Menu

GitOps

介绍

介绍

GitOps 应用场景

GitOps 优势

Alauda Container Platform (ACP) GitOps 优势

安装

安装 **Alauda Build of Argo CD**

前提条件

操作步骤

安装 **Alauda Container Platform GitOps**

前提条件

安装 Alauda Container Platform GitOps 集群插件

升级

升级 **Alauda Container Platform GitOps**

前提条件

升级 Alauda Container Platform GitOps 集群插件

架构

架构

问题描述

意译结果

weight: 20 i18n: title: en: Architecture zh: 架构

GitOps 和 Argo CD

GitOps 架构

Alauda 容器平台 GitOps 架构

核心概念

GitOps 介绍

介绍

核心原则

优势

常用 GitOps 工具

Argo CD 核心概念

Alauda Container Platform GitOps 核心概念

功能指南

创建 GitOps 应用

GitOps 可观测

实用指南

通过 Argo CD Dashboard 集成代码仓库

使用场景

前置条件

操作步骤

操作结果

通过 Argo CD dashboard 创建 Argo CD Application

前提条件

操作步骤

通过平台创建 **Argo CD Application**

用例

前提条件

步骤

如何获取 **Argo CD** 访问信息

使用场景

如何获取通过平台部署的 GitOps 集群插件的 Argo CD 访问信息？

如何获取通过 Argo CD Operator 部署的 Argo CD 访问信息？

故障排查

故障排查

介绍

GitOps 是一种现代的持续交付和运维方法，通过将 Git 作为“单一可信来源”（Single Source of Truth，SSOT），用于定义和管理基础设施、应用程序配置以及部署工作流。通过将应用程序代码、配置文件和基础设施即代码（Infrastructure as Code，IaC）的定义集中在 Git 仓库中，GitOps 允许对整个软件交付生命周期进行全面的版本控制和自动化管理。在这一模式下，开发团队和运维团队可以在软件开发、测试和部署阶段通过 Git 的强大分支管理、代码审查和合并请求机制无缝协作。当代码或配置的更改被推送到 Git 仓库时，自动化工具会检测到这些更新并启动一连串自动化的流程，包括构建、测试和部署。这一工作流程促进了软件的持续交付和持续部署（CI/CD），确保了快速且可靠的发布。

目录

GitOps 应用场景

- 容器化应用的持续交付：在 Kubernetes 生态系统中，GitOps 擅长于管理容器化应用的部署、更新和回滚。开发人员将应用代码和 Kubernetes 配置文件提交到 Git 仓库，GitOps 工具随后自动将这些应用部署到 Kubernetes 集群，并与任何配置文件的修改保持同步。
- 多环境管理：GitOps 简化了在不同环境（如开发、测试、预发布和生产）之间的基础设施和应用配置管理。通过战略性分支和环境标记，它保持配置的一致性，同时满足每个环境所需的定制化。
- 微服务架构的运维：在微服务架构中，GitOps 帮助团队高效管理多个微服务的部署和更新。每个微服务的代码和配置可以独立存储在 Git 仓库中，GitOps 工具根据微服务依赖关系和更

新策略自动进行部署和更新，从而确保系统的稳定性。

- 基础设施即代码（**IaC**）的管理：GitOps 无缝集成了 Terraform 和 Ansible 等 IaC 工具，用于管理云基础设施、服务器配置和网络资源。将 IaC 配置文件存储在 Git 仓库中实现版本控制和自动化基础设施部署，增强了可管理性和重复性。
- 跨团队协作与代码共享：在大型组织中，多个团队通常需要共享代码和配置。GitOps 提供了一个统一的平台，使团队能够通过 Git 仓库协作开发、共享代码和管理配置，从而提升协作效率和代码复用。

GitOps 优势

- 加速协作与交付
- 快速回滚与恢复
- 多环境管理
- 增强安全性与合规性

GitOps 优势详细介绍

Alauda Container Platform (ACP) GitOps 优势

> Alauda Container Platform (ACP) GitOps （简称：Alauda GitOps）

- 企业级 Argo CD Operator。
- Argo CD Operator 安全服务。
- 可视化的 GitOps 应用多环境分发管理。
- 可视化的 GitOps 应用运维与维护。
- 可视化的 GitOps 集群配置管理。
- 与平台所有产品集成的闭环 GitOps 应用管理。

Alauda GitOps 优势详细介绍

Menu

安装

安装 Alauda Build of Argo CD

前提条件

操作步骤

安装 Alauda Container Platform GitOps

前提条件

安装 Alauda Container Platform GitOps 集群插件

安装 Alauda Build of Argo CD

目录

前提条件

1. 下载与您的平台架构相对应的**Alauda Build of Argo CD** Operator安装包。
2. 上传通过上传包机制上传安装包。

操作步骤

> 安装到您希望使用GitOps功能的集群。

安装 Alauda Build of Argo CD Operator

1. 登录，进入平台管理页面。
2. 点击应用商店 > **OperatorHub**，进入**OperatorHub**页面。
3. 找到**Alauda Build of Argo CD** Operator，点击安装，进入安装 **Argo CD**页面。

配置参数：

参数	推荐配置
频道	默认频道为`alpha`。
安装模式	`集群`：集群中的所有命名空间共享一个Operator实例进行创建和管理，从而降低资源使用量。
命名空间	选择`推荐命名空间`：如果不存在则自动创建。
升级策略	`自动`：当有新版本可用时，OperatorHub将自动将Operator升级到最新版本。

1. 建议使用默认配置；只需点击安装以完成**Alauda Build of Argo CD** Operator的安装。

创建 Argo CD 实例

1. 点击应用商店 > **OperatorHub**。
2. 找到**Alauda Build of Argo CD** Operator，点击进入**Argo CD**详细信息页面。
3. 点击所有实例，
4. 点击创建实例，选择**Argo CD**实例卡片。
5. 点击创建实例。

> **INFO**
>
> 在配置实例参数页面，除非有特定要求，请使用默认配置。 注意：如果global集群不可高可用（例如仅有一个控制节点），则在创建实例时请切换至YAML视图，并将ha.enabled字段值设置为false。

