什么是应用工作台?

应用工作台是基于容器的 DevOps 云原生应用平台，提供了 DCE 5.0 应用创建的统一入口，通过界面化表单创建多种流水线、GitOps、金丝雀、蓝绿、AB 等渐进式发布策略、项目管理、工具链集成等多种功能。

应用工作台着重企业应用自动化交付和基础设施变更的过程，提供业务应用从“开发 -> 测试 -> 部署 -> 运维”的全生命周期管理，能有效帮助企业实现数字化转型，提升企业的 IT 交付能力和竞争力。

=== “层级多租户资源管理”

以[工作空间](../../ghippo/user-guide/workspace/ws-folder.md)作为最小的租户单元，不仅支持单集群的独享资源的能力，还支持跨集群共享资源的能力：

- 支持弱绑定集群，从而获得跨集群、跨命名空间共享资源的能力
- 支持强绑定集群，从而获得独享资源的能力
- 工作空间管理成员可以在关联的集群中创建命名空间资源
- 自助式资源创建的模式，用户可以在工作空间内自助创建命名空间来进行资源隔离

![多租户资源管理](https://docs.daocloud.io/daocloud-docs-images/docs/amamba/images/what01.png)

=== “以云原生应用为中心”

支持云原生场景下的“多形态”的云原生应用，包括 Kubernetes 原生应用、Helm 应用、OAM 应用等。
能够接入 SpingCloud、Dubbo、ServiceMesh 框架的微服务应用，实现微服务的治理，与 DCE 5.0 的[微服务引擎](../../skoala/intro/index.md)、[服务网格](../../mspider/intro/index.md)无缝集成。
支持云原生应用的全生命周期管理，例如应用的扩缩容、日志、监控查看、更新应用等等。

![多形态云原生应用](https://docs.daocloud.io/daocloud-docs-images/docs/amamba/images/what02.png)

=== “高效的持续集成”

支持 Jenkins 和 Tekton 双流水线引擎系统。采用图形化方式编辑流水线，做到所见即所得的效果。可以用不同来源的代码构建应用。

![持续集成](https://docs.daocloud.io/daocloud-docs-images/docs/amamba/images/what03.png)

=== “安全自动化渐进式交付”

应用工作台引入一种为云原生应用实现持续部署的理念 – GitOps，全面拥抱 GitOps，以集成渐进式交付组件 Argo Rollout，支持灰度发布，从而提高应用交付的稳定性和效率。

![自动化渐进式交付](https://docs.daocloud.io/daocloud-docs-images/docs/amamba/images/what04.png)
!!! info
渐进式交付是一种将应用程序的新版本逐步暴露给最初的一小部分用户，然后逐渐变得越来越大的子集的做法，以减轻负面影响（例如错误）的风险。

Argo-Rollout Kubernetes Progressive Delivery Controller，提供更强大的部署能力。包括灰度发布、蓝绿部署、更新测试 (experimentation)、渐进式交付 (progressive delivery) 等特性。

在 DCE 5.0 中的地位

以容器管理为底座，借助全局管理实现层级资源管理，以 CI/CD 流水线和 GitOps 流程增删改查云原生应用，实现渐进式交付。

应用工作台在 DCE 5.0 中的地位

应用工作台在 DCE 5.0 中的地位

部署方法

建议通过 DCE 5.0 商业版安装包安装应用工作台，因为这样可以一次性安装 DCE 5.0 的所有模块 ，无需担心组件不兼容的问题。

但是，如果你想单独安装或升级应用工作台模块，依次执行以下命令即可：

```
export VERSION=**** # (1)!
helm repo add amamba-release https: //release.daocloud.io/chartrepo/amamba
helm repo update amamba
helm upgrade --install --create-namespace --cleanup-on-fail amamba amamba-release/amamba -n amamba-system --version=${VERSION}
```

1. 修改为实际部署的版本

下载 DCE 5.0 安装 DCE 5.0

应用工作台功能特性

应用工作台是 DCE 5.0 商业版所包含的模块，提供以下功能特性。

功能特性	描述
应用管理	- 支持云原生场景下的“多形态”的云原生应用，包括 Kubernetes 原生应用、Helm 应用、OAM 应用等。- 提供云原生应用的全生命周期管理，例如应用的扩缩容、日志、监控查看、更新应用等。- 支持接入 SpingCloud、Dubbo、ServiceMesh 框架的微服务应用，实现微服务的治理，与 DCE 5.0 的微服务引擎、服务网格无缝集成。
流水线编排	- 支持四种创建流水线的模式：自定义创建、基于 jenkinsfile 创建、基于模板创建、创建多分支流水线。- 支持流水线图形化编辑。- 可通过 Git 源码、Jar 包、Helm chart、容器镜像的方式来构建应用。

功能特性	描述
凭证管理	为流水线中的代码库、镜像仓库提供不同类型的凭证管理功能。
持续部署	- 引入 GitOps 理念实现应用持续部署，用于把控代码构建之后的应用发布与部署交付流程。- 基于 Argo CD，以自动化方式，频繁且持续性地将企业的应用部署到生产环境。- 提供 Argo CD Application 的创建、同步、删除管理。
仓库管理	支持导入 Git 的代码仓库，导入后您可以在持续部署中使用该仓库进行应用的持续部署。
灰度发布	- 灰度发布可以保证整体系统的稳定，在初始灰度时就可以发现、调整问题，减少问题的影响范围。- 支持金丝雀发布、蓝绿部署、A/B Testing 高级发布策略。- 金丝雀发布支持自动化渐进式的发布流程。- 支持通

