Menu

GitOps

Введение

Введение

Сценарии использования GitOps

Преимущества GitOps

Преимущества GitOps в Alauda Container Platform

Установка

Установка Alauda Build of Argo CD

Предварительные требования

Процедура

Установка Alauda Container Platform GitOps

Предварительные требования

Установка плагина кластера Alauda Container Platform GitOps

Обновление

Обновление Alauda Container Platform GitOps

Предварительные требования

Обновление плагина кластера Alauda Container Platform GitOps

Архитектура

Архитектура

GitOps и Argo CD

Архитектура GitOps

Архитектура Alauda Container Platform GitOps

Основные понятия

GitOps

Введение

Основные принципы

Преимущества

Популярные инструменты GitOps

Концепция Argo CD

Концепции GitOps в Alauda Container Platform

Руководства

Создание GitOps приложения

Наблюдаемость GitOps

Как сделать

Интеграция репозиториев кода через панель управления Argo CD

Сценарии использования

Предварительные требования

Процедура

Результат операции

Создание приложения Argo CD через панель управления Argo CD

Предварительные требования

Процедура

Создание Argo CD Application через веб-консоль

Сценарии использования

Требования

Процедура

Как получить информацию для доступа к Argo CD

Сценарии использования

Как получить информацию для доступа к Argo CD для плагина кластера GitOps, установленного через веб-консоль?

Как получить информацию для доступа к Argo CD через Argo CD Operator?

Устранение неполадок

Устранение неполадок

Я удалил/повредил репозиторий и не могу удалить приложение?

Почему мое приложение сразу после успешной синхронизации остается в состоянии `OutOfSync`?

Почему мое приложение застряло в состоянии `Progressing`?

Как отключить пользователя admin?

Argo CD не может развернуть приложения на основе Helm Chart без доступа в интернет, как это исправить?

После создания Helm приложения через Argo CD я не вижу его в `helm ls` и других командах Helm?

Я настроил секрет кластера, но он не отображается в CLI/UI, как это исправить?

Почему мое приложение остается Out Of Sync даже после синхронизации?

Как часто Argo CD проверяет изменения в моем Git или Helm репозитории?

Как исправить ошибку invalid cookie, longer than max length 4093?

Почему при использовании CLI я получаю ошибку rpc error: code = Unavailable desc = transport is closing?

Почему ресурсы типа SealedSecret застревают в состоянии Progressing?

Как выполнить ротацию ключей Redis?

Введение

GitOps — это современный подход к непрерывной доставке и эксплуатации, который использует Git в качестве центрального «единого источника правды» (SSOT) для определения и управления инфраструктурой, конфигурациями приложений и процессами развертывания. Объединяя код приложений, конфигурационные файлы и определения Infrastructure as Code (IaC) в одном репозитории Git, GitOps обеспечивает полный контроль версий и автоматизированное управление всем жизненным циклом доставки программного обеспечения. В этой парадигме команды разработки и эксплуатации беспрепятственно сотрудничают на всех этапах разработки, тестирования и развертывания программного обеспечения, используя мощные механизмы ветвления, обзора кода и merge request в Git. Когда изменения в коде или конфигурациях отправляются в репозиторий Git, автоматизированные инструменты обнаруживают эти обновления и запускают цепочку автоматизированных процессов, включая сборку, тестирование и развертывание. Такой рабочий процесс способствует непрерывной доставке и непрерывному развертыванию (CI/CD) программного обеспечения, обеспечивая быстрые и надежные релизы.

Содержание

Сценарии использования GitOps

- **Непрерывная доставка контейнеризованных приложений**: В экосистеме Kubernetes GitOps отлично справляется с управлением развертыванием,

обновлениями и откатами контейнеризованных приложений. Разработчики коммитят код приложений и конфигурационные файлы Kubernetes в репозиторий Git, а инструменты GitOps автоматически развертывают эти приложения в кластере Kubernetes, синхронизируя их с изменениями в конфигурационных файлах.

- **Управление несколькими средами**: GitOps упрощает управление инфраструктурой и конфигурациями приложений в различных средах, таких как разработка, тестирование, staging и продакшн. С помощью стратегического ветвления и тегирования сред обеспечивается согласованность конфигураций при учёте необходимых кастомизаций для каждой среды.
- **Эксплуатация микросервисной архитектуры**: В микросервисных архитектурах GitOps помогает командам эффективно организовывать развертывание и обновления множества микросервисов. Код и конфигурации каждого микросервиса могут храниться отдельно в репозитории Git, что позволяет инструментам GitOps автоматизировать развертывания и обновления с учётом зависимостей микросервисов и стратегий обновления, обеспечивая стабильность системы.
- **Управление Infrastructure as Code (IaC)**: GitOps бесшовно интегрируется с инструментами IaC, такими как Terraform и Ansible, для управления облачной инфраструктурой, конфигурациями серверов и сетевыми ресурсами. Хранение файлов конфигураций IaC в репозитории Git обеспечивает версионирование и автоматизацию развертывания инфраструктуры, повышая управляемость и повторяемость.
- **Межкомандное сотрудничество и совместное использование кода**: В крупных организациях нескольким командам часто необходимо совместно использовать код и конфигурации. GitOps предоставляет единую платформу для совместной работы над разработкой, обмена кодом и управления конфигурациями через репозиторий Git, повышая эффективность сотрудничества и повторное использование кода.

Преимущества GitOps

- Ускоренное сотрудничество и доставка
- Быстрый откат и восстановление
- Управление несколькими средами

- Повышенная безопасность и соответствие требованиям

Подробное введение в преимущества GitOps

Преимущества GitOps в Alauda Container Platform

- Корпоративный оператор Argo CD.
- Сервис безопасности оператора Argo CD.
- Визуальное управление многоокружным распределением GitOps-приложений.
- Визуальная эксплуатация и сопровождение GitOps-приложений.
- Визуальное управление конфигурацией кластера GitOps.
- Замкнутое управление GitOps-приложениями, интегрированное со всеми продуктами платформы.

Подробное введение в преимущества GitOps в Alauda Container Platform

Menu

Установка

Установка Alauda Build of Argo CD

Предварительные требования

Процедура

Установка Alauda Container Platform GitOps

Предварительные требования

Установка плагина кластера Alauda Container Platform GitOps

Установка Alauda Build of Argo CD

Содержание

Предварительные требования

1. **Скачайте** пакет установки оператора **Alauda Build of Argo CD**, соответствующий архитектуре вашей платформы.
2. **Загрузите** пакет установки с помощью механизма Upload Packages.

Процедура

> Устанавливайте в кластер, где вы планируете использовать функциональность GitOps.

Установка оператора Alauda Build of Argo CD

1. Войдите в систему и перейдите на страницу **Administrator**.
2. Нажмите **Marketplace** > **OperatorHub**, чтобы перейти на страницу **OperatorHub**.

3. Найдите оператор **Alauda Build of Argo CD**, нажмите **Install** и перейдите на страницу **Install Argo CD**.

Параметры конфигурации:

Параметр	Рекомендуемая конфигурация
Channel	Канал по умолчанию — `alpha`.
Installation Mode	`Cluster`: Все пространства имён в кластере используют один экземпляр оператора для создания и управления, что снижает использование ресурсов.
Namespace	Выберите `Recommended Namespace`: создаётся автоматически, если отсутствует.
Upgrade Strategy	`Auto`: OperatorHub автоматически обновит оператор до последней версии при её доступности.

1. Рекомендуется использовать предложенную конфигурацию по умолчанию; просто нажмите **Install** для завершения установки оператора **Alauda Build of Argo CD**.

Создание экземпляра Argo CD

1. Нажмите **Marketplace** > **OperatorHub**.
2. Найдите оператор **Alauda Build of Argo CD**, кликните по нему, чтобы перейти на страницу с подробной информацией об **Argo CD**.
3. Нажмите **All Instances**,
4. Нажмите **Create Instance**, выберите карточку экземпляра **Argo CD**.
5. Нажмите **Create Instance**

INFO

На странице параметров конфигурации экземпляра используйте конфигурацию по умолчанию, если нет специальных требований. **Примечание**: Если глобальный кластер не является высокодоступным (например, содержит только один управляющий узел),

переключитесь в YAML-режим при создании экземпляра и установите значение поля ha.enabled в false.

1. Нажмите **Create**.

Создание экземпляра AppProject

INFO

Совет: Если вам не нужна функция платформенного управления **Cluster Configuration Management**, следующие шаги выполнять не нужно.

1. Найдите оператор **Alauda Build of Argo CD**, кликните по нему, чтобы перейти на страницу с подробной информацией о **Alauda Argo CD**.
2. Нажмите **All Instances**, затем **Create Instance**, выберите карточку экземпляра **AppProject**.
3. Переключитесь в YAML-режим и замените существующее содержимое YAML на интерфейсе следующим кодом:

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

4. Нажмите **Create**.

После выполнения вышеуказанных шагов вы успешно установили Argo CD. Немедленно приступайте к созданию приложения Argo CD через панель Argo CD, чтобы начать работу с GitOps.

Установка Alauda Container Platform GitOps

Содержание

Предварительные требования

1. **Скачайте** установочный пакет плагина кластера **Alauda Container Platform GitOps**, соответствующий архитектуре вашей платформы.
2. **Загрузите** установочный пакет с помощью механизма Upload Packages.
3. **Установите** пакет в кластер `global` с помощью механизма cluster plugins.

> **INFO**
>
> Upload Packages: Страница **Administrator** > **Marketplace** > **Upload Packages**. Нажмите **Help Document** справа, чтобы получить инструкции по публикации плагина кластера в кластер `global`. Для получения дополнительной информации обратитесь к разделу CLI.

Установка плагина кластера Alauda Container Platform GitOps

Ограничения и условия

- Поддерживается установка только в кластер `global`.
- После установки плагина экземпляр ArgoCD в `argocd-operator` будет ограничен в операциях.

Процедура

1. Войдите в систему и перейдите на страницу **Administrator**.

2. Нажмите **Marketplace** > **Cluster Plugins**, чтобы открыть страницу списка **Cluster Plugins**.

3. Найдите плагин кластера **GitOps**, нажмите **Install** и перейдите на страницу **Install GitOps Plugin**.

4. Рекомендуется использовать предложенную конфигурацию по умолчанию; просто нажмите **Install** для завершения установки плагина кластера **Alauda Container Platform GitOps**.

Описание параметров:

Параметр	Описание
Native Argo CD UI	Выберите, нужно ли использовать панель управления, предоставляемую Argo CD. Этот интерфейс включает функции мониторинга, управления репозиториями и настройки, и может использоваться для управления и мониторинга созданных приложений.
Single Sign-On	Рекомендуется включить SSO, что позволяет быстро получить доступ к нативному UI Argo CD с использованием учетных данных платформы, улучшая опыт входа, а также повышая безопасность и удобство.

