务（Provider）选择 Kubernetes

字段	是否必填	说明
云服务	是	云服务类型，目前支持 Kubernetes 和 腾讯云
云账号	是	管理资源对象的云账号
Namespaces	是	服务组所属的命名空间
集群	是	服务组所属的集群
禁用选项	是	指定具体的禁用规则
健康检查	是	执行此任务时，仅参考腾讯云提供的健康检查

集群缩容

可以选择是否对活跃的（正常运行）的服务组缩容；另外支持使指定数量的服务组保持当前规模，将其他服务组缩容。

- 云服务（Provider）选择 Kubernetes

字段	是否必填	说明
云服务	是	云服务类型，目前支持 Kubernetes 和 腾讯云
云账号	是	管理资源对象的云账号
Namespaces	是	服务组所属的命名空间
集群	是	服务组所属的集群
缩容选项	是	指定具体的缩容选项

- 云服务（Provider）选择 腾讯云

字段	是否必填	说明
云服务	是	云服务类型，目前支持 Kubernetes 和 腾讯云
云账号	是	管理资源对象的云账号

地域	是	服务组所属的地域
集群	是	服务组所属的集群
健康检查	是	执行此任务时，仅参考腾讯云提供的健康检查

启用服务组

启动服务组即让被禁用的服务组重新处理请求流量。负载均衡器的配置决定了新旧服务组之间的路由规则。启用服务组的同时也启用伸缩容策略。

配置选项说明：

- 云服务（Provider）选择 Kubernetes

字段	是否必填	说明
云服务	是	云服务类型，目前支持 Kubernetes 和 腾讯云
云账号	是	管理资源对象的云账号
Namespaces	是	服务组所属的命名空间
集群	是	服务组所属的集群
目标服务组	是	指定服务组需满足的匹配规则

- 云服务（Provider）选择 腾讯云

字段	是否必填	说明
云服务	是	云服务类型，目前支持 Kubernetes 和 腾讯云
云账号	是	管理资源对象的云账号
地域	是	服务组所属的地域
集群	是	服务组所属的集群
目标服务组	是	指定服务组需满足的匹配规则
健康检查	是	执行此任务时，仅参考腾讯云提供的健康检查

禁用服务组

禁用服务组意味着让服务组保持运行状态但不处理流量。另外针对禁用服务组的伸缩容操作也会被禁用。禁用服务组使得在新旧服务组之间做流量切换变得很简单。在阶段启动之前用户必须指定禁用最新、最旧或次新的服务组。

> 说明：
> 配置选项参见 启用服务组。

销毁服务组

销毁指定集群的服务组和相应资源。在阶段启动之前用户必须指定销毁最新、最旧或次新的服务组。

配置选项说明：

- 云服务（Provider）选择 Kubernetes

字段	是否必填	说明
云服务	是	云服务类型，目前支持 Kubernetes 和 腾讯云
云账号	是	管理资源对象的云账号
Namespaces	是	服务组所属的命名空间
集群	是	服务组所属的集群
目标服务组	是	指定服务组需满足的匹配规则

- 云服务（Provider）选择 腾讯云

字段	是否必填	说明
云服务	是	云服务类型，目前支持 Kubernetes 和 腾讯云
云账号	是	管理资源对象的云账号
地域	是	服务组所属的地域
集群	是	服务组所属的集群
目标服务组	是	指定服务组需满足的匹配规则

调整服务组规模

以当前服务组规模的百分比或指定数量来调整服务组大小。支持如下调整策略：

- 扩容：增大目标服务组的规模

- 缩容：减小目标服务组的规模
- 扩容至相对最大规模：将目标服务组规模扩容至与当前集群中最大服务组一致
- 调整为特定规模：将目标服务组调整至特定规模

腾讯云类型

Bake

从指定的软件包 Bake 云服务器镜像，Bake 是指创建云服务器镜像的过程。CODING 部署控制台基于 Hashicorp 的 Packer 抽象出 Baker 阶段，提供了默认的 Packer 模板和基本的云服务器镜像，以帮助用户快速入门。

需要注意的时，如果 Spinnaker 检测到不需要进行新的 Bake，则跳过 Bake 过程。 CODING 部署控制台会根据 Bake 阶段参数（基础操作系统，版本化的软件包等）为每次 Bake 生成唯一的键（key）。如果软件包或 Bake 阶段参数发生了更改，则会触发新的 Bake 操作。 要更改默认行为并在部署流程每次运行时重新 Bake 镜像，请在阶段阶段配置中勾选 Rebake 。详情请参见 Hashicorp Packer。

部署

通过指定部署策略部署提前准备好的镜像。CODING 持续部署提供了部分内置的部署策略，如：红黑（蓝绿）部署、Highlander 部署。您还可以选择对已有服务组无侵入式的部署方式，或者创建自定义的部署策略。

回滚集群

回滚集群中一个或多个区域的实例。

配置选项说明：

字段	是否必填	说明
云账号	是	腾讯云账号
地域	是	集群所属的地域
集群	是	指定需要回滚的集群
健康检查	是	执行此任务时，仅参考腾讯云提供的健康检查

克隆服务组

将已有服务组的所有属性克隆到新服务组（依赖镜像、容器等）。在创建新服务组时可以选择覆盖克隆服务组的任意属性。

Shrink Cluster

指定保留一定数量的最新或规模最大的服务组，将其他的服务组全部删除。用户可以选择是否删除不满足指定条件的活跃（正常运行）服务组

Modify Scaling Process

暂停/恢复 扩缩容操作

Kubernetes 类型

Bake (Manifest)

使用诸如 Helm 这样的模板渲染器 Bake 资源清单。详情请参见 Bake(Manifest) 阶段详解。

部署（Manifest）

包含两个主要的步骤：

- 指定要部署的 mainfest
- 在 manifest 中指定需要覆盖的制品（如 Docker 镜像）

配置 manifest

根据需求，有两个方法可以指定 manifest：

- 静态：直接在部署流程中指定
- 动态：运行时使用绑定的制品

不管使用哪种方式，都需提前选择 Deploy (Manifest) 阶段：

配置静态的 manifest

如果您提前知道即将部署的资源对应的 manifest（即使不知道制品的版本），那么可以在 Deployment(Manifest) 阶段配置中直接提供 manifest 纯文本内容。

> 说明：
> 选择 Text 类型后，用户在文本框直接编辑 YAML 文件内容。

如果是使用 JSON 定义的部署流程，对应的内容如下：

```
{
"name": "Deploy my manifest", // human-readable name
"type": "deployManifest", // tells orchestration engine what to run
"account": "nudge", // account (k8s cluster) to deploy to
"cloudProvider": "kubernetes",
"source": "text",
"manifest": {
// manifest contents go here
}
}
```

配置动态的 manifests

如果制品没有存储在部署流程仓库中，或者当一个阶段需要部署多种制品时，可以通过绑定制品来配置 manifest。CODING 持续部署的制品允许用户引用任何远程的可部署资源。Deploy(Manifest) 阶段引用的制品必须是包含 Manifest 定义的文本文件，这样的文本文件可能存在于 GitHub 仓库或 GCS。详情请参见 部署流程详细设置。

假设您已经在上游阶段声明了 Expected Artifacts，那么可以在 Deploy(Manifest) 阶段引用：

> **说明：**
> Manifest Source 勾选 Artifact 后，用户可以选择部署上游提供的制品。请确保云账号（Account）拥有下载制品的权限。

上游阶段可能会匹配到多个制品，例如我们可以配置正则表达式 .*\yml，将所有 yml 文件作为制品。Deploy(Manifest) 阶段执行时会部署所有匹配到的 yml 文件。

覆盖制品

通常当我们对基础设施做部署更新操作时，大多数的变更都是涉及 Docker 镜像或 ConfigMap 中的 flag。因此，CODING CD 针对这些资源类型的变更提供了优秀的支持。

- Docker 镜像
- Kubernetes ConfigMap
- Kubernetes Secret

如果上游阶段的部署流程上下文中存在这些资源对象，CODING CD 将会自动地尝试将它们注入到正在部署的 manifest 文件中。

例如，假设 Docker 镜像仓库类型的触发器触发部署流程执行，并且触发器携带了镜像 gcr.io/my-project/my-image，其 digest 值为 sha256:c81e41ef5e... 。在部署流程中，您配置了 manifest 内容如下的部署阶段。

```
# ... rest of manifest
containers:
- name: my-container
image: gcr.io/my-project/my-image
# rest of manifest ...
```

因为部署流程是由 Docker 镜像内容变更 所触发，所以部署流程编排引擎会将 Docker 镜像制品连同部署阶段中的 manifest 一起派发给 clouddriver 组件处理。最终得到部署的 manifest 内容如下：

```
# ... rest of manifest
containers:
- name: my-container
image: gcr.io/my-project/my-image@:sha256:c81e41ef5e...
# rest of manifest ...
```

为了确保部署阶段获取到正确的制品，您可以强制阶段绑定所有需要的制品，如果绑定失败，阶段将启动失败。以下配置的含义是 Docker 镜像 gcr.io/my-project/my-image 必须被绑定到 manifest，否则阶段将会执行失败：

启用（Manifset）

启用 Kubernetes 对象。

配置选项说明：

- Selector 选择 静态指定目标

字段	是否必填	说明
云账号	是	管理资源对象的云账号
Namespace	是	资源对象所在的命名空间
Selector	是	通过名称静态指定删除的目标资源
Kind	是	资源对象类型
Name	是	资源对象名称（例如 replicaSet 类型的资源 nginx-deployment-

		5dfd77bbf9)

- Selector 选择 动态选择目标

字段	是否必填	说明
云账号	是	管理资源对象的云账号
Namespace	是	资源对象所在的命名空间
Selector	是	通过名称静态指定删除的目标资源
Kind	是	资源对象类型
集群	是	资源对象名称（例如 replicaSet 类型的资源 nginx-deployment-5dfd77bbf9 ）
Target	是	选择规则匹配资源对象

删除（Manifest）

删除通过资源清单创建的 Kubernetes 对象。

配置选项说明：

- Selector 选择 静态指定目标

字段	是否必填	说明
云账号	是	管理资源对象的云账号
Namespace	是	资源对象所在的命名空间
Selector	是	通过名称静态指定删除的目标资源
Kind	是	资源对象类型
Name	是	资源对象名称（例如 replicaSet 类型的资源 nginx-deployment-5dfd77bbf9 ）

- Slector 选择 动态选择目标

字段	是否必填	说明
云账号	是	管理资源对象的云账号
Namespace	是	资源对象所在的命名空间

Selector	是	通过集群和 Target 字段动态选择资源对象
Kind	是	资源对象类型
Cluster	是	资源对象所在的集群
Target	是	选择规则匹配资源对象

- Selector 选择 使用标签匹配目标

字段	是否必填	说明
云账号	是	管理资源对象的云账号
Namespace	是	资源对象所在的命名空间
Selector	是	通过指定标签规则匹配资源对象
Kind	是	资源对象类型
Labels	是	不填写规则表示匹配所有满足指定类型的资源对象

设置选项说明：

字段	是否必填	说明
级联删除	否	勾选后，表示删除此资源对象管理的所有资源对象（例如：Replica Set 管理的所有 Pods），不勾选可能会产生孤儿资源
Grace Period	否	（可选）指定资源对象正式终止的时间，将会覆盖 manifest 设置的时间（如果已设置）

触发器配置

最近更新时间：2021-11-26 14:37:12

本文为您详细介绍在 CODING 持续部署中部署流程的触发器配置。

前提条件

使用 CODING 项目管理的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

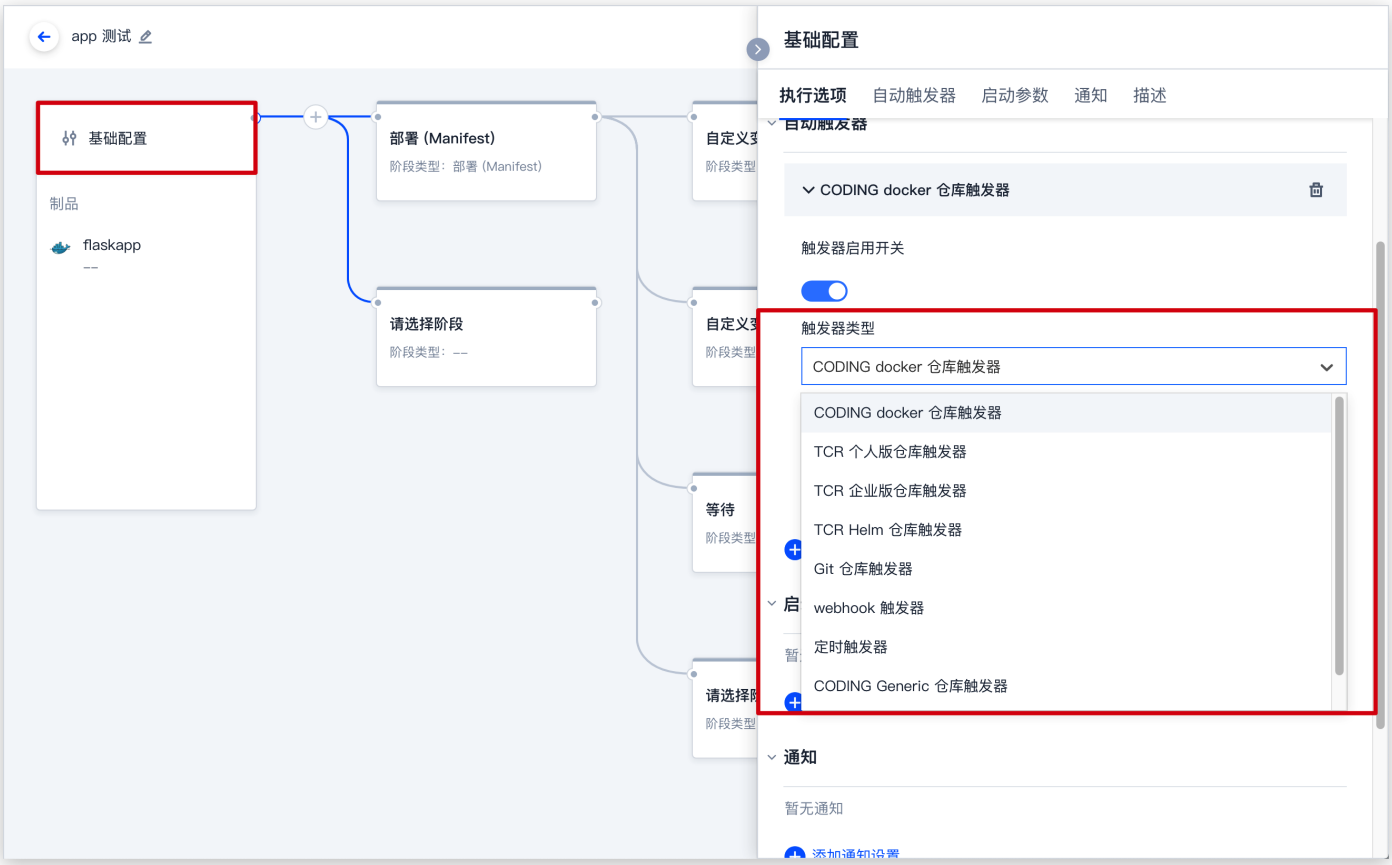
进入项目

1. 登录 CODING 控制台，单击**立即使用**进入 CODING 使用页面。

2. 单击工作台首页左侧的  ，进入持续部署控制台。

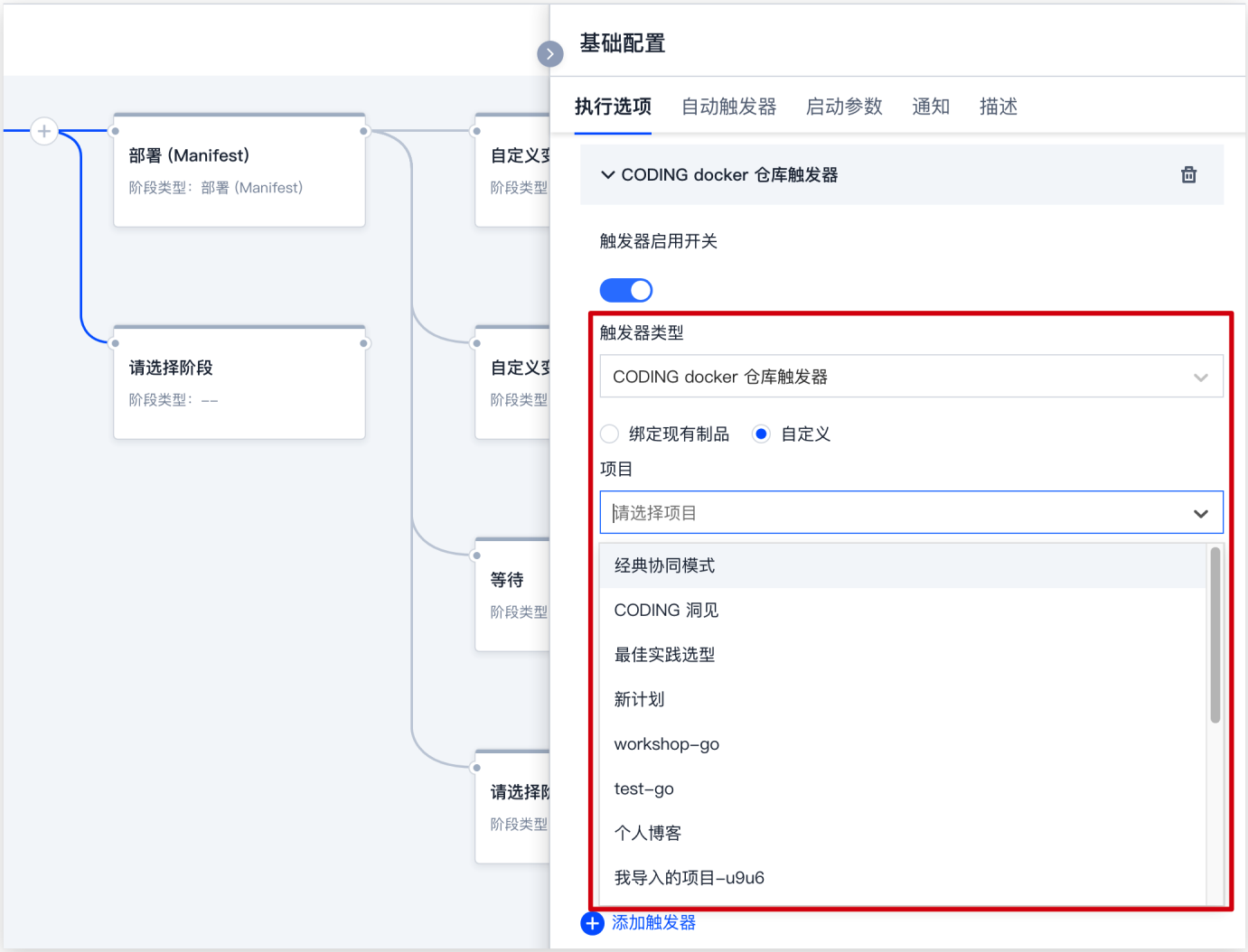
功能介绍

CODING 部署控制台支持多种自动触发条件，使之能够与 CODING 中的流水线相互匹配。目前支持 Docker 仓库触发器、TCR 个人版仓库触发器、TCR 企业版仓库触发器、Git 仓库触发器等触发条件。