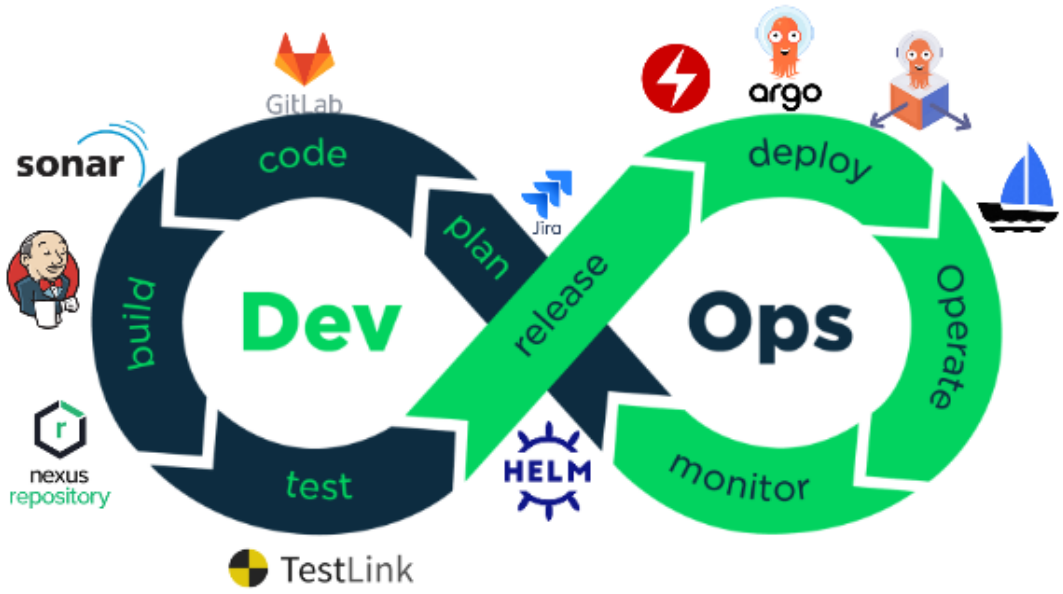
功能特性	描述
	过监控指标分析来快速回滚应用。

应用工作台技术概览

[Amamba]: DCE 5.0 应用工作台开发代号 [Mspider]: DCE 5.0 服务网格开发代号

目标

- 对应用工作台架构有个整体认识
- 说明包含组件及集成方式
- 示例及引导文档

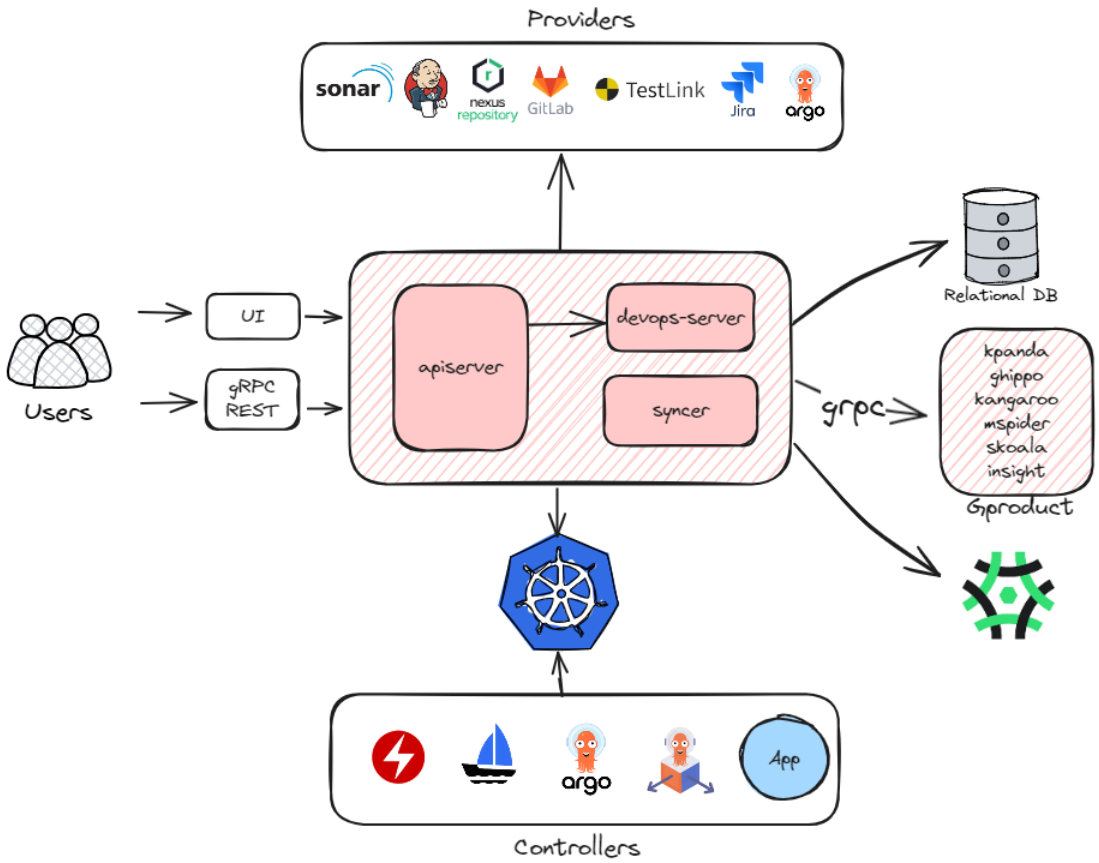


devops

系统组件

amamba-system 主要包含以下三个组件

- apiserver：所有服务的入口，无状态的，提供 gRPC 和 REST 两种方式访问，通过 REST 或 gRPC 调用 Providers 的服务，或者通过 kube-client 或者 kpanda 创建资源与 Controllers 交互
- devops-server：主要与 Jenkins 通信，并向 apiserver 提供相关接口以支持懒加载，基于事件的同步等功能
- syncer：主要负责维护 Providers 和 Controllers 的数据与系统的保持一致，确保系统的配置能正确地下发到各个组件当中



架构图

CI

Jenkins

Amamba Jenkins 基于 kubernetes 插件增强，相比于原生的 Jenkins 我们提供了以下优势：

1. 完整的云原生解决方案

   相比于传统的 VM 部署，云原生有着弹性和可伸缩性、高可用和灵活等优点，我们提供了整套的 Jenkins 云原生方案， 包括在 Kubernetes 中运行、构建镜像和制品、发布、和其他工具链集成、DR 等；

2. 优化的读取速度

   我们提供了基于关系型数据库的缓存，流水线及运行相关的数据都存储在 DB 里，并通过事件机制保证数据及时更新。

3. 一次创建，多处运行

   得益于 DB，Jenkins 的部署位置和持久化不再重要，我们时刻可以接入另一套集群中部署的 Jenkins，并在上面运行之前创建的流水线；

4. 更低的门槛

   Jenkinsfile 强大但有着比较高的理解和使用门槛，我们提供了图形化界面以拖拉拽的方式帮助用户创建和编辑流水线，帮助用户更快地上手；

5. 多租户

   基于 DCE5 的 Workspace 机制，我们对流水线和凭证做了隔离；

此外，目前应用工作台也有着以下问题：

1. 启动慢

   相比于传统的使用注册节点的 Agent，在基于 Kubernetes 的 Jenkins 上运行流水线需

要创建 Pod、等待调度和拉取镜像，这一过程导致启动的速度相对来说要更慢。

2. 缓存

由于 Pod 的沙箱环境，使得缓存无法被有效利用，我们可以为其挂载持久化存储，但这几乎总是意味着额外的规划和管理成本；

3. 插件

安装插件依然需要到 Jenkins 的管理界面上操作和配置，并且当迁移到其他集群上时会丢失。 我们建议通过 fork amamba-io/jenkins-agent 并修改 formula.yaml 后重新构建的方式，将插件持久化到镜像中；

4. Noisy Neighbour

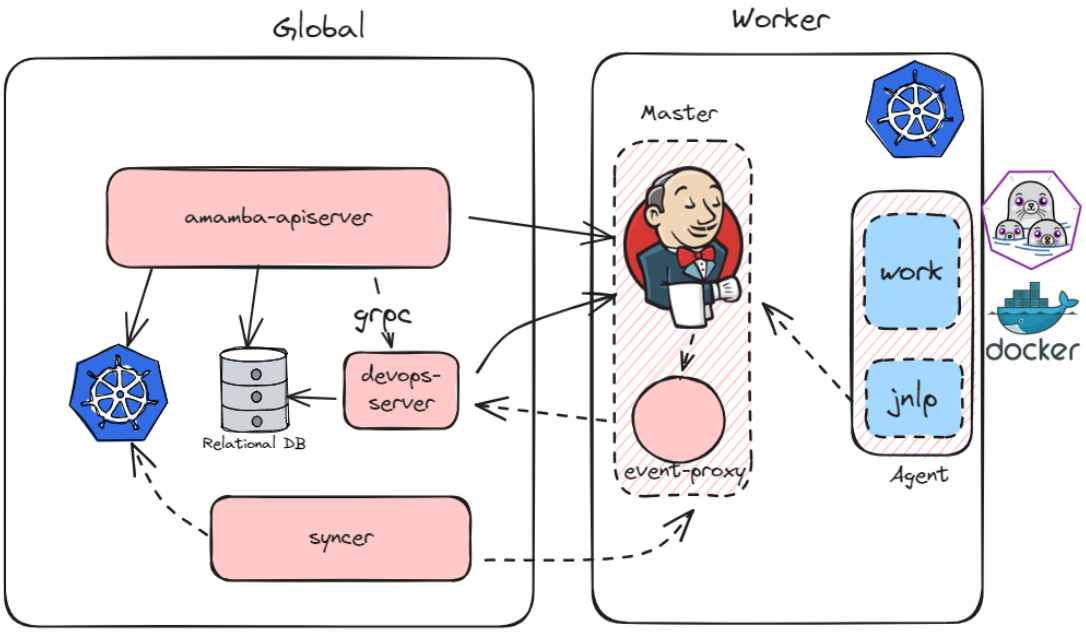
我们没有对流水线做资源的配额和限制，因为我们认为 CI 和部署的资源不应该放在一起考虑， 而是依赖于管理员对 CI 系统的前期规划，但这可能导致部分流水线抢占所有资源；

5. 构建镜像慢

除了前面说的缓存无法被有效利用的原因外，当 Kubernetes 的运行时不是 Dockerd 时，我们只能使用类似 dind 或 pind 的方式来构建镜像， 这种在容器里面运行容器的方式会导致一部分 IO 性能损失；