Параметр	Описание
	Примечание: функция SSO требует включения нативного UI Argo CD. • Поддерживается только доступ через HTTPS; при доступе через HTTP SSO работать не будет. • После включения SSO и открытия интерфейса входа Argo CD по адресу доступа, нажмите кнопку **LOG IN VIA OIDC** для однократного входа в нативный UI Argo CD.
Access Address	Рекомендуется: этот адрес динамически генерируется на основе адреса платформы для доступа к панели Argo CD. Ввод вручную не требуется.
Account	Учетная запись, используемая для входа и доступа к нативному UI Argo CD.
Password	После включения доступа к нативному UI Argo CD вы можете выполнить следующую команду в CLI инструмента глобального кластера для получения пароля. Получение информации для доступа к Argo CD
Resource Quota	Минимальные требования и рекомендации для платформы следующие: • **Минимум**: запросы CPU не должны быть меньше 100 m, запросы памяти не должны быть меньше 250 Mi, при этом значения запросов не должны превышать значения лимитов. • **Рекомендуется**: запросы CPU не должны быть меньше 250 m, запросы памяти не должны быть меньше 500 Mi; лимиты CPU не должны быть меньше 2 ядер, лимиты памяти не должны быть меньше 2 Gi.

Проверка

1. На странице **Administrator** в разделе **Cluster** в левой навигации появится пункт **Config**. Вы можете использовать возможности управления конфигурацией кластера.

2. Перейдите в **Container Platform**, в левой навигации появится пункт **GitOps Applications**, где можно создать GitOps-приложение и сразу же ознакомиться с Созданием приложения Argo CD через веб-консоль.

Menu

Обновление

Обновление Alauda Container Platform GitOps

Предварительные требования

Обновление плагина кластера Alauda Container Platform GitOps

Обновление Alauda Container Platform GitOps

Содержание

Предварительные требования

1. **Скачайте** установочный пакет плагина кластера **Alauda Container Platform GitOps**, соответствующий архитектуре вашей платформы.
2. **Загрузите** установочный пакет **Alauda Container Platform GitOps** с помощью механизма Upload Packages.
3. **Установите** плагин кластера **Alauda Container Platform GitOps** в кластер `global` с помощью механизма cluster plugins.

> **INFO**
>
> Upload Packages: **Administrator** > **Marketplace** > страница **Upload Packages**. Нажмите **Help Document** справа, чтобы получить инструкции по публикации плагина кластера в кластер `global`. Для получения дополнительной информации обратитесь к CLI.

Обновление плагина кластера Alauda Container Platform GitOps

Ограничения и условия

- Поддерживается обновление только в кластере `global`.

Процедура

1. Войдите в систему, перейдите на страницу **Administrator**.

2. Нажмите **Clusters** > **Clusters** > **global** > **Functional Components**, чтобы перейти на страницу списка компонентов.

3. Нажмите кнопку **Upgrade** и выберите новую версию **Alauda Container Platform GitOps** в качестве целевой на странице **Confirm Component Upgrade**.

4. Снова нажмите **Upgrade**, а затем подтвердите обновление в диалоговом окне, нажав **Upgrade**.

5. (Следующие шаги требуются только при обновлении с версии `ACP 3.16.0`). Вернитесь на страницу кластера `global`, нажмите **Actions** > **CLI Tools**, чтобы открыть окно CLI.

6. В окне CLI введите `kubectl delete cm -n argocd argocd-redis-ha-configmap` для пересоздания `argocd-redis-ha-configmap`.

7. В окне CLI введите `kubectl get cm -n argocd argocd-redis-ha-configmap`, чтобы убедиться, что configmap создан.

Проверка

1. На странице **Administrator** в разделе **Cluster** в левой навигации появится пункт **Config**. Вы сможете использовать возможности управления конфигурацией кластера.

2. Перейдите в **Container Platform**, в левой навигации отобразится пункт **GitOps Applications**, где вы можете создать GitOps-приложение и сразу же ознакомиться с Созданием приложения Argo CD через веб-консоль.

Архитектура

Содержание

GitOps и Argo CD

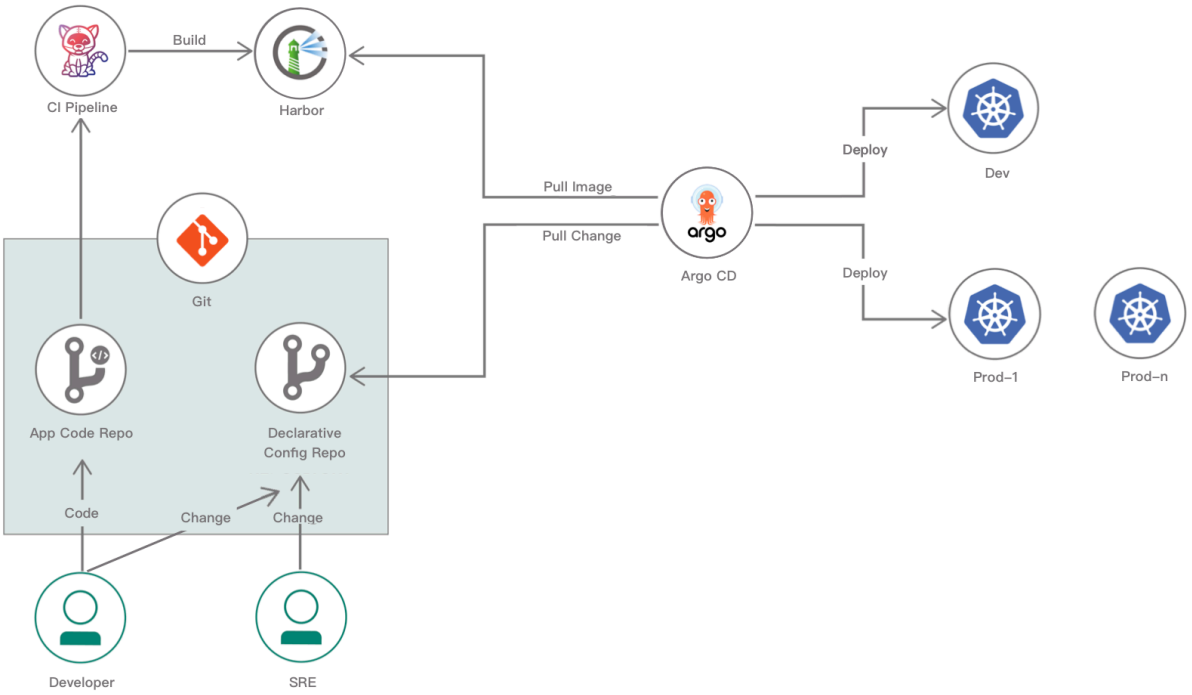
GitOps — это современная концепция для непрерывной доставки и эксплуатации, а Argo CD — мощный инструмент, реализующий GitOps путем мониторинга файлов конфигурации в Git-репозитории и автоматической синхронизации их с целевой средой. Такой подход повышает скорость, надежность и безопасность поставки программного обеспечения, интегрируя весь процесс доставки в систему контроля версий Git.

Alauda Container Platform GitOps, построенный на базе Argo CD, использует Git-репозиторий как единственный надежный источник для хранения приложений, конфигураций инфраструктуры и других файлов, обеспечивая быструю и точную доставку и развертывание в одном или нескольких кластерах Kubernetes.

Архитектура GitOps

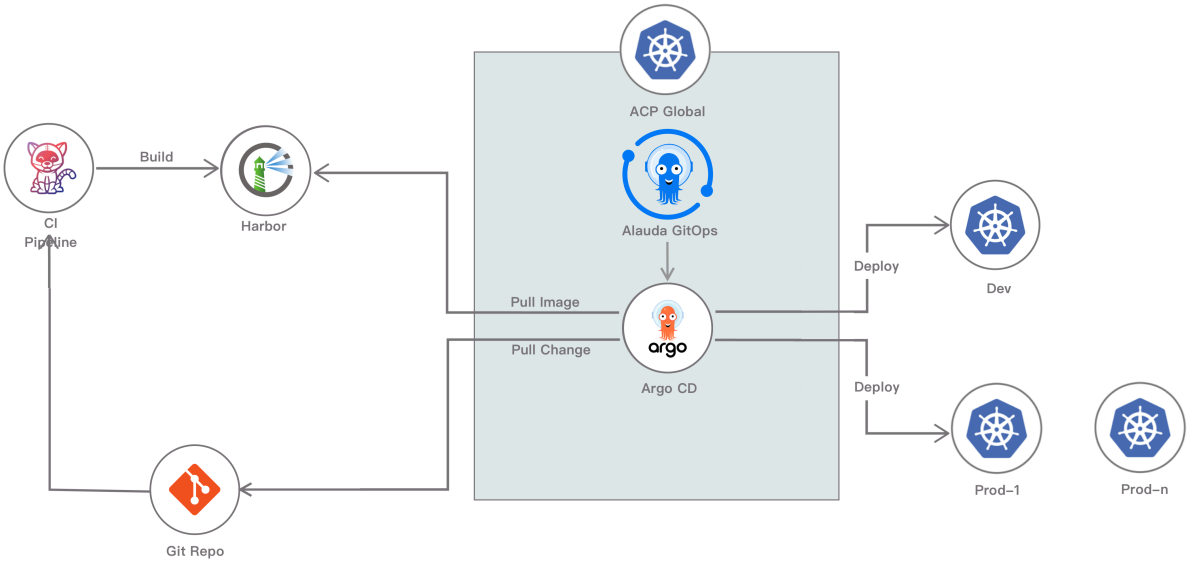
Основные отличия GitOps от традиционных методов управления приложениями заключаются в следующем:

- Вместо прямого управления средой выполнения, GitOps контролирует её через поддержание репозитория конфигураций приложений в Git.
- Argo CD постоянно опрашивает репозиторий и исправляет расхождения между средой выполнения и репозиторием конфигураций приложений, гарантируя соответствие среды ожиданиям, предотвращая дрейф конфигураций и обеспечивая быструю восстановимость в случае сбоев.



Архитектура Alauda Container Platform GitOps

Alauda Container Platform GitOps устанавливается как плагин кластера на `global` кластере и использует Argo CD для распределения приложений и обеспечения инфраструктуры в нескольких бизнес-кластерах.


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

Menu

Основные понятия

GitOps

GitOps

Введение

Основные принципы

Преимущества

Популярные инструменты GitOps

Концепция Argo CD

Введение

Сводка различий между Application и ApplicationSet

Статусы синхронизации Argo CD

Ссылки

Application

Введение

Сценарии использования Application

Пример Application

Справка

ApplicationSet

Введение

Сценарии использования ApplicationSet

Пример ApplicationSet

Ссылки

Tool

Введение

Поддерживаемые инструменты

Рабочий процесс разработки

Сравнение возможностей

Ссылки

Helm

Введение

Основные понятия Helm

Преимущества

Сценарии использования

Kustomize

Введение

Основные концепции Kustomize

Преимущества

Сценарии использования

Directory

Введение

Преимущества

Сценарии использования

Sync

Sync Overview

Sync Status Overview

Sync operation status Overview

Refresh Overview

References

Health

Введение

Область применения Health

Справка

Концепции GitOps в Alauda Container Platform

Введение

Почему Argo CD?

Преимущества

Alauda Container Platform GitOps Sync and Health Status

GitOps

Содержание

Введение

GitOps — это практика использования Git-репозитория в качестве авторитетного источника для конфигураций инфраструктуры и приложений. Все операционные изменения контролируются версиями, автоматизируются и подлежат аудиту через Git. Она основывается на декларативных конфигурациях, хранящихся в Git, где любые изменения должны быть зафиксированы коммитом для запуска автоматизированных процессов развертывания.