1. 点击创建。

创建 AppProject 实例

> **INFO**

提示：如果您不需要使用平台管理的集群配置管理功能，则无需执行以下步骤。

1. 找到**Alauda Build of Argo CD** Operator，点击进入**Alauda Argo CD**详细信息页面。

2. 点击所有实例，创建实例，选择**AppProject**实例卡片。

3. 切换到YAML视图，并用以下代码覆盖界面上现有的YAML内容。

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: cpaas-system
  namespace: argocd
spec:
  clusterResourceWhitelist:
  - group: '*'
    kind: '*'
  destinations:
  - namespace: '*'
    server: '*'
  sourceRepos:
  - '*'
```

4. 点击创建。

完成上述操作后，您已成功安装Alauda Build of Argo CD。立即通过Argo CD仪表板创建应用程序，开始您的GitOps旅程。

安装 Alauda Container Platform GitOps

目录

前提条件

1. 下载 与平台架构相对应的 **Alauda Container Platform GitOps** 集群插件安装包。
2. 上传 安装包，使用上传包机制。
3. 安装 安装包到 `global` 集群，使用集群插件机制。

> **INFO**
>
> 上传包： 平台管理 > 市场 > 上传包 页面。 点击右侧的 帮助文档 获取如何将集群插件发布到 `global` 集群的说明。详细信息，请参阅 CLI。

安装 Alauda Container Platform GitOps 集群插件

约束与限制

- 仅支持在 `global` 集群中安装。
- 插件安装后，`argocd-operator` 中的 ArgoCD 实例将限制操作。

操作步骤

1. 登录，进入 平台管理 页面。
2. 点击 市场 > 集群插件 进入 集群插件 列表页面。
3. 找到 **GitOps** 集群插件，点击 安装，并跳转到 安装 **GitOps** 插件 页面。
4. 建议使用推荐的默认配置；直接点击 安装 完成 **Alauda Container Platform GitOps** 集群插件的安装。

参数说明如下：

参数	说明
原生 **Argo CD UI**	选择是否按需访问 Argo CD 提供的仪表板。此界面包括监控、仓库管理和设置等功能，可用于管理和监控已创建的应用程序。
单点登录	建议启用 SSO，这样可以使用平台账号信息快速访问 Argo CD 原生 UI，增强登录体验，同时提高安全性和便利性。 注意：单点登录功能需要启用 Argo CD 原生 UI 特性。 • 仅支持通过 HTTPS 访问；如果使用 HTTP 访问，则 SSO 将失效。 • 启用 SSO 后，使用访问地址打开 Argo CD 登录界面，在界面中单击用 **OIDC** 登录 按钮即可实现一键登录 Argo CD 原生 UI。
访问地址	推荐：此地址基于访问 Argo CD 仪表板的平台地址动态生成，无需手动输入。
账号	用于登录并访问 Argo CD 原生 UI 的账号。
密码	开启访问 Argo CD 原生 UI 后，您可以在 global 集群的 CLI 工具中执行以下命令获取。 获取 Argo CD 访问信息
资源配额	平台的最低要求与建议如下：

参数	说明
	• 最低：CPU 请求值不得低于 100 m，内存请求值不得低于 250 Mi，且请求值不得超过限制值。 • 推荐：CPU 请求值应不少于 250 m，内存请求值应不少于 500 Mi；CPU 限制值应不少于 2 核，内存限制值应不少于 2 Gi。

验证

1. 在 平台管理 页面，左侧导航的 集群 部分会显示 配置 入口，您可以使用集群配置管理功能。

2. 访问 **Container Platform**，左侧导航将显示 **GitOps** 应用 入口，您可以创建 GitOps 应用，立即体验 通过网页控制台创建 Argo CD 应用。

Menu

升级

升级 **Alauda Container Platform GitOps**

前提条件

升级 Alauda Container Platform GitOps 集群插件

升级 Alauda Container Platform GitOps

目录

前提条件

1. 下载 与您的平台架构对应的 **Alauda Container Platform GitOps** 集群插件安装包。
2. 使用 Upload Packages 机制上传 **Alauda Container Platform GitOps** 安装包。
3. 使用集群插件机制将 **Alauda Container Platform GitOps** 集群插件安装到 `global` 集群。

INFO

Upload Packages： 进入 平台管理 > **Marketplace** > 上传软件包 页面。 点击右侧的 帮助文档 获取如何将集群插件发布到 `global` 集群的说明。更多详情请参考 CLI。

升级 Alauda Container Platform GitOps 集群插件

约束与限制

- 仅支持在 `global` 集群中升级。

操作步骤

1. 登录，进入 平台管理 页面。
2. 点击 集群 > 集群 > **global** > 功能组件，进入组件列表页面。
3. 点击 升级 按钮，在 确认组件升级 页面选择新的 **Alauda Container Platform GitOps** 版本作为升级目标。
4. 再次点击 升级，并在弹窗中点击 升级 确认升级操作。
5. （仅当从 `ACP 3.16.0` 升级时需要）返回 `global` 集群页面，点击 操作 > **CLI** 工具，进入 CLI 窗口。
6. 在 CLI 窗口输入 `kubectl delete cm -n argocd argocd-redis-ha-configmap`，以重新创建 `argocd-redis-ha-configmap`。
7. 在 CLI 窗口输入 `kubectl get cm -n argocd argocd-redis-ha-configmap`，确认 configmap 已创建。

验证

1. 在 平台管理 页面，左侧导航的 集群 区域将显示 配置 条目，您可以使用集群配置管理功能。
2. 访问 **Container Platform**，左侧导航将显示 **GitOps** 应用 条目，您可以创建 GitOps 应用，立即体验通过 Web 控制台创建 Argo CD 应用。

目录

问题描述

1. 在第一段中，"Alauda Container Platform GitOps" 翻译为 "ACP GitOps"，未能完整保留原文的名称，可能导致读者对产品的识别产生困惑。

2. 在第二段中，"GitOps" 和 "Argo CD" 的翻译未保持一致性，前者翻译为 "GitOps"，后者则直接使用英文，建议统一处理。

3. "GitOps 与传统应用管理方法之间的主要区别在于" 这句话的表达略显生硬，可以考虑更流畅的表达方式。

4. "确保环境符合预期，防止配置漂移，并在发生故障时实现快速恢复。" 这句话的结构较为复杂，可能导致理解困难。

意译结果

weight: 20 i18n: title: en: Architecture zh: 架构

架构

GitOps 和 Argo CD

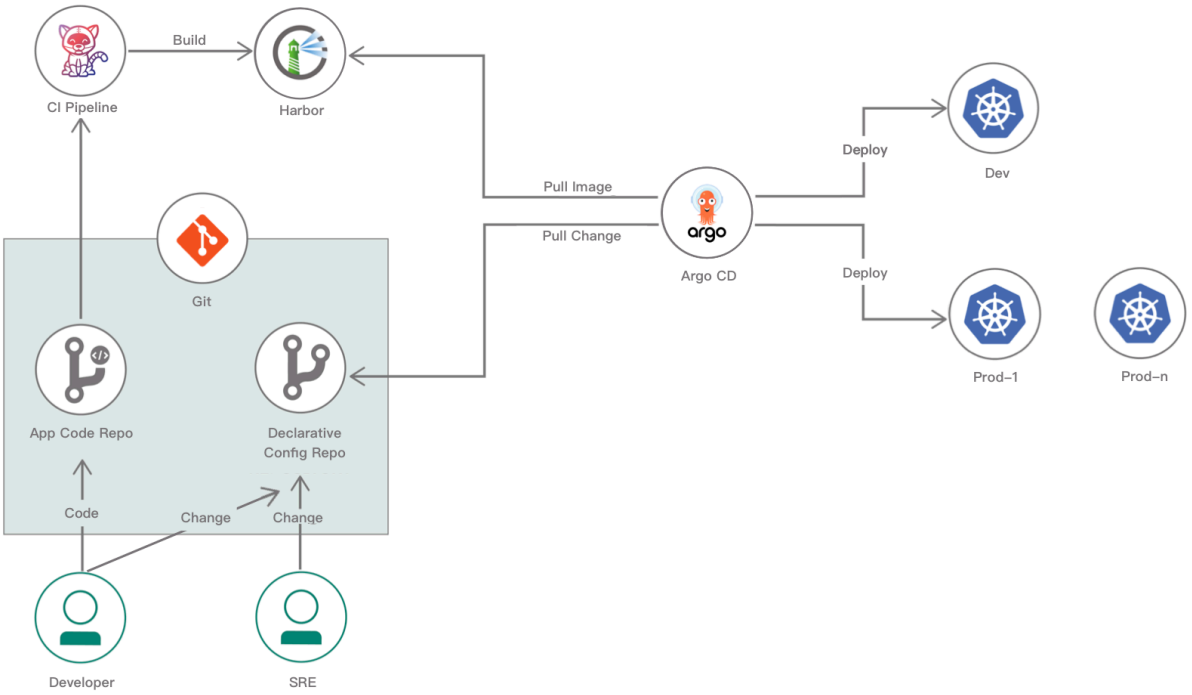
GitOps 是一种现代的持续交付和运营理论，而 Argo CD 是一款强大的工具，通过监控 Git 仓库中的配置文件并自动将其同步到目标环境来实现 GitOps。这种方法通过将整个交付过程纳入 Git 版本控制系统，提高了软件交付的速度、可靠性和安全性。

Alauda 容器平台 **GitOps** 基于 Argo CD，使用 Git 仓库作为唯一可信源，存储应用程序、基础设施配置和其他文件，以便快速准确地分发和部署到一个或多个 Kubernetes 集群。

GitOps 架构

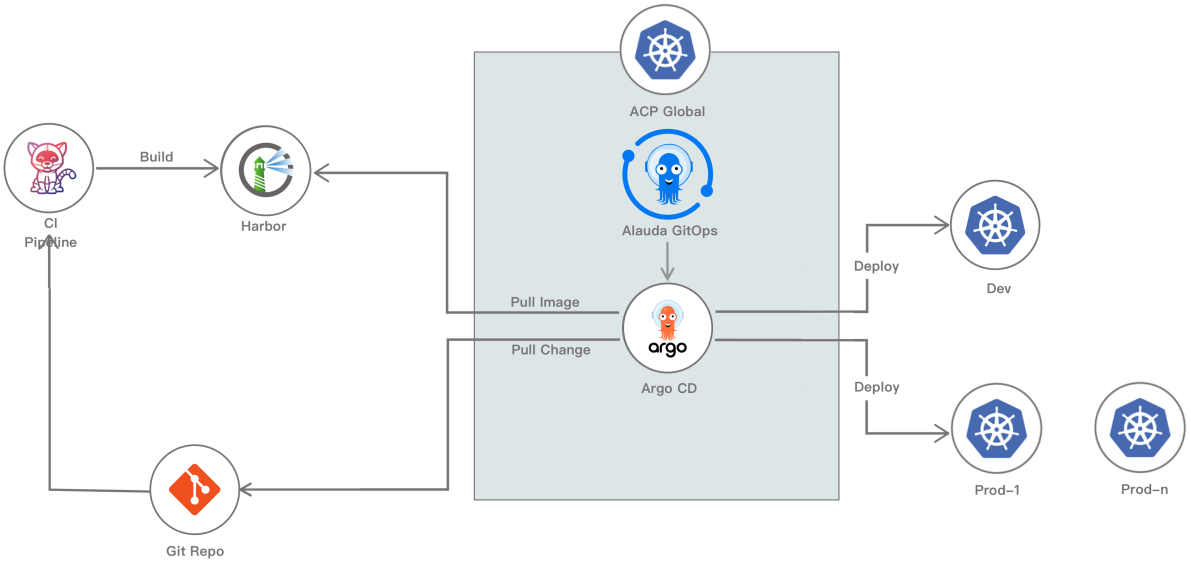
GitOps 与传统应用管理方法之间的主要区别在于：

- GitOps 通过在 Git 上维护应用配置仓库来控制运行环境，而不是直接操控运行环境。
- Argo CD 持续拉取该仓库，并纠正运行环境与应用配置仓库之间的差异，确保环境符合预期，防止配置漂移，并在发生故障时能够快速恢复。



Alauda 容器平台 GitOps 架构

Alauda 容器平台 **GitOps** 作为集群插件安装在 `global` 集群上，并利用 Argo CD 实现多个业务集群的应用分发和基础设施提供。


ACP Global
Alauda GitOps
CI
Pipeline
Build
Harbor
Pull Image
Pull Change
Argo CD
Deploy
Dev
Deploy
Prod-1
Prod-n
Git Repo

Menu

核心概念

GitOps 介绍

GitOps 介绍

介绍

核心原则

优势

常用 GitOps 工具

Argo CD 核心概念

Argo CD Introduction

总结 Application 和 ApplicationSet 的区别

Argo CD 同步状态

参考资料

Application 概念

介绍

Application 适用场景

Application 示例

参考资料

ApplicationSet 概念

介绍

适用场景

示例

参考资料

Tool 概念

介绍

支持的工具

开发工作流程

特性对比

参考资料

Helm 概念

介绍

Helm 的核心概念

优势

使用场景

Kustomize 概念

介绍

Kustomize 的核心概念

优势

使用场景

Directory 概念

问题描述

意译结果

sourceSHA: e8d7a40d5fe32bb85f84d87db6e2c3843acb713cf368ede32be799a98a9547b0
weight: 70

介绍

优势

使用场景

Sync 概念

Sync 概述

Sync 状态概述

Sync 操作状态概述

刷新概述

参考资料

Health 概念

介绍

健康范围

参考资料

Alauda Container Platform GitOps 核心概念

Alauda Container Platform GitOps 概念

为什么选择 Argo CD？

优势

Alauda Container Platform GitOps 的同步及健康检查

同步状态说明

健康状态说明

识别规则

GitOps

目录

介绍

GitOps 是使用 Git 仓库作为基础设施和应用程序配置的权威源的实践。所有操作变更都通过 Git 进行版本控制、自动化和审计。它依赖于存储在 Git 中的声明式配置，任何修改必须提交以触发自动化部署流程。

核心原则

- 声明式配置：GitOps 基本上需要声明式工具，将 Git 视为单一的真实来源。这使得在 Kubernetes 集群中实现一致的应用程序部署，并在故障情况下支持平台无关的恢复。
- 版本化和不可变状态：基础设施和应用程序版本直接映射到 Git 提交。通过 `git revert` 执行回滚，确保不可变的版本历史。
- 自动化协调：合并的声明式状态会自动应用到集群中。这消除了手动干预，防止了人为错误，并支持在部署工作流中的安全审批。
- 自我修复：控制器（例如 Argo CD）持续地将集群状态与 Git 定义的状态进行协调，从而实现自主的系统恢复。

优势

- 加速团队协作与交付：将基础设施、配置和目标状态的声明式定义存储在 Git 中，能够实现自动化部署。团队在验证后可实现一键环境配置，简化协作与交付过程。
- 快速回滚与恢复：利用 Git 的版本控制，异常情况触发即时回滚。GitOps 控制器通过自动化协调确保自我修复。
- 多环境治理：将 Git 作为唯一真实来源，结合配置重叠，能够在混合/多云环境中实现精确的批量部署。
- 增强的安全性与合规性：Git 的 RBAC、审计日志、分支保护和加密确保敏感配置的安全，确保遵循合规要求。

常用 **GitOps** 工具

- **Argo CD**：一个针对 Kubernetes 的声明式 GitOps 工具，用于定义、版本控制和自动化应用程序生命周期，具有审计功能。
- **Flux**：一个轻量级的 Kubernetes GitOps 操作工具，持续将 Git 仓库同步到集群。
- **Jenkins X**：一个支持 GitOps 集成的 CI/CD 平台，用于自动化管道和基于 Git 的部署。

Menu

Argo CD 核心概念

Argo CD Introduction

总结 Application 和 ApplicationSet 的区别

Argo CD 同步状态

参考资料

Application 概念

介绍

Application 适用场景

Application 示例

参考资料

ApplicationSet 概念

介绍

适用场景

示例

参考资料

Tool 概念

介绍

支持的工具

开发工作流程

特性对比

参考资料

Helm 概念

介绍

Helm 的核心概念

优势

使用场景

Kustomize 概念

介绍

Kustomize 的核心概念

优势

使用场景

Directory 概念

问题描述

意译结果

sourceSHA: e8d7a40d5fe32bb85f84d87db6e2c3843acb713cf368ede32be799a98a9547b0
weight: 70

介绍

优势

使用场景

Sync 概念

Sync 概述

Sync 状态概述

Sync 操作状态概述

刷新概述

参考资料

Health 概念

介绍

健康范围

参考资料

介绍

Argo CD 是一款非常流行的开源 GitOps 工具。使用 Argo CD 需了解如下核心概念：

1. Application：一组由清单定义的 Kubernetes 资源。这是一个自定义资源定义（CRD）。深入理解 Application
2. ApplicationSet：一个支持 ApplicationSet CRD 的 Kubernetes 控制器，能够从单个模板批量生成 Applications。可以将其视为一个根据参数创建实例的 Application 工厂。深入理解 ApplicationSet
3. Tool：指定 Application 源的配置管理工具（例如 Kustomize、Helm）。深入理解 Tool
4. Sync：将应用程序的实时状态与其期望状态进行协调的过程（例如，将更改应用于 Kubernetes 集群）。深入理解 Sync
5. Health：指示应用程序的操作状态，包括就绪性和服务请求的能力。深入理解 Health

目录

总结 **Application** 和 **ApplicationSet** 的区别

属性	**Application**	**ApplicationSet**
定义	单个应用程序的部署	用于生成多个 Application 实例的模板
配置	静态 YAML 定义	动态参数驱动的模板生成

属性	**Application**	**ApplicationSet**
部署	单个应用程序	多个相似的应用程序
适用场景	简单的单环境部署	复杂的多环境/集群部署，需要使用参数化实例
核心概念	Git 仓库、目标集群、部署策略	生成器、模板、参数、占位符
与 Argo CD 的关系	基本的部署单元	高级的批量管理层

Argo CD 同步状态

同步状态	说明
Synced	Application 的实时状态与期望状态完全一致。
OutOfSync	实时状态与期望状态不一致；需要进行同步。
Syncing	正在进行主动同步；实时状态正在向期望状态靠拢。

参考资料

- Argo CD 官方文档↗

Application

目录

介绍

Application 是一组 Kubernetes 资源，由清单定义。这是一个自定义资源定义（CRD）。

Application 适用场景

- 单组件部署：使用 Application CRD 声明性地管理单个命名空间内原子工作负载的部署。
- 静态配置：适合具有确定性清单的应用程序，这些清单不需要动态模板化或多环境变体。
- 单集群部署：通过 GitOps 工作流程将部署定向到单独的 Kubernetes 集群。

Application 示例

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
  labels:
    name: guestbook
spec:
  project: default

  source:
    repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: HEAD
    path: guestbook

    chart: chart-name
    helm:
      passCredentials: false
      parameters:
      - name: "nginx-ingress.controller.service.annotations.external-dns\\.al
        value: mydomain.example.com
      - name: "ingress.annotations.kubernetes\\.io/tls-acme"
        value: "true"
        forceString: true

      fileParameters:
      - name: config
        path: files/config.json

      releaseName: guestbook

      valueFiles:
      - values-prod.yaml

      ignoreMissingValueFiles: false

      values: |
        ingress:
          enabled: true
          path: /
          hosts:
```

```
            - mydomain.example.com
          annotations:
            kubernetes.io/ingress.class: nginx
            kubernetes.io/tls-acme: "true"
          labels: {}
          tls:
            - secretName: mydomain-tls
              hosts:
                - mydomain.example.com

      valuesObject:
        ingress:
          enabled: true
          path: /
          hosts:
            - mydomain.example.com
          annotations:
            kubernetes.io/ingress.class: nginx
            kubernetes.io/tls-acme: "true"
          labels: {}
          tls:
            - secretName: mydomain-tls
              hosts:
                - mydomain.example.com

      skipCrds: false
      skipSchemaValidation: false
      version: v2
      kubeVersion: 1.30.0
      apiVersions:
        - traefik.io/v1alpha1/TLSOption
        - v1/Service
      namespace: custom-namespace

    kustomize:
      version: v3.5.4
      namePrefix: prod-
      nameSuffix: -some-suffix
      commonLabels:
        foo: bar
      commonAnnotations:
        beep: boop-${ARGOCD_APP_REVISION}
      commonAnnotationsEnvsubst: true
      forceCommonLabels: false
```

```
    forceCommonAnnotations: false
    images:
    - gcr.io/heptio-images/ks-guestbook-demo:0.2
    - my-app=gcr.io/my-repo/my-app:0.1
    namespace: custom-namespace
    replicas:
    - name: kustomize-guestbook-ui
      count: 4
    components:
      - ../component
    patches:
      - target:
          kind: Deployment
          name: guestbook-ui
        patch: |-
          - op: add
            path: /spec/template/spec/nodeSelector/
            value:
              env: "pro"

    kubeVersion: 1.30.0
    apiVersions:
      - traefik.io/v1alpha1/TLSOption
      - v1/Service

  directory:
    recurse: true
    jsonnet:
      extVars:
      - name: foo
        value: bar
      - code: true
        name: baz
        value: "true"
      tlas:
      - code: false
        name: foo
        value: bar
    exclude: 'config.yaml'
    include: '*.yaml'

  plugin:
    name: mypluginname
    env:
```

```
      - name: FOO
        value: bar
    parameters:
      - name: string-param
        string: example-string
      - name: array-param
        array: [item1, item2]
      - name: map-param
        map:
          param-name: param-value

sources:
  - repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: HEAD
    path: guestbook
    ref: my-repo

destination:
  server: https://kubernetes.default.svc
  namespace: guestbook

info:
  - name: '示例:'
    value: 'https://example.com'

syncPolicy:
  automated:
    prune: true
    selfHeal: true
    allowEmpty: false
  syncOptions:
  - Validate=false
  - CreateNamespace=true
  - PrunePropagationPolicy=foreground
  - PruneLast=true
  - RespectIgnoreDifferences=true
  - ApplyOutOfSyncOnly=true
  managedNamespaceMetadata:
    labels:
      any: label
      you: like
    annotations:
      the: same
      applies: for
```

```
      annotations: on-the-namespace

  retry:
    limit: 5
    backoff:
      duration: 5s
      factor: 2
      maxDuration: 3m

ignoreDifferences:
- group: apps
  kind: Deployment
  jsonPointers:
  - /spec/replicas
- kind: ConfigMap
  jqPathExpressions:
  - '.data["config.yaml"].auth'
- group: "*"
  kind: "*"
  managedFieldsManagers:
  - kube-controller-manager
  name: my-deployment
  namespace: my-namespace
revisionHistoryLimit: 10
```