Docker 仓库触发

通过配置 Docker 仓库触发器，能够监听制品仓库下的更新情况。若有镜像更新，将自动触发 CD 的部署流程。



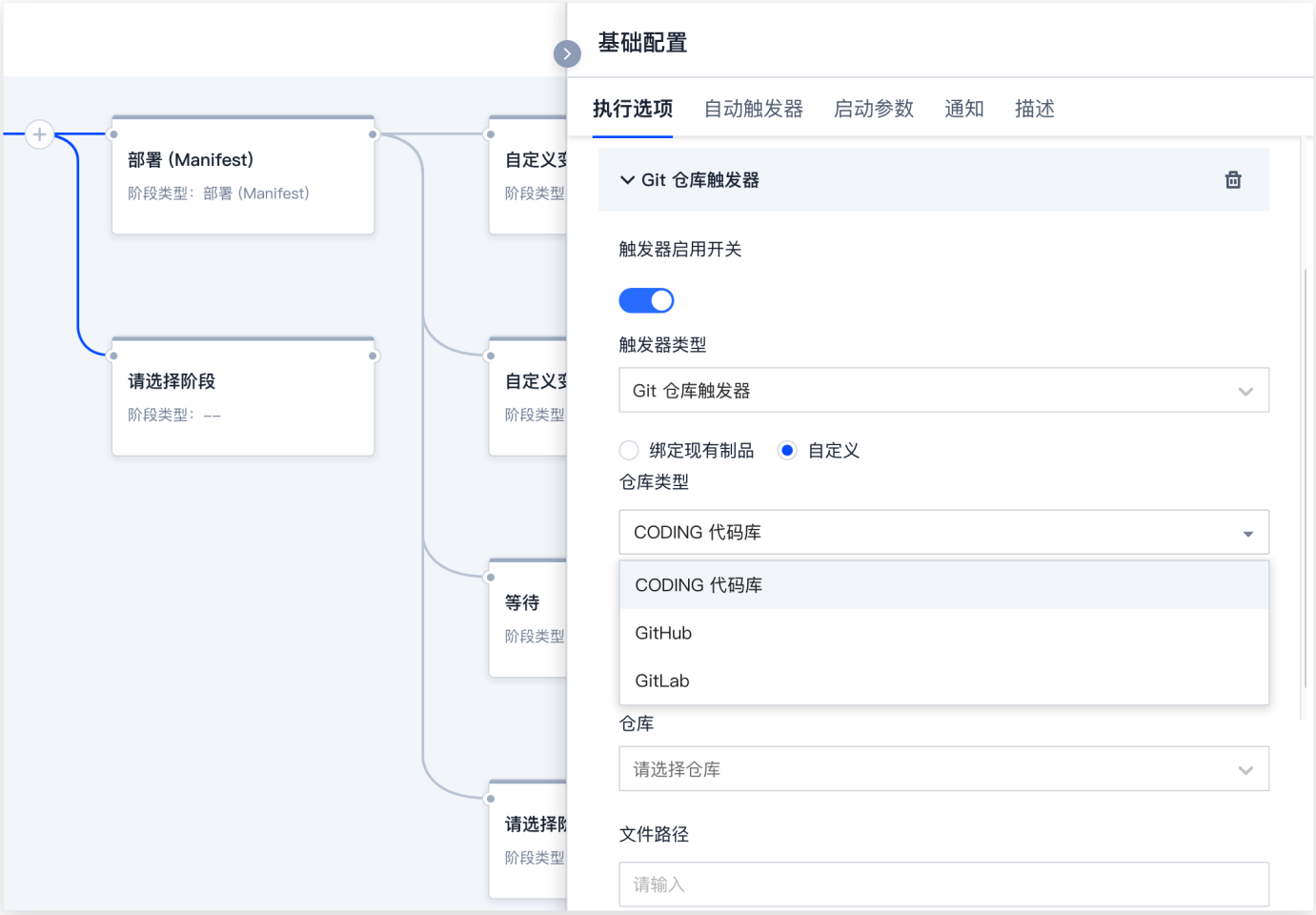
Git 仓库触发

支持 CODING 代码库、GitHub、GitLab 三种类型的 Git 仓库。

字段	说明
仓库类型	支持 CODING 代码库、GitHub、GitLab 三种类型的 git 仓库
项目	列出登录账号加入的所有项目
仓库	列出项目下的所有代码仓库
分支或标签规则	支持正则表达式，留空或.*表示不对分支或标签做限制

CODING 代码库

配置 cd-demo 项目下的 cd-demo 代码仓库作为触发器，分支或标签规则 release.* 表示所有以 release 开头命名的分支或 tag 才会触发部署流程执行。

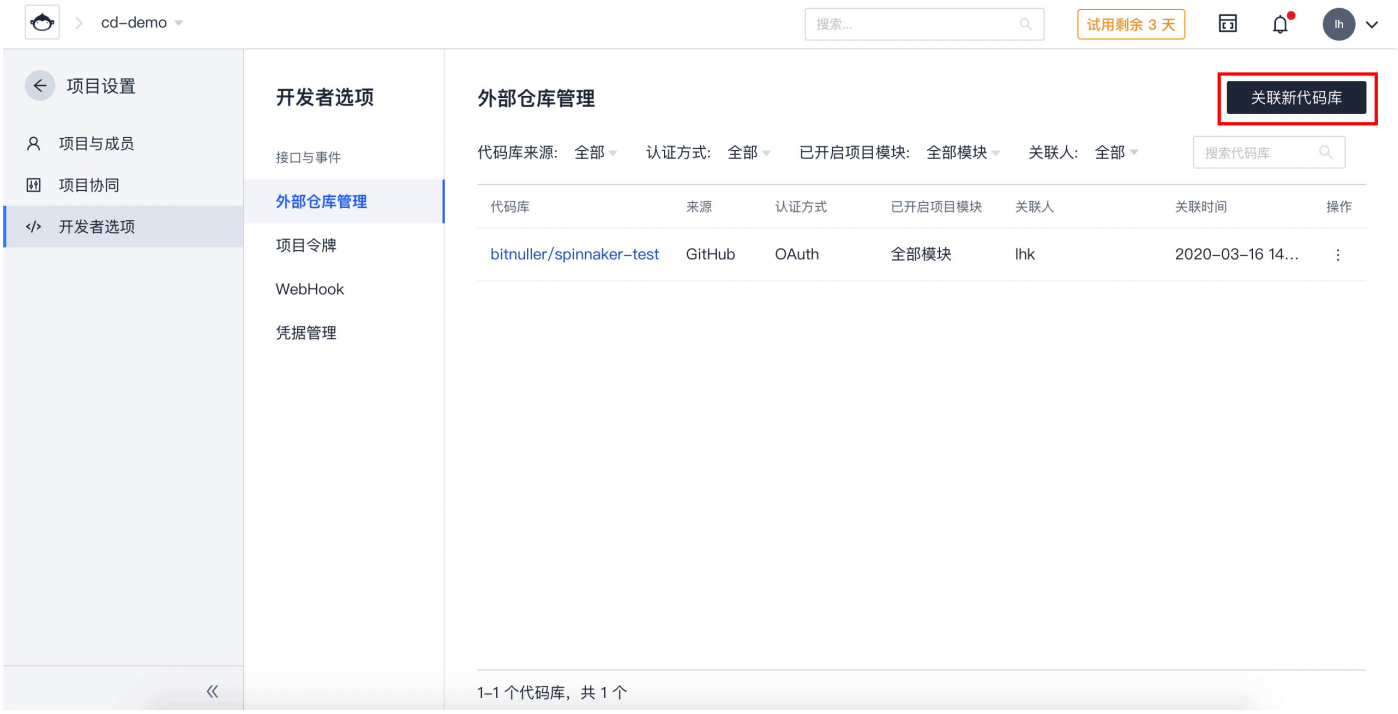


Github

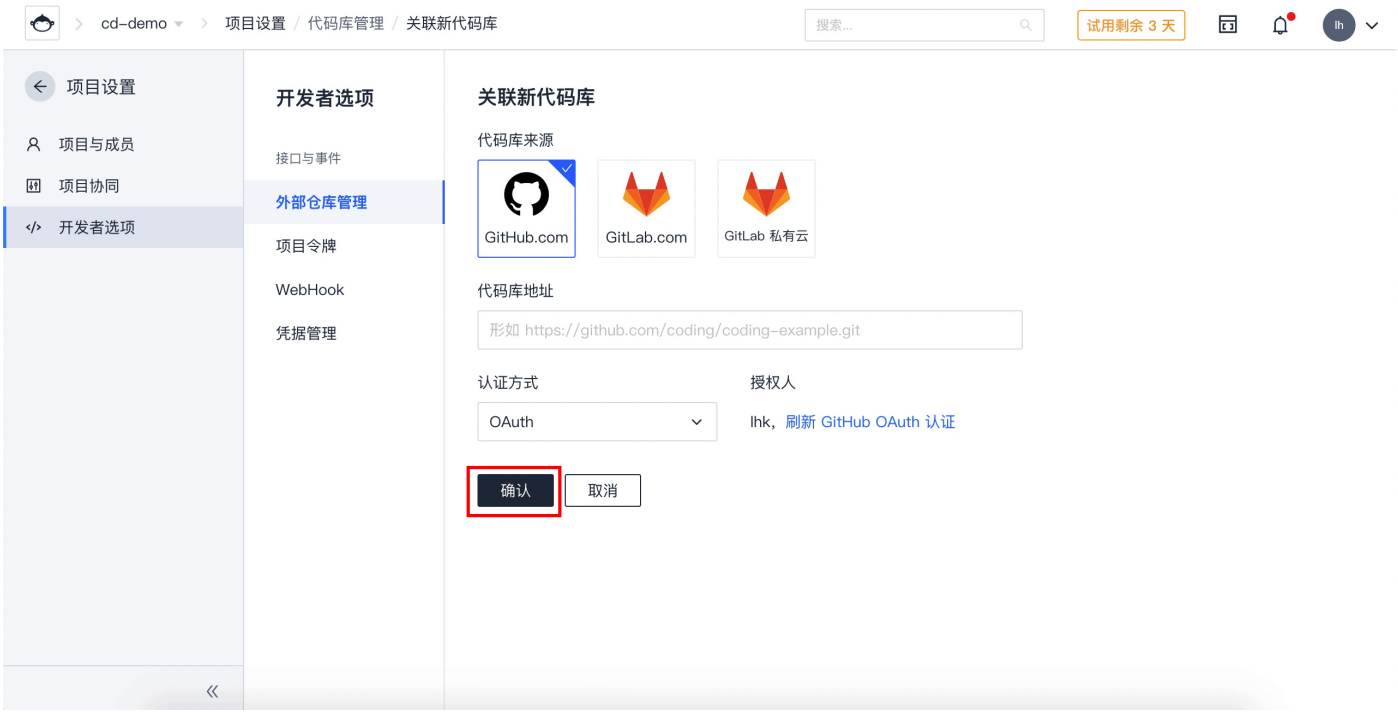
Github 代码库的支持需要提前在项目设置中关联代码库，具体步骤如下：

1. 进入项目概览页，单击**项目设置**。

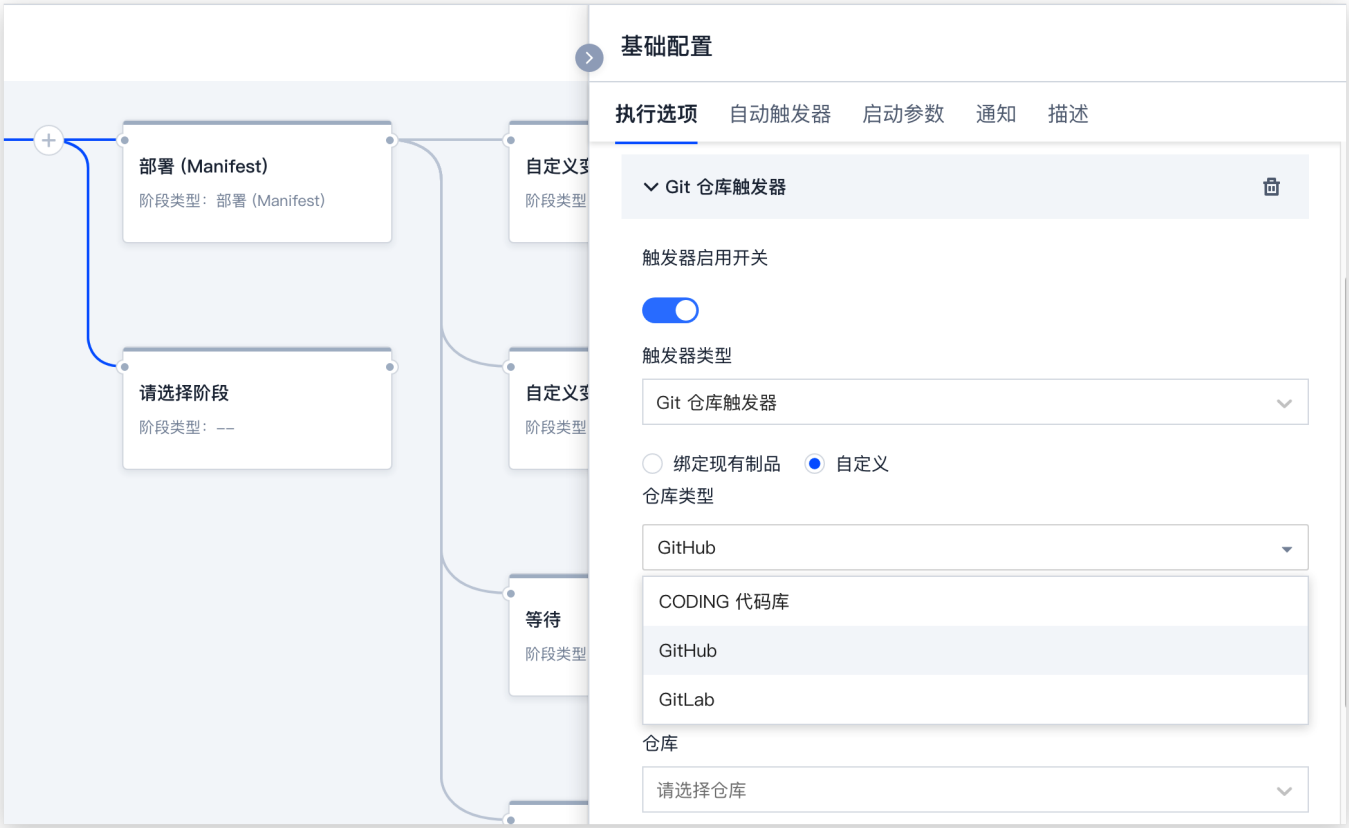
2. 在**开发者选项 > 外部仓库管理**页面的右上角，单击**关联新代码库**。



3. 进入**关联新代码库**页面，设置代码库地址，单击**确认**。



4. 关联完成后返回**基础配置 > 执行选项**，选择**GitHub仓库类型**。



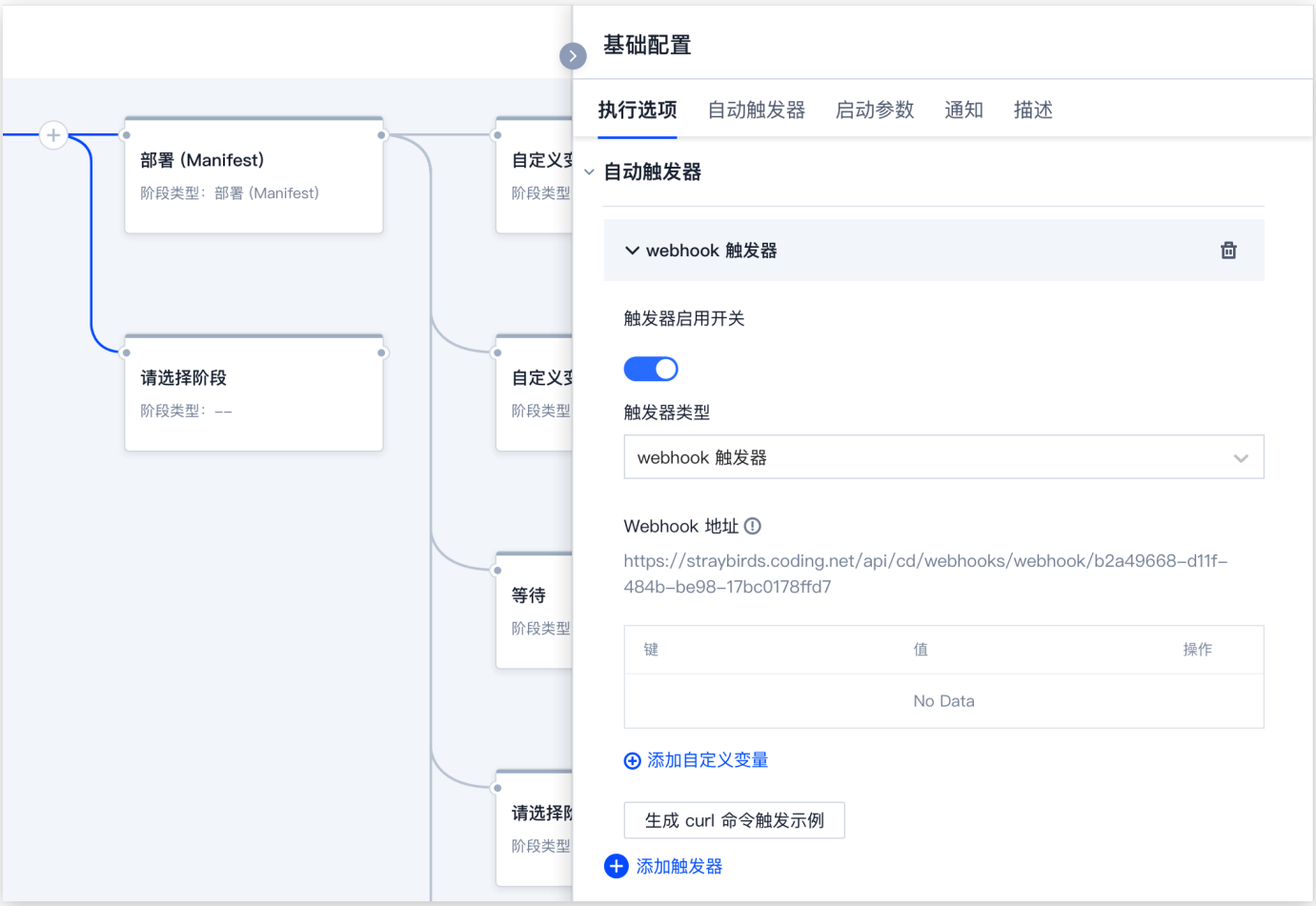
GitLab

需要提前关联 GitLab 账号后（关联步骤请参见 Github），在**基础配置 > 执行选项**中选择**GitLab仓库类型**。

Webhook 触发

选择 Webhook 触发，将生成全局唯一的 URL 触发地址，Payload Constraints 定义 Payload 请求内容必须提供的参数，支持正则表达式，留空或 .* 表示不对 Key 的 Value 值做限制。