Основные принципы

- **Декларативная конфигурация**: GitOps в своей основе требует декларативных инструментов, рассматривая Git как единственный источник правды. Это обеспечивает последовательное развертывание приложений в Kubernetes-кластерах и платформонезависимое восстановление в случае сбоев.
- **Версионированное и неизменяемое состояние**: версии инфраструктуры и приложений напрямую соответствуют коммитам в Git. Откаты выполняются с

помощью `git revert`, что гарантирует неизменяемую историю версий.

- **Автоматическое согласование**: объединённые декларативные состояния автоматически применяются к кластерам. Это исключает ручное вмешательство, предотвращает ошибки человека и поддерживает утверждения безопасности в рабочих процессах развертывания.
- **Самовосстановление**: контроллеры (например, Argo CD) непрерывно согласуют состояние кластера с состоянием, определённым в Git, обеспечивая автономное восстановление системы.

Преимущества

- **Ускоренное сотрудничество и доставка**: декларативные определения инфраструктуры, конфигураций и целевых состояний, хранящиеся в Git, позволяют автоматизировать развертывания. Команды получают возможность однокликового создания окружений после валидации, что упрощает сотрудничество и доставку.
- **Быстрый откат и восстановление**: используя контроль версий Git, аномалии вызывают мгновенные откаты. Контроллеры GitOps обеспечивают самовосстановление через автоматическое согласование.
- **Управление многими окружениями**: Git как единственный источник правды в сочетании с наложениями конфигураций позволяет точно выполнять массовые развертывания в гибридных и мультиоблачных средах.
- **Повышенная безопасность и соответствие требованиям**: RBAC Git, журналы аудита, защита веток и шифрование обеспечивают безопасность конфиденциальных конфигураций и соответствие нормативам.

Популярные инструменты GitOps

- **Argo CD**: нативный для Kubernetes декларативный инструмент GitOps для определения, версионирования и автоматизации жизненного цикла приложений с возможностью аудита.

- **Flux**: лёгкий оператор GitOps для Kubernetes, который непрерывно синхронизирует Git-репозитории с кластерами.
- **Jenkins X**: платформа CI/CD с интеграцией GitOps для автоматизированных пайплайнов и развертываний, управляемых через Git.

Menu

Концепция Argo CD

Введение

Сводка различий между Application и ApplicationSet

Статусы синхронизации Argo CD

Ссылки

Application

Введение

Сценарии использования Application

Пример Application

Справка

ApplicationSet

Введение

Сценарии использования ApplicationSet

Пример ApplicationSet

Ссылки

Tool

Введение

Поддерживаемые инструменты

Рабочий процесс разработки

Сравнение возможностей

Ссылки

Helm

Введение

Основные понятия Helm

Преимущества

Сценарии использования

Kustomize

Введение

Основные концепции Kustomize

Преимущества

Сценарии использования

Directory

Введение

Преимущества

Сценарии использования

Sync

Sync Overview

Sync Status Overview

Sync operation status Overview

Refresh Overview

References

Health

Введение

Область применения Health

Справка

Введение

Argo CD — это очень популярный open-source инструмент GitOps. Для использования Argo CD необходимо понимать следующие основные концепции:

1. Application: группа ресурсов Kubernetes, определённых манифестом. Это Custom Resource Definition (CRD). Application
2. ApplicationSet: контроллер Kubernetes, поддерживающий CRD ApplicationSet, позволяющий массово создавать Applications на основе одного шаблона. Можно рассматривать его как фабрику приложений, создающую экземпляры на основе параметров. ApplicationSet
3. Tool: указывает инструмент управления конфигурацией для источников Application (например, Kustomize, Helm). Tool
4. Sync: процесс согласования текущего состояния приложения с желаемым состоянием (например, применение изменений в кластерах Kubernetes). Sync
5. Health: показывает рабочее состояние приложения, включая готовность и способность обрабатывать запросы. Health

Содержание

Сводка различий между Application и ApplicationSet

Атрибут	Application	ApplicationSet
Определение	Развёртывание одного приложения	Шаблон для генерации нескольких экземпляров Application
Конфигурация	Статические YAML определения	Динамическое создание шаблонов на основе параметров
Развёртывание	Одно приложение	Несколько похожих приложений
Сценарии использования	Простые развёртывания в одной среде	Сложные развёртывания в нескольких средах/кластерах с параметрическими экземплярами
Основные концепции	Git Repo, целевой кластер, стратегии развёртывания	Генераторы, шаблоны, параметры, заполнители
Роль в Argo CD	Основная единица развёртывания	Расширенный уровень массового управления

Статусы синхронизации Argo CD

Статус Sync	Описание
Synced	Текущее состояние приложения полностью соответствует желаемому состоянию.
OutOfSync	Текущее состояние отличается от желаемого; требуется синхронизация.
Syncing	Активный процесс синхронизации; текущее состояние приближается к желаемому.

Ссылки

- Argo CD Official Documentation ↗

Application

Содержание

Введение

Application — это группа ресурсов Kubernetes, определяемая манифестом. Это Custom Resource Definition (CRD).

Сценарии использования Application

- Развертывание одного компонента: используйте Application CRD для декларативного управления развертыванием атомарных рабочих нагрузок в пределах одного namespace.
- Статическая конфигурация: идеально подходит для приложений с детерминированными манифестами, которые не требуют динамического шаблонирования или вариаций для нескольких окружений.
- Развертывание в одном кластере: целевое развертывание в отдельных кластерах Kubernetes через GitOps-процессы.

Пример Application

```
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\\.alpha\\.kubernetes\\.io/hostname"
        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: 'Example:'
      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
```

Справка

- Argo CD Official Documentation ↗

ApplicationSet

Содержание

Введение

Контроллер ApplicationSet — это контроллер Kubernetes, который добавляет поддержку CustomResourceDefinition (CRD) ApplicationSet. Этот контроллер/CRD обеспечивает как автоматизацию, так и большую гибкость в управлении Argo CD Applications на большом количестве кластеров и внутри монорепозиториев, а также позволяет реализовать самообслуживание в многопользовательских Kubernetes кластерах.

Сценарии использования ApplicationSet

- Развертывание нескольких похожих приложений: когда необходимо развернуть несколько приложений с похожими конфигурациями, можно использовать ApplicationSet для сокращения избыточных конфигураций. Например, можно использовать ApplicationSet для развертывания нескольких микросервисов, которые используют один и тот же шаблон, но имеют разные имена сервисов и номера портов.

- Мультикластерное развертывание: когда нужно развернуть одно и то же приложение в нескольких Kubernetes кластерах, ApplicationSet упрощает конфигурацию. Например, можно определить приложение с помощью ApplicationSet и развернуть его в нескольких кластерах, каждый из которых использует разные параметры.
- Динамическая генерация приложений: когда необходимо динамически создавать приложения на основе определённых условий, можно использовать ApplicationSet. Например, можно динамически создавать разные экземпляры приложения на основе веток или тегов в Git-репозитории.

Пример ApplicationSet

```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
```

Ссылки

- Argo CD ApplicationSet Documentation ↗

Tool

Содержание

Введение

Tool — это утилита, используемая для генерации или обработки Kubernetes ресурсов `Manifests`.

Поддерживаемые инструменты

Argo CD поддерживает несколько подходов к определению манифестов Kubernetes:

- **Kustomize Applications** Kustomize
- **Helm Charts** Helm
- **Directory**: Манифесты, содержащие файлы `YAML` / `JSON` / `Jsonnet`, включая `Jsonnet` Directory
- **Custom Configuration Management Plugins**: Любой пользовательский инструмент, настроенный как Config Management Plugin

Рабочий процесс разработки

Argo CD позволяет напрямую загружать локальные `manifests`, но это предназначено **только для целей разработки**. Переопределение требует пользователей с соответствующими правами (обычно администраторов) для загрузки локальных `manifests`. Поддерживаются все вышеупомянутые инструменты развертывания Kubernetes. Чтобы загрузить локальное приложение:

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

Сравнение возможностей

Функция	Helm	Kustomize	Directory (Pure YAML)
Метод конфигурации	Шаблонизация (динамическая генерация)	Декларативный (патчи и оверлеи)	Статические YAML файлы
Повторное использование	Высокое (через Charts)	Среднее (через base/overlay)	Низкое
Поддержка нескольких сред	Высокая (через values.yaml)	Высокая (через оверлеи)	Низкая
Постепенная доставка	Высокая (поддержка сложной логики)	Средняя (поддержка простых патчей)	Низкая
Кривая обучения	Высокая (синтаксис шаблонов)	Низкая (на основе YAML)	Низкая

Функция	Helm	Kustomize	Directory (Pure YAML)
Интеграция с Argo CD	Поддерживается	Родная поддержка	Поддерживается
Сценарии использования	Сложные приложения, мульти-среды, дистрибуция	Мульти-среды, повторное использование конфигураций	Небольшие проекты, быстрая прототипизация

Ссылки

Для более подробной информации, пожалуйста, обратитесь к: Tool ↗

Helm

Содержание

Введение

Helm — это инструмент управления пакетами для Kubernetes, позволяющий пользователям определять, устанавливать и обновлять сложные Kubernetes-приложения. **Helm Chart** — это шаблонизированный пакет конфигураций, содержащий определения ресурсов Kubernetes (YAML-файлы).

Основные понятия Helm

- **Chart**: Helm Chart — это шаблонизированный пакет конфигураций, содержащий определения ресурсов Kubernetes (YAML-файлы).
- **Release**: Helm Release — это экземпляр развернутого Helm Chart, представляющий конкретную конфигурацию ресурсов Kubernetes.
- **Values**: Helm Values — это параметризованные конфигурации для Helm Chart, позволяющие пользователям настраивать определения ресурсов Kubernetes.

Интеграция Argo CD с Helm улучшает практики GitOps, обеспечивая декларативную непрерывную доставку через веб-консоль, панель управления Argo CD или CLI. Пример:

```
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
```

Пример OCI Helm Chart:

```
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 not
included.
    targetRevision: 15.9.0
  destination:
    name: "in-cluster"
    namespace: nginx
```

INFO

** Жизненный цикл Application управляется Argo CD, а не Helm. ** При наличии нескольких источников значений порядок приоритета следующий: `parameters` > `valuesObject` > `values` > `valueFiles` > helm repository `values.yaml`.

Преимущества

- **Шаблонизация**: Helm использует движок шаблонов Go (gotpl) для динамической генерации файлов ресурсов Kubernetes.
- **Управление пакетами**: Helm упаковывает приложения в Charts (включая шаблоны, значения по умолчанию и зависимости), упрощая распространение и контроль версий.
- **Управление зависимостями**: Поддерживает зависимости между Charts.
- **Управление жизненным циклом**: Предоставляет команды для установки, обновления и отката для полного управления жизненным циклом.

Сценарии использования

- **Развертывание сложных приложений**: Идеально подходит для сценариев, требующих динамической генерации конфигураций (например, переменные окружения или ввод пользователя).
- **Развертывания в нескольких средах**: Поддерживает конфигурации, специфичные для среды, через файлы values.yaml.
- **Распространение приложений**: Позволяет упаковывать Charts для распространения в Helm-репозитории или OCI-реестры.

References

For more detailed information, please refer to: Helm ↗

Kustomize

Содержание

Введение

Kustomize — это нативный для Kubernetes инструмент управления конфигурациями, который позволяет пользователям настраивать определения ресурсов Kubernetes (файлы YAML) с помощью оверлеев и композиции без прямого изменения исходных файлов.