参考资料

- Argo CD 官方文档↗

ApplicationSet

目录

介绍

ApplicationSet 控制器是一个 Kubernetes 控制器，它扩展了对 ApplicationSet 自定义资源定义 (CRD) 的支持。该控制器/CRD 实现了自动化和更灵活的管理 Argo CD 应用程序，并且可以在大量集群和单一代码库中使用，同时还使得在多租户 Kubernetes 集群上实现自助服务成为可能。

适用场景

- 部署多个相似应用程序：当您需要部署多个配置相似的应用程序时，可以使用 ApplicationSet 来减少重复配置。例如，您可以使用 ApplicationSet 部署多个微服务，这些微服务使用相同的模板，但服务名称和端口号不同。
- 多集群部署：当您需要在多个 Kubernetes 集群中部署相同的应用程序时，可以使用 ApplicationSet 简化配置。例如，您可以通过 ApplicationSet 定义一个应用程序并在多个集群中部署，每个集群使用不同的参数。

- 动态生成应用程序：当您需要根据某些条件动态生成应用程序时，可以利用 ApplicationSet。例如，您可以根据 Git 仓库中的分支或标签动态生成不同的应用程序实例。

示例

```yaml
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: guestbook
spec:
  goTemplate: true
  goTemplateOptions: ["missingkey=error"]
  generators:
  - list:
      elements:
      - cluster: engineering-dev
        url: https://1.2.3.4
      - cluster: engineering-prod
        url: https://2.4.6.8
      - cluster: finance-preprod
        url: https://9.8.7.6
  template:
    metadata:
      name: '{{.cluster}}-guestbook'
    spec:
      project: my-project
      source:
        repoURL: https://github.com/infra-team/cluster-deployments.git
        targetRevision: HEAD
        path: guestbook/{{.cluster}}
      destination:
        server: '{{.url}}'
        namespace: guestbook
```

参考资料

- Argo CD ApplicationSet 文档↗

Tool

目录

介绍

Tool 是指用于生成或处理 Kubernetes 资源 `Manifests` 的工具。

支持的工具

Argo CD 支持几种不同的 Kubernetes 清单定义方式：

- **Kustomize** 应用 Kustomize
- **Helm** 图表 Helm
- 目录：包含 `YAML` / `JSON` / `Jsonnet` 文件的清单，包括 `Jsonnet` 目录
- 自定义配置管理插件：任何配置为配置管理插件的自定义工具

开发工作流程

Argo CD 允许直接上传本地 `manifests`，但这仅用于开发目的。覆盖操作需要具有权限的用户（通常是管理员）才能上传本地 `manifests`。它支持所有上述的 Kubernetes 部署工具。要上传本地应用程序：

```
$ argocd app sync APPNAME --local /path/to/dir/
```

特性对比

特性	**Helm**	**Kustomize**	目录（纯 **YAML**）
配置方式	模板化（动态生成）	声明式（补丁和覆盖）	静态 YAML 文件
复用性	高（通过 Charts）	中（通过基础/覆盖）	低
多环境支持	高（通过 values.yaml）	高（通过覆盖）	低
渐进式交付	高（支持复杂逻辑）	中（支持简单补丁）	低
学习成本	高（模板语法）	低（基于 YAML）	低
与 **Argo CD** 集成	支持	原生支持	支持
适用场景	复杂应用、多环境、分发	多环境、配置复用	小型项目、快速原型开发

参考资料

更详细的信息，请参考：Tool ↗

目录

介绍

Helm 是一个用于 Kubernetes 的包管理工具，允许用户定义、安装和升级复杂的 Kubernetes 应用程序。**Helm** 图表是一个模板化配置包，包含 Kubernetes 资源定义（YAML 文件）。

Helm 的核心概念

- **Chart**：Helm 图表是一个模板化的配置包，包含 Kubernetes 资源定义（YAML 文件）。
- **Release**：Helm 发布是已部署的 Helm 图表的实例，代表 Kubernetes 资源的特定配置。
- **Values**：Helm 值是 Helm 图表的参数化配置，允许用户自定义 Kubernetes 资源定义。

Argo CD 与 Helm 的集成通过 Web 控制台、Argo CD 仪表板或 CLI，增强了 GitOps 实践，实现声明式的持续交付。示例如下：

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: sealed-secrets
  namespace: argocd
spec:
  project: default
  source:
    chart: sealed-secrets
    repoURL: https://bitnami-labs.github.io/sealed-secrets
    targetRevision: 1.16.1
    helm:
      releaseName: sealed-secrets
  destination:
    server: "https://kubernetes.default.svc"
    namespace: kubeseal
```

OCI Helm Chart 示例：

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: nginx
spec:
  project: default
  source:
    chart: nginx
    repoURL: registry-1.docker.io/bitnamicharts  # note: the oci:// syntax is
    targetRevision: 15.9.0
  destination:
    name: "in-cluster"
    namespace: nginx
```