基于此，我们也在计划支持使用传统方式部署的 Jenkins。

整体架构



架构

- 利用 Generic Event 插件提供的事件机制， 我们可以及时地更新流水线的运行情况。为了避免不同网络环境带来的事件丢失的干扰， 我们以 sidecar 的方式部署了 event-proxy，这一组件提供更强的网络流量控制能力和可靠的事件系统。 当然，对于 Amamba 来说它是完全透明的，如果网络环境可控，可以不部署它；
- 为了保证 apiserver 是完全无状态可伸缩的，我们使用 devops-server 来处理状态相关的服务——包括接收 Jenkins 的事件、同步流水线状态、运行数据的懒加载、暴露 Jenkins 事件指标等，并提供了 SDK 给 apiserver 调用；
- 由于 Jenkins 的实例可能运行在任一集群上，syncer 帮助我们将其中系统的部分同步到不同实例里去，包括：
  - 我们的凭证统一使用 secret 管理，syncer 会将其中流水线的部分同步到 Jenkins 中；
  - Jenkins 的配置基于 CasC 生效， 我们会在系统中保留变更的部分，并使用

syncer 同步到子集群里面；

- 流水线实际在 Pod 里被执行，通过 jnlp 容器与 Jenkins 通信，这种架构使得即使 Jenkins 挂掉了大部分任务也能继续运行。

更多资源

- 更可靠的用法：
  - HA: Jenkins 高可用方案 - DaoCloud Enterprise
  - 配置和规划：Jenkins 场景配置 - DaoCloud Enterprise
- 扩展：
  - 自定义工具：在 Jenkins 中使用自定义工具链 - DaoCloud Enterprise
  - 复用流水线：自定义模板 - DaoCloud Enterprise
  - 自定义 Agent：自定义 Jenkins Agent 镜像 - DaoCloud Enterprise （**缺少 kubernentes 类型 agent 的定义方式**）
  - 自定义步骤：WIP
- 加速（WIP，缺一个系统的文档）；
  - 在指定的节点上运行流水线 - DaoCloud Enterprise
  - 在流水线中使用缓存 - DaoCloud Enterprise
- 集成：
  - SornaQube：使用流水线实现代码扫描 - DaoCloud Enterprise
  - 使用集成的 Gitlab：WIP（基本原理，如何使用）
- 实践：
  - 基于 Git 仓构建微服务应用 - DaoCloud Enterprise

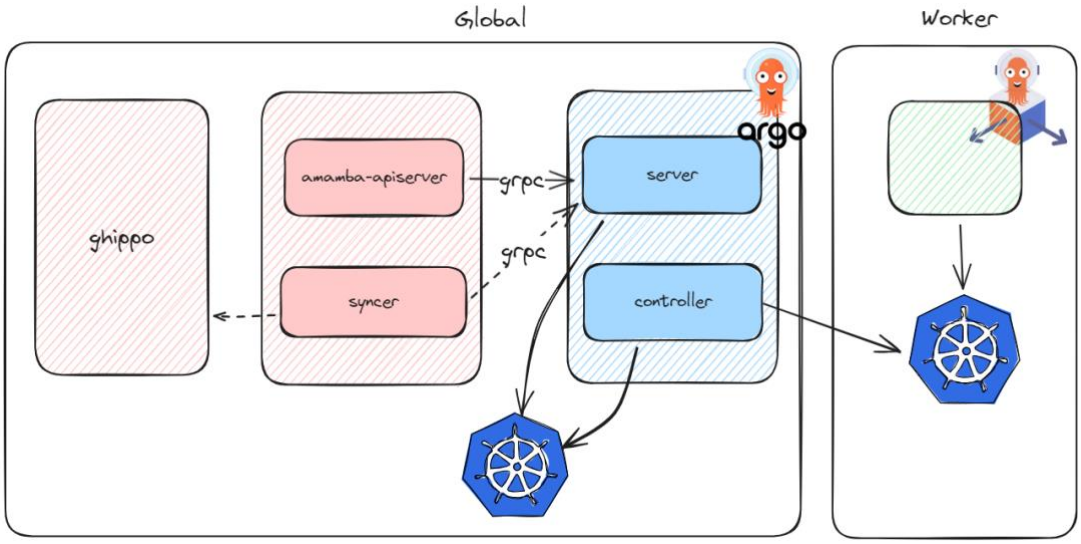
- 基于 Jar 包构建 Java 应用 - DaoCloud Enterprise
- 结合 ArgoCD 使用：流水线 + GitOps 实现 CI/CD - DaoCloud Enterprise
- 构建多架构镜像（WIP 有误）：构建多架构镜像 - DaoCloud Enterprise

CD

Argo CD

Amamba 使用 Argo CD 作为引擎实现 GitOps 的能力，相比原生的 Argo CD，我们主要在和 DCE 5.0 的集成上做了增强：

1. 应用按照租户的粒度隔离，只能部署到对应的集群和命名空间下；
2. 权限按照全局 RBAC 的策略控制，只有对应权限点的用户才能执行对应操作；



argo-cd

问题：

1. 代码仓库没有做租户隔离：这是 Argo CD 的设计缺陷，我们已经在积极推动社区优化：
   argoproj/argo-cd #18290
2. 暂不支持 ApplicationSet；

整体架构

- 我们通过 Argo 提供的 SDK 访问 argo-server，使用 AppProject - Wrokspace 一一映射的方式来为用户创建和更新相应的资源；
- syncer 会一直 watch 系统中租户的变化并将变化应用到 Argo 的资源中；

更多资源

- 更可靠地使用:
  - HA：Argo-CD 高可用部署 - DaoCloud Enterprise
  - 在 GitOps 中启用凭证加密功能 - DaoCloud Enterprise
- 多环境推进（Promote）部署（WIP）；
- How to Model Your GitOps Environments 提供了基于 kustomization 在多个环境间 Promote 的思路；
- GitHub - cloudogu/gitops-patterns: Collection of patterns, examples and resources for GitOps process design, GitOps repository structures, etc 总结了一些常见的 GitOps 的模式选择及利弊；

Argo Rollouts

Amamba 基于 Argo Rollouts 提供渐进式发布的能力，相比于原生的 Argo Rollouts，我们主要在以下方面做了增强：

1. 更加易于使用，提供了 Step By Step 的界面来将当前集群中的工作负载转换成 Rollout 开始灰度发布；
2. 支持跨集群创建和管理；

3. 与 Mspider（使用 istio 作为流量控制工具时）和 skoala（使用 contour 时）有更好的集成；

问题：

1. 现在 Mspider 的托管网格模式下，子集群部署的虚拟服务（VirtualService）和目标规则（DesinationRule）不会生效，这导致无法再这种场景下使用基于 Istio 的 Rollout，Mspider 已经在优化，预计在 v0.26 版本里支持；
2. 基于 Deployment 创建 Rollout 会导致暂时的访问失败，社区已经在着手修复这个问题 argoproj/argo-rollouts #3111，预计将在 v1.7 版本里解决；

更多资源

- 基于 Argo Rollout 实现灰度发布 - DaoCloud Enterprise
- 基于 Contour 的灰度发布 - DaoCloud Enterprise