Основные концепции Kustomize

- **Base**: Базовые конфигурации, содержащие общие определения ресурсов Kubernetes.
- **Overlay**: Слои кастомизации, которые изменяют базовые конфигурации.
- **kustomization.yaml**: Файл конфигурации, определяющий, как ресурсы компонуются и модифицируются.

Интеграция Argo CD с Kustomize улучшает практики GitOps, обеспечивая декларативную непрерывную доставку. Пример:

```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'
```

Если файл `kustomization.yaml` существует по адресу `repoURL` и в каталоге `path`, Argo CD отрендерит манифесты с использованием Kustomize.

Kustomize поддерживает следующие параметры конфигурации:

- `namePrefix`: Префикс, добавляемый к именам ресурсов, сгенерированных Kustomize.
- `nameSuffix`: Суффикс, добавляемый к именам ресурсов, сгенерированных Kustomize.
- `images`: Список переопределений образов Kustomize.
- `replicas`: Список переопределений количества реплик Kustomize.
- `commonLabels`: Карта меток, добавляемых ко всем ресурсам.
- `labelWithoutSelector`: Булево значение, определяющее, должны ли общие метки применяться к селекторам и шаблонам ресурсов.
- `forceCommonLabels`: Булево значение, позволяющее переопределять существующие метки.
- `commonAnnotations`: Карта аннотаций, добавляемых ко всем ресурсам.
- `namespace`: Пространство имён Kubernetes для ресурсов.
- `forceCommonAnnotations`: Булево значение, позволяющее переопределять существующие аннотации.
- `commonAnnotationsEnvsubst`: Булево значение, включающее подстановку переменных окружения в значениях аннотаций.
- `patches`: Список патчей Kustomize, поддерживающих встроенные обновления.

- `components`: Список компонентов Kustomize.

Для использования Kustomize с оверлеями укажите путь к каталогу оверлея.

Преимущества

- Декларативная конфигурация: Использует YAML-файлы (через kustomization.yaml) для определения композиции и изменений ресурсов.
- Без шаблонов: Настраивает конфигурации с помощью патчей и оверлеев без использования шаблонных движков.
- Нативная интеграция с Kubernetes: Kustomize встроен в kubectl и не требует дополнительных инструментов.

Сценарии использования

- Распределение по нескольким средам: Позволяет создавать конфигурации, специфичные для среды (например, приложения, кластеры) с помощью Base и Overlay.
- Повторное использование конфигураций: Идеально подходит для повторного использования базовых конфигураций в разных проектах.
- Прогрессивная доставка: Постепенно изменяет конфигурации ресурсов с помощью патчей.

References

For more detailed information, please refer to: Kustomize ↗

Directory

Содержание

Введение

Приложение типа **Directory** загружает `manifests` напрямую из файлов `.yml`, `.yaml` или `.json`. Приложения Directory могут быть созданы через UI платформы, Argo CD Dashboard, CLI или декларативно. Пример декларативного синтаксиса:

```
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
```

Поле spec.source.directory не требуется, если не нужны дополнительные параметры конфигурации. Argo CD автоматически определяет, содержит ли репозиторий/путь исходных данных обычные манифесты.

Преимущества

- **Простота**: Ресурсы загружаются напрямую из файлов манифестов без дополнительного уровня абстракции.
- **Низкие затраты на поддержку**: Отсутствует необходимость управления конфигурацией.

Сценарии использования

- Управление множеством ресурсов Kubernetes (например, Deployments, Services, ConfigMaps).
- Небольшие проекты, минимальное количество ресурсов или быстрая адаптация GitOps.
- Развертывание сырых YAML-файлов без динамического шаблонизирования или сложного управления конфигурацией.

> **WARNING**
>
> Приложения типа Directory **поддерживают только обычные файлы манифестов**. Если Argo CD обнаружит файлы `Kustomize`, `Helm` или `Jsonnet` в пути Directory, рендеринг манифестов завершится с ошибкой.

References

For more detailed instructions, refer to: Directory ↗

Sync

Содержание

Sync Overview

Sync — это основная функция Argo CD, отвечающая за сравнение **Desired State** приложения с его **Live State** и выполнение действий для устранения расхождений. По сути, Sync гарантирует, что состояние приложений в вашем Kubernetes кластере соответствует состоянию, определённому в Git репозитории.
Вы можете запускать Sync вручную или настроить Argo CD для автоматического выполнения. Авто-Sync может запускаться при отслеживании изменений в Git репозитории (например, коммиты, пуши тегов) или выполняться по расписанию.

Sync Status Overview

Sync Status показывает состояние синхронизации приложения, отражая, совпадает ли его **Live State** с **Desired State**. Sync Status включает следующие состояния:

- `Synced`: **Live State** приложения полностью соответствует **Desired State**.

- `OutOfSync` : **Live State** приложения отличается от **Desired State**.
- `Syncing` : Приложение находится в процессе синхронизации, и **Live State** приближается к **Desired State**.

Sync operation status Overview

Sync Operation Status отображает состояние выполнения операции синхронизации в Argo CD, показывая, успешно ли завершилась операция. Статусы операции синхронизации включают:

- `Succeeded` : Операция синхронизации успешно завершена.
- `Failed` : Операция синхронизации завершилась с ошибкой по причинам, таким как конфликты ресурсов Kubernetes, недостаточные права и т.д.
- `Running` : Операция синхронизации выполняется.

Refresh Overview

Эта операция получает последнюю конфигурацию приложения из Git репозитория и сравнивает её с текущим состоянием в Kubernetes кластере. Refresh можно запускать вручную или настроить для автоматического выполнения с заданным интервалом.

References

For more detailed information, please refer to: Sync ↗

Health

Содержание

Введение

Состояние здоровья приложения: работает ли оно корректно? Может ли оно обслуживать запросы?

Область применения Health

Статус здоровья	**Описание**
Health	Ресурс здоров.
Progressing	Ресурс пока не здоров, но прогрессирует и может скоро стать здоровым.
Degraded	Ресурс находится в деградированном состоянии.

Статус здоровья	Описание
Suspended	Ресурс приостановлен и ожидает внешнего события для возобновления (например, приостановленный CronJob или пауза Deployment).

Справка

Argo CD предоставляет встроенную оценку состояния здоровья для нескольких стандартных типов Kubernetes, которая затем отображается в общем статусе здоровья Application. Конечно, Argo CD также поддерживает пользовательские проверки здоровья.

Для более подробных объяснений, пожалуйста, обратитесь к: Health ↗

Menu

Концепции GitOps в Alauda Container Platform

Введение

Alauda Container Platform GitOps — это Kubernetes-нативное GitOps-решение, построенное на базе Argo CD. Оно отслеживает конфигурационные манифесты (приложения, определения инфраструктуры и т.д.) в Git-репозиториях и автоматически синхронизирует их с целевыми Kubernetes-кластерами, реализуя Git-управляемую непрерывную доставку. Кодируя весь конвейер доставки в системе контроля версий Git, решение повышает скорость развертывания, надежность и безопасность, а также обеспечивает точное распределение приложений по нескольким кластерам.

Решение нативно интегрирует Argo CD Operator для автоматизации операций жизненного цикла развертывания, включая подготовку, обновления и откаты.

Содержание

Почему Argo CD?

Argo CD является ведущим в отрасли open-source GitOps-движком благодаря своим уникальным преимуществам:

Технические преимущества	Операционные выгоды
Декларативный GitOps-движок • Согласование состояния через CRD (Application, ApplicationSet)	**Ускорение развертывания** • Циклы развертывания на 70% быстрее благодаря Git-

Технические преимущества	Операционные выгоды
• Поддержка нескольких источников (Helm, Kustomize, raw YAML)	автоматизации
Kubernetes-нативная архитектура • Глубокая интеграция с Kubernetes API server ≈ Нативная поддержка изоляции Namespace и RBAC	**Готовность для предприятий** • Встроенная мультиарендность и возможности аудита
Управление мультикластером • Централизованная контрольная плоскость для гибридных/ мультиоблачных развертываний • Конфигурация, специфичная для кластера, через ApplicationSets	**Операционная эффективность** • Снижение ошибок развертывания на 60% благодаря декларативному контролю
Расширяемая система плагинов • Сертифицированные интеграции с Helm, Kustomize, Istio • Custom Resource Definitions (CRD) для продвинутых рабочих процессов	**Оптимизация затрат** • Снижение облачных расходов на 40% за счет точной оркестрации ресурсов
Активная экосистема CNCF • Более 3500 звезд на GitHub • Более 200 активных участников	**Готовность к будущему** • Непрерывные инновации благодаря сообществу с открытым исходным кодом

Преимущества

> Помимо базовых преимуществ GitOps, Alauda Container Platform GitOps предлагает следующие расширенные возможности:

- **Корпоративный Argo CD Operator**
  - Обеспечивает полный набор функций нативного Argo CD Operator, включая развертывание приложений, обновления, откаты и все основные возможности Argo CD.
- **Служба безопасности Argo CD Operator**
  - Предоставляет специализированную техническую поддержку для Argo CD Operator, включая реагирование на сбои, исправления уязвимостей безопасности и обеспечение общей стабильности системы.
- **Визуальное управление многоокружной дистрибуцией GitOps-приложений**
  - Использует возможности платформы по управлению мультикластером и дифференцированной конфигурации для достижения стильного, визуального управления GitOps-приложениями и конфигурациями кластеров, упрощая точное распределение в мультиоблачных и многоокружных средах.
- **Визуальная эксплуатация и сопровождение GitOps-приложений**
  - Предоставляет прямой доступ к логам и событиям Kubernetes Workload-ресурсов в реальном времени под управлением GitOps-приложений. При возникновении аномалий в GitOps-приложениях пользователи могут быстро анализировать и устранять проблемы, используя информацию об аномалиях Argo CD и логи Workload в реальном времени, не покидая текущий интерфейс.
- **Визуальное управление конфигурациями кластеров GitOps**
  - Управляет конфигурациями кластеров через GitOps, обеспечивая единое визуальное управление и дистрибуцию конфигураций кластеров.
- **Замкнутая цепочка управления GitOps-приложениями с интеграцией всех продуктов платформы**
  - Интегрирует возможности DevOps платформы для непрерывной сборки, управления артефактами и серого релиза микросервисов, реализуя полностью автоматизированное управление GitOps-приложениями и формируя полный замкнутый цикл сотрудничества CI/CD.

- Взаимодействует с продуктами платформы, такими как CrossPlane, MySQL и Developer Portal, обеспечивая комплексный процесс от инициализации инфраструктуры, инициализации бизнес-приложений (включая код, pipeline, инициализацию GitOps-приложений и т.д.) до разработки и развертывания кода приложений.

Alauda Container Platform GitOps Sync and Health Status

Alauda Container Platform GitOps абстрагирует состояние ресурсов `Application`, используя статус базовых Kubernetes ресурсов. Состояние ресурсов `Application` напрямую управляет состоянием связанных ресурсов `ApplicationSet`.

Содержание

Sync Status Explanation

Как Kubernetes ресурсы, так и приложения имеют четыре состояния синхронизации: **Sync Failed**, **OutOfSync**, **Syncing** и **Synced**.