Payload Constraints：如果需要使用特定内容的 payload 触发 Webhook，可以在 Payload Constraints 一栏添加 key/value 键值对，当部署流程收到 Webhook 请求时，将会 payload 内容进行校验，value 支持正则表达式。



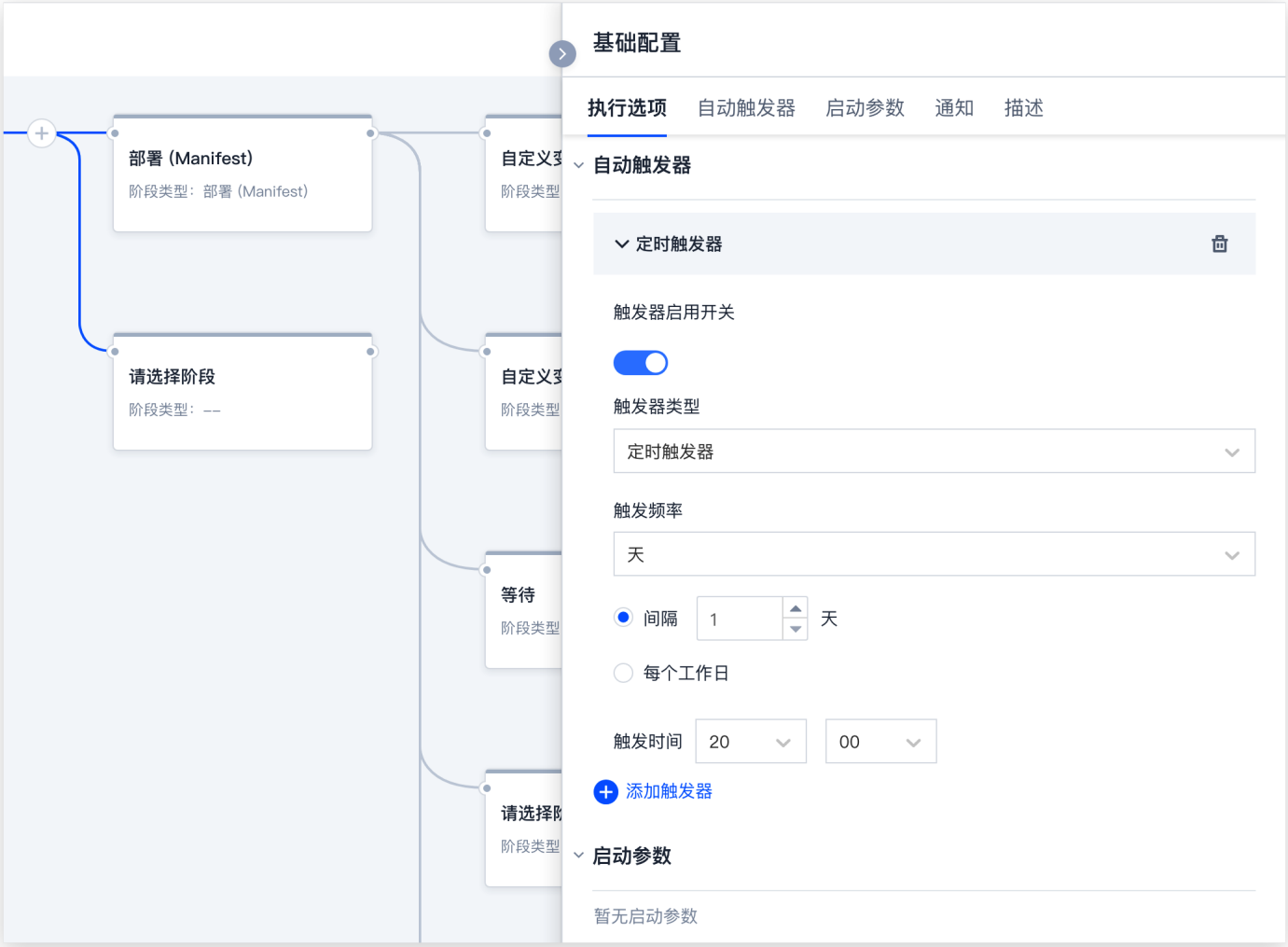
使用场景举例：部署流程 Webhook 地址对公网开放，但只有提供认证凭据才能触发部署流程执行。

如下 Payload 请求会成功触发部署流程执行：

```
curl --location --request POST 'http://codingcorp.coding.com/api/cd/webhooks/webhook/ba2e9f00-6445-11ea-88b5-a9bc004f5e0f' \
--header 'Content-Type: application/json' \
--data-raw '{"secret": "faiM4&KqJTTuEy8J"}'
```

定时触发

例如每天晚上 8 点触发部署流程：


部署 (Manifest)
阶段类型：部署 (Manifest)
请选择阶段
阶段类型：--
等待
阶段类型
基础配置
执行选项
自动触发器
启动参数
通知
描述
自动触发器
定时触发器
触发器启用开关
触发器类型
定时触发器
触发频率
天
间隔
1
天
每个工作日
触发时间
20
00
添加触发器
启动参数
暂无启动参数

部署方式
自动发布 Docker 制品时触发

最近更新时间：2021-08-26 11:00:32

CODING 持续部署的一大优势在于能够便捷的集成上下游产品为工作流，下文将演示如何通过三个步骤实现 **持续集成任务推送制品 > 制品仓库镜像更新 > 触发部署流程**这一基础的自动化流水线配置。

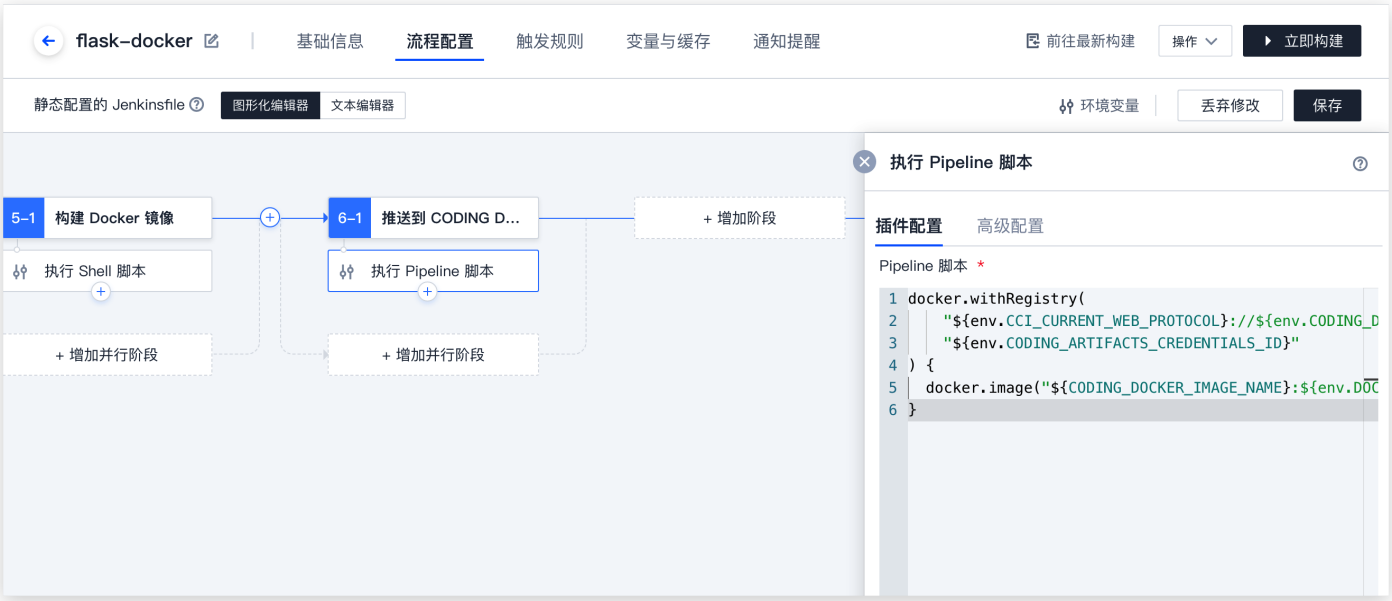
步骤1：应用与项目关联

部署流程控制台中的**应用**需提前与项目相关联。前往部署控制台，单击应用中的**关联项目**按钮，选择持续集成配置所在的项目并进行关联。

步骤2：配置持续集成

此步骤使用持续集成将制品推送至制品仓库。您可以通过持续集成计划模板创建，或直接编写 Jenkinsfile 手动增加此阶段。

在持续集成流程中手动增加此阶段：



Jenkinsfile 参考

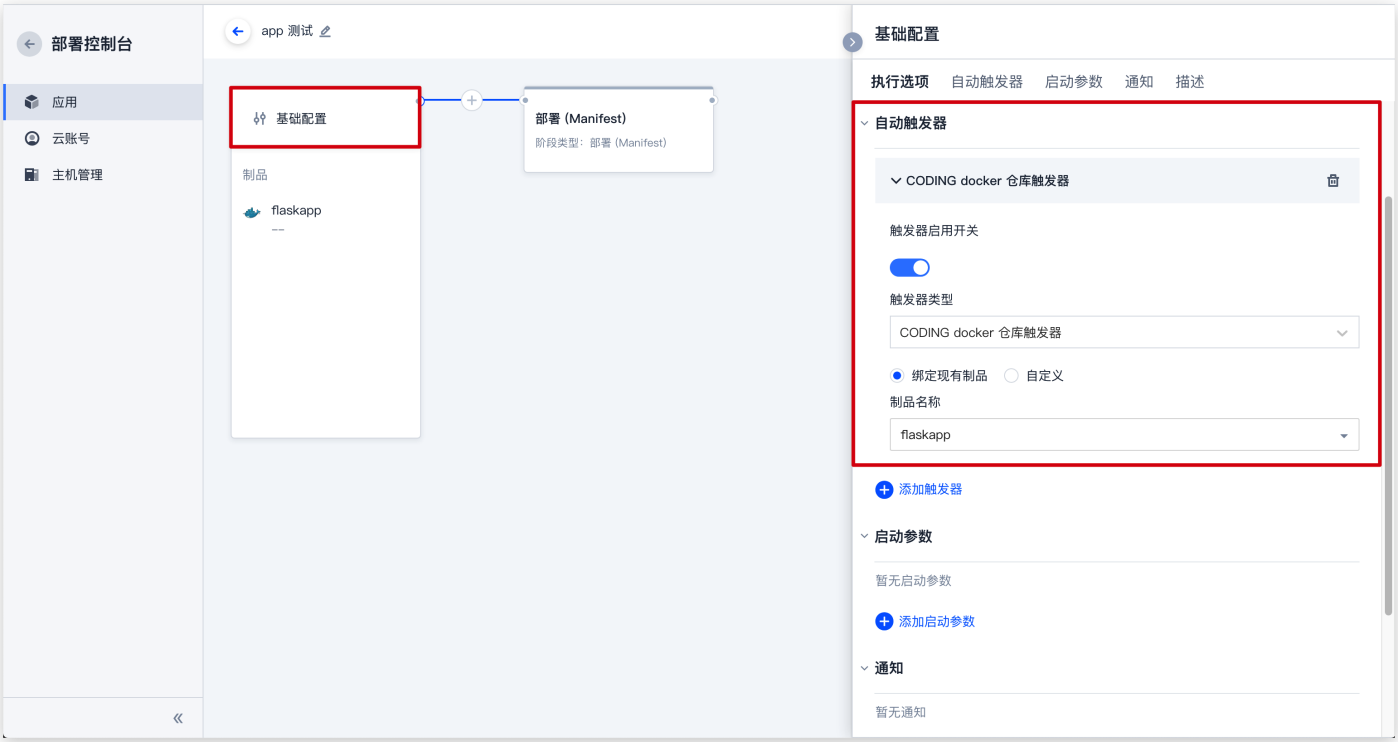
```
stage('部署到远端 Kubernetes 集群') {
steps {
cdDeploy([
deployType: 'PATCH_IMAGE',
application: "${CCI_CURRENT_TEAM}",
pipelineName: "${PROJECT_NAME}-${CCI_JOB_NAME}-${CD_CREDENTIAL_INDEX}",
image: "${CODING_DOCKER_REG_HOST}/${CODING_DOCKER_IMAGE_NAME}:${DOCKER_IM
```

```
AGE_VERSION}",
cloudAccountName: "${CD_ACCOUNT_NAME}",
namespace: "${CD_NAMESPACE_NAME}",
manifestType: "${CD_MANIFEST_TYPE}",
manifestName: "${CD_MANIFEST_NAME}",
containerName: "${CD_CONTAINER_NAME}",
credentialId: "${CD_CREDENTIAL_ID}",
personalAccessToken: "${CD_PERSONAL_ACCESS_TOKEN}",
])
}
}
```

步骤3：根据制品镜像版本触发

前往持续部署中的应用部署流程，单击基础配置中的触发器启用开关。此处选择通过 CODING Docker 制品更新触发，将监听关联项目中制品版本号。若持续集成将制品推送至制品仓库时，将自定触发部署流程；选择自定义能够监听其他项目的制品仓库更新情况。

除了通过 CODING Docker 制品更新触发，您还可以通过 Git 仓库或定时器触发此部署流程。



在构建计划中添加部署阶段

最近更新时间：2021-08-26 11:01:13

在持续集成中触发部署时请提前前往部署控制台，将应用与项目关联。

本文给出了两种设置方法，您可以按照需求有选择性阅读。

- **直接使用构建计划模板**
- **在已有构建计划中添加部署阶段**

使用构建计划模板

> **注意：**
>
> **请在部署控制台中关联涉及到相关集群的云账号，详情请参见 云账号。**

单击项目内左侧产品栏持续集成，右上角创建构建计划，选择部署分类下的推送到 Kubernetes模板。

按照模板提示选择相应的制品仓库、远端集群地址等信息，完成后勾选创建后触发构建。

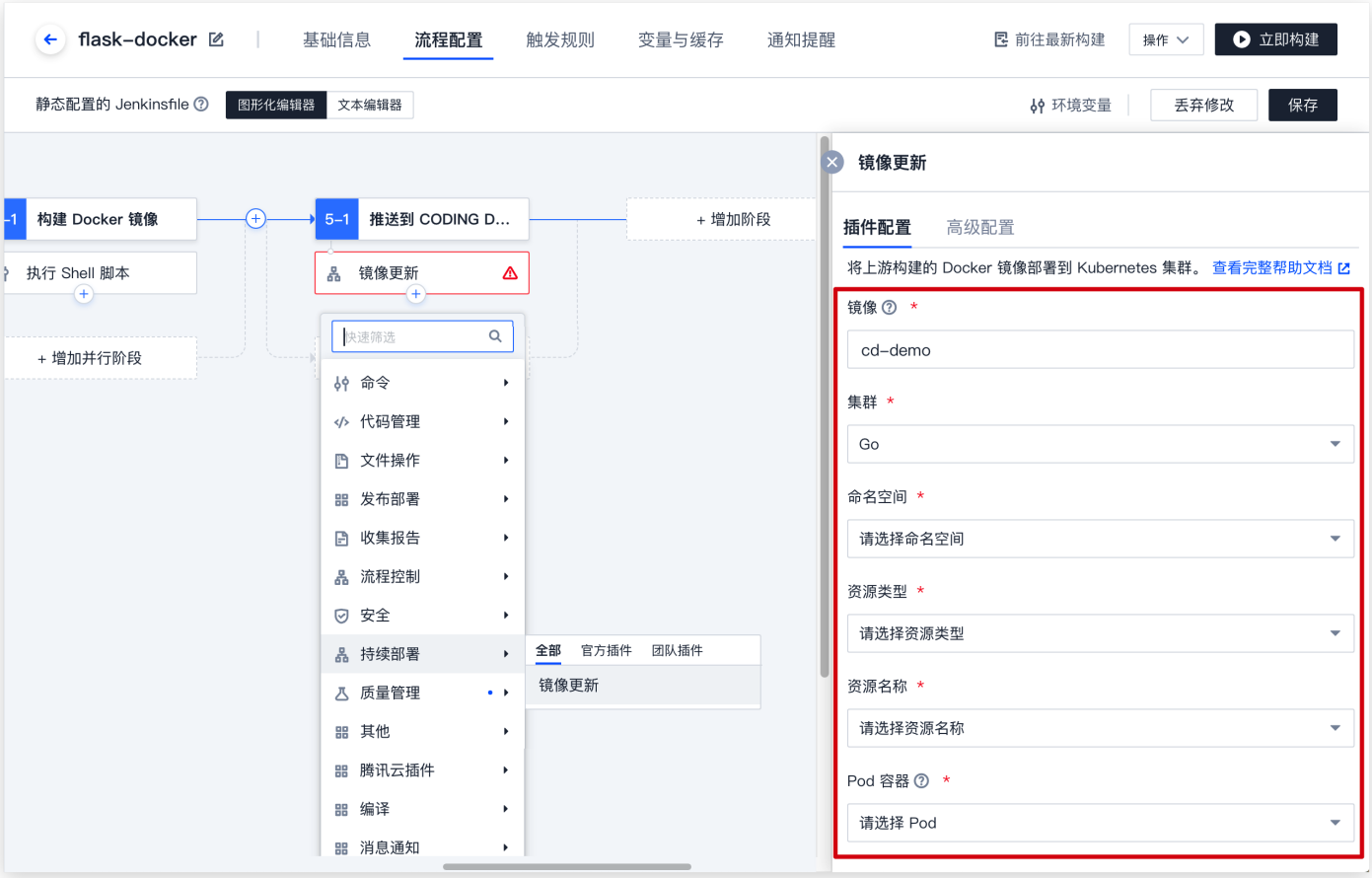
设置完成后，运行持续构建计划即可完成自动发布。

添加部署阶段

在此方法中您可以在**构建计划 > 流程配置**中使用编辑器或填写命令添加部署阶段。

图形化编辑器

在已有构建计划设置中添加**部署**阶段，填写镜像地址、集群与命名空间等关键信息。



Jenkinsfile 参考

```
stage('推送到 CODING Docker 制品库') {
steps {
script {
docker.withRegistry(
"${env.CCI_CURRENT_WEB_PROTOCOL}://${env.CODING_DOCKER_REG_HOST}",
"${env.CODING_ARTIFACTS_CREDENTIALS_ID}"
) }
docker.image("${CODING_DOCKER_IMAGE_NAME}:${env.DOCKER_IMAGE_VERSION}").push
()

}

}
```