应用

我们支持原生应用、Helm 应用、OAM 应用和 OLM 应用。

原生应用

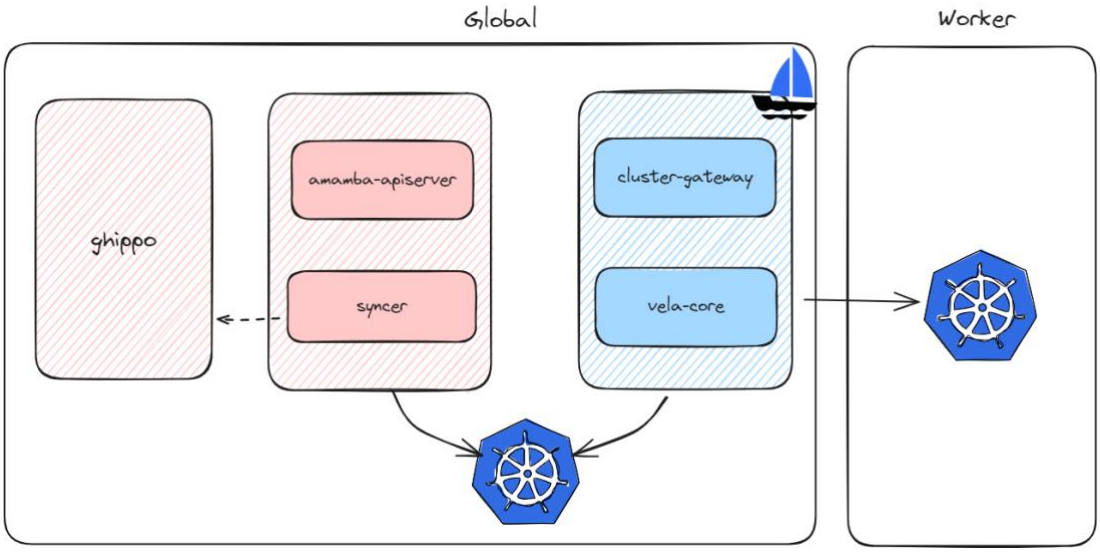
原生应用即 Kubernetes 原生的工作负载类型的资源，例如 Deployment，StatefulSet 和 DaemonSet。 我们没有增加任何的心智负担，但为了将相关资源关联起来，我们使用 kubernetes-sigs/application 来说明当前应用包含的资源类型和标签。

这个 CRD 对应的 Controller 是可选安装的，需要在每个子集群手动安装，不部署并不会导致资源创建失败， 但是部署后能够不断同步资源状态，返回资源的就绪情况。

OAM 应用

OAM 应用基于 kubevela 实现，我们主要在多租户方面做了增强：

1. 在当前租户下首次创建 OAM 应用时会提示需要创建或指定一个命名空间作为该 Workspace 下的所有 applications.core.oam.dev 放置的位置，以此实现不同租户下应用的隔离；
2. syncer 会一直 watch 系统中租户的变化并将变化应用到 KubeVela 的资源中；



kubevela

Helm 应用和 OLM 应用

OLM 应用和 Helm 应用主要基于 Kpanda 的相应能力封装。

版本信息

Amamba 的所有 Addon 从 v0.21 (对应安装器 v0.12) 开始，都可以在应用商店以 Helm 的方式部署， 没有版本限制。但是默认版本经过了完整的测试，可以更好地集成到当前系统中，下表是 Addon 和 Amamba 的版本关系：

Addon	起始版本	结束版本
amamba-io/Jenkins 0.3.1（2.413）	v0.21	v0.24
argocd 5.34.6（2.7.3）	v0.21	-
argo-rollouts 2.32.0（1.6.0）	v0.21	-
vela-core 1.7.7（1.7.7）	v0.21	v0.23
vela-core 1.9.7（1.9.7）	v0.23	-
amamba-io/Jenkins 0.3.2（2.413）	v0.24	-
amamba/kube-app-manager 0.1.4（0.8.3）	?	-

适用场景和优势

应用工作台是以流水线、GitOps、Jenkins 等方式实现应用持续交付的模块。 应用工作台提供了 DCE 5.0 应用部署的统一入口，支持云原生应用的整个生命周期管理。 降低了企业使用云原生应用门槛，提高了企业软件研发到应用交付的效率。

适用场景

应用工作台适用于以下场景：

- 持续交付流水线

  对于复杂的业务系统，从项目创建、编译、构建、自验、集成验证、类生产验证、上线的各个阶段都需要耗费大量的人力和时间，并且容易受到人为因素影响而出错。 持续集成和持续交付由于具有标准化和自动化特点，以代码变更为流动单元，基于发布流水线，拉通开发、测试、运维所有职能，持续、快速、高可靠地发布软件，可以很好的解决该问题。

image

image

- 基于流水线 + GitOps 实现云原生下的应用持续交付

  得益于 GitOps 理念，因此采用开源软件 Argo CD 来实践 Kubernetes 的应用发布。用户仅需将 Kubernetes 的 YAML 文件提交至代码仓库中，GitOps 会自动感知到 YAML 文件的变化，配合代码仓库的合并请求功能，将 YAML 文件的变更自动推送至集群中，整个过程无需学习 Kubernetes 的发布命令，也无需直接操作集群。

  image2

  image2

产品优势

应用工作台具有以下产品优势：

- 企业级 CI/CD

  使用集中管理的容器化工作流，持续开发持续交付，支持横向扩展，可以在高可用性 (HA) 环境中支持成千上万个并发用户和流水线。

- 云原生为底座

  应用工作台实现了应用构建环境的容器化管理，自定义构建机器等资源，确保流水线级资源隔离，同时支持多种类型的应用部署，使得云原生应用程序的交付更加简单。

- 提升研发效能

  在可视化的流水线构建环境中，预先封装声明性步骤，无需脚本就能创建复杂的流水线，操作简单。可根据资源大小自动创建/销毁环境，避免资源浪费。

- 灰度发布

  帮助用户渐进式更新应用，从而保障新特性能平稳上线。支持金丝雀部署、蓝绿部署等应用发布策略。

如果一个应用的发布策略配置为 ABTest，其 A 版本访问流量为 80%，B 版本访问流量为 20%，则灰度发布后将能够持续访问应用的两个版本，轻松统计出该应用不同版本的分流比例。

- GitOps

  提供基于 Kubernetes 的声明性 GitOps 持续交付能力，通过版本控制应用，从而实现自动化的应用部署与生命周期管理。

- 流水线即代码

  基于声明性（语法）的流水线易于学习，不同步骤的语法实现了标准化，其配置具有版本可控、模块化、可重用和声明性等特点。

- 综合集成

  集成社区流行的 DevOps 工具，单个流水线中的步骤可以在多操作系统、多体系结构节点上运行，支持在公有云、私有云或主机上独立部署，能很好地集成企业自有的系统和平台。 支持集成业界主流软件，如 Kubernetes、GitLab、SonarQube、Harbor 等。

- 多年行业沉淀

  在云原生领域耕耘多年，在金融、电商、制造等行业上百家企业有大规模实践经验，能够提供稳定可靠的 IT 交付生产线，帮助企业打造适合研发运行一体化平台。

基本概念

术语	描述
工作空间 Workspace	通过工作空间协调全局管理和子模块的权限关系，解决资源聚合和映射层级关系。一个工作空间对应一个项目，可以为每个工作空

术语	描述
	间分配不同的资源，指派不同的用户和用户组。
命名空间 Namespace	命名空间在平台上是工作空间下相互隔离的更小的资源空间，也是用户实现作业生产的工作区间。一个工作空间下可以创建多个命名空间，可占用的资源配额总和不能超过工作空间配额。命名空间更细粒度的划分了资源配额的同时，还限制了命名空间下容器的大小（CPU、内存），有效的提升了资源利用率。
流水线 Pipeline	流水线提供可视化、可定制的自动交付流水线，帮助企业缩短交付周期，提升交付效率。目前流水线是基于 Jenkins 实现。
凭证 Credential	流水线与第三方应用程序进行交互时，需要用户配置 Jenkins 凭证，流水线就可以与第三方应用程序交互。
GitOps	GitOps 的核心观点是使用包含当前期望的(生产环境基础设施的)声明式描述的 GIt 仓库，并通过自动化流程来确保生产环境与仓库中的期望状态保持一致。如果您想要部署一个新的应用或更新一个现有的应用，只需

术语	描述
	要更新相应的仓库即可(自动化流程会处理后续的事情)。这就像在生产中使用巡航控制来管理应用程序一样。
灰度发布	灰度发布是可以帮助用户渐进式更新应用的工具。它实现了多版本共存，发布暂停，流量百分比切换等功能，极大解放灰度发布过程中的手动操作，全自动化实现线上灰度流量切换。
工具链集成	集成 DevOps 工具链，允许团队将已了解和已在使用的现有工具引入到应用工作台之中。从而避免登录到多个平台之中以及应对在不同工具间没有统一视角的难题。