INFO

应用程序的生命周期由 **Argo CD** 管理，而非 **Helm**。 当提供多个值源时，优先顺序为：`parameters` > `valuesObject` > `values` > `valueFiles` > helm 仓库 `values.yaml`。

优势

- 模板化：Helm 使用 Go 模板引擎（gotpl）动态生成 Kubernetes 资源文件。
- 包管理：Helm 将应用打包为图表（包括模板、默认值和依赖关系），简化了分发和版本控制。
- 依赖管理：支持图表之间的依赖关系。
- 生命周期管理：提供 `install`、`upgrade` 和 `rollback` 等命令以进行完整的生命周期管理。

使用场景

- 复杂应用部署：适用于需要动态生成配置的场景（例如，根据环境变量或用户输入生成资源）。
- 多环境部署：通过 `values.yaml` 文件支持特定环境的配置。
- 应用分发：能够将图表打包进行分发到 Helm 仓库或 OCI 注册表。

参考资料

更多详细信息，请参考：Helm ↗

Kustomize 概念

目录

介绍

Kustomize 是一个 Kubernetes 原生配置管理工具，允许用户通过覆盖和组合的方式来定制 Kubernetes 资源定义（YAML 文件），而无需直接修改原始文件。

Kustomize 的核心概念

- **Base**：基础配置，包含通用的 Kubernetes 资源定义。
- **Overlay**：覆盖层，基于 Base 进行定制化修改。
- **kustomization.yaml**：定义如何组合和修改资源的配置文件。

Argo CD 与 Kustomize 的集成使得 GitOps 实践更加高效，能够实现声明式的持续交付，示例如下：

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: kustomize-example
spec:
  project: default
  source:
    path: examples/helloWorld
    repoURL: 'https://github.com/kubernetes-sigs/kustomize'
    targetRevision: HEAD
  destination:
    namespace: default
    server: 'https://kubernetes.default.svc'
```

如果 `repoURL` 和 `path` 指向的位置存在 `kustomization.yaml` 文件，Argo CD 将使用 Kustomize 呈现清单。

Kustomize 有以下配置选项：

- `namePrefix`：附加在 Kustomize 生成的资源名称上的前缀。
- `nameSuffix`：附加在 Kustomize 生成的资源名称上的后缀。
- `images`：Kustomize 图像覆盖列表。
- `replicas`：Kustomize 副本覆盖列表。
- `commonLabels`：附加到所有资源的标签映射。
- `labelWithoutSelector`：布尔值，定义是否将通用标签应用于资源选择器和模板。
- `forceCommonLabels`：布尔值，允许覆盖现有标签。
- `commonAnnotations`：附加到所有资源的注释映射。
- `namespace`：Kubernetes 资源命名空间。
- `forceCommonAnnotations`：布尔值，允许覆盖现有注释。
- `commonAnnotationsEnvsubst`：布尔值，允许在注释值中使用环境变量替换。
- `patches`：支持内联更新的 Kustomize 补丁列表。
- `components`：Kustomize 组件列表。

要将 Kustomize 与覆盖一起使用，请将路径指向覆盖目录。

优势

- 声明式配置：通过 YAML 文件（通过 kustomization.yaml）定义资源的组合和修改。
- 无模板：通过补丁和覆盖实现配置定制，而无需模板引擎。
- 与 Kubernetes 原生集成：Kustomize 内置于 kubectl 中，无需额外工具。

使用场景

- 多环境分发：通过 Base 和 Overlay 实现针对不同环境的特定配置（例如：应用、集群）。
- 配置复用：适合在多个项目中复用基础配置。
- 渐进式交付：通过补丁逐步调整资源配置。

参考资料

更加详细的说明请参考：Kustomize ↗

目录

问题描述

1. 不符合中文表达习惯：

- "Directory 类型应用程序直接从 `.yml`、`.yaml` 或 `.json` 文件加载 `manifests`。" 这句话中的“加载”可以更自然地表达为“从 `.yml`、`.yaml` 或 `.json` 文件中加载 `manifests`”。
- "声明式语法示例如下：" 这句话可以更流畅地表达为“以下是声明式语法的示例：”。
- "不需要指定 `spec.source.directory` 字段。" 这句话可以更清晰地表达为“除非需要额外的配置选项，否则可以不指定 `spec.source.directory` 字段。”。

2. 语句不通顺：

- "Argo CD 会自动检测源代码库/路径是否包含纯粹的清单文件。" 这句话中的“源代码库/路径”可以更清晰地分开为“源代码库或路径”。

3. 晦涩难懂：

- "低维护性" 这个表达可以更明确为“维护成本低”。

- "无动态模板或复杂的配置管理" 这部分可以更清晰地表达为“无需动态模板或复杂的配置管理”。

意译结果

---

sourceSHA: e8d7a40d5fe32bb85f84d87db6e2c3843acb713cf368ede32be799a98a9547b0 weight: 70

Directory

介绍

Directory 类型应用程序从 `.yml`、`.yaml` 或 `.json` 文件中加载 `manifests`。可以通过平台 UI、Argo CD Dashboard、CLI 或声明方式创建 Directory 应用程序。以下是声明式语法的示例：

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
spec:
  destination:
    namespace: default
    server: https://kubernetes.default.svc
  project: default
  source:
    path: guestbook
    repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: HEAD
```

除非需要额外的配置选项，否则可以不指定 `spec.source.directory` 字段。Argo CD 会自动检测源代码库或路径是否包含纯粹的清单文件。

优势

- 简单性：直接从清单文件加载资源，而无需额外的抽象。
- 维护成本低：没有配置管理的负担。

使用场景

- 管理多个 Kubernetes 资源（例如，Deployments、Services、ConfigMaps）。
- 小规模项目、资源有限或快速采用 GitOps。
- 部署原始 YAML 文件，无需动态模板或复杂的配置管理。

WARNING

Directory 类型应用程序 仅支持纯粹的清单文件。如果 Argo CD 在 Directory 路径中检测到 `Kustomize`、`Helm` 或 `Jsonnet` 文件，它将无法呈现清单。

参考资料

有关更详细的说明，请参阅：Directory↗

Sync

目录

Sync 概述

Sync 是 Argo CD 的核心功能，负责将应用程序的 期望状态 与 实际状态 进行比较，并采取措施以调和差异。
从本质上讲，Sync 确保 Kubernetes 集群中应用程序的状态与 Git 仓库中定义的状态保持一致。
您可以手动触发 Sync，或配置 Argo CD 自动执行 Sync。自动 Sync 可以通过监控 Git 仓库的变更（例如提交、标签推送）来触发，或在预定的时间间隔内执行。

Sync 状态概述

Sync 状态指示应用程序的同步状态，反映其 实际状态 是否与 期望状态 匹配。Sync 状态包括以下几种状态：

- `Synced`：应用程序的 实际状态 完全匹配 期望状态。
- `OutOfSync`：应用程序的 实际状态 与 期望状态 不一致。

- `Syncing`：应用程序正在进行同步，实际状态 正在向 期望状态 靠拢。

Sync 操作状态概述

Sync 操作状态表示 Argo CD 执行同步操作的状态，指示操作是否成功完成。Sync 操作状态包括以下几种状态：

- `Succeeded`：同步操作成功完成。
- `Failed`：同步操作因 Kubernetes 资源冲突、权限不足等原因失败。
- `Running`：同步操作正在进行中。

刷新概述

此操作从 Git 仓库获取最新的应用程序配置，并将其与 Kubernetes 集群中的实际状态进行比较。刷新可以手动触发，也可以配置为在定义的时间间隔内自动执行。

参考资料

有关更详细的信息，请参阅：Sync↗

Health

目录

介绍

应用程序的健康状况，是否正常运行？是否可以处理请求？

健康范围

健康状态	说明
健康	资源健康。
处理中	资源尚未健康，但仍在处理过程中，可能很快就会健康。
降级	资源健康状况降级。
暂停	资源已暂停，等待某些外部事件恢复（例如：暂停的 CronJob 或暂停的 Deployment）。

参考资料

Argo CD 为几种标准 Kubernetes 类型提供内置健康评估，然后将其反映到整个应用程序的健康状态中，当然 Argo CD 也支持自定义健康检查。

有关更详细的说明，请参考：Health↗

Menu

Alauda Container Platform GitOps 核心概念

Alauda Container Platform GitOps 概念

为什么选择 Argo CD？

优势

Alauda Container Platform GitOps 的同步及健康检查

同步状态说明

健康状态说明

识别规则

介绍

Alauda Container Platform GitOps 是一款基于 Argo CD 的 Kubernetes 原生 GitOps 解决方案。它监控 Git 仓库中的配置清单（应用程序、基础设施定义等），并将其自动同步到目标 Kubernetes 集群，实现基于 Git 的持续交付。通过将整个交付管道编入 Git 的版本控制系统，提高了部署速度、可靠性和安全性，同时实现精确的多集群应用分发。

该解决方案原生集成了 Argo CD Operator，自动化执行包括资源配置、升级和回滚等部署生命周期操作。

目录

为什么选择 **Argo CD**？

Argo CD 凭借其独特的优势，成为业界领先的开源 GitOps 引擎：

技术优势	操作收益
声明式 **GitOps** 引擎 • 通过 CRD 进行状态协调（Application、ApplicationSet） • 多来源支持（Helm、Kustomize、原生 YAML）	加速部署 • 通过基于 Git 的自动化将部署周期加快 70%
Kubernetes 原生架构	企业就绪

技术优势	操作收益
• 与 Kubernetes API 服务器深度集成 ≈ 原生支持命名空间隔离和 RBAC	• 内置多租户和审计功能
多集群管理 • 用于混合/多云部署的集中控制平面 • 通过 ApplicationSets 实现集群特定的配置	运营效率 • 通过声明性强制执行将部署错误减少 60%
可扩展插件系统 • 与 Helm、Kustomize、Istio 的认证集成 • 用于高级工作流的自定义资源定义（CRD）	成本优化 • 通过精确的资源编排将云成本降低 40%
活跃的 **CNCF** 生态系统 • 超过 3,500 个 GitHub stars • 超过 200 个活跃贡献者	面向未来 • 通过开源社区持续创新

优势

> 除了 GitOps 自身的优势外，Alauda Container Platform GitOps 还提供以下增强的益处：

- 企业级 **Argo CD Operator**
  - 提供原生 Argo CD Operator 的完整功能，涵盖应用部署、升级、回滚以及 Argo CD 的所有核心功能。
- **Argo CD Operator** 安全服务
  - 提供针对 Argo CD Operator 的专门技术支持，处理故障响应、安全漏洞修复及整体系统稳定性等问题。
- 可视化的 **GitOps** 应用多环境分发管理

    - 利用该平台的多集群管理和差异化配置能力，实现风格一致、可视化的 GitOps 应用管理和集群配置管理，从而简化多云和多环境的精确分发。

- 可视化的 **GitOps** 应用运维
    - 提供对 GitOps 应用下 Kubernetes Workload 资源的实时日志和事件的直接访问。在 GitOps 应用出现异常时，用户可以利用 Argo CD 的异常信息及实时 Workload 日志迅速分析并解决问题，而无需离开当前界面。
- 可视化的 **GitOps** 集群配置管理
    - 通过 GitOps 管理集群配置，实现集群配置的统一管理和可视化分发。
- 与平台所有产品集成实现 **GitOps** 应用的闭环管理
    - 将平台的 DevOps 能力与持续构建、制品管理、微服务灰度发布等相结合，实现在 GitOps 应用管理中的完全自动化，形成持续集成与持续交付协作的完整闭环。
    - 与平台的 CrossPlane、MySQL 和 Developer Portal 等产品结合，实现从基础设施初始化、业务应用初始化（包括代码、流水线、GitOps 应用初始化等）到应用代码开发与上线的完整流程。