手动提交发布单

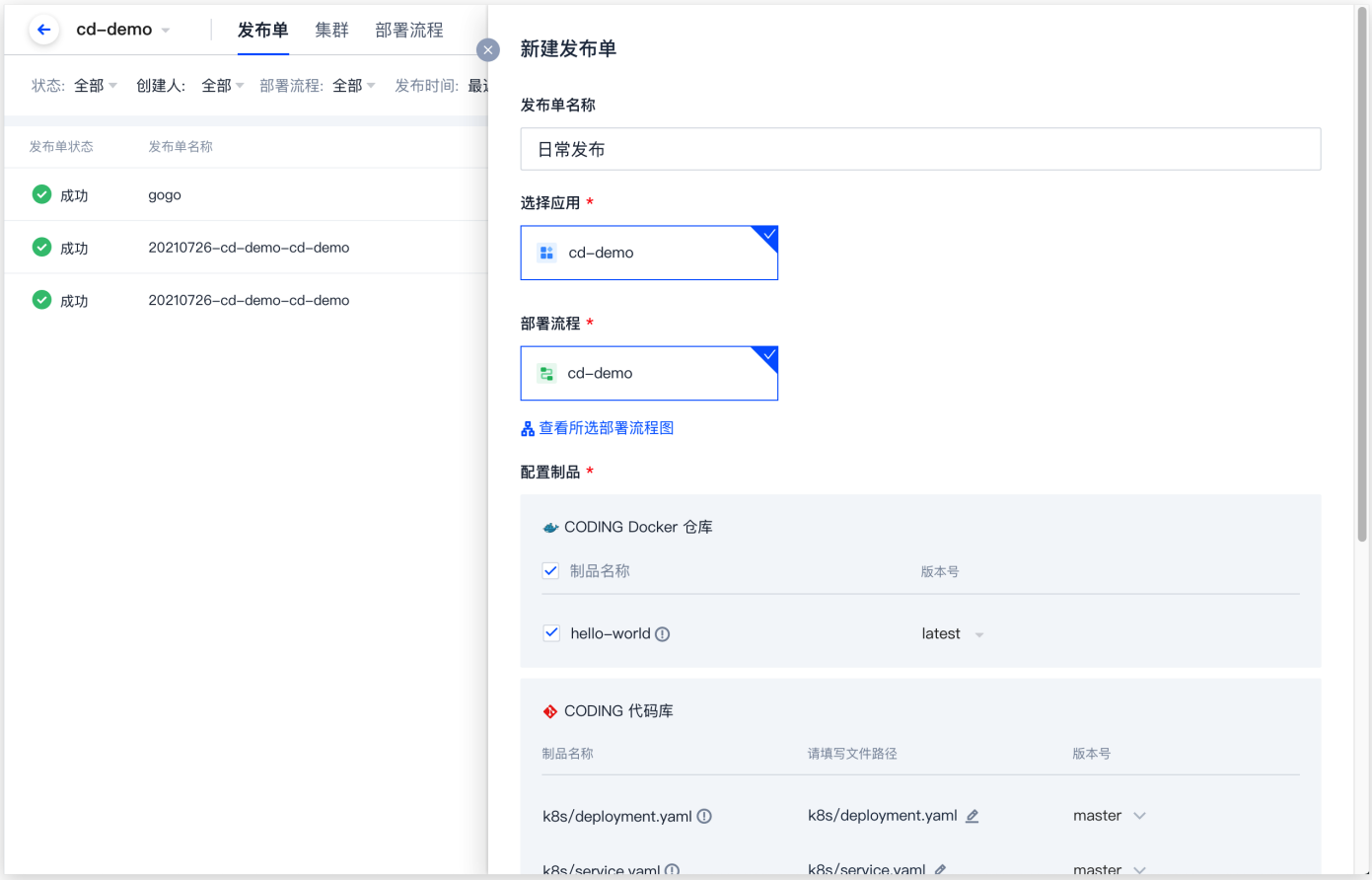
最近更新时间：2021-08-26 11:01:35

新建发布单

我们推荐赋予开发用户组访问与持续部署管理权限。详情请参见 权限控制 。

设置后，开发具备提交发布单的权限，不具备前往部署控制台修改部署配置的权限。运维组用户还可以在应用的部署流程中添加人工确认步骤，确保通过发布单发布时是经二次确认的，通过权限控制把控发布质量。

单击新建发布单，可以运行已有应用及部署流程。

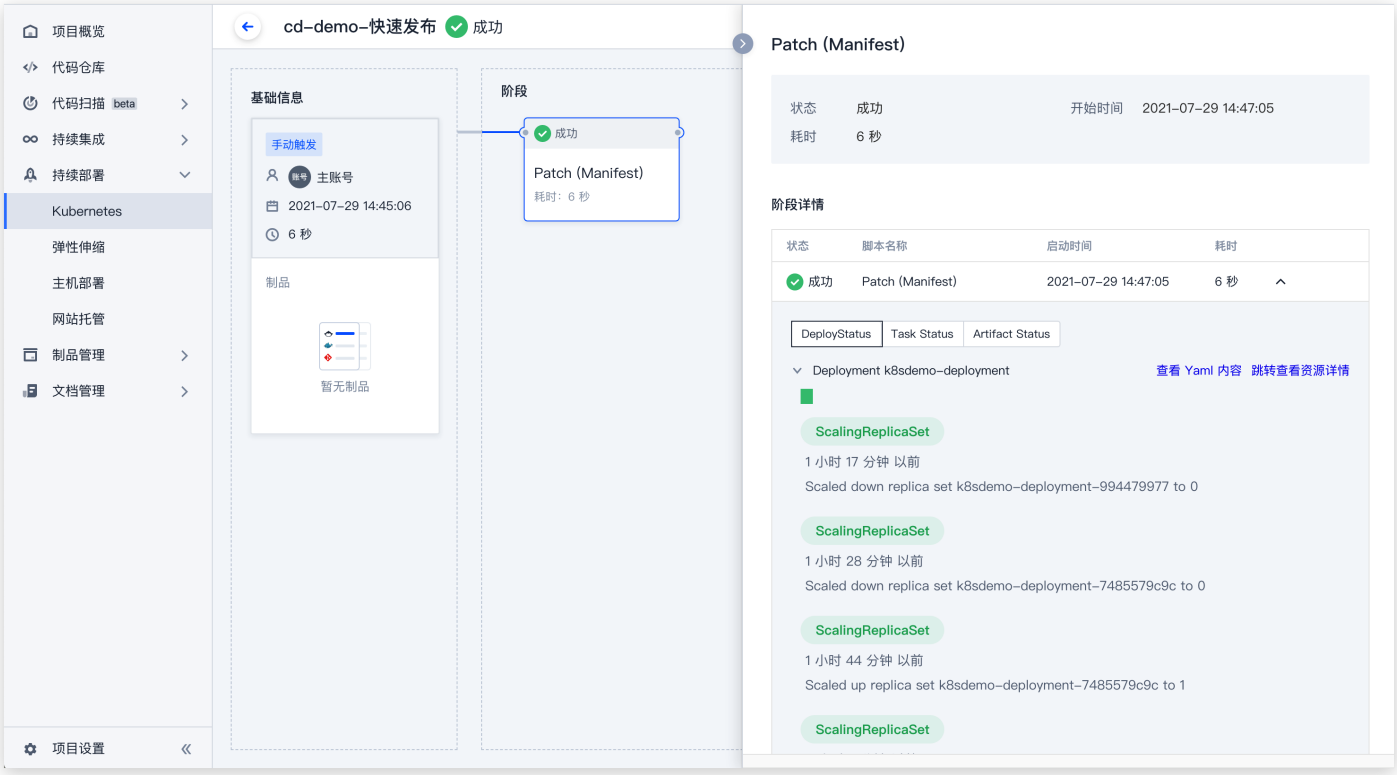


快速发布

若不希望在团队设置复杂的权限限制，而直接希望体验持续部署功能，可以使用**快速发布**功能。无需在控制台中配置部署流程即可将镜像发布至集群中，适用于更加灵活复杂的部署流程的场景，例如临时镜像变更等突发场景，无需快速将制品发布至集群中。

成员所在用户组需具备部署管理权限，所发布的制品的权限范围需设置为公开状态，从而能够被集群访问。

发布完成后可以在持续部署中查看发布详情。



制品
部署流程中的制品

最近更新时间：2023-06-15 15:59:20

本文为您详细介绍 CODING 持续部署中部署流程的制品。

前提条件

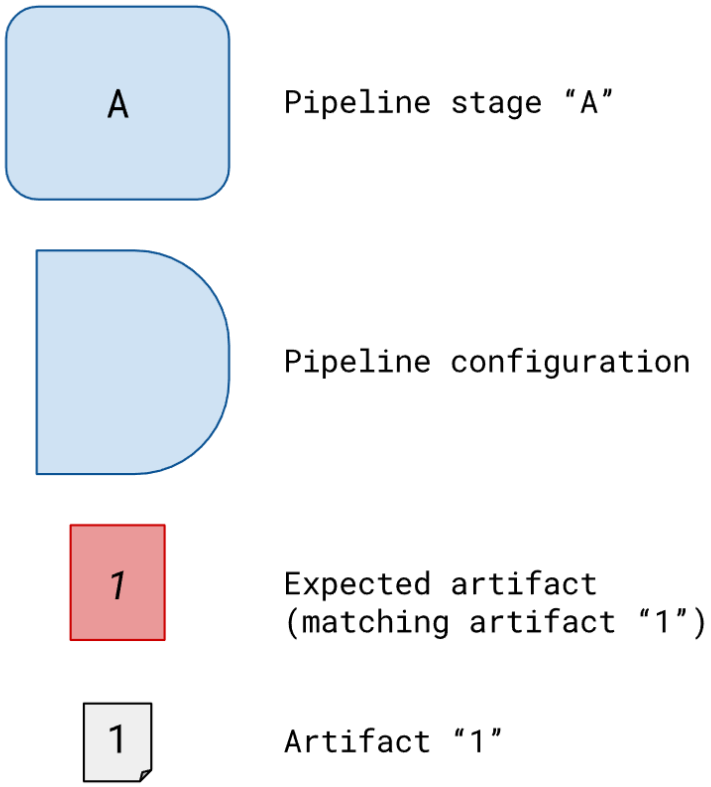
使用 CODING 持续部署的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

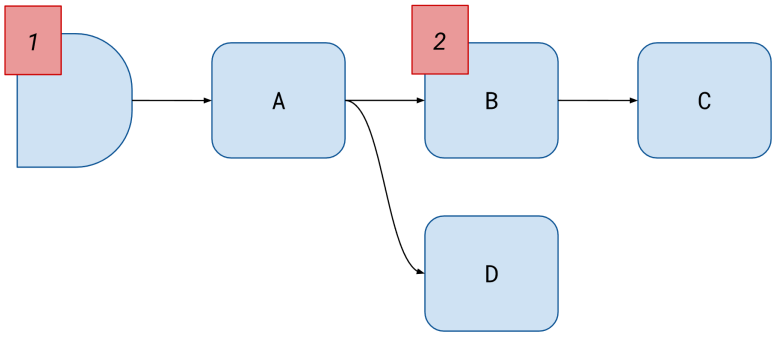
1. 登录 CODING 控制台，单击团队域名进入 CODING 使用页面。
2. 单击工作台首页左侧的 ，进入持续部署控制台。

图示

我们用一个 demo 部署流程帮助您理解制品和预期制品（expected artifacts）的工作原理。
以下是使用到的元素：

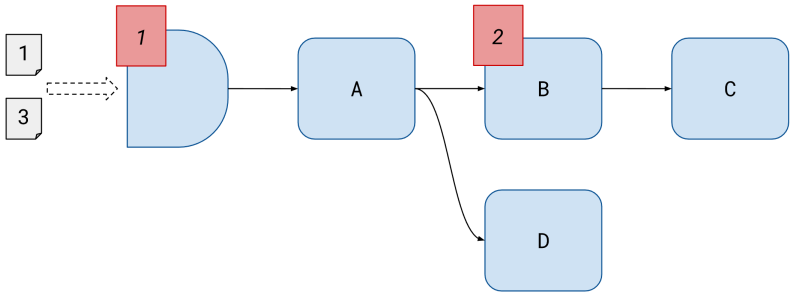


假设配置如下的部署流程：

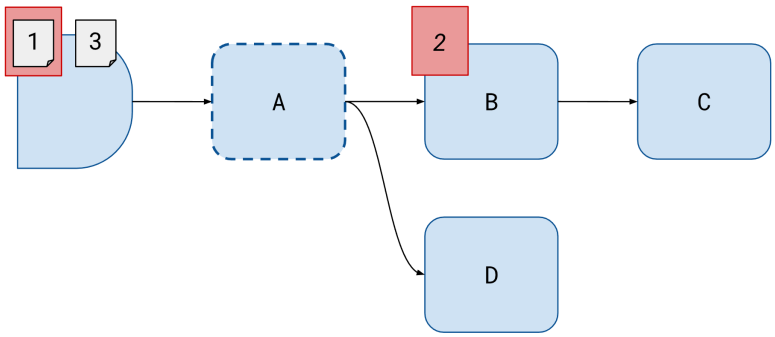


> 说明：
> 这个部署流程声明了在启动时需要获取到 制品 1（可以是一个 Docker 镜像），对应配置在部署流程配置阶段完成。另外还声明了在部署流程的 B 阶段（可以是 从执行环境中查找制品 阶段）需要获取 制品 2 。

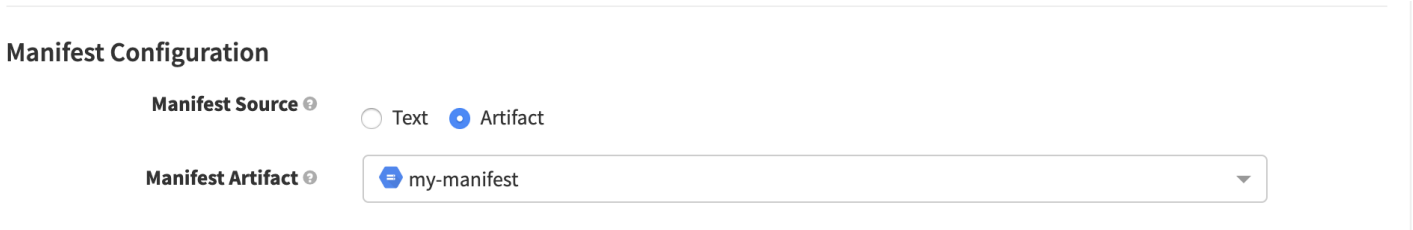
部署流程被触发器（假设由 Webhook 触发）触发执行，触发器提供了两个制品：



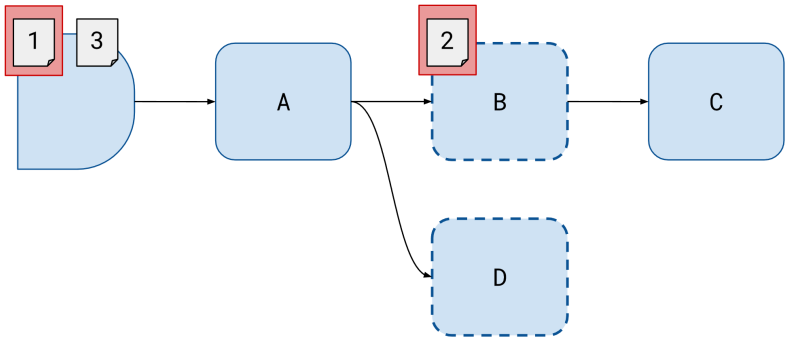
虽然只有 制品 1 被匹配成功被绑定，但是由于触发器携带了两个制品的信息，所以在下游阶段两个制品都可以被引用。预期制品(expected artifact) 的好处是下游阶段可以直接引用，而如果下游阶段引用了 制品3（没有被 expected artifacts 匹配成功的制品），那么在阶段运行时需要判断这个制品是否存在。



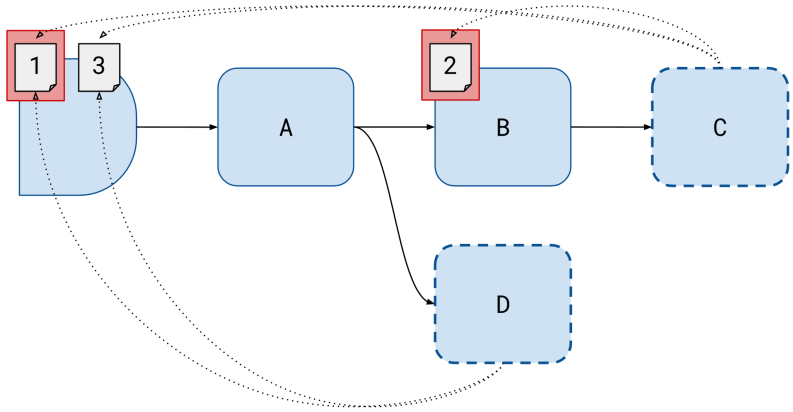
需要强调的是 制品 1 在部署流程启动时就被绑定了，如果我们像图示在下游阶段（如 Deploy Manifest）中引用制品，这个制品在部署流程开始执行时就被绑定到阶段，而不是阶段执行时才绑定。