Sync Status	Описание
Sync Failed	Синхронизация не удалась из-за сетевых ошибок, проблем с конфигурацией или правами доступа. Проверьте логи для выяснения причины.
OutOfSync	Состояние ресурса в кластере отличается от желаемого состояния, определённого в Git. Требуется ручная или автоматическая синхронизация.

Sync Status	Описание
Syncing	Активный процесс согласования состояния кластера с состоянием, определённым в Git.
Synced	Состояние ресурса в кластере соответствует желаемому состоянию, определённому в Git.

INFO

Приоритет отображения статуса синхронизации: порядок приоритета **Sync Failed** > **OutOfSync** > **Syncing** > **Synced**.

Примеры:

- Если у Application два ресурса со статусами **Syncing** и **Synced**, общий статус будет **Syncing**.
- Если ApplicationSet управляет двумя Applications со статусами **Sync Failed** и **Synced**, общий статус будет **Sync Failed**.\

Health Status Explanation

Kubernetes ресурсы и приложения имеют шесть состояний здоровья: **Unknown**, **Missing**, **Degraded**, **Paused**, **Progressing** и **Healthy**.

Health Status	Описание	Рекомендации по устранению
Unknown	Невозможно определить состояние здоровья, обычно из-за ошибок контроллера или отсутствия данных о статусе.	Проверьте `status.conditions` в YAML ресурса для диагностики.
Missing	Ресурс не найден в кластере.	При первоначальном создании: дождитесь

Health Status	Описание	Рекомендации по устранению
		завершения согласования При случайном удалении: инициируйте ручную синхронизацию.
Degraded	Ресурсы рабочей нагрузки (например, Deployment) не достигли здорового состояния в течение тайм-аута (по умолчанию 10 минут).	Исследуйте ошибки Pod (например, сбои, ограничения ресурсов).
Paused	Развёртывание ресурсов рабочей нагрузки намеренно приостановлено (например, с помощью `kubectl rollout pause`).	Возобновите развёртывание при необходимости.
Processing	Ресурс успешно создан, но ещё не полностью готов (например, Pods инициализируются).	Отслеживайте состояние до перехода в Healthy или Degraded.
Healthy	Ресурс работает нормально.	-

INFO

Приоритет состояния здоровья: порядок приоритета **Unknown** > **Missing** > **Degraded** > **Paused** > **Progressing** > **Healthy**

Примеры:

- Если у Application ресурсы со статусами **Healthy** и **Unknown**, общий статус здоровья будет **Unknown**.
- Если ApplicationSet управляет Applications со статусами **Missing** и **Progressing**, общий статус здоровья будет **Missing**.\

Recognition Rules

Правила распознавания статуса **Healthy** для Kubernetes ресурсов:

Тип ресурса	Статус
Deployment	Завершено rolling update, все реплики доступны.
StatefulSet	Обновление завершено, все Pods готовы.
ReplicaSet	Все Pods здоровы.
DaemonSet	Запланировано и здоровое желаемое количество Pods.
Ingress	В статусе заполнен IP/hostname LoadBalancer.
Service	Заполнен IP/hostname LoadBalancer (если применимо).
PVC	Статус **Bound**.
Pod	Все контейнеры готовы, количество перезапусков не превышает порог.
Job	Job успешно завершена (`.status.succeeded >= 1`).
HPA	Успешное масштабирование, текущее количество реплик соответствует желаемому.

Menu

Руководства

Создание GitOps приложения

Creating GitOps Application

Предварительные требования

Создание Argo CD Application через веб-консоль

Создание Argo CD Application через YAML

Создание Argo CD Application через CLI

Creating GitOps ApplicationSet

Overview

Prerequisites

Key Benefits

Creating GitOps Application

Managing GitOps Applications

Наблюдаемость GitOps

Argo CD Component Monitoring

Overview

Prerequisites

Viewing the Argo CD component dashboard

GitOps Applications Ops

Menu

Создание GitOps приложения

Creating GitOps Application

Creating GitOps ApplicationSet

Creating GitOps Application

Обзор

Используйте возможности управления приложениями **Alauda Container Platform GitOps** для визуального создания Argo CD ApplicationSet, обеспечивающего комплексное управление жизненным циклом контейнеризованных приложений через **GitOps Applications**.

Содержание

Предварительные требования

- **Установите Alauda Container Platform GitOps**:
  - Если не установлено, обратитесь к Администратору для Installing Alauda Container Platform GitOps

- **Интеграция Git-репозитория** (выберите один способ):
  - Integrating Code Repositories via Argo CD dashboard
  - Администратор должен предоставить Code Repositories через **DevOps Toolchain** > **Integrate**

Создание Argo CD Application через веб-консоль

> Оптимизируйте распространение приложений с помощью визуальных интерфейсов управления.

Процедура

1. Перейдите в **Container Platform** и откройте **GitOps Applications**.
2. Нажмите **Create GitOps Application**.
3. Настройте параметры в разделах **Basic Info** и **Code Repository**:

Параметр	Описание
Type	**Application**: объект Argo CD Application для развертывания в одном namespace **ApplicationSet**: Argo CD ApplicationSet для кросс-кластерных/кросс-namespace развертываний с дифференцированными конфигурациями
Source	**Platform integrated**: Предварительно настроенные репозитории GitLab/GitHub/Bitbucket **ArgoCD integrated**: репозитории GitLab/GitHub/Bitbucket/Gitee/Gitea, интегрированные через Argo CD. См. Integrating Code Repositories via Argo CD dashboard
Integration Project Name	Проект Toolchain, назначенный Администратором

Параметр	Описание
Version Identifiers	Основа развертывания: `Branch` / `Tag` / `Commit` **Примечание**: • `Branch` использует последний коммит • `Tag` / `Commit` по умолчанию последний, но настраиваемый
Source File Type	**Kustomize**: использует kustomization.yaml для оверлейных конфигураций; подробности в Kustomize Official Documentation ↗ **Helm**: использует values.yaml для шаблонизации; подробности в Helm Official Documentation ↗ **Directory**: исходные манифесты
Source Directory	Путь в репозитории, содержащий базовые манифесты. Поддерживается выбор корневой директории. Все ресурсы в этом пути будут развернуты в целевых кластерах
Custom Values	Если **Source File Type** — **Helm**, можно выбрать пользовательский файл Helm Values

4. Настройте параметры в разделе **Destination**:

   - **Application**: Дифференцированные конфигурации не изменяют базовые файлы в исходной директории.
   - **ApplicationSet**: Мультикластерное развертывание с **Дифференцированной Конфигурацией**.

   Примечание: **Дифференцированная Конфигурация** не изменяет базовые файлы в **Source Directory**.

5. **Sync Policy** (интервал согласования 3 минуты).

Параметр	Описание
Manually Sync	Требуется подтверждение пользователя при обнаружении расхождений.

Параметр	Описание
Automatic Sync	Автоматическое согласование без вмешательства пользователя.
Sync Ignore Configuration	Настройка с использованием встроенных/ пользовательских шаблонов игнорирования; можно View Sync Ignore Configuration Fields in YAML File. **Примечание**: пользовательские шаблоны требуют настройки администратором.

6. Нажмите **Create**.

> **INFO**
>
> **Примечание по ручной синхронизации**: Выберите **Synchronize Immediately** для немедленного развертывания или **Synchronize Later** для запуска вручную через страницу деталей.

Просмотр полей конфигурации игнорирования синхронизации в YAML файле

После настройки правил игнорирования синхронизации проверьте следующим образом:

1. Перейдите в **GitOps Application**
2. Выберите целевое приложение
3. Нажмите **Action > Update**
4. Просмотрите YAML файл.

```yaml
ignoreDifferences: # Конфигурация, фактически игнорируемая выбранным шаблоном пользовательской настройки игнорирования синхронизации
  - group: apps
    kind: Deployment
    jsonPointers:
      - /spec/replicas
```

Создание Argo CD Application через YAML

Процедура

1. Перейдите в **Container Platform** и откройте **GitOps Applications**.
2. Нажмите **Create GitOps Application**.
3. Переключитесь на вкладку **YAML**.
4. В разделе **YAML** используйте следующий YAML-файл и настройте соответствующую информацию. Замените `namespace` и `project` на свои namespace и проект.

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
  namespace: argocd # Замените на свой namespace
spec:
  project: default # Замените на свой проект
  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
```

5. Нажмите **Create**.

Создание Argo CD Application через CLI

Предварительные требования

Плагин web-cli установлен, и переключатель `web-cli` включён.

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

Creating GitOps ApplicationSet

Содержание

Overview

Используйте возможности управления приложениями **Alauda Container Platform GitOps** для визуального создания Argo CD ApplicationSet, обеспечивающего комплексное управление жизненным циклом контейнеризованных приложений через **GitOps Applications**.

Prerequisites

- **Установка Alauda Container Platform GitOps**:
  - Если не установлено, обратитесь к администратору для Installing Alauda Container Platform GitOps

- **Интеграция Git-репозитория** (выберите один способ):
  - Integrating Code Repositories via Argo CD dashboard
  - Администратор должен предоставить репозитории кода через **DevOps Toolchain** > **Integrate**

Key Benefits

- **Визуальное распределение GitOps приложений**: сочетает управление мультикластером, дифференцированные конфигурации и визуальные операции, согласованные с платформой, для упрощения развертывания в мультиоблаках/мультиокружениях.

Creating GitOps Application

> Оптимизируйте распределение приложений через визуальные интерфейсы управления.

Procedure

1. Перейдите в **Container Platform**, затем в **GitOps Applications**.
2. Нажмите **Create GitOps Application**.
3. Настройте параметры в разделах **Basic Info** и **Code Repository**:

Parameter	Description
Type	**Application**: объект Argo CD Application для развертывания в одном namespace **ApplicationSet**: Argo CD ApplicationSet для развертывания в нескольких кластерах/namespace с дифференцированными конфигурациями

Parameter	Description
Source	**Platform integrated**: предварительно настроенные репозитории GitLab/GitHub/Bitbucket **ArgoCD integrated**: репозитории GitLab/GitHub/Bitbucket/Gitee/Gitea, интегрированные через Argo CD. Смотрите Integrating Code Repositories via Argo CD dashboard
Integration Project Name	Проект toolchain, назначенный администратором
Version Identifiers	Основа развертывания: `Branch` / `Tag` / `Commit` **Примечание**: • `Branch` использует последний коммит • `Tag` / `Commit` по умолчанию последние, но настраиваемые
Source File Type	**Kustomize**: использует `kustomization.yaml` для overlay-конфигураций; подробности в Kustomize Official Documentation ↗ **Helm**: использует values.yaml для шаблонов; подробности в Helm Official Documentation ↗ **Directory**: сырые манифесты
Source Directory	Путь в репозитории, содержащий базовые манифесты. Поддерживается выбор корневой директории. Все ресурсы в этом пути будут развернуты в целевых кластерах
Custom Values	Если **Source File Type** — **Helm**, можно выбрать пользовательский файл Helm Values

4. Настройте параметры в разделе **Destination**:

- **Application**: дифференцированные конфигурации не изменяют базовые файлы в исходной директории.
- **ApplicationSet**: мультикластерное развертывание с **Дифференцированной конфигурацией**.

Примечание: **Дифференцированная конфигурация** не изменяет базовые файлы в **Source Directory**.

5. **Sync Policy** (интервал согласования 3 минуты).

Parameter	Description
Manually Sync	Требуется подтверждение пользователя при обнаружении отклонений
Automatic Sync	Автоматическое согласование без вмешательства человека
Sync Ignore Configuration	Настраивается с помощью встроенных/пользовательских шаблонов игнорирования, можно View Sync Ignore Configuration Fields in YAML File **Примечание**: пользовательские шаблоны требуют настройки администратором

6. Нажмите **Create**.

> **INFO**
>
> **Примечание по ручной синхронизации**: выберите **Synchronize Immediately** для немедленного развертывания или **Synchronize Later** для запуска вручную через страницу деталей.

View Sync Ignore Configuration Fields in YAML File

После настройки правил игнорирования синхронизации проверьте следующим образом:

1. Перейдите в **GitOps Application**.
2. Выберите целевое приложение.
3. Нажмите **Action** > **Update**.
4. Просмотрите YAML-файл.