Alauda Container Platform GitOps 的同步及健康检查

Alauda Container Platform GitOps 通过利用底层 Kubernetes 资源的状态，封装了 `Application` 资源的状态。`Application` 资源的状态直接决定了关联的 `ApplicationSet` 资源的状态。

目录

同步状态说明

Kubernetes 资源和应用有四种同步状态：同步失败、待同步、同步中和已同步。

同步状态	说明
同步失败	由于网络错误、配置问题或权限问题导致同步失败。请查看日志以确定根本原因。
待同步	集群资源状态与 Git 定义的期望状态不一致。需要手动或自动同步。
同步中	集群状态与 Git 定义的状态之间正在进行活动协调。
已同步	集群资源状态与 Git 定义的期望状态一致。

INFO

同步状态显示优先级：优先级顺序为 同步失败 > 待同步 > 同步中 > 已同步。

示例：

- 如果一个 Application 拥有两个资源，其状态分别为 同步中 和 已同步，则其整体状态为 同步中。
- 如果一个 ApplicationSet 管理两个 Application，其状态分别为 同步失败 和 已同步，则其整体状态为 同步失败。

健康状态说明

Kubernetes 资源和应用有六种健康状态：未知、丢失、降级、暂停、处理中和健康。

健康状态	说明	参考解决方案
未知	无法确定健康状态，通常由于控制器错误或缺失状态数据。	检查资源 YAML 的 `status.conditions` 获取诊断细节。
丢失	集群中未找到资源。	初始创建：等待协调 意外删除：触发手动同步。
降级	Workload 资源（例如 Deployment）在超时（默认：10 分钟）内未能达到健康状态。	检查 Pod 错误（例如，崩溃、资源限制）。
暂停	Workload 资源的滚动更新被故意暂停（例如，通过 `kubectl rollout pause`）。	如适用，恢复滚动更新。
处理中	资源成功创建但尚未完全就绪（例如，Pods 初始化中）。	监视，直到过渡到健康/降级状态。
健康	资源正常运行。	-

INFO

健康状态优先级：优先级顺序为 未知 > 丢失 > 降级 > 暂停 > 处理中 > 健康。

示例：

- 如果一个 Application 的健康状态为 健康 和 未知，则其整体健康状态为 未知。
- 如果一个 ApplicationSet 管理的 Applications 的状态为 丢失 和 处理中，则其整体健康状态为 丢失。

识别规则

对于 Kubernetes 资源，达到以下健康状态的规则如下：

资源类型	状态
Deployment	滚动更新已完成，且所有副本可用。
StatefulSet	更新完成，且所有 Pods 就绪。
ReplicaSet	所有 Pods 健康。
DaemonSet	已调度且健康的 Pods 数量达到所需数量。
Ingress	LoadBalancer IP/主机名已填充至状态中。
Service	LoadBalancer IP/主机名已填充（如果适用）。
PVC	状态为 已绑定。
Pod	所有容器均就绪且重启次数未超过阈值。
Job	Job 成功完成 (`.status.succeeded >= 1`)。
HPA	当前副本数与期望副本数匹配的成功扩缩容操作。

Menu

功能指南

创建 GitOps 应用

创建 GitOps 的 Application

前提条件

通过网页控制台创建 Argo CD 应用

通过 YAML 创建 Argo CD 应用

通过 CLI 创建 Argo CD 应用

创建 GitOps 的 ApplicationSet

问题描述

意译结果

weight: 10 i18n: title: en: Creating GitOps ApplicationSet zh: 创建 GitOps 的 ApplicationSet

概述

前提条件

主要收益

创建 GitOps 应用

管理 GitOps 应用

GitOps 可观测

Argo CD 组件监控

概述

前提条件

查看 Argo CD 组件仪表板

GitOps 应用运维

概述

前提条件

告警

日志

事件

Menu

创建 GitOps 应用

创建 GitOps 的 Application

前提条件

通过网页控制台创建 Argo CD 应用

通过 YAML 创建 Argo CD 应用

通过 CLI 创建 Argo CD 应用

创建 GitOps 的 ApplicationSet

问题描述

意译结果

weight: 10 i18n: title: en: Creating GitOps ApplicationSet zh: 创建 GitOps 的 ApplicationSet

概述

前提条件

主要收益

创建 GitOps 应用

管理 GitOps 应用

创建 GitOps 的 Application

概述

利用 **Alauda Container Platform GitOps** 的应用管理能力，通过 **GitOps Applications** 直观地创建 Argo CD ApplicationSet，实现容器化应用的全面生命周期管理。

目录

前提条件

- 安装 **Alauda Container Platform GitOps**：
  - 如果未安装，请联系管理员以 安装 Alauda Container Platform GitOps
- **Git** 仓库集成（选择一种方法）：
  - 通过 Argo CD 仪表板集成代码仓库

- 管理员必须通过 **DevOps** 工具链 > 集成 提供代码仓库

通过网页控制台创建 **Argo CD** 应用

> 通过可视化管理界面简化应用分发。

操作步骤

1. 容器平台，并导航到 **GitOps Applications**。
2. 点击 创建 **GitOps** 应用。
3. 在 基本信息 和 代码仓库 部分配置参数：

参数	描述
类型	应用： Argo CD 应用对象用于单命名空间部署 **ApplicationSet**：Argo CD ApplicationSet 用于跨集群/跨命名空间的差异化配置部署
来源	平台集成：预配置的 GitLab/GitHub/Bitbucket 仓库 **ArgoCD** 集成：通过 Argo CD 集成的 GitLab/GitHub/Bitbucket/Gitee/Gitea 仓库。请参考 通过 Argo CD 仪表板集成代码仓库
集成项目名称	管理员分配的工具链项目
版本标识	部署基础： `Branch` / `Tag` / `Commit` 注意： • `Branch` 使用最新提交 • `Tag` / `Commit` 默认使用最新但可配置
源文件类型	**Kustomize**：使用 kustomization.yaml 进行覆盖配置；有关更多详细信息，请参见 Kustomize 官方文档↗ **Helm**：使用 values.yaml 进行模板；有关更多详细信息，请参见 Helm 官方

参数	描述
	文档↗ 目录：原始清单
源目录	包含基础清单的仓库路径。支持选择根目录。该路径中的所有资源将被部署到目标集群
自定义值	如果 源文件类型 为 **Helm**，您可以选择自定义的 Helm 值文件

4. 在 目标 部分配置参数：

   - 应用：差异配置不会修改源目录中的基础文件。
   - **ApplicationSet**：具有 差异化配置 的多集群部署。

   注意：差异化配置 不会修改 源目录 中的基础文件。

5. 同步策略（3分钟调和间隔）。

参数	描述
手动同步	检测到偏差时需要用户确认。
自动同步	自动调和，无需人工干预。
同步忽略配置	使用内置/自定义忽略模板进行配置，您可以 查看 YAML 文件中的同步忽略配置字段。 注意：自定义模板需要管理员配置。

6. 点击 创建。

> **INFO**
>
> 手动同步说明：选择 立即同步 进行即时部署，或选择 稍后同步 通过详细页面手动触发。

查看 YAML 文件中的同步忽略配置字段

在配置同步忽略规则后，通过以下方式验证：

1. 导航到 **GitOps** 应用
2. 选择目标应用
3. 点击 操作 **>** 更新
4. 检查 YAML 文件。

```
ignoreDifferences: # 由所选的自定义同步忽略配置模板实际忽略的配置
- group: apps
  kind: Deployment
  jsonPointers:
  - /spec/replicas
```

通过 YAML 创建 Argo CD 应用

操作步骤

1. 容器平台，并导航到 **GitOps Applications**。
2. 点击 创建 **GitOps** 应用。
3. 切换到 **YAML** 标签。
4. 在 **YAML** 部分，参考以下 YAML 文件并配置相关信息。将 `namespace` 和 `project` 替换为您自己的命名空间和项目。

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
  namespace: argocd # 替换为您自己的命名空间
spec:
  project: default # 替换为您自己的项目
  source:
    repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: master
    path: helm-guestbook
  destination:
    server: https://kubernetes.default.svc
    namespace: guestbook
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
    - CreateNamespace=true
```

1. 点击 创建。

通过 CLI 创建 Argo CD 应用

前提条件

已安装 web-cli 插件，并启用了 `web-cli` 开关。

```
kubectl apply -f application.yaml
```

目录

问题描述

1. 不符合中文表达习惯：

- "利用 **ACP GitOps** 应用管理功能，直观地创建 Argo CD ApplicationSet，以通过 **GitOps** 应用 实现容器化应用的全面生命周期管理。" 这句话较长且结构复杂，建议简化。
- "通过可视化管理界面简化应用分发。" 这句话可以更自然地表达为“通过可视化管理界面来简化应用的分发”。

2. 语句不通顺：

- "在 容器平台 中，导航到 **GitOps** 应用。" 这句话可以更流畅地表达为“在 容器平台 中导航至 **GitOps** 应用”。
- "点击 创建 **GitOps** 应用。" 这句话可以更自然地表达为“点击 创建 **GitOps** 应用 按钮”。

3. 晦涩难懂：

- "差异配置不会修改源目录中的基本文件。" 这句话可以更清晰地表述为“差异配置不会对源目录中的基本文件进行修改”。

意译结果

---

weight: 10 i18n: title: en: Creating GitOps ApplicationSet zh: 创建 GitOps 的 ApplicationSet

创建 GitOps 的 ApplicationSet

概述

利用 **Alauda Container Platform GitOps** 应用管理功能，您可以直观地创建 Argo CD ApplicationSet，从而通过 **GitOps** 应用 实现容器化应用的全面生命周期管理。

前提条件

- 安装 **Alauda Container Platform GitOps**：
  - 如果尚未安装，请联系管理员进行 安装 Alauda Container Platform GitOps
- **Git** 仓库集成（选择一种方法）：
  - 通过 Argo CD 仪表板集成代码仓库

- 管理员必须通过 **DevOps** 工具链 > 集成 配置代码仓库

主要收益

- 可视化 **GitOps** 应用分发：结合多集群管理、差异配置和平台对齐的可视化操作，简化多云/多环境的部署过程。

创建 **GitOps** 应用

> 通过可视化管理界面来简化应用的分发。

流程

1. 在 容器平台 中导航至 **GitOps** 应用。
2. 点击 创建 **GitOps** 应用 按钮。
3. 在 基本信息 和 代码仓库 部分配置参数：

参数	描述
类型	应用：用于单命名空间部署的 Argo CD 应用对象 **ApplicationSet**：用于跨集群/跨命名空间部署并具有差异配置的 Argo CD ApplicationSet
来源	已集成的平台：预配置的 GitLab/GitHub/Bitbucket 仓库 与 **ArgoCD** 集成：通过 Argo CD 集成的 GitLab/GitHub/Bitbucket/Gitee/Gitea 仓库。有关更多信息，请参阅 通过 Argo CD 仪表板集成代码仓库
集成项目名称	管理员分配的工具链项目
版本标识	部署基础：`分支` / `标签` / `提交` 注意：

参数	描述
符	• `分支` 使用最新提交 • `标签` / `提交` 默认使用最新但可配置
源文件类型	**Kustomize**：使用 `kustomization.yaml` 进行叠加配置；有关更多详细信息，请参阅 Kustomize 官方文档↗ **Helm**：使用 values.yaml 进行模板化；有关更多详细信息，请参阅 Helm 官方文档↗ 目录：原始清单
源目录	包含基本清单的仓库路径。支持选择根目录。此路径中的所有资源将部署到目标集群
自定义值	当 源文件类型 为 **Helm** 时，可以选择自定义 Helm 值文件

4. 在 目标 部分配置参数：

   - 应用：差异配置不会对源目录中的基本文件进行修改。
   - **ApplicationSet**：使用 差异化配置 进行多集群部署。

   注意：差异化配置 不会修改 源目录 中的基本文件。

5. 同步策略（3 分钟的协调间隔）。

参数	描述
手动同步	检测到漂移时需要用户确认
自动同步	无需人工干预的自动协调
同步忽略配置	使用内置/自定义忽略模板进行配置，您可以 查看 YAML 文件中的同步忽略配置字段 注意：自定义模板需要管理员配置

6. 点击 创建。

INFO

手动同步注意：选择 立即同步 进行立即部署或选择 稍后同步 以便通过详情页面手动触发。

查看 YAML 文件中的同步忽略配置字段

在配置同步忽略规则后，请通过以下步骤进行验证：

1. 导航到 **GitOps** 应用。
2. 选择目标应用。
3. 点击 操作 > 更新。
4. 检查 YAML 文件。