应用工作台权限说明

应用工作台支持三种用户角色：

- Workspace Admin
- Workspace Editor
- Workspace Viewer

每种角色具有不同的权限，具体说明如下。

1. 前往 **应用工作台** -> **流水线** -> **流水线** ，创建流水线

   流水线步骤为：拉取代码 -> 代码构建 -> 部署应用程序，以下为一个省略代码构建步骤的示例：

```
pipeline {
    agent {
      kubernetes {
        inheritFrom 'base'
        yaml '''
        spec:
          containers:
          - name: ssh
            image: your-custom-tooling-image
            command:
            - cat
''''
        defaultContainer 'ssh'
      }

    }
    stages {
      stage('clone') {
        agent none
        steps {
          container('ssh') {
            git(branch: 'main', credentialsId: 'gitsecret1', url: 'https://gitlab.daocloud.cn/**
*/***.git')
          }

        }
      }

    stages {
      stage('build') {
        agent none
        steps {
          container('ssh') {
            sh 'build commend'
          }

        }
      }
      stage('deploy') {
        agent none
        steps {
          container('base') {
            withCredentials([string(credentialsId:'sshpasswd',variable:'sshpasswd')]) {
              sh '''sshpass -p $sshpasswd ssh  -o StrictHostKeyChecking=no root@10.1.
```

```
5.53 mv -f /usr/share/nginx/html/* /tmp
sshpass -p $sshpasswd scp -r -o StrictHostKeyChecking=no ./* root@10.1.5.53:/usr/share/nginx/html/
sshpass -p $sshpasswd ssh -o StrictHostKeyChecking=no root@10.1.5.53 nginx -s reload'"
            }

          }

        }
      }

    }
}
```

2. 创建成功后，运行流水线

在物理机/虚拟机上集成 Jenkins

风险

由于 Jenkins 部署在物理机或虚拟机中，集成之后会有一些基于 K8s 的功能可能无法使用。

- 由于应用工作台将流水线放在 WorkspaceID 同名的文件夹下，这会导致已经存在的流水线无法被发现和展示在工作台的界面中
- 所有基于 casc 的配置都会失效，包括 SonarQube 集成、全局邮件配置
- 如果 Jenkins 不是使用 KubernetesCloud 运行，那么相关的语法会失效，需要手动调整 Jenkinsfile

插件

必需的插件

安装 Jenkins 时会自动安装以下依赖的插件：

- Folders Plugin

- Pipeline: REST API Plugin
- Pipeline: Declarative
- Pipeline: Declarative Extension Points API
- Pipeline: Model API
- Git
- Blue Ocean
- Generic Webhook Trigger
- Sonar(SonarQube Scanner for Jenkins)
- generic-event

可选的插件

- Kubernetes 插件：如果希望流水线运行在 K8s 集群上，则需要安装此插件
- Docker 插件：如果希望流水线以容器的形式与物理机隔离运行，则需要安装此插件
- Chinese 插件：这是中文社区提供的汉化插件

配置节点

有关物理机节点的准备工作，请参阅 Using Jenkins agents。

Jenkins 配置

EventDispatcher

首先确保 generic-event 插件正常安装并且已经启用。

查看 amamba-devops-server：

```
# 获取 amamba-devops-server 的 svc
kubectl get svc amamba-devops-server -n amamba-system

NAME                     TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
          AGE
amamba-devops-server     ClusterIP   10.233.45.104   <none>        80/TCP,15090/TCP   230
d
```

要确保 Jenkins 可以正常和 amamba-devops-server 通信，通过 NodePort 或 LB 暴露出来，

然后 host 填对应 IP。

!!! note
此操作不太安全，这个服务不适合直接暴露出去。

将 amamba-devops-server 的地址填入到 Jenkins 系统管理配置的 Event Receiver 输入框中，

可参考以下格式。

http: //<hostIP:Port>/apis/internel.amamba.io/devops/pipeline/v1alpha1/webhooks/jenkins

SonarQube

有关 Sonarqube 的配置信息，参阅 Jenkins 接入 Sonarqube。

邮件配置

获取 Ghippo 的邮件配置。

kubectl get ghippoconfig smtp -n ghippo-system -o yaml

下面是一个简单的样例：

```yaml
apiVersion: ghippo.io/v1alpha1
kind: GhippoConfig
metadata:
  labels:
    k8slens-edit-resource-version: v1alpha1
  name: smtp
spec:
  from: test@163.com
  host: smtp.163.com
  password: test
  port: 25
  ssl: false
  starttls: false
  user: test@163.com
```

在 Jenkins 里配置邮件服务。

进入系统管理系统配置，输入管理员邮件地址 test@163.com，设置 SMTP 邮件服务器

smtp.163.com，选择 SMTP 认证填入用户名和密码，设置 SMTP 端口，进入用户设置页面，

设置邮件地址 test@163.com。

使用流水线发布多云资源到多云编排模块

本文介绍如何通过流水线来发布多云资源。

环境准备

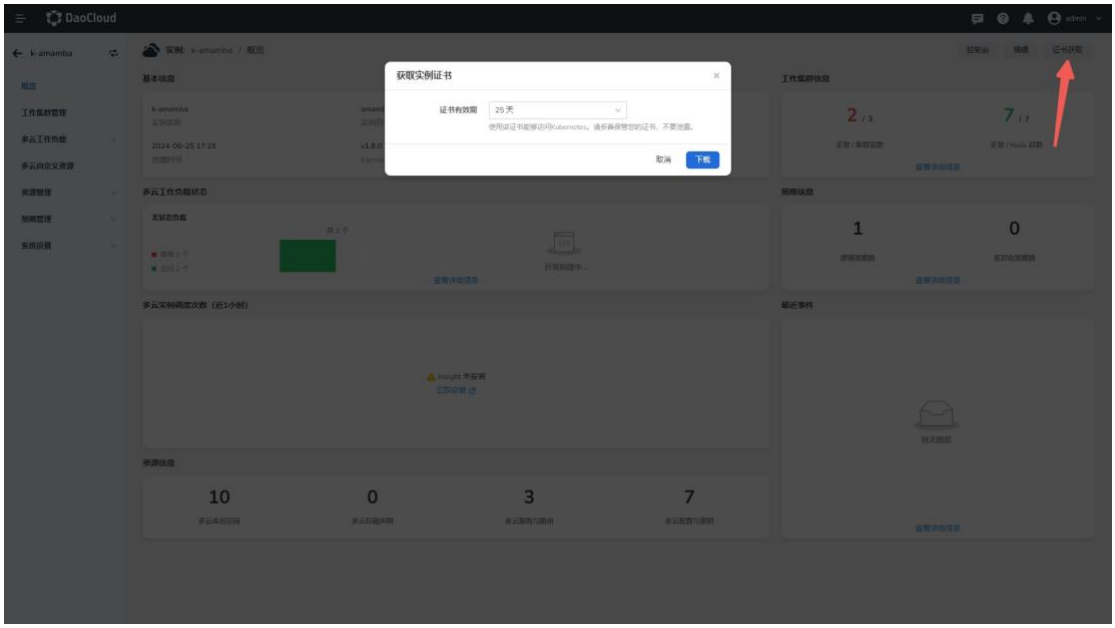
代码仓库准备

1. 需要一个文件夹存放需部署资源的 YAML，并且包含多云编排的 PropagationPolicy（在 resourceSelectors 中把期望的资源添加进去）
2. 一个 Dockerfile 用于构建镜像
3. 一个 Token 用于在流水线中访问这个代码仓库（或者设置为公开）

多云编排准备

1. 创建一个多云实例，获取到对应实例的 kubeconfig 证书，并把证书添加到应用工作台的流水线凭证当中
2. 在多云实例中创建一个多云命名空间

下图是获取证书的页面：



getkarishipconfig.jpg

应用工作台准备

1. 确保多云实例的 kubeconfig 证书被创建到应用工作台的流水线凭证中，并且确保已经同步到 Jenkins
2. 确保代码仓库的 token 已经被创建到应用工作台的流水线凭证中，并且确保已经同步到 Jenkins （如果代码仓库为公开仓库则不需要，如果为集成进来的 Gitlab，只需要 Gitlab 的连接状态正常即可）
3. 现阶段应用工作台无法获取多云编排创建的多云实例以及对应的多云命名空间(在创建流水线时），现在只能手动编写 JenkinsFile