```
ignoreDifferences: # Конфигурация, фактически игнорируемая выбранным шаблоном пользовательской синхронизации игнорирования
  - group: apps
    kind: Deployment
    jsonPointers:
      - /spec/replicas
```

Managing GitOps Applications

Action	Description
Update	Запуск обновлений через: • Иконку редактирования (✎) в списке **GitOps Application** • **Action** > **Update** в подробном просмотре. • **ВНИМАНИЕ**: эта операция перезапишет все созданные экземпляры приложения
Manually Sync	При **Sync Policy** = **Manually Sync** : • Запуск синхронизации через **Action** > **Sync** в подробном просмотре при обнаружении отклонений • Распространяет последние коммиты на все управляемые экземпляры
Delete	Удаление через: • Иконку удаления (🗑) на странице списка • **Action** > **Delete** в подробном просмотре • **ОПАСНО**: удаляет приложение и ВСЕ дочерние ресурсы

Action	Description
Automatic Sync	Включение авто-согласования для поддержания желаемого состояния. Все экземпляры автоматически синхронизируются с изменениями в репозитории каждые 3 минуты
Source	Для приложений типа **ApplicationSet**: • Нажмите ссылку **Source** для перехода на страницу деталей родительского Application.
Application Distribution	Расширение: 1. Обновите существующую конфигурацию **ApplicationSet** 2. В деталях **ApplicationSet**: **Applications** > **Add Distribution**

Menu

Наблюдаемость GitOps

Argo CD Component Monitoring

Overview

Prerequisites

Viewing the Argo CD component dashboard

GitOps Applications Ops

Overview

Prerequisites

Alert

Logs

Events

Argo CD Component Monitoring

Содержание

Overview

Панель мониторинга веб-консоли предоставляет визуальный способ отслеживания компонентов Argo CD. Она проактивно наблюдает за ресурсами и рабочими состояниями компонентов Argo CD, с целью обеспечения их здоровья и доступности. Здесь рабочие состояния означают условия работы и показатели производительности компонентов в среде Kubernetes (K8s). Тщательно отслеживая эти аспекты, мы можем своевременно выявлять и устранять потенциальные проблемы, поддерживая бесперебойную работу Argo CD в кластере K8s.

Prerequisites

- Installing Alauda Container Platform GitOps
- Installation of Monitoring Plugins

Viewing the Argo CD component dashboard

1. Войдите в систему, перейдите в раздел **Administrator** и выберите кластер `global`.
2. Нажмите **Operations Center** > **Monitoring** > **Monitoring Dashboard**.
3. Нажмите кнопку **Switch** и выберите **container-platform**, чтобы просмотреть панель **ArgoCD**.
4. Нажмите на панель **ArgoCD**, чтобы просмотреть информацию о мониторинге компонентов **Argo CD**.

GitOps Applications Ops

Содержание

Overview

Возможности управления **GitOps Applications** в веб-консоли позволяют просматривать мониторинг, логи и события **GitOps Applications**. Вы также можете создавать политики оповещений для **GitOps Applications**. При возникновении аномалий в **GitOps Applications** будут автоматически отправляться уведомления, что способствует быстрому выявлению, анализу и устранению проблем.

Prerequisites

- Веб-консоли уже создано GitOps приложение. Creating an Argo CD Application via the web console
- Installation of Monitoring Plugins

Alert

> Создайте правило оповещения заранее для настройки правил. При возникновении аномалий в GitOps приложениях будут автоматически отправляться уведомления, что позволит быстро выявлять, анализировать и устранять проблемы.

1. В **Container Platform** нажмите на **GitOps Applications**.
2. Выберите из списка имя GitOps приложения, для которого хотите создать правило оповещения.
3. Перейдите на вкладку **Alerts**.
4. Нажмите **Create Rule** и заполните основную информацию по требованию.
5. Нажмите **Add Alert Condition**, перейдите на страницу **Alert Conditions**. Описание соответствующих метрик приведено ниже:

INFO

Для других параметров конфигурации и настроек оповещений смотрите Alert Management.

Metric Name	Rule Description
GitOps Application Health Status `gitops.applicationset.healthy`	Состояние здоровья GitOps приложения: - 0: Неизвестно, Потеряно, Ухудшено или Приостановлено - 1: Синхронизация - 2: Здорово
GitOps Application Sync Status `gitops.applicationset.synced`	Статус синхронизации GitOps приложения: - 0: Синхронизация не удалась или в ожидании - 1: Синхронизация - 2: Синхронизировано

1. Нажмите **Create**.

Logs

> Просмотр логов всех рабочих ресурсов, созданных GitOps приложениями. Логи позволяют быстро выявлять информацию о сбоях системы без необходимости использования плагина **Log** кластера.

- На странице деталей GitOps приложения в разделе **Kubernetes Resources** нажмите на любое имя Workload, чтобы справа просмотреть информацию **Logs** для этого ресурса.

Events

> Просмотр событий всех ресурсов, распределённых **GitOps Applications**. События позволяют быстро выявлять информацию о сбоях системы без необходимости использования плагина **Log** кластера.

- На странице деталей GitOps приложения перейдите на вкладку **Events** для просмотра агрегированных событий всех ресурсов.
- На странице деталей GitOps приложения в разделе **Kubernetes Resources** нажмите на любое имя ресурса, чтобы справа просмотреть события для этого ресурса.

Menu

Как сделать

Интеграция репозиториев кода через панель управления Argo CD

Сценарии использования

Предварительные требования

Процедура

Результат операции

Создание приложения Argo CD через панель управления Argo CD

Предварительные требования

Процедура

Создание Argo CD Application через веб-консоль

Сценарии использования

Требования

Процедура

Как получить информацию для доступа к Argo CD

Сценарии использования

Как получить информацию для доступа к Argo CD для плагина кластера GitOps, установленного через веб-консоль?

Как получить информацию для доступа к Argo CD через Argo CD Operator?

Интеграция репозиториев кода через панель управления Argo CD

Используйте нативную панель управления Argo CD для интеграции репозиториев кода и их распределения, что позволяет разработчикам управлять GitOps-приложениями на протяжении всего их жизненного цикла через визуальный интерфейс.

Содержание

Сценарии использования

- Упростить процесс создания **GitOps Applications**, выбирая связанный репозиторий через веб-консоль при их создании.
- При создании **Application** через нативную панель управления Argo CD можно выбрать использование связанного репозитория.

Предварительные требования

- Установка Alauda Container Platform GitOps, и переключатель **Native Argo CD UI** должен быть включен.

- Доступ к URL **Native Argo CD UI** вместе с именем пользователя и паролем.
  - Администраторы могут напрямую получить доступ к URL через страницу деталей плагина GitOps кластера.

Процедура

Выполните следующие шаги для использования функций:

1. Подключение репозитория кода
   - Войдите в **Argo CD** с использованием URL доступа.
   - Нажмите на **Settings** в левой навигационной панели.
   - Нажмите на карточку **REPO**.
   - Нажмите **CONNECT REPO** в верхнем левом углу страницы.
   - Выберите метод подключения репозитория и заполните соответствующие параметры по необходимости.
   - Нажмите **CONNECT**.
2. Связывание проекта
   - Нажмите на **Settings** в левой навигационной панели.
   - Нажмите на карточку **Projects**.
   - Выберите проект, в котором необходимо создать GitOps-приложение.

   Примечание: Argo CD автоматически синхронизирует проекты в кластере, поэтому создавать их вручную не нужно.

   - Нажмите **EDIT** в разделе **SOURCE REPOSITORIES**.
   - Нажмите **ADD SOURCE**, введите URL репозитория из шага «Подключение репозитория» и свяжите его с проектом.