```yaml
ignoreDifferences: # 实际被选定的自定义同步忽略配置模板忽略的配置
- group: apps
  kind: Deployment
  jsonPointers:
  - /spec/replicas
```

管理 GitOps 应用

操作	描述
更新	通过以下方式发起更新： • 在 **GitOps** 应用 列表中点击编辑图标 (✎) • 在详情视图中选择 操作 > 更新 。 • 注意：此操作将覆盖所有创建的应用实例
手动同步	当 同步策略 为 手动同步 时： • 在检测到配置漂移后，通过在详情视图中选择 操作 > 同步 启动同步

操作	描述
	• 将最新提交传播到所有管理实例
删除	通过以下方式删除： • 在列表页面点击删除图标 (🗑) • 在详情视图中选择 操作 > 删除 • 破坏性：删除应用及所有子资源
自动同步	启用自动协调以维持所需状态。所有实例每 3 分钟自动与仓库的更改同步
来源	对于 **ApplicationSet** 类型的应用： • 点击 来源 链接导航到父应用的详情页面。
应用分发	扩展： 1. 更新现有 **ApplicationSet** 配置 2. 在 **ApplicationSet** 详情中：应用 > 添加分发

Menu

GitOps 可观测

Argo CD 组件监控

概述

前提条件

查看 Argo CD 组件仪表板

GitOps 应用运维

概述

前提条件

告警

日志

事件

Argo CD 组件监控

目录

概述

Web 控制台的监控仪表板提供了一种可视化的方法来监控 Argo CD 组件。它主动观察 Argo CD 组件的资源和操作状态，旨在确保它们的健康性和可用性。在这里，操作状态指的是 Kubernetes (K8s) 环境中组件的运行条件和性能指标。通过密切跟踪这些方面，我们可以及时发现并解决任何潜在问题，保持 Argo CD 在 K8s 集群中的顺利运行。

前提条件

- 安装 Alauda Container Platform GitOps
- 安装监控插件

查看 Argo CD 组件仪表板

1. 登录并导航到 平台管理，选择 `global` 集群。

2. 点击 运维中心 > 监控 > 监控仪表板。

3. 点击 切换 按钮，选择 **container-platform** 以查看 **ArgoCD** 仪表板。

4. 点击 **ArgoCD** 仪表板以查看 **Argo CD** 组件监控信息。

GitOps 应用运维

目录

概述

通过平台的 **GitOps** 应用 管理功能，可以查看 **GitOps** 应用 的监控、日志和事件。您还可以为 **GitOps** 应用 创建告警策略。当 **GitOps** 应用 发生异常时，将触发主动告警通知，以便快速识别、分析和解决问题。

前提条件

- 已在 web 控制台创建 GitOps 应用。通过 web 控制台创建 Argo CD 应用
- 安装监控插件

告警

> 提前创建告警规则以配置规则。当 GitOps 应用遇到异常时，将触发主动通知，以便快速识别、分析和解决问题。

1. 在 容器平台 中，单击 **GitOps** 应用。
2. 从列表中选择您要创建告警规则的 GitOps 应用名称。
3. 切换到 告警 页签。
4. 单击 创建规则，根据要求填写基本信息。
5. 单击 添加告警条件，导航至 告警条件 页面。相应的指标说明如下：

INFO

有关其他参数配置和告警设置，请参阅 告警管理。

指标名称	规则说明
GitOps 应用健康状态 `gitops.applicationset.healthy`	GitOps 应用的健康状态： - 0：未知、丢失、降级或暂停 - 1：正在同步 - 2：健康
GitOps 应用同步状态 `gitops.applicationset.synced`	GitOps 应用的同步状态： - 0：同步失败或待同步 - 1：正在同步 - 2：已同步

1. 单击 创建。

日志

> 查看所有由 GitOps 应用创建的工作负载资源的日志。日志可以快速识别系统故障信息，而无需依赖 日志 集群插件。

- 在 GitOps 应用详情页面的 **Kubernetes** 资源 下，单击任意工作负载名称以在右侧查看该资源的 日志 信息。

事件

> 查看 **GitOps** 应用 分发的所有资源的事件。事件可以快速识别系统故障事件信息，而无需依赖 日志 集群插件。

- 在 GitOps 应用详情页面，进入 事件 页签以查看所有资源的聚合事件。
- 在 GitOps 应用详情页面的 **Kubernetes** 资源 下，单击任意资源名称以在右侧查看该资源的事件信息。

Menu

实用指南

通过 **Argo CD Dashboard** 集成代码仓库

使用场景

前置条件

操作步骤

操作结果

通过 **Argo CD dashboard** 创建 **Argo CD Application**

前提条件

操作步骤

通过平台创建 **Argo CD Application**

用例

前提条件

步骤

如何获取 **Argo CD** 访问信息

使用场景

如何获取通过平台部署的 GitOps 集群插件的 Argo CD 访问信息？

如何获取通过 Argo CD Operator 部署的 Argo CD 访问信息？

使用 Argo CD Dashboard 集成代码仓库

使用原生的 Argo CD Dashboard 集成代码仓库并分配仓库，便于开发人员通过可视化界面管理 GitOps 应用的整个生命周期。

目录

使用场景

- 通过在创建 **GitOps** 应用 时在网页控制台中选择关联的仓库，简化 **GitOps** 应用 的创建过程。
- 在原生的 Argo CD Dashboard 上创建 **Application** 时，可以选择使用关联的仓库。

前置条件

- 安装 Alauda Container Platform GitOps，且 原生 **Argo CD UI** 开关已启用。
- 获取 原生 **Argo CD UI** 的 URL 以及用户名和密码。
  - 管理员可以通过 GitOps 集群插件详情页面直接访问该 URL。

操作步骤

按以下步骤利用这些功能：

1. 连接代码仓库

   - 使用访问 URL 登录 **Argo CD**。
   - 在左侧导航栏中点击 **Settings**。
   - 点击 **REPO** 卡片。
   - 在页面左上角点击 **CONNECT REPO**。
   - 选择连接仓库的方法，并按需填写相应参数。
   - 点击 **CONNECT**。

2. 关联项目

   - 在左侧导航栏中点击 **Settings**。
   - 点击 **Projects** 卡片。
   - 点击需要创建 GitOps 应用的项目。

   注意：Argo CD 会自动同步集群中的项目，因此无需手动创建。

   - 在 **SOURCE REPOSITORIES** 部分点击 **EDIT**。
   - 点击 **ADD SOURCE**，输入 连接仓库 步骤中的仓库 URL，并将其与项目关联。
   - 点击 **SAVE**。

操作结果

返回到网页控制台，导航到 容器平台 > **GitOps** 应用。在 创建 页面，您将看到已关联的仓库。

通过 Argo CD Dashboard 创建 Argo CD Application

目录

前提条件

- 安装（选择一种方法）：
  - 安装 Alauda Container Platform GitOps
  - 安装 Alauda Build of Argo CD
- 已获取 **Argo CD Dashboard** 的访问凭证（URL、用户名、密码），具体可参考 如何获取 Argo CD 访问信息

操作步骤