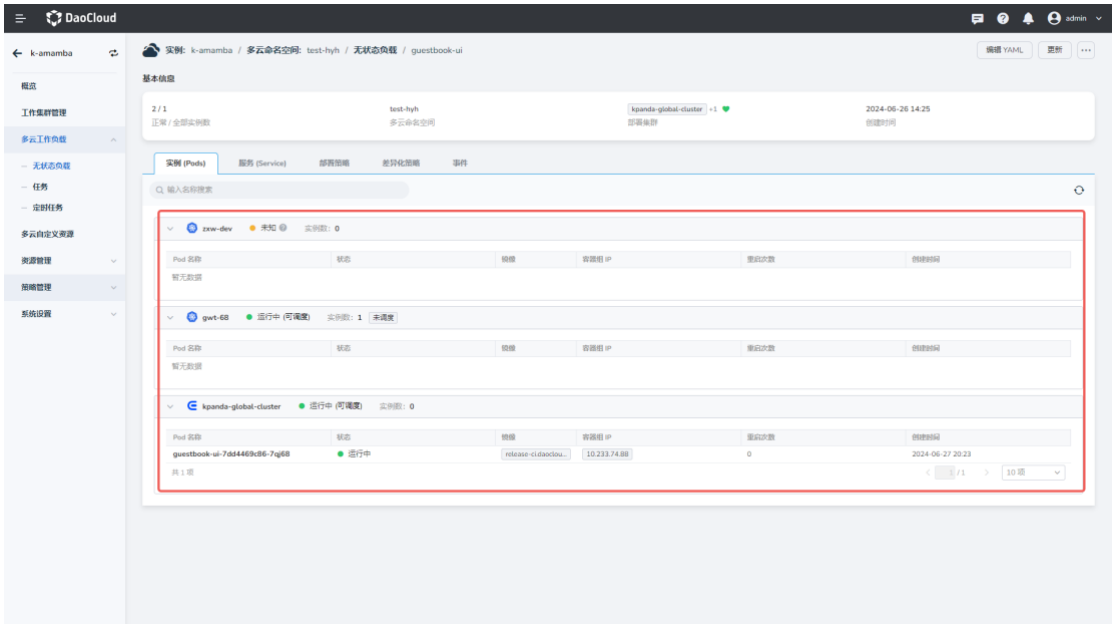
创建流水线

请参阅本文所用的 JenkinsFile 开源仓库。

根据仓库内的 pipelines/guestbook-karisip.jenkinsfile 创建对应的流水线，修改对应的参数后运行。对于部署到多云编排的流水线，本质上与其他流水线相同，关键在于应用工作台无法

获取到对应的多云实例和多云命名空间，所以只能根据多云编排提供的 kubeconfig 证书使用 kubectl 来部署资源。

运行结果



kairshipresult

基于云原生网关（contour）的灰度发布实践

本文将通过以下步骤来进行基于云原生网关的灰度发布的最佳实践，将分为以下几步：

1. 创建应用资源，用于灰度发布的发布对象
2. 创建微服务引擎相关的网关资源
3. 创建基于云原生网关的灰度发布任务
4. 观察发布任务的发布情况

前提条件

- 请在 argo-rollouts 添加 contour-plugin 配置，参考文档部署 argo-rollouts

- 当前平台部署了微服务引擎模块

创建应用资源

登录 DCE 5.0 之后，点击左上角的 ≡ 打开导航栏，依次选择 **应用工作台** -> **向导** -> **基于容器镜像** ，在目标集群/命名空间中创建应用 rollouts-demo。

注意事项:

- 容器/服务端口：8080
- 容器镜像：argoproj/rollouts-demo: red（镜像 tag 也可以是 blue/green/orange）

app01

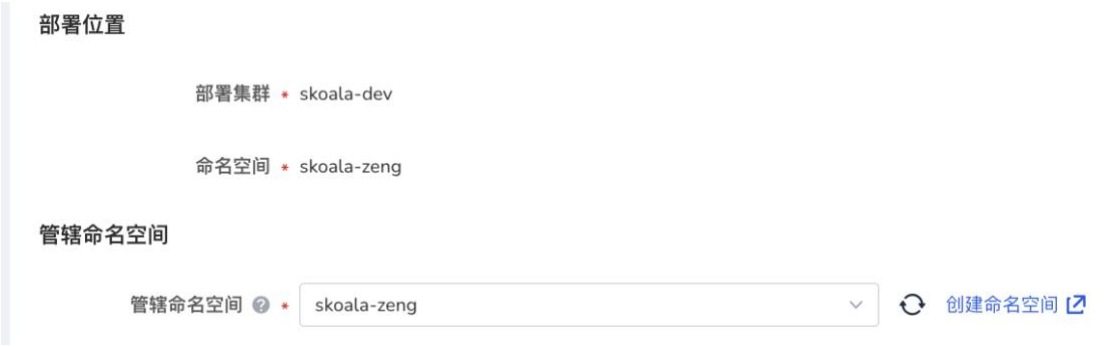
微服务引擎网关资源

创建网关

参考创建网关文档。

注意事项:

- 网关的管辖命名空间需要包含灰度发布服务所在的命名空间



skoala01

创建域名

进入网关详情界面，在域名管理中创建域名。

注意事项：

输入一个简单的 HTTP 域名，比如 gateway.canary

skoala02

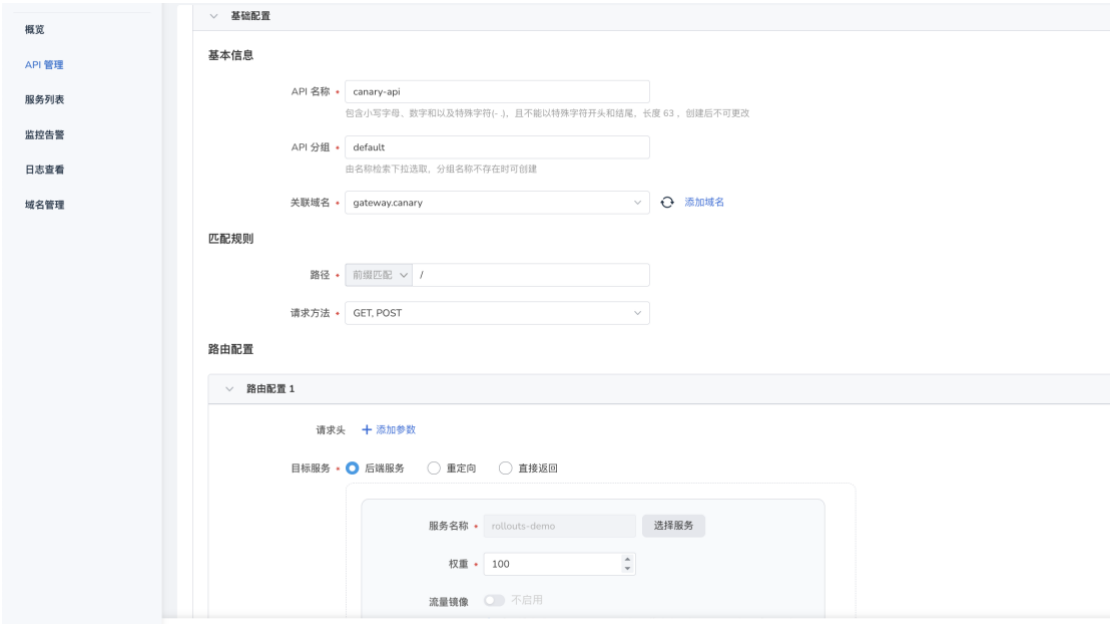
创建 API

进入网关详情界面，在 API 管理中创建 API。

注意事项：

- 域名选择 gateway.canary
- 路径匹配规则选择前缀匹配，路径填写 /

- 请求方法选择 GET 和 POST，当然也可以全选
- 路由配置选择后端服务中的自动发现服务 rollouts-demo



skoala03

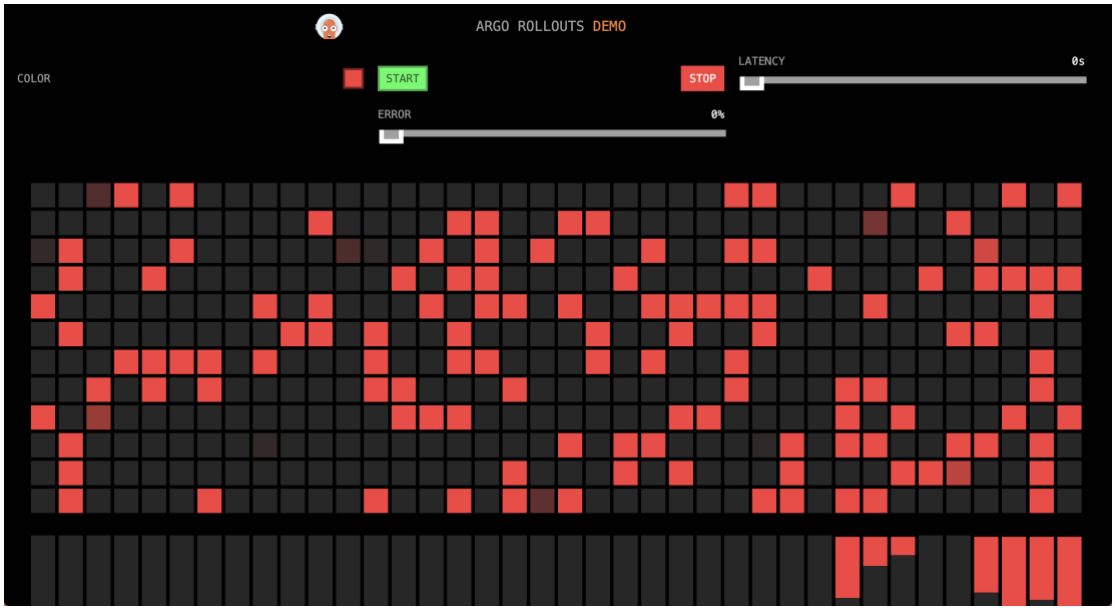
通过网关访问灰度发布的 Demo 应用

本地配置域名解析，将 gateway.canary 解析到网关地址上，其中网关地址可在网关详情界面概览模块的网络信息区域获取。

```
$ cat /etc/hosts | grep gateway.canary
10.6.222.21 gateway.canary
```

在浏览器中输入 <http: //gateway.canary:30000/> ，可以看到如下界面 (流量都是指向 red) :

- 需要关闭网络代理，否则访问可能会 502 报错
- 30000 端口是网关的 HTTP 端口，使用的是 HTTP 域名，所以使用 HTTP 端口

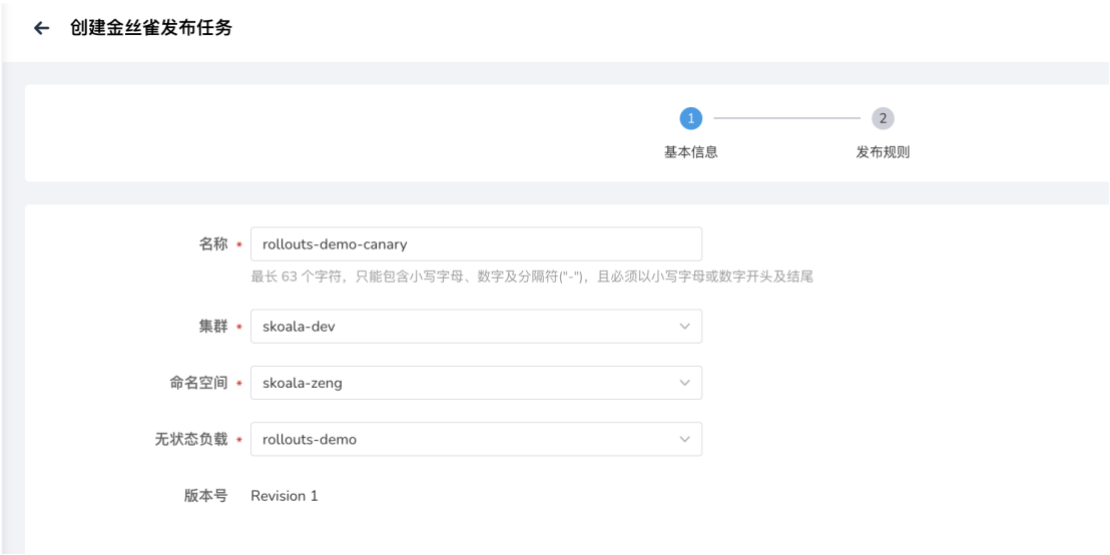


skoala05

创建金丝雀灰度发布任务

在应用工作台创建金丝雀灰度发布任务。

1. 第一步的基本信息选择目标位置的 rollouts-demo 应用

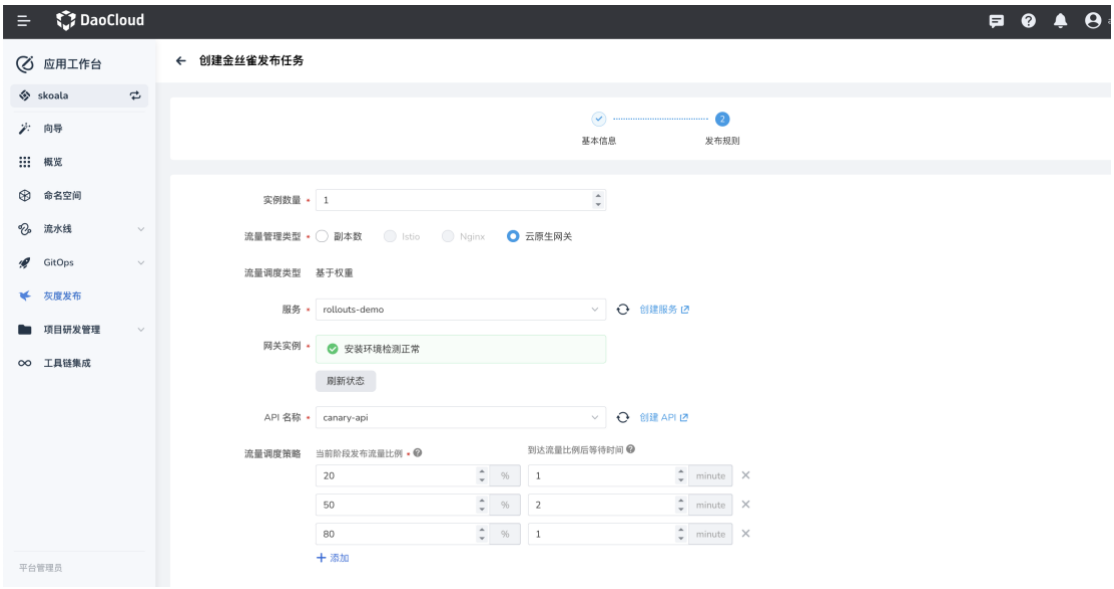


contour-rollout

2. 第二步的发布规则如下：

   - 流量管理类型选择云原生网关(该选项的是否可选取决于微服务引擎是否安装)
   - 服务选择基本信息中选择的无状态负载对应的服务（即灰度发布的服务）

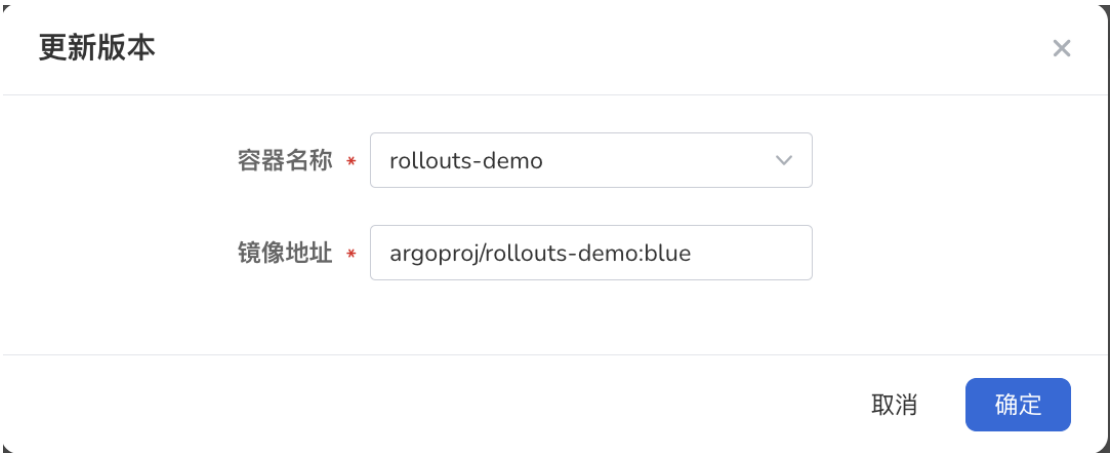
- 网关实例是由基本信息中目标位置是否是由某个网关管辖决定的
- API 名称选择对应网关中绑定了灰度发布服务的 API
- 流量调度策略参考如下