当 B 阶段执行时，需要绑定 制品2 。假设是 从执行环境中查找制品 阶段，B 阶段将从其他部署流程执行环境中查找制品并绑定。



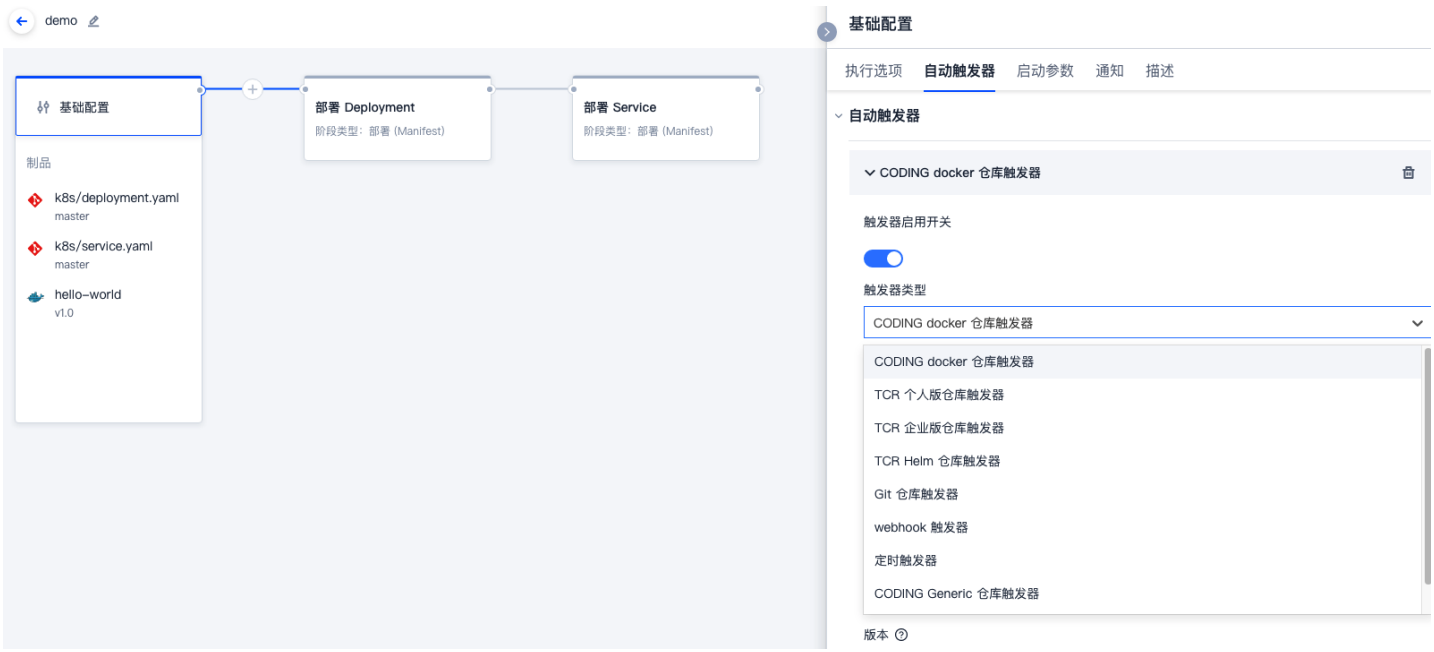
如果阶段 C 或 D 需要引用上游的制品，它们所能引用的制品源可能不同，因为阶段 C 和 D 各自的上游阶段也不相同。如下图，阶段 D 无法引用 制品2 。



部署流程执行上下文的制品可能来自于外部触发器（例如上传到 CODING 制品库的 Docker 镜像），也可以从其他阶段获取，制品可以被下游的阶段所使用。
CODING 部署控制台通过 Expected artifacts 特性将其他部署流程的制品、阶段的产物（例如 Bake 阶段）与特定阶段相绑定。

触发器

您可以通过手动触发部署流程，也可以通过设置触发器的方式自动触发。例如在**部署流程**的**基础配置**中使用 **Docker 触发器**就能够监听特定类型的制品上传，以此作为触发条件。



如果部署流程启动时无法匹配**制品匹配规则**中所定义的制品，将会按照优先级排序以下两项可获取制品的次选方案：

- 使用上一次执行的触发器
- 使用默认制品

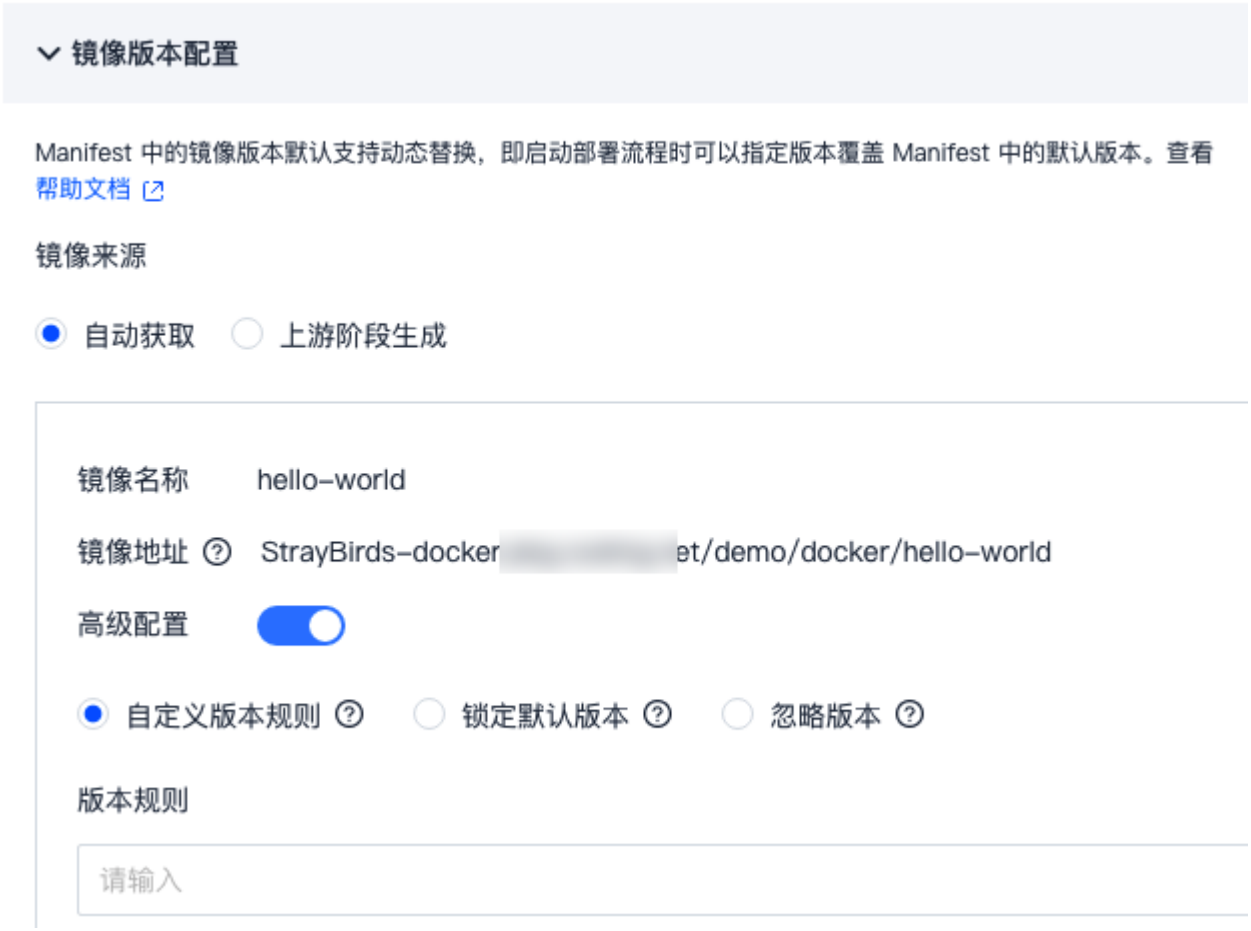
触发器 Payloads 携带制品信息

在触发器的 payload 内容中，名为 artifacts 的键（list 类型）提供了制品列表。当部署流程被触发时，artifacts 的值将会注入到部署流程执行上下文。如下以 payload 携带 CODING docker 制品为例：

```
{
"artifacts": [
{
"type": "coding_docker/image",
"name": "lhkprod-docker.pkg.coding.net/cd-demo/release/nginx",
"reference": "lhkprod-docker.pkg.coding.net/cd-demo/release/nginx:latest"
}
]
}
```

在部署流程执行环境中查找制品

部署阶段中支持自动获取镜像版本，Manifest 中的镜像版本默认支持动态替换，即启动部署流程时可以指定版本覆盖 Manifest 中的默认版本。



常见的使用场景是当某个镜像通过部署流程成功地部署在预发布环境后，使用正式环境对应的部署流程查找此制品用以部署到正式环境。

在部署流程之间传递制品

制品可以在部署流程之间传递，假设 A 部署流程触发了 B 部署流程执行，那么 B 部署流程可以获取到 A 部署流程中任何可用的制品；包括 A 部署流程触发器携带的制品以及 A 部署流程生成的制品。
以下两个案例说明了制品是如何在部署流程之间传递的：

- 在 B 部署流程的配置阶段，将 A 部署流程配置为自动触发器。当 A 部署流程执行完毕时，B 部署流程将会被触发执行，并且 B 部署流程将拥有访问 A 部署流程所有制品的权限。
- 子部署流程作为父部署流程的阶段，在 A 部署流程（父部署流程）中添加类型为部署流程的阶段并指向 B 部署流程。B 部署流程将拥有上游制品的权限（上游制品指 B 部署流程被触发之前 A 部署流程的所有制品）

在阶段中生成制品

阶段中生成的制品同样可以作为触发下一阶段的触发条件。在部署阶段中可以选择将镜像来源设置为**上游阶段生成**。

阶段生成的制品有如下两个使用场景：

- 绑定制品注入到阶段上下文
  在阶段（如 Deploy（Manifest））中生成的制品可供下游阶段使用。
- 人为地将制品注入到阶段上下文
  如果阶段本身不生成制品（如 Run Job 阶段），您可以配置默认制品，每次阶段执行时将默认制品注入到部署流程。需要注意的是，如果指定的镜像不匹配，阶段将会绑定部署流程中的任意制品。

Kubernetes 场景下的制品

最近更新时间：2023-05-24 10:29:36

本文为您详细介绍 CODING 持续部署中 Kubernetes 场景下的制品。

前提条件

使用 CODING 持续部署的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击团队域名进入 CODING 使用页面。
2. 单击工作台首页左侧的  ，进入持续部署控制台。

功能介绍

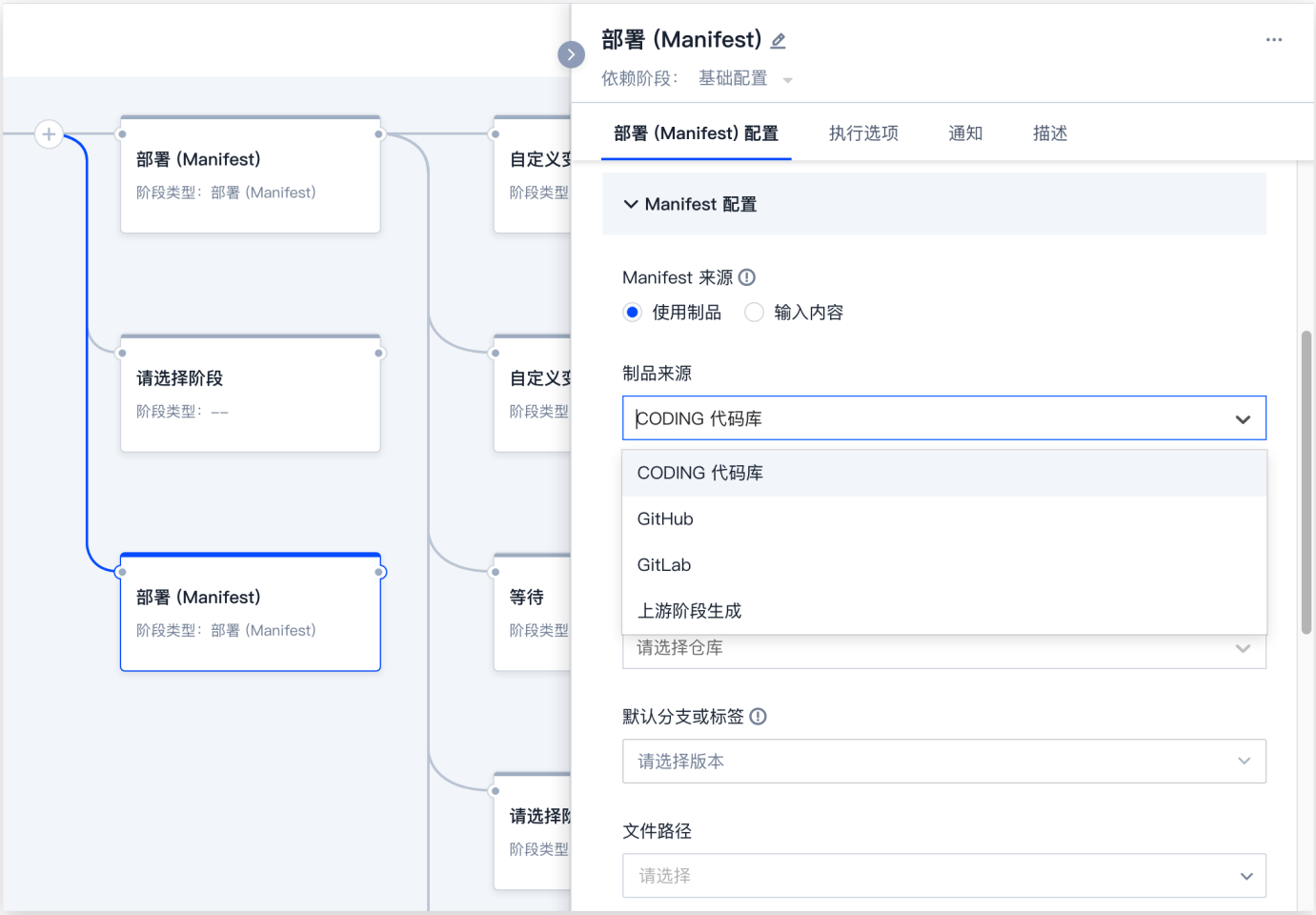
制品在 Kubernetes 资源对象的部署中扮演重要的角色，您在 manifest 文件中引用的如 Docker 镜像或 ConfigMap 都可以通过制品来进行绑定部署。

将 Manifest 作为制品

有两种方法可以部署 manifest：

- manifest 以静态的方式（text 格式）提供给部署流程
- manifest 作为制品提供给部署流程

在 Manifest 来源处可以选择本项目或其他项目中的代码仓库。



在 manifest 中绑定制品

通常来说，制品表示用户在流程部署中需要更新的资源。诸如 Docker 镜像和 ConfigMaps 这类通过 manifest 更新的资源类型，CODING 部署控制台（Spinnaker）提供了一种简单的方式将其注入到 manifest 文件中。

在 manifest 中绑定制品的过程中，CODING 部署控制台使用了一个便捷的规则实现。即当 manifest 中资源对象的 type 和 value 与制品的 type 和 name 分别匹配时，manifest 中的 value 将会被制品的 reference 替换。

manifest 中资源对象的 type 具体来说就是 spec.template.spec.containers.*.image 中所引用的 Docker 镜像，因此将匹配 docker/image 类型的制品。

spec.template.spec.volumes.*.configMap.name 引用 ConfigMap，因此将匹配 kubernetes/configMap 类型的制品。

它们的实现过程原理如下：

首先在部署流程中定义了需要部署的 manifest 内容：

```
apiVersion: apps/v1
kind: Deployment
metadata:
labels:
app: nginx
name: nginx-deployment
spec:
replicas: 3
selector:
matchLabels:
app: nginx
template:
metadata:
labels:
app: nginx
spec:
containers:
- image: lhkprod-docker.pkg.coding.net/cd-demo/release/nginx # possible artifact
name: nginx
ports:
- containerPort: 80
volumeMounts:
- mountPath: /opt/config
name: my-config-map
volumes:
- configMap:
name: configmap # possible artifact
name: my-config-map
```

当部署阶段执行时，如果执行上下文中有如下的制品（制品可能来自触发器或之前的部署过程），代码如下：

```
[
{
"type": "docker/image",
"name": "lhkprod-docker.pkg.coding.net/cd-demo/release/nginx",
"reference": "lhkprod-docker.pkg.coding.net/cd-demo/release/nginx@sha256:0cce25b9a55"
```

```
},
{
"type": "kubernetes/configMap",
"name": "configmap",
"version": "v001",
"location": "default",
"reference": "configmap-v001"
}
]
```

则 ConfigMap 和 Docker 镜像将会被上下文中的制品替换，最终被部署到集群的 manifest 内容为：

```
apiVersion: apps/v1
kind: Deployment
metadata:
labels:
app: nginx
name: nginx-deployment
spec:
replicas: 3
selector:
matchLabels:
app: nginx
template:
metadata:
labels:
app: nginx
spec:
containers:
- image: lhkprod-docker.pkg.coding.net/cd-demo/release/nginx@sha256:0cce25b9a55 # bound
by spinnaker
name: nginx
ports:
- containerPort: 80
volumeMounts:
- mountPath: /opt/config
name: my-config-map
```

```
volumes:
- configMap:
name: configmap-v001 # bound by spinnaker
name: my-config-map
```

阶段类型详情
人工确认

最近更新时间：2021-08-26 11:02:09

本文为您详细介绍 CODING 持续部署里阶段类型详情中的人工确认。

前提条件

使用 CODING 持续部署的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击团队域名进入 CODING 使用页面。
2. 单击工作台首页左侧的  ，进入持续部署控制台。

人工确认是每次应用发布前的审核机制，通过人工确认您可以实现：

1. 应用预发布至生产环境时，由确认人（一般是运维、团队所有者或管理员）审核后决定是否执行发布。
2. 选择分支流程。假设部署流程中有两条分支流程，一条用于发布测试环境，另一条用于预发布环境。在人工确认阶段中可以根据**确认人**的选择并配合**预置条件检查**在相应的分支流程执行发布策略。

配置入口

在**持续部署控制台** > **应用** > **添加阶段**中选择人工确认阶段并填写相关配置。

人工确认 配置参数说明：

参数	必填	说明

提示信息	否	提示信息会在确认框内显示，帮助用户确认。
确认人	否	选择确认人。若没有配置确认人，则具有部署设置权限的人员都可以进行确认操作。
确认选项	否	配置了确认选项后，确认人需要选择任一确认选项才能继续执行或终止阶段，下游阶段可使用表达式 ${#judgment('本阶段名称')} 引用选项值。
自定义参数	否	确认人通过自定义参数输入字符串，下游阶段可使用表达式 ${#stage('本阶段名称')["context"]["customParams"]["参数名"]} 引用参数值。

配置通知人

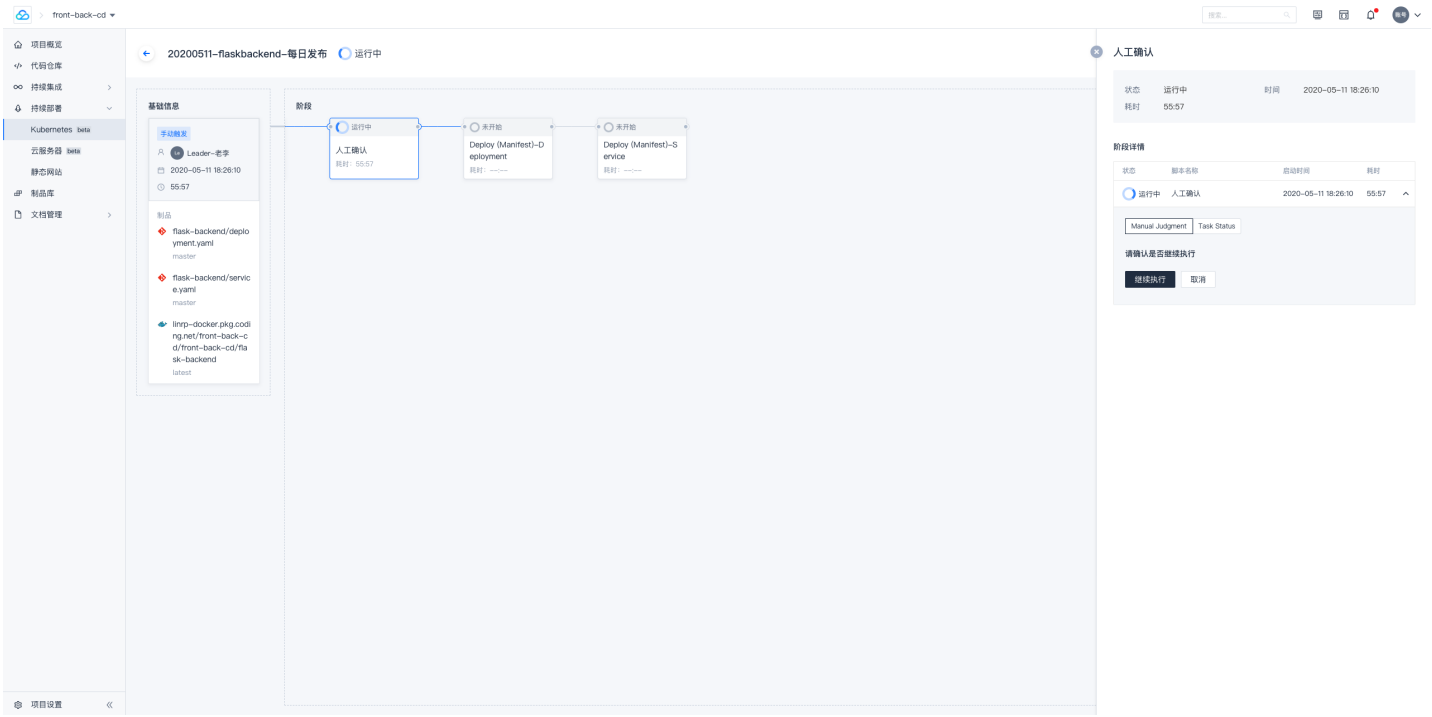
通知人和确认人两种角色是相互独立的，虽然被通知人不一定具备确认权限，但在收到通知后可以对发布单进行检查。另一种场景是确认人做出继续执行或终止阶段的操作后通知给提交发布单的人员。

确认入口

人工确认有项目内和部署控制台两个操作入口：

- 如果应用已关联项目，在通知内容中会将确认链接设置为第一个关联项目的发布单详情地址。
- 如果没有关联项目，在通知内容中会将确认链接设置为部署控制台的地址。

跳转到项目内发布单页面：



确认规则说明

- 若配置了确认人、则只有确认人具备确认权限。
- 若没有配置确认人，则具有部署设置权限的人员都具备确认权限。

常见问题

人工确认有超时限制么?