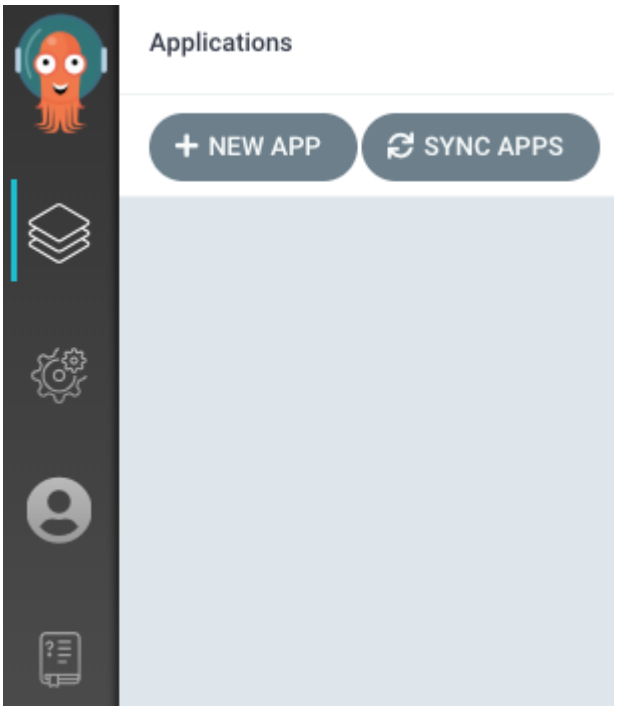
> 按以下步骤使用功能：

1. 在浏览器中输入 Argo CD dashboard 访问 URL，打开界面。

> 管理员可以通过 `global` 集群插件详情直接访问 **Argo CD** 原生 **UI**：定位到 GitOps 集群插件并点击访问地址。

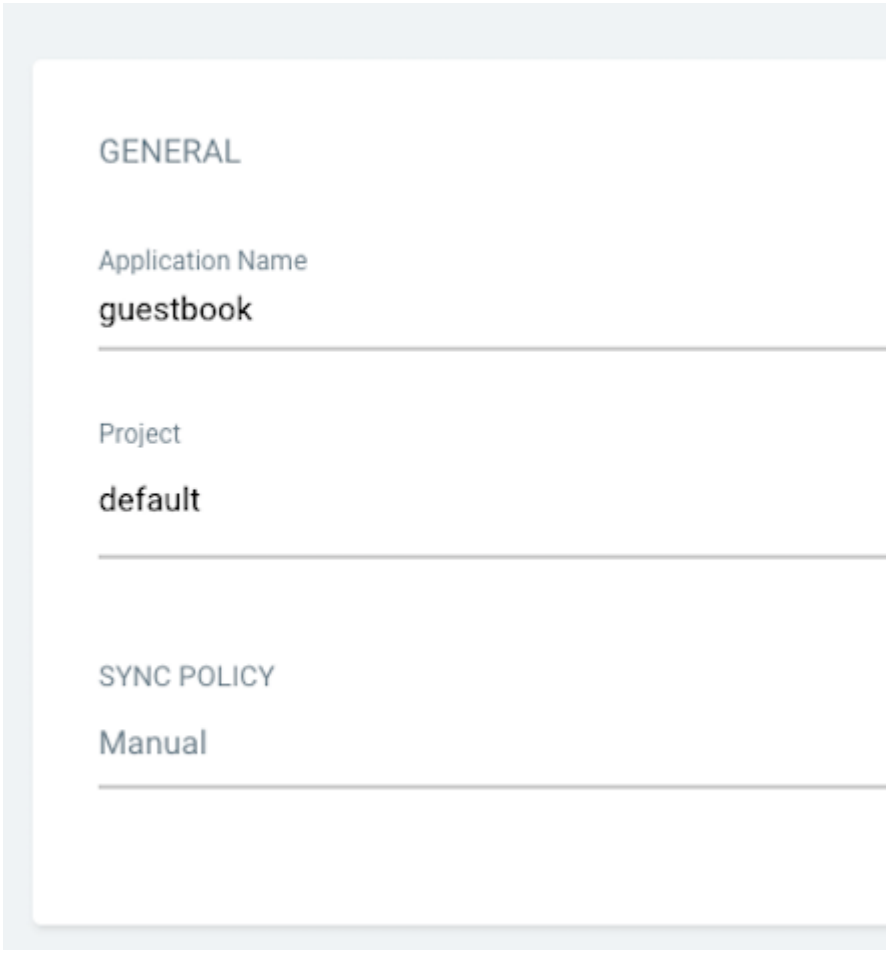
2. 输入 Argo CD 凭证进行身份验证并登录。

3. 点击如下所示的 **+ NEW APP** 按钮：



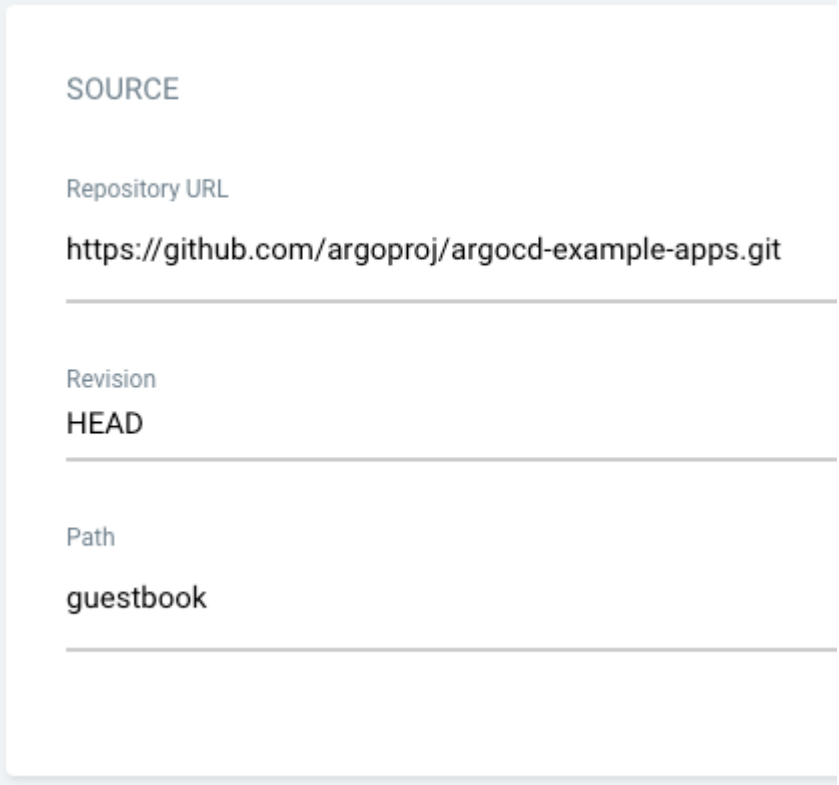
> 按照以下步骤配置应用：

基本信息配置



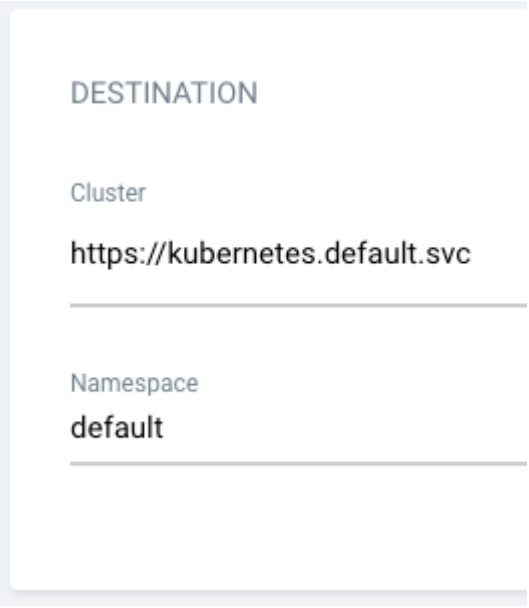
- **Application Name**：输入 `guestbook`
- **Project**：选择 `default`
- **Sync Policy**：保持为 `Manual`（建议用于初始配置）

源代码配置



- **Repository URL**：设置为 `https://github.com/argoproj/argocd-example-apps.git`
- **Revision**：使用默认的 `HEAD`
- **Path**：指定 `guestbook`（包含 Kubernetes manifests 的目录）

目标集群配置



- **Cluster**：设置为 `https://kubernetes.default.svc`（集群内访问）或选择具体的集群名称
- **Namespace**：设置为 `default`（或指定目标命名空间）

1. 创建 **Application** 完成配置后，点击右上角的 **Create** 按钮以初始化创建 guestbook 应用程序。

通过平台创建 **Argo CD Application**

本文将介绍通过网页控制台的 **GitOps** 应用程序 创建 Argo CD Application 的完整过程，实现对应用程序的 GitOps 管理。

目录

用例

- 使用网页控制台创建一个 SpringBoot Argo CD Application，体验通过 GitOps 管理应用程序的完整过程。

前提条件

- 安装 Alauda Container Platform GitOps
- 项目和命名空间已分配

步骤

按照以下步骤使用这些功能：

代码仓库配置

> 如果在 创建 **GitOps** 应用程序 的详细页面中没有看到集成代码仓库，您可以先集成代码仓库：

- 通过 Argo CD 控制面板集成代码仓库

INFO

如果您没有可用的代码仓库，您可以使用演示仓库进行演示。 仓库 **URL**: `https://github.com/argoproj/argocd-example-apps.git` 描述: 该仓库包含示例应用程序，可用于演示和测试 Argo CD 功能。

使用 GitOps 应用程序创建 Argo CD Application

1. 在 容器平台 中，点击 **GitOps** 应用程序。
2. 点击 创建 **GitOps** 应用程序。
3. 在 基本信息 和 代码仓库 部分，根据以下说明配置相关信息。

参数	输入内容
类型	`Application`
来源	`Argo CD 集成`
集成项目名称	`argocd-example-apps`
版本标识符	`Branch` `master`
源文件类型	`Helm`

参数	输入内容
源文件目录	`helm-guestbook`
自定义值	`values.yaml`

4. 在 分发 中，使用平台推荐的 *命名空间*，或者选择其他命名空间。

5. 将同步策略设置为默认的 手动同步。

6. 点击 创建。

如何获取 **Argo CD** 访问信息

> 本篇内容将介绍如何获取 Argo CD 的访问信息，包括平台上的 **Alauda Container Platform GitOps** 集群插件 Argo CD 和通过 **Alauda Build of Argo CD** Operator 安装的版本。

目录

使用场景

- 获得 Argo CD 访问信息后，您可以通过 Argo CD Dashboard 管理所有原生 Argo CD 资源。

如何获取通过平台部署的 **GitOps** 集群插件的 **Argo CD** 访问信息？

前提条件

- 安装 Alauda Container Platform GitOps
- （可选）已安装 CLI 插件，并开启 `web-cli` 开关
- 您拥有管理员权限

操作步骤

> **INFO**
>
> 建议在安装 **Alauda Container Platform GitOps** 集群插件时启用以下设置：
>
> - 启用 原生 **Argo CD UI** 开关。
> - 启用 单点登录 开关。

> 请按以下步骤，获取 Argo CD 访问信息：

1. 登录平台，进入 平台管理 页面。
2. 左侧导航栏选择 应用商店管理，进入 集群插件 列表页面。
3. 找到 **GitOps** 插件，点击 **GitOps**，弹出 **GitOps** 集群插件 详情信息。

如果未启用：返回 集群插件 列表页面，找到 **GitOps** 插件，点击 操作 按钮，选择 更新，启用 **Argo CD** 原生 **UI** 开关。
如果已启用：直接点击 访问地址，打开 Argo CD Dashboard。

4. Argo CD 原生 UI
   - 如果未启用：返回 集群插件 列表页面，找到 **GitOps** 插件，点击 更新 按钮，启用 **Argo CD** 原生 **UI** 开关。
   - 如果已启用：直接点击 访问地址，打开 Argo CD Dashboard。
5. 单点登录
   - 如果已启用：可通过平台账号登录 Argo CD Dashboard。
   - 如果未启用：默认账号为 `admin`，密码信息需在 **Kubectl** 中执行如下命令获取 获取 Argo CD 密码。

如何获取通过 **Argo CD Operator** 部署的 **Argo CD** 访问信息？

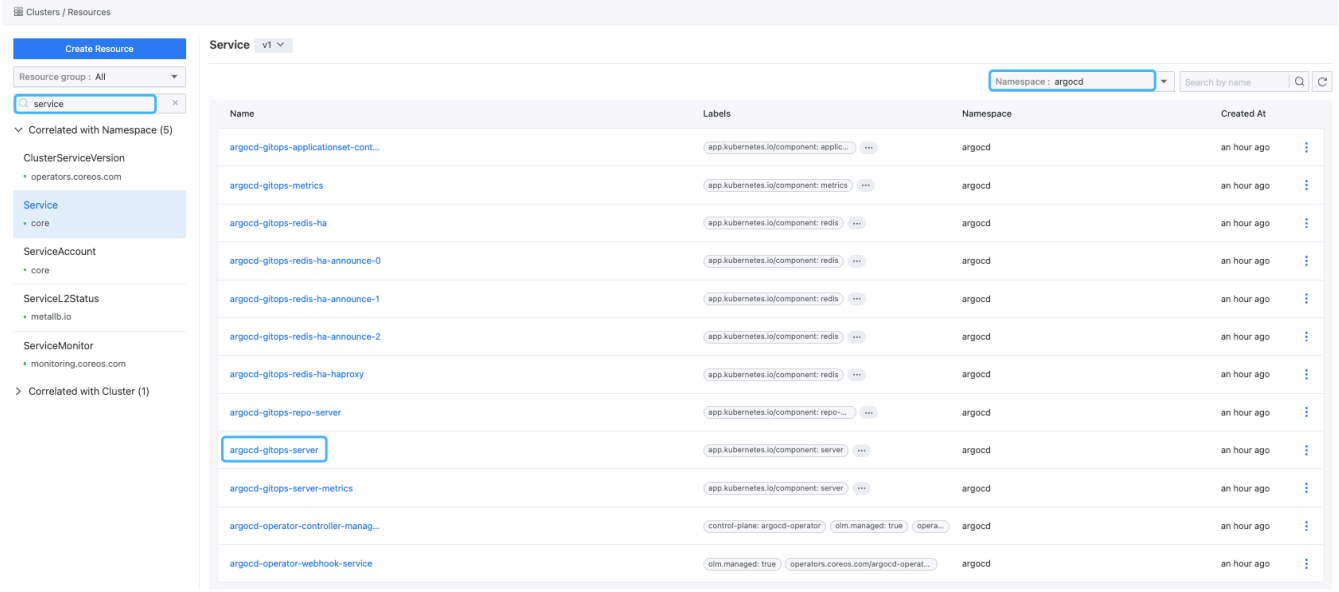
前提条件

- 安装 Argo CD
- （可选）已安装 CLI 插件，并开启 `web-cli` 开关
- 您拥有管理员权限

操作步骤

获取 **Argo CD Dashboard URL**

1. 登录平台，进入 平台管理 页面。
2. 左侧导航栏选择 集群管理，进入 资源管理 页面。
3. 在 资源组 中，搜索 `Service`，选择 ***argocd*** 命名空间（即部署 argocd 实例的命名空间）。通过平台部署的 Argo CD 默认命名空间为 ***argocd***。
4. 在右侧 资源列表 中，找到 `argocd-gitops-server` 资源，点击 *操作* 按钮，选择 更新，以查看 argocd-gitops-server 的 YAML 详情，详见下图：



5. 将 `type` 修改为 `NodePort`，记录下 `nodePort`，然后点击 更新 按钮。

6. 左侧导航栏选择 集群管理，进入 集群列表 页面。

7. 选择部署 `argocd operator` 的集群，进入 集群详情 页面，选择 节点。

8. 获取 *任意控制平面* 节点的 IP 地址。

9. 通过 `http://{控制平面节点 IP}:{nodePort}` 访问 Argo CD Dashboard。

获取 **Argo CD** 密码

在 **Kubectl** 中执行如下命令以获取密码：