contour-rollout

更新发布任务，观察发布情况

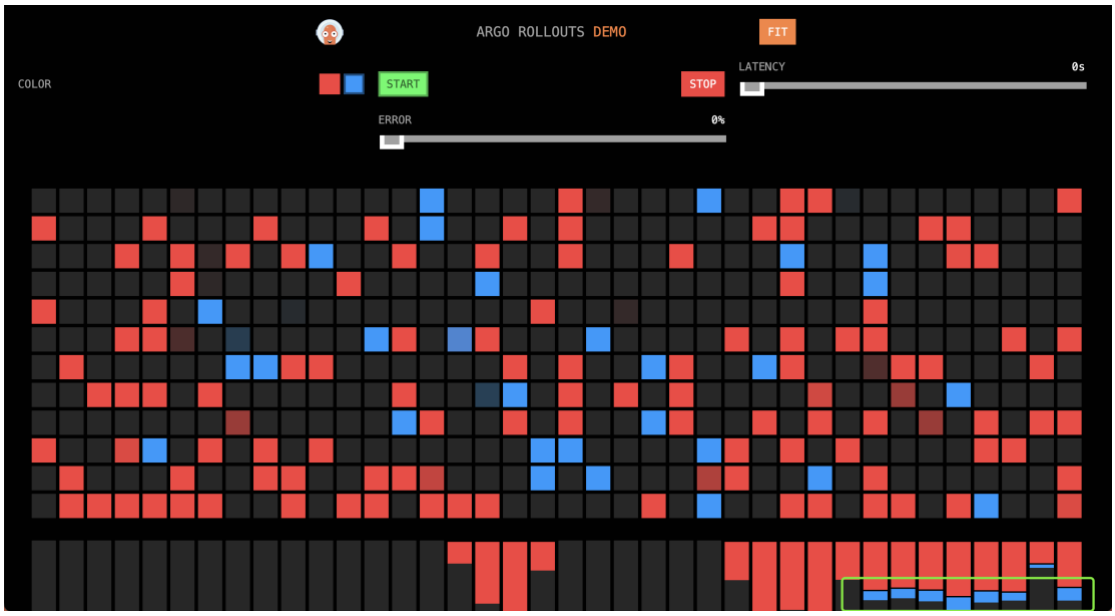
需要在发布界面中更新版本，镜像替换为：argoproj/rollouts-demo: blue。 版本更新后，在浏览器中访问 <http: //gateway.canary:30000/> ，然后观察流量统计：



contour-rollout

第一阶段

即第一分钟，流量比 red: blue 大概是 4:1



step01

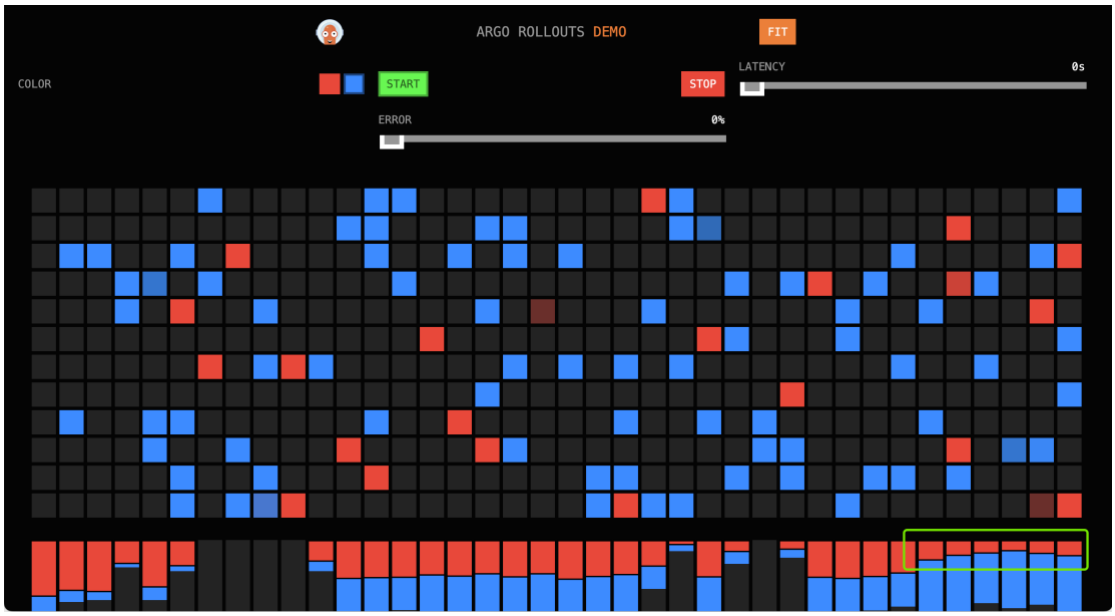
第二阶段

即第二三分钟，流量比 red: blue 大概是 1:1

step02

第三阶段

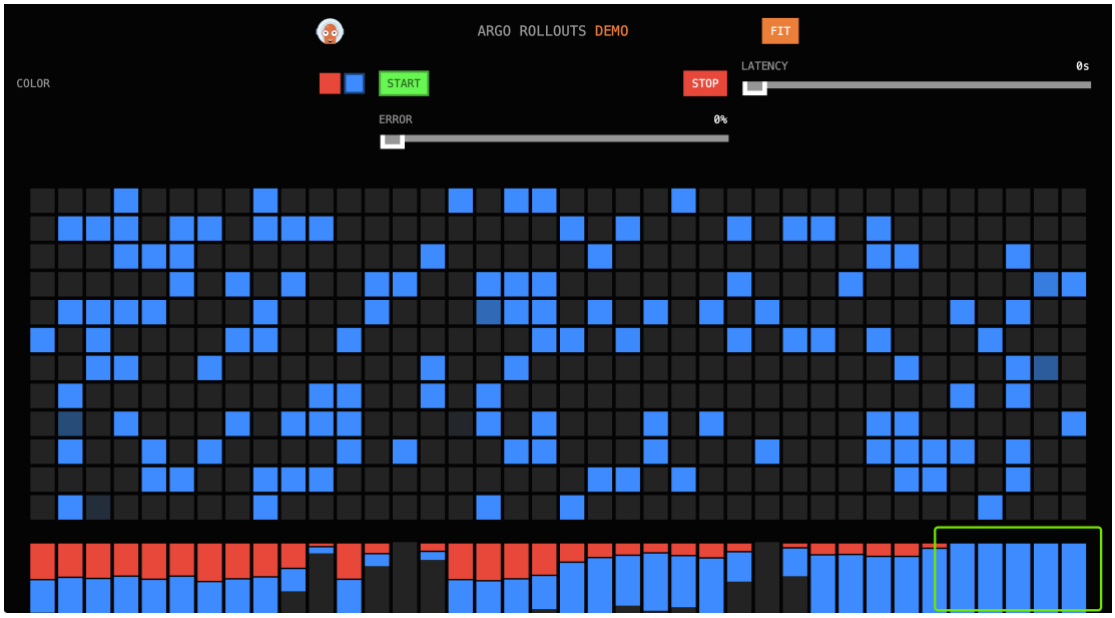
即第四分钟，流量比 red: blue 大概是 1:4



step03

最后阶段

即发布完成，全部都是 blue 的流量



step04