执行选项中的**超时时间**默认未设置，即为 72 小时，可设置为超过 72 小时。

收到人工确认通知，但跳转至确认页面后提示无确认权限?

确认人权限和通知人权限是相互独立的，被通知人是否具备确认权限取决于团队成员的权限设置。被通知人具备查看发布单执行详情的权限。

Patch (Manifest) 阶段

最近更新时间：2023-06-15 15:58:55

本文为您详细介绍 CODING 持续部署中部署流程的 Patch（Manifest）阶段。

前提条件

使用 CODING 持续部署的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击团队域名进入 CODING 使用页面。
2. 单击工作台首页左侧的  ，进入持续部署控制台。

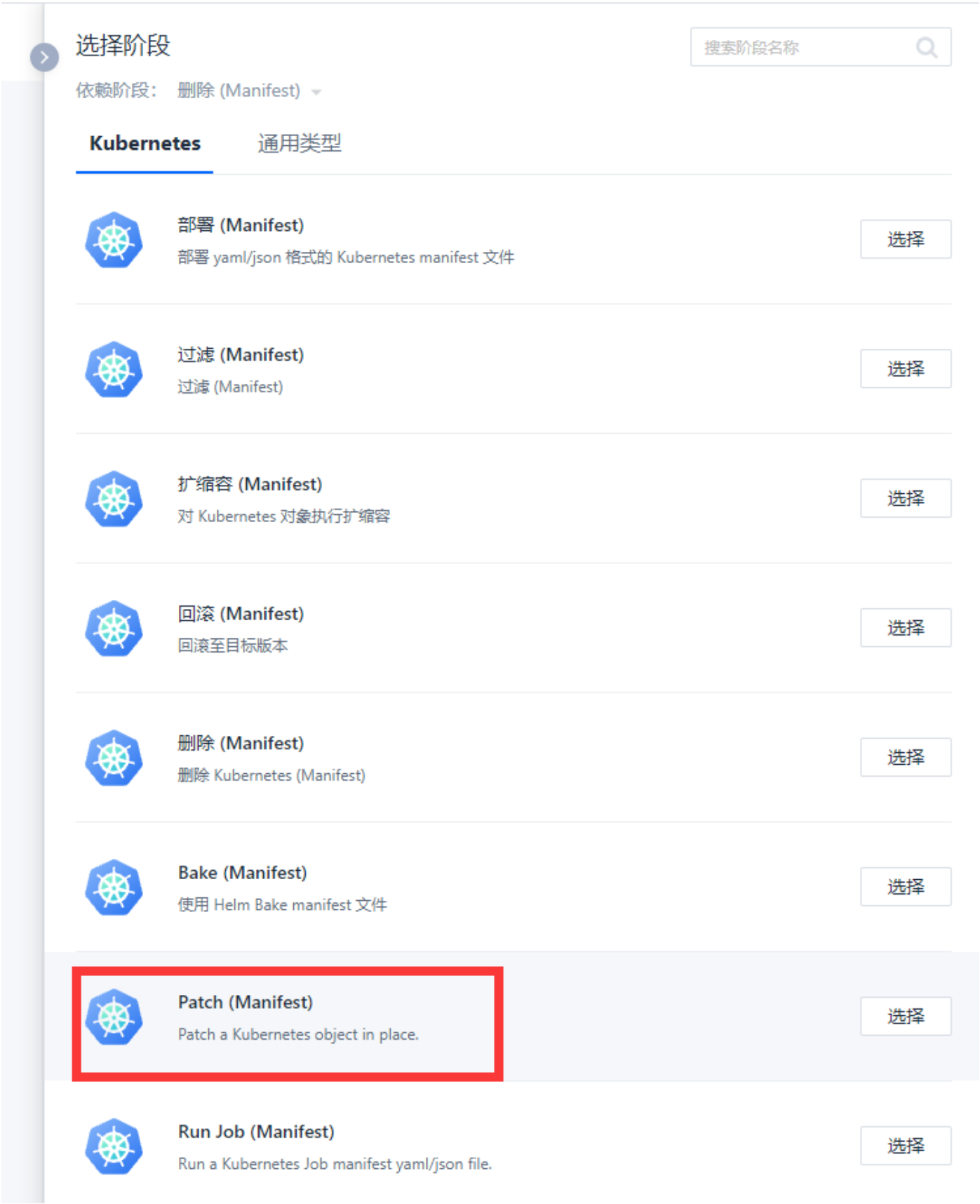
功能说明

Patch(Manifest) 阶段可以被理解为 kubectl patch 命令，该阶段能够对 Kubernetes 资源进行直接增量更新。本文档将会介绍如何在 Patch(Manifest) 阶段更新 Kubernetes 资源对象。将会包含以下几个操作说明：

- 指定更新的 manifest
- 指定更新内容
- （可选）在更新内容里覆盖制品
- （可选）覆盖特定的选项

指定更新资源

1. 在阶段下拉列表中选择 Patch（Manifest）。



2. 指定更新的 Kubernetes 资源，需要配置如下字段：
3. 云账号：管理 Kubernetes 资源的云账号

4. Namespace：资源所在的命名空间
5. 资源类型（例如 Deployment、Service 等）
6. 名称：资源名称

指定更新内容

更新内容与 Deploy（Manifest）类似。与部署阶段不同的是，更新阶段的内容只需填写更新的资源清单即可，不必提供所有资源的清单。根据不同需求，Patch 内容有两种提供方式：

- 静态：直接在阶段配置中填写
- 动态：运行时绑定制品

指定静态内容

您可以在阶段配置中直接输入 YAML 配置。例如给 manifest 添加新标签：

```
metadata:
labels:
foo: bar
```

动态绑定

与 Deploy（Manifest） 类似，您可以引用外部的 manifest 作为制品，制品必须是包含 patch 内容的文本文件。假设您在部署流程中声明了 Patch（Manifest）阶段需要引用的制品，那么在 Patch（Manifest）阶段可

以通过如下配置引用：

覆盖制品

当对 Kubernetes 资源对象使用 strategic 或 merge strategy 方式进行 Patch 操作时，Patch（Manifest）阶段可以像部署阶段一样配置覆盖制品。例如，假设部署流程中有如下 manifest 内容的更新阶段：

```
spec:
template:
spec:
containers:
- name: my-container
image: lhkprod-docker.pkg.coding.net/cd-demo/release/nginx
```

假设您更新了 Docker 镜像 tag（my-image：2.0），并上传镜像至 Docker 仓库触发部署流程执行，Spinnaker 将会使用新版本的 Docker 镜像覆盖掉旧版本：

```
#...rest of manifest
containers:
- name: my-container
image: lhkprod-docker.pkg.coding.net/cd-demo/release/nginx:2.0
```

指定更新选项

- 记录更新注解
  默认是 true。此选项勾选后，Kubernetes 将会使用注解 kubernetes.io/change-cause 将更新的操作与内容记录至被更新的资源中。
- 策略合并
  - strategic：默认选项。它是自定义版本的 JSON 合并更新，允许 Kubernetes 对象基于结构标签替换或者合并。当您需要在 pod spec 列表中添加注解、标签或容器时非常有用。
  - json：使用标准的 RFC 6902 JSON patch 更新 manifest。
  - merge：使用 RFC 7386 JSON Merge Patch 更新 manifest。

Run Job (Manifest) 阶段

最近更新时间：2023-06-20 15:05:05

本文为您详细介绍 CODING 持续部署中部署流程中的 Run Job (Manifest) 阶段。

前提条件

使用 CODING 持续部署的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击团队域名进入 CODING 使用页面。
2. 单击工作台首页左侧的  ，进入持续部署控制台。

Run Job（Manifest）阶段将运行一个 Kubernetes Job 作为部署流程的一部分。

查看执行日志

对于 job 类型的 Kubernetes 对象，日志显得尤其重要。考虑到 CODING 持续部署的用户对 Kubernetes 的理解程度有所不同，我们提供了两种获取和查看日志的方式。

将日志连接到外部系统

大多数 Kubernetes 部署过程都支持将日志从容器转发到外部系统以方便做日志分析。如果您也使用了这类日志平台，可以给 Job 配置注解 job.spinnaker.io/logs 以及日志系统的模板化 URL，注解的值将在 UI 界面中渲染成链接地址。

为了让查找 Job 更简单，注解值支持模板变量。模板变量在注解中以 {{ templateKey }} 格式使用，`emplateKey 根据不同的资源类型有不同的取值范围。已部署的 manifest 将会以 JSON 格式传入模板。

例如，假设您部署了名为 myjob-12345 的 Job，并使用了注解：

```
job.spinnaker.io/logs: 'https://internal-logging/jobs/{{ manifest.metadata.name }}'
```

注解内容将会被解析为：

```
https://internal-logging/jobs/myjob-12345
```

在部署控制台查看日志

除了将日志发送给外部系统，您也可以在 CODING 部署控制台中查看 Job 日志。CODING 部署控制台可以直接获取 Job 信息和日志并在 UI 界面展示。这些日志只在 Job 部署完成后保留一小段时间，因为 Kubernetes 针对对象的事件也只会保留一小段时间。如果需要将日志保留更长的时间，推荐您使用类似 Fluentd 这样的工具将日志转发至持久化平台（例如 Elasticsearch 或 Datadog）

抓取输出日志

在大多数场景中，Job 用于完成一些特定的工作并在结构化的输出（例如 JSON ）中记录了执行信息。这些输出可以通过部署流程表达式（SpEL）影响到下游阶段，您也可以使用日志或制品达到相同的目的。一些 Jobs 可能会输出大量的数据，处理起来很复杂。如果 Job 输出 megabytes 或 gigabytes 量级的日志数据，我们推荐您使用制品的方式。最简单和最容易理解的方式是将 Job 日志写到标准输出（stdout）。当 Job 执行完毕后，日志数据将会被两个不同的 markers 抓取和分析。

SPINNAKER_PROPERTY 文件

SPINNAKER_PROPERTY_*=* 可以用来提供单独的键值对，类似 Jenkins 的 build.properties 文件。跟随在 SPINNAKER_PROPERTY_*=* 后面的内容都会被作为键值对解析。例如有如下输出日志：

```
Checkout spinnaker/spinnaker source code...
Latest tag is: 1.1.0.
Tag spinnaker/spinnaker with next minor version...
Uploading release...
Release uploaded.

SPINNAKER_PROPERTY_RELEASE=1.1.1
SPINNAKER_PROPERTY_URL=https://github.com/spinnaker/spinnaker/releases/tag/1.1.1
```

最终将会被解析为如下内容：

```
{
"RELEASE": "1.1.1",
"URL": "https://github.com/spinnaker/spinnaker/releases/tag/1.1.1"
}
```

SPINNAKER_CONFIG_JSON

SPINNAKER_CONFIG_JSON 用于提供复杂的或结构化的数据。在以下案例中，SPINNAKER_CONFIG_JSON 后面的内容被解析为 JSON。

```
Checkout spinnaker/spinnaker source code...
Latest tag is: 1.1.0.
Tag spinnaker/spinnaker with next minor version...
Uploading release...
Release uploaded.

SPINNAKER_CONFIG_JSON={"RELEASE": "1.1.1", "URL": "https://github.com/spinnaker/spinnaker/releases/tag/1.1.1"}
```

最终将会被解析为如下内容：

```
{
"RELEASE": "1.1.1",
"URL": "https://github.com/spinnaker/spinnaker/releases/tag/1.1.1"
}
```

如果在输出日志中多次出现 SPINNAKER_CONFIG_JSON，那么每一个 SPINNAKER_CONFIG_JSON 对应的内容都会被解析并添加到 JSON 内容。如果出现了多个相同的 key，例如 key foo 同时出现在多个 SPINNAKER_CONFIG_JSON 中，将会使用最后出现的 key 所对应的 value 值。

如果 SPINNAKER_CONFIG_JSON 指向的是 JSON 文件（如 SPINNAKER_CONFIG_JSON=$(cat file.json）），请确保文件中没有换行符，因为在日志文件中 SPINNAKER_CONFIG_JSON 会被当做一整行解析。

制品

CODING 持续部署中的制品机制使得部署流程可以引用存储在外部系统的资源。制品可以使任何数据类型，例如 Docker 镜像、Kubernetes manifests 以及本案例将演示的 Run Job 阶段生成的数据。正如上述提到的，如果 Job 产生的数据量很大导致处理成本太高，那么使用制品是最佳的解决方案。

您必须使用 CODING 持续部署支持的制品格式才能使用制品捕获 Job 输出的数据。在 Job 执行完成后，其输出数据被作为制品注入到部署流程上下文，下游阶段可以使用 SpEL 表达式引用制品。Job 的输出必须是 JSON 格式才能将其作为制品。

假设 Job 执行完成后将输出数据推送到 S3 bucket 。

```
...
Operation completed successfully.
Publishing output to S3.
Output published to s3://my-bucket/my-job/output.json
```

在 Run Jon（manifest） 阶段中配置使用制品捕获输出数据。

清理旧文件

CODING 部署控制台（Spinnaker）不会回收 Run Job（Manifest） 阶段部署的 Jobs。Kubernetes 在 1.12 版本中针对 Job 发布了 alpha 版本的垃圾回收功能。如果需要用到垃圾回收，请联系您的 Kubernetes 集

群管理员确认是否支持此特性。

Bake (Manifest) 阶段

最近更新时间：2023-06-15 15:57:54

本文为您详细介绍 CODING 持续部署中部署流程的 Bake (Manifest) 阶段。

前提条件

使用 CODING 持续部署的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击团队域名进入 CODING 使用页面。
2. 单击工作台首页左侧的  ，进入持续部署控制台。

功能介绍

Bake（Manifest）阶段是对 Kubernetes（Manifest）模板进行渲染的阶段，目前支持的类型有 Helm2、Helm3、Kustomize。

在 Bake（Manifest）阶段，可以使用 Helm 将模板渲染为 manifest。相关的 Helm 命令是 `helm template`，如果您在本地安装了 Helm，运行 `helm template --help` 可了解更多细节。

Bake（Manifest）可以帮助您打包和部署应用，一般用于开发和部署频率高的应用。**如果只是一次性安装第三方包，不推荐使用 Bake（Manifest）。**