```
kubectl get secret -n argocd argocd-gitops-cluster -o template --template='{{
```

更新 **Argo CD admin** 账户密码

通过 **Alauda Container Platform GitOps** 或者 **Alauda Build of Argo CD** Operator 部署的 Argo CD 默认创建的 `admin` 账户密码不能通过 **Argo CD dashboard** 界面修改。如需更改请在 CLI 工具中执行下述命令。其中，`newpassword` 是您希望设置的新密码。

```
kubectl patch -n argocd secrets argocd-gitops-cluster -p '{"stringData":{"adm
```

故障排查

我删除/损坏了我的仓库，无法删除我的应用？

如果 Argo CD 无法生成清单，则无法删除应用。您需要：

1. 恢复/修复您的仓库。
2. 使用 `--cascade=false` 删除应用，然后手动删除资源。

为什么我的应用在成功同步后仍然显示 `OutOfSync`？

请参阅 Diffing Documentation↗ 了解资源可能处于 OutOfSync 状态的原因，以及如何配置 Argo CD 忽略预期差异的字段。

为什么我的应用卡在 `Progressing` 状态？

Argo CD 为多个标准 Kubernetes 类型提供健康检查。`Ingress`、`StatefulSet` 和 `SealedSecret` 类型存在已知问题，可能导致健康检查返回 `Progressing` 状态而不是 `Healthy`。

- 如果 `status.loadBalancer.ingress` 列表非空，并且至少有一个 `hostname` 或 `IP` 的值，则 `Ingress` 被视为健康。一些 ingress 控制器（如 contour、traefik）不会更新 `status.loadBalancer.ingress` 字段，这会导致 `Ingress` 永远卡在 `Progressing` 状态。
- 如果 `status.updatedReplicas` 字段的值与 `spec.replicas` 字段匹配，则 `StatefulSet` 被视为健康。由于 Kubernetes 的 bug kubernetes#68573↗，`status.updatedReplicas` 未被填充。因此，除非您运行的 Kubernetes 版本包含修复 kubernetes#67570↗，否则 StatefulSet 可能会保持在 Progressing 状态。
- 您的 `StatefulSet` 或 `DaemonSet` 使用的是 `OnDelete` 而不是 `RollingUpdate` 策略。
- 有关 `SealedSecret` 的问题，请参见 为什么类型为 `SealedSecret` 的资源卡在 `Progressing` 状态？

作为解决方法，Argo CD 允许提供 健康检查↗ 自定义，以覆盖默认行为。

如果您使用 Traefik 作为 Ingress，您可以更新 Traefik 配置，通过 publishedservice↗ 发布 loadBalancer IP，从而解决此问题。

```
providers:
  kubernetesIngress:
    publishedService:
      enabled: true
```

如何禁用管理员用户？

在 `argocd-cm` ConfigMap 中添加 `admin.enabled: "false"`。

Argo CD 无法在没有互联网访问的情况下部署 Helm Chart 基于的应用，我该如何解决？

如果 Helm Chart 的依赖项位于外部仓库，Argo CD 可能无法生成 Helm Chart 清单。要解决此问题，您需要确保 `requirements.yaml` 仅使用内部可用的 Helm 仓库。即使 Chart 仅使用来自内部仓库的依赖项，Helm 也可能决定刷新 `stable` 仓库。作为解决方法，在 `argocd-cm` 配置映射中覆盖 `stable` 仓库 URL：

```
data:
  repositories: |
    - type: helm
      url: http://<internal-helm-repo-host>:8080
      name: stable
```

在使用 Argo CD 创建 Helm 应用后，我无法通过 helm ls 和其他 Helm 命令查看它？

在部署 Helm 应用时，Argo CD 仅将 Helm 用作模板机制。它运行 `helm template`，然后将生成的清单部署到集群，而不是执行 `helm install`。这意味着您无法使用任何 Helm 命令查看/验证应用。它完全由 Argo CD 管理。请注意，Argo CD 原生支持一些您可能在 Helm 中缺失的功能（例如历史记录和回滚命令）。

做出此决定是为了使 Argo CD 对所有清单生成器保持中立。

我已配置集群密钥，但在 CLI/UI 中未显示，如何修复？

检查集群密钥是否具有标签 `argocd.argoproj.io/secret-type: cluster`。如果密钥具有该标签但集群仍不可见，可能是权限问题。尝试使用 `admin` 用户列出集群（例如：`argocd login --username admin && argocd cluster list`）。

检查集群密钥是否具有 `argocd.argoproj.io/secret-type: cluster` 标签。如果密钥具有该标签但集群仍不可见，则确保可能是权限问题。尝试使用 `admin` 用户列出集群（例如：`argocd login --username admin && argocd cluster list`）。

为什么我的应用在同步后仍然显示 Out Of Sync？

在某些情况下，您使用的工具可能会通过添加 `app.kubernetes.io/instance` 标签与 Argo CD 冲突。例如，使用 Kustomize 的公共标签功能。

Argo CD 会自动设置 `app.kubernetes.io/instance` 标签，并使用它来确定哪些资源构成应用。如果工具也这样做，则会导致混淆。您可以通过在 `argocd-cm` 中设置 `application.instanceLabelKey` 值来更改此标签。我们建议您使用 `argocd.argoproj.io/instance`。

> **INFO**
>
> 当您进行此更改时，您的应用将变为不同步，并需要重新同步。

Argo CD 多久检查一次我的 Git 或 Helm 仓库的更改？

默认轮询间隔为 3 分钟（180 秒），并具有可配置的抖动。您可以通过更新 `argocd-cm` 配置映射中的 `timeout.reconciliation` 值和 `timeout.reconciliation.jitter` 来更改此设置。如果有任何 Git 更改，Argo CD 仅会更新启用了 `auto-sync` 设置的应用。如果您将其设置为 0，则 Argo CD 将停止自动轮询 Git 仓库，您只能使用其他方法（如 `webhooks` 和/或 `手动同步`）来创建应用。

如何修复无效的 cookie，长度超过最大长度 4093？

Argo CD 使用 JWT 作为身份验证令牌。您可能属于多个组，并且超过了为 cookie 设置的 4KB 限制。您可以通过打开“开发者工具 -> 网络”获取组的列表：

1. 点击登录到 Argo CD 仪表板 如何获取 Argo CD 访问信息
2. 找到对 `<argocd_instance>/auth/callback?code=<random_string>` 的调用

在 jwt.io↗ 解码令牌。这将提供您可以移除的团队列表。

为什么在使用 CLI 时我会收到 rpc error: code = Unavailable desc = transport is closing？

您可能在一个不支持 HTTP 2 的代理后面？尝试使用 `--grpc-web` 标志：

```
argocd ... --grpc-web
```

为什么类型为 SealedSecret 的资源卡在 Progressing 状态？

`SealedSecret` 资源的控制器可能会在其提供的资源上公开状态条件。从 `v2.0.0` 版本开始，Argo CD 会获取该状态条件以推导 `SealedSecret` 的健康状态。

在 `SealedSecret` 控制器的 `v0.15.0` 之前的版本受到此状态条件更新问题的影响，因此在这些版本中此功能默认禁用。可以通过使用 `--update-status` 命令行参数启动 `SealedSecret` 控制器或设置 `SEALED_SECRETS_UPDATE_STATUS` 环境变量来启用状态条件更新。

要禁用 Argo CD 检查 `SealedSecret` 资源的状态条件，请通过 `resource.customizations.health.<group_kind>` 键在 `argocd-cm` ConfigMap 中添加以下资源自定义。

resource.customizations.health.bitnami.com_SealedSecret: | hs = hs.status = "Healthy" hs.message = "Controller doesn't report resource status" return hs

如何轮换 Redis 密钥？

- 删除安装 Argo CD 的命名空间中的 `argocd-redis` 密钥。

```
kubectl delete secret argocd-redis -n <argocd namesapce>
```

- 如果您在 HA 模式下运行 Redis，请重启 HA 模式下的 Redis。

```
kubectl rollout restart deployment argocd-redis-ha-haproxy
kubectl rollout restart statefulset argocd-redis-ha-server
```

- 如果您在非 HA 模式下运行 Redis，请重启 Redis。

```
kubectl rollout restart deployment argocd-redis
```

1. 重启其他组件。

```
kubectl rollout restart deployment argocd-server argocd-repo-server
kubectl rollout restart statefulset argocd-application-controller
```

如何修复清单生成错误（缓存）？

`Manifest generation error (cached)` 意味着在生成清单时发生了错误，并且错误消息已被缓存以避免重复重试。

进行强制刷新（忽略缓存错误）可以克服瞬态问题。但是，如果清单生成失败的原因仍然存在，强制刷新将无济于事。

相反，尝试在 repo-server 日志中搜索应用名称，以识别导致清单生成失败的错误。