   - Нажмите **SAVE**.

Результат операции

- Вернитесь в веб-консоль и перейдите в **Container Platform** > **GitOps Applications**. На странице **Create** вы увидите связанные репозитории.

Создание приложения Argo CD через панель управления Argo CD

Содержание

Предварительные требования

- Установите (выберите один из способов):
  - Installing Alauda Container Platform GitOps
  - Installing Alauda Build of Argo CD
- Получены учетные данные для доступа (URL, имя пользователя, пароль) к панели управления Argo CD How to Obtain Argo CD Access Information

Процедура

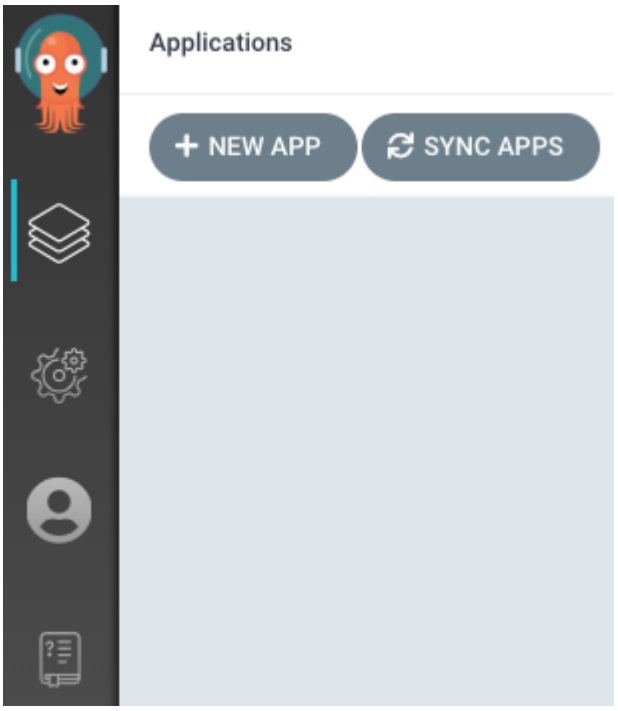
> Следуйте этим шагам для использования функций:
>
> 1. Введите URL доступа к панели управления Argo CD в вашем браузере, чтобы открыть интерфейс.

> Администраторы могут напрямую получить доступ к **Argo CD Native UI** через детали плагина кластера `global`: найдите плагин кластера GitOps и нажмите на адрес

> доступа.

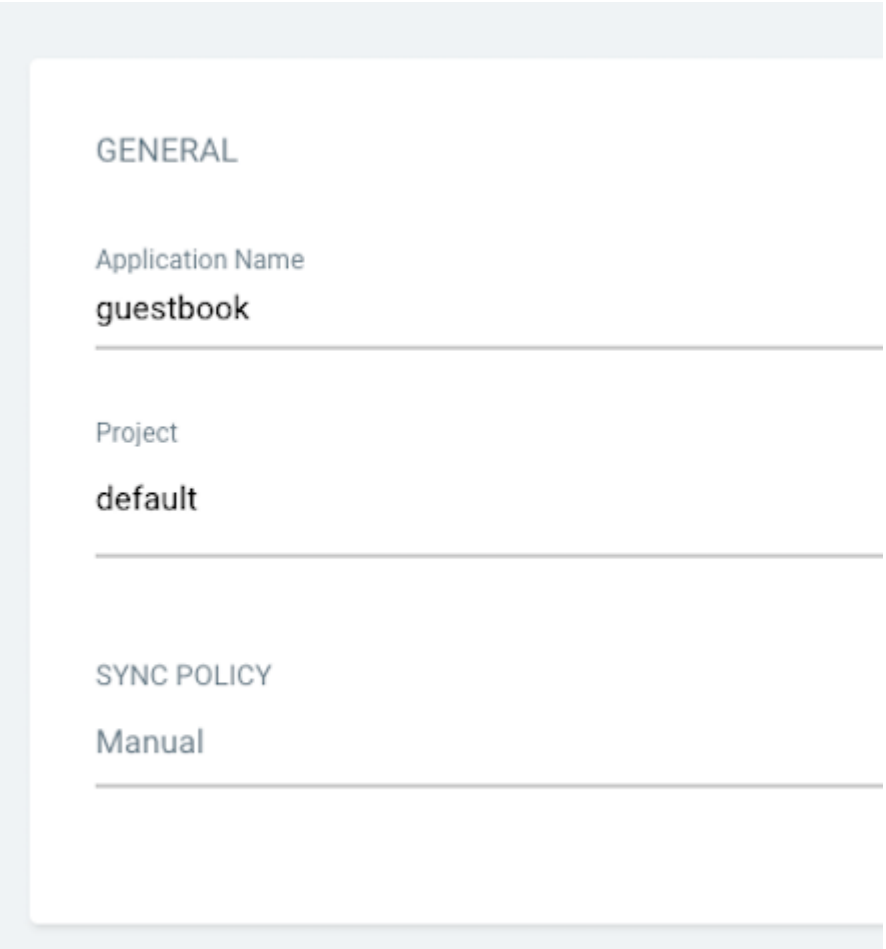
2. Авторизуйтесь, используя свои учетные данные Argo CD, и войдите в систему.

3. Нажмите кнопку **+ NEW APP**, как показано ниже:



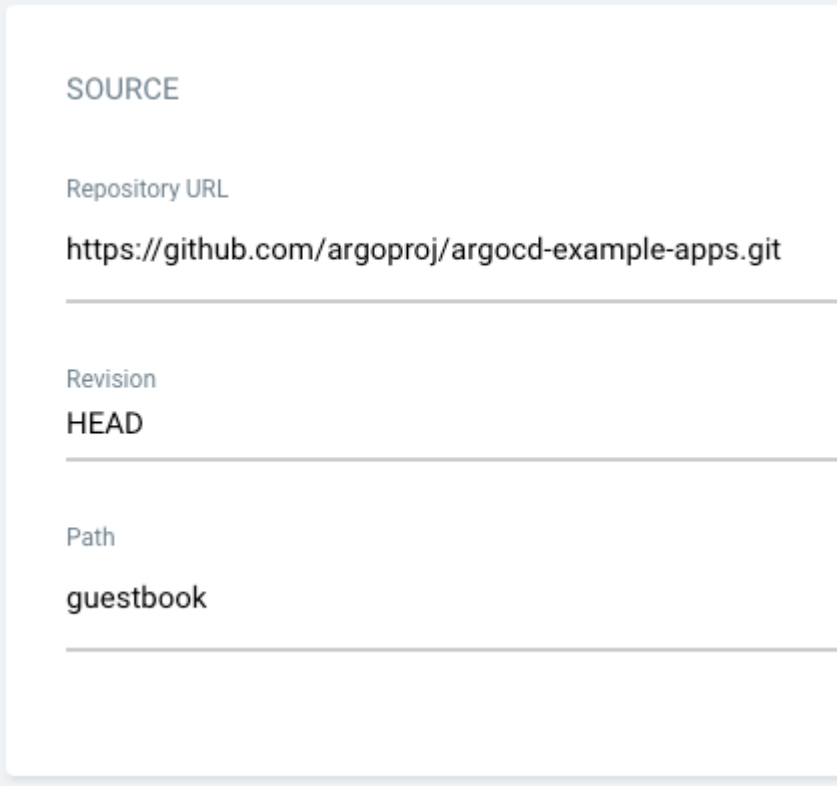
> Настройте приложение согласно следующим шагам:

Конфигурация основных данных



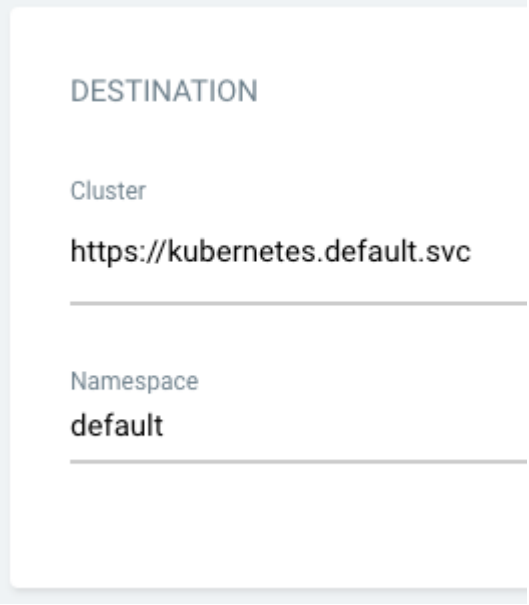
- **Application Name**: Введите `guestbook`
- **Project**: Выберите `default`
- **Sync Policy**: Оставьте `Manual` (рекомендуется для первоначальной настройки)

Конфигурация исходного репозитория



- **Repository URL**: Установите `https://github.com/argoproj/argocd-example-apps.git`
- **Revision**: Используйте значение по умолчанию `HEAD`
- **Path**: Укажите `guestbook` (папка с манифестами Kubernetes)

Конфигурация целевого кластера



- **Clusternew**: Установите `https://kubernetes.default.svc` (доступ внутри кластера) или выберите конкретное имя кластера
- **Namespace**: Установите `default` (или укажите целевой namespace)

1. Создайте **Application**
После завершения настройки нажмите кнопку **Create** в правом верхнем углу, чтобы инициализировать создание приложения guestbook.

Создание Argo CD Application через веб-консоль

В этой статье представлен полный процесс создания Argo CD Application через веб-консоль в разделе **GitOps Applications**, что позволяет управлять приложением с помощью GitOps.

Содержание

Сценарии использования

- Создать SpringBoot Argo CD Application с помощью веб-консоли, чтобы ознакомиться с полным процессом управления приложениями через GitOps.

Требования

- Установка Alauda Container Platform GitOps
- Проекты и пространства имён уже выделены

Процедура

Следуйте этим шагам для использования функционала:

Конфигурация репозитория кода

> Если вы не видите Integrated Code Repository на странице деталей **Create GitOps Applications**, сначала интегрируйте репозиторий кода:

- Интеграция репозиториев кода через Argo CD dashboard

INFO

Если у вас нет доступного репозитория кода, вы можете использовать демонстрационный репозиторий для ознакомления. **URL репозитория**: `https://github.com/argoproj/argocd-example-apps.git` **Описание**: Этот репозиторий содержит примерные приложения, которые можно использовать для демонстрации и тестирования функционала Argo CD.

Создание Argo CD Application с помощью GitOps Applications

1. В **Container Platform** нажмите на **GitOps Applications**.
2. Нажмите **Create GitOps Application**.
3. В разделах **Basic Info** и **Code Repository** настройте соответствующую информацию согласно инструкции ниже.

Параметр	Вводимое значение
Type	`Application`
Source	`Argo CD Integration`
Integrated Project Name	`argocd-example-apps`

Параметр	Вводимое значение
Version Identifier	`Branch` `master`
Source File Type	`Helm`
Source File Directory	`helm-guestbook`
Custom Values	`values.yaml`

4. В разделе **Distribution** используйте рекомендуемое платформой ***Namespace*** или выберите другое пространство имён.

5. Установите политику синхронизации по умолчанию в значение **Manually Sync**.

6. Нажмите **Create**.

Как получить информацию для доступа к Argo CD

> В этой статье подробно описывается, как получить информацию для доступа к Argo CD, включая как плагин кластера **Alauda Container Platform GitOps**, установленный через веб-консоль, так и Argo CD, установленный через оператор **Alauda Build of Argo CD**.

Содержание

Сценарии использования

- Получив информацию для доступа к Argo CD, вы сможете управлять всеми нативными ресурсами Argo CD через панель управления Argo CD.

Как получить информацию для доступа к Argo CD для плагина кластера GitOps, установленного через веб-консоль?

Предварительные требования

- Установка Alauda Container Platform GitOps
- (Опционально) Установлен CLI плагин, и включён переключатель `web-cli`
- У вас есть права администратора

Процедура

INFO

Рекомендуется включить следующие настройки при установке плагина кластера Alauda Container Platform GitOps:

- Включить переключатель **Native Argo CD UI**.
- Включить переключатель **Single Sign-On**.

> Следуйте этим шагам для использования функций:

1. Войдите в систему и перейдите на страницу **Administrator**.
2. Нажмите **Marketplace**, чтобы открыть страницу списка **Cluster Plugins**.
3. Найдите плагин **GitOps**, нажмите на него, и в появившемся окне отобразятся детали **GitOps Cluster Plugin**.

Если он не включён: вернитесь на страницу списка Cluster Plugins, найдите плагин GitOps, нажмите кнопку Действия, выберите Обновить и включите переключатель Argo CD Native UI. Если включён: просто нажмите на Адрес доступа, чтобы открыть панель управления Argo CD.

4. Argo CD Native UI

   - Если не включён: перейдите на страницу списка **Cluster Plugins**, найдите плагин **GitOps**, нажмите кнопку **Update** и включите переключатель **Argo CD Native UI**.
   - Если включён: нажмите напрямую на **Access Address**, чтобы открыть панель управления Argo CD.

5. **Single Sign-On**

   - Если включён: войдите в панель управления Argo CD, используя учётную запись платформы.
   - Если не включён: учётная запись по умолчанию — `admin`, пароль необходимо получить, выполнив следующую команду в **Kubectl** Retrieve Argo CD Password.

Как получить информацию для доступа к Argo CD через Argo CD Operator?

Предварительные требования

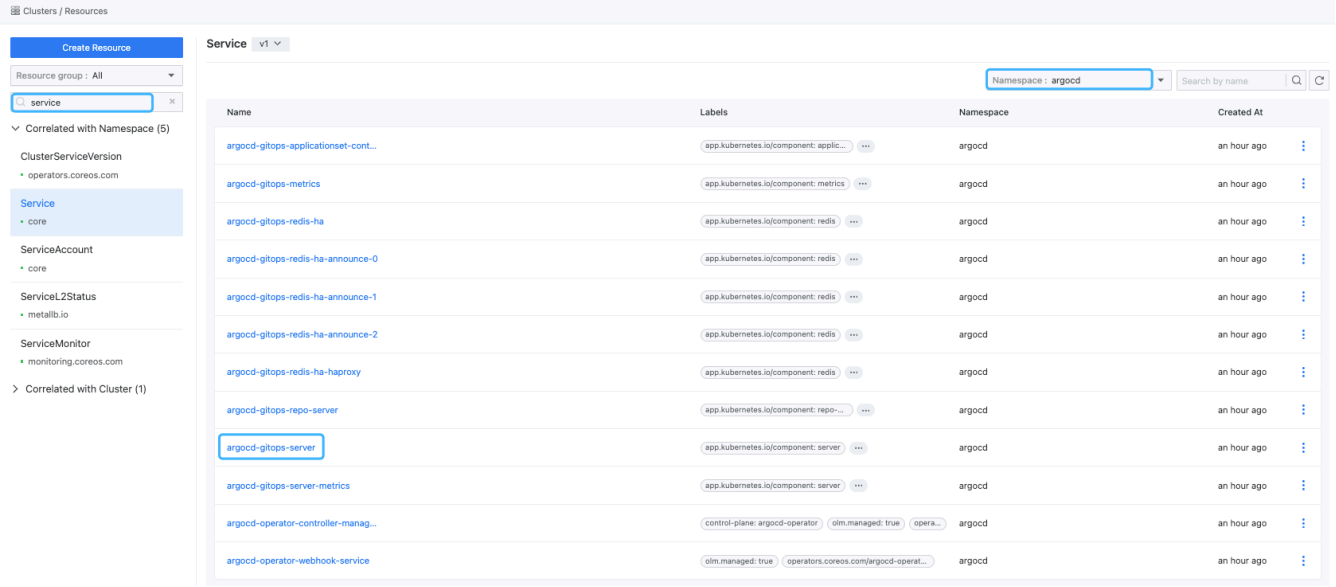
- Установка Argo CD
- (Опционально) Установлен CLI плагин, и включён переключатель `web-cli`
- У вас есть права администратора

Процедура

Получение URL панели управления Argo CD

1. Войдите в систему и перейдите на страницу **Administrator**.
2. Выберите **Cluster Management**, чтобы перейти на страницу **Resource Management**.
3. В разделе **Resource Group** найдите `Service`, выберите пространство имён ***argocd*** (пространство имён, в котором создан экземпляр argocd). Пространство имён по умолчанию для Argo CD, установленного через веб-консоль, — ***argocd***.

4. В правом списке ресурсов найдите `argocd-gitops-server`, нажмите кнопку ***Actions*** и выберите **Update**, чтобы открыть YAML с деталями argocd-gitops-server, как показано на изображении ниже.



5. Измените `type` на `NodePort` и запишите значение `nodePort`, затем нажмите кнопку **Update**.

6. В левой боковой панели выберите **Cluster Management**, чтобы перейти на страницу **Cluster List**.

7. Выберите кластер, где установлен `argocd operator`, перейдите на страницу **Cluster Details** и выберите **Nodes**.

8. Получите IP-адрес ***любого управляющего узла***.

9. Откройте панель управления Argo CD по адресу `http://{IP управляющего узла}:{nodePort}`.

Получение пароля Argo CD

Выполните следующую команду в **Kubectl** для получения пароля:

```
kubectl get secret -n argocd argocd-gitops-cluster -o template --template='{{index .data "admin.password"}}'|base64 -d
```

Обновление пароля учётной записи администратора Argo CD

Пароль по умолчанию для учётной записи `admin`, автоматически созданной при установке Argo CD через **Alauda Container Platform GitOps** или оператор **Alauda Build of Argo CD**, нельзя изменить через интерфейс **Argo CD dashboard**. Вы можете изменить его, выполнив следующую команду в CLI. Здесь `newpassword` — новый пароль, который вы хотите установить.

```
kubectl patch -n argocd secrets argocd-gitops-cluster -p '{"stringData":
{"admin.password":"<newpassword>"}}'
```

Устранение неполадок

Содержание

Я удалил/повредил репозиторий и не могу удалить приложение?

Argo CD не может удалить приложение, если не может сгенерировать манифесты. Вам нужно либо:

1. Восстановить/исправить репозиторий.

2. Удалить приложение с помощью `--cascade=false`, а затем вручную удалить ресурсы.

Почему мое приложение сразу после успешной синхронизации остается в состоянии `OutOfSync`?

Смотрите Diffing Documentation↗ для причин, по которым ресурсы могут быть OutOfSync, и способов настройки Argo CD для игнорирования полей, когда ожидаются различия.

Почему мое приложение застряло в состоянии `Progressing`?

Argo CD предоставляет проверку состояния для нескольких стандартных типов Kubernetes. Типы `Ingress`, `StatefulSet` и `SealedSecret` имеют известные проблемы, которые могут привести к тому, что проверка состояния возвращает `Progressing` вместо `Healthy`.

- `Ingress` считается здоровым, если список status.loadBalancer.ingress не пуст и содержит хотя бы одно значение для `hostname` или `IP`. Некоторые ingress-контроллеры (contour, traefik) не обновляют поле `status.loadBalancer.ingress`, из-за чего `Ingress` застревает в состоянии `Progressing` навсегда.
- `StatefulSet` считается здоровым, если значение поля `status.updatedReplicas` совпадает с полем `spec.replicas`. Из-за бага Kubernetes kubernetes#68573↗ поле status.updatedReplicas не заполняется. Поэтому, если вы используете версию Kubernetes без исправления kubernetes#67570↗, StatefulSet может оставаться в состоянии Progressing.
- Ваш `StatefulSet` или `DaemonSet` использует стратегию `OnDelete` вместо `RollingUpdate`.

- Для `SealedSecret` смотрите Почему ресурсы типа `SealedSecret` застревают в состоянии `Progressing`?

В качестве обходного решения Argo CD позволяет настроить проверку состояния↗, которая переопределяет поведение по умолчанию.

Если вы используете Traefik для вашего Ingress, вы можете обновить конфигурацию Traefik, чтобы публиковать IP loadBalancer с помощью publishedservice↗, что решит эту проблему.

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

Как отключить пользователя admin?

Добавьте `admin.enabled: "false"` в ConfigMap `argocd-cm`.

Argo CD не может развернуть приложения на основе Helm Chart без доступа в интернет, как это исправить?

Argo CD может не сгенерировать манифесты Helm chart, если у чарта есть зависимости из внешних репозиториев. Чтобы решить проблему, убедитесь, что `requirements.yaml` использует только внутренние Helm репозитории. Даже если чарт использует только зависимости из внутренних репозиториев, Helm может попытаться обновить репозиторий `stable`. В качестве обходного решения переопределите URL репозитория `stable` в ConfigMap `argocd-cm`:

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

После создания Helm приложения через Argo CD я не вижу его в `helm ls` и других командах Helm?

При развертывании Helm приложения Argo CD использует Helm только как механизм шаблонизации. Он выполняет `helm template`, а затем разворачивает полученные манифесты в кластере вместо `helm install`. Это означает, что вы не можете использовать команды Helm для просмотра или проверки приложения. Управление полностью осуществляется Argo CD. Обратите внимание, что Argo CD нативно поддерживает некоторые возможности, которых может не хватать в Helm (например, команды истории и отката).

Такое решение принято, чтобы Argo CD был нейтрален к генераторам манифестов.

Я настроил секрет кластера, но он не отображается в CLI/UI, как это исправить?

Проверьте, есть ли у секрета кластера метка `argocd.argoproj.io/secret-type: cluster`. Если метка есть, но кластер все равно не виден, возможно, проблема с правами доступа. Попробуйте вывести список кластеров, используя пользователя `admin` (например: `argocd login --username admin && argocd cluster list`).

Почему мое приложение остается Out Of Sync даже после синхронизации?

В некоторых случаях инструмент, который вы используете, может конфликтовать с Argo CD, добавляя метку `app.kubernetes.io/instance`. Например, при использовании функции common labels в Kustomize.

Argo CD автоматически устанавливает метку `app.kubernetes.io/instance` и использует её для определения ресурсов, входящих в приложение. Если инструмент делает то же самое, это вызывает путаницу. Вы можете изменить эту метку, установив значение application.instanceLabelKey в `argocd-cm`. Рекомендуется использовать `argocd.argoproj.io/instance`.

INFO

После этого изменения ваши приложения станут Out Of Sync и потребуют повторной синхронизации.

Как часто Argo CD проверяет изменения в моем Git или Helm репозитории?

Интервал опроса по умолчанию — 3 минуты (180 секунд) с настраиваемым джиттером. Вы можете изменить настройки, обновив значения `timeout.reconciliation` и `timeout.reconciliation.jitter` в ConfigMap `argocd-cm`. Если есть изменения в Git, Argo CD обновит только приложения с включенной настройкой `auto-sync`. Если установить значение в 0, Argo CD прекратит автоматический опрос Git репозиториев, и вы сможете использовать только альтернативные методы, такие как `webhooks` и/или `ручная синхронизация` для создания приложений.

Как исправить ошибку invalid cookie, longer than max length 4093?

Argo CD использует JWT в качестве токена аутентификации. Вероятно, вы состоите во многих группах, и размер cookie превысил лимит в 4 КБ. Вы можете получить список групп, открыв "developer tools -> network":

1. Нажмите вход в панель Argo CD How to Obtain Argo CD Access Information
2. Найдите вызов `<argocd_instance>/auth/callback?code=<random_string>`

Декодируйте токен на jwt.io ↗. Это покажет список команд, из которых вы можете удалить себя.

Почему при использовании CLI я получаю ошибку rpc error: code = Unavailable desc = transport is closing?

Возможно, вы находитесь за прокси, который не поддерживает HTTP 2? Попробуйте флаг `--grpc-web`:

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

Почему ресурсы типа SealedSecret застревают в состоянии Progressing?

Контроллер ресурса `SealedSecret` может выставлять статусное условие на ресурс, который он создал. Начиная с версии `v2.0.0` Argo CD учитывает это статусное условие для определения состояния здоровья `SealedSecret`.

Версии контроллера `SealedSecret` до `v0.15.0` страдают от проблемы с обновлением этих статусных условий, поэтому эта функция по умолчанию отключена в этих версиях. Обновления статусных условий можно включить, запустив контроллер `SealedSecret` с параметром командной строки `--update-status` или установив переменную окружения `SEALED_SECRETS_UPDATE_STATUS`.

Чтобы отключить проверку Argo CD статусного условия у ресурсов `SealedSecret`, добавьте следующую настройку кастомизации ресурсов в ConfigMap `argocd-cm` через ключ `resource.customizations.health.<group_kind>`.

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

Как выполнить ротацию ключей Redis?

- Удалите секрет `argocd-redis` в namespace, где установлен Argo CD.

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

- Если вы используете Redis в HA режиме, перезапустите Redis в HA.

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

- Если вы используете Redis в не-HA режиме, перезапустите Redis.

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

- Перезапустите остальные компоненты.

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

Как исправить Manifest generation error (cached)?

`Manifest generation error (cached)` означает, что при генерации манифестов произошла ошибка, и сообщение об ошибке было закэшировано, чтобы избежать бесконечных повторных попыток.

Жесткое обновление (игнорирование кэшированной ошибки) может помочь при временных проблемах. Но если причина ошибки генерации манифестов сохраняется, жесткое обновление не поможет.

Вместо этого попробуйте поискать в логах repo-server по имени приложения, чтобы определить ошибку, вызывающую сбой генерации манифестов.