Bake（Manifest）阶段配置

Bake（Manifest）相关配置项有：

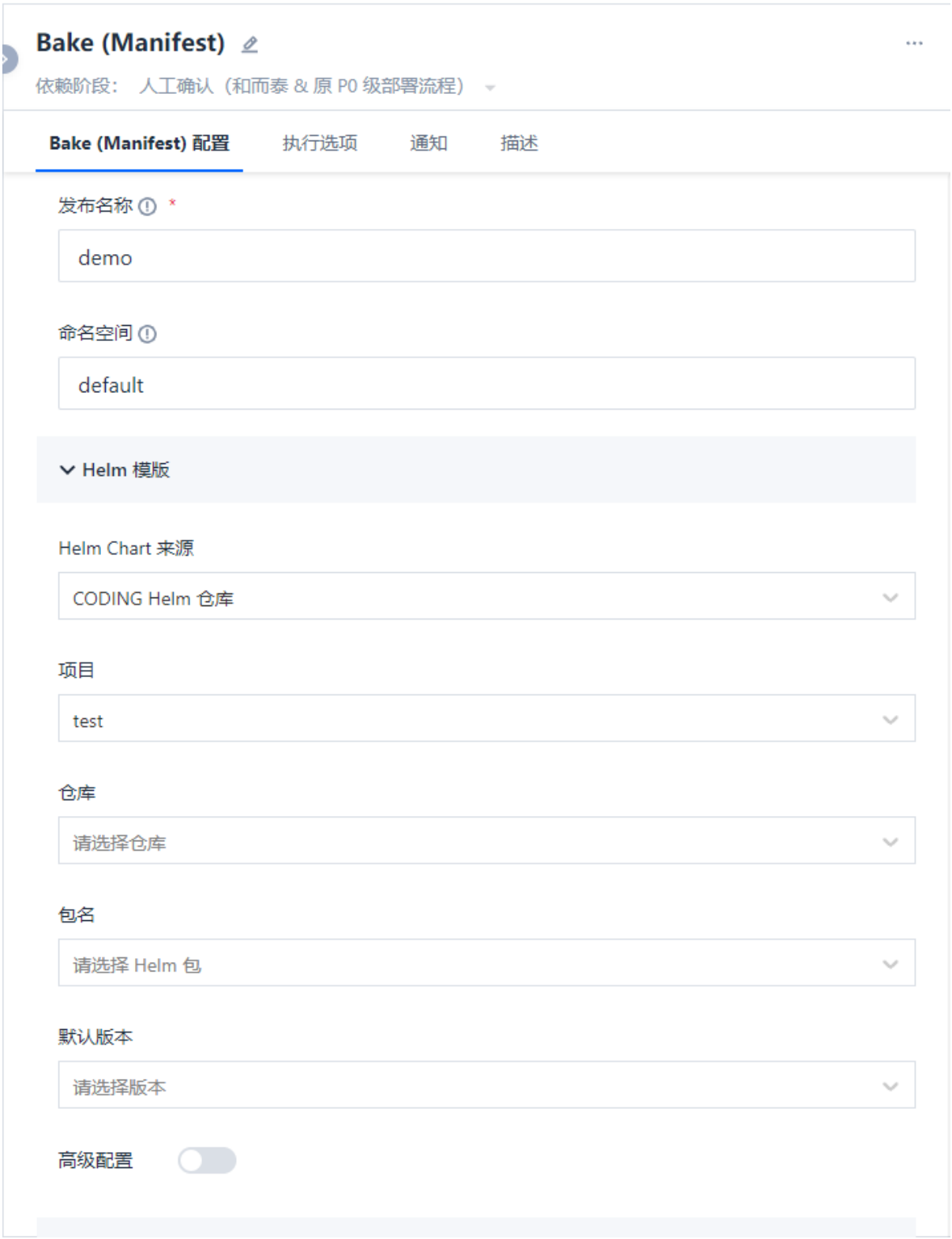
- 发布名称（必填）
  Helm chart 的发布名称。

  > 说明：
  > 这里配置的名称将会覆盖 Produces Artifacts 处配置的名称。

- 命名空间（可选）
  Kunernetes 安装发布包的命名空间。如果不指定参数，将使用 default 命名空间。需要注意的是并非所有的 Helm charts 都会在 manifest 文件有 namespace 定义，请确保 manifests 文件中含有如下内容：

```
metadata:
namespace: {{ .Release.Namespace }}
```

如下提供一个完整的 Bake(Manifest) 阶段配置示例：

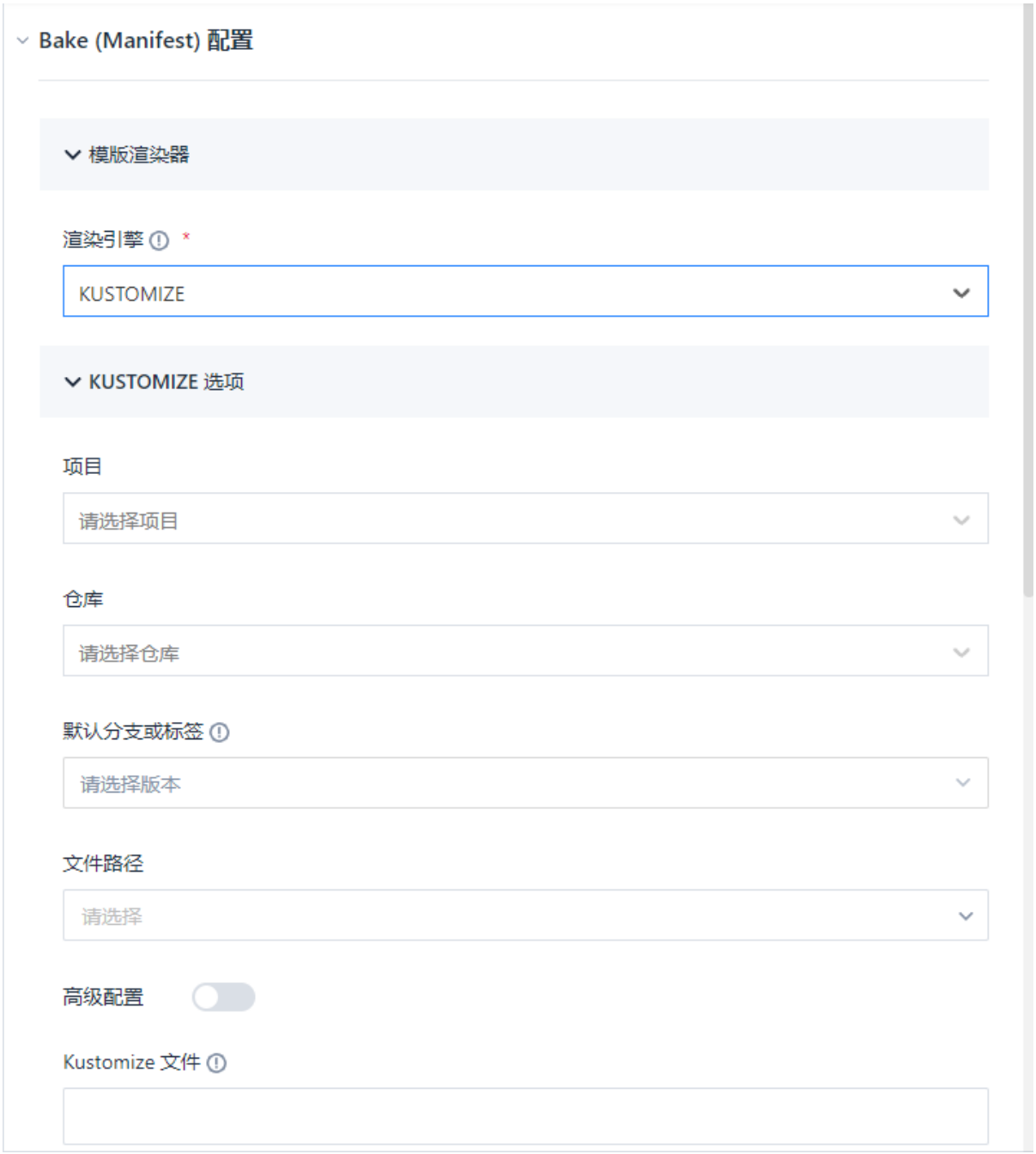


配置下游的部署阶段

当资源清单 Bake（manifest）集被 Helm bake 成功后，便可以继续配置下游的阶段（可以是同一个部署流程的阶段也可以是被当前部署流程所触发的新阶段）。配置如下：

其他的模板引擎

除了 Helm，CODING CD 还支持 Kustomize 模板引擎。



部署 (Manifest) 阶段

最近更新时间：2023-06-15 15:57:26

本文为您详细介绍 CODING 持续部署中的部署（Manifest）阶段。

前提条件

使用 CODING 持续部署的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

进入项目

1. 登录 CODING 控制台，单击团队域名进入 CODING 使用页面。
2. 单击工作台首页左侧的  ，进入持续部署控制台。

功能介绍

Deploy (Manifest) 阶段是部署 Kubernetes(Manifest) 的阶段，例如 YAML 或 JSON 文件，可以将其理解成 kubectl apply 命令。
该阶段包含两个主要的步骤：

- 指定要部署的 manifest。
- 在 manifest 中指定需要覆盖的制品（例如 Docker 镜像）。该步骤为可选项，您可以在阶段执行时指定需要覆盖的制品。

Deploy manifest

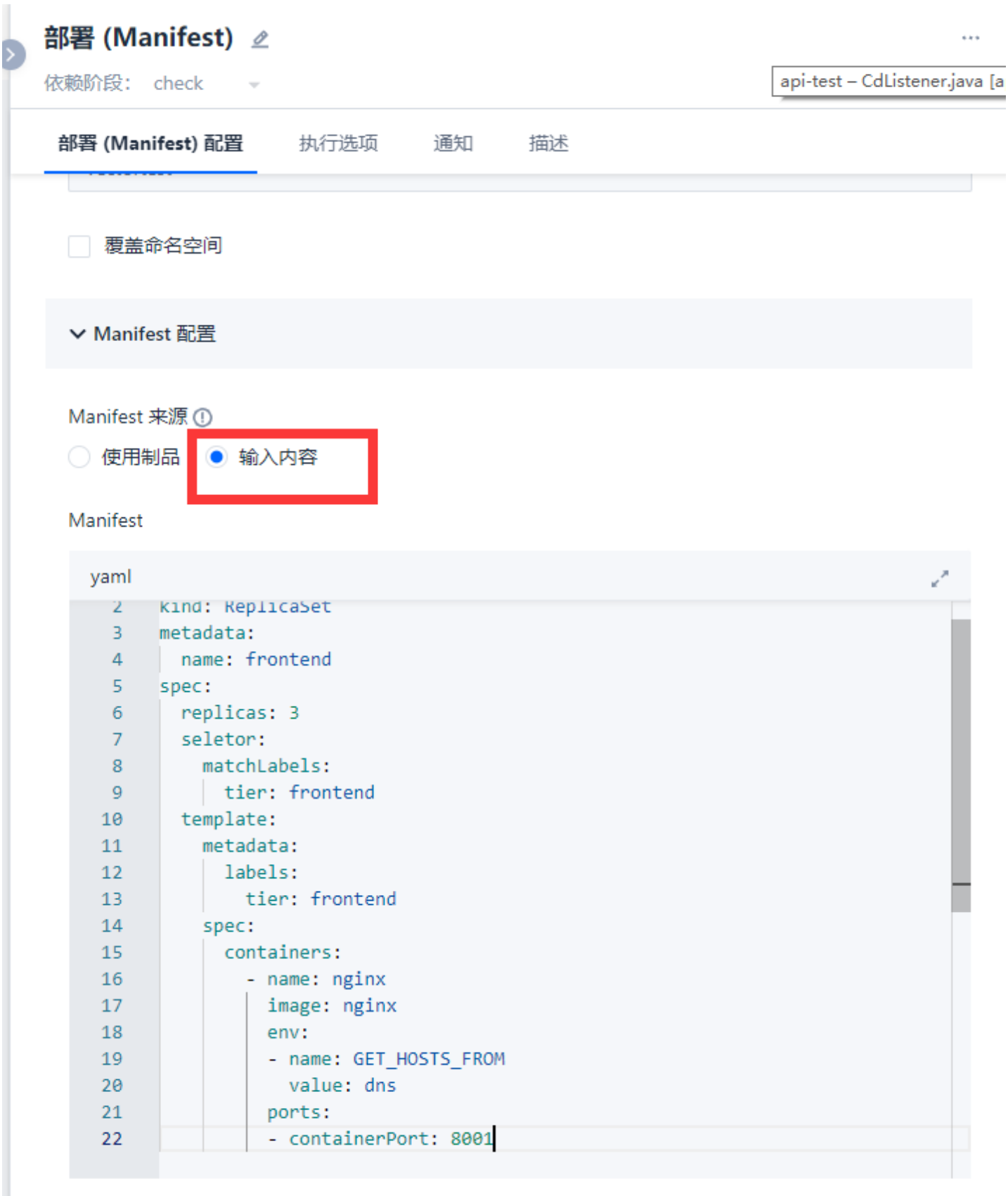
根据实际的生产需求，有两个方法可以指定 manifest：

- 静态：直接在部署流程中指定。
- 动态：运行时使用绑定的制品。

不管使用何种方式，都需要先选择 部署（Manifest）阶段：

配置静态的 manifest

如果您预知即将部署的资源所对应的 manifest，那么可以在 Deployment（Manifest）阶段配置中直接提供 manifest 纯文本内容。选择 **输入内容** 选项后，可以在文本框直接编辑 YAML 文件内容。



如果使用了 JSON 定义的部署流程，则对应的内容如下：

```
{
"name": "Deploy my manifest", // human-readable name
```

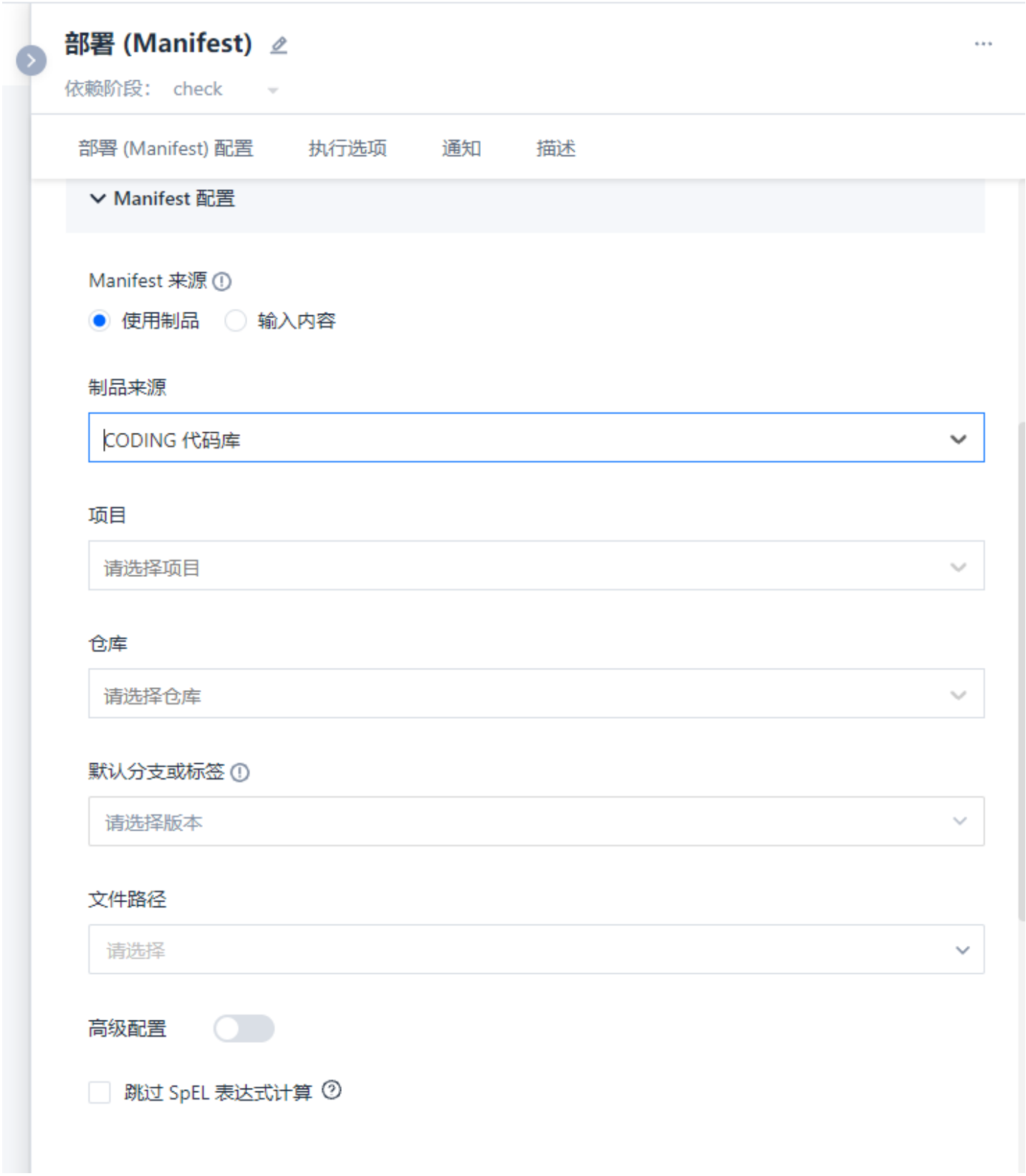
```
"type": "deployManifest", // tells orchestration engine what to run
"account": "nudge", // account (k8s cluster) to deploy to
"cloudProvider": "kubernetes",
"source": "text",
"manifest": {
// manifest contents go here
}
}
```

配置动态的 manifests

如果需要引用外部仓库的 manifest，或者当部署阶段需要同时部署多个制品时，可以通过绑定制品来配置 manifest。

CODING 持续部署的制品允许您引用任何远程的可部署资源。部署（Manifest）阶段所引用的制品必须是 manifest 文件，一般存储于 Git 仓库（如 CODING 代码仓库）。如果您已经在上游阶段声明了**启动所需制品**，那么可以在 部署（Manifest）阶段进行引用。

上游阶段可能会匹配到多个制品，例如配置正则表达式 .*\yml，将所有 yml 文件作为制品，部署（Manifest）阶段执行时会部署所有匹配到的 yml 文件。



注意：

制品来源勾选使用制品后，您可以自行选择部署上游所提供的制品，请确保云账号拥有下载制品的权限。

覆盖制品

通常情况下，当我们对 Kubernetes 资源进行部署更新操作时，大多数的变更都会涉及到 Docker 镜像或 ConfigMap 中的 flag。因此，CODING CD 针对这些资源类型的变更提供了优秀的适配性支持。

- Docker 镜像
- Kubernetes ConfigMap
- Kubernetes Secret

如果上游阶段的部署流程中存在这些资源对象，CODING CD 将会尝试自动将它们注入到正在部署的 manifest 文件中。

例如，假设 Docker 镜像仓库类型的触发器触发部署流程执行，并且触发器携带了镜像 lhkprod-docker.pkg.coding.net/cd-demo/release/nginx ，其 digest 值为 sha256:c81e41ef5e... 。在部署流程中，您配置了内容如下的部署阶段。

```
# ... rest of manifest
containers:
- name: my-container
image: lhkprod-docker.pkg.coding.net/cd-demo/release/nginx
# rest of manifest ...
```

因为部署流程是由 Docker 镜像内容变更所触发，所以部署流程编排引擎会将 Docker 镜像制品连同部署阶段中的 manifest 一起派发给 clouddriver 组件处理。最终得到部署的 manifest 内容如下。

```
# ... rest of manifest
containers:
- name: my-container
image: lhkprod-docker.pkg.coding.net/cd-demo/release/nginx@:sha256:c81e41ef5e...
# rest of manifest ...
```

为了确保部署阶段获取到正确的制品，您可以强制阶段绑定所有需要的制品，如果绑定失败，阶段将执行失败。以下配置的含义是 Docker 镜像 lhkprod-docker.pkg.coding.net/cd-demo/release/nginx 必须被绑定到 manifest，否则阶段将会执行失败。

主机部署
主机部署介绍

最近更新时间：2021-08-10 16:51:28

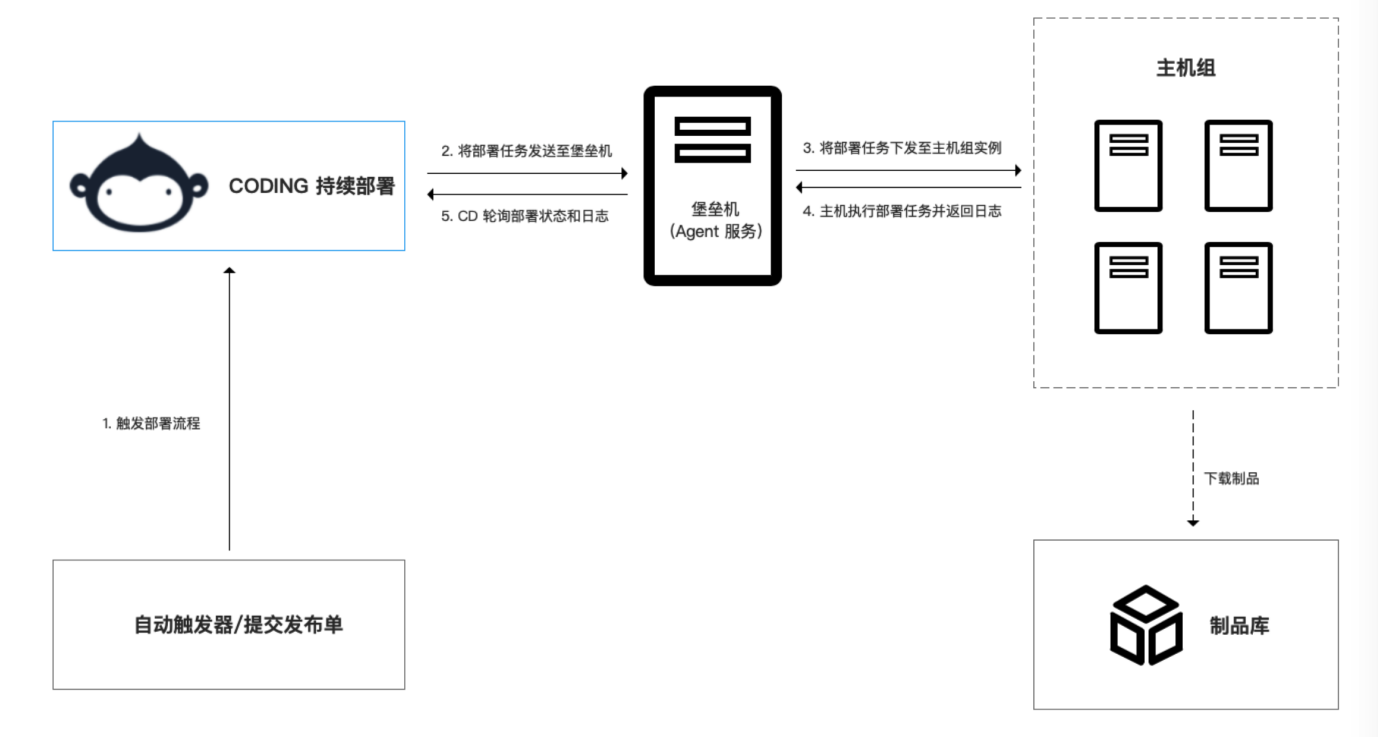
功能介绍

主机部署提供了基于物理机或云服务器的通用部署能力，支持将应用部署到公有云、混合云、私有云/私有环境中。主机部署是早期软件开发里常见、常用的部署方式，但随着 Docker、Kubernetes 的兴起，针对此类部署方式的支持也愈加稀少。因此一套整合了代码、制品、部署的整套流程更是屈指可数。如今 CODING CD 推出的主机组部署功能，正好能够弥补上这个缺口，让这种看似“原始”的方式也能加入进 DevOps 环中。

相关概念

概念	说明
堡垒机	堡垒机是 CODING 持续部署服务与主机之间的代理，CODING CD 通过堡垒机上运行的 Agent 服务管控应用发布过程。
主机组	主机组是主机实例的集合，通常一个主机组对应应用的一个发布集群（测试集群、预发集群、生产集群）。
服务	CODING 持续部署抽象的概念。在部署（主机组）阶段需要定义本阶段部署的服务名称，CODING CD 基于此服务名实现版本管理和回滚。

架构图


CODING 持续部署
2. 将部署任务发送至堡垒机
5. CD 轮询部署状态和日志
堡垒机
(Agent 服务)
3. 将部署任务下发至主机组实例
4. 主机执行部署任务并返回日志
主机组
1. 触发部署流程
下载制品
自动触发器/提交发布单
制品库

堡垒机

最近更新时间：2021-08-26 11:03:18

本文为您介绍 CODING 持续部署中的堡垒机。

前提条件

使用 CODING 持续部署的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

功能介绍

堡垒机是 CODING CD 与主机之间的代理，CODING CD 通过堡垒机上运行的 Agent 服务管控应用发布过程。您可以在 CODING 首页工作台中的**功能设置** > **持续部署** > **主机部署-堡垒机**页面管理堡垒机。

添加堡垒机

只需两步即可添加堡垒机：

1. 指定堡垒机名称（支持中文、字母、中划线和下划线）
2. 拷贝 Agent 服务的下载/安装链接，在堡垒机上执行命令快速安装 Agent（Agent 安装成功后将自启动）。

主机组

最近更新时间：2021-08-26 11:03:27

本文为您介绍 CODING 持续部署中的主机组。

前提条件

使用 CODING 持续部署的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

功能介绍

主机组是主机实例的集合，通常一个主机组对应应用的一个发布集群（测试集群、预发集群、生产集群）。

您可以在 CODING 首页工作台中左侧的**功能设置** > **持续部署** > **主机部署-主机组**页面管理主机组。

添加主机组

表单名	说明
堡垒机	此主机组绑定的堡垒机，堡垒机与主机组是一对多的关系
主机组名称	支持中文、字母、中划线和下划线
认证方式	堡垒机和主机实例之间的通信支持 密码 和 密钥 两种认证方式
标签	自定义字段；可用于标识主机组所在地域，分类等信息
实例 IP	主机实例的 IP，多个 IP 地址换行填写

堡垒机 Agent

最近更新时间：2021-08-26 11:03:40

本文为您介绍如何在 CODING 持续部署中安装堡垒机 Agent。

前提条件

使用 CODING 持续部署的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

功能介绍

cloud-agent 是运行在堡垒机上的服务。堡垒机凭借此服务，接受来自 CODING CD 的指令，并利用 ssh 隧道对主机实施部署操作。

首次安装

您可以在 CODING 首页工作台中的**功能设置** > **持续部署** > **主机部署-堡垒机**页面中获取 cloud-agent 的安装命令。复制后，在堡垒机的适当目录下，直接粘贴运行即可启动 cloud-agent。

cloud-agent 成功运行后会出现以下提示

```
# root @ VM-91-209-centos in ~ [16:59:28]
$ curl -sL "                                                                    " | bash -s
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   505  100   505    0     0   1518      0 --:--:-- --:--:-- --:--:--  1516
100 11.4M  100 11.4M    0     0  3129k      0  0:00:03  0:00:03 --:--:-- 4276k
cloud-agent
INFO[2020-10-19 16:59:36.086] connecting to wss://codingcorp.coding.net/ncd/ws/connection
INFO[2020-10-19 16:59:36.456] 连接响应数据：&{Status:101 Switching Protocols StatusCode:101 Proto:HTTP/1.1 ProtoM
ader:0xc00025c180} ContentLength:0 TransferEncoding:[] Close:false Uncompressed:false Trailer:map[] Request:0xc00
INFO[2020-10-19 16:59:38.457] done init db
sort migrate
ready to run db migration: 1597831087
run db migration: 1597831087 succeed
Cloud agent is running.
```

在**主机部署-堡垒机**页面中即可看到相应 cloud-agent。

后续运行

如果后续遇到 cloud-agent 断开的情况，可以人工登录到堡垒机上，先通过运行 ./cloud-agent stop 确保 cloud-agent 服务关闭

```
# root @ VM-91-209-centos in ~ [16:59:38]
$ ./cloud-agent stop
cloud-agent is stopped.
```

然后再通过 ./cloud-agent up -d 手动启动 agent。

```
# root @ VM-91-209-centos in ~ [17:04:26]
$ ./cloud-agent up -d
Cloud agent is running.
```

更新

可以通过运行 ./cloud-agent update 来进行更新。当有新版本的 agent 时，会进行相应的更新操作。

```
# root @ VM-91-209-centos in ~ [17:05:20]
$ ./cloud-agent update
INFO[2020-10-19 17:08:09.306] Cloud agent is the latest version.
```

隐藏目录介绍

位置（堡垒机）： ~/.coding-cd

```
.
├── config
│   └── cloud-agent.yaml # 配置信息
```

```
├── db
│   ├── cloud-agent.db # sqlite 数据库，存放主机部署相关信息，例如部署内容、部署日志等。
│   └── cloud-agent.sql # 数据库表结构。
├── log
│   └── cloud-agent.log # 运行日志
└── scripts
    └── cloud-agent.sh # 辅助脚本
```

位置（主机）： ~/.cloud-agent

cloud-agent 是否执行过某次部署中的某个脚本的标记

子命令概览

子命令	功能
completion	设置自动补全。详情可通过 cloud-agent completion --help 查看
init	初始化 cloud-agent 所需配置
language	设置语言，默认为英语(en/cn)
stop	停止 cloud-agent 服务
up	启动 cloud-agent 服务
update	更新 cloud-agent 版本
version	打印 cloud-agent 版本

常见问题

如果由于配置问题，导致 cloud-agent 启动失败，怎么办?

可以考虑先使用 rm -rf ~/.coding-cd/config 删除配置信息。删除配置项之后，再次运行首次安装时所运行的命令。

部署阶段配置

最近更新时间：2023-06-15 15:55:43

本文为您介绍主机组部署阶段中的阶段配置。

前提条件

使用 CODING 持续部署的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

功能介绍

部署（主机组）包含 5 个配置项。

基础设置

在基础设置部分选择部署的目标主机组，并指定部署的服务名称（CODING 持续部署基于服务名称实现版本管理和回滚等操作）。

前置脚本

前置脚本会在主机组的所有实例上运行，常为停服、备份旧版本等操作。脚本有**使用制品**和**输入内容**两种来源，其中**制品**支持 Git 仓库文件和 Generic 仓库文件。

制品下载

制品下载有两项配置：

- 选择需要下载的制品，目前支持 Generic 仓库文件。
- 指定目标主机的绝对路径，例如：/opt/download/artifacts/package.tgz。

后置脚本

前置脚本会在主机组的所有实例上运行，常为制品解压缩、启动服务等操作。脚本有**使用制品**和**输入内容**两种来源，其中**制品**支持 Git 仓库文件和 Generic 仓库文件。

健康探针

健康探针有 HTTP 探针和 SHELL 探针两种类型。

- HTTP 探针

  字段说明：

字段	说明
请求路径	指定 HTTP 探针的请求路径，注意是主机组中的实例自己对自己进行 HTTP 请求。
间隔时间	每次请求的间隔时间。
超时时间	HTTP 请求的超时时间。

- SHELL 探针

  表单项说明：

字段	说明
脚本来源	脚本有**使用制品**和**输入内容**两种来源，其中**制品**支持 Git 仓库文件和 Generic 仓库文件。
间隔时间	Shell 脚本执行的间隔时间。
超时时间	Shell 脚本执行的超时时间。

- **运行脚本**

  部署流程中可以从主机组部署中选择运行脚本阶段。

- **基础设置**

  基础设置中选择主机组和部署策略（普通部署或滚动部署）。

- **脚本配置**
  脚本会在主机组的所有实例上运行，脚本来源支持制品和自定义输入内容。

运行脚本阶段

最近更新时间：2023-06-15 15:55:15

本文为您介绍主机组部署中的“运行脚本”阶段。

前提条件

使用 CODING 持续部署的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

功能介绍

在部署流程中，可以从主机组部署中选择运行脚本阶段。

基础设置

在基础设置中选择主机组和部署策略（普通部署或滚动部署）。

脚本配置

脚本会在主机组的所有实例上运行，脚本来源支持制品和自定义输入内容。

查看部署详情

最近更新时间：2023-06-15 15:54:47

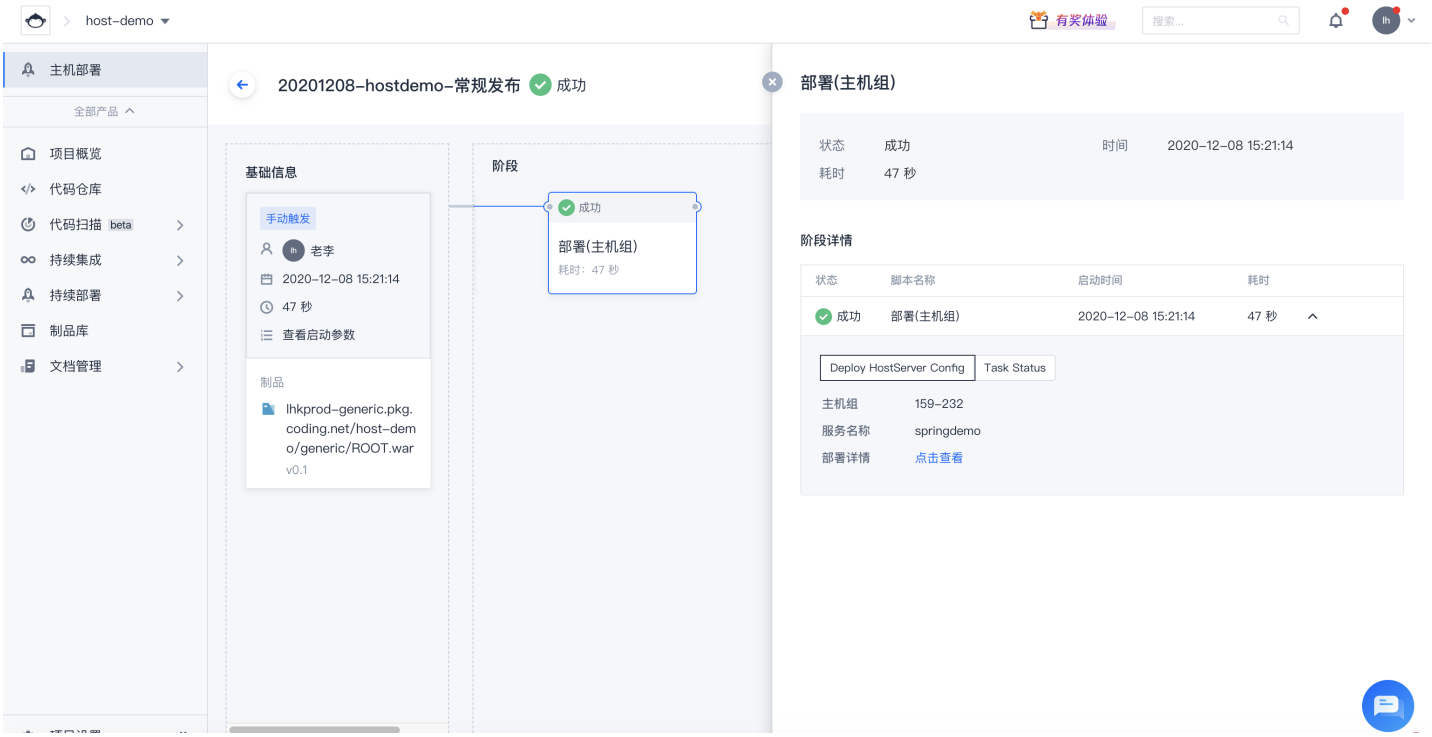
本文为您介绍如何查看部署详情。

前提条件

使用 CODING 持续部署的前提是，您的腾讯云账号需要开通 CODING DevOps 服务，详情请参见 开通服务。

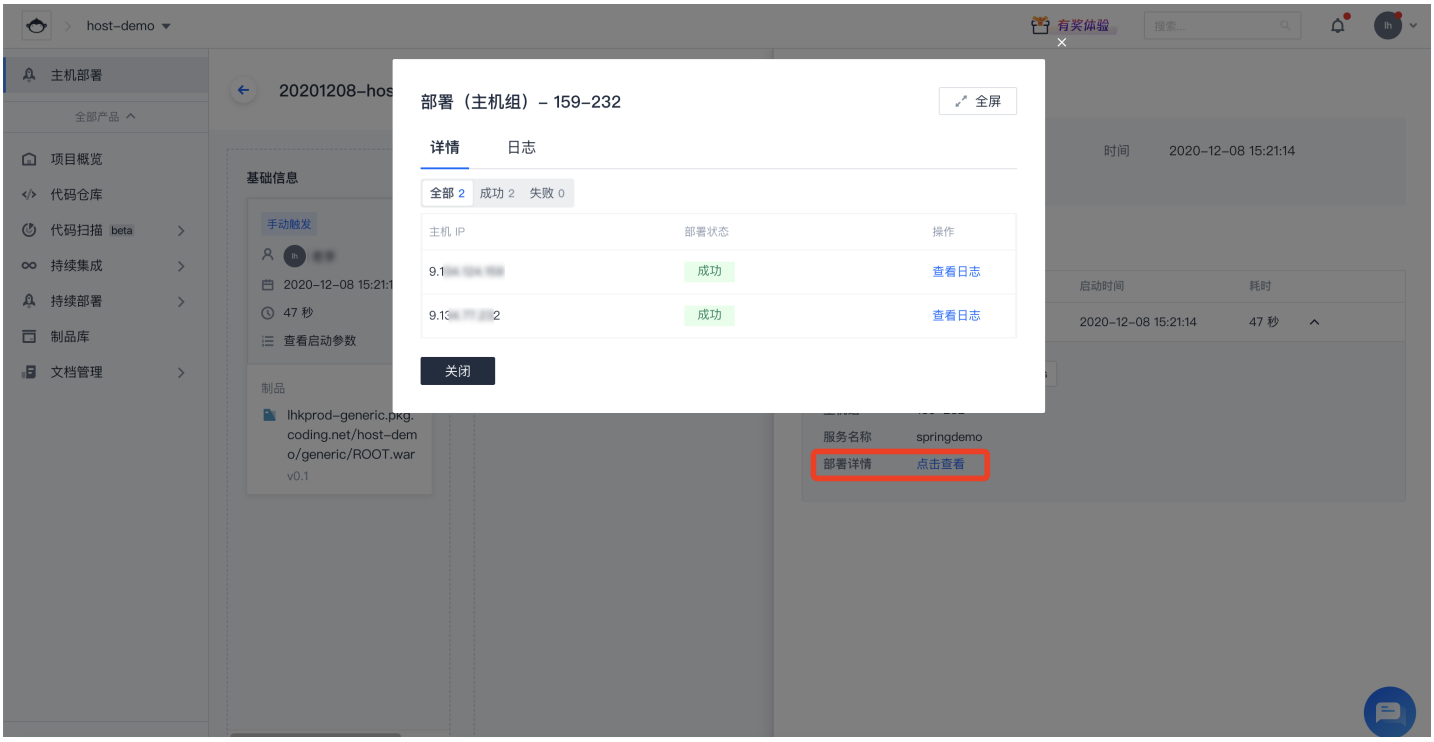
功能介绍

在项目内提交发布单，完成应用部署后，可点击对应阶段查看执行详情。



部署详情

部署详情可查看所有主机实例的部署状态和详细日志。



部署日志

可查看每台主机实例的详细部署日志，日志包含执行内容和执行结果。